UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

José Teixeira da Silva Junior

π FlowMR - Uma nova arquitetura a fluxo de dados
dinâmico, escalável e com múltiplos anéis, implementada
em um *cluster* de FPGAs de baixo custo

São Carlos

2021

José Teixeira da Silva Junior

**π FlowMR - Uma nova arquitetura a fluxo de dados
dinâmico, escalável e com múltiplos anéis, implementada
em um *cluster* de FPGAs de baixo custo**

Tese apresentada ao Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para obtenção do título de Doutor em Ciências.

Área de concentração: Física Aplicada
Opção: Física Computacional

Orientador: Prof. Dr. Carlos Antônio Ruggiero

Versão corrigida

(versão original disponível na Unidade que aloja o Programa)

São Carlos

2021

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Silva Junior, José Teixeira da
piFlowMR - Uma nova arquitetura a fluxo de dados
dinâmico, escalável e com múltiplos anéis, implementada em
um cluster de FPGAs de baixo custo / José Teixeira da
Silva Junior; orientador Carlos Antonio Ruggiero - versão
corrigida -- São Carlos, 2021.
225 p.

Tese (Doutorado - Programa de Pós-Graduação em Física
Aplicada Computacional) -- Instituto de Física de São
Carlos, Universidade de São Paulo, 2021.

1. Arquiteturas de computadores não-convencionais
paralelas. 2. Fluxo de dados. 3. Escalabilidade. 4.
Bluespec. 5. Multianel. I. Ruggiero, Carlos Antonio,
orient. II. Título.

AGRADECIMENTOS

O primeiro agradecimento deste trabalho é dirigido ao orientador exímio, professor dedicado e grande amigo Dr. Carlos Antônio Ruggiero, quem sempre empregou muitos esforços para a realização deste trabalho, desde a sua introdução, ainda para o planejamento e implementação do trabalho que compôs a dissertação de mestrado deste autor.

Ainda neste sentido, gostaria de deixar exposto neste trabalho, meus sinceros agradecimentos a nomes que foram fundamentais para sua realização plena, os pesquisadores Drs. Paulo Matias, Felipe Ferreira, Alberto Tannús e Mateus José Martins, que também dispuseram de tanto esforço e dedicação em atividades diversas como discussões, ideias e orientações importantes para a tomada de decisões, que certamente enriqueceram substancialmente este desenvolvimento. Não menos importante, é necessário destacar todo o esforço empregado pelo técnico Ricardo Silva dos Santos, para a construção do sistema de comunicação implementado para o correto funcionamento do protótipo construído.

Agradeço ainda, imensamente, ao grupo CIERMag (Centro de Imagens e Espectroscopia *in vivo* por Ressonância Magnética), que forneceu diversos subsídios físicos, tão importantes para a realização deste trabalho.

Destaco também a importância da Lhaís Visentin, grande companheira, esposa dedicada e que me acompanha desde o início do desenvolvimento deste projeto. Agradeço à todos os membros de minha família, em especial deixo registrado neste texto toda a gratidão a minha mãe, Ediene Santos da Silva, a melhor pessoa com quem eu tive o imenso prazer de compartilhar parte importante de minha vida e que certamente foi fundamental para a construção do caráter que me conduz.

Aproveito para registrar também um agradecimento especial a todo o corpo docente e funcionários do Instituto de Física de São Carlos, afinal de contas, sem todo o esforço empregado por todos ao longo de tantos anos em que sigo em aprendizado, nada disso teria valor algum.

*“Toda a nossa ciência, comparada com a realidade, é primitiva e infantil – e, no entanto,
é a coisa mais preciosa que temos.”*

Albert Einstein

RESUMO

SILVA JUNIOR, J. T. π FlowMR - Uma nova arquitetura a fluxo de dados dinâmico, escalável e com múltiplos anéis, implementada em um *cluster* de FPGAs de baixo custo. 2021. 225p. Tese (Doutorado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

Durante muitos anos, o ganho em desempenho computacional foi alcançado principalmente a partir da evolução na tecnologia, com a introdução de novas famílias de processadores com frequência de operação cada vez mais alta. Entretanto, limitações físicas passaram a impedir que este aumento pudesse seguir avançando. Nos últimos anos, o paralelismo se tornou a principal fonte de exploração para ganho em desempenho computacional. Porém, a grande quantidade de elementos de processamento, métodos de programação e arquiteturas distintas, acabaram tornando o processo de exploração desta capacidade computacional demasiado laborioso. O modelo a fluxo de dados trata de uma arquitetura de computadores não convencional, que tem a finalidade de garantir a exploração de paralelismo de forma nativa. Este trabalho visa a implementação de um protótipo funcional deste modelo de computação paralela, inspirada no conjunto de instruções da Máquina a Fluxo de Dados de Manchester, a partir de um *cluster* **FPGAs** de baixo custo, em um formato que garanta uma boa capacidade para escalabilidade, capaz de explorar até centenas de unidades de processamento, em um sistema que ficou conhecido como Multanel. A implementação deste protótipo funcional levou à verificação de *speedups* que atingem a ordem de algumas dezenas, para programas com grande capacidade de paralelismo, não derivados de estruturas de dados regulares.

Palavras-chave: Arquiteturas de computadores não-convencionais paralelas. Fluxo de dados. Escalabilidade. Bluespec. Multanel.

ABSTRACT

SILVA JUNIOR, J. T. π FlowMR - A new dynamic, scalable and multiring dataflow architecture, implemented in a low cost FPGA cluster. 2021. 225p.
Thesis (Doctor in Science) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

For many year, the computational performance gain has been achieved mainly through technology advances, with the manufacturers introducing new processor families with increasingly higher frequency operation. However, physical limitations has prevented such frequency increase, limiting this strategy for performance gain advances. In the last few years, the parallelism has become the main source of exploring this capacity. In spite of the large amount of processing elements, programming methods and architecture models ended up turning the process of exploiting this computational capacity too laborious. The dataflow model is a non-conventional computer architecture, designed to exploit a large amount of parallelism in a native way. This work aim to implement a functional prototype of this parallel computer model, inspired by the Manchester Dataflow Machine instruction set, through a low cost FPGA cluster, ensuring a well capacity for scalability to explore up to hundreds of processing elements, using an approach known as Multiring. The implementation of this working prototype led to speedups in the order of tens, for programs with a high degree of paralelism, not derived from regular data structures.

Keywords: Non-conventional parallel computer architectures. Dataflow. Scalability. Blue-spec. Multiring.

LISTA DE FIGURAS

Figura 1 – Nas últimas cinco décadas a quantidade de transistores em sistemas integrados dobraram a cada período de dois anos, bem como a frequência de processamento que seguia um crescimento proporcional, até o início dos anos 2000.	26
Figura 2 – Composição da estrutura básica de uma FPGA	30
Figura 3 – Diagrama de Blocos de um Módulo de Lógicas Adaptáveis da Altera / Intel.	32
Figura 4 – Complexidade dos níveis de abstrações para a especificação de circuitos digitais ao longo dos anos.	34
Figura 5 – Diagrama representativo dos módulos, regras e interfaces do Bluespec SystemVerilog	36
Figura 6 – Diagrama representativo do escalonamento para a composição de regras dentro de um mesmo ciclo do relógio no Bluespec SystemVerilog	38
Figura 7 – Representação bidimensional do trecho de programa apresentado no Algoritmo 1 de acordo com a definição do modelo a fluxo de dados.	40
Figura 8 – Formato em grafo do modelo a fluxo de dados, para a representação do trecho de programa apresentado no Algoritmo 1, que pode ser executada por uma implementação.	42
Figura 9 – Diagrama de representação da visão geral da Máquina a Fluxo de Dados de Manchester.	45
Figura 10 – Diagrama ilustrativo da Fila de Fichas na MDFM	47
Figura 11 – Diagrama ilustrativo da Unidade de Pareamento na MDFM	49
Figura 12 – Diagrama ilustrativo da Unidade de Instruções na MDFM	51
Figura 13 – Diagrama ilustrativo da Unidade de Processamento na MDFM	53
Figura 14 – Diagrama de representação da π Flow , primeira versão do presente projeto.	58
Figura 15 – Diagrama de representação da Fila de Fichas na primeira versão do projeto.	59
Figura 16 – Diagrama de representação da Unidade de Pareamento na primeira versão do projeto.	61
Figura 17 – Diagrama de representação da Unidade de Instruções na primeira versão do projeto.	64
Figura 18 – Diagrama de representação da Unidade de Processamento na primeira versão do projeto.	66
Figura 19 – Diagrama de representação das Unidades Funcionais na primeira versão do projeto.	68

Figura 20 – Diagrama de representação da segunda versão do projeto, que ficou conhecida como π FlowBW	72
Figura 21 – Diagrama de representação da Unidade de Pareamento , com as modificações propostas para leitura da segunda porta de comunicação da memória RAM desta unidade.	80
Figura 22 – Diagrama de representação da Unidade de Instruções , com as modificações propostas para leitura da segunda porta de comunicação da memória RAM desta unidade e introdução da pseudo predição de desvios.	82
Figura 23 – Topologia do Multianel segundo a implementação proposta para a π FlowMR	89
Figura 24 – Diagrama de representação da arquitetura e fluxo de informações da π FlowBW com a adição dos módulos de comunicação.	94
Figura 25 – Diagrama de representação do conector de um cabo que segue a especificação ATA/ATAPI-6 para comunicação paralela.	101
Figura 26 – Diagrama esquemático do módulo de três estados utilizado para a elaboração dos pinos de comunicação das vias de transmissão de dados no Multianel	103
Figura 27 – Diagrama de representação das filas de sincronismo entre diferentes domínios de frequências, padrões do BSV	104
Figura 28 – Diagrama esquemático do circuito de proteção presente nos principais <i>kits</i> de desenvolvimento da Terasic , porém omitidas na DE10-Nano utilizada para implementação deste projeto.	105
Figura 29 – Diagrama de representação do cabeçalho de um conector que segue a especificação ATA para comunicação paralela.	106
Figura 30 – Diagrama representativo da disposição dos pinos da GPIO do <i>kit</i> de desenvolvimento DE10-Nano. Segundo o protocolo de comunicação, o GPIO 0 é definido como o módulo de comunicação direito e o GPIO 1 o módulo esquerdo.	107
Figura 31 – Diagrama representativo da primeira versão proposta para o módulo de comunicação. Uma falha sistemática impedia a capacidade para ganho em desempenho a partir dessa solução.	109
Figura 32 – Curva de onda da interface de comunicação entre os diferentes domínios do relógio, da adição de uma ficha no <i>buffer</i> de entrada do módulo de comunicação até sua disponibilidade no anel vizinho.	110
Figura 33 – Diagrama de representação da implementação de validação da π FlowMR , com todos os dutos de largura de banda em comunicação imediata aos anéis vizinhos da estrutura.	111

Figura 34 – Diagrama representativo da versão final proposta para o módulo de comunicação. A partir dessa implementação definitiva foi possível verificar um desempenho ainda melhor com relação ao método de validação.	114
Figura 35 – Diagrama representativo das pontes de comunicação AXI no contexto dos pinos de entrada e saída da FPGA e a interconexão L3 do módulo HPS	119
Figura 36 – Diagrama de representação sobre o formato de expansão do grafo em fluxo de dados dos programas utilizados para validação da máquina, que neste modelo é representado por uma árvore binária de altura definida pelo valor de referência, que pode ser manipulado em instruções específicas dos programas.	128
Figura 37 – Curva de <i>Speedup</i> na π Flow , para o programa de Integração, Seção 6.2, em todas as suas versões disponíveis: valores inteiros, valores em ponto flutuante e com funções trigonométricas.	132
Figura 38 – Curva de <i>Speedup</i> na π Flow com a introdução do sistema de atraso na execução de instruções. Para esta análise foi utilizado apenas o programa de integração que executa funções trigonométricas, variando a quantidade de atraso necessária a fim de identificar o melhor <i>speedup</i> possível dentro desta implementação.	134
Figura 39 – Curva de <i>Speedup</i> na π FlowBW , para a execução do programa utilizado para realizar o cálculo do fatorial modificado. Neste gráfico foi disposta a evolução para ganho em desempenho para o mesmo programa, em diferentes tamanhos.	136
Figura 40 – Evolução e extrapolação da utilização de recursos no processo de compilação do projeto para utilização em FPGAs , de acordo com a adição de unidades funcionais e consequente quantidade de dutos de largura de banda.	142
Figura 41 – Comportamento da curva de <i>speedup</i> da última versão da π FlowBW , antes de ser iniciado o processo de implementação da arquitetura do Multianel . O ajuste teórico tem a finalidade de encontrar uma aproximação no comportamento da máquina com mais unidades funcionais.	143
Figura 42 – Conjunto de resultados da π FlowMR para os programas do fatorial e de integração, obtidos ainda na primeira versão da implementação do módulo de comunicação. Como destacado nos gráficos, é possível verificar que o <i>speedup</i> segue sempre abaixo ou muito próximo do valor 8, principalmente com a utilização de mais unidades funcionais.	146

Figura 43 – Curva de <i>speedup</i> consolidada na primeira versão dos módulos de comunicação da π FlowMR . Para a elaboração destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel.	154
Figura 44 – Resultados da π FlowMR para o programa de integração, obtidos nas versões de validação da implementação, com a definição de recursos mínimos no pré-processador. É possível verificar, neste conjunto de resultados, que o <i>speedup</i> já apresenta um comportamento muito mais próximo do desejado.	156
Figura 45 – Comportamento da curva de <i>speedup</i> consolidada nas versões de validação da π FlowMR . Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução do programa de integração operando sobre valores inteiros apenas.	160
Figura 46 – Conjunto de resultados da versão final da π FlowMR para os programas de fatorial e integração operando sobre valores inteiros apenas. É possível verificar, neste conjunto de resultados, que o <i>speedup</i> se aproxima ainda mais do comportamento ideal.	164
Figura 47 – Curva de <i>speedup</i> consolidada na versão final da π FlowMR . Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução dos programas do Fatorial e Integração operando sobre valores inteiros apenas.	167
Figura 48 – Conjunto de resultados da versão final da π FlowMR para os programas de integração operando sobre valores em ponto flutuante. É possível verificar, neste conjunto de resultados, que o <i>speedup</i> se aproxima ainda mais do comportamento ideal desta métrica.	170
Figura 49 – Curva de <i>speedup</i> consolidada na versão final da π FlowMR . Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução dos programas de Integração operando sob ponto flutuante e com operações trigonométricas.	173
Figura 50 – Comportamento da curva de <i>speedup</i> para a execução dos programas de integração operando sobre valores inteiros apenas e sob operações trigonométricas. Para a composição destes gráficos foram realizadas as execuções no simulador para diferentes quantidades de <i>threads</i>	175

Figura 51 – Ilustração do processo de evolução do algoritmo CORDIC após a realização de quatro iterações.	198
Figura 52 – Diagrama esquemático utilizado para a elaboração do projeto do circuito de proteção dos pinos de entrada e saída de propósito geral do <i>kit</i> de desenvolvimento DE10 Nano . Elaborado a partir do software <i>Eagle 9.6.2</i> . Construído inspirado pelo <i>datasheet</i> da fabricante. ¹	204
Figura 53 – Gráfico de especificação sobre o princípio de funcionamento do diodo <i>schottky</i> , capaz de limitar a tensão para valores altos de corrente, com tempo de resposta muito rápido.	205
Figura 54 – Disposição do circuito e componentes distribuídos no lado superior do circuito impresso construído a partir do diagrama esquemático mostrado na Figura 52.	206
Figura 55 – Disposição do circuito e componentes distribuídos no lado inferior do circuito impresso construído a partir do diagrama esquemático mostrado na Figura 52.	207
Figura 56 – Foto ilustrativa do lado inferior da placa de circuito impresso de proteção para os pinos de comunicação de propósito geral dos <i>kits</i> de desenvolvimento Terasic DE10 Nano.	208
Figura 57 – Foto ilustrativa do lado superior da placa de circuito impresso de proteção para os pinos de comunicação de propósito geral dos <i>kits</i> de desenvolvimento Terasic DE10 Nano.	209
Figura 58 – Foto ilustrativa do extensor utilizado para corrigir a falha de projeto encontrada durante a realização dos testes preliminares, para uso do circuito de proteção sob os <i>kits</i> de desenvolvimento Terasic DE10 Nano.	210
Figura 59 – Foto ilustrativa dos conectores construídos para a ligação dos kits de desenvolvimento a partir dos pinos de comunicação de propósito geral dos <i>kits</i> de desenvolvimento Terasic DE10 Nano.	211
Figura 60 – Foto ilustrativa do cabo construído para a comunicação entre os <i>kits</i> de desenvolvimento Terasic DE10 Nano.	211
Figura 61 – Foto ilustrativa de parte do aparato experimental, com destaque para a ligação entre os <i>kits</i> de desenvolvimento Terasic DE10 Nano utilizadas para a elaboração deste projeto, com as placas de proteção e cabos de comunicação construídas.	212

LISTA DE TABELAS

Tabela 1	– Distribuição dos pinos de comunicação nos módulos GPIO de acordo com as definições do protocolo. Para a distribuição dos pinos foi utilizada uma estratégia semelhante ao padrão ATA . ²	108
Tabela 2	– Valores de S_0 , S_∞ , paralelismo médio π e a quantidade total de instruções de passagem direta de cada um dos micro <i>benchmarks</i> utilizados para validação da arquitetura, para diferentes tamanhos destes programas.	130
Tabela 3	– Valores utilizados para a determinação do gráfico que compõe a Figura 37. Todos os valores presentes nesta tabela referem-se à última curva do gráfico, para o programa de integração utilizando funções trigonométricas. Este programa possui um total de 51183 instruções e valor de $\pi = 234.7$	133
Tabela 4	– Resultados obtidos para a determinação do crescimento em <i>speedup</i> no programa de integração que opera sob funções trigonométricas, com a introdução de um atraso adicionado intencionalmente na operação das unidades funcionais, com a finalidade de promover sua maior utilização quando comparadas ao restante da máquina.	135
Tabela 5	– Valores utilizados para a construção do gráfico da Figura 39, referentes à execução do programa do fatorial modificado, para o cálculo do valor 1000, que trata-se da curva de melhor desempenho no gráfico referenciado. Este programa possui um total de 28985 instruções e valor de $\pi = 223$	138
Tabela 6	– Distribuição de recursos para a execução dos programas de tamanho intermediário na π FlowBW , com a adição das soluções de passagem direta. O programa de integração utilizado nesta análise refere-se a versão que opera sobre valores inteiros apenas.	148
Tabela 7	– Análise do <i>Speedup</i> Efetivo, considerando a quantidade máxima de recursos necessários para atingir o limite do valor assintótico desta métrica, para a execução de ambos os programas, operando sobre valores inteiros apenas, em seus maiores tamanhos analisados.	150
Tabela 8	– <i>Speedup</i> Efetivo, para a execução do programa de integração, com profundidade 11, em ambas as versões do programa.	158
Tabela 9	– <i>Speedup</i> Efetivo, para a execução do programa do fatorial modificado com tamanho igual a 1500 e de integração, com tamanho distribuído em 11 níveis, na versão final da implementação.	163

Tabela 10 – Valores do <i>Speedup</i> Efetivo para a execução dos programas de integração em ambas as versões que operam sob valores em ponto flutuante, obtidos na versão final da implementação.	171
Tabela 11 – Lista de componentes eletrônicos utilizados para a montagem do circuito impresso construído para proteção dos pinos de GPIO do <i>kit</i> de desenvolvimento DE10 Nano.	208

LISTA DE ABREVIATURAS E SIGLAS

ALM	<i>Adaptive Logic Module</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
ATA	<i>Advanced Technology Attachment</i>
AXI	<i>Advanced eXtensible Interface</i>
BRAM	<i>Block Random Access Memory</i>
BSV	<i>Bluespec SystemVerilog</i>
BW	<i>BandWidth</i>
CORDIC	<i>COordinate Rotation DIgital Computer</i>
CPLD	<i>Complex Programmable Logic Device</i>
EDGE	<i>Explicit Data Graph Execution</i>
EHR	<i>Ephemeral History Register</i>
ENUM	<i>Enumerate</i>
FIFO	<i>First In, First Out</i>
FPGA	<i>Field-programmable Gate Array</i>
GCC	<i>GNU Compiler Collection</i>
GHRD	<i>Golden Hardware Reference Design</i>
GPIO	<i>General Purpose Input/Output</i>
GPV	<i>Global Programmers View</i>
HDL	<i>Hardware Description Language</i>
HDMI	<i>High Definition Multimedia Interface</i>
HPS	<i>Hard Processor System</i>
I ² C	<i>Inter-Integrated Circuit</i>
IDE	<i>Integrated Drive Eletronics</i>
IEEE	<i>Institute of Electrical and Eletronics Engineers</i>

IP	<i>Intellectual Property</i>
LED	<i>Light-emitting Diode</i>
LUT	<i>LookUp Tables</i>
MDFM	<i>Manchester Dataflow Machine</i>
MIT	<i>Massachusetts Institute of Technology</i>
MR	<i>MultiRing</i>
RAM	<i>Random Access Memory</i>
RTL	<i>Register Transfer Language</i>
SoC	<i>System-on-chip</i>
SMD	<i>Surface-Mount Device</i>
SSH	<i>Secure Shell</i>
TRS	<i>Term Rewriting System</i>
UDMA	<i>Ultra Direct Memory Access</i>
VCD	<i>Value Change Dump</i>
VHDL	<i>VHSIC Hardware Description Language</i>
VHSIC	<i>Very High-Speed Integrated Circuits</i>

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Motivação	25
1.2	Objetivos	27
1.3	Estrutura da tese	28
2	FUNDAMENTAÇÃO TEÓRICA	29
2.1	FPGA	29
2.2	Bluespec <i>SystemVerilog</i>	32
2.3	Fluxo de Dados	38
2.4	Máquina a Fluxo de Dados de Manchester	44
2.4.1	Fila de Fichas	47
2.4.2	Unidade de Pareamento	48
2.4.3	Unidade de Sobrecarga	50
2.4.4	Unidade de Instruções	50
2.4.5	Unidade de Processamento	52
2.5	Outras implementações	53
2.5.1	TRIPS	54
2.5.2	Maxeler	54
2.5.3	Teraflux	54
2.5.4	WaveScalar	55
3	PRIMEIRA VERSÃO	57
3.1	π Flow	57
3.1.1	Fila de Fichas	58
3.1.2	Unidade de Pareamento	60
3.1.3	Unidade de Instruções	64
3.1.4	Unidade de Processamento	65
3.1.5	Unidade Funcional	67
3.2	π FlowBW	69
3.2.1	Análise dos resultados	69
3.2.2	Escopo	70
3.2.3	Implementação	71
3.2.4	Limitações	73
4	OTIMIZAÇÕES NA π FLOWBW	75
4.1	Otimizações de <i>hardware</i>	75

4.2	Otimizações de passagem direta	78
4.2.1	Unidade de Pareamento	78
4.2.2	Unidade de Instruções	80
4.2.3	Unidade de Processamento	83
4.3	Conjunto de Instruções	84
5	π FLOWMR	87
5.0.1	Arquitetura	88
5.0.2	Distribuição de Fichas	94
5.0.3	Sobrecarga	98
5.1	Protocolo de Comunicação	99
5.1.1	Primeira Versão	108
5.1.2	Validação do Método	110
5.1.3	Versão Final	113
5.2	Síntese	114
5.3	Operação	117
6	RESULTADOS	121
6.1	Métricas de Análise	121
6.1.1	<i>Speedup</i>	121
6.1.2	Paralelismo Médio	124
6.2	Programas	126
6.2.1	Fatorial Modificado	127
6.2.2	Integração binária	128
6.3	π Flow	131
6.4	π FlowBW	135
6.5	Otimizações na π FlowBW	139
6.6	Multianel	144
6.6.1	Primeiras Impressões	145
6.6.2	Validação do Método	155
6.6.3	Versão Final	161
6.6.4	Arquiteturas convencionais	174
7	CONCLUSÃO	179
7.1	Resumo do que foi atingido	179
7.2	Trabalhos Futuros	186
	Referências	189

	Apêndices	195
	APÊNDICE A – CORDIC	197
A.1	Definições	197
A.2	Implementação	201
	APÊNDICE B – CIRCUITO DE PROTEÇÃO	203
B.1	Projeto	203
B.2	Implementação	207
	Anexos	213
	ANEXO A – PROGRAMAS	215
A.1	SISAL	215
A.2	Linguagem Montadora	217
A.3	C++	222

1 INTRODUÇÃO

Este trabalho é uma evolução do objeto de estudos construído para compor a dissertação de mestrado do autor. Seu principal objetivo é propor e implementar uma nova arquitetura baseada no modelo a fluxo de dados dinâmico clássico, construída a partir de recursos de *hardware* de baixo custo, que ficou conhecida como π **FlowMR**, otimizada e baseada na exploração de múltiplos anéis de processamento. Esta máquina vem apresentando grande potencial de escalabilidade no decorrer de seu desenvolvimento, atingindo *speedups* da ordem de dezenas para programas com paralelismo suficiente, mesmo que não derivado de estruturas de dados regulares.

1.1 Motivação

Durante muito tempo acreditou-se que seria possível o aumento indefinido na frequência de relógio dos processadores, segundo previsto empiricamente pela “Lei de Moore”.³ De acordo com esta teoria, o aumento na frequência de operação, promovido pela redução da dimensão dos componentes eletrônicos que compõem os processadores *, seria sempre suficiente para garantir o ganho em desempenho computacional, para os mais diversos propósitos. Mas ainda em meados dos anos 60, foi lançada a hipótese de que haveria um limite superior, e essa frequência não poderia aumentar indefinidamente como pensado inicialmente. Com isso, uma das principais sugestões lançadas para garantir um ganho em desempenho computacional contínuo passou a ser o uso de processamento paralelo, ou seja, a execução concorrente de instruções, em nível da arquitetura de computadores, o que poderia proporcionar melhores desempenhos com a mesma frequência de relógio.⁴⁻⁷

Por muitos anos a hipótese inicial permaneceu intacta e perfeitamente coerente com a experiência verificada com a evolução da tecnologia, de forma que era realmente possível garantir um relativo ganho em desempenho computacional, com o lançamento de novos modelos de processadores, a partir de implementações mais modernas, como pode ser visto no gráfico da Figura 1. Entretanto, nas últimas décadas, foi possível verificar com clareza que a outra hipótese estava ainda mais correta, e foi encontrado de fato um limite superior que não permitia que a estratégia anterior pudesse seguir adiante.^{8,9} Atualmente, cada vez mais unidades de processamento são adicionadas aos processadores comerciais, formando múltiplos núcleos por pastilha, sendo esta uma das únicas fontes de evolução que se mostra capaz de oferecer um ganho em desempenho computacional contínuo e confiável.

* A redução dos componentes promove a maior proximidade entre eles, o que garante um tempo menor para a realização de operações equivalentes, por questões puramente físicas.

Figura 1 – Nas últimas cinco décadas a quantidade de transistores em sistemas integrados dobraram a cada período de dois anos, bem como a frequência de processamento que seguia um crescimento proporcional, até o início dos anos 2000.

Fonte: WALDROP⁹

Apesar desta constatação, pouca modificação foi realizada em nível de arquitetura de computadores ao longo deste período. O modelo de computação utilizado ainda nos dias atuais possui poucas alterações significativas, com relação a exploração mais eficiente de recursos paralelos, utilizando ainda o mesmo formato de execução sequencial que se tinha referência quando esta evolução ainda contava com o aumento na frequência do relógio para identificar um ganho em desempenho, principalmente com a finalidade de garantir compatibilidade com versões e modelos de arquiteturas legadas. Dessa forma, este modelo computacional ainda possui características puramente sequenciais, apesar de contar com processadores com múltiplos núcleos para caracterizar sua evolução.

Segundo este modelo de computação, a exploração dos recursos paralelos dos processadores deve ser realizada exclusivamente a partir de métodos de programação específicos, mesmo em nível de sistema operacional ou compiladores. Afinal, muito esforço teve de ser empregado pelos desenvolvedores destes sistemas, a fim de garantir que estas soluções possam explorar o paralelismo disponível nos processadores. Portanto, note que apenas o algoritmo do programa em execução é capaz de proporcionar a exploração deste recurso em processadores baseados neste modelo de computação convencional. Por este motivo, o processo de exploração de paralelismo em arquiteturas convencionais acaba ficando demasiado complexo, exigindo do programador grande conhecimento da arquitetura do processador, bem como técnicas e métodos muito específicos para conseguir extrair estas capacidades de forma otimizada.

O estado da arte de arquiteturas de computadores paralelas com múltiplos núcleos convencionais encontra-se no nível de até dezenas de unidades de processamento, como pode ser visto nas famílias de processadores comerciais lançadas mais recentemente,¹⁰ apesar de haver estudos com milhares de unidades de processamento convencionais,¹¹ bem como computadores implementados com esta quantidade de recursos em super computadores (mas geralmente utilizando *clusters*), como pode ser visto na lista do *Top 500*, que trata sobre os principais computadores com os melhores desempenhos possíveis de serem alcançados na atualidade.¹²

Desde o advento da computação, muito esforço foi empregado para a exploração de paralelismo em nível de arquitetura de computadores. Diversos paradigmas de processamento paralelo foram propostos a fim de garantir grande capacidade para ganho em desempenho, sem a necessidade de intervenção direta do programador. Dentre as alternativas já bastante exploradas na literatura, uma que se destaca bastante por sua capacidade para exploração de paralelismo é o paradigma da computação dirigida por dados. Segundo este modelo, a execução de instruções é controlada apenas pela simples disponibilidade de todos os dados necessários para fazê-lo. Na prática, é também necessária a disponibilidade de recursos na máquina para a sua execução efetiva.^{4,6,13-15}

Diversas implementações foram propostas para a exploração deste paradigma de computação paralela, dentre as quais pode-se destacar a Máquina a Fluxo de Dados de Manchester (que será referenciada como **MDFM** deste ponto em diante). Esta implementação particular utiliza o modelo de computação a fluxo de dados dinâmico, o que permite grande capacidade para exploração de paralelismo. Além disso, a **MDFM** ainda conta com uma vasta literatura, o que a torna um excelente ponto de partida para a pesquisa na área, além de contar com uma estrutura de implementação simples e de fácil entendimento.

1.2 Objetivos

Uma das primeiras análises realizadas neste trabalho foi o estudo sobre a viabilidade da implementação da arquitetura da **MDFM** com o uso de *hardware* moderno. Este estudo foi realizado ainda durante o desenvolvimento da dissertação de mestrado do presente autor, quando foi possível verificar uma baixa capacidade para ganho em desempenho computacional a partir desta aplicação, utilizando **FPGAs** de baixo custo. Deste ponto em diante, foram buscadas alternativas que permitissem uma real evolução deste modelo, para exploração de paralelismo massivo. Uma das primeiras otimizações neste sentido se deu diante da necessidade de aumentar o tráfego de informações entre as unidades que compõem sua estrutura por unidade de tempo, a fim de otimizar a utilização das unidades de processamento. Esta solução, que exigiu modificações marcantes com relação à arquitetura de origem, introduziu um conceito que recebeu o nome de *dutos de largura*

de banda.

Neste modelo, foi possível identificar uma baixa capacidade para a exploração de uma quantidade maior de recursos de processamento. É importante destacar que a própria **MDFM** possuía uma quantidade fixa de 20 unidades de processamento em sua composição definitiva.¹⁴ Em contrapartida, nas primeiras versões deste protótipo, havia uma limitação para a exploração de mais de 10 unidades, principalmente devido a introdução dos dutos de largura de banda, que acabou adicionando circuitos complexos, que não permitiam a exploração de mais recursos de processamento computacional nesta máquina.

Desta forma, um dos principais objetivos deste trabalho é a verificação sobre a viabilidade para a exploração de uma quantidade ilimitada, a priori, na quantidade de recursos de processamento, seguindo as estratégias para a exploração dos sistemas baseados em múltiplos núcleos da atualidade. Além de procurar garantir o melhor compromisso em escalabilidade possível, a partir dos recursos disponíveis para esta implementação - utilizando **FPGAs** de baixo custo - em uma linguagem de descrição de hardware simples, que garante uma boa capacidade para documentação e fácil distribuição.

No decorrer da implementação e análise parcial dos resultados foi possível verificar o impacto no desempenho desta máquina para a exploração de programas que apresentam um pequeno aumento na granularidade do seu conjunto de instruções. É possível verificar que existem instruções de granularidade mais grossa, em comparação com a granularidade convencional do modelo a fluxo de dados, no conjunto de instruções da **MDFM**, como a realização de cálculos trigonométricos por exemplo,¹⁶ que acabam por beneficiar o ganho em desempenho para execução dentro deste modelo de computação, de acordo com o acréscimo de recursos computacionais. Esta constatação levou a determinação desta meta pontual, qual seja um pequeno aumento na granularidade da máquina, a fim de verificar o impacto sobre seu desempenho global.

1.3 Estrutura da tese

No Capítulo 2 será introduzida uma série de fundamentações teóricas importantes para a construção deste trabalho. Em seguida, no Capítulo 3, será detalhado brevemente o princípio de funcionamento do trabalho que originou o desenvolvimento desta tese, um protótipo funcional do modelo a fluxo de dados que ficou intitulado como π **Flow**, bem como sua evolução imediata a π **FlowBW**. No Capítulo 4 serão apresentadas as principais modificações realizadas no escopo da π **FlowBW**, para sua operação dentro do Multianel e o Capítulo 5 irá detalhar o princípio de funcionamento desta nova arquitetura. Finalmente, o Capítulo 6 trará os principais resultados e discussões obtidos durante o desenvolvimento deste trabalho e o Capítulo 7 encerra este documento com as principais conclusões e sugestões de trabalhos futuros que podem ser explorados a partir deste objeto de estudos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão introduzidas as principais ferramentas e fundamentos utilizados para o trabalho desta tese, para que então seja possível utilizar destes conceitos de forma mais compreensível e livre no decorrer do texto.

2.1 FPGA

As **FPGAs**, acrônimo derivado da expressão em inglês *Field Programmable Gate Arrays*, são dispositivos de silício pré-fabricados que podem ser programados eletricamente, com a finalidade de se tornarem praticamente qualquer tipo de sistema ou circuito integrado digital.¹⁷ É possível citar muitas vantagens das **FPGAs** sobre os circuitos integrados, popularmente conhecidos como **ASICs**. Diferentemente da construção destas soluções integradas, que já são fabricadas de acordo com suas características de funcionamento específicas e que contam com uma imensa capacidade para a disposição de componentes integrados, as **FPGAs** são fabricadas por um processo de construção bastante similar, mas que permitem a escrita elétrica nas células de sua composição, garantindo sua versatilidade para a realização de qualquer operação lógica. Normalmente, os circuitos integrados levam de meses a anos para sua construção, considerando todo o planejamento até sua implementação efetiva, enquanto que as **FPGAs** podem ser configuradas em segundos e inclusive serem reconfiguradas em caso de algum equívoco no seu processo de planejamento, por simples modificação de linhas de código. Além disso, em contraste com a quantia de até centenas de milhares de dólares para o processo de construção de um circuito integrado, uma **FPGA** pode ser adquirida até por apenas algumas dezenas de dólares.

Em comparação com um circuito integrado, este caráter flexível de uma **FPGA** traz algumas desvantagens como: maior área de ocupação; atraso para a realização de operações equivalentes; e maior consumo de energia. Uma **FPGA** exige aproximadamente de 20 a 35 vezes mais área para a implementação de uma solução equivalente, de acordo com o processo de fabricação das células convencionais em circuitos integrados. A perda de desempenho em frequência de operação chega a ser de 3 a 4 vezes quando comparados com sua contraposição em **ASICs**. E, em geral, acaba por consumir até 10 vezes mais energia elétrica para a realização das mesmas operações. Portanto, é possível verificar este compromisso entre área, frequência de operação e consumo de energia, em detrimento de sua capacidade para reutilização de um “sistema integrado”.¹⁷ Apesar destas desvantagens, a flexibilidade oferecida por uma **FPGA**, em comparação com o processo de fabricação de um circuito integrado, ainda é bastante relevante para a construção de soluções eletrônicas, principalmente em nível de prototipação.

O princípio de funcionamento destes dispositivos pode ser visto como uma evolução natural da tecnologia para a construção de circuitos eletrônicos a partir de reescrita elétrica.¹⁸ A **FPGA** moderna foi introduzida no mercado a partir da empresa **Xilinx**, no ano de 1984, em um dispositivo contendo apenas 64 blocos lógicos e 58 conexões de entrada e saída. Apenas para fins de comparações, os modelos mais recentes de **FPGAs** contam com valores que chegam ao total de 330.000 blocos lógicos equivalentes e aproximadamente 1100 conexões pré definidas de entrada e saída.^{17,19} Como pode ser visto na Figura 2, estes dispositivos consistem de um arranjo de blocos lógicos programáveis de diferentes naturezas, incluindo circuitos de lógicas combinacionais de propósito geral, memórias e multiplicadores, dispostos de uma tal maneira que todos estes componentes são conectados por rotas de comunicações programáveis. Este arranjo é cercado por blocos de entrada e saída também programáveis, que permitem que o circuito seja capaz de se comunicar com o mundo externo. Esta composição é suficiente para garantir a capacidade destes dispositivos para programar funções generalizadas e abstratas mesmo depois de sua fabricação completa em silício.

Figura 2 – Composição da estrutura básica de uma **FPGA**.

Fonte: KUON¹⁷

Estas matrizes de silício reprogramáveis são importantes para o processo de construção de protótipos por possuírem a mesma flexibilidade obtida na implementação de linhas de código de *softwares*, porém, aplicada para a geração de *hardwares*. De acordo com a disposição dos componentes que compõem a construção destes dispositivos, como pode ser visto na Figura 2, é possível destacar a natureza generalizada de sua composição, o que permite a criação de circuitos integrados exclusivos, destinados para a realização das mais diversas operações de forma isolada, sem a necessidade de concorrência por recursos físicos, como ocorre naturalmente em sistemas computacionais de propósito geral. As **FPGAs** são formadas por uma quantidade limitada de recursos pré-definidos, determinada pelo processo de sua construção, de acordo com as estratégias comerciais de suas fabricantes. Os blocos lógicos programáveis são compostos essencialmente por dois componentes principais:

- **Flip-Flops:** Circuitos digitais compostos por dois estados estáveis, utilizados normalmente para sincronizar lógicas e registrar níveis de estados entre diferentes ciclos de relógio; e
- **Look-Up Tables:** Toda e qualquer operação lógica combinacional disponível para uso nestes dispositivos são dispostas dentro de “tabelas verdades” em seus blocos lógicos de construção, a partir de listas pré-definidas de saídas para cada combinação de entradas possíveis.¹⁸

Nas **FPGAs** da Altera/Intel, utilizadas para a construção do objeto de estudos desta tese, os blocos lógicos são formados estritamente por estes componentes, contando com configurações bastante restritas da fabricante. De acordo com as especificações e levantamentos realizados pela própria empresa, esta composição garante uma eficiência bastante superior em comparação com seus principais concorrentes do mercado.²⁰ Os Módulos de Lógicas Adaptáveis, em tradução literal do acrônimo **ALM**, do inglês *Adaptive Logic Module*, são formados de acordo com o diagrama da Figura 3. Trata-se de um circuito de lógica combinacional, com capacidade para até 8 entradas específicas, com até duas saídas exclusivas, de acordo com as configurações determinadas pelo código de descrição de hardware que se deseja implementar. A realização da operação lógica prevista por estes blocos são armazenados em registradores de estados primitivos, compostos essencialmente por cadeias de *Flip-Flops*, que tem sua operação coordenada pelo relógio do sistema, o que também pode ser definido a partir da solução proposta pelo código de descrição.

Nestes dispositivos, a reconfiguração do circuito é realizada a partir das linguagens de descrição de hardware, tradução literal para o termo em inglês *Hardware Description Language* (**HDL** como referenciado na literatura), linguagens estas adotadas para esta finalidade principalmente a partir do advento do **VHDL**. As linguagens de descrição de hardware nada mais são do que um conjunto de anotações, bastante similares à linguagens de programação de software, utilizados para a modelagem de funções lógicas de circuitos

Figura 3 – Diagrama de Blocos de um Módulo de Lógicas Adaptáveis da Altera / Intel.

Fonte: ALTERA INC.²⁰

ou sistemas digitais. O **VHDL** foi proposto pelo departamento de defesa dos EUA em meados dos anos 70, com a finalidade de possibilitar o projeto, a documentação e a análise de suas soluções de hardware para a composição de circuitos integrados naquela época.^{18,21} O **VHDL** é uma linguagem de descrição de hardware com nível de abstração baseada em registradores e que garante grande controle sobre as especificações destes componentes de estado, presentes de forma explícita na composição das **FPGAs**, como mostrado na Figura 3. Em meados dos anos 90, diversas empresas desenvolveram métodos e soluções completas que possibilitavam a gravação do circuito descrito neste tipo de documento, diretamente nos dispositivos reconfiguráveis existentes naquela época, como as **CPLDs** e as **FPGAs**, em um processo de compilação bastante rigoroso, que ficaram conhecidos como sintetizadores de hardware, possíveis de serem alcançados a partir da geração de um mapeamento dos recursos lógicos em um padrão de síntese conhecido como *netlist*.

2.2 Bluespec *SystemVerilog*

Ao longo da evolução das linguagens de descrição de hardware que foram surgindo de acordo com as necessidades de cada época, foi possível verificar uma evolução natural no processo de elaboração sob o nível de abstração tratado por estas ferramentas. Diagramas esquemáticos e descrições em linguagens de baixo nível como o *SPICE**, eram suficientes para a descrição e documentação de sistemas em um nível de abstração que levava em

* Linguagem destinada para a descrição de circuitos em nível de portas lógicas e componentes eletrônicos, usados principalmente para a simulação de circuitos eletrônicos integrados.²²

consideração as interconexões realizadas entre componentes eletrônicos e portas lógicas. É importante observar que dependendo da complexidade do circuito que se deseja implementar, este nível de abstração é totalmente suficiente, e muitas vezes, necessário para a documentação e descrição precisa de uma solução. Entretanto, de acordo com a evolução da tecnologia, é natural que a complexidade dos circuitos digitais evoluam, apresentando necessidades não verificadas anteriormente. Perceba que este nível de abstração mais restrito, é capaz de representar qualquer tipo de circuito e necessidade específica, porém, seu processo de construção pode acabar se tornando demasiado laborioso e por vezes até impraticável. Note que neste nível de detalhamento a descrição de hardware pode não deixar clara qual a finalidade do circuito, enquanto que com uma linguagem com nível de abstração mais alto fica evidente os propósitos do circuito, facilitando o processo de documentação.

Principalmente pela necessidade de especificações mais abstratas e diretas, que diversas linguagens para descrição de hardware com nível de abstração mais alto começaram a surgir dos anos 60 em diante, até o momento quando o departamento de defesa dos EUA introduziu o **VHDL**, que acabou se tornando um padrão **IEEE** para especificações de hardware, no ano de 1987.²³ Paralelo à construção do **VHDL** estava sendo criado também o **Verilog**, a partir da iniciativa privada, por uma empresa chamada *Gateway Design Automation*. No ano de 1985, esta linguagem foi introduzida no mercado, desenvolvida com a finalidade de possibilitar a criação de circuitos de forma mais prática e intuitiva. Como resultado, esta linguagem possui muito menos construções em comparação com o **VHDL**, possuindo até algumas deficiências, por deixar algumas especificações indefinidas, permitindo a introdução de certas falhas na construção do projeto proposto. Porém, sua semântica é muito menos complexa e a legibilidade do código descrito a partir desta linguagem é bem mais simples em comparação com o **VHDL**. O **Verilog** acabou se tornando um padrão **IEEE** também, no ano de 1995.²⁴ Atualmente, ambas as linguagens ainda são largamente exploradas para a especificação de circuitos eletrônicos. Além disso, muitos simuladores e ferramentas para a geração de *netlist*, destinados para síntese de soluções, foram desenvolvidos desde a introdução destas linguagens para a automação da análise e implementação destas soluções em **FPGAs** e **CPLDs**.²¹

O nível de abstração de uma linguagem de descrição de hardware se refere à quantidade de decisões de “projeto” que devem ser especificadas para a descrição precisa de um circuito, ou seja, quanto menos decisões forem necessárias para a especificação, maior será o nível de abstração explorado por ela. Conforme pode ser visto na Figura 4, nos dias atuais, a quantidade de linhas de código, mesmo para a solução de problemas corriqueiros, acabou se tornando bastante impraticável, uma vez que sua evolução possui um caráter exponencial. Mesmo com o uso de **IPs**[†] e bibliotecas surgidas ao longo do

[†] Acrônimo para a expressão do inglês *Intellectual Property*, que tratam de soluções pré definidas

tempo, para linguagens que operam em nível de abstração de registradores, como o **VHDL** e o **Verilog**, a quantidade de linhas de código para a descrição de soluções acabou se tornando impeditiva. Por este motivo, diversas outras linguagens de descrição de hardware, com nível de abstração ainda mais alto, acabaram surgindo nas últimas décadas, dentre as quais pode-se destacar o **SystemVerilog**²⁵ e o **SystemC**,²⁶ como exemplo. Estas linguagens referem-se a soluções que permitem maior facilidade tanto para simulação quanto para implementação de circuitos eletrônicos, para uso em dispositivos reprogramáveis, por permitirem a abstração de diversas características que deveriam ser descritas explicitamente em linguagens com nível de abstração mais baixo.

Figura 4 – Complexidade dos níveis de abstrações para a especificação de circuitos digitais ao longo dos anos.

Fonte: BLACK *et al.*²⁶

Ambas as linguagens destacadas já são suficientes para reduzir a complexidade do código necessário para a descrição de soluções completas para o projeto de circuitos digitais, destinados para a síntese em dispositivos reprogramáveis ou mesmo para a construção de circuitos integrados. Entretanto, uma outra linguagem que se destaca bastante neste contexto por se basear no paradigma da programação funcional, é o **Bluespec**. Esta linguagem é uma extensão da linguagem de alto nível **Haskell**, destinada à descrição de hardware, adaptada para lidar com um Sistema de Reescrita de Termos (em tradução literal sob a expressão *Term Rewriting System*, que define o acrônimo **TRS**), estratégia

de circuitos específicos de propriedade intelectual de outros desenvolvedores e que já foram largamente explorados e testados para as mais diversas aplicações.

utilizada para a elaboração estática da solução descrita a partir desta ferramenta, e que posteriormente foi reprojeta para compatibilidade com o **SystemVerilog**, transformando-se na linguagem que ficou conhecida como **Bluespec SystemVerilog** (que será referenciada como **BSV**). O BSV foi proposto inicialmente com a finalidade de oferecer ganho em produtividade, a partir de uma abordagem radicalmente diferente para síntese, em uma ferramenta de alto nível.^{27,28} Além disso, a partir de janeiro de 2020, o código desta linguagem foi aberto, fornecendo com isso garantia de suporte estendido, mesmo que pela comunidade de ferramentas livres para construção de hardware.

Diferentemente de outras linguagens que tentam aumentar o nível de abstração para a construção de circuitos digitais de forma baseada em estruturas convencionais de programação sequencial para a especificação comportamental do projeto, o **BSV** foi construído com base em uma semântica tradicional para descrição de hardware, o **SystemVerilog**. Dessa forma, apesar de possuir este nível de abstração mais alto, esta linguagem se torna mais natural para profissionais que já possuem familiaridade com estas construções, além de contar exclusivamente com o subconjunto de funcionalidades sintetizáveis do **SystemVerilog**, o que garante sua capacidade plena para a geração de código destinado para síntese em **FPGAs**. No **BSV**, a descrição do comportamento dos circuitos é expressa a partir de um conjunto de ações atômicas, compostas por máquinas de estados finitos concorrentes e cooperativas, bastante familiar e intuitivo para projetistas de hardware, apenas expandido por sua garantia na atomicidade destas operações.²⁷

O **BSV** utiliza o modelo padrão e familiar para a definição das estruturas de hardware do **Verilog** e do **VHDL**, conhecidas como *Módulos*, estruturas capazes de determinar o escopo da ação de um circuito exclusivo, composto por suas entradas e saídas características. Uma extrapolação muito útil, possível de ser integrada no **BSV** é sua capacidade para a instância de módulos descritos diretamente em **Verilog** ou **VHDL** (para a comunicação com sistemas legados ou **IPs** externos). As definições de entrada e saída dos módulos são alcançadas a partir de uma construção conhecida como *Interfaces*, com seu sistema de tipos também herdados do **SystemVerilog**. Toda estrutura comportamental destas interfaces são determinadas a partir de uma outra construção da linguagem, conhecidas como *Métodos*, especificados por uma coleção de fragmentos de regras, governados pelo sistema de reescrita de termos nativo do compilador do **BSV**, que opera sob todos os âmbitos para a determinação do comportamento de toda a solução, em um sistema de ações atômicas.

Toda estrutura comportamental dos módulos presentes nas construções do **BSV**, ou seja, sua estrutura para o registro e manipulação de estados, diferentemente das construções síncronas nativas, presentes em descrições de nível **RTL**, como os processos do **VHDL** e os blocos *always* do **Verilog**, no **BSV** são determinadas a partir de um conjunto de ações atômicas “guardadas”, como definidas em seu contexto. Tanto o comportamento de

operação dos módulos, que são definidos por um conjunto de regras atômicas, como dos métodos das interfaces, são governados por um sistema de reescrita de termos, que trata de uma modelagem matemática de redução, que passo a passo em uma análise restrita sobre as possibilidades de manipulações neste conjunto de operações globais, resultam em uma forma normal, indivisível e perfeitamente única.^{29,30} A Figura 5 destaca cada uma das construções nativas desta linguagem e a forma como elas interagem entre si.

Figura 5 – Diagrama representativo dos módulos, regras e interfaces do **Bluespec SystemVerilog**.

Fonte: Adaptada de DAVE³¹

Desta forma, ambas as construções, tanto das regras, que determinam o comportamento dos módulos, quanto os métodos das interfaces, consistem de pares de elementos que possuem o formato $\langle \text{condição}, \text{ação} \rangle$. O identificador “condição” é a especificação de uma função puramente *booleana*, que determina univocamente quando uma regra pode ser disparada, ou quando um método pode ser invocado, dentro da análise do sistema de reescrita de termos. Já o identificador “ação”, determina explicitamente o que deve ser realizado no estado do circuito, quando a condição permitir a realização destas modificações. É importante destacar que uma regra pode invocar os métodos presentes nas interfaces de outros módulos, que por sua vez, podem invocar os métodos de ainda outros módulos. Portanto, apesar das regras estarem distribuídas dentro de múltiplos módulos encadeados, elas podem ser identificadas de forma modular, ou seja, com condições e ações encapsuladas dentro de cada módulo. As “condições” - definidas nos pares de elementos que determinam as permissões para a execução de regras ou métodos - que garantem a atomicidade das ações dentro do sistema de reescrita de termos, de forma nativa, de acordo com as definições da linguagem. Elas são determinadas segundo a especificação de cada módulo, indicando as consequências para a realização de eventuais operações

concorrentes, principalmente em elementos de estados primitivos. Mesmo espalhadas por entre múltiplos módulos, as regras do **BSV** possuem propriedades globais sob sua garantia de atomicidade, de forma que o comportamento de todo o sistema pode ser visto como uma composição sequencial para o disparo de regras.

No **BSV**, durante seu processo de compilação, toda regra é candidata para execução, em todo ciclo de relógio da solução proposta. Além disso, todas as ações definidas no escopo principal de uma regra são alocadas para execução simultaneamente, dentro de todo ciclo de relógio. Com isso, ao final de um determinado ciclo, todos os elementos de estado afetados por esta regra serão modificados de acordo com suas definições, e estarão prontos para a realização de qualquer nova modificação no próximo ciclo do relógio. O sistema de reescrita de termos característico do **BSV** garante que uma regra “não” será disparada somente se uma das três considerações seguintes forem satisfeitas:

- Um predicado, ou condição explícita é verdadeira. Estas condições podem ser adicionadas no processo de construção de uma regra ou método e, será incorporado na análise do sistema de reescrita de termos;
- Um método que concorre pelos mesmos recursos de determinada regra foi invocado dentro do mesmo ciclo de relógio. Os métodos possuem maior prioridade para execução em comparação com as regras;
- A execução de uma outra regra que concorre pela manipulação dos mesmos recursos já foi alocada para execução. É possível determinar a prioridade de cada regra de forma explícita, a partir de algumas diretivas definidas na construção da linguagem.

Este escalonamento verificado no processo de compilação do **BSV** determina, com precisão, as análises realizadas pelo sistema de reescrita de termos utilizado neste processo. Este atributo se baseia tanto nos predicados das regras, como nas condições implícitas definidas no processo de construção de todo módulo do **BSV**. É importante destacar que para a solução de circuitos muito complexos a aplicação deste formalismo matemático tende a ser muito mais consistente em comparação com uma verificação manual, que deve ser realizada explicitamente dentro de outras linguagens de descrição que não contam com este tipo de análise. A Figura 6 exemplifica o processo de escalonamento realizado pelo compilador do **BSV** durante a aplicação da análise do **TRS**.

Esta garantia de atomicidade para a execução de regras é uma poderosa ferramenta para estabelecer a construção correta de um circuito livre de conflitos e trata do principal diferencial do **BSV**, uma vez que esta solução deve ser garantida de forma manual e explícita em outras linguagens de descrição de hardware. Portanto, é possível destacar este formato de construção como um dos principais motivos que promovem o ganho em

Figura 6 – Diagrama representativo do escalonamento para a composição de regras dentro de um mesmo ciclo do relógio no **Bluespec SystemVerilog**.

Fonte: NIKHIL³²

produtividade proposto por esta linguagem. Por esta razão principal, que foi escolhido o **BSV** para a construção do presente trabalho, dentre tantas opções disponíveis no mercado.

O compilador do **BSV** é capaz de fazer a geração de um código **Verilog** livre de conflitos, mas além disso, ele ainda possibilita a geração de um código descrito em linguagem **C**. A saída em **Verilog** pode ser utilizada em simuladores padrões, ou compilados a partir de ferramentas exclusivas para a geração de *netlists*, para síntese em sistemas físicos. Já a saída em linguagem **C** pode ser compilada e executada a partir de compiladores padrões desta linguagem. Ambas as saídas possíveis de serem obtidas possuem definições precisas de execução com relação à sua definição explícita de ciclos de relógio e, ambas são suficientes para a geração de arquivos **VCD** para a análise das curvas de onda do circuito proposto, a fim de verificar sua correta composição.

2.3 Fluxo de Dados

No modelo de computação convencional as instruções são dispostas em um formato essencialmente sequencial. Este fator se dá por uma condição fundamental imposta naturalmente no modelo proposto pela arquitetura de *Von Neumann*, que conta com um sistema de controle central para a definição unívoca da sequência de ações necessária para a correta execução de programas. Para possibilitar a realização de operações paralelas, conforme proposto pela estratégia dos computadores baseados em múltiplos núcleos, este modelo de computação exige a replicação de toda, ou parte de sua construção e segue com múltiplas estruturas que em essência continuam apresentando características puramente sequenciais. Desta forma, estas construções podem apresentar propriedades de paralelismo, mas elas devem ser totalmente controladas a partir da implementação de um algoritmo que seja capaz de explorá-las. Este paradigma é normalmente identificado como computação dirigida por controle, por contar com este sistema “centralizador”, responsável por controlar todo o processo de execução de programas de forma explícita.

Foram propostos diversos modelos de computação ao longo do tempo, que buscam

sanar esta deficiência intrínseca ao modelo convencional. São exemplos imediatos de estratégias com essa finalidade, os modelos de computação dirigidos por demanda e por dados.¹³ No modelo dirigido por demanda, normalmente referenciado como máquina de redução, a necessidade por um valor específico “dispara” a operação que vai realizar toda computação necessária para calcular seu resultado. Já no modelo dirigido por dados, atribuído normalmente ao modelo a fluxo de dados, a simples disponibilidade dos operandos necessários para a realização de determinada ação “dispara” a execução das operações subsequentes, que devem ser realizadas para o tratamento destes novos valores.³³ Ambas as soluções já são suficientes para remover a necessidade de intervenção direta do programador no algoritmo para possibilitar a exploração de paralelismo e, esta é uma das principais motivações para o estudo destas arquiteturas não convencionais.

O presente trabalho foi desenvolvido para um estudo profundo acerca do paradigma da computação dirigida por dados, a partir de uma implementação específica, baseada no modelo a fluxo de dados dinâmico, que será tratada em detalhes na Seção 2.4. De acordo com este paradigma de computação, os programas podem ser representados por um grafo bidimensional direcionado, destinado para a exploração de granularidade fina. Conforme sua definição, de acordo com o formato de representação de seus programas, as instruções que podem ser executadas paralelamente são dispostas lado a lado, enquanto que as instruções que possuem dependência de dados, ou seja, que dependem da execução de outras instruções, são dispostas uma abaixo da outra.¹⁴ Um fator importante que pode ser destacado sobre este modelo de representação de programas, é que todo o paralelismo disponível pode ser visto em detalhes, imediatamente, bem como toda a dependência de dados existente na implementação do algoritmo. A seguir, é apresentado o Diagrama 7, suficiente para expor com clareza este formato de representação, bem como a transcrição de um código de programação, no Algoritmo 1, que poderia ser descrito por uma linguagem de alto nível, que ilustram esses pontos.

Este formato de representação é utilizado em diversas ferramentas gerais na computação, como o próprio **BSV**, que a utiliza para a elaboração estática, na composição de seu código **Verilog**, durante o processo de compilação, principalmente para a aplicação do sistema de reescrita de termos característico desta linguagem, como descrito na seção anterior. A representação gráfica de um programa para o modelo a fluxo de dados é feita

```
1 // ...  
2 b = 10;  
3 bsq = b*b;  
4 discr = bsq - 4*a*c;  
5 // ...
```

Algoritmo 1 – Algoritmo simples destinado para o cálculo do fator discriminante da fórmula de *Bhaskara*.

Figura 7 – Representação bidimensional do trecho de programa apresentado no Algoritmo 1 de acordo com a definição do modelo a fluxo de dados.

Fonte: Adaptada de NIKHIL³²

no formato de um grafo direcionado, com quantidade de entradas e saídas definidas de acordo com a estratégia de arquitetura proposta para sua implementação. Segundo este formato de representação, as instruções são determinadas pelos nós dos grafos e as arestas descrevem a dependência de dados entre estas instruções. Os círculos da Figura 7 têm a finalidade de representar a introdução de valores imediatos (como definido no modelo de *Von Neumann*), adicionados como entrada das instruções, que podem ser interpretadas como atributos dos nós deste grafo.

Em tempo de execução de um programa, segundo o modelo a fluxo de dados, os dados que serão tratados por cada instrução presente no grafo são organizados dentro de uma “estrutura” conhecida como *fichas*. De acordo com o grafo da Figura 7, as instruções presentes neste programa são sempre fixas e as fichas serão responsáveis por percorrer as arestas que determinam a dependência de dados do programa, em direção às instruções.

Perceba pelo diagrama que os processos que descrevem as operações de multiplicação,

presentes no centro da figura, podem ser realizados de forma paralela, desde que haja recursos suficientes para fazê-lo. Os valores das “variáveis” a e c , são determinados fora do escopo de representação destacado, e portanto, esta atribuição não se encontra dentro dos limites do grafo do modelo a fluxo de dados mostrado na figura. Note ainda que a operação de subtração presente no diagrama não pode ser realizada até que ambas as multiplicações tenham sido calculadas anteriormente. Uma implementação capaz de suportar a execução deste tipo de programa deveria possuir ao menos duas unidades de processamento para ser capaz de realizar a execução deste trecho de programa em três períodos de tempo, a ser determinado pelo desempenho destas unidades. Na Figura 7, a altura do grafo é definida como h , e trata de uma propriedade que pode ser utilizada para determinar precisamente o trecho sequencial deste programa.

Com relação à realização de computação a partir deste formato, é ainda bastante abstrata a representação do programa como um grafo apenas. Por este motivo, a Figura 8 foi utilizada para apresentar este mesmo grafo, trazendo-a para um novo formato, que já poderia ser comparado a uma linguagem de máquina de *Von Neumann*. Já é possível, a partir desta nova representação, verificar o equivalente aos “códigos de operações” de instruções capazes de realizar os cálculos aritméticos e operações de controle deste programa, bem como o que poderia ser identificado como o endereço destas instruções dentro de uma memória destinada para o armazenamento deste programa. A instrução identificada como **DUP** é bastante conhecida do modelo a fluxo de dados clássico, tendo a finalidade de duplicar o valor presente em sua entrada, para suas duas saídas, que podem ser direcionadas para novas instruções em ciclos subsequentes do relógio, durante a execução do programa. As instruções **MUL** e **SUB** tratam das operações aritméticas presentes neste trecho de programa e, referem-se às operações de multiplicação e subtração, respectivamente. Por fim, a instrução **OPT** é o operador de saída da máquina, capaz de encaminhar uma mensagem contendo o resultado do programa.¹⁶ É importante destacar que o formato mostrado nesta figura já possui muitas características exclusivas da Máquina de Manchester.

Apesar de já trazer um formato de representação mais relacionado à computação de programas, a representação gráfica ainda pode deixar um aspecto abstrato para o entendimento do princípio de funcionamento deste modelo. O trecho de programa descrito no equivalente a uma linguagem montadora, presente no Algoritmo 2, determina com detalhes a representação do grafo da Figura 8, em uma semântica possível de ser executada na estrutura de uma máquina que implementa este modelo. Note tratar-se da representação de um grafo apenas, determinando as informações descritas nos nós do grafo, as conexões destes nós e mesmo, eventuais valores adicionados de forma explícita, que podem ser tomados como os atributos dos nós. Perceba que este código pode ser gravado em uma memória de propósito geral e será suficiente para a descrição de todo o grafo de representação deste trecho de programa, que tem a finalidade de realizar o cálculo do

Figura 8 – Formato em grafo do modelo a fluxo de dados, para a representação do trecho de programa apresentado no Algoritmo 1, que pode ser executada por uma implementação.

Fonte: Adaptada de NIKHIL³²

fator discriminante da fórmula de *Bhaskara*. Desta forma, as fichas seriam responsáveis por resgatar estas informações quando conveniente, dentro de uma estrutura capaz de implementar o modelo a fluxo de dados.

Neste formato, cada linha do algoritmo é responsável por determinar um nó do grafo no modelo a fluxo de dados. O primeiro número determina o endereço, seguido pela instrução efetiva, definida para o nó corrente. Na segunda sequência de caracteres, são representados o endereço do nó de ligação da instrução, ou seja, a dependência de dados do programa, seguido pelo lado do nó, que esta instrução fará conexão com o nó subsequente[‡],

[‡] No modelo descrito por este programa, cada nó presente na representação gráfica do modelo a fluxo de dados, pode possuir o máximo de duas ligações às instruções subsequentes, esta quantidade de conexões entre nós é específico da implementação proposta pela Máquina de Manchester, e não é uma limitação do modelo de computação.


```
1 0KIL 0LBY
2 1DUP 3LBY 3RBYI10
3 2MUL 4RBY
4 3MUL 5LEW
5 4MUL 5REWI4
6 5SUB 6LBY
7 6OPT 0LBY
```

Algoritmo 2 – Algoritmo simples destinado para o cálculo do fator discriminante da fórmula de *Bhaskara* em formato de linguagem de máquina destinado para o modelo a fluxo de dados.

além do formato de pareamento que deve ser realizado pelos dados destinados para esta instrução. Outro valor que pode ser representado nesta segunda sequência de caracteres pode ser visto na linha 5 do algoritmo e apresenta o tipo de dado, aqui representado pela letra *I*, indicando tratar-se de um valor inteiro, seguido pelo dado adicionado de forma imediata no código do programa. Por fim, existe o suporte para a definição da segunda ligação do nó corrente, como pode ser visto na linha 2 do algoritmo, que possui exatamente as mesmas propriedades da segunda sequência, de acordo com este formato. Perceba que esta definição refere-se às especificações da Máquina de Manchester, se houvesse suporte para mais ligações por instruções, novas sequências equivalentes de informações poderiam ser utilizadas, por exemplo.

Uma outra caracterização importante comum aos modelos a fluxo de dados clássicos, trata da capacidade de uma implementação para a manipulação dinâmica do grafo, em tempo de execução do programa, que possui uma relação direta com o tamanho do grafo de representação. Todas as especificações e exemplos apresentados nesta seção, referem-se ao modelo estático do fluxo de dados, ou seja, a quantidade de nós presentes no seu grafo de representação é inalterada durante a execução do programa. Este tipo de implementação é interessante para exemplificar a operação do modelo. Entretanto, para fins práticos, o tratamento de programas mais complexos, acaba por tornar esta estratégia um tanto limitada, por induzir um aumento expressivo na necessidade de armazenamento do código fonte do programa em execução. No modelo a fluxo de dados dinâmico, existem algumas instruções e construções específicas que possibilitam a identificação do equivalente a uma “instância” para a execução de determinada instrução. Este modelo permite a realização de laços e recursividade por exemplo, pois trata-se de um artifício que permite a execução da mesma instrução, em uma instância de iteração específica. No modelo estático, todas as instruções necessárias em um nível de iteração de um laço, devem ser descobertas ainda no processo de compilação, devendo ser criadas explicitamente no código do programa para a exploração deste recurso dentro da máquina, ou copiadas em tempo de execução (método conhecido como “*Code Copying*”).

O modelo a fluxo de dados tem grande impacto na representação de programas.

Esta representação é totalmente distinta da convencional, que considera apenas uma sequência de ações capazes de adicionar um contexto dentro de determinado algoritmo. Na verdade, o formato convencional tende a ser um pouco mais intuitivo a princípio, principalmente por conta da semântica e sintaxe das linguagens de programação, em especial as que consideram um paradigma de programação imperativa, que é baseada exatamente na mesma estratégia de representação sequencial de ações, o que torna mais simples o processo de compilação. Com isso, note que esta noção mais intuitiva possui relação principalmente com a ativação de uma memória afetiva com a qual os programadores estão mais acostumados a lidar. O formato da representação do modelo a fluxo de dados, estabelece benefícios principalmente para a visualização do paralelismo disponível em um programa, destacando suas propriedades paralelas e dependência de dados imediatamente. Desta forma, é importante perceber que o modelo computacional estabelece os princípios e conceitos relacionados a como as operações computacionais serão realizadas durante a execução de programas (o que ficou conhecido como “mecanismo de controle”). Diversas propostas e implementações específicas de máquinas que operam segundo este paradigma foram propostas, principalmente no decorrer dos anos 80, como será descrito em detalhes na sequência, com maior aprofundamento sobre uma implementação específica.

2.4 Máquina a Fluxo de Dados de Manchester

Do final dos anos 70 em diante, inspirados principalmente pelo trabalho desenvolvido por *Jack Dennis* no MIT,^{4,34,35} começaram a surgir diversas implementações de máquinas baseadas no modelo a fluxo de dados. Grandes centros de pesquisa se aprofundaram neste modelo, a partir de pesquisas tanto teóricas, como aplicadas, dentre as quais é possível destacar alguns exemplos notáveis como aqueles desenvolvidos no MIT nos EUA,⁴ em Manchester na Inglaterra⁵ e em Ibaraki no Japão.⁶ Muitas conclusões e perspectivas puderam ser extraídas destes trabalhos que forneceram uma base sólida para o desenvolvimento do presente trabalho.^{13,33}

Em particular, a principal fonte de inspiração para o desenvolvimento desta tese derivou-se da Máquina a Fluxo de Dados de Manchester, um poderoso processador, baseado no modelo a fluxo de dados dinâmico, que possuía um conjunto de instruções um tanto misto, com instruções de granularidade extremamente finas, como proposto pelo próprio modelo, mas que também contava com algumas instruções de granularidade relativamente mais grossas, como a realização de operações trigonométricas, por exemplo. A topologia da MDFM era em anel, onde as *fichas* transitavam entre as diferentes unidades de sua composição, em busca de todas as informações necessárias para a sua execução efetiva, em um formato que se assemelha de várias formas a uma estrutura de *pipeline*. Aproveitando-se desta topologia, a execução das instruções promoviam a geração de outras novas que retornavam ao primeiro estágio e o ciclo era repetido.

Figura 9 – Diagrama de representação da visão geral da Máquina a Fluxo de Dados de Manchester.

Fonte: Elaborada pelo autor.

A Figura 9 apresenta a visão geral deste processador, como definido em sua especificação. A Unidade de Chaveamento (*Switch E/S*), tinha a finalidade de receber e enviar informações para o mundo externo. A Fila de Fichas era composta por um *buffer*, caracterizado por uma grande fila (*FIFO*) de armazenamento interno, responsável por manter quantos dados fossem necessários, a fim de regular o fluxo dos novos dados gerados durante a execução de programas.¹⁴ A Unidade de Pareamento, era responsável por combinar dados destinados para a mesma instrução, ou seja, era a unidade responsável por indicar quais instruções que já estavam habilitadas para execução. Esta unidade ainda possuía um anexo conhecido como Unidade de Sobrecarga, que era utilizada para garantir uma capacidade para o armazenamento de absolutamente todos os dados que passavam pela Unidade de Pareamento, que era endereçada a partir de uma função *hash* não perfeita. A Unidade de Instruções era composta essencialmente por uma memória RAM, com a finalidade de manter armazenado todo o código do programa (grafo do fluxo de dados) em execução na máquina. Por fim, a Unidade de Processamento era responsável por distribuir os dados que chegavam a ela - que já possuíam todas as informações necessárias para a execução de sua instrução - à primeira Unidade Funcional livre que fosse identificada. Cada uma destas unidades serão descritas em detalhes na sequência.

Neste modelo de implementação, os dados percorrem pelas unidades que compõem a máquina, a partir da estrutura destacada anteriormente, conhecida como *ficha*. Este elemento era responsável por manter armazenada todas as informações necessárias para a execução de instruções, sofrendo diversas modificações no decorrer de sua trajetória até alcançar as unidades responsáveis pela realização de seu processamento efetivo. Diante da criação de um novo elemento de dados na estrutura da máquina, as fichas possuíam o formato descrito pela Equação 2.1. Note que uma ficha contém todas as informações

necessárias para resgatar as informações de uma instrução na estrutura da máquina. Ela era formada por uma grande cadeia de *bits* com largura de *96-bits*, organizada de forma a identificar todas as características de representação de um nó do grafo. O identificador indicado como “dado”, era composto por uma palavra de *37-bits*, dos quais *5-bits* eram reservados para o armazenamento do tipo de dado carregado pela ficha, como valores inteiros, reais, *booleanos*, dentre outros,¹⁶ e os *32-bits* restantes levavam o valor efetivo que seria tratado pela instrução à qual a ficha se destinava.

$$< \text{dado}(37\text{-bits}), \text{rótulo}(36\text{-bits}), \text{destino}(22\text{-bits}), \text{marcador}(1\text{-bit}) > \quad (2.1)$$

A próxima palavra determinada na definição de uma ficha era identificada como “rótulo”: que era responsável por garantir as características dinâmicas do modelo a fluxo de dados. Ele era composto por três campos de *12-bits*, cada qual identificando um tipo de capacidade dinâmica diferente na máquina. O primeiro campo era conhecido como “Índice” que tinha a finalidade de lidar com dados estruturados, identificando o índice de um vetor, por exemplo, o qual a ficha corrente deveria processar. O segundo e o terceiro campos definiam a “Cor” da ficha. Esta palavra de *24-bits* identificava os outros dois principais rótulos que determinavam as propriedades dinâmicas desta implementação, capazes de verificar a ocorrência de instruções “re-entrantes”. O “Nível de Iteração” era um rótulo gerenciado pelo próprio programa em execução e tinha a finalidade principal de permitir a realização de operações de laços na **MDFM**, identificando a qual iteração uma ficha pertencia. Por fim, o último rótulo possível de ser explorado era conhecido como “Nome de Ativação”: este atributo possibilitava a identificação de uma propriedade dinâmica que possuía seus recursos gerenciados por uma estrutura física da máquina, com a finalidade de permitir distinguir as diferentes funções e diferentes instâncias de uma mesma função em tempo de execução. Note portanto, que este rótulo era o principal responsável pela realização de recursividade e possuía um papel fundamental para a exploração de paralelismo em larga escala. A informação global dos nomes de ativação era armazenada em um registrador de estados primitivo na estrutura da máquina, que tinha seu valor apenas incrementado diante da ocorrência de uma nova chamada de função em tempo de execução. Em essência, este rótulo dinâmico tinha a finalidade de identificar univocamente em qual instância recursiva, por exemplo, que uma determinada ficha estava inserida.

$$< \text{endereço}(18\text{-bits}), \text{lado}(1\text{-bit}), \text{pareamento}(3\text{-bits}) > \quad (2.2)$$

Seguindo a especificação de cada identificador presente na composição de uma ficha, o “destino” tratava de uma palavra de *22-bits* que trazia o endereço da memória RAM da Unidade de Instruções (que será detalhada na sequência), onde estava armazenada a instrução do programa a qual a ficha se destinava, este campo é formado de acordo com

a especificação definida na Equação 2.2, sendo composto ainda, pelo lado o qual o dado da ficha corrente seria inserido na sua instrução, bem como a função de pareamento que deveria ser analisada por ela. Por fim, o “marcador”, na **MDFM** tinha a finalidade de identificar as fichas que eram destinadas para a saída da máquina, para apresentação dos resultados do programa.

2.4.1 Fila de Fichas

Um diagrama ilustrativo da Fila de Fichas pode ser visto na Figura 10. Esta unidade era composta por uma grande fila de armazenamento interno, capaz de suportar o total de até 32K palavras de 96-bits de largura, a fim de acomodar as fichas descritas na Equação 2.1, como dispostas na saída da Unidade de Processamento e mesmo na entrada de fichas a partir do sistema de Chaveamento de Entrada e Saída. Dessa forma, a principal função desta unidade era atuar como um *buffer* para a estrutura da máquina, a fim de equilibrar um possível desbalanço que podia ocorrer entre a geração de novas fichas na Unidade de Processamento e o consumo de dados, pelas unidades de Pareamento e de Instruções.

Figura 10 – Diagrama ilustrativo da Fila de Fichas na **MDFM**.

Fonte: Adaptada de GURD¹⁴

A **MDFM** era composta por um total de 20 Unidades Funcionais, como eram chamados os elementos de processamento da máquina. Considere o exemplo de um ciclo de relógio que apresente um sincronismo perfeito entre essas unidades, em um momento

no qual apenas instruções de saídas duplas fossem geradas. Neste exemplo hipotético, um total de 40 fichas seriam enviadas simultaneamente a partir da Unidade de Processamento, e as outras unidades da máquina não seriam capazes de absorver essa quantidade de dados imediatamente. Sendo esta a principal função da Fila de Fichas, garantir um controle entre o fornecimento e consumo de nova fichas.

2.4.2 Unidade de Pareamento

A Unidade de Pareamento era composta principalmente por uma memória de armazenamento interno capaz de acumular o total de 1,25M fichas de endereçamento associativo. Esta unidade era composta por seis registradores, dispostos em um formato de *pipeline*, no fluxo principal da composição do anel e dois *buffers* intermediários para comunicação com a Unidade de Sobrecarga, como pode ser visto no diagrama mostrado na Figura 11. O endereçamento associativo desta unidade era alcançado a partir da aplicação de uma função *hash* sobre as informações da composição da ficha, como seu endereço e rótulo, que possuem valores equivalentes para os pares de fichas destinadas para a mesma instrução, dentro do mesmo nível de iteratividade, sob os aspectos dinâmicos da máquina. Por se tratar de uma função *hash* não perfeita, existia a possibilidade de haver conflito de endereços para fichas que não fossem destinadas para a mesma instância de uma instrução.

Na **MDFM**, existiam fichas destinadas para instruções de entrada única, que eram reconhecidas por passarem diretamente pela Unidade de Pareamento, sem a necessidade de aguardar por uma ficha parceira. As fichas que possuíam essa característica eram indicadas explicitamente no código do programa, a partir da função de pareamento **BY**, derivado da expressão em inglês “*Bypass*”, como pode ser visto no Algoritmo 2. Estas fichas de passagem direta simplesmente eram encaminhadas do *buffer* de entrada desta unidade para sua saída e naturalmente não consumiam seus recursos.

Na ocorrência de fichas destinadas a pareamento, elas eram encaminhadas aos *buffers* intermediários desta unidade e tinham o endereço de seu armazenamento calculado, obtido a partir de uma composição de suas informações. O tipo de ação que deveria ser promovido pela ficha recém chegada era analisado diante da resposta desta leitura. Existiam diversas ações possíveis de serem realizadas, como definido no manual da máquina.¹⁶ Entretanto, as ações mais comuns observadas a partir das últimas soluções de compilação, era de manter a ficha recém chegada armazenada neste endereço, caso ele estivesse vazio, ou extrair a ficha armazenada neste endereço, se ela fosse destinada para a mesma instrução da ficha corrente, formando o par de saída característico desta unidade, como definido na Equação 2.3. Porém, se já houvesse uma outra ficha armazenada neste endereço, que não fosse destinada para a mesma instrução e composta pela mesma cor da ficha corrente, então a ficha recém chegada era encaminhada para a Unidade de Sobrecarga, para aguardar pela disponibilidade de seu endereço. De forma equivalente às instruções de passagem

Figura 11 – Diagrama ilustrativo da Unidade de Pareamento na MDFM.

Fonte: Adaptada de GURD¹⁴

direta, a necessidade de pareamento também era indicada explicitamente no código do programa. No Algoritmo 2, ela pode ser vista nas funções de pareamento indicadas como **EW**, derivado da expressão em inglês “*Extract and Wait*”.

$$\begin{aligned}
 &< \text{dado esq}(37\text{-bits}), \text{ dado dir}(37\text{-bits}), \text{ rótulo}(36\text{-bits}), \text{ destino}(22\text{-bits}), \\
 &\hspace{20em} \text{marcador}(1\text{-bit}) > \quad (2.3)
 \end{aligned}$$

Em todo caso, na ocorrência de pareamento, ambas as fichas eram unidas dentro do pacote que pode ser visto na Equação 2.3. Perceba que em comparação com a definição de uma ficha, apenas são adicionados ambos os dados de composição das fichas pareadas para encaminhamento à Unidade de Instruções. Afinal, todas as outras informações presentes nesta composição são equivalentes, para fichas destinadas para a mesma instrução, durante a execução de um programa. É importante destacar que fichas de passagem direta também sofriam esta modificação e eram encaminhadas com o dado à direita do pacote com um valor nulo.

2.4.3 Unidade de Sobrecarga

Esta unidade tratava de uma ação extraordinária à operação convencional da máquina. Principalmente por este motivo, ela foi implementada de forma menos rigorosa, com relação ao seu desempenho. Esta unidade era composta essencialmente por uma lista encadeada, que devia ser analisada diante da ocorrência de qualquer ação de pareamento verificada na unidade principal. Afinal, não havia nenhuma premissa que pudesse permitir desconsiderar a existência de fichas que fossem destinadas para o endereço que se tornou vago. Desta forma, perceba tratar-se de uma operação bastante lenta, sob o ponto de vista do desempenho desta solução. A Unidade de Sobrecarga contava com um simples processador de microcódigo, para a resolução dos endereços ocupados em seus recursos, com os dados e ponteiros armazenados nesta estrutura de dados. Com isso, diante da descoberta de uma ficha destinada para um endereço que se tornou vago, ela era encaminhada em retorno à Unidade de Pareamento, para seguir o processo de operação natural desta unidade. Perceba que existia ainda a possibilidade de ambas as fichas destinadas para a mesma instância de uma instrução estarem armazenadas juntas na Unidade de Sobrecarga, então ela devia ser consultada novamente, diante da transmissão de uma ficha para o armazenamento na memória da unidade principal. Por estes motivos, a Unidade de Pareamento possuía um desempenho bastante lento, o que acabava transformando-a em um grande gargalo para o processo de execução de programas dentro desta implementação.

2.4.4 Unidade de Instruções

Esta unidade possuía uma composição simples, como pode ser visto no diagrama da Figura 12. Sua principal função tratava de manter armazenado em uma memória RAM, todo o programa (grafo do modelo a fluxo de dados) que estava em execução na máquina. Diante da chegada de novas fichas em seu *buffer* de entrada principal, o endereço da instrução - informação nativa na composição de uma ficha, de acordo com a Equação 2.2 - era consultado, para o resgate das informações do nó do grafo. Os *18-bits* de endereço, presente nas fichas eram distribuídos de acordo com a Equação 2.4. De acordo com a definição da **MDFM**, os programas eram distribuídos em segmentos, que em composição com o deslocamento, formavam o endereço para o armazenamento dentro da memória RAM deste unidade, que tinha a capacidade para o armazenamento total de até $64K$ instruções.

$$< \text{segmento}(6\text{-bits}), \text{deslocamento}(12\text{-bits}) > \quad (2.4)$$

As instruções eram formatadas de acordo com uma das representações dispostas na Equação 2.5. Alguns exemplos de programas com estas características podem ser vistos no Anexo A.2. Havia portanto estas três possibilidades de formatos para a composição

Figura 12 – Diagrama ilustrativo da Unidade de Instruções na **MDFM**.

Fonte: Adaptada de GURD¹⁴

de instruções dentro do modelo de implementação da **MDFM**. O código da operação era comum dentre todas as possibilidades, ele se refere à instrução específica que uma ficha deve executar dentro das Unidades Funcionais, o resgate desta informação nesta unidade se assemelha de muitas formas ao processo de busca e decodificação, comuns ao processo de funcionamento de um *pipeline*, em arquiteturas convencionais. Os campos de destino referem-se aos endereços das instruções subsequentes à execução da instrução corrente por uma Unidade Funcional. Note que pode haver até dois destinos específicos, conforme o formato de operação da **MDFM**. Ainda existia a possibilidade de se definir um dado literal, de acordo com os formatos possíveis de serem explorados. Esta informação refere-se a dados adicionados de forma imediata, no código do programa, indicados como os atributos dos nós do grafo, de forma equivalente aos círculos dispostos no exemplo da Figura 8.

$$\begin{aligned}
 &< opcode(10-bits), destino esq(22-bits) > \\
 &< opcode(10-bits), destino esq(22-bits), destino dir(22-bits) > \quad (2.5) \\
 &< opcode(10-bits), destino esq(22-bits), dado literal(37-bits) >
 \end{aligned}$$

Por fim, ao serem resgatadas as informações presentes no endereço definido na ficha corrente, eram formados os pacotes de saída desta unidade, que adicionavam estas informações em sua composição, como definido na Equação 2.6. Note que de acordo com esta definição, todas as informações necessárias para a execução da instrução já estão

disponíveis, ou seja, ambos os dados destinados para a realização da operação, em conjunto com todas as informações dinâmicas dos dados que serão computados, além dos endereços de destino das instruções subsequentes, dos novos dados que serão gerados a partir desta execução.

$$\begin{aligned} < \text{dado esq}(37\text{-bits}), \text{dado dir}(37\text{-bits}), \text{opcode}(10\text{-bits}), \text{rótulo}(36\text{-bits}), \\ &\quad \text{destino esq}(22\text{-bits}), \text{destino dir}(22\text{-bits}), \text{marcador}(1\text{-bit}) > \quad (2.6) \end{aligned}$$

É importante notar que apesar de ser composta por um processo relativamente simples, a unidade de instruções envolvia a leitura em uma memória RAM, e, por este motivo, apresentava uma latência alta natural e sistemática, que impedia um melhor desempenho.

2.4.5 Unidade de Processamento

Com todas as informações necessárias, os pacotes executáveis eram encaminhados para a Unidade de Processamento, que era responsável pela execução da instrução. Uma pequena quantidade de instruções, mais simples e de controle, eram executadas imediatamente no escopo da própria Unidade de Processamento, na sub unidade conhecida como Pré-Processador, com isso, não consumindo recursos das Unidades Funcionais. É importante destacar, que os principais benefícios desta arquitetura podem ser obtidos a partir da exploração massiva destas Unidades Funcionais. Afinal, quanto maior a ocupação destas unidades, mais poder computacional é possível de ser explorado por esta solução. Após a descoberta dos pacotes que não podiam ser executados no escopo do pré-processador, eles eram encaminhados para a primeira Unidade Funcional livre para execução, a partir do barramento de distribuição disposto na Figura 13.

Todas as Unidades Funcionais possuíam exatamente as mesmas propriedades e eram implementadas no formato de microcódigos, com a descoberta das ações necessárias para as operações a serem realizadas para o tratamento dos dados correntes, definidos a partir do código de operação obtido na Unidade de Instruções. As instruções eram executadas de forma totalmente independente umas das outras, e em paralelo, dentro das várias Unidades Funcionais que não possuíam nenhum tipo de sincronismo entre elas. Existiam o total de 20 Unidades Funcionais na **MDFM**.¹⁴

Após a execução das instruções, novas fichas eram geradas. Estas fichas eram encaminhadas para a Unidade de Chaveamento, que verificava se se tratavam de fichas para a saída; neste caso, a ficha era encaminhada como resultado do programa, e caso contrário, ela era direcionada à Fila de Fichas dando sequencia à execução do programa.

Figura 13 – Diagrama ilustrativo da Unidade de Processamento na MDFM.

Fonte: Adaptada de GURD¹⁴

2.5 Outras implementações

Os benefícios do modelo a fluxo de dados vem sendo demonstrados por muitos anos, de forma que diversas implementações de arquiteturas de computadores que exploram este modelo foram propostas desde sua introdução ainda em meados dos anos 70.³⁵ Apesar de ter sido explorada a partir de visões bastante distintas, totalmente originais e independentes entre elas, a estrutura global da grande maioria das estruturas clássicas implementadas possuem muitas similaridades.

Atualmente, além da proposta sugerida pelo presente trabalho, existem diversos estudos contemporâneos sendo realizados para exploração deste modelo de computação, sendo possível destacar alguns exemplos notáveis, como descrito brevemente a seguir.

2.5.1 TRIPS

Esta máquina foi desenvolvida na Universidade do Texas, em parceria com diversas empresas especializadas do setor, como a IBM, Intel e Sun Microsystems, em um projeto em atividade desde o ano de 2005. A *TRIPS* foi a primeira máquina construída com base no conjunto de instruções conhecido como *EDGE* e possui a finalidade principal de lidar com capacidades de paralelismo intrínseco à arquitetura, em processadores baseados em múltiplos núcleos, em consonância com a própria definição do paradigma da computação dirigida por dados. Este novo conjunto de instruções utiliza um modelo de execução por blocos atômicos, onde cada bloco é constituído por instruções e chamadas em fluxo de dados, com a microarquitetura explorada dentro destes blocos seguindo as mesmas características do x86 convencional.^{36,37} O objetivo principal deste projeto é otimizar a capacidade para exploração de paralelismo para alto desempenho com foco nos desafios promovidos pela redução de componentes advindos de eventuais novidades em tecnologias emergentes.

2.5.2 Maxeler

A *Maxeler Technologies* é uma empresa que possui foco na exploração massiva de paralelismo a partir de uma estrutura baseada no modelo a fluxo de dados, para lidar com as mais variadas aplicações que demandam alta capacidade de desempenho. Sua estrutura global conta com um sistema de processamento baseado em múltiplos núcleos x86 convencionais, conectados a um sistema de aceleração composto por **FPGAs** interligadas, que implementam uma microarquitetura baseada no modelo a fluxo de dados.^{38,39} Note tratar-se de um sistema de aceleração apenas, figurando portanto, como um sistema de propósito específico capaz de explorar o modelo a fluxo de dados, em um sistema equivalente à aceleradores de hardware convencionais.

2.5.3 Teraflux

O *Teraflux* é um projeto iniciado no ano de 2010, em um consórcio que reuniu importantes centros de pesquisa da União Européia. Sua principal finalidade era garantir a exploração de paralelismo em processadores compostos por até dezenas de milhares de núcleos de processamento convencionais, a fim de garantir retro compatibilidade com sistemas legados. A estratégia prevista por este projeto utiliza o modelo a fluxo de dados em nível de compilação apenas, para a exploração otimizada de paralelismo em estruturas convencionais.^{40,41}

2.5.4 WaveScalar

O *WaveScalar* é a implementação de um modelo a fluxo de dados que teve como motivação principal a construção de um núcleo de processamento superescalar descentralizado.⁴² Na evolução do desenvolvimento, a existência de um único contador de programas em uma estrutura superescalar levou à necessidade de um modelo dirigido por dados no processo de busca de instruções, durante a execução de programas. Sua principal vantagem é a capacidade para compilação de códigos escritos em linguagens tradicionais para execução dentro deste modelo de implementação, além do sistema de cache de instruções distribuída, que garante uma capacidade de localidade muito interessante para a exploração do modelo a fluxo de dados.

3 PRIMEIRA VERSÃO

Nessa etapa do desenvolvimento, o principal objetivo era a verificação da viabilidade de implementação da **Máquina a Fluxo de Dados de Manchester** a partir do uso de dispositivos modernos, que para fins práticos, foram utilizadas **FPGAs** para este propósito, utilizando uma linguagem de descrição de hardware de alto nível, o **Bluespec SystemVerilog**. Essa linguagem é reconhecida por sua capacidade de simulação em diversos ambientes, e pela garantia de que todo código compilado por ela pode ser sintetizado imediatamente, sem a necessidade de qualquer modificação explícita no código *Verilog* gerado neste processo.

Durante o desenvolvimento deste protótipo inicial, foi possível verificar a inviabilidade dessa implementação. Como será descrito em detalhes a seguir, as **Unidades Funcionais** foram implementadas sob o mesmo domínio de relógio com relação ao restante da estrutura da máquina, diferentemente da **MDFM**, onde estas unidades eram muito mais lentas. Com isso, foi possível verificar que tal estrutura acaba se tornando um gargalo na arquitetura proposta, não oferecendo qualquer ganho em desempenho com a adição de novos recursos de processamento.

Diante desta constatação, foi proposta uma nova solução que apesar de ainda ser inspirada pela **MDFM**, apresenta uma construção totalmente nova. Esta proposta baseia-se em um conceito de expansão na capacidade de comunicação entre as unidades que compõem a máquina, a partir de uma propriedade que ficou conhecida como dutos de largura de banda. Essa nova implementação foi capaz de apresentar excelentes resultados quando comparados com os valores de referência, que eram possíveis de ser obtidos na estrutura original.^{14,43-45} Apesar de apresentar bons índices sobre ganho em desempenho, a nova estrutura apresentou algumas limitações, devido à quantidade de recursos necessários para implementá-la.

3.1 π Flow

Como a proposta inicial era a recriação da **MDFM** a partir de **FPGAs**, a estrutura da máquina original foi completamente mantida, com pequenas exceções. De acordo com a descrição da Seção 2.4, esta estrutura consiste na construção de cinco módulos fundamentais, conforme a descrição detalhada sobre a implementação de cada uma delas a seguir. A Figura 14 apresenta a visão geral desta versão da máquina. Neste ponto, é importante destacar a semelhança desta estrutura em comparação com a **MDFM**, como mostrado na Figura 9.

Figura 14 – Diagrama de representação da π Flow, primeira versão do presente projeto.

Fonte: Elaborada pelo autor.

Neste projeto, todas as unidades que compreendem a estrutura da máquina foram conectadas entre si a partir da implementação nativa do **BSV** de filas de tamanho definido. Essa estrutura apresenta uma latência de um ciclo do relógio para efetivar a comunicação em uma fila vazia.⁴⁶ Perceba portanto, que se houver mais elementos armazenados, essa estrutura é capaz de fornecer recursos em todo ciclo do relógio. O tamanho de armazenamento dessas estruturas foi definido a partir de um valor constante, determinado em tempo de compilação no **BSV**.

Uma das principais estratégias buscadas durante esta fase do desenvolvimento, foi garantir a parametrização dos principais recursos da máquina. Dessa forma, a quantidade de **Unidades Funcionais** e o tamanho das filas de comunicação entre as unidades de composição da máquina, são exemplos de recursos implementados a partir destes valores constantes no código de descrição do **BSV**. Este aspecto foi proposto inicialmente, a fim de possibilitar a verificação do desempenho e funcionamento geral da máquina, a partir do uso de diversas capacidades, comportamentos, ou estados que poderiam não ter sido verificados na máquina original.

3.1.1 Fila de Fichas

Bem como na **MDFM**, aqui a **Fila de Fichas** funciona como um *buffer* para a máquina. Esta unidade acaba por controlar e *sequencializar* o fornecimento de fichas para a **Unidade de Pareamento**, característica importante para a arquitetura proposta, especialmente para programas que oferecem uma quantidade muito grande de paralelismo. Afinal, as **Unidades Funcionais** são capazes de fornecer uma quantidade muito grande

de fichas durante a execução de programas.

Figura 15 – Diagrama de representação da **Fila de Fichas** na primeira versão do projeto.

Fonte: Elaborada pelo autor.

Segundo a estrutura original, uma ficha é organizada com todas as informações necessárias para o resgate - dentro das unidades que compõem a máquina - dos valores necessários para a execução de instruções.¹⁴ Nesta versão do desenvolvimento, alguns atributos das fichas características da **MDFM**, Seção 2.1, foram omitidos, devido principalmente à composição dos programas utilizados para a validação desta estrutura, que não necessitavam de todos os recursos existentes na máquina de inspiração para serem executados. Dessa forma, na π **Flow**, uma ficha era composta pelos seguintes atributos e larguras.

$$\langle \text{dado}(37\text{-bits}), \text{rótulo}(12\text{-bits}), \text{destino}(20\text{-bits}), \text{marcador}(1\text{-bit}) \rangle \quad (3.1)$$

A **Fila de Fichas** tem a capacidade para o armazenamento de até 4096 fichas, com um total de 70-bits em cada endereço. Em sua composição interna existe apenas uma grande fila de tamanho definido destinada para a realização deste armazenamento interno, que tem a finalidade de proporcionar o envio das primeiras fichas recebidas na unidade, adiante. Sendo assim, essa é uma estrutura suficiente para a implementação desta unidade, seguindo os preceitos da arquitetura.

Ainda nesta primeira versão, foi proposta uma pequena modificação com relação a estrutura original. A fim de favorecer um princípio de funcionamento natural da arquitetura da máquina, foi implementada uma comunicação dupla entre a **Unidade de Processamento** e a **Fila de Fichas**. Afinal, de acordo com sua definição, cada **Unidade Funcional** é capaz de fornecer até duas novas fichas subsequentes diante da execução de instruções.⁴⁴ Essa modificação ofereceu apenas um pequeno ganho em desempenho diante da execução de programas, mas serviu de grande inspiração para a segunda versão do projeto, como será descrito na Seção 3.2.

A estrutura dessa unidade conta apenas com duas filas de tamanho definido de entrada da unidade (em comunicação com a **Unidade de Processamento**), uma grande fila de armazenamento interno e uma fila de comunicação com a **Unidade de Pareamento**. Além disso, existe uma interface ativa que permite a comunicação desta unidade com o equivalente à unidade de **Chaveamento** da **MDFM**, que neste instante do desenvolvimento, era representada pelo *Top Level* da aplicação*. Esta interface é utilizada para a entrada das fichas de início de programas.

Foi proposta ainda uma solução capaz de favorecer o desempenho da máquina no início e final de programas, quando ainda não existe uma quantidade considerável de paralelismo. As filas de entrada da unidade foram conectadas diretamente às de saída, e seguirá com este comportamento enquanto este elemento de troca com a **Unidade de Pareamento** ainda não estiver completamente cheio. Com isso, as fichas não precisam aguardar pela latência de armazenamento e leitura na fila de armazenamento persistente da unidade. Tratando-se de uma outra estrutura que forneceu um ganho em desempenho pouco significativo, porém que também serviu de inspiração para uma das implementações seguintes, que tinha o intuito de oferecer ganho em desempenho entre as unidades da estrutura da máquina, como descrito na Seção 4.2.

3.1.2 Unidade de Pareamento

A **Unidade de Pareamento** tem a finalidade de unir fichas que sejam destinadas à mesma instrução, durante a execução de programas. Imagine como exemplo, uma operação de soma entre números inteiros, a serem definidos em tempo de execução. Para realizar esta operação é necessária a existência de duas fichas distintas - que deverão ser geradas por instruções anteriores - que naturalmente serão destinadas para a mesma instrução e se encontram na mesma instância, de acordo com as propriedades dinâmicas do modelo a fluxo de dados. Estas fichas são identificadas e unidas em um único pacote no processo de funcionamento natural da **Unidade de Pareamento**.

Na **MDFM**, a fim de identificar a instância em que uma ficha está inserida, eram

* Unidade primária do projeto descrito em **BSV**, onde é realizada a instância de todos os módulos que compõem a solução.

Figura 16 – Diagrama de representação da **Unidade de Pareamento** na primeira versão do projeto.

Fonte: Elaborada pelo autor.

utilizados os rótulos dinâmicos, definidos na composição de uma ficha, como destacado pela Equação 3.1.¹⁴ Devido ao algoritmo de implementação dos programas utilizados para validação desta estrutura inicial - que apresentam grande capacidade de paralelismo, derivado principalmente de seu formato recursivo - apenas o rótulo conhecido como **Nome de Ativação** foi implementado. No conjunto de instruções da arquitetura, este rótulo tem a finalidade de realizar chamadas a funções,¹⁶ tratando-se de uma propriedade largamente explorada em programas recursivos, e portanto, suficiente para identificar a instância de recursividade em que se encontra determinada ficha, no contexto do programa em execução, de acordo com o princípio de funcionamento da Unidade de Pareamento.

Como pode ser visto na Figura 16, esta unidade é composta por uma memória

RAM, implementada a partir do módulo **BRAM**, nativo do **BSV**, que tem a finalidade de explorar os blocos de memórias distribuídos entre os componentes fundamentais das **FPGAs**, como mostrado na Figura 2. Este módulo possui interfaces ativas específicas para leitura e escrita, que levam o período fixo de um ciclo do relógio para serem efetivadas. Note tratar-se de uma estrutura que impõe um gargalo natural para o desempenho da arquitetura. Conforme descrito no capítulo anterior, na **MDFM**, o endereçamento de fichas era realizado a partir da aplicação de uma função *hash* entre o endereço de destino e o rótulo da ficha corrente. Perceba que estes valores, em conjunto, são únicos para quaisquer par de fichas destinadas a mesma instrução. Para validação da estrutura, na π **Flow** o endereçamento foi realizado a partir da simples concatenação entre o endereço de destino da ficha corrente e seu **Nome de Ativação**.

A partir desta “função *hash*” não é verificada a ocorrência de conflitos para endereçamento, fato que permitiu omitir a **Unidade de Sobrecarga** neste projeto. Entretanto, foi verificado ao longo do desenvolvimento que esta estratégia promoveu a utilização de uma grande quantidade de memória para a síntese deste módulo. Afinal, seu endereçamento depende da soma entre as larguras do Endereço e do **Nome de Ativação** das fichas. Apesar de tratar-se de um fator limitante, essa propriedade ainda permitiu a análise do desempenho da estrutura implementada, para a execução de programas menores, que apresentavam menos paralelismo. Porém, a simulação da solução permitiu a exploração de valores maiores de armazenamento, dependendo apenas da quantidade de memória disponível no computador que está gerando o código de simulação e que irá realizar a execução deste simulador. Essa capacidade permitiu a omissão da **Unidade de Sobrecarga** sem prejuízos à qualidade da análise.

As fichas de passagem direta (como conhecidas na estrutura da **MDFM**) são destinadas a instruções de entrada única, ou seja, fichas que não dependem de pareamento nesta unidade. Esta característica foi implementada a partir da verificação da cadeia de *bits* identificada como “destino”, indicado no formato de uma ficha, como definido na Equação 3.1. Esta informação é determinada no momento da criação de novas fichas, identificada pelo programa que está sendo executado, em seu formato em linguagem montadora. Esta cadeia de bits é formada pelo padrão destacado na Equação 3.2. Dessa forma, as fichas que possuem o valor “BY” na informação de “pareamento”, na composição de seu formato (como pode ser visto em detalhes no Anexo A.2), são encaminhadas imediatamente ao construtor de pacotes de saída desta unidade. Perceba portanto, que as fichas que apresentam esta propriedade não carregam consigo a latência existente para leitura e escrita da memória RAM que compõe o princípio de funcionamento convencional desta unidade, sendo assim responsável por grande ganho em desempenho durante a execução de programas.

$$< \text{endereço}(16\text{-bits}), \text{lado}(1\text{-bit}), \text{pareamento}(3\text{-bits}) > \quad (3.2)$$

Apesar de existirem outras funções de pareamento especificadas no conjunto de instruções da **MDFM**,¹⁶ para os programas utilizados neste projeto foram identificadas apenas fichas de dois tipos distintos. As fichas de passagem direta, que possuem o comportamento descrito anteriormente, não utilizam os recursos da memória interna da **Unidade de Pareamento**. A outra função de pareamento verificada era conhecida como “Extrair e Esperar” no conjunto de instruções da **MDFM**, identificada como “EW”, proveniente de sua descrição em inglês *Extract and Wait*. As outras funções de pareamento definidas no manual do conjunto de instruções da máquina¹⁶ de inspiração não estão presentes nesta implementação, devido a ausência de quaisquer outras características notáveis nos programas utilizados para validação e, portanto foram omitidas desta documentação.

O outro tipo de ficha, portanto, possui o valor “EW” na cadeia de *bits* identificada como pareamento, destacada na Equação 3.2. A chegada destas fichas no *buffer* de acumulação de entrada desta unidade disparam a requisição de leitura na memória RAM da unidade para o endereço formado pela concatenação entre o **Nome de Ativação** e o Endereço de Destino da ficha recém chegada. Caso exista uma ficha armazenada neste endereço, ambas são encaminhadas ao construtor de pacotes de saída da unidade. Caso contrário, é encaminhada uma requisição de escrita à esta memória RAM, para o armazenamento da ficha recém chegada. Perceba que o armazenamento de fichas de pareamento em um *buffer* intermediário, permite a passagem de possíveis fichas subsequentes que sejam de passagem direta, independente da latência de leitura e escrita da memória RAM.

O construtor de pacotes tem a finalidade de gerar os pacotes de saída dessa unidade. Trata-se de uma cadeia de *bits* estruturados, que possui o formato definido pela Equação 3.3.

$$< \text{dado esq}(37\text{-bits}), \text{dado dir}(37\text{-bits}), \text{rótulo}(12\text{-bits}), \text{destino}(20\text{-bits}), \\ \text{marcador}(1\text{-bit}) > \quad (3.3)$$

É importante reforçar neste momento, que devido às restrições de integridade do **BSV**, os circuitos e registradores que recebem estas informações devem possuir um tipo de dado exatamente igual a esta definição. Por este motivo, na ocorrência de instruções de entrada única, como as fichas de passagem direta, a informação com o dado da direita é apenas completado com valores nulos. Essa característica acaba promovendo uma subutilização dos recursos da máquina. Entretanto, ela promove consideravelmente a coerência no caminho por onde essas fichas devem ser encaminhadas. Apesar desta construção ter sido herdada da **MDFM**, o **BSV** ainda reforça essa necessidade, de que

todas as interfaces de comunicações sejam verificadas exatamente com o mesmo tipo de dado, a fim de garantir a integridade de toda transmissão de informações entre as unidades que compõem a estrutura da máquina.

3.1.3 Unidade de Instruções

Essa unidade armazena todo o código em linguagem montadora (que caracteriza o grafo do modelo a fluxo de dados), do programa que está sendo executado pela máquina. Para isso, como pode ser visto no diagrama da Figura 17, ela é composta por uma memória RAM, implementada a partir do mesmo módulo **BRAM** da unidade anterior. Perceba portanto, que existe novamente a mesma latência fixa de um ciclo do relógio durante a passagem de pacotes por essa unidade, com a desvantagem de não contar com nenhuma propriedade capaz de mitigar essa latência, como as fichas de passagem direta, descritas na Seção 3.1.2. O endereço da instrução que um pacote deve executar é definido na cadeia de *bits* identificada como “destino”, destacado nas Equações 3.1, 3.3.

Figura 17 – Diagrama de representação da **Unidade de Instruções** na primeira versão do projeto.

Fonte: Elaborada pelo autor.

Dessa forma, a partir do endereço da instrução contido no pacote recém chegado à unidade, é realizada a leitura na memória RAM presente na sua composição. Cada endereço

dessa memória armazena as informações necessárias para a execução de instruções, sendo elas:

- **Código de Operação:** código da instrução que deve ser executada, equivalente ao “*Opcode*” do modelo de *Von Neumann*, esse código será responsável por identificar o circuito que será ativado na **Unidade Funcional**, para a execução da instrução, com o conjunto de dados contidos no pacote;
- **Destino da Esquerda / Direita:** endereço das instruções de destino das fichas que serão criadas diante da execução da instrução presente no pacote que gerou a execução de leitura na memória desta unidade.
- **Dado Literal:** a arquitetura proposta pela **MDFM** prevê a possibilidade de entrada de dados definidos explicitamente no código em linguagem montadora, equivalente ao endereçamento imediato do modelo de *Von Neumann*. Essas informações tomam o lugar das informações de destino, descrito anteriormente, nestas instruções específicas, como pode ser visto no manual do conjunto de instruções.¹⁶

O processo de leitura nesta memória dispara o construtor do pacote de saída desta unidade, formados de acordo com a estrutura determinada na Equação 3.4. Após a geração deste pacote, ele é encaminhado ao *buffer* de saída da **Unidade de Instruções**. Perceba que os pacotes gerados de acordo com esse procedimento possuem todas as informações necessárias para a execução de instruções, e portanto, são chamados de pacotes executáveis. Nessa versão do protótipo, o código do programa que deve ser executado é indicado explicitamente no escopo da memória RAM. Ou seja, o programa é definido como o conteúdo padrão da memória, em tempo de compilação do **BSV**.

$$\begin{aligned} < \text{dado esq}(37\text{-bits}), \text{dado dir}(37\text{-bits}), \text{opcode}(10\text{-bits}), \text{rótulo}(12\text{-bits}), \\ &\quad \text{destino esq}(20\text{-bits}), \text{destino dir}(21\text{-bits}), \text{marcador}(1\text{-bit}) > \quad (3.4) \end{aligned}$$

3.1.4 Unidade de Processamento

A **Unidade de Processamento** tem a finalidade de distribuir os pacotes executáveis às **Unidades Funcionais** que compõe a estrutura da máquina. A quantidade de **Unidades Funcionais** presentes na estrutura da π **Flow** é definida em tempo de compilação do **BSV**, a partir de um valor constante definido no escopo do projeto. Essa característica de parametrização foi possível a partir da implementação de um complexo circuito de distribuição, que provocou um aumento considerável na quantidade de recursos necessários para síntese da solução em **FPGAs**.

Figura 18 – Diagrama de representação da **Unidade de Processamento** na primeira versão do projeto.

Fonte: Elaborada pelo autor.

Como pode ser visto na Figura 18, a chegada de novos pacotes no *buffer* de entrada dessa unidade dispara o sistema de pré-processamento, que nessa versão da implementação tinha a finalidade de lidar apenas com a manipulação do **Nome de Ativação**. Como este identificador deve ser único durante a execução de sua rotina, ele é centralizado em um registrador exclusivo para essa finalidade. A geração de novos Nomes de Ativação é realizada a partir do simples incremento do valor contido neste registrador, diante da ocorrência de instruções destinadas para a manipulação desta estrutura da máquina.

Após a realização do pré-processamento, o pacote é encaminhado ao circuito destinado à verificação da **Unidade Funcional** que irá recebê-lo para execução. Um

registrador exclusivo armazena a posição da unidade que recebeu o último pacote chegado à **Unidade de Processamento**. Perceba portanto, que as **Unidades Funcionais** são identificadas por um valor inteiro, definido em tempo de compilação da solução. Todas as unidades possuem uma interface que retorna seu estado atual, que pode ser identificado como: Ociosa ou Em Execução.

O circuito de distribuição analisa a interface de estado de todas as unidades, a partir da última que foi alimentada no último ciclo do relógio, em ordem crescente, segundo um algoritmo *round-robin*. Essa característica garantiu a melhor distribuição de aproveitamento das **Unidades Funcionais**. Ela foi proposta a fim de evitar qualquer prioridade dentre as unidades disponíveis na solução. Foi observado, durante alguns testes preliminares, que a ausência deste circuito promovia a utilização máxima apenas das três primeiras unidades presentes na máquina. Este aspecto ressaltou as conclusões que serão destacadas no início da próxima Seção 3.2.

A execução de instruções pode promover a geração de novas fichas subsequentes, destinadas aos endereços da esquerda e direita, destacadas na Equação 3.4.¹⁶ Estas fichas são encaminhadas em retorno à **Unidade de Processamento**, que os armazena em *buffers* intermediários, para serem então encaminhados aos *buffers* de saída da unidade. Perceba que existe a probabilidade de todas as **Unidades Funcionais** gerarem fichas de saída dentro do mesmo ciclo do relógio e por esse motivo foi sugerido o uso de um *buffer* intermediário para armazenar todas as fichas de saída, sem a necessidade de bloquear uma **Unidade Funcional**.

Uma quantidade considerável de instruções nesta arquitetura fornecem duas fichas em sua saída.¹⁶ Essa característica motivou a implementação de uma solução distinta da MDFM. Note pela Figura 18 que existem dois *buffers* de saída na estrutura da **Unidade de Processamento**. Essa estrutura foi proposta para se aproveitar deste paralelismo intrínseco da arquitetura. Entretanto, conforme descrito anteriormente, as fichas são serializadas novamente na **Fila de Fichas**.

3.1.5 Unidade Funcional

As **Unidades Funcionais** são responsáveis pela execução efetiva das instruções. Estas unidades possuem todos os seus registradores de armazenamento reiniciados após a execução de instruções. Ao receber um novo pacote executável, este é armazenado em um registrador que irá mantê-lo até o final da execução. Neste momento, o registrador de estado da **Unidade Funcional** é marcado com o estado: Em Execução. As operações dentro destas unidades seguem a estrutura de um *pipeline*, porém, elas foram implementadas para executarem pacotes isoladamente. Essa característica foi proposta em busca de aumentar a capacidade de exploração de mais **Unidades Funcionais** durante a execução de programas mais complexos.

Figura 19 – Diagrama de representação das **Unidades Funcionais** na primeira versão do projeto.

Fonte: Elaborada pelo autor.

Após o armazenamento do pacote neste registrador de entrada, é analisado seu código de operação. Existe uma regra destinada para cada instrução da arquitetura, identificada por uma condição explícita em cada uma destas regras. Os estados promovidos pela execução de uma instrução é armazenada em registradores exclusivos, para todas as características descritas sobre a definição da Equação 3.1, de acordo com as propriedades da operação a ser executada. Ou seja, existe um registrador destinado para o armazenamento do dado de saída esquerdo, outro para o dado direito e outro para o armazenamento do rótulo das fichas de saída e, assim por diante.

Após a execução da instrução, a unidade entra no estágio onde é avaliada a saída lógica considerada pela instrução. As instruções da **MDFM** podem possuir até duas saídas exclusivas, de acordo com a operação executada por ela. Durante a execução de programas, existem instruções que utilizam apenas uma, ou ambas as saídas lógicas geradas pela

execução de determinada instrução. Dessa forma, neste estágio de operação da **Unidade Funcional**, as informações que não serão utilizadas pelo programa são descartadas e seus registradores são colocados em um estado inválido.

Por fim, a partir dos registradores de estado que armazenam os valores gerados pela execução dos pacotes, são geradas novas fichas, de acordo com a descrição da Equação 3.1. Ambas as fichas de saída são geradas simultaneamente, dentro do mesmo ciclo do relógio e encaminhadas aos *buffers* de saída da unidade. Ao final da execução de uma instrução, todos os registradores de estado são reiniciados. Durante este processo de reinício, o registrador de estado da **Unidade Funcional** é marcado com o estado: Ocioso.

Conforme destacado em todas as estruturas de fichas e pacotes, existe *1-bit* identificado como “marcador”. Nesta versão do projeto, essa característica tinha a finalidade apenas de identificar quando uma ficha seria destinada para a saída de informações da máquina. Essa propriedade é verificada no *TopLevel* da aplicação e pode ser destinada à qualquer porta de saída da **FPGA** com a finalidade de apresentar o resultado da execução de um programa. Dessa forma, este *bit* pode representar dois estados em uma ficha, ou ela será uma ficha de instruções, que seguirá o fluxo comportamental padrão, ou ela será uma ficha de saída. Essa propriedade é determinada nas instruções de saída da máquina, segundo o manual do conjunto de instruções da **MDFM**.¹⁶

3.2 π FlowBW

3.2.1 Análise dos resultados

Durante a implementação da primeira versão do projeto, Seção 3.1 foi possível detectar uma série de limitações impostas pela estrutura proposta para a implementação da arquitetura. O processo de leitura e escrita nas memórias RAM das unidades de **Pareamento** e de **Instruções**, impõem uma grande latência fixa para a operação da máquina durante a execução de programas. Além disso, a própria passagem das fichas e pacotes por entre todas as unidades que compõem esta estrutura acabaram se tornando um grave gargalo, que acaba impedindo o ganho em desempenho em nível de paralelismo.

É simples a análise da quantidade de ciclos do relógio necessários entre a geração de novas fichas subsequentes à execução das fichas correntes e sua execução efetiva, em retorno à **Unidade de Processamento**. Uma vez gerada por uma **Unidade Funcional**, uma ficha tomará dois ciclos do relógio para ser encaminhada em retorno à **Unidade de Processamento**, no *buffer* intermediário entre as **Unidades Funcionais** e o *buffer* de saída dessa unidade. Supondo o melhor caso possível, essa ficha passará imediatamente do *buffer* de entrada ao *buffer* de saída da **Fila de Fichas**, se aproveitando dessa característica implementada na π Flow. Em seguida, novamente supondo o melhor caso, essa ficha será

de passagem direta e tomará apenas um novo ciclo do relógio para passar do *buffer* de entrada à saída da **Unidade de Pareamento**.

Já na **Unidade de Instruções**, em todos os casos, será necessário o período fixo de dois ciclos do relógio entre a entrada do pacote, leitura da memória RAM dessa unidade, geração do pacote executável e envio ao seu *buffer* de saída. Na **Unidade de Processamento**, são tomados três ciclos do relógio para os pacotes passarem pelo estágio de **pré-processamento** e circuito de distribuição, para só então serem encaminhados à uma nova **Unidade Funcional** que irá executar a instrução efetivamente. As fila de tamanho definido do **BSV** realizam a escrita nos seus registradores de saída no final de um ciclo do relógio.

Dessa forma, note que existe um período fixo de 13 ciclos do relógio, entre a criação de uma nova ficha e a execução efetiva da instrução a que ela se destina, sendo este o melhor caso possível, com o menor consumo de ciclos entre as unidades. Note ainda que uma **Unidade Funcional** é capaz de executar uma instrução simples, por um período fixo de 5 ciclos do relógio, até entrar no estado de “Ociosa” novamente. Ou seja, o processo de geração e execução de fichas proposto pela arquitetura não é suficiente para se sustentar, dentro da estratégia implementada, onde as **Unidades Funcionais** operam na mesma frequência de relógio das outras unidades que compõem a estrutura da máquina.

Na **MDFM**, as **Unidades Funcionais** operavam a uma frequência muito mais baixa em comparação com o restante da sua estrutura. Além disso, todas as unidades operavam de forma totalmente assíncrona e independentes entre si, sincronizadas apenas pelos *buffers* de comunicação que as conectavam. Isso garantia que o gargalo observado neste projeto não fosse suficiente para impedir essa estrutura de explorar paralelismo. Dessa forma, vale destacar, que o projeto descrito em **FPGA** permite que todas as unidades possam ser sincronizadas, implementadas sob o mesmo domínio de relógio. Com isso, a operação realizada de forma totalmente sincronizada entre todas as unidades que compõem a estrutura da máquina, não permite o ganho em desempenho verificado na **MDFM**, diante da adição de novas **Unidades Funcionais**, impossibilitando níveis em *speedup* que sejam ao menos próximos do ideal[†], que fossem apreciáveis.

3.2.2 Escopo

Dessa forma, após a constatação dos principais gargalos existentes na arquitetura, segundo os critérios adotados na implementação, foram propostas algumas alterações que permitissem a exploração de paralelismo a partir desta estratégia. Com isso, o principal objetivo dessa etapa do projeto foi a busca pela viabilização para a implementação de

[†] Principal métrica utilizada para análise do ganho em desempenho com a adição de novas unidades de processamento. Essa métrica será descrita em detalhes na Seção 6.1.

um protótipo inspirado pela **MDFM** a partir do uso de **FPGAs**, que permitisse esta exploração massiva de paralelismo.

Uma das principais fontes de inspiração para essa evolução do projeto foi a concepção da dupla ligação entre a **Unidade de Processamento** e a **Fila de Fichas**, realizada ainda na primeira versão, descrita na Seção 3.1. Essa característica ofereceu apenas um pequeno ganho em desempenho para a execução de programas. Entretanto, foi possível verificar a capacidade do **BSV** para abstração da comunicação entre as unidades que compõem a visão geral da arquitetura.

A partir dessa análise, foi introduzido o conceito que ficou conhecido como *dutos de largura de banda*[‡], que será definido pela expressão BW deste ponto em diante. Este conceito se mostrou suficiente para aumentar consideravelmente o *throughput* de fichas / pacotes que transitam entre as unidades da arquitetura. De forma equivalente a outros valores destacados nas seções anteriores, a quantidade de dutos de largura de banda BW , presentes na solução também foi implementada de forma parametrizável, caracterizada por um valor constante, definida em tempo de compilação do **BSV**. Em uma análise mais restrita, este valor constante que caracteriza a quantidade de BW presente na solução permitiu definir com relativa precisão o *throughput* geral da máquina.

3.2.3 Implementação

Todas as filas de comunicação que conectam as unidades de composição da arquitetura foram replicadas para a quantidade de BW . Além disso, todos os recursos fundamentais de cada unidade foram divididos por esse valor constante. Dessa forma, as estruturas que apresentavam gargalos no formato anterior, tiveram seus recursos divididos, permitindo que múltiplas operações pudessem passar a ser realizadas simultaneamente, de forma paralela. Cada sub estrutura consome os recursos da fila de entrada equivalente à sua definição no contexto dos dutos de largura de banda.

É importante reforçar neste ponto que uma ficha / pacote irá percorrer por toda a estrutura da máquina dentro de um duto fixado. A escolha do duto a partir do qual uma ficha irá transitar por entre as unidades é definida pela instrução que essa ficha deve executar. A memória de armazenamento do programa, na **Unidade de Instruções**, também foi dividida de acordo com as instruções armazenadas, pelo valor de BW . Por exemplo, a primeira instrução do programa ficará armazenada no primeiro duto, a segunda instrução do programa no segundo duto e, assim sucessivamente. Essa distribuição segue o formato *round-robin*, ou seja, a partir da instrução equivalente ao último duto de largura de banda, a seguinte ficará armazenada no primeiro duto, seguindo a mesma estratégia de distribuição na sequência.

[‡] Do inglês *Bandwidth Lines*, que deu nome à essa nova solução de implementação, a π FlowBW

Figura 20 – Diagrama de representação da segunda versão do projeto, que ficou conhecida como π FlowBW.

Fonte: Elaborada pelo autor.

Na **Fila de Fichas**, a fila de armazenamento interno teve sua capacidade dividida pelo valor constante que define a quantidade BW . Conforme descrito anteriormente, cada sub-fila de armazenamento consome os recursos de seu *buffer* de entrada equivalente, ou seja, a primeira fila de armazenamento consome os recursos do primeiro *buffer* de entrada da unidade, a segunda fila de armazenamento consome os recursos do segundo *buffer* de entrada e, assim sucessivamente. Esta característica de divisão dos recursos se mostrou bastante eficaz. Tome como exemplo um cenário no qual um duto completo entra em estado de bloqueio, devido a um eventual acúmulo de pacotes na **Unidade de Processamento**. Aqui não é possível ocorrer o bloqueio de toda a máquina, pois os outros dutos podem seguir em sua operação naturalmente. Ademais, na **Fila de Fichas**, cada duto seguiu com a ligação entre a fila de entrada e de saída de sua estrutura, bem como na primeira versão do projeto.

Na **Unidade de Pareamento**, tanto o arco das fichas de passagem direta quanto o de fichas destinadas ao pareamento tiveram seus recursos divididos de acordo com o valor constante de largura de banda. É importante reforçar que todos os dutos de largura de banda são independentes entre si. Dessa forma, percebe-se que existe a probabilidade de entrar nesta unidade a quantidade BW de novas fichas de passagem direta. Com isso, a unidade passa a ser capaz de fornecer essa quantidade de novas fichas à **Unidade de Instruções**, paralelamente. Entretanto, para fichas de pareamento, toda a latência para leitura e escrita na memória equivalente à determinado duto de largura de banda, continuou inalterado.

A **Unidade de Instruções** teve sua memória de armazenamento dividida conforme a descrição anterior. Ela apresenta uma característica semelhante à **Unidade de Pareamento**, pois, apesar de todos os dutos contarem com a latência de leitura, é possível a saída de diversos pacotes dentro de um mesmo ciclo do relógio. Isso se mostrou suficiente para lidar com a baixa taxa de aproveitamento das **Unidades Funcionais**. Como foi utilizado o endereço da instrução que uma ficha deve executar, como parâmetro para a definição do duto por onde ela irá percorrer a estrutura da máquina, nenhum tratamento especial teve de ser realizado na **Unidade de Instruções**. Os pacotes apenas promovem a leitura de sua instrução na memória de armazenamento e são encaminhados aos *buffers* de saída.

A **Unidade de Processamento** contou com um complexo circuito para a distribuição de pacotes entre as **Unidades Funcionais** livres para execução e os dutos de largura de banda. Isso foi necessário para manter essa distribuição livre de prioridades. A estrutura em formato *round-robin* foi mantida, entretanto, devido aos aspectos dinâmicos parametrizáveis buscados durante a implementação, este circuito acabou se tornando demasiado complexo. Essa característica acabou impactando negativamente na capacidade de síntese do projeto em **FPGAs** de baixo custo. Além deste circuito, nenhuma outra estrutura sofreu grande impacto nessa nova implementação. As **Unidades Funcionais** permaneceram implementadas exatamente sobre os mesmos critérios adotados na primeira versão do projeto, Seção 3.1.5.

Com essas alterações foi possível obter níveis de *speedup* promissores. A adição de novas **Unidades Funcionais**, em conjunto com o controle na quantidade *BW* de forma proporcional, garantiu um ganho em desempenho próximo aos níveis ideais. O objetivo principal dessa etapa do desenvolvimento foi plenamente alcançado. Foi verificado a contento a capacidade de implementação de um protótipo inspirado pela **MDFM**, em **FPGAs** de baixo custo, conforme a hipótese inicial. Entretanto, foram encontradas diversas limitações para que este aspecto pudesse ser atingido.

3.2.4 Limitações

Devido a complexidade dos circuitos necessários para a distribuição das operações de cada unidade dentro dos dutos de largura de banda, a capacidade de síntese da solução acabou ficando bastante prejudicada, principalmente pela implementação parametrizável buscada durante o desenvolvimento. Este aspecto se mostrou bastante importante para análise de desempenho da organização da π FlowBW. Entretanto, a complexidade dos circuitos acabou prejudicando a implementação, principalmente com relação à baixa frequência do relógio a qual foi possível a síntese da solução, que era de apenas 5MHz, devido ao grande caminho crítico necessário para a implementação.

Um dos circuitos que apresentavam grande complexidade e, portanto caminho

crítico muito longo, é o de distribuição de pacotes entre os dutos de largura de banda e as **Unidades Funcionais**. Entretanto, foi possível observar outros trechos da descrição que também apresentavam caminhos críticos muito longos. Um deles é o circuito disparado diante da resposta de leitura na memória RAM da **Unidade de Pareamento**, que ainda realiza a geração do pacote de saída desta unidade. E da mesma forma, o circuito equivalente da **Unidade de Instruções**, que garante a geração do seu pacote de saída, também apresentava um longo caminho crítico, impedindo a síntese à frequências mais altas.

Mesmo para fins de simulação, apenas, passou a se tornar inviável a construção da solução com maior quantidade de recursos. Isso porque o **BSV** armazena todo o código executável do simulador dentro da memória RAM do computador que está realizando a compilação. Para soluções mais complexas, que neste projeto pode ser manejada facilmente a partir das constantes que determinam os valores de referência do protótipo, não é possível realizar a simulação imediatamente. O **BSV** oferece algumas opções de compilação que permitem o aumento dos limites, como por exemplo, da memória RAM que pode ser utilizada durante o processo de compilação. Com a utilização destas opções, para soluções mais complexas, foram verificados alguns erros de sobrecarga no uso de memória durante o processo de compilação, impedindo até mesmo a simulação desta implementação com uma quantidade maior de recursos na construção da máquina.

Foi observado durante os testes preliminares, que é necessário o mínimo de duas vezes a quantidade de **Unidades Funcionais** de *BW* para que haja níveis de *speedup* que sejam suficientemente apreciáveis. Dessa forma, para uma solução com 10 **Unidades Funcionais**, é necessário o mínimo de 20 dutos de largura de banda. Este se tornou o limite superior aceitável para análise do comportamento do protótipo nesta versão do desenvolvimento, tratando-se de um valor bastante baixo para a análise de uma arquitetura paralela. Portanto, este critério acabou se tornando grande fonte de inspiração para o desenvolvimento da próxima etapa, que será apresentada em detalhes no próximo Capítulo 5, que deu origem ao desenvolvimento efetivo do presente projeto.

4 OTIMIZAÇÕES NA π FLOWBW

Conforme descrito em detalhes no Capítulo 3, referente à implementação realizada ainda durante o desenvolvimento do trabalho de mestrado deste autor, foi possível verificar que a estrutura da **MDFM** não é capaz de oferecer níveis em *speedup* apreciáveis quando implementada com todos os recursos sob o mesmo domínio do relógio*. Entretanto, foi proposta uma implementação que permitiu a exploração de paralelismo a partir da introdução de um conceito que ficou conhecido como dutos de largura de banda. Essa característica promoveu um aumento significativo no *throughput* da máquina, garantindo um aproveitamento bastante razoável das **Unidades Funcionais** que compõem sua estrutura. Com isso, foi validada a hipótese inicial, sobre a possibilidade de implementação e viabilidade dessa arquitetura, utilizando **FPGAs** de baixo custo.

Apesar de verificar com sucesso estes aspectos, também foi possível verificar que a quantidade de recursos necessários para a exploração de níveis apreciáveis em paralelismo acaba se tornando demasiado alto. Alguns circuitos específicos se mostraram muito complexos para implementação em **FPGA**, principalmente devido a adição de certa capacidade de parametrização na distribuição destes recursos. Dessa forma, o objetivo principal desta etapa do desenvolvimento foi a verificação do comportamento da π **FlowBW**, promovendo a divisão dos recursos necessários para implementá-la de forma mais otimizada. Com isso, sendo esperada a possibilidade de atingir uma quantidade ilimitada, a priori, na quantidade de recursos possíveis de serem explorados, em um sistema paralelo e distribuído, utilizando uma estratégia que ficou conhecida como **Multianel**.⁴⁷⁻⁴⁹

4.1 Otimizações de *hardware*

Tendo em vista a série de limitações verificadas no projeto inicial, o início do presente projeto concentrou a maior parte dos esforços em otimizar a estrutura original. Foram verificados diversos circuitos que apresentavam longos caminhos críticos e com isso impediam a síntese da solução a frequências mais altas do relógio. A fim de reduzir os caminhos críticos, estes circuitos foram divididos em diversas ações distintas. Para isso, as regras do **BSV** que determinam o comportamento dessas ações foram distribuídas em múltiplas operações, ou seja, algumas operações que haviam ficado concentradas em apenas uma regra do **BSV**, foram distribuídas em diversas regras.

Na **Unidade de Pareamento** o circuito com maior caminho crítico verificado estava presente na ação de resposta da leitura na memória interna e consequente geração

* Notar que na **MDFM** as **Unidades Funcionais** operavam a uma frequência de relógio mais lenta em comparação com o restante da estrutura da máquina^{5,14}

do pacote de saída desta unidade. Ambas as ações eram realizadas dentro de uma mesma regra. Como o **BSV** realiza todas as operações contidas em uma regra dentro do mesmo circuito, em elaboração estática, isso acabava promovendo este longo caminho crítico. Dessa forma, a operação foi dividida em duas etapas distintas. O armazenamento de uma ficha na fila de saída da memória RAM dispara a ação da regra destinada para o armazenamento desta ficha em um registrador de estados simples. Este armazenamento então, dispara a ação da regra seguinte, destinada para a geração do pacote de saída da unidade. Na **Unidade de Instruções**, foi adotada uma solução exatamente igual, pois o principal gargalo verificado nesta unidade tratava da mesma etapa de seu funcionamento.

Já na **Unidade de Processamento**, algumas soluções foram revisadas e restrições foram adotadas para remover o longo caminho crítico que se mantinha persistente desde a primeira versão do projeto. A estrutura em *round-robin* foi removida, pois foi possível verificar que a quantidade de fichas que chegavam a esta unidade já era suficiente para explorar melhor sua distribuição entre as **Unidades Funcionais** disponíveis, sem a necessidade de uma estratégia específica para otimizá-la. Ou seja, a ocorrência de prioridade nas primeiras **Unidade Funcionais** verificada antes da implementação dos dutos de largura de banda deixou de ocorrer. Além disso, ainda foi adicionado um *buffer* de armazenamento interno na entrada das **Unidades Funcionais**, com tamanho mínimo, de duas fichas. Essa propriedade tornou o circuito para verificação de disponibilidade da unidade ainda mais simples, dependendo apenas da verificação na capacidade de armazenamento neste *buffer*.

Para síntese, a quantidade de dutos de largura de banda possíveis de serem implementados foi restringida apenas a valores maiores que a quantidade de **Unidades Funcionais** presentes na solução. Essa restrição foi importante, pois permitiu a remoção de algumas cadeias de laços aninhados que eram utilizados para a geração do circuito de distribuição entre os dutos e as **Unidades Funcionais**. Dessa forma, foi mantido apenas um registrador de histórico efêmero,^{18,50} que verifica se determinada ficha já foi alocada à uma **Unidade Funcional** anterior. Perceba portanto, que esta implementação foi realizada em um laço limitado pela quantidade de **Unidades Funcionais** verificada em ordem crescente. Com isso, foi mantida a capacidade de todos os dutos se comunicarem com todas as unidades, a todo ciclo do relógio. A análise de disponibilidade é realizada ainda nas condições explícitas dessas regras, portanto implementado de forma estritamente estática no processo de compilação do **BSV**.

Além da revisão dessas características, diversos circuitos dependiam de controle realizado a partir de registradores de histórico efêmero. Este elemento de estados primitivo tem como função o armazenamento de todo o histórico de valores que foram atribuídos a ele no decorrer de cada ciclo completo do relógio. Dessa forma, perceba tratar-se de um elemento que exige muitos recursos para ser implementado no *hardware*. Grande parte

destes recursos foram substituídos por registradores simples, de forma que uma revisão detalhada teve de ser realizada na lógica de funcionamento das unidades que faziam uso deste recurso. Além disso, para a distribuição de fichas e pacotes por entre os dutos de largura de banda, foram utilizados diversos laços aninhados, a fim de garantir controle de parametrização para a geração da solução. A maioria destes laços puderam ser removidos e passaram a ser constituídos a partir de lógica combinacional, em elaboração estática apenas, utilizando-se de estratégias existentes na construção nativa do **BSV**.⁴⁶

A revisão destes circuitos permitiu a síntese da solução a uma frequência de 50MHz, que era a principal expectativa buscada neste instante do desenvolvimento, e trata-se de uma frequência dez vezes mais alta com relação a versão anterior. Porém, essa revisão também acabou promovendo uma perda considerável no desempenho global da implementação. Afinal, as operações de resgate das informações presentes nas memórias **RAM** de composição das **Unidades de Pareamento e Instruções**, que antes já alimentavam o *buffer* de saída dessas unidades, acabou sendo atrasado. Dessa forma, foi perdido um ciclo do relógio para a manipulação de cada ficha que passa por essas unidades. Note que estes atrasos são sistemáticos durante a execução de programas, e não podem ser observados na análise das curvas de *speedup*. Entretanto, foi verificado anteriormente a capacidade de desempenho que essa estrutura é capaz de oferecer. Com isso, foram buscadas alternativas que permitissem obter os mesmos desempenhos verificados anteriormente, porém com circuitos mais curtos e portanto, com frequência de operação mais alta.

Outra deficiência verificada, que acabava promovendo o aumento do caminho crítico, foram sobre as operações de divisão. Esta operação é necessária para a distribuição de fichas entre os dutos de largura de banda, principalmente na **Fila de Fichas** e na **Unidade de Processamento**. Diante da criação de novas fichas, é necessário identificar em qual duto ela será alocada durante o resgate de suas informações. Para isso, deve ser realizada a operação da divisão do endereço de destino dessa ficha pela quantidade total de dutos de largura de banda presentes na solução. Devido ao processo de alocação da memória na **Unidade de Instruções**, Seção 3.2, essa operação, quando realizada a partir das operações nativas do **BSV**, acaba promovendo a geração de um longo caminho crítico, que depende exclusivamente da largura, em *bits*, dos elementos utilizados na operação. Na π **FlowBW**, este tamanho refere-se à quantidade de *bits* necessários para descrever a quantidade de dutos de largura de banda da solução ($\log_2 BW$).

Para lidar com essa limitação foram adicionadas implementações próprias para a realização de divisão. Nessa proposta, é fixada de forma explícita a largura dos registradores utilizados para classificar a quantidade de dutos de largura de banda. Além disso, foi implementado de forma parametrizável, a capacidade de identificar a largura de *bits* do divisor em operações de divisão. Se essa largura for uma potência de dois, então a operação é realizada a partir de deslocamento de *bits* apenas, caso contrário, é implementado um

circuito para a realização da divisão. Essa característica permitiu que a simulação pudesse utilizar qualquer valor de largura de banda, para a capacidade de geração de todos os arranjos possíveis para a composição da máquina. Porém, para a síntese foi tomada a convenção de que a quantidade de dutos será sempre fixada em valores que sejam potências de dois, para facilitar a geração do circuito lógico disposto para este processo. Dessa forma, é tomado como quantidade de dutos, a próxima potência de dois mais próxima do dobro da quantidade de **Unidades Funcionais**. Ou seja, para 2 unidades, serão dispostos 4 dutos; para 3 e 4 unidades, 8 dutos; para 5, 6, 7 e 8 unidades, 16 dutos e assim sucessivamente.

Apesar de reduzir a quantidade de operações nos caminhos críticos mais longos da **Unidade de Processamento**, não foi verificado nenhum atraso sistemático que fosse capaz de prejudicar o desempenho nesta unidade. Dessa forma, foram buscadas alternativas que permitissem a obtenção de um desempenho equivalente àquele verificado anteriormente, mantendo a mesma capacidade de frequência. Inspirado pelas características das fichas de passagem direta introduzidas já na estrutura da **MDFM**, as alternativas buscadas têm o objetivo de garantir o mesmo efeito. Apesar de apenas as **Unidades de Pareamento** e de **Instruções** apresentarem a redução de desempenho verificada, foram buscadas estratégias de aproveitamento em todas as unidades, a fim de obter o melhor desempenho que essa nova estrutura fosse capaz de oferecer. Estas soluções serão descritas em detalhes na próxima seção.

4.2 Otimizações de passagem direta

É importante reforçar neste momento, que todas as implementações descritas nesta seção foram realizadas em todos os dutos de largura de banda, de cada trecho descrito em detalhes a seguir. Na **Fila de Fichas**, a comunicação entre os *buffers* de entrada e o de saída em cada duto de largura de banda, já foi considerada suficiente para mitigar o atraso proveniente da passagem de fichas pela fila de armazenamento interna da unidade. Portanto, esta solução já foi tomada como suficiente para evitar perda em desempenho em qualquer instante durante a execução de programas. Com isso, não foi empregado maior esforço sobre as operações realizadas nesta unidade da máquina. Perceba que essa característica foi proposta ainda nas primeiras versões da implementação e portanto, também foi tomada como inspiração das implementações descritas nesta seção.

4.2.1 Unidade de Pareamento

Apesar de já contar com as características nativas da arquitetura, com as fichas de passagem direta, que já são suficientes para oferecer um desempenho diferenciado na **Unidade de Pareamento**, foram buscadas alternativas que pudessem aperfeiçoar o desempenho da máquina com relação à passagem de fichas para pareamento. Para isto, foi

implementada a utilização da segunda porta de comunicação na memória **BRAM** que compõe esta unidade. Este componente nativo do **BSV** possui duas portas de comunicação, que garantem a capacidade para o envio de duas solicitações de leitura e/ou escrita simultaneamente, ou mesmo em ciclos subsequentes do relógio [†].⁴⁶ Dessa forma, a partir desta solução, em cada duto de largura de banda, na ocorrência de fichas para pareamento em ciclos subsequentes do relógio, passou a ser possível a realização da requisição de leitura e eventualmente consequente escrita nesta memória, sem a necessidade de ociosidade destas fichas em um *buffer* intermediário.

Para a realização de uma análise assertiva sobre a utilização corrente das portas de comunicação em uso nesta memória, foi necessária a criação de um circuito complexo o suficiente para causar um aumento significativo no caminho crítico da solução. Inicialmente então, foi implementado um circuito que permitia identificar a utilização de quaisquer das portas de comunicação no ciclo corrente, para a partir de então realizar a alocação da outra porta para a análise das fichas subsequentes. Porém, este circuito se tornou demasiado complexo, sendo removido imediatamente diante da análise do consumo de recursos. Foi utilizado então, um circuito muito mais simples para esta função, que utiliza os parâmetros do endereço da instrução que a ficha recém chegada à unidade deve executar e seu **Nome de Ativação**. Para evitar que esta análise pudesse se tornar enviesada, dando maior prioridade para uma das portas, foi realizada uma operação lógica “Ou Exclusivo”, entre os primeiros *bits* destes parâmetros, para determinar qual porta deve ser utilizada para a comunicação. Com isso, diante da necessidade de comunicação com essa memória, são verificadas essas propriedades em um circuito simples, que apenas analisa os parâmetros contidos na ficha corrente, sendo determinado de forma estática qual porta será utilizada para realizar a operação de escrita e/ou leitura para esta ficha específica. Apesar de não garantir que ambas as portas serão utilizadas na ocorrência de fichas para pareamento subsequentes, esta implementação promove a utilização de ambas as portas se aproveitando da casualidade para alcançar esta capacidade.

Para verificar a utilização de cada uma destas portas de comunicação, foi adicionado um registrador de histórico efêmero que apenas é incrementado diante da ocorrência de fichas de pareamento na segunda porta de comunicação implementada. Foi possível verificar um baixo aproveitamento de utilização desta segunda porta de comunicação, a partir dos programas utilizados para validação da máquina. Entretanto, os princípios desta estrutura se mostraram realmente interessantes segundo o princípio de funcionamento da arquitetura, que já apresentava este gargalo ainda na máquina original. A implementação foi realizada em poucas ações adicionais, de forma a não impactar negativamente no caminho crítico, ou substancialmente na quantidade de recursos necessários para tal. Com isso, como pode ser visto na Figura 21, a **Unidade de Pareamento** passou a contar com duas estratégias

[†] Lembrando que cada porta de comunicação leva o período fixo de um ciclo do relógio para efetivar a realização de leituras ou escritas nesta memória **RAM**.

Figura 21 – Diagrama de representação da **Unidade de Pareamento**, com as modificações propostas para leitura da segunda porta de comunicação da memória **RAM** desta unidade.

Fonte: Elaborada pelo autor.

capazes de evitar a ociosidade de fichas aguardando por ações de comunicação com sua memória interna, problema persistente desde a implementação da **MDFM**.

4.2.2 Unidade de Instruções

Inspirado pelas estratégias de passagem direta para ganho em desempenho verificadas da **Unidade Pareamento**, na **Unidade de Instruções** também foi promovida a implementação de dois recursos para esta finalidade. Afinal, esta unidade conta com um grande gargalo para a arquitetura desde a primeira versão do projeto, levando um período fixo de ciclos do relógio para efetivar suas ações. Para lidar com este problema, foi

proposta a implementação de uma pseudo predição de desvios, como ficou conhecida esta solução, adicionando o equivalente a um registrador de “*cache*” a esta unidade, conforme será descrito em detalhes a seguir. Além disso, foi implementada a utilização da segunda porta de comunicação na memória **RAM** que compõe essa unidade, bem como a solução verificada na seção anterior.

É imediato verificar, segundo a discussão da Seção 2.3, que durante a execução de programas - principalmente aqueles que apresentam grande potencial de paralelismo - em uma arquitetura baseada no modelo a fluxo de dados dinâmico, grande parte das instruções executadas simultaneamente referem-se à instruções equivalentes, apenas em instâncias distintas. Conforme descrito anteriormente na Seção 3.1.2, na **MDFM** estas instâncias são obtidas a partir das características dinâmicas desta arquitetura. Ao considerar o modelo de implementação utilizado na introdução dos dutos de largura de banda, que determina o duto por onde uma ficha irá percorrer todas as unidades que compõem a máquina no momento de sua criação, é possível verificar que as fichas que estão transitando por entre estas estruturas, em geral, são destinadas à mesma instrução, apenas nestas instâncias distintas. Dessa forma, foi tomada a hipótese de que a última instrução lida em determinado duto de largura de banda na memória da **Unidade de Instruções**, deveria ser bastante utilizada pelas fichas subsequentes, transitando por determinado duto, durante certo período na execução de programas que apresentem alto grau de paralelismo.

Com isso, foi implementado um registrador de estados, em cada duto de largura de banda, que armazena a última instrução lida na memória da **Unidade de Instruções**. Diante da chegada de uma nova ficha nessa unidade, é verificado o endereço dessa instrução. Se essa instrução for referente àquela armazenada neste registrador, já é gerado o pacote de saída da unidade imediatamente. Caso contrário, se o endereço da ficha recém chegada à unidade não for igual ao endereço armazenado neste registrador de “*cache*”, então é realizada uma nova leitura na memória **RAM**. Ao final do processo de leitura, o pacote de saída é gerado, conforme o funcionamento convencional dessa unidade. Porém, esta instrução fica armazenada no registrador de “*cache*”, para a verificação de acerto nas próximas fichas subsequentes. Perceba portanto, que a capacidade de acerto desta estrutura acaba sendo proporcional à quantidade de dutos de largura de banda. Afinal, cada duto irá armazenar instruções bastante específicas durante a execução de programas, conforme descrito em detalhes na Seção 3.2.

Além do sistema de “*cache*” introduzido, ainda foi promovida a utilização da segunda porta de comunicação da memória **BRAM** que compõe essa unidade, bem como a solução proposta para a **Unidade de Pareamento**, como pode ser visto na Figura 22. A estratégia utilizada para essa implementação foi exatamente análoga àquela descrita em detalhes anteriormente. A verificação para identificação da porta que será utilizada também foi realizada a partir de uma análise lógica nos primeiros *bits* dos parâmetros

Figura 22 – Diagrama de representação da **Unidade de Instruções**, com as modificações propostas para leitura da segunda porta de comunicação da memória **RAM** desta unidade e introdução da pseudo predição de desvios.

Fonte: Elaborada pelo autor.

de endereço da instrução e **Nome de Ativação**, portanto, dependente exclusivamente da ficha que realizou a geração dessa operação. Essa análise é realizada no conjunto de condições explícitas da regra que determina este comportamento e portanto, efetuada de forma puramente estática na descrição do código **BSV**. Essa característica é bastante importante de ser analisada, pois leva a uma menor utilização de recursos necessários no *hardware*.

Para a análise de ambas as estruturas de aperfeiçoamento desta unidade, foram adicionados alguns registradores de histórico efêmeros que são apenas incrementados diante da ocorrência de quaisquer das capacidades de passagem direta destacadas. Foi possível observar uma grande capacidade de aproveitamento do sistema de *cache* nos programas utilizados para validação da máquina. Para estes programas, em geral, mais de 50% das fichas que chegam a essa unidade acabam se aproveitando dessa estrutura, um número realmente bastante promissor. Algumas outras análises foram feitas, adicionando capacidades maiores para armazenamento de instruções, simulando realmente uma memória *cache* para o armazenamento da vizinhança da última instrução lida nesta memória. Entretanto, o caminho crítico da solução acabou aumentando consideravelmente. Para

lidar com esse aumento, uma nova memória **RAM** poderia ser utilizada, mas ela traria consigo a mesma latência para leitura, não fazendo sentido seu uso, de acordo com a proposta desta solução. Então, ficou mantida a estratégia de armazenamento apenas da última instrução lida nessa memória, em cada duto de largura de banda. Perceba ainda, que o armazenamento da vizinhança da última instrução lida, promove o acúmulo de instruções relativamente aleatórias, segundo a distribuição de instruções nas memórias que compõem os dutos de largura de banda da solução, fugindo portanto da estratégia utilizada para essa solução.

Por terem sido utilizados exatamente os mesmos critérios de análise para a escolha de uma ou outra porta de comunicação na memória **RAM**, a estrutura de utilização da segunda porta de comunicação seguiu o mesmo baixo aproveitamento que já havia sido verificado na **Unidade de Pareamento**. Da mesma forma, esta solução foi mantida, por sua capacidade, mesmo que limitada, em impedir o armazenamento de fichas ociosas durante processos de comunicação com a memória. Entretanto, a utilização do sistema de *cache* se mostrou tão eficaz quanto à utilização da estrutura responsável pelo tratamento de fichas de passagem direta da **Unidade de Pareamento**. Lembre-se que a exploração dessa característica depende exclusivamente do código do programa que está sendo executado. A equipe de desenvolvimento do compilador da **MDFM** promovia de forma bastante contundente a geração de fichas para esta finalidade.⁵¹ A capacidade de implementar uma propriedade, em nível de *hardware*, que possui um aproveitamento equivalente àquele obtido em nível de *software*, mostra um grande sucesso nesta proposta de implementação. Entretanto, é importante reforçar que a pseudo predição de desvios apresenta grande potencial, principalmente para programas que possuem características de distribuição muito “comportadas”, para uma quantidade suficiente de dutos e consequente distribuição do programa nestes componentes da máquina.

4.2.3 Unidade de Processamento

Para ganho em desempenho na **Unidade de Processamento** foi aproveitada a estrutura já existente do sistema de pré-processamento. Foi verificado que grande parte das instruções executadas pelos programas utilizados para validação da máquina, referiam-se a instruções bastante simples. Essas instruções podiam ser realizadas em apenas um ciclo do relógio. Entretanto, de acordo com a implementação das **Unidades Funcionais**, a execução dessas instruções acabavam tomando o período fixo de 5 ciclos para serem completamente executadas. Perceba que este princípio de funcionamento é, inclusive, incentivado de acordo com a definição desta arquitetura. Afinal, quanto maior a porcentagem de utilização das **Unidades Funcionais**, melhor será sua capacidade para escalabilidade, conforme pôde ser verificado na análise dos resultados da primeira versão do projeto⁵² ou mesmo na **MDFM**.¹⁴

Apesar da busca pelo máximo aproveitamento das **Unidades Funcionais**, neste instante do desenvolvimento deste trabalho, eram buscadas estratégias que permitissem a execução de programas no menor tempo possível. A pseudo unidade de pré-processamento tinha a finalidade de lidar apenas com a geração de novos Nomes de Ativação. Em busca de melhorar o desempenho da máquina, foi utilizada esta estrutura já existente, para a execução das principais instruções mais simples, que exigissem poucos recursos para serem implementadas e que não impactassem negativamente na utilização das **Unidades Funcionais**. Diversos testes foram realizados, mantendo instruções como soma e subtração entre números inteiros; e duplicação de fichas, dentro do sistema de pré-processamento. A partir desta estratégia, foi verificado um baixo aproveitamento das **Unidades Funcionais**, o que acabou promovendo uma degradação na capacidade para ganho em *speedup* a partir desta solução. Dessa forma, foram mantidas apenas instruções de controle e a instrução completa para a geração de Nomes de Ativação dentro do pré-processador.

Este módulo é “ativado” para ser utilizado diante da chegada de pacotes que possuam o código de operação de quaisquer instruções destinadas para execução a partir dele. O código de operação dos pacotes de entrada da **Unidade de Processamento** são analisados nas condições explícitas da regra destinada para implementar esta capacidade. Quando ativada, essa regra possui uma estrutura de análise de casos para cada código de operação específico. Ao término da execução, as fichas resultantes são armazenadas em uma fila destinada para suas passagens imediatas aos *buffers* de saída da **Unidade de Processamento**. As fichas subsequentes geradas pelas **Unidades Funcionais** possuem maior prioridade para utilização destes *buffers* de saída. Dessa forma, as fichas geradas pelo módulo de pré-processamento ficam aguardando em uma fila intermediária, até que possam ser encaminhadas ao *buffer* de saída do duto de comunicação com a **Fila de Fichas**.

Com relação à quantidade de instruções executadas pelos programas, a utilização do sistema de pré-processamento ficou bastante baixa. Entretanto, bem como em algumas das soluções verificadas nas outras unidades, essa estratégia acaba favorecendo apenas um pequeno ganho em desempenho na máquina, principalmente em instantes na execução dos programas quando não há uma grande quantidade de paralelismo disponível para ser explorado, normalmente identificados no início e final de programas. E essa foi a principal estratégia utilizada na **Unidade de Processamento** para ganho em desempenho nesta etapa do desenvolvimento.

4.3 Conjunto de Instruções

Neste instante sobre a realização de ajustes iniciais da π **FlowBW** para os propósitos do **Multanel**, foram realizadas também algumas alterações e melhorias na implementação

das **Unidades Funcionais**. Algumas instruções foram apenas revisadas, como as operações de soma e subtração por exemplo, que foram definidas para serem realizadas dentro da mesma regra do **BSV**, com isso, reduzindo a quantidade de recursos necessários para implementá-las. Conforme descrito no Capítulo 3.1.5, existe um estado específico na operação das **Unidades Funcionais** responsável pela execução efetiva da instrução, no formato de *pipeline* utilizado em sua implementação. Entretanto, existem algumas instruções que levam mais de um ciclo do relógio para serem executadas. Um exemplo notável herdado dessa versão da implementação é a instrução **TUP**. Essa instrução é semelhante à **DUP**, sendo responsável pela duplicação em massa de fichas. A quantidade de cópias é definida por sua entrada direita. Na versão antiga da implementação um único registrador era utilizado para marcar o estado da execução destas unidades. Essa característica acabava obrigando um tamanho maior para este registrador, que utilizava valores superiores apenas em poucas instruções. Dessa forma, foi adicionado um novo registrador responsável por lidar com a contagem de operações dentro de instruções de múltiplos ciclos. Além da **TUP**, instruções novas, que foram definidas no conjunto de instruções neste instante da implementação, também se aproveitaram desta nova construção.

Ainda foram implementadas instruções para lidar com o rótulo dinâmico conhecido como **Nível de Iteração**. Para isso, também foram adicionadas as instruções responsáveis por gerenciar os recursos deste rótulo nas **Unidades Funcionais**, conforme as definições do conjunto de instruções da **MDFM**.¹⁶ Entretanto, a principal modificação implementada nessa unidade foi a adição de suporte à instruções que operam sob valores em Ponto Flutuante. Para isso, foi utilizada uma biblioteca nativa do **BSV** que dá suporte a estas operações. Dessa forma, foram instanciados diversos módulos da biblioteca ***FloatingPoint*** no escopo das **Unidades Funcionais**. Foram adicionados apenas os módulos de soma, multiplicação e **CORDIC**. Este último não possui suporte a partir das bibliotecas padrões e portanto, teve de ser implementada em detalhes, conforme descrito no Apêndice A.

Uma característica importante dessas operações para o conjunto de instruções desta arquitetura paralela é a necessidade de diversos ciclos do relógio para sua execução. Tanto para a realização de soma (e consequente subtração), quanto a multiplicação (e consequente divisão) são necessários o período fixo de 5 estágios para serem completamente executadas, que são distribuídos entre múltiplos ciclos do relógio para fazê-los. Note que estes estágios adicionais destas operações específicas são realizados dentro do estágio de execução efetiva de uma instrução na estrutura da **Unidade Funcional**. Com isso, somando-se aos outros 4 estágios já nativos dessas unidades, a execução destas instruções levam o período total de 9 ciclos do relógio para serem efetivadas. Essa característica promove um maior aproveitamento no uso das **Unidades Funcionais** que receberem este tipo de instruções durante a execução de programas, garantindo melhor ganho em *speedup*, conforme será destacado em detalhes nos próximos capítulos. Estes módulos

presentes nas bibliotecas nativas do **BSV** foram implementados em formato de *pipeline*. Porém, para permitir ganho em desempenho a partir desta estrutura, elas foram utilizadas para a execução de cada instrução por vez. Perceba portanto que existe um relativo desperdício de recursos devido à necessidade de registradores intermediários entre cada processo durante sua execução. Entretanto, o ganho em desempenho verificado acaba compensando tal desperdício. Note que a implementação dessas instruções também se aproveita do registrador de estágios inserido na composição da **Unidade Funcional**, uma vez que as operações precisam de mais ciclos do relógio para sua execução.

Para completar o suporte a essas operações ainda foram adicionadas as instruções de comparações sobre valores reais. A instrução **CLR** e **CGR** são responsáveis por verificar se seus valores de entrada são menores ou iguais ou maiores entre si, respectivamente. Para sua implementação foram utilizadas apenas as funções de suporte presentes na biblioteca padrão do **BSV**. E por fim, as instruções **SIN** e **COS** são responsáveis pelo cálculo do seno e cosseno, respectivamente. As operações implementadas a partir do módulo **CORDIC** possui como principal destaque a necessidade de 32 ciclos do relógio para serem executadas. Perceba que essa característica é capaz de aumentar consideravelmente o ganho em desempenho durante a execução de programas que se utilizem destas instruções, garantindo um rico aproveitamento das **Unidades Funcionais** alocadas para a execução dessas operações. Os detalhes da implementação deste módulo estão disponíveis no Apêndice [A](#).

5 π FLOWMR

Nos capítulos anteriores, foi possível verificar que a quantidade de recursos necessários para implementar a arquitetura da **MDFM** em **FPGA** é extremamente alta, a fim de obter níveis de desempenho apreciáveis nesta estrutura paralela, principalmente da forma como foi realizada essa implementação, buscando alternativas que possibilitassem a parametrização dos valores que determinam suas características de operação. Apesar do conceito do **Multianel** ter sido verificado anteriormente a partir de simulações,^{47,48} é importante observar que ele surgiu como uma estratégia natural para a evolução do presente projeto. O nome π **FlowMR** foi proposto como acrônimo para o termo em inglês que descreve a estrutura da máquina *MultiRing*.

Ainda na primeira versão da construção da π **Flow**, já foi verificada a necessidade de distribuição de recursos a fim de obter capacidade para a exploração de paralelismo e escalabilidade. O conceito que ficou conhecido como dutos de largura de banda, que além da comunicação paralela entre as unidades que compõem sua estrutura, ainda promove a repartição dos recursos de cada unidade, foi tomado como principal motivação para o desenvolvimento desta tese. O **Multianel** consiste principalmente na distribuição dos recursos necessários para implementar uma máquina baseada no modelo a fluxo de dados. Nesta solução, o anel representado pela π **FlowBW** passa a constituir uma unidade básica e fundamental da máquina. Em analogia direta, o anel que constitui a arquitetura, nesta nova abordagem, pode ser interpretado como uma **Unidade Funcional** da versão inicial.

Tome como exemplo o cenário de uma aplicação com um total de 10 **Unidades Funcionais**. Na π **FlowBW**, seria necessária uma quantidade muito grande de recursos para conseguir obter ganho em desempenho. Para garantir o aproveitamento dessa quantidade de unidades de processamento, seria necessário o mínimo de 20 dutos de largura de banda. Será discutido em detalhes no Capítulo 6, mas é importante reforçar que a quantidade de recursos necessários para implementar essa solução cresce de acordo com uma função polinomial quadrática aproximadamente, entre as quantidades de **Unidades Funcionais** e dutos de largura de banda, de acordo com a Equação 6.7. Entretanto, nesta nova abordagem, seria possível alcançar essa capacidade de processamento utilizando outras configurações. Por exemplo, poderiam ser utilizados 5 anéis da π **FlowBW** cada qual com um total de apenas 2 **Unidades Funcionais** e conseqüentemente, apenas 4 dutos de largura de banda.

A quantidade de recursos necessários para implementar a solução na configuração apontada, é muito menor do que a solução inicial, na π **FlowBW**, apesar de serem necessários mais recursos físicos, neste caso um total de 5 **FPGAs**. Porém, cada **FPGA** estará subutilizada nesta distribuição, permitindo o uso destes recursos extras em outras

características da máquina, como a implementação de memória adicional por exemplo. Conforme será discutido em detalhes na sequência, essa propriedade acaba promovendo também a remoção de gargalos naturais existentes nesta arquitetura, provenientes do acúmulo de fichas em determinados momentos durante a execução de programas, semelhante ao que foi possível verificar na análise da solução que ficou conhecida como *Throttle*⁴³ na **MDFM**. Isso porque as fichas devem transitar entre os anéis que compõem esta estrutura, e este “atraso” adicionado naturalmente nesta implementação, permite a remoção de qualquer sobrecarga que poderia estar presente dentro de determinados dutos de largura de banda segundo a estratégia de anel único. Além disso, foi possível verificar também que esta solução é ainda capaz de promover um relativo aumento na granularidade implementada na arquitetura.

5.0.1 Arquitetura

A principal análise realizada neste momento, se deu de acordo com os recursos disponíveis para implementação. Para esta tese, foram utilizados *kits* de desenvolvimento **Terasic** DE10-Nano, que possuem um **SoC Cyclone V SE** embarcado em sua composição. Este **SoC** possui uma **FPGA** embutida com capacidade de 110K elementos lógicos. O **SoC** ainda possui um sistema de processamento **HPS** composto por um **ARM Cortex-A9** e portando, o *kit* de desenvolvimento é composto por diversos periféricos destinados para comunicação com este processador embarcado. Entretanto, devido aos protocolos específicos e limitados utilizados por estes periféricos - para o uso necessário no presente projeto - foi definido o uso dos conectores **GPIO** para a comunicação entre as **FPGAs**, no conceito do **Multianel**. Cada *kit* DE10-Nano possui dois módulos de **GPIO** de 40 pinos cada, conectados diretamente à **FPGA** do **SoC**. Dessa forma, foi proposto um protocolo de comunicação paralelo exclusivo para a comunicação entre as **FPGAs** que compõem a estrutura da máquina, conforme será descrito em detalhes nas próximas seções.

Em contraposição ao modelo previsto pelo **Multianel** proposto para a **MDFM**^{*, 47} na π **FlowMR**, a comunicação entre os anéis é realizada apenas entre seus vizinhos mais próximos. Dessa forma, foi tomada como inspiração a configuração de cada anel, para a definição da topologia de comunicação entre os anéis. Essa característica foi proposta devido a limitação encontrada nos recursos disponíveis nos *kits* de desenvolvimento utilizados no projeto. Cada *kit* possui um total de apenas dois módulos de **GPIO**. A fim de obter uma estrutura genérica o suficiente para os critérios de avaliação buscados nesta implementação, foi proposto que os módulos de **GPIO** devem se comunicar aos pares apenas. Com isso, estes módulos podem ser explorados em toda sua largura de banda disponível, ou seja, todos os 40-bits disponíveis em cada módulo podem ser utilizados em cada comunicação

* Segundo este modelo, todos os anéis podiam se comunicar entre si a partir de uma topologia de comunicação em estrela.

Figura 23 – Topologia do **Multianel** segundo a implementação proposta para a π **FlowMR**.

Fonte: Elaborada pelo autor.

necessária. Além disso, foi proposto neste projeto que cada módulo deve ser capaz de promover comunicações bidirecionais entre os anéis, como destacado na Figura 23.

Neste instante da implementação, as fichas sofreram algumas alterações devido a necessidade da realização de alguns testes na máquina, para a execução de programas distintos e também para viabilizar a implementação do **Multianel**. Com isso, as fichas dentro dessa nova estrutura passaram a ser formadas de acordo com a seguinte definição:

$$\langle \text{dado}(37\text{-bits}), \text{rótulo}(24\text{-bits}), \text{destino}(10\text{-bits}), \text{paralelismo}(10\text{-bits}), \\ \text{anel}(5\text{-bits}), \text{marcador}(3\text{-bit}) \rangle \quad (5.1)$$

Com relação à definição das fichas na primeira versão do projeto, é possível destacar algumas mudanças consideráveis. O tamanho do “rótulo” dinâmico presente em cada ficha aumentou em *12-bits* devido à definição de um rótulo extra que não havia sido utilizado até o presente momento. Esta cadeia de *bits* tinha a finalidade de identificar apenas o rótulo dinâmico conhecido como **Nome de Ativação**, com o tamanho de *12-bits* para fazê-lo. Este rótulo dinâmico da arquitetura é responsável pela capacidade da máquina em realizar chamadas a funções em seu conjunto de instruções. Foi adicionado então um outro rótulo, conhecido como **Nível de Iteração**. Este rótulo tem a finalidade de lidar com laços dentro do conjunto de instruções da **MDFM**. Foi utilizado o mesmo tamanho para ambos os rótulos, cada qual com um total de *12-bits*, caracterizando com isso, a propriedade dinâmica identificada como “Cor”, dentro do contexto definido na literatura da **MDFM**.

A cadeia de *10-bits* identificada como “paralelismo” foi adicionada na solução apenas para fins de simulação, a partir da propriedade de compilação condicional do **BSV**, que será descrita com mais detalhes na sequência. Esta cadeia de *bits* tem a finalidade de permitir a identificação no nível de paralelismo em que se encontra determinada ficha, a fim de determinar a métrica conhecida como *Paralelismo Médio* do programa em execução. Essa métrica será definida em detalhes na Seção 6.1.

A propriedade destacada como “anel” tem a finalidade de identificar em qual anel da estrutura do **Multianel** a ficha deve ser executada. Para isso, é armazenado em cada ficha a identificação do anel para o qual esta ficha deve ser encaminhada. Essa definição é analisada na **Unidade de Processamento**, em um circuito disposto antes do envio das fichas ao *buffer* de saída desta unidade. E finalmente, a propriedade definida como “marcador” na Equação 5.1, tem a finalidade de identificar a qual operação uma ficha é destinada. Para lidar com todas as etapas necessárias para o correto funcionamento do **Multianel**, este marcador pode conter diversos estados. Os estados possíveis que uma ficha pode assumir dentro desta solução são os seguintes: Negociação; Saída; Comunicação à Esquerda ou à Direita; Instrução para Execução; Sobrecarga na Comunicação; Ficha de Entrada; e Ficha de Programa. Cada um destes estados serão descritos em detalhes ao longo deste capítulo, quando pertinente.

De acordo com a Figura 23, é importante reforçar que apesar de contar com linhas simples para representar a comunicação entre as unidades de cada anel, a estratégia da π **FlowBW** que foi implementada nesta etapa do projeto. Por esse motivo que as linhas seguem o padrão destacado no diagrama. Além disso, a comunicação entre a **Unidade de Processamento** e a **Fila de Fichas** segue com a implementação de comunicação dupla. A representação tomou esse formato por medidas estéticas apenas. Além disso, percebe-se que a representação da comunicação entre os anéis possui uma característica única. Essa representação foi adotada pois cada ficha precisou ser desmembrada em múltiplas partes,

afinal elas possuem um tamanho de *89-bits* segundo a Equação 5.1, porém existem apenas *40-bits* disponíveis para transmissão em cada módulo **GPIO**. O protocolo de comunicação paralelo proposto será descrito em detalhes a seguir, na Seção 5.1. O módulo de **Troca de Entrada e Saída** representado na figura refere-se ao *TopLevel* da aplicação descrita em **BSV**. Todas as interfaces de comunicação entre as **FPGAs**, e consequentemente, entre os anéis da máquina, foram determinados nesta instância do projeto, que é responsável por definir as portas de comunicação da solução, no código **Verilog** gerado após o processo de compilação.

Cada anel da implementação é capaz de se comunicar apenas com seus vizinhos mais próximos. Apesar da limitação de localidade, diante da geração de novas fichas dentro de determinado anel, percebe-se que qualquer ficha é capaz de ser encaminhada a qualquer anel da solução proposta. Porém, para isso, é necessário que essa ficha percorra por todos os anéis necessários até atingir seu destino final. As unidades de Chaveamento, análogo à estrutura da **MDFM**, foram implementadas no *TopLevel* da π **FlowBW**. Além disso, um módulo específico de comunicação foi implementado, a fim de permitir que essa operação pudesse ser realizada em um outro domínio do relógio e que fosse padronizada sobre a estrutura do **Multianel**. Sendo assim, a operação de comunicação entre os anéis é realizada a uma frequência de operação mais alta em comparação ao restante das unidades que compõem cada anel.

Cada anel nesta implementação possui dois módulos de comunicação definidos em seu escopo, em conjunto com as outras unidades que compõem a π **FlowBW**. Toda a solução referente ao fluxo de informações relacionado aos módulos de comunicação dessa estrutura[†] ficaram definidos no *TopLevel* da aplicação. Para a operação da máquina, foi necessária a definição de um anel principal na sua composição, de forma que este será o anel responsável por iniciar a execução de programas e negociação entre os anéis. Essa operação ocorre de acordo com a descrição seguinte.

Diante da operação de *Reset* no anel principal, é iniciado um processo de negociação por toda a máquina. Um registrador é responsável por armazenar a identificação exclusiva de cada anel. Para isto, neste instante, o registrador de identificação do anel principal é escrito com o valor 0 e neste processo, uma ficha marcada com o estado de negociação é lançada ao anel vizinho, com a finalidade de identificar todos os anéis presentes na solução. Esta ficha é criada no anel principal com o valor 1 definido como o dado da ficha, segundo a Equação 5.1. Neste processo, as fichas são sempre encaminhadas à direita do anel corrente, até atingir o anel identificado com o valor 0 novamente. Ao chegar no anel subsequente, o valor desta ficha é armazenado em um registrador local e na sequência é incrementado. Após essa operação a ficha é encaminhada ao anel subsequente, seguindo por toda a

[†] Registradores exclusivos e regras de comportamento sobre a entrada e saída de fichas nestes módulos.

extensão da máquina. Perceba que ao retornar para o anel principal, a ficha armazena a quantidade total de anéis presentes na composição do **Multianel**. Essa informação é armazenada em um outro registrador do anel principal e a ficha percorre por todo o caminho novamente a fim de determinar este tamanho em cada unidade.

Cada anel da máquina possui um registrador de estados que é atualizado diante da realização de cada operação. Este registrador possui o tipo de dado **Enum**[‡] e pode assumir os seguintes valores: Ocioso, Negociação, Pronto, Recebendo, Esperando e Executando. Após a realização do *Reset*, o anel principal entra no estado “Ocioso”. Uma vez neste estado, já no primeiro ciclo do relógio, este anel faz a geração da ficha de negociação, alterando o estado deste registrador. A passagem da ficha de negociação altera o estado de cada anel. Ao final deste processo, todos os anéis encontram-se no estado identificado como “Pronto”, indicando que a máquina está pronta para receber um novo programa. Na ocorrência da entrada de fichas pelo *buffer* de alimentação do anel principal, este entra no estado “Recebendo”. Este *buffer* de alimentação se mantém em comunicação direta com o módulo **HPS** do **SoC** no *kit* de desenvolvimento utilizado como anel principal da máquina. Esta operação de alimentação será detalhada na Seção 5.3. Neste estado o anel principal encaminha todas as fichas passando por este *buffer* para o anel conectado à sua direita. Uma vez no estado “Pronto”, todo anel que recebe uma nova ficha de seu módulo de comunicação à esquerda entra no estado “Recebendo”. Neste estado, diante da ocorrência de novas fichas, os anéis encaminham-nas à **Unidade de Instruções**, para escrita na memória de armazenamento desta unidade. E uma cópia é encaminhada ao seu módulo de comunicação da direita.

Após a escrita de todo o programa em todos os anéis da estrutura é necessário invocar uma interface passiva do anel principal. Essa interface altera o registrador de estados deste anel para “Esperando”, e se manterá neste estado até que todas as fichas de entrada do programa corrente entrem na máquina a partir de seu *buffer* de alimentação. Perceba que essas fichas são marcadas com essa característica, e, portanto, seus marcadores possuem o estado identificado como “Ficha de entrada”. Essas fichas são encaminhadas diretamente à interface de alimentação da **Fila de Fichas** do anel principal. Essa interface tem a finalidade de inserir fichas explicitamente dentro de um duto de largura de banda de um anel específico. Dessa forma, ao final do recebimento das fichas de entrada, o anel principal entra no estado “Executando” e a partir deste instante a máquina já se encontra em plena execução do programa corrente.

Para a realização de testes e simulações, foi criado um *TestBench*, que para essa finalidade se tornou o *TopLevel* do projeto. Este módulo realiza a instância de quantos anéis forem determinados a partir de um valor constante presente no arquivo de definições da máquina. Os módulos de comunicação são conectados aos seus vizinhos da mesma maneira

[‡] Tipo de dado especial que pode assumir exclusivamente os valores definidos em seu escopo.⁴⁶

como deverá ser implementado em **FPGA**. O *Testbench* se encarrega de realizar todas as chamadas ativas que serão executadas pelo módulo **HPS** do *kit* de desenvolvimento a partir da solução sintetizada. A leitura do programa de entrada da máquina é realizada a partir do comando *\$fopen* nativo do **BSV**, destinado para a leitura e escrita de arquivos, que é totalmente equivalente à função de mesmo nome, nativa da linguagem **C**, por exemplo. Uma máquina de estados simples é utilizada para a execução de todo processo ativo presente na descrição da negociação à execução do programa corrente.

De acordo com a especificação da solução criada inicialmente, para a comunicação entre as unidades que compreendem a π **FlowBW** foi utilizada uma interface nativa do **BSV**, conhecida como *mkConnection*. Essa interface possui métodos muito bem definidos para troca de informações entre módulos. Como as interfaces dos módulos de comunicações ficam dispostos no *TopLevel* de cada anel, foi necessária uma pequena modificação dessa característica para possibilitar o resgate de fichas destinadas a outros anéis. Para isso, toda ficha encaminhada entre a **Unidade de Processamento** e a **Fila de Fichas** é interceptada neste *TopLevel*. Se uma ficha for destinada a outro anel da máquina, ela é transmitida imediatamente ao *buffer* de alimentação do módulo de comunicação que estiver mais próximo do destino dessa ficha, com o fluxo de informações seguindo as especificações do diagrama da Figura 24. Caso contrário, se for uma ficha destinada ao anel corrente, então ela apenas é encaminhada ao *buffer* de entrada convencional da **Fila de Fichas**. Para identificar uma ficha que seja destinada à comunicação é analisado nessa interface a identificação do anel que deve executar a ficha, informação presente em todas as fichas, de acordo com a Equação 5.1. Além disso, para determinar para qual módulo de comunicação esta ficha é destinada são utilizados os marcadores “Comunicação à Esquerda, ou Direita”. Essas informações são calculadas e adicionadas a todas as fichas, ainda dentro da **Unidade de Processamento**[§].

De acordo com a topologia de comunicação implementada, existe a possibilidade de transitarem fichas dentro dos anéis, que não sejam destinadas ao anel corrente. Por isso, foi implementada ainda uma transmissão direta de conexão entre ambos os módulos de comunicação presentes em cada anel. Diante desta ocorrência, a ficha é encaminhada imediatamente ao *buffer* de entrada do módulo de comunicação oposto, conforme destacado na Figura 24. A descoberta do destino de determinada ficha será descrita em detalhes na próxima Seção 5.0.2.

Diversos indicadores foram utilizados para identificar os recursos utilizados na máquina durante a execução de programas[¶]. Entretanto, essas análises puderam ser

[§] Esta definição foi realizada na **Unidade de Processamento**, ao invés da **Unidade Funcional**, para permitir que ela pudesse ser centralizada, a partir de um conjunto de regras parametrizadas do **BSV**. Afinal, na **Unidade Funcional**, existe uma regra para a determinação das propriedades de cada uma das duas fichas de saída possíveis.

[¶] São exemplos notáveis de indicadores analisados a: Utilização máxima da **Fila de Fichas** e

Figura 24 – Diagrama de representação da arquitetura e fluxo de informações da π FlowBW com a adição dos módulos de comunicação.

Fonte: Elaborada pelo autor.

implementadas apenas para o processo de simulação, devido à utilização de registradores de histórico efêmero para a verificação destes valores, que são os únicos elementos de estados que permitem múltiplas manipulações dentro de um mesmo ciclo do relógio. Essa capacidade é importante para permitir o incremento de seu valor na ocorrência da utilização de algum recurso em múltiplos dutos de largura de banda, dentro do mesmo ciclo do relógio. Este elemento de estados consome uma grande quantidade de recursos para ser implementado em **FPGAs**.⁵⁰ Para identificar as características dispostas na implementação para síntese ou simulação, foi utilizado o recurso de compilação condicional do **BSV**.⁴⁶ Essa propriedade pode ser utilizada a partir de uma opção de compilação, que determina o equivalente a uma “variável global” para a solução. Com isso, diversos recursos foram implementados apenas para fins de simulação, devido à limitação de recursos para o processo de síntese.

5.0.2 Distribuição de Fichas

Inspirado pela solução utilizada no armazenamento de fichas na **Unidade de Pareamento** da π FlowBW, foi definido um método equivalente para a distribuição de fichas entre os anéis do **Multianel**. Para determinar o anel onde uma ficha deve ser executada, é tomado o valor do resto da divisão entre a soma dos rótulos **Nome de Ativação** e **Nível de Iteração**; e a quantidade total de anéis que compõem a solução. Por esse motivo, durante o processo de negociação entre os anéis, as informações com a

da memória **RAM** da **Unidade de Pareamento**; Porcentagem de utilização das **Unidades Funcionais**; e Utilização da pseudo predição de desvios na **Unidade de Instruções**.

identificação do anel e a quantidade total de anéis são armazenadas em registradores de estados simples, dentro da **Unidade de Processamento**.

Diante da descoberta do anel responsável pela execução de uma ficha particular, na **Unidade de Processamento**, ela já é marcada com essa informação. Ainda é realizada uma verificação que analisa qual o menor caminho entre o anel corrente e o anel ao qual a ficha é destinada. A marcação na ficha é feita de acordo com essa informação, ou seja, se o menor caminho estiver à esquerda do anel corrente, a ficha será marcada como “Comunicação à esquerda” e vice-versa. Note que o menor caminho é sempre fixo, se mantendo igual diante da movimentação dessa ficha por entre os anéis da máquina, permitindo que este cálculo possa ser realizado uma única vez durante o tempo de vida de uma ficha. A identificação do anel que irá executar a ficha também é colocada na sua composição, conforme destacado na Equação 5.1. Essa informação já é determinada no processo de identificação descrito e trata-se da condição de parada para uma ficha transmitida de um anel a outro. Ao ser encaminhada ao *buffer* de saída dessa unidade, o *TopLevel* da aplicação intercepta a comunicação com a **Fila de Fichas** e analisa se a identificação do anel da ficha é igual a do anel corrente. Em caso negativo, essa ficha é encaminhada ao *buffer* de entrada do módulo de comunicação determinado em sua marcação.

O **Nome de Ativação** é o rótulo dinâmico responsável por lidar com chamadas de funções. De forma equivalente, o **Nível de Iteração** é responsável pelas instruções que controlam a execução de laços. Os principais programas utilizados para validação da máquina foram implementados de forma recursiva e não contam com laços em sua composição. Dessa forma, foram analisadas principalmente a capacidade da máquina para lidar com programas que apresentam grande capacidade de paralelismo derivado da recursividade.

Uma propriedade importante de reforçar neste momento é que no conjunto de instruções da **MDFM**, o **Nome de Ativação** é gerenciado pelo *hardware*, pois existem instruções que apenas requisitam um novo **Nome de Ativação**. Na π Flow, este valor é armazenado em um registrador de histórico efêmero da **Unidade de Processamento** que é apenas incrementado na ocorrência da instrução de geração de um novo **Nome de Ativação**. Note que novamente, este elemento de estados complexo é utilizado no lugar de um registrador simples, para lidar com o gerenciamento de estados provenientes de eventuais conflitos de recursos entre os dutos de largura de banda. Na sequência da execução, estes programas apenas utilizam este valor gerado que acaba se tornando intrínseco da arquitetura. Já o **Nível de Iteração**, deve ser gerenciado pelo próprio programa. Diante da necessidade de um novo valor de referência, neste caso, uma nova iteração no laço, uma instrução realiza o incremento de um valor indicado explicitamente no código do programa. Com isso, existe uma forte dependência das instruções anteriores para

a exploração de paralelismo. O **Nome de Ativação** é capaz de explorar um paralelismo nativo da estrutura da máquina, sem a necessidade da execução de iterações anteriores para determinar seu escopo. Por esse motivo, para validação na **MDFM**, programas recursivos eram mais utilizados, por apresentar melhor capacidade de paralelismo. E da mesma forma, se tornaram os principais programas utilizados para a validação do presente projeto.

De toda forma, independente de qual rótulo dinâmico for utilizado, a π **FlowMR** é capaz de realizar uma distribuição de recursos de forma equivalente em ambas as estratégias. Afinal, é utilizada uma composição entre os dois valores para determinar o destino de uma ficha. Perceba que de acordo com este método todas as instruções dependentes de um determinado rótulo serão executadas dentro de um único anel. Ou seja, todas as instruções de uma função serão executadas pelo mesmo anel. Note que essa estratégia acaba promovendo naturalmente um relativo aumento na granularidade da máquina. O conjunto de instruções se manteve intacto com relação à **MDFM**, e portanto utilizando uma granularidade bastante fina. Entretanto, devido à essa aglutinação de instruções que serão executadas em cada “unidade” da estrutura do **Multianel**, é possível observar este aumento de granularidade indicado. Afinal, esta propriedade promove certa especificidade à cada anel da máquina, conceito que será aprofundado no Capítulo 6.

Os principais programas utilizados para validação neste projeto, possuem uma distribuição bastante “comportada”. Por terem sido implementados de forma recursiva, as instruções executadas em cada instância, segundo o modelo dinâmico do fluxo de dados, são perfeitamente homogêneas. Essa característica dos programas é capaz de se aproveitar bastante da estratégia de distribuição analisada durante o processo de validação da máquina, garantindo um balanceamento de carga bastante homogêneo de acordo com esta estratégia. Porém, é imediata a verificação de que programas mais generalizados, que não apresentem esta capacidade de homogeneização pode acabar promovendo certa tendência na distribuição de fichas, para determinado anel. Entretanto, para fins práticos, essa distribuição é realizada a partir de uma função estática que pode inclusive possuir um caráter reprogramável para aplicações futuras. Apesar de que, neste momento, ela ficou estática de acordo com as definições descritas.

É importante observar também, que esta implementação ainda é capaz de remover quaisquer gargalos que pudessem ser verificados dentro da implementação anterior. O paralelismo possível de ser extraído nesta arquitetura é extremamente grande, principalmente se aproveitando de programas recursivos. Na π **FlowBW**, as instruções presentes em cada escopo de função podem atingir um estado de bloqueio com bastante facilidade, a partir da distribuição de fichas nos dutos de largura de banda, que podem se acumular devido ao problema de localidade dessas instruções, na memória da **Unidade de Instruções**. Porém, nessa nova abordagem, este cenário não é verificado, devido a distribuição de

fichas entre os anéis, de acordo com a instância de cada chamada de funções, ou mesmo de iteração em laços. Dessa forma, o **Multianel** acaba promovendo certa capacidade de atraso para a execução de instruções, principalmente devido ao caminho necessário para uma ficha atingir o anel que irá executá-la. Conforme dados da literatura, essa capacidade de atraso é benéfica para a exploração de grande grau de paralelismo.^{43, 53}

O **Nome de Ativação** é um rótulo que deve ser único em toda a estrutura da máquina. Se este rótulo for duplicado, ele causará conflito de recursos na **Unidade de Pareamento**, independentemente do anel onde ele for gerado e executado, pelo fato da memória desta unidade ser endereçada a partir da simples concatenação entre este rótulo e o endereço da instrução a ser executada. Dessa forma, para garantir a unicidade deste valor, os Nomes de Ativações foram distribuídos por “cotas” na π **FlowMR**. Para isso, durante o processo de negociação, diante da descoberta da quantidade total de anéis é disparado um método na interface da **Unidade de Processamento**. Este método armazena as informações do **Multianel** em registradores locais e determina o valor do primeiro **Nome de Ativação** disponível para o anel corrente.

Para identificar o primeiro **Nome de Ativação** é necessária a realização da divisão entre a quantidade total que este rótulo é capaz de atingir (determinado pela largura em *bits* deste rótulo), pela quantidade de anéis presentes na configuração. Por este motivo, essa operação foi realizada de forma estática, considerando apenas o tamanho do **Nome de Ativação** e a quantidade total de anéis definida em tempo de compilação do **BSV**. Após a determinação deste primeiro valor, a geração de novos Nomes de Ativação é realizada exatamente da mesma forma que anteriormente, ou seja, a partir do incremento no valor de um registrador de histórico efêmero. Essa implementação permite a geração deste valor em qualquer duto de largura de banda no caso de conflitos dentro de um mesmo ciclo do relógio. Além disso, na geração de novos valores, é verificado se o novo **Nome de Ativação** encontra-se dentro da “cota” disponível para o anel corrente. A partir desta estratégia, a distribuição de fichas no **Multianel** se tornou mais homogênea. Cada anel é capaz de gerar fichas para ele próprio e para todos os outros anéis da máquina.

Durante a execução de programas, pode ocorrer a geração em massa de fichas destinadas a outros anéis da estrutura. A análise do menor caminho para a realização dessa distribuição já é capaz de mitigar um acúmulo de fichas destinadas para os outros anéis. Porém, ainda pode haver uma grande aglomeração de fichas nos *buffers* de alimentação de ambos os módulos de comunicação presentes em cada anel. Este aspecto pode acabar desencadeando o bloqueio de um anel, pois o preenchimento completo destes *buffers* impede que fichas possam ser encaminhadas aos anéis vizinhos. Além disso, pode impedir que os anéis vizinhos possam encaminhar suas fichas a eles, devido ao caráter bidirecional desta comunicação. Dessa forma, foram buscadas estratégias que fossem capazes de lidar com eventuais sobrecargas dessa natureza.

5.0.3 Sobrecarga

Desde as versões iniciais deste projeto, foi lançada uma hipótese interessante para resolver o problema de sobrecarga na construção de cada anel, inspirado pela unidade de mesmo nome da **MDFM**. Essa análise foi revisitada para lidar com eventuais sobrecargas sobre a estrutura de comunicação do **Multianel**. Para lidar com o problema verificado anteriormente, de sobrecarga de fichas destinadas aos módulos de comunicação, seria suficiente a existência de um grande *buffer* que fosse capaz de acumular tantas fichas quanto necessárias, enquanto os anéis subsequentes consumiam os recursos que já tivessem sido encaminhados a eles. Entretanto, a criação de filas intermediárias de tamanhos grandes entre cada unidade que compõe cada anel acabaria promovendo a utilização de muitos recursos lógicos das **FPGAs**. Porém, percebeu-se que a descrição deste problema, define com bastante precisão uma unidade já presente na estrutura da π **FlowBW**. A **Fila de Fichas** funciona como um *buffer* para o anel, e naturalmente já realiza o armazenamento massivo de fichas, enquanto as unidades subsequentes consomem os recursos já destinados a elas.

Dessa forma, foi proposta uma pseudo **Unidade de Sobrecarga**, como ficou conhecida. Esta “pseudounidade” se aproveita de recursos já existentes e bem definidos na máquina. A **Fila de Fichas** possui uma grande fila de armazenamento interno com capacidade para o total de 4096 fichas em sua composição, quantidade herdada ainda da **MDFM**.¹⁴ Este valor inclusive foi determinado de forma parametrizável e pode ser alterado com bastante facilidade na π **FlowBW**. Mesmo sua distribuição por entre os dutos de largura de banda já foi implementada com base neste valor. Sendo assim, qualquer manipulação nessa capacidade de armazenamento pode ser analisada com bastante detalhes, por ser definida em tempo de compilação. Com isso, foi proposto um novo estado de marcação para as fichas. Este valor de marcação determina que uma ficha se encontra em um estado de **Sobrecarga** dentro da estrutura de um anel.

Para isso, foram criados novos arcos de comunicação em cada duto de largura de banda, de cada unidade que compõe a π **FlowBW**. Estes arcos não interferem em nenhum circuito já definido nessas unidades, apenas se aproveitam dos *buffers* de entrada e saída já existentes. As fichas marcadas com este estado possuem maior prioridade para serem tratadas dentro de cada unidade. Estas fichas são encaminhadas imediatamente do *buffer* de entrada para a saída de cada módulo componente dos anéis. Na **Fila de Fichas**, essas fichas marcadas são naturalmente armazenadas no final do *buffer* de armazenamento interno dela. Com isso, sofrendo um atraso natural da arquitetura para ser encaminhada novamente ao arco de sobrecarga de cada unidade do anel. Todas as fichas que deixam a **Unidade de Processamento** são interceptadas pelo *TopLevel* da aplicação. Ao identificar fichas marcadas com essa característica, elas passam por um novo processo de tentativa de envio aos módulos de comunicação. Em caso negativo, ou seja, se ainda houver sobrecarga nestes módulos, essa ficha retorna para o armazenamento interno da **Fila de Fichas**. Essa

rotina é repetida enquanto houver sobrecarga nos módulos de comunicação.

Foi verificado apenas um aumento sensível no armazenamento da **Fila de Fichas**, a partir deste procedimento. Essa característica reforça o sucesso dessa implementação e valida a hipótese inicial, de que a **Fila de Fichas** é suficiente para atuar contra eventuais sobrecargas da **MDFM**. Com isso, essa mesma estrutura pode ser utilizada para lidar, por exemplo com eventuais sobrecargas na **Unidade de Pareamento**, característica prevista pela **MDFM** e omitida nas primeiras versões deste projeto. Essa propriedade seria capaz de permitir uma redução considerável na capacidade de armazenamento dessa unidade, reduzindo a quantidade de recursos lógicos necessários para o processo de síntese. Dessa forma, o endereçamento na memória desta unidade poderia ser realizada de acordo com uma função *hash* mais assertiva entre rótulos e endereço da instrução de execução da ficha. Entretanto, neste momento, foi buscada apenas a passagem de fichas de comunicação por esta estrutura de sobrecarga, uma vez que o consumo exagerado de recursos lógicos já pôde ser mitigado naturalmente na estratégia do **Multianel**.

5.1 Protocolo de Comunicação

Para implementar o processo de comunicação entre os anéis no **Multianel**, foi necessário lidar com a escassez de recursos de comunicação disponíveis nos *kits* de desenvolvimento utilizados para a validação deste projeto. Atualmente existem diversos *kits* que possuem recursos de comunicação muito mais abundantes para a entrada e saída de informações nas **FPGAs** destas soluções. Entretanto, como foi proposto o uso de dispositivos de baixo custo para a implementação deste projeto, foi escolhido o *kit* de desenvolvimento DE10-Nano descrito anteriormente. Eles foram adquiridos em um período próximo de seu lançamento com valores especiais para fins acadêmicos. Apesar de existirem soluções com preços ainda mais acessíveis a partir de outros fabricantes, com recursos mais especializados em comunicação de maior desempenho, foi escolhida a marca utilizada principalmente pela carência de fornecedores dos fabricantes concorrentes, no período de aquisição destas soluções.

Foi analisado brevemente o protocolo *Gigabit Ethernet* para a realização da comunicação entre as **FPGAs** nesta tese. Com o uso de um *switch* comercial convencional seria possível a implementação da topologia proposta para a **MDFM**, conectando todos os anéis da estrutura entre si, em uma topologia de estrela. Porém, este protocolo possui grande limitação para o envio de informações dentro de sua frequência de operação. Afinal, ele dispõe de apenas *8-bits* para a transmissão de informações à 125MHz. Essa capacidade é excelente para o tráfego de um grande volume de informações, chegando ao limite de largura de banda que dá nome ao protocolo, sendo capaz de entregar até 1Gb/s. Entretanto, fazendo uma análise mais simplista e menos rigorosa, temos que a π **FlowBW** estava

configurada para operar à frequência de 50MHz e uma ficha possui o tamanho de *89-bits*. Com isso, percebe-se que seriam necessários o mínimo de 10 ciclos do relógio da máquina para a transmissão de uma ficha completa, de um anel a outro. Esta que poderia ser uma capacidade bastante razoável considerando a topologia de comunicação possível de ser alcançada.

Entretanto, segundo o protocolo *Ethernet* padrão **IEEE 802.3**, o quadro mínimo para transmissão de um pacote é de *512-bits*.⁵⁴ Dessa forma, na verdade seriam necessários o mínimo de 26 ciclos do relógio da máquina para a transmissão de uma única ficha, de um anel até o *switch* de distribuição. Ou seja, 53 ciclos para transmissão de fichas completas entre os anéis da máquina. Essa aproximação ainda desconsidera quaisquer atrasos que ocorrem naturalmente para a transmissão de informações entre diferentes domínios de frequências para comunicação com o módulo **HPS**, além da hipótese de que as informações seriam encaminhadas imediatamente chegando ao *switch*. Com isso, a utilização deste protocolo para a transmissão de informações entre os anéis foi completamente desconsiderado.

O *kit* de desenvolvimento utilizado possui diversos outros módulos de comunicação destinados principalmente para abrigar os periféricos de entrada e saída no dispositivo **HPS** embarcado no **SoC** da DE10-Nano. Tratam-se, portanto, de protocolos de comunicação muito específicos, que ainda contam com a frequência de operação deste **HPS** para sua operação, além das definições de frequências características de cada protocolo, de acordo com suas especificações. Esta troca de domínios de frequências e a possibilidade de perda de informações durante este processo, acabou desmotivando o uso destes recursos para esta tarefa também.

Com isso, passou a serem verificadas as possibilidades de uso dos módulos de entrada e saída de propósitos gerais, presentes aos pares e que possuem comunicação direta com as **FPGAs** nestes *kits* de desenvolvimento. Ou seja, os pinos destes módulos são conectados diretamente aos pinos de entrada e saída da **FPGA**, não dependendo de um intermediário, como é o caso dos outros módulos de comunicação presentes na DE10-Nano, que dependem do dispositivo **HPS** para que possam ser acessados. Apesar de não serem destinados para comunicações de alto desempenho, foram avaliadas estratégias que permitissem a operação dos módulos **GPIO** a frequências relativas aos 50MHz que a máquina já era capaz de atingir. Estes módulos de comunicação são dispostos em um formato equivalente ao padrão **ATA**. Este padrão apresenta 40 pinos sendo distribuídos entre duas fileiras de 20 pinos e distanciamento equivalente entre todos eles, conforme a especificação do diagrama da Figura 25. Essa característica foi avaliada com bastante cuidado, uma vez que a possibilidade da ocorrência de interferência e atenuação entre os pinos e as vias de comunicação é bastante alta, em um protocolo paralelo.

Para lidar com a possível ocorrência de interferências, foi proposta a utilização do padrão **ATA** introduzido no modo **UDMA/66**.⁵⁶ Neste padrão, todas as vias de

Figura 25 – Diagrama de representação do conector de um cabo que segue a especificação ATA/ATAPI-6 para comunicação paralela.

Fonte: NCITS⁵⁵

transmissão são pareadas a vias conectadas ao terra, em um cabo do tipo *flat*. Essa propriedade é suficiente para amenizar a ocorrência de interferências durante sua operação a frequências mais baixas. O valor de referência buscado durante a implementação era a operação dos módulos de comunicação a uma frequência capaz de atingir pelo menos $5 \times$ os 50MHz já característicos de cada anel. Ou seja, foi buscada uma frequência de operação de 250MHz neste protocolo de comunicação paralelo. Para isso, foi proposto o uso de um cabo flexível com comprimento muito menor que os 457mm do padrão **UDMA**.

Na construção dos módulos de comunicação, tentou-se torná-los o mais integrado quanto possível ao funcionamento de cada anel. Para isto, no *TopLevel* de cada anel suas interfaces de entrada e saída são conectados diretamente às interfaces de comunicação destes módulos, disponibilizando os métodos para troca de informações de cada anel. A adição dessa característica facilitou bastante a construção do ambiente de simulação mas também, para a utilização deste projeto no processo de síntese em **FPGA**.

Os módulos de **GPIO** dos *kits* de desenvolvimento DE10-Nano são distribuídos em duas fileiras de 20 pinos. Destes 40 pinos, apenas 36 são destinados para propósito

geral ao usuário. Os quatro pinos restantes são utilizados para alimentação de dois níveis de energia e conexão ao terra do *kit*. Com isso, segundo o protocolo criado, 31 pinos são utilizados para o transporte de dados. Ou seja, quase o dobro da capacidade de transmissão do protocolo **ATA**, que utilizava apenas 16 vias do cabo para esta finalidade, com o restante distribuído para a realização de controle e negociação. Na π **FlowMR**, os 5 pinos restantes ficaram destinados para a realização de controle. Além disso, foi utilizado o método de comunicação *half-duplex*. Apesar da comunicação ser bidirecional, o mesmo meio de transmissão é utilizado em ambos os sentidos.

Na estrutura do *kit* de desenvolvimento, os pinos 0 e 1 são capazes de transportar sinais de relógio. Por este motivo, o pino 0 ficou encarregado de transportar este sinal, a fim de sincronizar os módulos de comunicação que estejam em contato direto dentro do sistema de comunicação proposto. Neste protocolo, as comunicações serão realizadas segundo uma estratégia de mestre e escravo. Dessa forma, o mestre será responsável por determinar todas as operações que devem ser realizadas entre cada par de módulos. Este recurso é controlado a partir dos 5 pinos que são destinados ao controle. Como o *kit* de desenvolvimento possui 2 módulos de **GPIO**, um em cada lado da **FPGA**, para fins práticos, foi tomada a convenção de que o módulo à direita da placa será identificado como o mestre de uma comunicação, sendo o módulo à esquerda sempre identificado como escravo. Todos os módulos **GPIO** devem estar conectados para obter a topologia de comunicação proposta, de acordo com a Figura 23. Com isso, sempre haverá um mestre e um escravo determinado em cada *kit* de desenvolvimento.

O módulo mestre sempre terá maior prioridade para a realização da comunicação. Dessa forma, caso ambos os módulos possuam fichas disponíveis para transmissão, a ficha presente à esquerda, segundo a estrutura do **Multianel**, terá maior prioridade para o envio. Apenas na ausência de quaisquer fichas neste módulo, o escravo da comunicação terá “permissão” para encaminhar suas fichas. Para isso, dois pinos do módulo **GPIO** são utilizados para determinar a direção da comunicação. Este conjunto de pinos é definido a partir de um tipo de dado **Enum** presente na composição do **BSV**. Este tipo de dado é capaz de tornar explícito os estados que determinado elemento pode conter dentro de sua definição.⁴⁶ Estes pinos de controle podem possuir 3 estados distintos, que são: Ocioso; Mestre-para-Escravo; Escravo-para-Mestre. Existe uma regra exclusiva no módulo do mestre responsável por determinar este estado da comunicação que é transmitido a todo ciclo do relógio do mestre ao escravo. Esta regra analisa diversos recursos do módulo de comunicação para determinar este estado.

Para a determinação da direção da comunicação em determinado instante, foi definido que um pino do módulo é responsável por indicar ao mestre que o escravo da conexão possui fichas e pode encaminhá-las imediatamente, diante de disponibilidade para tal. Neste caso, ainda é verificado se o *buffer* de recebimento de fichas do módulo mestre

é capaz de armazenar novas informações. E finalmente, o último pino de controle tem a finalidade de indicar ao módulo mestre que o escravo está cheio e não é capaz de receber novas fichas no ciclo corrente. Com isso, os *31-bits* restantes nos módulos **GPIO** são destinados ao envio de dados dentro deste protocolo, que foi implementado a partir de um *buffer* de comunicação de três estados, nativo do **BSV**. Este módulo consiste em um pino de controle que ativa a operação a ser realizada pelas vias de entrada e saída deste *buffer*. Este pino de ativação é gerenciado de acordo com os estados dos pinos de controle do protocolo. Este módulo é definido de acordo com o diagrama da Figura 26. O *buffer* de três estados (*tristate*) utiliza uma interface do tipo *Inout* para a transmissão de informações. Essa interface é apenas um apelido para o módulo de mesmo nome definido na sintaxe do **Verilog**. O **Bluesim**^{||} não possui suporte para a implementação desta interface de comunicação. Dessa forma, para a simulação do **Multianel** foi necessária a utilização do simulador **Icarus Verilog**.⁵⁷ Apesar de não contar com as estratégias de otimização verificadas no **Bluesim**, este simulador também é bastante rigoroso com relação à solução de projetos de larga escala.

Figura 26 – Diagrama esquemático do módulo de três estados utilizado para a elaboração dos pinos de comunicação das vias de transmissão de dados no **Multianel**.

Fonte: Adaptada de **BLUESPEC**⁴⁶

Uma característica importante deste protocolo de comunicação é a capacidade do módulo em encaminhar qualquer quantidade de *bits* a partir da cadeia de pinos de dados. Uma ficha, em sua descrição atual, possui *89-bits* de comprimento, conforme a Equação 5.1, mas este valor pode ser alterado com bastante facilidade de acordo com a descrição da implementação. Para a realização da divisão de uma ficha em blocos de tamanhos iguais a *31-bits* foram utilizados os módulos *mkFunnel* e *mkUnfunnel*, presentes na biblioteca **PacLib** do **BSV**.⁵⁸ O módulo *mkFunnel* realiza a divisão das informações em vetores de

^{||} Simulador padrão e nativo do **BSV** utilizado em todas as etapas do desenvolvimento até este ponto.

28, presente em todos os pinos dos módulos de **GPIO** das principais placas comerciais desta empresa. Este circuito tem a finalidade de proteger estes pinos de comunicação contra baixa e alta tensão. O aparato constituído pelos diodos *schottky* tem a finalidade de limitar os níveis de tensão que passam por estes pinos, de acordo com a corrente verificada. Este limitador, em conjunto com o resistor mostrado na figura, são bastante importantes a fim de evitar sobrecarga sobre estes pinos, principalmente na borda inicial de uma transmissão, quando o sinal ainda não está estabilizado e pode apresentar um pico de tensão em todos os instantes que uma corrente começa a passar por este componente. Por esse motivo, para utilização dessa placa, foi instituída a necessidade da construção de uma placa de expansão para ser utilizada no *kit* de desenvolvimento, de acordo com as necessidades deste projeto, como definido em detalhes no Apêndice B.

Figura 28 – Diagrama esquemático do circuito de proteção presente nos principais *kits* de desenvolvimento da **Terasic**, porém omitidas na **DE10-Nano** utilizada para implementação deste projeto.

Fonte: TERASIC⁵⁹

Este módulo de expansão consiste em uma placa de circuito impresso simples, com um conector **IDE** “fêmea” em sua base, que segue as especificações do diagrama da Figura 25, que deve ser conectado ao módulo **GPIO** do *kit* de desenvolvimento. Deste conector saem vias de transmissão de todos os pinos e que passam pelo circuito destacado da Figura 28. A partir deste circuito suas vias são dirigidas à um outro conector **IDE** “macho”, segundo as especificações dispostas na Figura 29, presente no topo desta placa de expansão, para a conexão dos cabos de transmissão. Esta solução é aplicada a todos os 36 pinos de propósito geral dos módulos **GPIO** do *kit*, de acordo com a distribuição

destacada na Tabela 1. É utilizado um cabo de comunicação *flat* de 80 vias, introduzido no padrão de comunicação **UDMA/66**.⁵⁶ Os cabos de comunicação tiveram de ser produzidos também, uma vez que o padrão **ATA** possui pinos de ligação à terra diferentes quando comparados com os módulos **GPIO** presentes na DE10-Nano. Com isso, foi necessário revisar a estrutura física dos conectores **ATA** de 80 vias e 40 pinos que possui uma pequena chapa condutora composta por algumas estrias que tem a finalidade de fazer contato com a ligação ao terra neste padrão. Estas chapas foram modificadas, removendo as estrias do padrão **ATA** e adicionando novas, coerentes com os pinos de conexão ao terra do *kit*, segundo os padrões especificados.⁶⁰

Figura 29 – Diagrama de representação do cabeçalho de um conector que segue a especificação **ATA** para comunicação paralela.

Fonte: NCITS⁵⁵

Dessa forma, de acordo com todas as especificações detalhadas até este ponto, o protocolo de comunicação proposto consiste na disposição de pinagem determinada no diagrama da Figura 30 e a distribuição definida na Tabela 1. Na tabela, cada par de sequência identificada como “Pino” e “Sinal” refere-se a estas informações equivalentes. O sinal identificado como “SLAVED” refere-se ao sistema de controle utilizado para o escravo de uma comunicação indicar ao mestre que ele possui novas fichas para transmissão. Já o sinal “SLAVEF” é utilizado pelo escravo também, mas para indicar ao mestre que ele encontra-se cheio e não pode receber novas fichas. A estratégia de distribuição entre os pinos do módulo adotada, foi fortemente inspirada pela distribuição utilizada pelo padrão ATA.² Os sinais de controle foram distribuídos de forma sistemática, localizados próximos aos

Figura 30 – Diagrama representativo da disposição dos pinos da **GPIO** do *kit* de desenvolvimento DE10-Nano. Segundo o protocolo de comunicação, o **GPIO 0** é definido como o módulo de comunicação direito e o **GPIO 1** o módulo esquerdo.

Fonte: Terasic⁶⁰

pinos conectados ao terra dos conectores. E os sinais de dados utilizam a mesma estratégia de distribuição adotada pelo padrão supramencionado. Neste padrão, as vias de dados mais distantes são dispostas lado-a-lado. Como exemplo, os pinos **GPIO_34** e **GPIO_35** são responsáveis por carregar o último e o primeiro *bit* de dados respectivamente, seguindo uma distribuição homogênea e linear destes pinos em diante.

Com isso foi possível obter uma capacidade de transmissão de $7.75Gb/s$ ou $87.08MFichas/s$ em um modo de transmissão *half-duplex*, a partir dos módulos de comunicação de propósito geral de entrada e saída dos *kits* de desenvolvimento. Este valor é apenas teórico, uma vez que o processo de controle teve de ser bastante restrito a fim de garantir essa capacidade no modo de transmissão proposto. Além disso, percebe-se que não existe qualquer cenário em um programa que seja capaz de gerar a quantidade de fichas necessárias para atingir essa capacidade limite de transmissão, principalmente considerando o método utilizado para a distribuição de fichas dentro deste protocolo, que se apresentou bastante homogêneo durante a realização dos testes de validação da máquina.

Tabela 1 – Distribuição dos pinos de comunicação nos módulos **GPIO** de acordo com as definições do protocolo. Para a distribuição dos pinos foi utilizada uma estratégia semelhante ao padrão **ATA**.²

Pino	Sinal	Pino	Sinal	Pino	Sinal
GPIO[0]	CLK	GPIO[12]	DATA[19]	GPIO[24]	DATA[25]
GPIO[1]	DATA[13]	GPIO[13]	DATA[9]	GPIO[25]	DIR[1]
GPIO[2]	DATA[14]	GPIO[14]	DATA[20]	GPIO[26]	DATA[26]
GPIO[3]	DATA[12]	GPIO[15]	DATA[8]	GPIO[27]	SLAVEF
GPIO[4]	DATA[15]	GPIO[16]	DATA[21]	GPIO[28]	DATA[27]
GPIO[5]	DATA[11]	GPIO[17]	DATA[7]	GPIO[29]	DATA[3]
GPIO[6]	DATA[16]	GPIO[18]	DATA[22]	GPIO[30]	DATA[28]
GPIO[7]	DATA[10]	GPIO[19]	DATA[6]	GPIO[31]	DATA[2]
GPIO[8]	DATA[17]	GPIO[20]	DATA[23]	GPIO[32]	DATA[29]
GPIO[9]	DIR[0]	GPIO[21]	DATA[5]	GPIO[33]	DATA[1]
GPIO[10]	DATA[18]	GPIO[22]	DATA[24]	GPIO[34]	DATA[30]
GPIO[11]	SLAVED	GPIO[23]	DATA[4]	GPIO[35]	DATA[0]

Fonte: Elaborada pelo autor.

5.1.1 Primeira Versão

Na primeira tentativa de implementação da solução de comunicação entre os anéis do **Multanel** foi possível verificar uma falha sistemática na operação da solução proposta. Toda a rotina descrita anteriormente foi adicionada na solução. Entretanto, a troca no domínio da frequência acabou se tornando um grande gargalo na operação. A fila utilizada para realizar o sincronismo entre estes domínios leva o período fixo de dois ciclos do relógio, da frequência mais baixa, para efetivar a operação. Este período é verificado desde a adição da informação no *buffer* de entrada do módulo, ainda na frequência mais baixa, até a sua disponibilidade para leitura, já na frequência mais alta.

Diante da adição de uma ficha no módulo de comunicação, a frequência mais alta tinha efeito prático apenas para a transmissão dos dados entre os módulos de comunicação. Apenas uma ficha era adicionada ao módulo de comunicação por ciclo do relógio da máquina. Note que o atraso de dois ciclos da frequência mais baixa acaba promovendo esse comportamento. Uma ficha leva o período de 3 ciclos do relógio para ser transmitida pelo módulo de comunicação, uma vez que ela será repartida em blocos de *31-bits* e é composta por *89-bits*. Entretanto, o módulo de comunicação foi implementado para operação a uma frequência $5\times$ mais alta com relação ao restante dos módulos que compõem cada anel.

Assim sendo, para a realização da transmissão de uma ficha é levado o período de dois ciclos do relógio para sincronismo entre o *TopLevel* do anel corrente e o *buffer* de entrada do módulo de comunicação. Uma vez que a ficha já se encontra na frequência mais alta ela é adicionada imediatamente ao *buffer* de transmissão de fichas. Chegando ao módulo de destino, a ficha já é adicionada imediatamente ao *buffer* de saída do módulo de

Figura 31 – Diagrama representativo da primeira versão proposta para o módulo de comunicação. Uma falha sistemática impedia a capacidade para ganho em desempenho a partir dessa solução.

Fonte: Elaborada pelo autor.

comunicação. Como são necessários 2 ciclos da frequência mais baixa para a realização do sincronismo, então é tomado o período fixo de 6 ciclos dessa frequência para que uma ficha possa ser transmitida completamente, do *TopLevel* de um anel ao outro. Uma visão geral deste protocolo de comunicação pode ser visto na Figura 31.

Na curva de onda exibida na Figura 32 é possível verificar as etapas descritas. As duas primeiras linhas da curva referem-se aos sinais dos relógios presentes na comunicação. Como essa informação foi obtida a partir do processo de simulação, todos os sinais de relógio são sincronizados entre si. A linha identificada como “FIFO_RING” refere-se ao

buffer de alimentação presente no *TopLevel* de cada anel da máquina. A identificação “FIFO_COMM” refere-se ao *buffer* de entrada do módulo de comunicação. Na sequência é possível verificar o sinal de escrita neste *buffer*. A linha “FIFO_OUTPUT” refere-se ao registrador de saída deste *buffer* de alimentação e na sequência seu sinal de leitura. Em seguida, a linha “INPUT_OUTPUT” é o módulo de entrada e saída do *buffer* de três estados que compõe o protocolo de comunicação. As linhas restantes referem-se aos registradores de escrita e leitura do *buffer* de saída do módulo de comunicação do anel vizinho na estrutura da máquina. Observe o sinal de escrita no *buffer* de saída do módulo de comunicação e note que o término do processo depende do sincronismo entre as frequências distintas.

Figura 32 – Curva de onda da interface de comunicação entre os diferentes domínios do relógio, da adição de uma ficha no *buffer* de entrada do módulo de comunicação até sua disponibilidade no anel vizinho.

Fonte: Elaborada pelo autor.

Essa característica acabou se tornando um grande gargalo para a máquina, impedindo-a de apresentar boas curvas de *speedup*. É importante observar que durante o processo de execução de programas, muitas fichas devem ser transmitidas entre os anéis da máquina. Por esta razão, a possibilidade de haver conflito de fichas entre módulos mestres e escravos durante esse processamento é bastante alta. Então, ainda existe este outro foco de atraso para a realização de comunicações. Com isso, foram buscadas alternativas que possibilitassem inicialmente apenas a validação deste método. Até este momento, não havia qualquer indicativo capaz de determinar a viabilidade do **Multianel**.

5.1.2 Validação do Método

Para a validação do método do **Multianel** foi proposta uma comunicação entre os anéis da máquina que é totalmente impraticável atualmente, principalmente considerando os *kits* de desenvolvimento utilizados neste projeto. Para verificar a capacidade dessa estratégia, foi proposta a ligação de todos os dutos de largura de banda de cada anel, em comunicação direta, sem o módulo de comunicação intermediário. Essa comunicação é totalmente impraticável pelo fato de que seria necessária uma quantidade ilimitada de

recursos de comunicação de alto desempenho em todas as placas de desenvolvimento que fossem utilizadas. Afinal, a quantidade de dutos de largura de banda é uma característica parametrizável na π **FlowBW**, ou seja, é possível adicionar quantos dutos de largura de banda forem desejados. Dessa forma, essa implementação intermediária foi proposta apenas para fins de simulação. Essa capacidade de implementação era um dos principais atributos buscados durante a escolha da linguagem de descrição de *hardware* que seria utilizada no decorrer do desenvolvimento do projeto. O suporte para simulação do **BSV** e a facilidade para a adição de novas características ao protótipo segue como uma das melhores escolhas definidas no decorrer deste trabalho.¹⁸

Figura 33 – Diagrama de representação da implementação de validação da π **FlowMR**, com todos os dutos de largura de banda em comunicação imediata aos anéis vizinhos da estrutura.

Fonte: Elaborada pelo autor.

Este método de comunicação entre os anéis pode ser identificado como o limite máximo em desempenho de comunicação que a estratégia do **Multianel** pode alcançar, na topologia proposta, como pode ser visto no diagrama representativo da Figura 33. Com isso, foi possível verificar o que deveria ser o melhor desempenho possível. Dessa forma, seguindo todos os critérios de implementação estabelecidos, foram adicionados *buffers* de

comunicação no *TopLevel* de cada anel, lembrando que foi adicionada uma fila exclusiva para lidar com a comunicação de cada duto de largura de banda da solução. Segundo essa estratégia, diante da ocorrência de uma ficha destinada para comunicação, passando entre a **Unidade de Processamento** e a **Fila de Fichas**, ela é encaminhada imediatamente ao anel vizinho, no próximo ciclo do relógio. A análise do menor caminho para atingir o anel de destino ainda é realizada, evitando com isso qualquer tendência que pudesse promover acúmulo de fichas em determinada direção. Além disso, os dutos de largura de banda ainda promovem uma boa distribuição das fichas em execução, mitigando ainda mais essa tendência.

Foi verificado um baixo aproveitamento das **Unidades Funcionais**, conforme será discutido em detalhes no Capítulo 6. Neste momento, foi lançada a hipótese de que a estrutura de pré-processamento adicionada à solução pudesse estar provocando a subutilização dessas unidades. Afinal, tinham sido adicionadas nesta estrutura algumas das instruções mais utilizadas pelos programas analisados para a validação da máquina. Por esse motivo, para validar a viabilidade do **Multianel**, o pré-processador foi totalmente removido. É importante perceber que a distribuição de fichas por entre os anéis da π FlowMR já é uma estratégia suficiente para remover qualquer acúmulo ou gargalo durante a execução de programas. Apesar da sua estrutura não ser suficiente para lidar com sua capacidade de processamento - afinal, seria necessária uma quantidade ilimitada de **Unidades Funcionais** para garantir essa capacidade - esta solução se mostrou capaz de realizar a exploração de grande quantidade de paralelismo, tanto nas versões anteriores deste projeto, quanto na própria MDFM.^{14,61} Por esse motivo, quanto mais fichas forem capazes de ocupar a utilização das **Unidades Funcionais**, melhor será a capacidade para ganho em *speedup* na máquina.

Com as modificações propostas implementadas, foi possível finalmente verificar curvas de *speedup* realmente promissoras e escaláveis. Com isso, deste momento em diante, a estratégia finalmente pôde sair do plano da hipótese, permitindo a análise real no comportamento dessa estratégia de implementação. Note que para uma configuração com 8 dutos de largura de banda por exemplo, considerando uma hipotética síntese da solução à 50MHz e fichas com comprimento de *89-bits*, a capacidade de comunicação entre os anéis desta estrutura seria de 35,6Gb/s, o que mostra que essa solução é realmente impraticável, principalmente considerando que este número pode aumentar consideravelmente apenas alterando um valor constante na solução, que é a quantidade de dutos de largura de banda. Porém, este número destaca a capacidade de comunicação que essa implementação é capaz de atingir, e por este motivo foi tomado como o limite superior de comunicação para o **Multianel**, a partir da topologia utilizada.

5.1.3 Versão Final

Inspirado pela solução utilizada no método de validação, a versão final do módulo de comunicação também conta com uma estratégia baseada na utilização dos dutos de largura de banda em sua composição. Essa é a principal diferença com relação à implementação inicial. A introdução dos dutos de largura de banda na entrada dos módulos de comunicação permite que mais de uma ficha possa ser sincronizada com a frequência mais alta, dentro do mesmo ciclo do relógio. Com isso, o acúmulo de fichas nessa frequência, garante um aumento considerável na capacidade de transmissão de fichas entre os anéis. Além disso, o módulo de pré-processamento foi reintegrado à estrutura de cada anel com as características descritas anteriormente, na Seção 4.2. Foram mantidas apenas instruções de início e final de programas, além da instrução responsável pela criação de novos Nomes de Ativação. Esta última que foi mantida na **Unidade de Processamento** por ser a responsável pelo gerenciamento deste recurso na arquitetura.

A partir desta implementação, que pode ser vista no diagrama da Figura 34, foi possível verificar ganho em *speedup* equivalente e ainda superiores àqueles obtidos durante a realização da implementação de validação da estratégia. É importante observar, que apesar de um aumento considerável na capacidade de transmissão e recebimento de informações entre os anéis, devido ao aumento na frequência com que as fichas passaram a transitar entre os módulos de comunicação, ainda existe um atraso sistemático na mudança de domínios de frequência, conforme destacado na curva de onda da Figura 32. Ao chegar no anel vizinho, é calculado o duto de largura de banda por onde a ficha recém chegada irá percorrer este anel. De qualquer forma, independente dessa ficha ser destinada ao anel corrente, ou não, ela deve ser encaminhada ao *TopLevel* para a realização do devido direcionamento. Dessa forma, este atraso pode impactar bastante no caminho de uma ficha até que ela possa atingir seu destino definitivo.

Os resultados verificados nesta etapa do desenvolvimento se mostraram ainda melhores que aqueles verificados na versão experimental de validação. Lembre-se que esta versão é descrita como o limite superior de comunicação dentro da topologia prevista por este projeto. Isso levou a verificar a capacidade do **Multianel**, nesta topologia, em promover um efeito de atraso, já verificado na literatura como benéfico para o ganho em desempenho na estrutura da **MDFM**.⁴³ Este atraso garante uma utilização coordenada nos recursos da máquina, impedindo o acúmulo excessivo de fichas transitando entre as unidades que compõem cada anel, apesar de ainda garantir uma boa taxa de aproveitamento das **Unidades Funcionais**. Essa característica destaca o sucesso na escolha da topologia utilizada e mesmo da implementação realizada.

Figura 34 – Diagrama representativo da versão final proposta para o módulo de comunicação. A partir dessa implementação definitiva foi possível verificar um desempenho ainda melhor com relação ao método de validação.

Fonte: Elaborada pelo autor.

5.2 Síntese

Diversos recursos da máquina são removidos da solução na preparação da estrutura destinada para o processo de síntese. Todos os indicadores de utilização da máquina são obtidos apenas no processo de simulação. Devido a utilização dos dutos de largura de banda, é necessária a realização de uma análise dentro de todo ciclo do relógio em busca de conflitos de recursos. Por exemplo, para analisar a máxima utilização da **Fila de Fichas** um registrador é incrementado sempre que uma ficha é armazenada em sua fila de armazenamento interno. Lembre-se que durante a implementação da π FlowBW, esta

fila foi dividida igualmente entre cada duto, cada qual gerenciando seus próprios recursos. Dessa forma, existe a possibilidade de todos os dutos de largura de banda precisarem realizar essa operação, simultaneamente, dentro do mesmo ciclo do relógio. Assim, é imprescindível o uso de um registrador de histórico efêmero para gerenciar essa informação, devido a este conflito de recursos entre os dutos de largura de banda. Todas as unidades possuem analisadores de recursos que tiveram de ser implementados a partir deste elemento de estado primitivo complexo.

Uma das primeiras análises realizadas durante o período de revisão dos caminhos críticos mais longos, Seção 4.1, foi a remoção da maior quantidade de módulos **EHR** do escopo do projeto. Para isso, o recurso da compilação condicional passou a ser utilizado para controlar a disposição destes módulos. A análise destes indicadores foram de fundamental importância para a evolução da implementação. A fim de atingir um estado equivalente à **MDFM** nas primeiras versões descritas anteriormente, Seção 3.1, foi utilizado um dos principais simuladores desta máquina.^{62,63} Este programa foi criado pela equipe de desenvolvimento da **MDFM**, implementado com base nos períodos de execução reais deste protótipo. Ao final da execução de programas neste simulador, são mostrados diversos estados da máquina em um arquivo de saída. O programa foi escrito em versões antigas do **Pascal** e por este motivo foi necessário o uso de sistemas operacionais descontinuados a fim de conseguir realizar sua compilação. O binário do programa é compatível com sistemas mais modernos, a partir do qual foi possível realizar essa análise.

Os indicadores de análise de desempenho foram mantidos na π **FlowMR**. Porém, para fins práticos eles foram utilizados apenas para o processo de simulação. Para a síntese da solução é necessária a adição de uma opção de compilação ao **BSV**, que insere o equivalente a uma variável global à solução, promovendo a remoção de diversos recursos da máquina no processo de compilação. O único indicador remanescente para o processo de síntese é o contador de ciclos do relógio, implementado em um registrador simples, que pode ser analisado a partir de uma operação definida no conjunto de instruções da máquina. Essa instrução é executada a partir do módulo de pré processamento. Sendo, portanto, um recurso gerenciado pela **Unidade de Processamento**. Uma configuração importante que deve ser realizada manualmente é a definição do *TopLevel* da aplicação para o processo de síntese. Lembre-se que para o processo de simulação é definida a descrição do *Testbench* como *TopLevel* do projeto. Entretanto, para a síntese, cada anel deve ser implementado nos *kits* de desenvolvimento. O ambiente de comunicação entre as **FPGAs** será o equivalente ao *Testbench* criado para simulação.

Após a realização da compilação no **BSV**, os arquivos **Verilog** gerados estão prontos para serem adicionados ao projeto no **Quartus**, para o processo de síntese. Neste processo é importante a revisão do arquivo de configurações para a definição das portas de entrada e saída determinadas no projeto principal. Essa definição deve considerar os

pinos dos módulos **GPIO** da **FPGA** conforme definidos na Tabela 1, de acordo com as especificações do *kit* de desenvolvimento.⁶⁰ Para fins práticos foi adicionado um circuito para o controle de um **LED** que irá piscar em uma frequência visível, para indicar a correta implantação da solução na **FPGA**. Para isso, a porta de comunicação com o sistema de **LED** deve estar conectada ao sinal **LEDG**. Um outro sinal importante é identificado como **LEDS**, que tem a finalidade de indicar o estado de cada anel e ele também deve ser conectado nos circuitos de **LED** do *kit*. Além disso, no arquivo de configurações de restrições do projeto, foi fixada a frequência de operação da solução a 50MHz. Por fim, as interfaces de comunicação que conectam a **FPGA** com o **HPS** do *kit* devem ser configuradas pelo menos na solução que irá atuar como anel principal da máquina, em um projeto exclusivo para esta finalidade, que conta com o **GHRD** disponibilizado pela fabricante deste *kit*.

Após a gravação nas **FPGAs** os *kits* de desenvolvimento devem ser conectados corretamente. A placa de expansão criada deve ser conectada apenas no módulo de comunicação mestre. Segundo a convenção instituída, este módulo refere-se ao **GPIO_0** conforme identificado no *kit*. Dessa forma, a **FPGA** disposta à direita no aparato experimental deverá estar conectada no módulo **GPIO_1** do *kit* e não irá utilizar a placa de proteção descrita anteriormente. Isso porque os sinais que serão encaminhados pelo escravo da comunicação ainda passarão pelo mesmo circuito do *kit* vizinho. Perceba que o circuito de proteção tem justamente a finalidade de prevenir excessos de corrente na transmissão de sinais de entrada nos pinos presentes nos módulos **GPIO** das **FPGAs**. Essa solução já é suficiente para garantir isso, pois todos os sinais passarão pelo circuito de proteção de cada pino e via de transmissão dos cabos nos módulos de comunicação.

Com a solução descrita, presente em todos os *kits* de desenvolvimento e os módulos de comunicação devidamente conectados, é possível iniciar a operação da máquina. Cada anel da estrutura se mantém em um estado identificado como “Ocioso”, até o instante em que receber uma ficha em seu módulo de comunicação secundário. Para iniciar o processo de negociação entre os anéis é necessário pressionar o botão de **Reset** no *kit* de desenvolvimento do anel que será definido como principal dentro da estrutura do **Multianel**. Este anel é importante pois será responsável por realizar todo o processo de sincronismo da máquina. Além disso, será a partir desse anel que novos programas poderão ser escritos na máquina. É importante destacar, que ao final da execução de um programa, qualquer outro anel poderá ser definido como principal para uma nova execução, contanto que ele esteja configurado com o **GHRD** do kit de desenvolvimento. Ao término da execução de um programa todos os anéis retornam ao estado “Ocioso” e passarão a aguardar por novas fichas em seu módulo de comunicação secundário. É importante notar que para que a alteração do anel principal tenha efeito, todo o processo de negociação deverá ser realizado novamente. Para isso, basta pressionar o botão de **Reset** no novo *kit* de desenvolvimento que se deseja utilizar como principal.

Ao finalizar a execução de um programa, todos os anéis devem entrar no estado “Ociosos” novamente. Este sinal é importante pois permite identificar erros de execução em programas. Com isso, após a verificação da ocorrência de todas as fichas de saída, é esperado que todos os anéis estejam neste estado específico. Se isso não for verificado, ainda existem fichas no anel que não se encontra neste estado e provavelmente ocorreu um erro de processamento no programa executado. Para identificar esse estado, o sinal **LEDS** do *kit* de desenvolvimento deve ser analisado, de acordo com sua identificação no tipo de dado **Enum**, utilizado para caracterizar os estados de cada anel.

5.3 Operação

Após a realização do processo de negociação, os anéis entram no estado identificado como “Pronto”. Neste estado, os anéis ficam aguardando pelas fichas de programas, para escrita dessa informação na memória de armazenamento da **Unidade de Instruções** de cada anel. Na π **FlowMR** esta operação é realizada por meio do módulo **HPS** do **SoC** presente no *kit* de desenvolvimento. As placas DE10-Nano são distribuídas com um cartão **MicroSD** contendo uma distribuição Linux *Angstrom* com uma interface **XFCE**, baseada na versão 4.1.33 do *kernel* deste sistema operacional. Esta versão do Linux é utilizada especialmente para a configuração de sistemas embarcados. Para fins práticos, foi utilizado um sistema sem a interface gráfica padrão, também disponibilizado pela fabricante do *kit* de desenvolvimento. Esta distribuição trata exatamente da mesma versão do *kernel* do Linux, porém a omissão dos recursos gráficos são importantes para a operação do **GHRD**, uma vez que diversos circuitos podem ser removidos da **FPGA**, permitindo o uso de mais elementos lógicos para a operação do sistema principal da π **FlowMR**. O módulo **HPS** possui todo suporte necessário para operar como um processador dedicado, além de contar com interfaces de comunicação de alto desempenho com a **FPGA** do **SoC**. Por esse motivo, este módulo foi escolhido para realizar a interface de entrada e saída de programas, para comunicação com o protótipo.

Durante o processo de negociação, simultaneamente com a distribuição de fichas com a identificação de cada anel, é realizado o processo de negociação entre o anel principal e seu módulo **HPS**. Um programa escrito em **C** é responsável por realizar toda a comunicação entre a **FPGA** e este módulo. É importante que este programa esteja em execução antes de iniciar o processo de negociação entre os anéis da máquina. Ele tem a finalidade de habilitar a ponte de comunicação entre a **FPGA** e o processador embarcado no *kit* de desenvolvimento. Essa operação é realizada em um registrador especial presente no espaço de memória de usuário do módulo **HPS**. Com isso, no mesmo ciclo do relógio que o anel principal faz a criação da ficha de negociação, ele ainda encaminha uma ficha de comunicação para o módulo **HPS**. Este módulo responde com o incremento simples no valor desta ficha para atestar o correto funcionamento da comunicação.

Ao final do processo de negociação, o anel principal encaminha uma nova ficha ao módulo **HPS** que deve iniciar a transmissão do programa que será executado pela máquina. As informações são encaminhadas já no formato binário que será escrito diretamente na memória da **Unidade de Instruções**. Essas fichas são encaminhadas a essa unidade e uma cópia é encaminhada ao anel vizinho na estrutura da máquina. Após a transmissão de todos os nós do programa, o módulo **HPS** encaminha uma ficha com um tipo de dado específico para indicar o término da transmissão. Essa operação coloca o anel principal em espera para garantir que todos os anéis já receberam todas as instruções do programa. Ao receber a ficha de término do programa do seu módulo à esquerda, o anel principal realiza uma nova comunicação com o módulo **HPS** para indicar que a máquina está pronta para execução. O módulo finalmente começa a transmissão das fichas de entrada do programa e a máquina inicia o processamento imediatamente. Todas as instruções de saída de programas do conjunto de instruções da **MDFM** foram mantidas na π **FlowMR**. Qualquer anel que identificar fichas com essa característica as encaminham para o anel principal, que se encarregará de transmiti-las ao módulo **HPS** para apresentação ao usuário final. É prevista a geração das fichas de erros a partir desta implementação, conforme previsto pelo manual da **MDFM**.¹⁶ Entretanto, este tratamento de erros durante o processamento de programas não foi implementado até o momento da escrita desta tese.

O programa escrito em **C** tem a finalidade apenas de realizar o sincronismo descrito. A partir dos registradores **GPV** presentes no módulo **HPS**, essa operação é alcançada de forma bastante facilitada. O endereço de acesso a estes registradores podem ser gerados automaticamente em uma biblioteca de cabeçalhos, a partir de um assistente de compilação disponibilizado como ferramenta de implementação para dispositivos *Cyclone V* pela Intel. Como o sistema operacional nativo do módulo **HPS** trata-se de uma distribuição Linux convencional, foi possível configurar uma interface para acesso remoto a partir do protocolo **SSH**. Dessa forma, é possível deixar a máquina em operação, com o processo de negociação corretamente configurado e realizar a execução de programas de forma remota. O programa de interface para acesso a máquina faz o uso de assistentes escritos em **Python** responsáveis pela tradução dos programas em linguagem montadora da **MDFM** para o formato binário característico da π **Flow**.¹⁸

Existem diversos recursos que permitem a transmissão de informações entre a **FPGA** e o módulo **HPS** nos **SoCs** *Cyclone V* da Intel. Normalmente estes recursos são utilizados para permitir o acesso aos periféricos disponíveis para operação do processador, a partir da **FPGA**. Dessa forma, é possível ter acesso aos recursos assistentes como os controladores **HDMI**, **Ethernet**, **I²C**, memória **RAM**, dentre outros. Porém, foi escolhido o uso dos módulos de comunicação dedicados que existem para a realização dessa transmissão. Dentre os recursos disponíveis, existe uma ponte de comunicação bidirecional compartilhada, que é limitada à uma largura de *32-bits*, que inclusive abriga os registradores necessários para a ativação das outras portas dedicadas. Estas pontes exclusivas possuem

Figura 35 – Diagrama representativo das pontes de comunicação **AXI** no contexto dos pinos de entrada e saída da **FPGA** e a interconexão **L3** do módulo **HPS**.

Fonte: Terasic⁶⁰

capacidade de até *128-bits* de transmissão entre a **FPGA** e o **HPS** dedicadas em cada sentido da comunicação. Dentro do módulo **HPS** estes sistemas de comunicação podem ser acessados a partir de endereços exclusivos na memória do usuário do sistema operacional, conforme a configuração do projeto no **Quartus**, a partir do **GHRD**. O diagrama da Figura 35 apresenta essas pontes de comunicação. Estes módulos possuem o modelo de mestre (M) e escravo (S) que são destacados no diagrama, além dos domínios de frequência de cada interconexão.

6 RESULTADOS

Neste capítulo serão dispostos os principais resultados de desempenho da arquitetura proposta nos Capítulos 3, 4 e 5. É importante destacar que todos os resultados apresentados neste capítulo foram obtidos a partir do processo de simulação do protótipo, uma vez que apenas neste formato que foi possível verificar todos os critérios utilizados para avaliação de desempenho da máquina. Na sequência serão definidas as principais métricas utilizadas no processo de análise, bem como os principais programas utilizados para validação. Em seguida serão apresentados os principais resultados e discussões acerca dos objetivos alcançados a partir da solução aqui proposta.

6.1 Métricas de Análise

Para avaliar o desempenho de algoritmos e arquiteturas paralelas, existem diversas estratégias bem estabelecidas na literatura.^{14,45,47,61,64-66} Entretanto, em geral, essas métricas tem a finalidade principal de avaliar o desempenho de algoritmos dentro de arquiteturas paralelas distintas.⁶⁴⁻⁶⁶ Como o intuito principal deste projeto é a análise do desempenho da solução implementada, foram utilizadas principalmente as métricas mais tradicionais para avaliação de modelos clássicos em fluxo de dados, em busca de verificar a capacidade da máquina para ganho em desempenho de acordo com a simples adição de elementos de processamento. Em grande parte, essa escolha se deu pela facilidade na capacidade desta implementação em permitir certa parametrização em sua estrutura, a fim de verificar os melhores resultados possíveis de serem obtidos a partir desta arquitetura. Além disso, para a realização desta análise, diferentemente de outros autores,^{64,66-70} o tamanho dos programas foram mantidos constantes*, permitindo a avaliação do ganho em desempenho real da arquitetura, diante apenas da adição de elementos de processamento. Portanto, não é o objetivo deste trabalho buscar por valores ótimos de *speedup* para a execução de algoritmos, mas avaliar o desempenho da máquina construída a partir de programas estáticos e com uma estrutura bem definida para a exploração de paralelismo.

6.1.1 *Speedup*

Uma das principais métricas utilizadas para avaliar o ganho em desempenho em arquiteturas paralelas é o *speedup*. Esta métrica tem o objetivo de verificar o comportamento da estrutura avaliada de acordo com a adição de mais recursos de processamento paralelo. Para isso, é analisado o tempo de execução T_P de um programa na arquitetura com uma

* Principalmente pelas características dos programas utilizados para análise, conforme será descrito em detalhes na próxima seção.

quantidade N de unidades de processamento, dividido pelo tempo de execução do mesmo programa com apenas uma unidade, T_S , como pode ser visto na Equação 6.1. De acordo com essa métrica, é possível verificar em detalhes o ganho em desempenho obtido a partir da configuração paralela avaliada. Em processamento paralelo, para que o *speedup* seja considerado ideal, é esperado um aumento linear em desempenho em função do número de elementos processadores. Ou seja, com a utilização de duas unidades paralelas, é esperado o dobro em desempenho, para três unidades, o triplo em desempenho e assim sucessivamente. Essa métrica tem a finalidade principal de avaliar a capacidade de escalabilidade de uma estrutura paralela. A fim de determinar o tempo de processamento de determinado programa, foi utilizado o registrador de contagem de ciclos presente na composição da unidade de processamento, descrito em detalhes no Capítulo 5. Dessa forma, a contagem da quantidade de ciclos tomados para a execução de um programa é totalmente precisa, uma vez que não existe nenhum tipo de escalonamento entre processos, por exemplo, e a execução de um programa é realizada imediatamente, em sua totalidade, diante da entrada de todos os recursos necessários para fazê-lo.

$$S = \frac{T_P}{T_S} \quad (6.1)$$

Alguns autores tomam o *speedup* como função de seu melhor desempenho possível em execução sequencial, mesmo que realizado por outra estrutura, seja paralela ou não.^{64,65,68-70} Entretanto, devido à facilidade de modificação dos elementos que compõem a arquitetura da máquina, essa métrica foi avaliada com relação ao seu próprio desempenho, utilizando uma única unidade funcional. O uso desta métrica a partir desta estratégia foi proposto a fim de verificar o ganho real em desempenho com a adição de novas unidades de processamento, ainda que a execução sequencial tenda a apresentar um baixo desempenho, devido aos fatores limitantes da arquitetura para este tipo de execução, quando comparado à uma máquina de *Von Neumann*. Afinal, as fichas devem enfrentar todos os gargalos naturais da arquitetura para o resgate de todas as informações necessárias para a execução de suas instruções. O baixo desempenho nesta execução foi discutido e exposto na descrição da primeira versão da implementação, Seção 3.1. Apesar da verificação deste baixo desempenho, ela possibilita a análise detalhada do impacto na adição de mais recursos paralelos na construção da máquina, testando a escalabilidade da proposta. Neste sentido, foi promovido o uso de dutos de largura de banda para garantir o melhor desempenho possível da máquina, mesmo nesta configuração.

Além disso, a utilização da própria estrutura da máquina em sua análise sequencial ainda possibilita a remoção de quaisquer gargalos, atrasos ou até mesmo vantagens para a realização da análise[†]. Afinal, todas essas propriedades estarão presentes em ambas as versões consideradas para o levantamento desta métrica, que é composta essencialmente

[†] Como por exemplo máquinas capazes de apresentar um comportamento superescalar⁷¹

por uma operação de divisão e portanto, suficiente para anular estas eventuais propriedades. Esta avaliação permite a verificação apenas do aumento na complexidade da arquitetura, de acordo com a adição de mais recursos computacionais para sua execução paralela, além da capacidade da máquina para exploração de paralelismo. Por este motivo, a versão sequencial da máquina explora exatamente todos os mesmos recursos de sua versão paralela, inclusive utilizando múltiplos dutos de largura de banda. Para o levantamento desta métrica, a partir da realização de várias execuções dos mesmos programas em diversas configurações da máquina, foram tomados os valores que apresentaram melhores resultados.

Diversos fatores impedem uma arquitetura paralela de alcançar um *speedup* ideal. Na presente proposta, a passagem de fichas pelas unidades que compõem a estrutura de cada anel da máquina passou a ser apenas parte do gargalo que impede essa capacidade. É importante considerar ainda a necessidade de transmissão de fichas entre os anéis da estrutura do **Multianel**, que acaba promovendo ainda mais gargalo. A existência de fichas que dependem de pareamento com eventuais fichas que podem ainda nem ter sido geradas (considerando ainda sua geração por um outro anel desta estrutura), atenuam ainda mais este problema. O *speedup* poderia ser ideal, se todas as instruções pudessem ser executadas imediatamente após a sua criação. Além disso, seria necessária uma quantidade ilimitada de elementos de processamento capazes de consumir quantas instruções fossem necessárias, paralelamente. Devido a todas essas considerações, esta métrica acaba atingindo um limite superior para ganho em desempenho, de forma que a partir de determinada quantidade de recursos computacionais não é possível verificar qualquer variação de desempenho de acordo com a adição de mais recursos de processamento, em um regime estacionário assintótico.

Note que uma máquina capaz de atingir essa capacidade trata de uma aproximação totalmente idealizada. Entretanto, foi buscada a capacidade máxima para exploração das unidades funcionais que compõem a estrutura da máquina. É importante destacar que o *speedup* ideal também pode ser atingido caso todas as unidades funcionais presentes na composição da máquina se mantiverem ocupadas durante todo o tempo de processamento de um programa. Se não houver ociosidade na utilização destas unidades, o desempenho da máquina será o máximo possível, de acordo com o máximo desempenho de cada uma delas, independente da quantidade disponível. Afinal, devido a construção desta métrica, quaisquer gargalos verificados durante o processamento estarão presentes em ambas as versões da análise, e portanto, serão anuladas naturalmente, principalmente ao considerar o formato utilizado para esta verificação no presente projeto, onde foram utilizados apenas programas de mesmo tamanho e a comparação ser realizada a partir da mesma estrutura, apenas com menos recursos de processamento. Mesmo neste cenário ainda existem muitos instantes quando ocorrem desperdícios de recursos, principalmente no início e final de programas, quando ainda não existe qualquer capacidade de paralelismo disponível, segundo a implementação dos programas.

O máximo aproveitamento das unidades funcionais foi buscado através da implementação das melhorias da π **FlowBW**, descritas na Seção 4.2, tais como a introdução de instruções capazes de manter as unidades de processamento ocupadas por mais tempo, permitindo a melhor distribuição possível entre essas unidades. Entretanto, de acordo com a adição de ainda mais recursos na estrutura da máquina, principalmente no contexto do **Multianel**, começam a surgir ociosidades na distribuição e o ganho em desempenho acaba por atingir um limite superior. Alguns autores sugerem um aumento constante no tamanho do programa em execução, para que sua complexidade seja capaz de acompanhar o acréscimo de recursos na estrutura da máquina, a fim de obter um melhor *speedup*. Entretanto, conforme destacado anteriormente, neste projeto, o uso desta métrica tem a finalidade apenas de verificar a capacidade de escalabilidade da máquina e não buscar por valores ótimos em *speedup*.

É importante reforçar ainda que em algumas análises realizadas, foi utilizado um conceito amplo de *speedup* na verificação do ganho em desempenho promovido pelo **Multianel**. Isso porque foi extrapolada a analogia desta composição da máquina com relação a estrutura de anel único. Conforme definido anteriormente, o *speedup* tem a finalidade de avaliar o comportamento da máquina em uma composição paralela, em comparação com seu desempenho sequencial. Para a análise do comportamento do **Multianel**, foi verificado o desempenho da máquina com N anéis em sua estrutura, em comparação com a mesma composição paralela da máquina (considerando todas as suas capacidades de quantidades de unidades funcionais e dutos de largura de banda), utilizando um único anel. Dessa forma, são levadas em consideração duas estruturas paralelas e não mais sua versão puramente sequencial. Com isso, é possível verificar a capacidade de escalabilidade do **Multianel**, apesar de se distanciar conceitualmente da definição desta métrica. Note que este formato pode inclusive se tornar desfavorável para a verificação do ganho em desempenho desta configuração, uma vez que nesta análise sua versão “sequencial” já se aproveita de recursos paralelos, e portanto, já apresenta um melhor desempenho em comparação com sua versão estritamente sequencial.

6.1.2 Paralelismo Médio

Além do *speedup*, outra métrica bastante utilizada para a avaliação de desempenho de máquinas clássicas baseadas no modelo a fluxo de dados dinâmico, refere-se exclusivamente à implementação do algoritmo e tamanho dos programas utilizados. Essa métrica é conhecida como paralelismo médio, e tem a finalidade de avaliar a capacidade de paralelismo do programa em máquinas baseadas neste modelo computacional. O paralelismo médio é obtido a partir da divisão entre a quantidade total de instruções presentes na árvore de representação do programa paralelo, que será definido como S_0 , com relação à altura desta árvore, aqui representada como S_∞ , conforme a disposição apresentada na

Equação 6.2. O valor de S_∞ tem a finalidade de evidenciar a quantidade de instruções do algoritmo que podem ser executadas de forma estritamente paralela, considerando a dependência de dados entre instruções. Dessa forma, esta métrica apresenta o limite máximo de desempenho que um algoritmo paralelo é capaz de atingir, ou seja, considerando uma quantidade ilimitada de recursos paralelos, capazes de consumir todas as instruções possíveis, simultaneamente. O valor de S_∞ teve de ser implementado na estrutura das fichas, Equação 5.1, tendo seu valor apenas incrementado durante a execução dos programas, dentro da unidade de processamento, para fins de medição apenas, no processo de simulação da máquina. Já o valor de S_0 é igual ao número total de instruções presentes no código do programa, com seu grafo de representação em fluxo de dados totalmente expandido.

$$\pi = \frac{S_0}{S_\infty} \quad (6.2)$$

Por possuir este caráter idealizado, essa métrica é capaz de determinar o *speedup* assintótico ótimo de um programa. No valor de S_∞ é considerado um cenário onde todas as instruções que podem ser executadas paralelamente, serão executadas imediatamente, sem qualquer atraso, bem como em sua versão sequencial, no valor de S_0 , que considera apenas cada nó do grafo de representação do programa em fluxo de dados. Dessa forma, o valor obtido a partir dessa métrica é capaz de mostrar o maior *speedup* possível de ser obtido para a execução de determinado programa. Obviamente, conforme destacado anteriormente, devido a fatores limitantes de qualquer arquitetura paralela, este valor não é possível de ser alcançado, tratando-se apenas de uma aproximação ótima deste valor.

Conforme será destacado na sequência, os programas utilizados para a validação da máquina foram implementados de forma a permitir uma clara definição de seu tamanho. Apesar do código possuir características estáticas, com uma quantidade fixa de instruções, as propriedades dinâmicas da arquitetura permitem que sua árvore de representação possa crescer a partir de um valor bem definido. É possível verificar na análise dos resultados que o tamanho do programa tem uma influência imediata com a capacidade da arquitetura para ganho em desempenho paralelo, fato este totalmente consistente com o encontrado na literatura.^{64,68–70} Por este motivo, essa métrica foi analisada, a fim de verificar a relação entre a variação do *speedup* de acordo com a quantidade de paralelismo presente nos programas analisados no processo de validação da máquina.

A partir da análise desta métrica, é importante reforçar ainda que o comportamento assintótico não está limitado exclusivamente à fatores relacionados a arquitetura, mas também à implementação do programa. Essa característica é encontrada devido a ocorrência de dependências de dados, intrínseca à definição do modelo a fluxo de dados, que acaba por introduzir um caráter sequencial natural a este modelo de execução. Se este fator não fosse verificado, o grafo de representação bidimensional deste modelo não teria o formato de

uma árvore binária (como verificado nos principais programas de validação), mas de uma árvore horizontal, desbalanceada. Dessa forma, a altura desta árvore de representação é suficiente para expor esta dependência de dados para a execução de programas, e portanto, a presença deste caráter sequencial intrínseco ao modelo.

6.2 Programas

Para a validação do correto funcionamento do conjunto de instruções da máquina, foram criados diversos micro *benchmarks* que tinham a finalidade de avaliar apenas um pequeno grupo de instruções, conforme o programa destacado anteriormente, Seção 2.3. Estes programas eram extremamente metodológicos e buscavam apenas atestar o correto funcionamento de determinadas instruções, não contando por exemplo, com as propriedades dinâmicas da estrutura baseada no modelo a fluxo de dados dinâmico. Além disso, foram identificados alguns trabalhos na literatura^{14,72} que permitiram a extração de alguns programas, que apesar de explorarem as capacidades dinâmicas da arquitetura, ainda eram simples o suficiente para que não pudessem ser tratados para a aplicação das métricas de análise descritas anteriormente.

Para a análise de desempenho da máquina foram utilizados dois programas principais, implementados ainda para a estrutura da **MDFM** e compilados a partir da solução criada especificamente para este protótipo. Estes programas foram escritos ainda nos anos 80, obtidos já no formato em linguagem montadora da máquina, o que permitiu sua utilização dentro da estrutura da π **FlowMR**. Infelizmente não foi possível obter uma cópia do compilador da **MDFM** em tempo hábil, durante o desenvolvimento deste projeto. Diversas tentativas de recuperação deste compilador foram buscadas em diversas fontes, porém, aparentemente, o sistema de compilação da **MDFM** acabou se perdendo ao longo deste período. Por este motivo a análise ficou limitada apenas a estes programas já compilados e testados dentro da estrutura original. Apesar de tratar-se de programas simples, eles possuem grande potencial para a exploração de um alto grau de paralelismo. Deve-se ressaltar que a elaboração de um sistema de compilação para a π **FlowMR** foge do escopo deste trabalho, sendo deixada como proposta de trabalhos futuros.

Ambos os programas foram implementados de forma duplamente recursiva, ou seja, cada nível de recursividade destes programas são expandidos em outras duas chamadas da mesma função. Portanto, estes programas, em tempo de execução, crescem de forma bastante “comportada” e de maneira previsível, fornecendo funções bastante homogêneas em cada nível de sua recursividade. Além disso, é importante observar que eles são expandidos no formato de uma árvore binária, garantindo ainda mais homogeneidade em sua composição. Estes programas serão descritos em detalhes a seguir e sua implementação em linguagem de alto nível pode ser vista no Anexo A.1, bem como sua implementação

em linguagem montadora, apresentada no Anexo A.2.

6.2.1 Fatorial Modificado

Este programa calcula o fatorial de um número definido em “tempo de compilação”. Apesar de não se possuir o compilador da máquina, uma instrução específica do código do programa em linguagem montadora permite a modificação deste valor de referência. Esta instrução está presente no código principal do programa descrito em **Sisal**, como pode ser visto no Anexo A.1, e portanto é determinada antes do início das chamadas recursivas da sua implementação. Assim, esta instrução tem a finalidade de determinar a condição de parada destas chamadas. Essa característica permitiu a análise de diversos tamanhos do grafo de representação em fluxo de dados deste programa, permitindo uma exploração maior de paralelismo.

Como o resultado da operação do fatorial cresce de forma muito acelerada, foi proposta uma modificação na funcionalidade deste programa. Ao invés de realizar a operação de multiplicação, padrão no cálculo do fatorial, é calculada a soma entre os resultados de cada iteração na sua evolução. Isso porque não foram implementadas instruções capazes de promover a representação de números muito grandes. A arquitetura desta máquina é limitada a números com tamanho máximo de *32-bits*[‡]. Dessa forma, passou a ser possível a expansão no tamanho do programa, sem danos à realização da operação calculada por ele. Perceba que com essa modificação, o programa passa a realizar uma soma de **Gauss**, definida conforme a expressão a seguir, Equação 6.3, onde n é o valor de referência que pode ser variado no código do programa em linguagem montadora.

$$S_n = \frac{n \times (n + 1)}{2} \quad (6.3)$$

Conforme visto no Anexo A.1 o programa é bastante simples, mas apresenta grande potencial de paralelismo para valores de referência n suficientemente grandes. Cada iteração do programa realiza duas novas chamadas recursivas. Dessa forma, a árvore de representação do programa em fluxo de dados é expandida no formato de uma árvore binária e sua altura é determinada de acordo com o valor de referência destacado anteriormente, conforme mostrado na Figura 36. Entretanto, por se tratar da condição de parada do programa, este valor tem a finalidade de determinar apenas a quantidade de iterações que serão realizadas. Por este motivo, para valores que sejam uma potência de dois, tem-se uma árvore binária perfeitamente balanceada e sua distribuição é mais homogênea na estrutura da máquina. Note que a altura da árvore é bem definida de acordo com este valor de referência pois, para o cálculo do “fatorial” de 1023 por exemplo, a altura da árvore

[‡] Como apresentado na Equação 5.1, o dado de uma ficha possui o tamanho de *37-bits*, dos quais *5-bits* são utilizados para descrever o tipo de dado carregado pela ficha.

será de apenas 10 níveis. Perceba que sua expansão é feita de forma bastante rápida, por ser realizada neste formato duplamente recursivo. Este fator tem um impacto essencial sobre seu desempenho nesta arquitetura, afinal, pouco recurso é desperdiçado no início do programa, enquanto sua árvore de representação ainda não apresenta muito paralelismo.

Figura 36 – Diagrama de representação sobre o formato de expansão do grafo em fluxo de dados dos programas utilizados para validação da máquina, que neste modelo é representado por uma árvore binária de altura definida pelo valor de referência, que pode ser manipulado em instruções específicas dos programas.

Fonte: Elaborada pelo autor.

6.2.2 Integração binária

O outro programa utilizado para validação da máquina possui características bastante semelhantes ao programa do fatorial modificado. Da mesma maneira ele foi implementado de forma duplamente recursiva e portanto, sua expansão em fluxo de dados conta com as mesmas vantagens para a exploração de paralelismo. Ele realiza o cálculo da integração da função descrita na Equação 6.4. Sua implementação utiliza o método do trapézio, onde a curva de referência é dividida igualmente, em trapézios de mesma largura. Este programa recebe como entrada os limites de integração a e b , conforme destacados na expressão, sob a função de referência, bem como a quantidade de divisões que serão realizadas neste intervalo, segundo o método da implementação. De acordo com sua definição, a quantidade de divisões da curva determina a altura do grafo de

representação do programa em fluxo de dados, conforme destacado na Figura 36. De forma equivalente ao programa anterior, o grafo em tempo de execução deste programa também possui o formato de uma árvore binária, devido ao seu caráter duplamente recursivo.

$$f(x) = \int_a^b (x^2 + 6x + 10) dx \quad (6.4)$$

Diferentemente do programa do fatorial, a altura da árvore de representação deste programa é definida de forma explícita. Ou seja, sua condição de parada é realizada a partir da altura da árvore e não pela quantidade de iterações a serem realizadas. Dessa forma, para a execução do programa com este valor igual a 10, por exemplo, serão realizadas 1024 chamadas recursivas da função destinada a efetuar o cálculo da área do trapézio em sua subdivisão específica. Dessa forma, a quantidade de paralelismo explorado por este programa é definida de forma mais explícita, apesar de mais limitada. Quanto mais divisões nesta curva, maior será a capacidade de paralelismo a ser explorado dentro da estrutura da máquina, bem como a precisão para a realização do cálculo da integração. Esta “profundidade” (como é definido no escopo do programa, Anexo A.1), está relacionada ao tamanho do programa, em capacidade de exploração de paralelismo. É importante reforçar ainda que a representação no grafo em fluxo de dados deste programa possui o formato de uma árvore binária balanceada, necessariamente.

Devido aos resultados cada vez mais satisfatórios para a exploração das instruções que operam sob números em ponto flutuante, ainda foram realizadas algumas modificações neste programa. A primeira delas se deu apenas nos valores calculados por ele, que passaram a ser realizados sob números em ponto flutuante. Esta modificação permitiu a análise de cálculos muito mais precisos da integração em intervalos menores da curva analisada. Mas, principalmente, foi possível verificar um grande ganho em desempenho com a adição de mais unidades de processamento. Conforme descrito anteriormente, as operações sob ponto flutuante levam um período de tempo maior para serem executadas completamente, fator este que promove um aumento considerável do uso das unidades funcionais durante a execução do programa. Portanto, como será visto, é possível se aproximar ainda mais do comportamento ideal do *speedup*, nesta arquitetura.

Outra modificação deste programa que promoveu um aumento considerável na sua complexidade e distribuição, foi a introdução de operações trigonométricas. Conforme a implementação do algoritmo **CORDIC**, descrita em detalhes no Apêndice A, é necessário o período fixo de 32 ciclos do relógio para a execução completa de cada operação. Dessa forma, cada unidade funcional alocada para execução destas instruções obtém uma grande taxa de utilização durante a execução do programa. Com operações trigonométricas, a execução do programa fica bem mais lenta, o que torna o programa mais realista e deve levar a um *speedup* bem melhor, já que as unidades funcionais tendem a ficar menos ociosas. Nos testes foi utilizada a função dada pela Equação 6.5. Como não se tem acesso a um

sistema de compilação, o grafo do programa teve que ser implementado diretamente em linguagem montadora.

$$f(x) = \int_a^b (x^2 + 6\sin(x)\cos(x) + 10) dx \quad (6.5)$$

Na Tabela 2 são apresentados os valores do paralelismo médio de ambos os principais programas utilizados para validação da máquina. Para o programa de integração, dois casos foram considerados, pois a adição das operações para o uso do módulo **CORDIC**, influenciaram na quantidade de instruções dentro de cada nível de paralelismo e, com isso, o valor do paralelismo médio apresentou uma pequena variação. Entretanto, conforme será destacado em detalhes na análise dos resultados seguintes, apesar de apresentar apenas uma pequena variação no valor do paralelismo médio, o *speedup* obtido neste programa é bastante superior quando comparado ao programa que realiza o cálculo sobre valores inteiros apenas, ou mesmo do programa que opera sobre valores em ponto flutuante. Isso ocorre devido a definição das operações trigonométricas promovidas pela implementação do algoritmo do **CORDIC**. Estas operações levam uma grande quantidade de ciclos do relógio para serem executadas, aparentemente garantindo um melhor aproveitamento das unidades funcionais alocadas para a execução dessas instruções particulares. Ou seja, apesar de apresentar apenas um pequeno aumento na árvore de representação do programa, conforme pode ser visto na tabela do paralelismo médio, a “complexidade” do programa aumenta de forma não constante, devido a presença das operações trigonométricas. Além disso, está disposta na tabela, na coluna **BY**, a quantidade total de instruções de passagem direta de cada versão dos programas.

Tabela 2 – Valores de S_0 , S_∞ , paralelismo médio π e a quantidade total de instruções de passagem direta de cada um dos micro *benchmarks* utilizados para validação da arquitetura, para diferentes tamanhos destes programas.

	Tamanho	S_1	S_∞	π	BY
Fatorial modificado	500	14485	118	123	6994
	1000	28985	130	223	13994
	1500	43485	142	306	20994
Integração	512	11759	96	122	5881
	1024	23535	106	222	11769
	2048	47087	116	406	23545
Integração Trigonométrica	512	12783	99	129	6649
	1024	25583	109	235	13305
	2048	51183	119	430	26617

Fonte: Elaborada pelo autor.

Foram buscadas alternativas que permitissem operar sob valores em ponto flutuante no programa do fatorial modificado, mas sem sucesso. Afinal, a implementação deste programa permitiria a adição de apenas uma instrução com essa característica, que é a

operação de soma, que substitui a multiplicação do fatorial, não oferecendo mudanças significativas no padrão de *speedup* verificado no programa em sua versão operando sob valores inteiros. É importante reforçar também, que no programa de integração operando sob valores em ponto flutuante, não existe qualquer modificação de suas propriedades, conforme mostrado na Tabela 2. Foram realizadas alterações apenas no tipo de instruções destinadas para o cálculo das operações da integração, de forma que, ao invés de realizar operações em valores inteiros, passaram a ser utilizadas instruções que operam sob valores em ponto flutuante. Apesar de não apresentar mudanças no perfil do grafo de representação do programa, e portanto nos dados da tabela anterior, esta modificação apresenta uma variação não constante, devido ao caráter das operações em ponto flutuante, que possuem maior “complexidade” durante sua execução, levando o período de quatro ciclos do relógio para serem executadas completamente, além do período para o tratamento de cada ficha nas unidades funcionais. Dessa forma, apesar de não haver modificações em seus valores de referência, o ganho em desempenho verificado neste programa difere de sua versão que opera sob valores inteiros.

6.3 π Flow

A seguir, serão descritos e discutidos sucintamente, os principais resultados obtidos ao término da implementação da primeira versão da arquitetura π Flow, apresentados na dissertação de mestrado do autor deste trabalho. Estes resultados serão importantes para a discussão dos resultados desta tese e, assim, serão apresentados brevemente nesta seção. Os resultados obtidos nesta versão do protótipo ficaram muito abaixo das expectativas. Entretanto, esta característica já era esperada a priori, devido ao formato de implementação adotado para a π Flow. Afinal, diferentemente da MDFM, a π Flow possui todas as unidades sincronizadas sobre o mesmo domínio de frequência. Dessa forma, todo o tempo de vida de uma ficha dentro da estrutura da máquina passa a se tornar um grande gargalo da arquitetura, pois o número de ciclos que uma ficha leva para circular o anel, que pode ser considerado como um *overhead*, é maior que aquele gasto no processamento desta ficha dentro das unidades funcionais.

O gráfico da Figura 37 refere-se a execução do programa de integração binária em todas as suas versões descritas anteriormente, e destaca o baixo aproveitamento das unidades funcionais nesta abordagem. Foram utilizados os limites de integração entre 0 e 1 § e profundidade da árvore de representação em fluxo de dados igual a 10, o que leva a uma expansão de 1024 chamadas da função principal deste programa. Não foram usadas mais unidades funcionais, pois claramente o *speedup* já se encontra em seu comportamento assintótico e estacionário. Perceba tratar-se de programas que em

§ A menos de sua versão que opera apenas sobre valores inteiros, que teve limites de integração entre 0 e 1024, a fim de obter um resultado de execução mais próximo do esperado.

Figura 37 – Curva de *Speedup* na π **Flow**, para o programa de Integração, Seção 6.2, em todas as suas versões disponíveis: valores inteiros, valores em ponto flutuante e com funções trigonométricas.

Fonte: Elaborada pelo autor.

todas as versões destacadas apresentam valores de paralelismo médio superiores a 200. Entretanto, verifica-se que o comportamento assintótico é encontrado para valores inferiores a 4. Ou seja, menos de 2% das instruções destes programas, que podem ser executadas paralelamente, efetivamente o são. Apesar do programa apresentar grande capacidade de paralelismo, a arquitetura não permite sua exploração.

Na Tabela 3 podem ser vistos os valores utilizados para a construção do gráfico da Figura 37. Nesta tabela as colunas **MTQ** e **MMU** referem-se à máxima utilização dos componentes de armazenamento interno da fila de fichas e da unidade de pareamento, respectivamente. Os valores destacados nesta tabela são relacionados à execução do programa de integração que executa operações trigonométricas, curva mais alta mostrada na representação gráfica. Uma característica importante que pode ser analisada nesta tabela é o baixo aproveitamento das unidades funcionais, apesar desta versão do protótipo se aproveitar de uma estrutura circular para a realização de uma distribuição mais equilibrada de fichas entre estas unidades. Além deste programa contar com diversas instruções em ponto flutuante e operações trigonométricas, que levam mais ciclos do relógio para serem concluídas, note que a porcentagem de uso possui valores maiores quando a curva encontra-se em evolução dinâmica, para quantidades até 3 unidades funcionais, reforçando que

Tabela 3 – Valores utilizados para a determinação do gráfico que compõe a Figura 37. Todos os valores presentes nesta tabela referem-se à última curva do gráfico, para o programa de integração utilizando funções trigonométricas. Este programa possui um total de 51183 instruções e valor de $\pi = 234.7$.

FU	MTQ	MMU	Ciclos	<i>Speedup</i>	%FUs
1	3698	3707	450442	1.00	0.89
2	2654	3934	225501	1.99	0.89
3	2232	3676	155995	2.88	0.85
4	2256	3838	135056	3.33	0.74
5	2243	3873	126682	3.55	0.63
6	2256	3883	124979	3.60	0.53
7	2283	3894	124999	3.60	0.46
8	2170	3808	125005	3.60	0.34

Fonte: Elaborada pelo autor

quanto maior o aproveitamento dessas unidades, melhor o ganho em desempenho da máquina, com relação à análise do *speedup*.

Para atestar que o baixo aproveitamento da estrutura paralela está diretamente relacionado ao desempenho das unidades funcionais, com relação à implementação da **MDFM**, foi proposta uma modificação nessa característica da máquina. Foi adicionado um atraso sistemático nessas unidades, capaz de promover a degradação de seu desempenho, a partir do retardo na execução de instruções. Dessa forma, ao invés de levar o período fixo de 5 ciclos do relógio, para a execução completa das instruções menos complexas, este valor passou a ser fixado em um período parametrizado de ciclos. Dessa forma, foi possível atingir um *speedup* próximo ao ideal, utilizando valores ótimos na quantidade de ciclos presentes neste atraso intencional.

No gráfico da Figura 38 é possível verificar a capacidade da máquina em atingir um *speedup* ideal. Para a execução deste programa foram utilizados os mesmos parâmetros de entrada do gráfico da Figura 37. Nesta análise, apenas o programa de integração com operações trigonométricas foi utilizado. Um resultado importante encontrado a partir deste gráfico é o comportamento da quantidade de ciclos do relógio de atraso necessária para atingir este perfil ideal. Este crescimento é dado por uma expressão linear, que apresenta sua evolução de acordo com a Equação 6.6. Esta expressão é dada como função da quantidade de unidades funcionais “ f ” da solução, ou seja, é necessário um atraso cada vez maior de acordo com a adição de novas unidades de processamento. Este atraso é equivalente à aproximadamente o dobro da quantidade de f , em cada uma dessas unidades, a fim de obter o melhor desempenho em cada composição da máquina. Este valor de referência é importante para a sequência do projeto, pois se tornou base para a determinação do novo fator de liberdade que foi introduzido na sequência, que é a quantidade de dutos de largura de banda da solução. Conforme descrito em detalhes anteriormente, na Seção 3.2,

Figura 38 – Curva de *Speedup* na π **Flow** com a introdução do sistema de atraso na execução de instruções. Para esta análise foi utilizado apenas o programa de integração que executa funções trigonométricas, variando a quantidade de atraso necessária a fim de identificar o melhor *speedup* possível dentro desta implementação.

Fonte: Elaborada pelo autor.

este valor é capaz de determinar com uma precisão suficiente, o *throughput* de fichas que chegam à unidade de processamento.

$$\begin{aligned}
 D(f) &= \alpha f + \beta, \\
 \alpha &= 2,08 \pm 0,06 \\
 \beta &= 6,4 \pm 0,7
 \end{aligned}
 \tag{6.6}$$

O ajuste utilizado para a determinação da expressão anterior foi tomado a partir do programa **Gnuplot**, utilizando os valores presentes na Tabela 4. Uma informação importante que pode ser verificada na tabela, em comparação com os resultados anteriores, é a porcentagem de utilização das unidades funcionais a partir da adição deste atraso intencional. Este valor se mantém bastante alto para as quantidades de atrasos que apresentaram melhor desempenho nesta estrutura. Esta característica é uma das principais maneiras para conseguir obter o melhor desempenho possível para a execução de programas dentro desta arquitetura. Note ainda que estes valores são muito maiores com relação aos resultados anteriores, onde o aproveitamento dessas unidades eram muito baixos,

principalmente quando a evolução no *speedup* atingia seu limite assintótico.

Tabela 4 – Resultados obtidos para a determinação do crescimento em *speedup* no programa de integração que opera sob funções trigonométricas, com a introdução de um atraso adicionado intencionalmente na operação das unidades funcionais, com a finalidade de promover sua maior utilização quando comparadas ao restante da máquina.

FU	Atraso	MTQ	MMU	Ciclos	Speedup	%Uso FUs
1	3	3441	3813	597846	1.00	0.91
2	12	3500	3669	520202	1.99	0.95
3	14	3354	3630	379755	2.99	0.95
4	15	3133	3679	297274	3.99	0.96
5	18	3045	3689	267495	4.99	0.96
6	20	2961	3698	239454	5.99	0.96
7	21	2864	3756	212386	6.98	0.96
8	24	2791	3816	204438	7.97	0.97
9	26	2784	3874	192780	8.96	0.97
10	27	2656	3859	178533	9.95	0.97
11	30	2643	3874	175886	10.94	0.97
12	32	2614	3906	169541	11.93	0.97
13	34	2580	3884	164200	12.92	0.97
14	35	2569	3938	156125	13.90	0.97
15	36	2425	3917	149124	14.88	0.97
16	40	2497	3970	152245	15.87	0.97
17	42	2511	4017	149245	16.85	0.97
18	44	2443	4005	146534	17.83	0.97
19	45	2396	4004	141526	18.81	0.97
20	48	2391	3975	142015	19.78	0.87

Fonte: Elaborada pelo autor

Quanto maior o período de atraso para a geração das fichas subsequentes nas unidades funcionais durante a execução de programas, maior será o aproveitamento de todo o conjunto de unidades presentes na composição da máquina, principalmente devido a distribuição mais igualitária entre estas unidades, pois elas se mantêm ocupadas por um período de tempo muito maior, permitindo que o restante tenha a “oportunidade” de receber fichas para execução paralela. Entretanto, este fator reforça o argumento de que ao se aproximar de um aproveitamento ótimo destas unidades, melhor será o ganho em *speedup* na máquina, e conseqüentemente, melhor será sua escalabilidade.

6.4 π FlowBW

Foi então introduzido o conceito de dutos de largura de banda na π Flow. Este atributo foi implementado de forma parametrizável, permitindo que ele pudesse ser alterado em tempo de compilação no **BSV**. Este parâmetro passou a permitir um maior ganho

em desempenho, muito melhor quando comparado com a versão anterior, conforme pode ser visto no gráfico da Figura 39. Para programas menores, o desempenho segue o mesmo padrão da versão anterior. A razão é que, para programas pequenos, não existe paralelismo suficiente para ser explorado por esta arquitetura. Dessa forma, como esperado, independente da adição de novas unidades de processamento, não existe ganho em desempenho pois não existem instruções suficientes para distribuição durante a execução do programa. Entretanto, para programas maiores e que apresentam grande potencial de paralelismo, a utilização dessas unidades tende a manter-se mais equilibrada de acordo com seu aumento. Este é o propósito desta arquitetura, sua capacidade em distribuir a carga de processamento com quantas unidades funcionais estiverem disponíveis. É importante observar que durante a execução de programas maiores, existem muitas fichas sendo geradas simultaneamente, seguindo as capacidades dinâmicas deste modelo. De forma que, o grafo de representação deste programa, apresenta uma quantidade muito grande de instruções paralelas. Com a introdução dos dutos de largura de banda, houve um considerável aumento na capacidade da estrutura em realizar a distribuição destes recursos de forma mais eficiente para o consumo das unidades funcionais, promovendo sua capacidade para ganho em desempenho diante de seu aumento.

Figura 39 – Curva de *Speedup* na π **FlowBW**, para a execução do programa utilizado para realizar o cálculo do fatorial modificado. Neste gráfico foi disposta a evolução para ganho em desempenho para o mesmo programa, em diferentes tamanhos.

Fonte: Elaborada pelo autor.

Note que de maneira equivalente aos gráficos anteriores, existe uma forte tendência ao comportamento assintótico do programa de maior tamanho presente nesta figura, para valores próximos a 10 unidades funcionais. Apesar deste resultado ser superior em comparação com a versão anterior do protótipo, ainda não trata-se de um aumento considerável. De acordo com a tendência do gráfico ainda pode haver um pequeno aumento neste valor, mas definitivamente ele não é capaz de atingir um *speedup* de 10, independente da quantidade de unidades que forem adicionadas, é notável que a curva já está muito próxima de seu comportamento estacionário. Lembre-se que este programa possui paralelismo médio superior a 200. Conforme destacado anteriormente, isso significa que menos de 5% das instruções deste programa, que podem ser executadas paralelamente, se aproveitam desta propriedade da máquina. É importante reforçar que essa característica é esperada em qualquer ambiente paralelo, devido aos gargalos naturais presentes em qualquer arquitetura destinada para exploração de paralelismo. Entretanto, ainda é possível verificar um gargalo bastante limitado e sistemático, que não era verificado na **MDFM**.

Depois de várias simulações, foi verificada a necessidade de uma quantidade de dutos de largura de banda b que fosse pelo menos superior ao dobro da quantidade de unidades funcionais f , principalmente para valores grandes de f . De acordo com a Tabela 5, é possível verificar que o valor de b não obedece a um padrão muito bem definido. Este resultado já era esperado devido ao formato de distribuição do programa dentro destes dutos de comunicação, já que este valor acaba apresentando uma grande dependência com relação ao algoritmo utilizado. Durante a execução de um trecho de programa que se aproveita das propriedades dinâmicas da máquina, pode ocorrer um relativo acúmulo de fichas destinadas para uma instrução específica, gerando uma certa sobrecarga dentro de determinado duto de comunicação. Apesar desta característica, é possível verificar que os valores de b estão sempre próximos ao dobro de f . A principal exceção na distribuição foi verificada para o uso de 2 unidades funcionais, em que o valor de b é igual ao valor de f . Estes pontos foram escolhidos diante da execução do mesmo programa, com a variação dos parâmetros de configuração da máquina a partir do qual, foram extraídos os melhores desempenhos de acordo com a variação no valor de f . Apesar dos valores de b equivalentes ao dobro de f sempre se apresentaram bem próximos do melhor desempenho, foram escolhidos os pontos que apresentaram estritamente os melhores resultados para a análise.

É importante destacar as limitações encontradas nesta fase do desenvolvimento. Conforme destacado na Tabela 5, é possível verificar que foram utilizados apenas 18 dutos de largura de banda na configuração com 10 unidades funcionais. Entretanto, como é possível verificar nas distribuições com 5 e 6 unidades, por exemplo, existem exceções expressivas na quantidade de b , a fim de obter o melhor desempenho possível. Conforme destacado anteriormente, essas características possuem forte dependência com o algoritmo do programa que está sendo executado, devido principalmente à distribuição das instruções do programa dentro dos dutos de largura de banda. Dessa forma, é possível que este

Tabela 5 – Valores utilizados para a construção do gráfico da Figura 39, referentes à execução do programa do fatorial modificado, para o cálculo do valor 1000, que trata-se da curva de melhor desempenho no gráfico referenciado. Este programa possui um total de 28985 instruções e valor de $\pi = 223$.

FU	BW	MMU	Ciclos	Speedup	%Uso FUs
1	2	2254	87039	1.00	0.99
2	2	2218	43656	1.99	0.99
3	7	2592	29204	2.98	0.99
4	7	2434	22706	3.83	0.96
5	14	2364	17974	4.84	0.97
6	14	2108	15225	5.72	0.95
7	14	2899	13926	6.25	0.89
8	14	2041	12428	7.00	0.87
9	18	2722	11468	7.59	0.84
10	18	2408	11028	7.89	0.79

Fonte: Elaborada pelo autor

programa possa apresentar melhor desempenho com o uso de 10 unidades funcionais, para valores de b que sejam superiores a 20. Porém, não foi possível realizar nem ao menos a simulação destas soluções, principalmente pela complexidade do circuito de distribuição das fichas entre os dutos e as unidades funcionais, trecho do circuito que apresenta maior caminho crítico, de acordo com o relatório de síntese da solução no *Quartus*. Este circuito foi implementado dentro de laços complexos no **BSV**, o que prejudicou o processo de compilação do projeto, no que parece uma limitação da linguagem. Durante o processo de compilação, todo o projeto fica armazenado em memória e para soluções mais complexas, mesmo havendo memória suficiente, foi verificada a ocorrência de falha de segmentação durante o processamento. Por este motivo não foram exploradas distribuições com mais unidades funcionais durante o desenvolvimento, deste ponto em diante. Este circuito se tornou demasiado complexo devido a capacidade da implementação em distribuir qualquer quantidade de fichas, limitada pelo valor de b , às unidades funcionais, em todo ciclo do relógio.

Apesar destas limitações é possível verificar um expressivo aumento na utilização das unidades funcionais, em comparação com a versão anterior do projeto. O aumento na capacidade de exploração destas unidades reforça a característica destacada anteriormente de que a maximização na utilização das unidades de processamento da arquitetura é capaz de promover melhor escalabilidade para ela. As informações da Tabela 5, em conjunto com a análise do gráfico, Figura 39, destaca com clareza esta propriedade. Perceba que até a distribuição com 6 unidades funcionais, as utilizações destas unidades se mantêm superiores a 90% e é neste trecho de análise que a escalabilidade da máquina mantém sua melhor evolução. A partir deste valor, tem início uma degradação nessa utilização, principalmente devido a eventuais sobrecargas dentro de dutos específicos ao longo da

execução completa deste programa, fazendo com que o caráter assintótico da curva ainda seja alcançado de forma precoce.

Perceba pelo gráfico que apesar de ser possível se aproximar ainda mais do *speedup* ideal, a implementação está próxima de seu limite para a exploração deste programa. Essa característica se destaca pela proximidade entre as curvas dos tamanhos do programa para o cálculo do “fatorial” de 500 e de 1000. Mesmo para programas muito maiores, o comportamento do *speedup* mantém-se, com apenas pequenas variações.

6.5 Otimizações na π FlowBW

Os resultados apresentados na sequência foram todos obtidos a partir das implementações realizadas durante o período de desenvolvimento da presente tese de doutorado. A solução dos dutos de largura de banda implementados para a π FlowBW ainda durante o desenvolvimento da dissertação de mestrado do autor deste trabalho, reforçaram uma característica importante para a realização deste desenvolvimento. Esta propriedade acabou destacando os benefícios verificados na distribuição da carga de trabalho em cada unidade que compõe a solução. É bastante evidente nesta distribuição o aumento na capacidade da máquina em manter as unidades funcionais mais ocupadas. Esta solução oferece uma boa estratégia que permite controlar, com certa precisão, a quantidade de informações que transitam por cada unidade da máquina e conseqüentemente na unidade de processamento. Porém, conforme destacado anteriormente, na Seção 4.2, a quantidade de recursos necessários para alcançar essa capacidade, acabou impedindo até mesmo a simulação de soluções mais robustas. Com isso, o principal desafio deste projeto se tornou a distribuição dos recursos necessários para a execução de programas em um formato que garanta melhor escalabilidade.

Apesar de já possuir todas as características necessárias para a execução de programas, foram realizadas as adaptações descritas na seção com a descrição das soluções de passagem direta de fichas, Seção 4.2, a fim de reduzir os maiores caminhos críticos verificados na solução. Estes longos caminhos acabaram impedindo a capacidade de síntese do projeto em frequências mais elevadas. Portanto foram buscadas alternativas que permitissem ganho em desempenho a partir da estrutura já construída. Com a divisão dos recursos necessários para a realização das mesmas operações em circuitos menores, foi verificado um aumento que apesar de natural, se mostrou considerável na quantidade de ciclos do relógio necessários para a execução dos mesmos programas analisados. Como este aumento é sistemático e verificado durante o tratamento de todas as fichas, não foi possível verificar alteração no aspecto das curvas de *speedup*, uma vez que este excesso de ciclos acaba se anulando na análise de uma e de várias unidades funcionais, devido ao formato da expressão referente à análise do *speedup*.

A introdução das estratégias de passagem direta de fichas em cada unidade na composição da máquina garantiram uma considerável melhora na execução dos programas, em comparação ao estado anterior, mas agora, com menores caminhos críticos. A princípio pode parecer que essas estruturas são capazes de adicionar uma grande complexidade na ordem de tratamento das fichas dentro da estrutura da arquitetura. Entretanto, em uma análise mais detalhada, é possível verificar que este ganho em desempenho ainda apresenta uma ordem de tratamento de fichas bastante controlada durante a execução de programas. Dependendo da quantidade de dutos de largura de banda da solução e consequente distribuição de um programa dentro dessa estrutura, a ocorrência de “acerto” no sistema de “predição de desvios” da unidade de instruções pode ser considerada bastante “comportada”, independente da quantidade de unidades funcionais, apesar da ordem na execução das instruções poder sofrer algumas variações na execução de programas, dependendo da quantidade de unidades de processamento presentes na solução. Entretanto, caso sejam utilizadas distribuições equivalentes em dutos de largura de banda, as variações na análise de *speedup* acabam por apresentar comportamentos perfeitamente equivalentes. Afinal, apenas esta distribuição deve ser considerada para a análise desta nova estrutura, de predição de desvios, pois ela depende exclusivamente da distribuição das instruções do programa, dentro das memórias **RAMs** da unidade de instruções, que são distribuídas dentre os dutos de largura de banda presentes na solução, caracterizando a taxa de acerto, dentro deste sistema.

Um comportamento equivalente pode ser encontrado sobre a utilização do sistema de pré-processamento, que depende exclusivamente da quantidade de instruções presentes no programa em execução, conforme descrito em detalhes na Seção 4.2. Portanto, esta estrutura também não é capaz de modificar o comportamento do *speedup*, ou a ordem de tratamento das fichas durante a execução de programas, uma vez que os resultados apresentados nesta análise considera a execução de programas de mesmo tamanho apenas, tratando-se então de outra solução que não acrescenta modificações muito marcantes com relação a esta métrica, em comparação com o estado anterior da implementação.

Foram realizadas novas medidas de *speedup* para atestar o comportamento da escalabilidade desta versão da implementação, além de uma análise da ocupação desta estrutura revisada, em **FPGAs**, a fim de verificar a capacidade de síntese desta solução no processo de compilação do *Quartus*. Pelo gráfico da Figura 40 vê-se que a utilização de recursos nas **FPGAs** pode ser ajustada de acordo com uma série polinomial aproximadamente quadrática, conforme a expressão indicada na Equação 6.7. O processo de compilação foi realizado para ser compatível com a **FPGA Cyclone V 5CSEA5**, que é o modelo presente nos *kits* de desenvolvimento **DE1-SoC**, utilizados para a verificação da π **FlowBW**, ainda durante o desenvolvimento da dissertação de mestrado deste autor. Note pelo gráfico que este modelo de **FPGA** possui menos recursos lógicos em comparação com o modelo *5CSEA6*, utilizado para o desenvolvimento deste trabalho. Como as **FPGAs**

da Altera / Intel possuem características bastante equivalentes entre si, foi adicionado no gráfico ainda outros modelos compatíveis, de acordo com a quantidade de recursos lógicos presentes em cada um deles.

$$\begin{aligned}
 U(f) &= \alpha f^\beta + \gamma, \\
 \alpha &= 3461 \pm 562 \\
 \beta &= 1.62 \pm 0.09 \\
 \gamma &= 2422 \pm 1036
 \end{aligned}
 \tag{6.7}$$

Outro modelo de **FPGA** disponível para o desenvolvimento deste projeto é o *kit Arria V GX*, que possui o modelo de **FPGA** também destacado no gráfico. Em sua **FPGA**, de acordo com a extrapolação analisada, seria possível a utilização de até 9 unidades funcionais em sua estrutura. Apesar de contar com este outro dispositivo, infelizmente o *kit* de desenvolvimento não possui módulos de comunicação que sejam compatíveis com a estratégia utilizada na composição do **Multianel**. O *kit* de desenvolvimento possui grande enfoque em comunicação e módulos bastante específicos para entrada e saída, porém eles não são compatíveis com o protocolo paralelo implementado. Além disso, ainda analisando a disposição do gráfico, é possível verificar que a **FPGA Stratix V GT C5**, poderia ser capaz de utilizar até 10 unidades funcionais.

É importante reforçar que o uso de 10 unidades funcionais na composição da máquina já era tomado como o limite superior de análise nas versões anteriores do projeto, devido a uma limitação verificada no processo de compilação do **BSV**. Este processo impedia a implementação da máquina com a quantidade de recursos necessários para a utilização de mais de 10 unidades funcionais e sua consequente quantidade de dutos de largura de banda necessária para atingir seu melhor desempenho. Este comportamento não pôde ser modificado com as implementações propostas. A limitação verificada refere-se ao processo de compilação para a geração do código para simulação principalmente. Na compilação para geração do código **Verilog**, destinado ao processo de síntese, não foi verificada esta limitação. Com isso, a distribuição destinada para o uso de **FPGAs** de maior capacidade segue como uma alternativa plausível, considerando mesmo que por absurdo, que a extrapolação realizada possa ser tomada como uma aproximação válida. Entretanto, como o enfoque deste projeto tem a finalidade de verificar o desempenho desta máquina com o uso de **FPGAs** de baixo custo principalmente, a análise desta característica ficou limitada apenas a esta extrapolação.

A partir da Figura 40 foi verificada a utilização dos componentes conhecidos como Módulos de Lógicas Adaptáveis (**ALM**, do inglês *Adaptive Logic Module*), exclusivos da Altera / Intel. Na linha de **FPGAs Cyclone V**, estes módulos são formados por uma **LUT** de 8 entradas com 4 registradores. Por possuir esta distribuição, segundo a divulgação promovida pela fabricante, as **ALMs** são equivalentes a 2.5 elementos lógicos nesta família

de dispositivos.⁷³ Como a **FPGA** da **DE10-Nano** apresenta um total de 110 mil elementos lógicos, ela possui então ~44 mil **ALMs**, conforme mostrado no gráfico. Foi adotada a quantidade de **ALMs** como métrica principal, devido ao relatório obtido pelo processo de síntese do *Quartus*, que se utiliza desta métrica por padrão.

Figura 40 – Evolução e extrapolação da utilização de recursos no processo de compilação do projeto para utilização em **FPGAs**, de acordo com a adição de unidades funcionais e consequente quantidade de dutos de largura de banda.

Fonte: Elaborada pelo autor.

De acordo com o gráfico da Figura 40, por apresentar a limitação de 4 unidades funcionais por *kit* de desenvolvimento e possuir ainda a limitação de 4 *kits* para a realização deste projeto, foi realizada uma nova análise de *speedup* para quantidades próximas a este valor. Como pode ser visto no gráfico da Figura 41, foi adotado o limite máximo de 5 unidades funcionais na solução de anel único. Este valor de referência foi tomado com base nos resultados anteriores, pois a máquina sempre apresentou um bom comportamento em *speedup* para essa quantidade de recursos, mantendo a curva bastante próxima do ideal. Além disto, é a partir deste ponto que a curva começa a encontrar um certo limite de maior degradação para ganho em desempenho. Note pela Tabela 5 que a partir do valor de 5 unidades funcionais a quantidade de dutos de largura de banda necessárias para obtenção do melhor desempenho se torna fixa em 14 elementos, no que parece ser a melhor distribuição possível ao longo da execução do programa utilizado. Porém, apesar desta melhor distribuição, a capacidade de utilização das unidades funcionais passa a decair

a uma taxa frequente, quase constante, com a adição de mais recursos. É possível que existam melhores distribuições do programa nesta configuração para quantidades de dutos de largura de banda superiores ao valor limite de 20, imposto pela limitação verificada no processo de compilação do **BSV**. Em todo caso, para esta quantidade de recursos limitada, a utilização das unidades de processamento mantém-se ainda bastante elevada.

Figura 41 – Comportamento da curva de *speedup* da última versão da π **FlowBW**, antes de ser iniciado o processo de implementação da arquitetura do **Multianel**. O ajuste teórico tem a finalidade de encontrar uma aproximação no comportamento da máquina com mais unidades funcionais.

Fonte: Elaborada pelo autor.

Foi realizada uma nova extrapolação, desta vez para a análise da curva de *speedup*, a fim de verificar qual seria o limite em desempenho possível, com a quantidade de recursos físicos a disposição para o desenvolvimento deste projeto. A curva apresentada na Figura 41 possui uma tendência que pode ser descrita a partir da expressão presente na Equação 6.8. Como a quantidade de unidades funcionais utilizadas na composição do **Multianel** físico será limitada a 4 unidades, passou a ser esperado que o máximo *speedup* possível de ser obtido nesta composição tenha o valor de 12,93 na execução do programa que realiza o cálculo do fatorial modificado de 500, de acordo com a Equação 6.8, considerando o uso de todos os recursos físicos disponíveis. A hipótese desta avaliação considera que ao incluir o ponto de início na degradação para ganho em desempenho, segundo os resultados dispostos na Tabela 5, os gargalos promovidos a partir da estratégia do **Multianel**, serão menores que os gargalos verificados na estratégia da largura de banda. Esta parece uma

hipótese razoável pois é esperada a introdução de um novo gargalo a partir desta nova implementação, proveniente da necessidade de comunicação entre os anéis desta estrutura. Porém, é esperado também que este gargalo seja menor em comparação com a degradação verificada para ganho em desempenho a partir da estratégia exclusiva da distribuição de recursos entre os dutos de largura de banda, principalmente devido ao formato de distribuição dentro desta estratégia. A estrutura do **Multianel** tende a remover quaisquer gargalos existentes na estratégia de anel único, provenientes principalmente pelo acúmulo excessivo de fichas em determinados dutos de largura de banda.

$$\begin{aligned} S(f) &= \alpha f^{1/\beta}, \\ \alpha &= 1,04 \pm 0,02 \\ \beta &= 1.10 \pm 0.02 \end{aligned} \tag{6.8}$$

Nesta análise não foi considerado que o comportamento assintótico da curva será atingido dentro dos limites dispostos no gráfico, que também parece uma hipótese razoável ao considerar o paralelismo médio deste programa que é de 123. É importante reforçar ainda que o programa utilizado para a realização desta análise possui o pior desempenho dentre os programas disponíveis para validação. Essa característica foi buscada a fim de verificar qual seria o limite inferior dentre as possibilidades que podiam ser avaliadas de acordo com os recursos disponíveis. Com isso, é esperado que qualquer programa utilizado para validação da máquina física tenha um desempenho superior ao limite em *speedup* de 12,93 aqui obtido.

6.6 Multianel

A partir da versão revisada da π **FlowBW** foram iniciadas as primeiras tentativas de implementação da π **FlowMR**. O módulo de comunicação foi implementado inicialmente em um projeto exclusivo para essa finalidade, com apenas este módulo instanciado no *TopLevel* de um novo projeto do **BSV**, a fim de verificar seu funcionamento nos ambientes de simulação e de síntese, de acordo com as descrições detalhadas no Capítulo 5. Foi possível identificar um grande gargalo introduzido pelo módulo de comunicação implementado nesta primeira versão, que não possuía nenhuma estratégia razoável para se aproveitar da frequência de operação mais alta deste módulo, conforme proposto pelo protocolo paralelo, em comparação com as outras unidades que compõem os anéis da solução.

Diante deste problema, foi implementada uma solução paliativa, buscando apenas a validação da estratégia do **Multianel**, em uma estrutura de comunicação impraticável, principalmente ao considerar os recursos físicos a disposição para a implementação deste projeto. A partir da revisão dessa implementação, em um módulo de comunicação plausível, passou a ser possível a exploração de um alto grau de paralelismo na máquina. Esta

implementação ainda é capaz de se aproveitar do atraso promovido pela necessidade de comunicação entre os anéis da máquina, em uma estratégia semelhante ao efeito produzido pelo *Throttle* proposto para a **MDFM**.⁴³ Segundo esta estratégia, a introdução de uma redução do paralelismo no fluxo de operação natural da máquina, pode ser benéfico para ganho em desempenho, principalmente na análise do *speedup*. Esta redução promove a remoção de eventuais acúmulos excessivos de fichas na Fila de Fichas, diante do baixo consumo promovido pelas outras unidades que compõem a estrutura da máquina, uma vez que estas unidades são bastante limitadas para este consumo, principalmente considerando a existência de apenas um duto de comunicação entre elas, como era realizado na **MDFM**. Todos os resultados obtidos em cada uma destas fases da implementação serão destacados e discutidos em detalhes na sequência.

6.6.1 Primeiras Impressões

Na primeira tentativa de implementação do módulo de comunicação os resultados ficaram muito abaixo do esperado, como pode ser visto em detalhes nos gráficos da Figura 42. Todos os programas utilizados nesta análise referem-se às versões dos programas que operam apenas sobre valores inteiros. Perceba pelos gráficos das Figuras 42a e 42b, que a π **FlowBW**, com as modificações de passagem direta implementadas, segue o mesmo padrão de *speedup* verificado nas versões anteriores do protótipo. Porém, o comportamento assintótico dessa métrica é atingido ainda mais precocemente. Essa característica fica mais evidente na análise da curva apresentada na Figura 42a, quando comparada com a análise anterior à implementação das otimizações de passagem direta, vistas no gráfico da Figura 39. Aqui, é possível verificar pouca variação de desempenho entre os diferentes tamanhos deste programa, que em todos os casos possuem paralelismo médio superiores a 100, fato que não era verificado na versão anterior, quando era bastante perceptível o ganho em *speedup* de acordo com o aumento no tamanho do programa. Perceba neste gráfico, que para nenhuma execução esta métrica atinge ao menos o valor de 5. Este resultado é coerente com a análise feita anteriormente, quando foi destacado que é a partir de 5 unidades funcionais que o comportamento do *speedup* começa a se degradar consideravelmente na implementação.

Em comparação com os resultados da π **FlowBW**, mostrados na Tabela 5 é possível verificar um grande ganho em desempenho para a execução dos programas. Enquanto no programa do fatorial modificado para o cálculo do valor 1000, na versão anterior, eram necessários 87039 ciclos do relógio para a sua execução completa, aqui este valor caiu para 64532 em sua forma sequencial, com o uso de apenas 1 unidade funcional. Ou seja, houve um ganho em desempenho de ~26% para a execução deste programa. É importante reforçar que para a realização desta análise estavam definidas as instruções de soma e subtração no escopo do pré-processador, operações muito importantes para a execução da

(a) Fatorial modificado na π FlowBW. (b) Integração binária na π FlowBW. (c) Fatorial modificado na π FlowMR com 2 unidades funcionais por anel. (d) Integração binária na π FlowMR com 2 unidades funcionais por anel. (e) Fatorial modificado na π FlowMR com 4 unidades funcionais por anel. (f) Integração binária na π FlowMR com 4 unidades funcionais por anel.

Figura 42 – Conjunto de resultados da π FlowMR para os programas do fatorial e de integração, obtidos ainda na primeira versão da implementação do módulo de comunicação. Como destacado nos gráficos, é possível verificar que o *speedup* segue sempre abaixo ou muito próximo do valor 8, principalmente com a utilização de mais unidades funcionais.

Fonte: Elaborada pelo autor.

função recursiva de ambos os programas, conforme pode ser visto em detalhes no Anexo A.1. Nesta versão da implementação mais de 40% das instruções destes programas acabam se aproveitando da estrutura do pré-processador, deixando as unidades funcionais com uma taxa de utilização inferior a 75%. Este fato já é suficiente para identificar a causa do comportamento assintótico da curva, que ocorre de forma tão precoce e se mantém perfeitamente estável a partir deste ponto. Como o pré-processador é bastante explorado, a execução dos programas segue exatamente a mesma distribuição dentro da unidade de processamento, independente da quantidade de unidades funcionais definidas na solução. Este fato é comprovado pela análise da utilização destas unidades, que decaem a uma taxa praticamente constante de acordo com o acréscimo de recursos. A utilização é de 74% precisamente, mesmo com a utilização de apenas uma unidade funcional, destacando assim a larga utilização do pré-processador. Neste caso, a complexidade na ordem de execução do programa se dá exclusivamente pela distribuição das instruções nos dutos de largura de banda, resultando na baixa capacidade para ganho em *speedup* verificado.

Apesar de apresentar um melhor comportamento para ganho em desempenho, a execução do programa de integração, disposto na Figura 42b segue exatamente o mesmo padrão verificado no programa do fatorial modificado. A única diferença é que o comportamento assintótico é atingido apenas a partir da utilização de 8 unidades funcionais na solução, além de apresentar uma distância ligeiramente maior de acordo com o tamanho do programa, entre as curvas dispostas no gráfico, principalmente para quantidades maiores de recursos. Esta análise é mais próxima dos resultados anteriores e suficiente para verificar que apesar da introdução de instruções utilizadas com grande frequência na solução de passagem direta da unidade de processamento, a máquina segue com uma capacidade apenas razoável para escalabilidade, o que é natural, ao considerar a baixa taxa de utilização das unidades funcionais. Os programas utilizados para validação da máquina possuem características de implementação muito parecidas. Porém, o programa de integração possui mais instruções condicionais e de controle dentro do escopo da sua função recursiva. Por esse motivo que este programa possui uma escalabilidade um pouco melhor. As unidades funcionais se mantêm mais ocupadas durante sua execução, em comparação com o programa do fatorial.

A Tabela 6 destaca os principais indicadores utilizados para avaliar o uso de recursos da máquina durante a execução de programas. Diferentemente das tabelas anteriores, aqui é possível verificar os indicadores de utilização das otimizações de passagem direta das unidades de processamento e de instruções, nas colunas *PUBY* e *ISBY*, respectivamente. Note que a quantidade de instruções que se aproveitam dos recursos do pré-processador são estáticas e independem da quantidade de unidades funcionais presentes na solução. Este indicador depende exclusivamente do tamanho do programa, que não foi variado nesta análise. Para ambos os programas, foram utilizados seus respectivos tamanhos intermediários, segundo as informações presentes na Tabela 2. Perceba que de forma

coerente com o gráfico da Figura 42a, o programa do fatorial se aproveita mais da estrutura do pré-processador. Ou seja, a função recursiva deste programa possui poucas instruções além da soma e da subtração em seu escopo, em comparação com o programa de integração, o que implica no alcance tão precoce ao comportamento assintótico na análise da curva de *speedup*. Ainda sob esta perspectiva, mesmo com o uso de apenas uma unidade funcional, o aproveitamento de utilização desta unidade é sempre inferior a 75%, para ambos os programas, indicando que a implementação do pré-processador, com o uso de instruções de frequência mais alta pode afetar bastante o desempenho da máquina. Verifica-se que o tempo global para a execução destes programas acabou sendo bastante reduzida, principalmente devido ao desempenho promovido pelo uso considerável deste artifício. Porém, como foi buscada principalmente a viabilidade sobre a capacidade de escalabilidade da arquitetura, foram propostas medidas capazes de evitar o baixo aproveitamento das unidades funcionais.

Tabela 6 – Distribuição de recursos para a execução dos programas de tamanho intermediário na π **FlowBW**, com a adição das soluções de passagem direta. O programa de integração utilizado nesta análise refere-se a versão que opera sobre valores inteiros apenas.

	FU	BW	MTQ	MMU	# Ciclos	PUBY	ISBY	%FUs
Fatorial	1	7	1613	3172	64532	12990	19888	0.74
	6	14	394	1225	13549		27400	0.59
Integração	1	10	859	1585	57871	9722	15528	0.75
	6	16	550	1525	10661		20235	0.68

Fonte: Elaborada pelo autor

Outra constatação importante da análise da Tabela 6 é a taxa de acerto da solução de passagem direta da unidade de instruções, além da sua relação estrita com a quantidade de dutos de largura de banda presentes na composição da máquina, coerente com o esperado. Esta característica deve ser relacionada com a quantidade total de instruções presentes em cada programa, conforme os valores de S_0 também dispostos na Tabela 2. Dessa forma, a taxa de acerto geral se manteve superior a 65% na análise com uma única unidade funcional em ambos os programas e ainda uma taxa de acerto de 95% no programa do fatorial com o uso de 6 unidades funcionais e 14 dutos de largura de banda e mesmo uma taxa de 86% no programa de integração que também utiliza uma quantidade alta de dutos de largura de banda. Este valor tende a crescer de acordo com a adição de mais dutos, uma vez que existem mais possibilidades de acerto de acordo com a distribuição das instruções do programa dentro destes elementos de estado. Isso reforça que definitivamente a unidade de instruções não é capaz de adicionar mais quaisquer atrasos durante a execução de programas, um problema que se mantinha persistente desde as primeiras versões deste trabalho, uma vez que existia um atraso com o período fixo de 2 ciclos do relógio na ocorrência de qualquer ficha de entrada nesta unidade. Este fato

pôde ser mitigado com a introdução dos dutos de largura de banda, que aumentava o número de operações paralelas para lidar com este atraso natural da implementação, mas que finalmente se tornou totalmente obsoleto a partir das soluções de passagem direta propostas.

Ainda acerca dos gráficos presentes na Figura 42, foi analisado também o ganho em *speedup* como função da adição de novos anéis na máquina, mantendo-se fixa a quantidade de unidades funcionais por anel. Esta análise tem um maior impacto na prática, uma vez que a proposta é a criação de anéis de processamento com quantidades de recursos fixos, que devem apenas ser incorporados à estrutura do **Multianel** já estabelecida. Existe um leve aumento no ganho em desempenho com a adição de anéis na estrutura, como mostrado nos gráficos das Figuras 42c e 42d, utilizando apenas 2 unidades funcionais por anel, sendo esperado portanto um melhor aproveitamento destas unidades no contexto do **Multianel**, de acordo com o desempenho verificado nos gráficos das Figuras 43a e 43b, onde é possível atestar que o desempenho da máquina com essa quantidade de recursos ainda é ótimo. No programa do fatorial, o *speedup* continua limitado ao valor de 5, exatamente como verificado no comportamento da π **FlowBW**. Apesar de possuir exatamente o mesmo limite em *speedup*, este valor é atingido apenas a partir da utilização de mais recursos, neste caso, de 16 unidades funcionais. Deste limite em diante a evolução do *speedup* passa a decair a uma taxa menos abrupta em comparação com a execução do programa de integração.

Apesar do **Multianel** já promover a distribuição dos recursos necessários para a execução de programas, aqui é notável o impacto do gargalo introduzido pelo módulo de comunicação proposto nesta primeira tentativa de implementação. Isto pode ser visto com ainda mais detalhes no gráfico da execução do programa de integração da Figura 42d, em comparação com sua versão analisada em um único anel. O comportamento assintótico é mais abrupto e a partir de 8 anéis o desempenho da máquina passa a decair a uma frequência muito maior em comparação com o programa do fatorial, indicando que ambos os gargalos passaram a contribuir negativamente em conjunto, impedindo o ganho em desempenho com a adição de ainda mais recursos na máquina. Entretanto, neste cenário já é possível verificar os benefícios do **Multianel**. A capacidade da máquina para a exploração de mais unidades de processamento chega a 20, valor que era possível de ser implementado apenas nas primeiras versões da π **Flow**, devido a simplicidade de seus circuitos. Neste limite de 8 anéis, a máquina já se encontra com mais de 16 unidades funcionais e a estrutura construída até este momento não era capaz de manter estas unidades ocupadas, devido principalmente ao trajeto de uma ficha entre os diferentes anéis da estrutura até atingir as unidades funcionais, que ainda contava com o gargalo identificado, que impedia a passagem de mais fichas por entre os módulos de comunicação.

Apesar do baixo aproveitamento de recursos é importante destacar o conceito

amplo sob a definição da métrica do *speedup*. Esta métrica deveria ser aplicada como função de uma estrutura puramente sequencial, de acordo com sua definição. Entretanto, sob a perspectiva do **Multianel**, como esta solução tem a finalidade exclusiva de remover quaisquer sobrecargas e acúmulos de fichas dentro das unidades que compõem a estrutura de cada anel e mesmo sob o contexto dos dutos de largura de banda que os compõem, é importante destacar uma característica marcante desta análise. Como a relação do *speedup* foi realizado como função de uma estrutura já paralela, é mais acurado analisar esta métrica como função de seu valor dentro da estrutura em questão. Ou seja, nos gráficos das Figuras 42c e 42d, como a métrica foi aplicada como função de um anel com duas unidades funcionais definidas em seu escopo, o valor do *speedup* alcançado por esta distribuição, pode ser analisado como uma função de seu desempenho no formato em um único anel. Neste caso, o *speedup* obtido no contexto do **Multianel**, pode ser analisado como um múltiplo do *speedup* verificado nesta configuração, que para 2 unidades funcionais é de ~ 1.99 , em ambos os programas. Dessa forma, segundo esta perspectiva, o limite assintótico do *speedup* verificado no gráfico da Figura 42c não é de 5 explicitamente, como indicado no gráfico, mas sim um valor próximo a ~ 9.89 . E de forma equivalente, o limite assintótico do programa de integração binária, disposto na Figura 42d, pode ser tomado pelo valor de ~ 11.83 , conforme a compilação dos resultados dispostos na Tabela 7.

Tabela 7 – Análise do *Speedup* Efetivo, considerando a quantidade máxima de recursos necessários para atingir o limite do valor assintótico desta métrica, para a execução de ambos os programas, operando sobre valores inteiros apenas, em seus maiores tamanhos analisados.

Progr	# FUs	# Anéis	# Ciclos	Speedup	Speedup Ef.
Fatorial	1	1	96547	1.00	–
	2	1	48613	1.99	–
	4	1	25169	3.84	–
	2	8	9777	4.97	9.89
	4	5	9538	2.64	10.13
Integração	1	1	115645	1.00	–
	2	1	58131	1.99	–
	4	1	29553	3.91	–
	2	8	9782	5.94	11.83
	4	5	9024	3.27	12.80

Fonte: Elaborada pelo autor

Com isso, nota-se um ganho efetivo em desempenho, no aspecto da escalabilidade. Apesar do limite assintótico ainda ser bastante precoce, principalmente tendo em conta os valores de paralelismo médio destes programas, é possível verificar um ganho em desempenho considerável, em comparação com a versão de único anel. Os valores de *speedup* atingidos não puderam ser alcançados nas versões anteriores deste projeto e isso fica ainda mais evidente na análise seguinte, mostrada na Figura 43. Além disso, os tamanhos de programas utilizados até este momento, apresentam um baixo paralelismo,

uma vez que a exploração dos recursos do **Multianel** leva um período de tempo maior para atingir sua plena capacidade para exploração de paralelismo, devido exclusivamente a necessidade de comunicação entre os anéis da estrutura, principalmente com as limitações verificadas nos módulos de comunicação nesta etapa do desenvolvimento. Do momento que a distribuição de fichas entre estes anéis já foram realizadas, em diante, a execução dos programas apresenta um desempenho impressionante para este tipo de execução paralela. Entretanto, esta capacidade é alcançada muito mais tardiamente, em comparação com a análise de um único anel. As maiores vantagens do **Multianel** se encontram para programas muito maiores e essa característica já pode ser observada nos gráficos das Figuras 42c e 42d, a partir da distância entre as curvas referentes aos programas de tamanhos distintos, em comparação com os gráficos das Figuras 42a e 42b, destacando uma melhor capacidade da máquina para a distribuição de carga, em programas de tamanhos diferentes, para a exploração do **Multianel**.

É importante considerar o processo de distribuição de fichas entre os anéis dentro da estrutura do **Multianel**. Para isto, foram utilizados os rótulos dinâmicos do modelo, conforme descritos em detalhes na Seção 5.0.1. Durante a execução de um programa, seu grafo de representação é expandido de acordo com os critérios dinâmicos de sua implementação. Nos principais programas utilizados para a validação da máquina, o rótulo dinâmico mais importante, é aquele conhecido como Nome de Ativação. O Nome de Ativação é utilizado principalmente para diferenciar as chamadas de funções,¹⁶ e os principais programas utilizados para a validação da máquina exploram bastante esta capacidade por terem sido implementados de forma recursiva. A geração de novos Nomes de Ativação é realizada a partir de instruções específicas do conjunto de instruções da arquitetura. Diante da ocorrência destas instruções, um registrador exclusivo tem seu valor apenas incrementado. Note então, que este processo tem a finalidade de realizar a chamada a uma nova função, que necessariamente conta com a execução de um agrupamento de instruções, que serão executadas dentro do escopo desta função. Este escopo é definido a partir do valor presente neste rótulo dinâmico exclusivo, como definido na expressão da Equação 5.1.

A distribuição de fichas entre os anéis na estratégia do **Multianel** é feita baseada neste Nome de Ativação. Na ocorrência de uma ficha que possui um Nome de Ativação que não é destinado para o anel corrente, esta ficha será encaminhada para a estrutura de comunicação deste anel, em busca do seu destino definitivo, que será responsável pela execução da instrução destino da ficha. Assim, cada anel desta estrutura é capaz de fazer chamadas a funções que serão executadas tanto em sua própria estrutura, como para qualquer outro anel presente na máquina. Perceba que diante da ocorrência destas fichas “cruzadas”, destinadas para outro anel da estrutura, uma única ficha será destinada a explorar esta propriedade, e a partir da execução da instrução desta ficha particular, novas fichas serão geradas já dentro do escopo de sua função particular, em seu Nome de

Ativação específico, e portanto ficarão no mesmo anel que irá executar todas as instruções desta instância da função. Dessa forma, no âmbito do **Multianel**, as fichas que dependem de comunicação e portanto, são destinadas para uma outra “unidade” desta estrutura, podem ser tomadas como “macro-instruções” da arquitetura que acabam promovendo um aumento natural na granularidade da máquina, por consequência da distribuição escolhida, apesar de a granularidade explorada por cada anel manter sua característica original, capaz de oferecer todos os benefícios largamente discutidos na literatura da exploração de granularidade fina.

Nos gráficos das Figuras 42e e 42f é possível verificar como o gargalo do módulo de comunicação em conjunto com a presença do pré-processador marcou esta versão da máquina, apesar do comportamento do *speedup* ser bastante equivalente aos resultados anteriores. Aqui, esta métrica foi analisada como função da quantidade de anéis na estrutura do **Multianel** novamente, mas utilizando um total de 4 unidades funcionais fixas em cada anel. No programa do fatorial, o comportamento assintótico é alcançado a partir do uso de 3 anéis, ou seja, para um total de 12 unidades funcionais. A comparação com a análise da Figura 42c, onde este comportamento é atingido apenas com o uso de 16 unidades, reforça que o atraso promovido pelo gargalo do módulo de comunicação é ainda mais prejudicial ao ganho em *speedup* do que o pré-processador. O programa de integração segue um padrão equivalente ao da curva anterior, com a diferença que o comportamento assintótico é atingido apenas para *speedup*'s que flutuam ao redor do valor 3, que segundo a análise anterior é equivalente ao valor 12.80, considerando o *speedup* efetivo que no modelo de único anel é de 3.91, para o uso de 4 unidades funcionais, de acordo com a Tabela 7. Apenas para fins de análise, de forma equivalente, o *speedup* efetivo máximo do programa do fatorial modificado atingiu o valor máximo de 10.13, considerando o valor de 3.84 de *speedup* para o uso de 4 unidades funcionais. Apesar de não ser possível verificar essa capacidade na análise gráfica explicitamente, em comparação com os gráficos das Figuras 42c e 42d, houve uma evolução na capacidade da máquina para ganho em desempenho em ambos os programas, de acordo com a adição de mais recursos computacionais.

A quantidade de recursos utilizados para avaliar o desempenho da máquina foi limitado ao valor de 4 unidades funcionais por anel, pois é a partir deste limite que começa a ocorrer a diminuição do ganho em desempenho da máquina, de acordo com os gráficos das Figuras 42a e 42b. Ou seja, como ambas as versões da máquina, de anel único e múltiplos anéis, já estão próximas do limite assintótico para esta quantidade de recursos, então a adição de ainda mais unidades funcionais definitivamente não teria como acrescentar qualquer análise relevante. Além disso, conforme destacado no gráfico da Figura 40, esta é a quantidade máxima de recursos que pode ser disposta dentro dos *chips* de **FPGAs** utilizados neste projeto.

Ainda foi verificada a capacidade de escalabilidade real da máquina que pode ser

vista nos gráficos da Figura 43. Para tanto, foi analisado o *speedup* da máquina como função da quantidade de elementos de processamento, para a execução de ambos os programas utilizados para validação, operando sobre valores inteiros apenas. Aqui, não foi adotado nenhum padrão específico para a determinação da distribuição de recursos dos pontos escolhidos, sendo tomado apenas o melhor desempenho possível para cada quantidade de unidades funcionais. Apenas como exemplo, para a execução do programa de integração com 10 unidades funcionais na composição da máquina, foi tomada a distribuição com 5 anéis, 2 unidades funcionais por anel e 15 dutos de largura de banda. De forma equivalente, para a execução do mesmo programa com 50 unidades, foi utilizada a distribuição com 5 anéis, 10 unidades funcionais e 14 dutos de largura de banda em cada anel. Os gráficos apresentam a capacidade de escalabilidade da máquina, que nestes resultados se mostrou capaz de atingir o limite de 100 unidades de processamento. Apesar de seguir a baixa capacidade para ganho em *speedup*, o programa de integração atingiu seu limite assintótico para valores que flutuam ao redor de 12, chegando em seu limite superior para o valor 13.09. Este resultado se mostra coerente com a aproximação indicada nos gráficos da Figura 42, que indicava o máximo *speedup* para o programa de integração próximo ao valor ~ 12.80 , no caso analisado com a maior quantidade de recursos, de 4 unidades funcionais por anel, e portanto próximo do limite assintótico na análise de anel único, a partir da aproximação do *speedup* efetivo. De acordo com os valores identificados neste gráfico, é factível indicar este resultado como suficiente para validar a aproximação do *speedup* efetivo, como definido neste trabalho.

Os pontos finais dos gráficos, para quantidades superiores a 80 unidades funcionais possuem um caráter de decaimento na evolução do *speedup* pelos mesmos fatores verificados na π **FlowBW**. Devido a limitação identificada no processo de compilação do **BSV**, não é possível realizar a simulação de soluções com mais recursos de dutos de largura de banda. Dessa forma, para estes pontos que necessitam de mais recursos, uma vez que foi limitada a quantidade de anéis do **Multianel**, não foi possível encontrar as distribuições em dutos de largura de banda que fossem suficientes para apresentar os melhores resultados possíveis. Para contornar este problema poderiam ter sido analisadas distribuições com mais anéis. Entretanto, estas outras configurações não foram verificadas.

O formato logarítmico dos gráficos foi inspirado por diversas publicações de resultados de *speedup* da literatura como por exemplo da máquina **Sigma-1**.⁶ Ele ajuda a destacar o trecho que se aproxima do formato ideal do *speedup*, acumulando o caráter assintótico em um espaço menor do gráfico e por isso foi utilizado para representar este resultado consolidado que apresenta a capacidade de escalabilidade da máquina com melhor qualidade a fim de enaltecer o trecho que se aproxima do caráter ideal desta métrica.

O alcance precoce do formato assintótico destes gráficos tem relação direta com a deficiência verificada nos módulos de comunicação, bem como a proximidade entre as curvas

(a) Evolução do desempenho na primeira versão da π **FlowMR** para a execução do programa do fatorial modificado.

(b) Evolução do desempenho na primeira versão da π **FlowMR** para a execução do programa de integração.

Figura 43 – Curva de *speedup* consolidada na primeira versão dos módulos de comunicação da π **FlowMR**. Para a elaboração destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel.

Fonte: Elaborada pelo autor.

que representam os diferentes tamanhos de programas que tem bastante dependência com as instruções utilizadas no pré-processador.

6.6.2 Validação do Método

Até esta etapa da pesquisa ainda não havia sido possível atestar a viabilidade do **Multianel** com a implementação proposta, apesar de já contar com resultados que apresentavam melhores capacidades para ganho em desempenho e escalabilidade. Entretanto, os resultados obtidos a partir da **MDFM** eram capazes de apresentar melhores resultados em *speedups*,^{44,51} mesmo em uma estrutura de anel único. Dessa forma, foram buscadas alternativas que melhorassem essa métrica, independentemente da implementação. Foi proposto então um novo formato de comunicação entre os anéis da estrutura do **Multianel**, formato este, impraticável para uso em **FPGAs** comerciais, uma vez que a quantidade de recursos necessários para fazê-lo se tornou dinâmico. Cada anel da máquina passou a se comunicar com seus vizinhos mais próximos a partir da comunicação direta de seus dutos de largura de banda, lembrando que o número de dutos é definido em tempo de compilação da solução. Assim, todos os dutos de largura de banda de um anel são conectados aos dutos de ambos os vizinhos mais próximos a ele, de forma que na ocorrência de uma ficha destinada a um outro anel da estrutura, ela estará disponível ao anel vizinho imediatamente, no próximo ciclo do relógio, já dentro do duto de largura de banda destinado à evolução do resgate de informações de sua instrução. Ainda foi mantido o circuito que realiza a análise do menor caminho entre o anel corrente e o anel de destino de determinada ficha, como detalhado na Seção 5.1.

Este é o limite máximo em *throughput* para comunicação possível de ser implementado utilizando a topologia de comunicação proposta, em formato de anel. Além disso, em todas as análises realizadas, as instruções de soma e subtração foram removidas do pré-processador. Em uma das versões analisadas, foram mantidas nesta estrutura de pré-processamento apenas instruções de controle de fluxo, que possuem uma quantidade de fichas bastante limitadas durante a execução de programas, principalmente os programas principais utilizados para a validação da máquina. A ocorrência destas fichas particulares é observada principalmente no início e final da execução destes programas, quando não existe uma quantidade apreciável de paralelismo em execução. A partir desta estratégia, era esperado que os principais gargalos verificados no estado anterior do desenvolvimento tivessem sido removidos.

A remoção das instruções mais frequentes, do pré-processador, foi suficiente para apresentar um desempenho bastante superior, quando se considera a curva que apresenta o desempenho da máquina em um único anel de acordo com os gráficos das Figuras 44a e 44b, que em comparação com os resultados anteriores, mostrados nas Figuras 42a e 42b foi capaz de apresentar capacidades para escalabilidade muito melhores. Os gráficos não

(a) π FlowBW sem Pré-Processador. (b) π FlowBW com Pré-Processador mínimo. (c) π FlowMR com 2 unidades funcionais e sem Pré-Processador. (d) π FlowMR com 2 unidades funcionais e Pré-Processador mínimo. (e) π FlowMR com 4 unidades funcionais e sem Pré-Processador. (f) π FlowMR com 4 unidades funcionais e Pré-Processador mínimo.

Figura 44 – Resultados da π FlowMR para o programa de integração, obtidos nas versões de validação da implementação, com a definição de recursos mínimos no pré-processador. É possível verificar, neste conjunto de resultados, que o *speedup* já apresenta um comportamento muito mais próximo do desejado.

Fonte: Elaborada pelo autor.

consideram nenhum recurso de comunicação, tratando-se exclusivamente do desempenho da nova π **FlowBW**, para a execução em um único anel, variando a quantidade de recursos presentes na solução. É possível verificar em detalhes o ganho em desempenho possível de ser alcançado com as modificações propostas para o funcionamento de cada anel da estrutura, modificações estas que ficaram conhecidas como solução de passagem direta. Note ainda o impacto da presença de instruções mais significativas dentro do pré-processador da solução. É possível verificar o ganho em desempenho a partir da comparação dos resultados destacados nestes gráficos, com os resultados verificados no gráfico da Figura 39, que apesar de tratar-se da análise sobre a execução do programa do fatorial modificado, a curva se apresenta bastante diferente. É interessante observar que a proximidade entre as curvas para diferentes tamanhos de programas, mostram que mesmo os programas com menor capacidade de paralelismo, acabam se aproveitando melhor dos recursos da máquina, reforçando que os gargalos verificados nas versões antigas da implementação foram bastante mitigados pelas soluções de passagem direta.

Na Figura 44, os gráficos a esquerda referem-se à implementação da solução de validação sem o uso de qualquer instrução dentro do sub-módulo de pré processamento, enquanto que os gráficos mostrados à direita referem-se a implementação utilizando apenas os recursos destinados para o controle de fluxo. Ainda analisando os gráficos das Figuras 44a e 44b, existe um pequeno detalhe, qual seja, que para a execução de programas que possuem menor quantidade de instruções, o uso do pré-processador acaba oferecendo menor capacidade para ganho em desempenho, analisando cautelosamente as curvas mais baixas destes gráficos. Esta característica já era esperada a priori, uma vez que a presença do pré-processador foi identificada como um dos principais gargalos para ganho em desempenho dentro da implementação anterior. A execução dessas instruções dependem exclusivamente da distribuição dos programas dentro dos dutos de largura de banda do anel, garantindo a exploração de um outro nível de paralelismo, que naturalmente se apresenta prejudicial para a exploração dos recursos de processamento efetivos da máquina, para a análise do *speedup*.

De acordo com a figura, analisando o desempenho do **Multianel**, com o uso de 2 unidades funcionais em cada anel, como destacado nas Figuras 44c e 44d, é possível notar uma pequena vantagem na versão da implementação com o uso de recursos mínimos no pré-processador, evidenciando que apesar de apresentar pouca evolução no ganho em desempenho, esta solução de passagem direta se apresenta importante para a exploração de paralelismo. A exploração deste recurso é estritamente relacionada ao tamanho do programa em execução, principalmente ao considerar a implementação dos programas utilizados para validação, que possuem uma evolução de paralelismo bastante “comportada”, conforme mostrado na Figura 36. As instruções que podem se aproveitar do recurso de pré-processamento são muito bem definidas e podem ser descobertas facilmente mesmo antes da sua execução. Com o uso de apenas 2 unidades funcionais por anel, a principal

diferença pode ser observada nos menores programas analisados, para o uso de mais anéis na estrutura da máquina. A variação observada não é muito significativa, mas, nas condições em que é observada, é possível notar que quanto menos instruções se aproveitam dessa capacidade, melhor será o ganho em desempenho. Reforçando que a exploração do pré-processador deve ser realizada de forma bastante cautelosa, a fim de obter níveis apreciáveis para a análise desta métrica. Perceba ainda, que no âmbito do **Multianel**, essa característica se inverte portanto. Afinal, na evolução do desempenho de um único anel, a presença do pré-processador se apresenta prejudicial ao ganho em desempenho, enquanto que no **Multianel**, sua presença é benéfica. Isto se dá pelo fato de que no **Multianel**, existem múltiplos pré-processadores por duto que podem ser explorados paralelamente, enquanto que com um único anel, existe apenas um pré-processador por duto, o que força a execução sequencial quando verificada.

Tabela 8 – *Speedup* Efetivo, para a execução do programa de integração, com profundidade 11, em ambas as versões do programa.

Pré-Proc.	# FUs	# Anéis	# Ciclos	Speedup	Speedup Ef.
Ausente	1	1	192679	1.00	–
	2	1	96556	1.99	–
	4	1	48659	3.96	–
	2	10	13214	7.31	14.55
	4	10	9748	4.99	19.76
Mínimo	1	1	184494	1.00	–
	2	1	92512	1.99	–
	4	1	46609	3.96	–
	2	10	12557	7.38	14.69
	4	10	6939	6.72	26.61

Fonte: Elaborada pelo autor

Quanto a análise do *Speedup* Efetivo, é possível verificar uma evolução real nesta versão de validação, que já apresenta níveis realmente apreciáveis para ganho em desempenho pela estrutura construída. Na Tabela 8 é possível verificar *speedups* que ultrapassam todos os valores obtidos até este momento do desenvolvimento. Entretanto, a análise dessa nova métrica é suficiente apenas para indicar um limite inferior do *speedup* possível de ser alcançado, uma vez que foram atingidos valores superiores àqueles dispostos na tabela, durante a realização da análise consolidada, dos gráficos mostrados na Figura 45. Na tabela são mostrados os valores desta métrica para ambas as versões analisadas, destacando os *speedups* obtidos para a elaboração dos gráficos da Figura 44.

Nas Figuras 44e e 44f é possível verificar as principais motivações para a utilização do pré-processador no **Multianel**. Apesar da diferença pequena vista nos gráficos anteriores, neste ponto é possível verificar que este submódulo é de extrema importância para o ganho em desempenho da máquina, com a utilização de mais recursos em cada anel da solução. Aqui, a distância entre as curvas de *speedup* para diferentes tamanhos do programa

mostram a exploração dos recursos paralelos da máquina durante a execução, trazendo de volta os resultados obtidos ainda nas análises das primeiras versões da π **FlowBW**. Torna-se visível este retorno da exploração massiva das unidades funcionais, de acordo com a disponibilidade de paralelismo presente no programa em execução. A partir do uso de 4 anéis, a diferença entre as curvas se torna bastante evidente.

No **Multianel**, o tempo total de execução dos programas se torna cada vez menor, com isso, o ganho em desempenho promovido pelo módulo de pré-processamento é evidente. As instruções presentes na composição deste módulo se dão principalmente no início e final de programas, quando não há muito paralelismo disponível para exploração. Com isso, a execução destas instruções, realizadas de forma mais imediata, acaba trazendo uma grande diferença na execução global dos programas. A execução do programa de integração, com profundidade de 11 níveis, operando apenas sobre valores inteiros, em uma composição da máquina com 1 anel e 1 unidade funcional leva o período total de 184494 ciclos. Em comparação, para a execução do mesmo programa, com o uso de mais recursos, como por exemplo para 10 anéis e 4 unidades funcionais, de acordo com o gráfico da Figura 44f, o período total para a execução se dá em apenas 6939 ciclos. Esta verificação reforça que qualquer estratégia que ofereça vantagens para a execução mais rápida de instruções, ou tratamento diferenciado para operação das fichas, acaba se tornando fundamental para a escalabilidade, sob o ponto de vista do *speedup* no **Multianel**.

Os gráficos das Figuras 45 mostram a capacidade da máquina para escalabilidade aliado a um ganho em desempenho mais significativo. Nestes gráficos foram utilizados os mesmos programas de integração, operando sobre valores inteiros. É importante observar que sem o uso do pré-processador, apresentado no gráfico da Figura 45a, o limite assintótico da curva flutua ao redor de valores próximos a 20, ainda bastante consistente com a análise do *speedup* efetivo mostrado na Tabela 8, que apresentava um *speedup* máximo de exatos 19.76, para a execução do programa de maior tamanho.

Novamente, apenas para fins de comparação, para a execução do maior programa apresentado na figura, com apenas 1 anel e 1 unidade funcional na composição da máquina, a quantidade de ciclos necessários para sua execução é de 192679 ciclos. Já o período para a execução do mesmo programa, com o uso de 10 anéis e 10 unidades funcionais presentes em cada anel, é de apenas 9390 ciclos. E apesar deste grande ganho em desempenho, o limite assintótico ainda é atingido de forma precoce. É importante verificar que a flutuação deste limite se torna mais acentuada quando analisada em valores absolutos. Afinal, para o uso de 5 anéis, com 7 unidades funcionais, o programa é executado em apenas 8825 ciclos.

Com relação ao gráfico disposto na Figura 45b, é possível verificar a melhor capacidade para ganho em desempenho da máquina, dentre todas as versões implementadas até este momento, pois aqui é alcançado o valor de 40 em *speedup*. As curvas apresentam uma distância considerável entre os diferentes tamanhos do programa, mostrando uma

(a) Comportamento na evolução do desempenho na versão de validação da $\pi\mathbf{FlowMR}$ sem o uso de qualquer instrução no Pré-Processador da solução.

(b) Comportamento na evolução do desempenho na versão de validação da $\pi\mathbf{FlowMR}$ com o uso de recursos mínimos na solução do Pré-Processador.

Figura 45 – Comportamento da curva de *speedup* consolidada nas versões de validação da $\pi\mathbf{FlowMR}$. Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução do programa de integração operando sobre valores inteiros apenas.

Fonte: Elaborada pelo autor.

boa distribuição de carga entre os elementos de processamento da máquina. Novamente destaca-se a importância do módulo de pré-processamento que foi mantido na versão final da implementação do **Multianel**, que em comparação com o gráfico anterior, mostra que sua utilização acaba por mitigar a subutilização das unidades funcionais, pelo fato dessas instruções particulares serem executadas de forma imediata nos múltiplos anéis da máquina. Os valores atingidos em *speedup*, se distanciam consideravelmente dos valores do *Speedup* Efetivo destacados na Tabela 8. Isso se dá pelo fato de que o ganho em desempenho atingido a partir dessa versão continua em ascensão após o limite de 40 unidades funcionais diferentemente dos comportamentos obtidos anteriormente. Para valores próximos a esta quantidade de recursos o *speedup* flutua ao redor do valor 26.61, conforme previsto pela análise desta nova métrica, o que na verdade torna a validá-la.

Note portanto, que os valores obtidos a partir da métrica do *Speedup* Efetivo tem a finalidade de verificar o *speedup* real, possível de ser alcançado a partir das análises dispostas nos gráficos que apresentam a evolução do desempenho da máquina com relação a variação na quantidade de anéis presentes em sua estrutura. Análise esta, que como destacado anteriormente, possui melhores argumentos para a avaliação de uma aplicação prática desta solução. Lembrando que para a determinação desta nova métrica foi utilizado o conceito amplo destacado anteriormente, do qual ela passou a ser avaliada com relação a sua versão já paralela, contrário à sua definição. Note portanto, que essa ferramenta se mostra bastante útil a fim de avaliar o desempenho real em *speedup*, a partir desta análise intermediária e que ainda dependia dos experimentos propostos pela presente discussão para ser tomado como uma ferramenta plausível.

6.6.3 Versão Final

Finalmente, a partir de todos os resultados e conclusões obtidos nas seções anteriores, que foram suficientes para comprovar a viabilidade do **Multianel**, foram realizadas as modificações definitivas na proposta deste trabalho. Os módulos de pré-processamento foram utilizados de acordo com as definições mostradas anteriormente. O módulo de comunicação foi implementado de acordo com a descrição detalhada mostrada na Seção 5.1, que utilizou a estratégia dos dutos de largura de banda para o sincronismo entre a máquina e os módulos de comunicação. Dessa forma, com o acúmulo de fichas em uma fila intermediária nos módulos de frequência de operação mais alta, passou a ser possível a exploração dos módulos de comunicação do **Multianel** de forma mais eficiente. Formato de implementação que possibilitou a verificação de um melhor ganho em desempenho, agora já na versão de implementação definitiva da máquina.

Os gráficos mostrados na Figura 46 apresentam os resultados obtidos para a execução de ambos os programas utilizados neste trabalho, o programa de fatorial modificado, com vários tamanhos, cujos resultados são mostrados do lado esquerdo da figura, e o

programa de integração operando apenas sobre valores inteiros, mostrado do lado direito da figura. É interessante notar no gráfico da Figura 46a que a curva obtida para um único anel é bastante consistente com os resultados verificados no gráfico da Figura 39, com o alcance precoce do limite assintótico do programa próximo a quantidade de 10 unidades funcionais. Com uma pequena melhora para ganho em desempenho, proveniente exclusivamente pelas soluções de passagem direta implementadas nesta versão do projeto. Além disso, a proximidade entre as curvas do gráfico ainda se mostra consistente com a análise anterior que apresentava os resultados para programas de tamanho máximo igual a apenas 1000 chamadas recursivas. Aqui o tamanho máximo explorado refere-se a este mesmo programa utilizando de 1500 chamadas. Portanto, mostrando grande consistência entre as implementações distintas. Afinal, para a exploração de um tamanho de programa igual a 100, conforme aquela análise, é esperado um comportamento perfeitamente equivalente, de acordo com essa nova avaliação deste programa.

É importante destacar ainda, que este não é o limite máximo para ganho em desempenho deste programa, como pode ser visto nos gráficos seguintes, a partir da análise do *Speedup* Efetivo. Este alcance precoce do limite assintótico ocorre para a execução deste programa principalmente devido a sua distribuição nos dutos de largura de banda, indicando certo acúmulo de instruções em dutos específicos, impedindo um melhor ganho em desempenho. É possível que existam melhores distribuições acima dos valores limites trabalhados no decorrer desta análise para quantidades superiores a 20 dutos de largura de banda, o que seria suficiente para obter um *speedup* assintótico mais tardio. Porém, essa verificação não foi realizada, devido a limitação verificada na linguagem, que impedia a compilação da solução para distribuições mais robustas, e também pelo fato de a estratégia do **Multianel** já ser suficiente para um ganho em desempenho mais expressivo para a execução deste programa, se aproveitando de sua capacidade para uma melhor distribuição de recursos dentro da estrutura da máquina.

Já no gráfico da Figura 46b é possível verificar que o limite assintótico não pode ser verificado dentro dos limites na quantidade de unidades funcionais aqui praticadas. O perfil desta curva apresenta um dos melhores resultados exibidos nas implementações deste projeto até este momento. A proximidade entre as curvas de *speedup* para programas com tamanhos que variam em uma faixa considerável de valores, com uma variação praticamente constante de paralelismo médio, destaca a capacidade de distribuição de recursos da máquina. É possível, a partir destes resultados, levantar a hipótese de que para a execução de programas que apresentem capacidades ainda melhores de paralelismo, estes valores pudessem ser ainda mais otimizados, se aproximando tanto quanto necessário do comportamento ideal desta métrica, mesmo para a exploração do sistema de único anel, até o limite de 10 unidades funcionais. A análise dos resultados em valores absolutos já se mostra surpreendente mesmo nestes primeiros resultados. A execução do programa de integração com tamanho de 11 níveis, para o uso de apenas 1 unidade funcional e 4 dutos

de largura de banda, é de 184570 ciclos de relógio, enquanto que a execução deste mesmo programa com o uso de 10 unidades funcionais e 18 dutos de largura de banda é de apenas 19564, resultando no *speedup* de 9.43 que pode ser visto no gráfico.

Nos gráficos das Figuras 46c e 46d pode-se verificar um comportamento bastante semelhante àqueles obtidos na versão de validação da máquina, para o uso de 2 unidades funcionais em cada anel, com algumas peculiaridades interessantes. O gráfico da Figura 46c é totalmente original, não tendo sido verificado anteriormente e nele pode-se perceber uma evolução constante e ascendente para ganho em desempenho no **Multianel**, com o uso mais limitado de recursos, para o uso de 2 unidades funcionais por anel, considerando a execução de um programa mais simples, com uma grande quantidade de operações de controle e aritméticas operando sob valores inteiros apenas, como pode ser verificado na implementação do programa apresentado no Apêndice A.2. O *Speedup* Efetivo mostrado na Tabela 9, tem um mínimo de 14.43, quando da exploração de 20 unidades funcionais. Resultado ainda coerente com a extrapolação realizada durante as implementações das soluções de passagem direta, presentes no gráfico da Figura 41, que apesar de tratar deste programa em sua versão de menor tamanho, apresenta uma boa aproximação que não considerava qualquer gargalo de operação, proveniente principalmente das operações de comunicação entre os anéis que compõem a estrutura da máquina, e mesmo que ainda contava com as operações aritméticas mais frequentes no sub-módulo de pré-processamento, recurso que se mostrou bastante prejudicial para ganho em desempenho.

Tabela 9 – *Speedup* Efetivo, para a execução do programa do fatorial modificado com tamanho igual a 1500 e de integração, com tamanho distribuído em 11 níveis, na versão final da implementação.

Progr.	# FUs	# Anéis	# Ciclos	Speedup	Speedup Ef.
Fatorial	1	1	162195	1.00	–
	2	1	81377	1.99	–
	4	1	41156	3.94	–
	2	10	11221	7.25	14.43
	4	10	6536	6.30	24.82
Integração	1	1	184570	–	–
	2	1	92578	1.99	–
	4	1	46677	3.95	–
	2	10	12701	7.29	14.51
	4	10	7233	6.45	25.48

Fonte: Elaborada pelo autor

É importante destacar que a Figura 46c apresenta uma das únicas análises do **Multianel** capaz de apresentar uma tendência de evolução em *speedup* ainda crescente para esta quantidade de anéis, indicando uma fonte de recursos que poderia ser mais explorada, apesar de sua verificação não ter evoluído a partir deste ponto. Afinal, mesmo atingindo essa capacidade, sua evolução é mais lenta em comparação com os outros programas

(a) π FlowBW executando o programa do Fatorial modificado.

(b) π FlowBW executando o programa de integração.

(c) π FlowMR com 2 unidades funcionais, programa do Fatorial.

(d) π FlowMR com 2 unidades funcionais, programa de Integração.

(e) π FlowMR com 4 unidades funcionais, programa do Fatorial.

(f) π FlowMR com 4 unidades funcionais, programa de Integração.

Figura 46 – Conjunto de resultados da versão final da π FlowMR para os programas de fatorial e integração operando sobre valores inteiros apenas. É possível verificar, neste conjunto de resultados, que o *speedup* se aproxima ainda mais do comportamento ideal.

Fonte: Elaborada pelo autor.

utilizados para validação, como pode ser visto imediatamente no gráfico da Figura 46d. Além disso, é interessante observar a consonância deste gráfico com sua versão em anel único mostrado na Figura 46a que possui características de evolução bastante semelhantes, destacando que os módulos de comunicação acabaram por adicionar uma complexidade menos acentuada que as versões anteriores da construção da máquina.

De acordo com o resultado mostrado no gráfico da Figura 46d é possível verificar um comportamento perfeitamente equivalente àquele obtido na versão de validação da máquina, da Figura 44d, apenas com variações muito pequenas entre ambos os gráficos, principalmente para o programa de menor tamanho. Entretanto, é muito importante lembrar que a versão de validação da máquina foi implementada com características que destacam o limite máximo de *throughput* em comunicação, possível de ser alcançado a partir da topologia em anel utilizada. A grande proximidade entre estes resultados destacam um aspecto importante da estrutura desta máquina, como descrito em detalhes no trabalho.⁴³ De acordo com esta constatação verificada ainda na **MDFM**, o controle do paralelismo para o fornecimento de fichas à estrutura da máquina, pode ser benéfica para o ganho em desempenho global. Esta propriedade é verificada principalmente devido ao uso crescente do armazenamento na unidade de pareamento, durante a execução de um programa, unidade esta, que naturalmente possui um desempenho mais lento, por contar com a necessidade de leitura e escrita em sua memória **RAM** de acordo com a verificação de uma função *hash* para seu endereçamento. Diversas estratégias do presente projeto se mostraram bastante otimizadas para mitigar este gargalo, como por exemplo o uso das duas portas de comunicação para permitir a leitura e escrita nesta memória simultaneamente. E ainda assim, é possível verificar que a estratégia do *Throttle* ainda pode ser utilizada sobre o contexto do **Multianel** no presente trabalho. Esta constatação pode ser analisada por este gráfico, pois seu desempenho se mostra perfeitamente equivalente à sua versão de implementação com melhor capacidade para comunicação entre anéis. Esta análise destaca que os gargalos provenientes da necessidade de comunicação de fichas entre os anéis da máquina em sua versão com desempenho mais “limitado”, pode oferecer a mesma capacidade para ganho em desempenho que a versão com a taxa de comunicação entre anéis mais rápida. Este fato observado é bastante contraintuitivo, mas pode se aproveitar da teoria para controle de paralelismo para ser explicado e ainda se mostra suficiente para re-validar esta teoria já consolidada na literatura.

Ainda sobre o gráfico da Figura 46d, o desempenho apresentado se mostra bastante superior com relação a primeira versão da implementação, destacando o sucesso da nova solução de comunicação proposta. A proximidade moderada entre as curvas destacam a boa distribuição e exploração dos recursos de processamento da máquina e apresentam um *Speedup* Efetivo de 14.51 para o uso de 20 unidades funcionais, como mostrado na Tabela 9, e ainda coerentes com os resultados mostrados nos gráficos da Figura 47.

Os gráficos das Figuras 46e e 46f apresentam um perfil equivalente aos resultados anteriores, apresentando um ganho em desempenho mais moderado, a partir do uso de mais elementos de processamento na composição da máquina. Novamente, destacando que apesar da boa distribuição dos programas entre os anéis, ainda existe um grande desperdício para o uso de mais recursos de processamento, fato esperado a priori, mas em todo caso, extremamente superior aos resultados obtidos inicialmente. A semelhança entre os perfis encontrados nestes gráficos, com relação àqueles que exploram menos recursos, mostrados nas Figuras 46c e 46d, em comparação com os gráficos das Figuras 42e e 42f da primeira versão da implementação dos módulos de comunicação, mostram a evolução obtida a partir das soluções definitivas na implementação deste projeto. Nesta análise, conforme mostrado na Tabela 9 foram atingidos os valores de 24.82 e 25.48 em *Speedup* Efetivo, para os programas de fatorial modificado e de integração operando sobre valores inteiros, respectivamente. A proximidade entre estes valores inclusive já destacam que o ganho em desempenho nestes programas ainda não atingiu seu limite assintótico dentro dos limites de 40 unidades funcionais. Comportamento atingido apenas na versão de validação da implementação, até este momento da pesquisa. Nesta versão da máquina, a quantidade de ciclos de relógio tomados para a execução de programas, que podem ser vistos na tabela do *Speedup* Efetivo para o uso máximo de até 40 unidades funcionais se mostram bastante surpreendentes, indicando um ganho em desempenho de 96% em ambos os programas, chegando a limites inferiores a 8000 ciclos do relógio para sua execução completa. Isso mostra que o módulo de pré-processamento e as principais soluções de passagem direta são fundamentais para a execução destes programas, uma vez que sua execução é bastante imediata diante da larga exploração de paralelismo possível de ser alcançada a partir da distribuição de recursos proposta pelo **Multianel**.

Nos gráficos da Figura 47 é possível verificar novamente a capacidade de escalabilidade da máquina. Foram tomadas as distribuições de recursos capazes de oferecer os melhores desempenhos, independentemente da composição da máquina. Os comportamentos verificados são perfeitamente coerentes com as implementações de anel único, como visto nos resultados iniciais.¹⁸ Porém, aqui, ambos os programas são capazes de atingir valores em *speedup* muito maiores. O programa do fatorial modificado é capaz de atingir valores máximos ao redor de 33, número impossível de ser obtido na implementação de anel único, apesar de apresentar um perfil de curva que se aproxima bastante daquela análise, para os diferentes tamanhos de programas, como pode ser visto no gráfico da Figura 39. A distância entre estas curvas, bem como a consonância entre elas, reforçam a boa distribuição do programa dentro dos recursos de processamento da máquina, independente de seu tamanho. No gráfico da Figura 47a é possível observar ainda a capacidade oferecida pelo *Multianel* para ganho em desempenho em comparação com sua versão de anel único, que atingia seu limite assintótico para até 10 unidades funcionais, mas aqui, este valor é ultrapassado com um comportamento linear e ascendente, se aproveitando da distribuição

(a) Desempenho na versão final da π FlowMR para a execução do programa do Fatorial. (b) Desempenho na versão final da π FlowMR para a execução do programa de Integração.

Figura 47 – Curva de *speedup* consolidada na versão final da π FlowMR. Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução dos programas do Fatorial e Integração operando sobre valores inteiros apenas.

Fonte: Elaborada pelo autor.

das instruções do programa entre os anéis, provando a capacidade do **Multianel** em remover gargalos provenientes principalmente do acúmulo de fichas em determinados dutos de largura de banda.

Nos gráficos da Figura 47, é possível verificar um fato que, apesar de esperado e bem explorado na literatura, se mostra interessante e evidente na relação do *speedup* com o paralelismo médio do programa. Nas curvas referentes à execução dos menores programas, os valores de paralelismo médio são equivalentes e além disso, os perfis das curvas também, com o *speedup* atingindo valores máximos próximos a 20 e 30 respectivamente, e apresentam um perfil de evolução igualmente equivalente. Mesmo em pequenas variações, é possível perceber que algumas flutuações esperadas - por terem sido verificadas em outras versões da implementação - se mostram perfeitamente equivalentes entre os dois gráficos. Lembrando que os pontos utilizados para a elaboração das figuras foram escolhidos de acordo com os melhores desempenhos observados, que são obtidos a partir da melhor distribuição de recursos da máquina. De forma que esta equivalência observada entre programas totalmente distintos, não é um comportamento intuitivo, mas se mostra bastante evidente nestes gráficos, na análise do *speedup* em conjunto com o paralelismo médio. Já o programa de integração de maior tamanho possui um valor de paralelismo médio maior do que no programa do fatorial, e seu ganho em desempenho é igualmente superior, atingindo o limite máximo de 40. Este valor já havia sido verificado na versão de validação da máquina, e foi tomado como fator determinante para a permanência do sub-módulo de pré-processamento na estrutura do **Multianel** e é o mesmo encontrado na implementação definitiva, reforçando inclusive a análise do efeito de *throttle* destacado anteriormente. Portanto, é interessante verificar como o paralelismo médio é suficiente para inferir o comportamento do *speedup*.

Estes gráficos apresentam valores ligeiramente distintos com relação aos resultados esperados, de acordo com a Tabela 9, e com os gráficos da Figura 46. Ao longo desta análise, é possível verificar que o *Speedup Efetivo* é capaz de apresentar melhor acurácia em cenários quando o programa já atingiu seu limite assintótico no **Multianel**. Esta constatação pode ser verificada nas discussões referentes aos resultados mostrados na primeira versão da implementação dos módulos de comunicação, nos gráficos da Figura 42, quando este limite era verificado muito precocemente e o valor desta métrica tinha melhor capacidade para avaliar o ganho em desempenho real da máquina. Entretanto, da versão de validação da máquina em diante, o limite assintótico passou a ser obtido mais tardiamente, característica buscada durante todo o desenvolvimento do projeto. Entretanto, a métrica do *Speedup Efetivo* como definido, pode ser tomado apenas como uma boa aproximação para a determinação do limite inferior mínimo do *speedup* que a implementação é capaz de alcançar. Apesar de uma boa aproximação, esta métrica desconsidera quaisquer gargalos provenientes da necessidade de comunicação entre os anéis da máquina. Dessa forma, para fins de análises mais precisas, a definição do *Speedup Efetivo* deveria considerar

um fator proveniente destes gargalos. Porém, como o foco principal deste trabalho era a verificação do método aplicado aos limites de tecnologias disponíveis, o método de verificação da escalabilidade se mostrou suficiente para a análise do ganho em desempenho real da máquina, de forma que a expressão para obtenção desta nova métrica ficou limitada realmente a verificação deste limite inferior apenas. A fim de obter melhor precisão para esta medida, poderiam ser exploradas distribuições com mais unidades funcionais por anel, para a elaboração dos gráficos da Figura 46, para quantidades de recursos em que o limite assintótico já tivesse sido encontrado. De acordo com a análise descoberta sobre esta métrica, esta forma de aplicação já seria suficiente para oferecer valores mais precisos sobre o *speedup* da máquina.

Finalmente, os gráficos da Figura 48 apresentam os resultados obtidos para a execução do programa de integração operando sob valores em ponto flutuante a esquerda da figura, bem como sua versão operando sob operações trigonométricas, a direita. Estes programas possuem as melhores características para ganho em desempenho de acordo com a estratégia do fluxo de dados, pois algumas de suas principais instruções possuem um tempo de execução não homogêneos, garantindo melhor utilização das unidades de processamento. Dessa forma, as unidades funcionais alocadas para a execução destas instruções específicas mantêm-se ocupadas por um período de tempo maior, enquanto que as outras unidades seguem a execução massiva das instruções de controle dos programas, de granularidade extremamente fina característica deste modelo, garantindo um melhor aproveitamento destas unidades.

Nos gráficos das Figuras 48a e 48b é possível atestar a hipótese levantada anteriormente, sobre a capacidade de se aproximar tanto quanto desejado do comportamento ideal do *speedup*, a partir da exploração de programas que apresentam maior capacidade de paralelismo. De fato, é possível perceber que o ganho em desempenho para a execução dos programas de maior tamanho se aproximam ainda mais deste comportamento ideal. O limite assintótico do *speedup* destes programas não se encontra dentro do intervalo utilizado para a composição destes gráficos, para nenhum tamanho de programa analisado. Estes resultados demonstram a capacidade da máquina para a exploração de um paralelismo massivo, mesmo operando sobre a abordagem de anel único, apresentando as melhores capacidades para ganho em desempenho que foram possíveis de serem obtidas durante todo o desenvolvimento deste trabalho.

Os gráficos das Figuras 48c e 48d mostram uma ligeira melhora para ganho em desempenho, em comparação com a execução do mesmo programa operando sobre valores inteiros, visto na Figura 46d. É possível verificar que no intervalo entre as quantidades de 4 e 8 anéis o *speedup* se aproxima ainda mais do comportamento ideal, mas os programas utilizados para a composição destes gráficos possuem paralelismo mais alto e portanto, este comportamento já era esperado. Porém, para quantidades de anéis superiores a 8

(a) π FlowBW executando o programa de Integração sob Ponto Flutuante.

(b) π FlowBW executando o programa de Integração com operações trig.

(c) π FlowMR com 2 unidades funcionais, Integração sob Ponto Flutuante.

(d) π FlowMR com 2 unidades funcionais, Integração com operações trig.

(e) π FlowMR com 4 unidades funcionais, Integração sob Ponto Flutuante.

(f) π FlowMR com 4 unidades funcionais, Integração com operações trig.

Figura 48 – Conjunto de resultados da versão final da π FlowMR para os programas de integração operando sobre valores em ponto flutuante. É possível verificar, neste conjunto de resultados, que o *speedup* se aproxima ainda mais do comportamento ideal desta métrica.

Fonte: Elaborada pelo autor.

unidades o comportamento assintótico já é atingido, apesar de sua verificação mais tardia em comparação com as outras versões. Considerando-se o *Speedup* Efetivo, mostrado na Tabela 10, os resultados atingem valores de referência ainda mais significativos, chegando novamente a valores de *speedup* que não haviam sido atingidos. Note-se nos valores da tabela que o aumento do *Speedup* Efetivo não se mostra muito significativo com relação aos resultados anteriores, da Tabela 9. Certamente, a realização desta análise aplicada em anéis com mais recursos de processamento, seria capaz de resgatar as variações obtidas desta métrica quando aplicada aos gráficos da Figura 42, para a execução destes programas que apresentam melhor paralelismo, como por exemplo, aplicando esta mesma análise para o uso de 6 ou 8 unidades funcionais por anel. Afinal, o *Speedup* Efetivo apresenta resultados mais significativos quando o programa avaliado já se encontra no estado assintótico para a quantidade de recursos de processamento explorado. Entretanto, esta verificação ficou limitada às quantidades determinadas no início desta seção, para o uso de até 4 unidades funcionais por anel, que segue um comportamento ideal, mesmo como pode ser verificado nos gráficos das Figuras 48a e 48b.

Tabela 10 – Valores do *Speedup* Efetivo para a execução dos programas de integração em ambas as versões que operam sob valores em ponto flutuante, obtidos na versão final da implementação.

Progr.	# FUs	# Anéis	# Ciclos	Speedup	Speedup Ef.
Ponto Flutuante	1	1	252093	1.00	–
	2	1	126414	1.99	–
	4	1	63613	3.96	–
	2	10	17075	7.40	14.73
	4	10	9368	6.79	26.89
Operações Trigonométrica	1	1	342271	1.00	–
	2	1	171429	1.99	–
	4	1	86201	3.97	–
	2	10	22626	7.58	15.08
	4	10	12139	7.10	28.19

Fonte: Elaborada pelo autor.

A análise dos valores absolutos na quantidade de ciclos de relógio para a execução destes programas com mais paralelismo, se torna ainda mais significativa. Para a execução do programa que opera sob funções trigonométricas, com a maior quantidade de recursos verificado (10 anéis, com 10 unidades funcionais e 15 dutos de largura de banda), foi possível atingir um limite de desempenho 98% superior à execução em anel único (1 anel, com 1 unidade funcional e 8 dutos de largura de banda), com o uso de apenas uma unidade de processamento. A execução deste programa na configuração de anel único é bastante limitada, devido ao acúmulo imenso de fichas que aguardam por sua execução, enquanto as operações trigonométricas ainda estão sendo executadas pelo único recurso de processamento disponível. Porém, é interessante verificar a capacidade para ganho em desempenho em um programa que explora os principais benefícios do paradigma

da computação em fluxo de dados, que apesar de ser definida para operar em uma granularidade mais fina, se mostra bastante eficiente em um ambiente de granularidade moderadamente mais grossa.

Entre 4 e 8 anéis nos gráficos das Figuras 48e e 48f, verifica-se uma maior aproximação do comportamento ideal do *speedup*. São bons resultados, quando comparados com os resultados anteriores, principalmente em comparação com a execução do programa operando sob valores inteiros, apresentado na Figura 46f. No entanto, para quantidades de anéis superiores a 8, a máquina atinge novamente seu limite para ganho em desempenho. Este resultado se dá principalmente devido a distribuição das instruções dos programas dentro de cada anel. É possível que existam melhores distribuições em dutos de largura de banda para a abordagem de anel único, capazes de oferecer um desempenho superior. Na π FlowMR de 8 anéis a maior distância entre quaisquer dois anéis, é de 4 saltos entre anéis, devido ao caráter bidirecional do processo de comunicação. Desta forma, considerando os principais resultados obtidos até este momento, é possível destacar este comportamento como uma limitação da estrutura proposta por este projeto. É possível que existam outras topologias de comunicação, que sejam capazes de ultrapassar este limite, e que sejam suficientes para atenuar este platô - atingido de 8 anéis em diante em todos os resultados - como pôde ser visto na Figura 46c, o qual apresenta uma evolução crescente a partir deste limite, ou até mesmo oferecer um crescimento perfeitamente linear para a evolução do ganho em desempenho a partir da adição de mais anéis à estrutura da máquina. Entretanto, este projeto se limitou a análise da topologia de comunicação em anel, principalmente por conta dos recursos físicos disponíveis para a implementação desta solução.

Nos gráficos consolidados, mostrados na Figura 49, pode-se verificar um desempenho melhor no que se refere à escalabilidade. Um destaque importante é o *speedup* máximo atingido, que para o programa que opera sob operações trigonométricas atinge um valor próximo a 60, neste programa que possui paralelismo médio de 430. Desta forma, percebe-se que o *speedup* máximo obtido para este tamanho do programa representa menos de 14% de seu paralelismo médio, em comparação explícita entre as duas métricas, uma vez que o paralelismo médio representa o *speedup* máximo possível de ser obtido pelo programa em uma máquina ideal. Já no programa com profundidade 9, o *speedup* máximo alcançado é de ~20, para o paralelismo médio de 129, o que representa pouco menos de 16% nesta mesma comparação entre as métricas. Ou seja, apesar de apresentar um ganho em desempenho inferior, como pode ser visto no gráfico, é possível que o programa de menor tamanho tenha seu paralelismo explorado de uma forma mais eficiente. Ao considerar os resultados anteriores, os menores programas utilizados apresentavam capacidades ainda melhores nesta comparação explícita, chegando em valores superiores a 20% do paralelismo médio. Apesar desta comparação tratar de condições de operação muito distintas, principalmente ao considerar o caráter ideal do paralelismo médio, ela pode indicar a existência de gargalos que não havia sido analisados anteriormente. Entretanto, neste projeto, esta extrapolação

(a) Comportamento na evolução do desempenho na versão final da π **FlowMR** para a execução do programa de Integração operando sob valores em ponto flutuante.

(b) Comportamento na evolução do desempenho na versão final da π **FlowMR** para a execução do programa de Integração com o uso de operações trigonométricas.

Figura 49 – Curva de *speedup* consolidada na versão final da π **FlowMR**. Para a composição destes gráficos foram tomados os melhores desempenhos da máquina como função da quantidade de unidades funcionais, independente da composição de recursos de cada anel para a execução dos programas de Integração operando sob ponto flutuante e com operações trigonométricas.

Fonte: Elaborada pelo autor.

entre as métricas utilizadas para validação da máquina acabaram se limitando a esta pequena avaliação.

Em linhas gerais, a evolução gráfica das curvas de *speedup* de ambos os programas são muito parecidas, apenas apresentando um pequeno ganho em desempenho para a exploração do programa que opera sob operações trigonométricas. A proximidade moderada entre as curvas de desempenho deste programa mostrado na Figura 49b, exibem um melhor aproveitamento de recursos e distribuição de carga entre os recursos de processamento da máquina.

6.6.4 Arquiteturas convencionais

Além das simulações feitas até aqui, foi realizado um teste menos rigoroso, a fim de verificar o desempenho do programa de integração em uma arquitetura convencional. Este programa foi executado em suas versões operando sobre valores inteiros e também com o uso de operações trigonométricas.

Para a execução destes programas foi utilizado um simulador desenvolvido por pesquisadores das universidades do MIT e de **Stanford**, que tinha como objetivo permitir a simulação de processadores com até milhares de núcleos de processamento, denominado **ZSim**.¹¹ Este simulador foi escolhido devido a seu desempenho durante o processo de simulação, que se mostrou muito mais otimizado para a simulação de processadores com arquitetura *x86* com múltiplos núcleos, como descrito em detalhes no trabalho que descreve o desenvolvimento da π **FlowMT**.⁷⁴

Para a elaboração deste teste, foi utilizado um número fixo de núcleos de processamento de 128 unidades, alterando-se apenas a quantidade de *threads* utilizadas para a execução dos programas. Algumas análises verificadas no trabalho citado sugerem que o impacto da alteração na quantidade de núcleos de processamento é muito pequeno em comparação com a variação na quantidade de *threads*, que explorarão os recursos de processamento de forma coordenada e sob demanda. Por este motivo que foi escolhida esta estratégia, de forma que a exploração dos recursos computacionais se torna equivalente as análises realizadas na π **FlowMR**. Para a compilação desta solução foi utilizado o **GCC 8**, com o uso do **OpenMP**, aplicando as diretivas convencionais desta ferramenta para a exploração dos recursos paralelos da máquina, aplicados sob as chamadas recursivas da implementação destes programas, como pode ser visto em detalhes no Anexo A.3. Além disso, foi utilizada a opção de compilação **-OFast** que gera código com praticamente todas as opções de otimização disponíveis neste compilador, que é bem documentado e largamente explorado para as mais diversas aplicações pela comunidade acadêmica.

É possível verificar pelas curvas da Figura 50 a baixa capacidade para a exploração do paralelismo disponível nestes programas, principalmente ao considerar uma comparação

(a) Comportamento na evolução do desempenho para a execução do programa de integração implementado de forma duplamente recursiva, operando sob valores inteiros apenas.

(b) Comportamento na evolução do desempenho para a execução do programa de integração operando sob valores em ponto flutuante, com o uso de operações trigonométricas.

Figura 50 – Comportamento da curva de *speedup* para a execução dos programas de integração operando sobre valores inteiros apenas e sob operações trigonométricas. Para a composição destes gráficos foram realizadas as execuções no simulador para diferentes quantidades de *threads*.

Fonte: Elaborada pelo autor.

explícita com os resultados mostrados nas Figuras 47b e 49b. Ambos os programas, para todos os tamanhos analisados, apresentam picos em *speedup* bastante característicos. Estes picos destacam os limites quando a quantidade de recursos disponíveis na máquina - como quantidade máxima de *threads* / núcleos de processamento, memória virtual e cache - para determinado tamanho de programa, alcança sua melhor distribuição dentro do *pipeline* característico desta arquitetura. Essa análise se torna mais evidente ao comparar os picos obtidos para os diferentes tamanhos de programas. É interessante observar que mesmo se tratando de uma arquitetura essencialmente sequencial, o tamanho do programa possui um grande impacto no *speedup*, bem como é verificado no modelo a fluxo de dados, que trata de arquiteturas essencialmente paralelas.

No gráfico da Figura 50a, é possível verificar que para quantidades de 2 e 3 *threads* o *speedup* encontra-se no intervalo entre 0 e 1, não sendo mostrado no gráfico logarítmico utilizado. A ocorrência destes pontos acentua a distância do *speedup*, quando comparado ao caso ideal. Deste ponto em diante, a curva apresenta uma pequena flutuação e segue crescente até o pico de desempenho possível de ser atingido, de forma bastante equivalente entre os diferentes tamanhos do programa. Para a profundidade do programa de integração igual a 9, que implica na ocorrência de 512 chamadas recursivas, o alcance do pico de desempenho é obtido especificamente para uma quantidade de 24 *threads*. De forma equivalente, para o programa com profundidade 10, que se expande em 1024 chamadas recursivas, o pico é obtido para a quantidade de 32 *threads*, desconsiderando o ponto isolado de 42 *threads*, onde o desempenho já se encontra em um regime decrescente. E para o programa com profundidade 11, expandido em 2048 chamadas recursivas, o pico em desempenho é obtido para 46 *threads*.

Os valores para os quais o melhor desempenho é obtido possuem uma relação com o tamanho do programa, um comportamento esperado a priori, uma vez que existe uma maior quantidade de paralelismo para cada tamanho. Mas além disso, estes valores se aproximam da distribuição de sua carga de trabalho. Considerando que a distribuição de carga seja realizada de forma perfeitamente equivalente entre os núcleos de processamento disponíveis para a execução de cada *thread*, a quantidade de funções que será tratada por cada unidade será de 21.333 para o programa de menor tamanho, que possui seu pico obtido para a quantidade de 24 *threads*. Já para o programa de tamanho intermediário teria uma distribuição de 32.000 funções recursivas para o tratamento de cada *thread*, que apresenta o melhor desempenho para o total de 32 *threads*. E finalmente, o maior programa teria a quantidade de 44.522 funções recursivas para cada *thread*, para o total de 46 *threads* em sua composição.

Essa distribuição é equivalente para ambos os programas das Figuras 50a e 50b, apesar de a segunda execução possuir um melhor desempenho geral, como esperado, uma vez que as funções recursivas possuem maior carga de processamento. Mas além disso, o

segundo programa possui melhores desempenhos em quantidades de recursos equivalentes. De forma que é possível concluir que este comportamento é encontrado para as melhores distribuições de carga dentro desta solução. Em seguida, destes pontos destacados em diante, ambos os programas apresentam uma decadência característica para ganho em desempenho em uma taxa bastante equivalente na execução de ambos os programas.

É importante destacar o comportamento fraco do *speedup* em *multicores* convencionais, em comparação com os resultados obtidos na execução destes mesmos programas, na π **FlowMR**. Este comportamento era previsível, uma vez que trata de um programa implementado para escalar melhor em granularidade mais fina, além do seu formato de implementação de dupla recursividade, o qual o modelo de *Von Neumann* não possui boa capacidade para a distribuição de carga. A literatura é vasta ao tratar das vantagens do modelo a fluxo de dados para a exploração deste tipo de programa em comparação com arquiteturas convencionais, principalmente ao considerar a exploração de programas que não se utilizam de estruturas de dados regulares. Esta comparação foi buscada a fim de destacar mais uma vez esta capacidade do modelo proposto e colocá-lo novamente como uma solução viável para a exploração de problemas desta natureza.

7 CONCLUSÃO

A proposta original deste trabalho consistia no projeto e implementação de uma arquitetura não convencional baseada no paradigma de fluxo de dados dinâmico, implementadas com **FPGAs** de baixo custo, que apresentasse boa escalabilidade. Para isso, foi tomada como fonte de inspiração principal a **Máquina a Fluxo de Dados de Manchester**, um poderoso processador que era capaz de apresentar desempenho compatível com as principais soluções computacionais de sua época - que consistiam principalmente de modelos convencionais e essencialmente sequenciais - aplicada sob o modelo de computação a fluxo de dados. Como descrito nos capítulos anteriores deste trabalho, os objetivos foram atingidos chegando-se a uma arquitetura original, a π **FlowMR**, que demonstrou grande potencial de *speedup*, apesar de aplicada para o uso de programas bem simples, que eram os únicos disponíveis já que inexistia, ainda, um sistema de compilação específico para esta arquitetura.

Inspirado também pelo crescente interesse em arquiteturas de computadores convencionais livres, promovido principalmente pelo advento do **RISC-V**, este projeto também busca incentivar a comunidade acadêmica sobre a realização de pesquisas de base em arquiteturas não convencionais. Movimento que era bastante comum nos anos 80, mas que acabou se tornando obsoleta nos últimos anos. A proposta inicial é de compartilhar esta solução em uma plataforma de código aberto, a fim de promover este interesse e permitir o avanço desta área do conhecimento que possui grande pertinência e potencial para a exploração de soluções originais e de alto impacto para dispositivos cotidianos.

7.1 Resumo do que foi atingido

Um dos problemas de modelos computacionais não convencionais é que, quando comparados com arquiteturas de *Von Neumann*, apresentam grande desvantagem em desempenho, já que as arquiteturas convencionais tem sido extremamente otimizadas durante as últimas décadas, ao contrário das máquinas não convencionais propostas na literatura acadêmica. Uma das otimizações apresentadas nesta tese que merece destaque são as soluções de **passagem direta**, como foram por nós nomeadas. Sua descrição detalhada foi vista no Capítulo 4 e os principais resultados obtidos, que destacam o sucesso dessa implementação foram apresentados nos gráficos das Figuras 44a, 44b, 46a, 46b, 48a e 48b. A ideia da **passagem direta** foi uma evolução natural do projeto, surgido de acordo com a necessidade de uma melhora no desempenho global da máquina. Após a revisão dos circuitos que apresentavam maiores caminhos críticos, de acordo com o relatório gerado no processo de compilação do **Quartus**, foi possível identificar um desempenho global que

não era ótimo com relação aos ciclos do relógio necessários para a execução dos programas avaliados, em comparação com os resultados obtidos antes desta revisão. Inspirado pela solução de passagem direta convencional da **MDFM**, que opera exclusivamente sob a Unidade de Pareamento e sempre foi tomado como um fator essencial para o ganho em desempenho daquela máquina, foram buscadas estratégias que fossem suficientes para otimizar o desempenho das outras unidades que compõem a solução de anel único, em uma estratégia semelhante.

Para isso, na Unidade de Instruções, foi proposta a implementação da “Pseudo Predição de Desvios”, que consiste em manter armazenado em um registrador de estados primitivo, a última instrução lida na memória de armazenamento desta unidade, em cada duto de largura de banda do anel. Esta implementação se mostrou surpreendente sob o ponto de vista da π **FlowBW**, pois ela é capaz de apresentar uma capacidade na taxa de acerto que pode ultrapassar facilmente 80% sob a quantidade total de fichas que passam por esta unidade, como foi apresentado nas Tabelas 2 e 6. Esta capacidade de exploração é suficiente para sanar qualquer deficiência no desempenho desta unidade, que sempre apresentou um baixo desempenho, devido a necessidade de leitura em uma memória **RAM**, a partir de todas as fichas que passavam por esta unidade, para resgatar uma das informações mais importantes da máquina, que é o código da instrução que será executada. Este resultado destaca que apesar do modelo a fluxo de dados já ter sido largamente explorado, ainda existem muitas possibilidades de implementações capazes de otimizar ainda mais sua capacidade para exploração de paralelismo, a partir de técnicas conhecidas, que já são utilizadas em arquiteturas convencionais, mas que sob o ponto de vista deste modelo de computação, também são capazes de proporcionar boas otimizações. A implementação desta estratégia, dentro da estrutura da π **FlowBW**, explorada no que diz respeito à largura de banda, pôde superar as capacidades da solução de passagem direta nativa da **MDFM**, tendo sido implementada de forma natural na estrutura da máquina, sendo capaz de remover qualquer gargalo desta unidade, otimizando o desempenho global de toda a solução.

Outra implementação de passagem direta que se mostrou importante para o desempenho da máquina, foi realizada na Unidade de Processamento. Esta solução foi inspirada por uma capacidade semelhante, presente na estrutura original da **MDFM**, que era conhecida como Unidade de Pré-Processamento. Em versões anteriores da π **Flow**, esta pseudo unidade tinha a finalidade de lidar apenas com a manipulação de Nomes de Ativação, mas aqui ela passou a realizar a execução efetiva de instruções mais simples, que pudessem ser executadas em apenas um ciclo do relógio. A princípio esta implementação se mostrou prejudicial, pois nela foram implementadas instruções muito significativas nos programas utilizados para validação, como pode ser visto na comparação entre os gráficos das Figuras 44c e 44d por exemplo. Por este motivo, foi possível concluir que esta unidade é suficiente para induzir um impacto negativo na execução de programas em uma estrutura baseada

no modelo a fluxo de dados, quanto à escalabilidade. O principal fator responsável pelo sucesso deste paradigma de computação paralela se dá na sua capacidade para distribuição de carga homogênea entre os diversos recursos de processamento disponíveis. A execução de muitas instruções dentro desta nova pseudo unidade acabou por explorar outro nível de paralelismo no **Multianel**, de forma que o desempenho global da máquina para execução de programas completos, reduziu consideravelmente. A execução dos programas passou a depender apenas da distribuição de instruções dentro dos dutos de largura de banda que formam esta unidade, de forma que independentemente da adição de novos recursos de processamento, as instruções tratadas por ela seriam distribuídas exatamente do mesmo modo.

Após a constatação destas limitações e realização de algumas avaliações experimentais, foi possível encontrar um limite ótimo para a utilização do pré-processador, sendo mantidas em sua estrutura apenas instruções de controle de fluxo. Estas instruções dependem de melhor desempenho por serem frequentes no início e final de programas, período que tipicamente não há paralelismo suficiente para exploração. Esta solução se mostrou bastante sensível entre sua capacidade de favorecer e prejudicar o desempenho da máquina, mas acabou se tornando uma solução importante para otimizar a capacidade para ganho em desempenho no **Multianel**, como pode ser visto na comparação entre os gráficos das Figuras 44e e 42f. Para a utilização de mais recursos de processamento, principalmente no **Multianel**, a quantidade de ciclos necessários para a execução de programas diminui bastante. Após a distribuição das principais instruções entre os anéis da máquina e o alcance ao seu comportamento predominantemente paralelo, a execução das instruções ocorrem de forma bastante imediata, garantindo essa larga capacidade para exploração de paralelismo. Dessa forma, o ganho em desempenho promovido pela execução imediata das instruções de controle, enquanto esta “expansão” ainda não foi finalizada, acaba por ser capaz de garantir um desempenho global melhor.

A contabilização de tempo utilizado para a execução dos programas tem início ainda durante o processo de negociação entre os anéis da máquina e finaliza diante da chamada à instrução de impressão do resultado. Neste processo, diversos ciclos do relógio são gastos antes mesmo de o programa ser inserido na máquina. Tecnicamente, seria mais preciso iniciar a contagem de ciclos apenas diante da ocorrência das fichas de início do programa. Entretanto, como esta estratégia foi utilizada desde as primeiras versões da implementação, para fins de comparação, esta análise se manteve no início da operação da máquina. Note ainda que este período gasto no processo de negociação é fixo de acordo com o programa em execução, podendo ser desconsiderado a posteriori. Porém este desconto não foi contabilizado, de forma que todos os resultados consideram este período de tempo que definitivamente não conta com qualquer capacidade de paralelismo. Além disso, é importante destacar que a métrica do *speedup* é capaz de desconsiderar este período fixo, gasto com o processo de negociação, de acordo com a construção de sua definição, Equação

6.1. Mas de qualquer forma, é importante destacar o sucesso na implementação do sistema de pré-processamento conforme definido em sua versão final e também toda análise que teve de ser realizada para garantir sua otimização.

Outra estrutura que surgiu naturalmente no processo de implementação e se mostrou um recurso excelente para exploração de eventuais novas capacidades da máquina no futuro é o sistema de Sobrecarga para fichas de comunicação, introduzida na π **FlowMR**. Nas versões anteriores do projeto foi lançada a hipótese de que a própria estrutura da máquina já seria suficiente para lidar com eventuais sobrecargas, principalmente no processo de armazenamento de fichas na Unidade de Pareamento, que possuía este recurso definido explicitamente na **MDFM**. Nesta máquina, a Unidade de Sobrecarga era definida dentro da Unidade de Pareamento e composta essencialmente por uma lista encadeada, que devia ser consultada na ocorrência de qualquer pacote pareado gerado por esta unidade.¹⁴ Note que trata-se de uma estrutura extremamente custosa, necessitando de hardware caro para sua implementação. Porém, de acordo com a hipótese levantada ainda durante a implementação da π **FlowBW**, a própria Fila de Fichas poderia ser considerada para o armazenamento dessas fichas órfãs - por tratar-se de uma grande capacidade de armazenamento nativa da máquina - possibilitando uma nova consulta na Unidade de Pareamento, apenas após sua passagem por esta fila completa.

Durante a implementação da π **FlowMR**, foi constatada a necessidade de uma grande fila de armazenamento, que fosse capaz de armazenar quantas fichas fossem necessárias para o processo de comunicação entre anéis. Esta definição levou à recordação desta hipótese antiga e que se mostrou extremamente efetiva, tornando a Fila de Fichas uma Unidade de Sobrecarga “flutuante”. A fila desta unidade se mantém bastante ocupada durante o processamento de programas, de forma que a introdução do armazenamento das fichas de comunicação que não puderam ser encaminhadas, devido a ocorrência de sobrecarga nos módulos responsáveis, teve um impacto desprezível diante de sua operação natural, provando esta solução como viável para este formato de operação. Em seguida, com as fichas marcadas para fins de comunicação, ela passa por toda a estrutura da máquina, até atingir os módulos de comunicação para nova tentativa de envio. Caso os módulos se encontrem cheios novamente, esta ficha retornará ao início do processo. Os resultados obtidos nessa verificação são suficientes para a extrapolação desta solução para a operação em diversos níveis de sobrecarga que podem ocorrer durante a execução de programas, nas diferentes unidades que compõem a estrutura de cada anel, abrindo diversas possibilidades de implementações para trabalhos futuros.

Durante a construção da π **FlowMR** foi identificado um comportamento que não se apresenta muito intuitivo, mas que foi lançado como um dos principais desafios a serem tratados por este projeto, que diz respeito à granularidade do conjunto de instruções da **MDFM**. É conhecido da literatura que uma das principais vantagens para o uso do modelo

a fluxo de dados, principalmente em sua versão dinâmica, é a fina granularidade que seu conjunto de instruções pode tratar. Apesar da **MDFM** tratar instruções de granularidade mais grossa, como o cálculo de funções trigonométricas por exemplo, implementadas na solução do presente projeto e destacadas como um dos principais recursos disponíveis capazes de promover um aumento considerável no desempenho global para a execução de programas, em nível de escalabilidade. Entretanto, de forma geral, a grande maioria das instruções implementadas na π **FlowMR** tratam de operações de controle de fluxo e aritméticas bastante simples. Mas a hipótese levantada durante a realização da revisão bibliográfica deste trabalho, é a de que o próprio **Multianel** é suficiente para promover um aumento na granularidade da máquina, de forma indireta.

Na implementação aqui proposta, as fichas são distribuídas entre os anéis da π **flowMR** a partir de seu Nome de Ativação, identificador presente em cada ficha, que está relacionada às chamadas de funções. Diante da ocorrência destas fichas, que podem ser criadas a partir de todos os anéis e destinadas a todos os outros, é verificada a ocorrência deste aumento indireto na granularidade do conjunto de instruções, através de um fenômeno que foi chamado de aglutinação de instruções. Este comportamento ocorre devido a esta estratégia para distribuição de fichas entre os anéis. Diante da geração de novos nomes de ativação, uma nova ficha, já destinada para este identificador específico será gerada e portanto, direcionada para o anel que será responsável por sua execução. Neste momento, todas as outras instruções desta função serão geradas e permanecerão no mesmo anel. Com isso, sob o ponto de vista do **Multianel**, uma função completa é alocada a uma unidade de processamento (anel), enquanto que dentro de cada anel a mesma granularidade da **MDFM** é identificada, carregando consigo todos os benefícios para exploração de paralelismo dentro deste paradigma de computação paralela. Dessa forma, bem como pôde ser verificado na execução de programas que possuem instruções de granularidade mais grossa, como discutido nos gráficos das Figuras 48 e 49, o **Multianel** da π **FlowMR** possui ainda este novo nível de abstração capaz de explorar um paralelismo ainda melhor distribuído.

É conhecido na literatura, que na estrutura da **MDFM**, o controle do paralelismo é capaz de promover um certo ganho em desempenho computacional.⁴³ Apesar deste comportamento parecer contra intuitivo, foi possível revisar a partir do presente projeto, esta estratégia que ficou conhecida como *Throttle* na **MDFM**. De acordo com sua definição, a Unidade de Pareamento acaba se comportando como um grande gargalo para a estrutura dos anéis, uma vez que muitas operações devem ser realizadas por esta unidade, até que seus pacotes de saída sejam gerados e fornecidos para a Unidade de Instruções. De fato, na ocorrência de fichas destinadas para pareamento, uma memória RAM deve ser lida, o pacote de saída deve ser gerado e encaminhado para os *buffers* de saída desta unidade, enquanto que a lista da Unidade de Sobrecarga é consultada para a análise de liberação do endereço que acabou de se tornar vago, e caso seja encontrada uma ficha que pode

ocupar este novo endereço, ela deve ser armazenada nesta memória RAM na sequência. Note que até que todo este processo seja finalizado, não é possível aceitar novas fichas nesta unidade, para pareamento e todo o processamento acaba sendo afetado.

Por este motivo, na **MDFM**, foi lançada a hipótese sobre este controle para o fornecimento de novas fichas para esta unidade, que acabou apresentando bons resultados neste protótipo. No presente projeto, a ocorrência de fichas destinadas para outro anel da estrutura do **Multianel**, ainda considerando eventuais conflitos para a utilização dos recursos de comunicação entre módulos mestres e escravos, que acabou promovendo a necessidade de uma operação de sobrecarga para o consumo dessas fichas, acaba promovendo o mesmo efeito de *throttle*, como definido. A comparação entre os gráficos das Figuras 45 e 47 destacam a ocorrência deste efeito, uma vez que a implementação dos módulos de comunicação verificada no estado de validação da máquina trata do limite máximo em desempenho possível de ser obtido pela topologia de comunicação proposta e os resultados são totalmente compatíveis, mesmo com a introdução destes gargalos provenientes da implementação dos módulos de comunicação. Dessa forma, é possível atribuir estes resultados ao efeito destacado, re-validando esta estratégia já explorada pela máquina de referência, e verificada no presente projeto de forma indireta. Esta constatação fornece ainda mais subsídios para a validação da estratégia proposta por este trabalho, por tratar de um comportamento que acabou se tornando natural na implementação da máquina de referência.

Este projeto ainda buscou validar uma métrica exclusiva para análise desta estratégia de computação, a partir de métodos experimentais. Trata-se do *speedup* efetivo, que foi utilizado para mostrar o ganho em desempenho da máquina, de acordo com a variação na quantidade de anéis na sua estrutura. Verificação esta, que se mostra muito mais eficiente sob o ponto de vista do **Multianel**, uma vez que este é o formato mais prático de análise desta implementação, verificado de acordo com a adição de novos anéis à solução. A análise gráfica desta capacidade acaba sendo afetada negativamente, porém, de acordo com esta métrica, é possível analisar esta estratégia de forma mais assertiva, destacando o *speedup* real que pode ser obtido de acordo com a adição de recursos à máquina. Esta métrica foi utilizada como uma ferramenta de análise durante a discussão dos resultados, sendo validada como uma aproximação viável com relação ao limite inferior do *speedup* possível de ser alcançado. Ferramenta esta que pode ser explorada por eventuais análises futuras, que sejam baseadas nas estratégias verificadas neste projeto.

Diante das dificuldades encontradas para a implementação do sistema de comunicação proposto, certamente os *kits* de desenvolvimento **Terasic DE10-Nano** utilizados, não podem ser considerados como a melhor escolha possível. Os módulos de comunicações disponíveis neste *kit* de desenvolvimento não possuem ao menos as condições mínimas aceitáveis para a implementação deste sistema, o que inviabilizou o processo de desenvol-

vimento pela necessidade da construção de uma placa de circuito impresso exclusiva, a fim de permitir sua realização, como descrito no Apêndice B. Por um custo um pouco mais elevado, existem soluções no mercado voltadas para sistemas de comunicações mais robustos. Porém, como um dos principais focos do projeto era a utilização de um sistema de baixo custo, foi realizada esta escolha. O gráfico da Figura 40 destaca algumas outras possibilidades que poderiam ter sido empregadas, como a própria **DE1-SoC** da mesma fabricante, a qual implementa o circuito de proteção, que inclusive serviu de inspiração para a construção do circuito construído para este projeto. Porém, no instante da aquisição dos *kits* de desenvolvimento, este outro modelo possuía um custo mais elevado e foi descartado, além dele contar com menos elementos lógicos. A estrutura construída se mostrou bastante eficiente para a exploração deste projeto, porém, o tempo de desenvolvimento se tornou demasiado longo, de forma que é possível concluir que a escolha de outros modelos poderiam ter um impacto positivo no processo de desenvolvimento.

Uma importante análise que não pôde ser tratada neste projeto em tempo hábil, devido às dificuldades para a plena implementação do sistema, trata sobre a topologia de comunicação proposta. É possível identificar as limitações físicas verificadas na estrutura em anel construída, uma vez que de acordo com a implementação, cada unidade do **Multianel** é capaz de gerar fichas de instruções para todas as outras unidades da máquina. Uma possível solução imediata para este problema poderia ser a limitação na geração de fichas destinadas para outras unidades, apenas à vizinhança mais próxima do anel corrente. A introdução deste controle já poderia ser suficiente para evitar eventuais gargalos provenientes do percurso que uma ficha deve seguir para sua execução efetiva. Existem benefícios com a introdução deste atraso proveniente da necessidade de comunicação, porém, a linha que separa estes benefícios e o impedimento da execução de instruções importantes para a execução de programas é tênue.

Outra solução que poderia ser empregada a fim de permitir a exploração de melhores distribuições de fichas entre anéis, poderia ser o uso de outra topologia de comunicação, solução impraticável nos recursos disponíveis para a implementação. Uma hipótese surgida durante o desenvolvimento deste projeto, é de que uma topologia em malha, já poderia ser suficiente para a exploração de até milhares de unidades de processamento. Apesar que para a exploração desta quantidade de recursos, outras modificações deveriam ser consideradas, bem como as estratégias de validações utilizadas. Certamente, a exploração de múltiplos programas paralelos seria uma condição inevitável para a exploração deste nível de processamento, de forma que outras estratégias em nível de arquitetura de computadores deveriam ser empregadas nesta solução. Em todo caso, a topologia de comunicação acabou se tornando uma das principais limitações no processo de desenvolvimento, mas que deixa em aberto diversas possibilidades para exploração futura.

Com base nas constatações destacadas, e principalmente diante da comparação

entre as análises gráficas discutidas sobre as Figuras 39 e 49, onde é possível verificar a grande evolução de desempenho e capacidade para escalabilidade obtida durante o desenvolvimento deste projeto, com um *speedup* máximo que atingiu valores próximos a 60, é possível tomar esta implementação como uma solução perfeitamente plausível e viável, avaliado como uma verdadeira evolução na estratégia do fluxo de dados, para o tratamento de até centenas de recursos de processamento.

7.2 Trabalhos Futuros

Diversas estratégias discutidas ao longo do texto não se fizeram necessárias no decorrer do desenvolvimento deste projeto, principalmente devido a ausência de programas mais complexos, compilados para sua exploração, de forma que um dos principais recursos para melhor avaliar o **Multianel** seria o desenvolvimento de um sistema de compilação para o conjunto de instruções da π **FlowMR**. Foram feitas diversas tentativas de se obter o compilador original da **MDFM**, mas todas sem sucesso, sendo possível que esta solução tenha se perdido ao longo do tempo e realmente não esteja mais disponível para quaisquer finalidades. A linguagem de programação **Sisal** encontra-se descontinuada atualmente, mas segue distribuída por alguns entusiastas, principalmente com relação ao modelo a fluxo de dados e certamente esta linguagem é capaz de oferecer um ponto de partida muito bem estabelecido, a partir da sua geração de código intermediário, a **IF1**. A partir de uma solução desta natureza, com o controle que o **BSV** é capaz de oferecer sobre o conjunto de instruções da máquina, abrem-se muitas possibilidades que podem ser introduzidas na π **FlowMR**. Outra estratégia que pode ser destacada, seria a criação de uma solução completa em *front-end*, linguagem intermediária e *back-end* para esta arquitetura, a partir do **LLVM**, um *framework* destinado para o desenvolvimento de compiladores, já bastante consolidado, principalmente para fins acadêmicos.

Uma característica que se destacou bastante durante a construção deste trabalho foi o sistema de predição de desvios da Unidade de Instruções. Alguns testes foram realizados a fim de manter neste sistema o armazenamento da vizinhança mais próxima das últimas instruções lidas em sua memória, em vetores de registradores primitivos. Devido ao caráter dos programas explorados para validação da máquina, o impacto desta implementação foi muito pequeno, em comparação com o armazenamento apenas da última instrução lida em cada duto de largura de banda. Dessa forma, a fim de manter o uso de recursos lógicos das **FPGAs** mais controlado, foi mantida a solução destacada nos capítulos anteriores. Entretanto, para a aplicação em programas mais genéricos, que podem não apresentar as características recursivas utilizadas nos programas de validação, outras estratégias podem ser muito úteis. É possível que novas estratégias, ou mesmo apenas o armazenamento da vizinhança mais próxima das últimas instruções lidas na memória RAM desta unidade, sejam soluções capazes de atingir níveis de desempenho ainda melhores para aplicações

diversas.

A introdução do sistema de Sobrecarga flutuante pode trazer novos limites para a exploração da Unidade de Pareamento principalmente. Na **MDFM**, a operação de Sobrecarga convencional era extremamente custosa, uma vez que tratava sobre operações em uma lista encadeada. Como demonstrado neste projeto, sob o ponto de vista da sobrecarga proveniente dos módulos de comunicação, o impacto para o armazenamento de fichas de transição na Fila de Fichas da máquina é desprezível em comparação com a taxa de armazenamento convencional desta unidade. Dessa forma, dependendo da função *hash* aplicada para a realização do endereçamento na memória da Unidade de Pareamento, a extrapolação da solução de Sobrecarga flutuante pode trazer grandes benefícios para a implementação. Atualmente, a Unidade de Pareamento conta com um sistema de endereçamento completo, para o armazenamento de quantas fichas forem necessárias dentro de sua memória, de acordo com o programa em execução, e essa característica acaba promovendo a utilização de muitos recursos lógicos para síntese em **FPGA**. O uso desta solução de sobrecarga para o armazenamento de fichas órfãs pode promover uma diminuição considerável para essa memória de armazenamento, restando espaço mesmo para a implementação do sistema de armazenamento da Fila de Fichas. O impacto desta solução pode ser considerado como nulo, portanto, e os benefícios seriam muitos, como a capacidade para execução de programas de tamanhos muito maiores, por exemplo.

Considerando a implementação desta solução dentro de um circuito integrado, em um processador de propósito geral, as memórias RAM utilizadas atualmente na Unidade de Pareamento e de Instruções, poderiam passar a operar como o equivalente a uma memória *cache* para a máquina, que deveria consultar a disponibilidade de recursos em sistemas de memórias compartilhadas, por exemplo. Esta que é uma outra proposta para a realização de trabalhos futuros, a expansão ilimitada, a priori, na capacidade de armazenamento destas memórias tão importantes para a operação da máquina. Mesmo a Fila de Fichas poderia se aproveitar deste recurso caso necessário, permitindo sua exploração para qualquer operação de sobrecarga necessária.

Uma importante unidade que foi omitida durante o desenvolvimento deste projeto é a Unidade de Estruturas, que pode ser implementada em versões futuras. Esta unidade tem a finalidade de oferecer à máquina uma capacidade para lidar com dados estruturados, como matrizes e vetores. Nesta solução teria-se uma implementação relativamente simples, com a finalidade apenas de armazenar os dados estruturados em uma grande memória de armazenamento, que ficaria alocada dentro da estrutura convencional de cada anel da solução. Note portanto, tratar-se de outra estrutura que depende de uma capacidade de armazenamento massiva, e portanto deve contar com as capacidades de exploração de memórias externas à solução. Mas certamente, para o uso de programas com estruturas de dados complexa, esta unidade será fundamental.

A estrutura do **Multianel** favorece a exploração de conceitos de computação verde. Neste projeto foram buscadas estratégias que permitiam a exploração máxima dos recursos a disposição, mas foi possível notar que mesmo dentro de um sistema controlado, para a execução de programas com capacidades de paralelismo bem definidas, muitos recursos acabam subutilizados. Por este motivo, uma outra estratégia que pode ser explorada por esta solução, seria a ativação de anéis da máquina sob demanda apenas, de forma que diante da necessidade de mais poder computacional, novos anéis poderiam ser adicionados à máquina dinamicamente. Esta solução que possui grande apelo nos dias atuais, poderia trazer novas capacidades para o modelo a fluxo de dados que ainda não puderam ser exploradas, mesmo por tratar-se de conceitos levantados apenas mais recentemente. Com a introdução de novas topologias de comunicação entre os anéis, este conceito poderia inclusive ser extrapolado para ramos, ou grupos de anéis que seriam ativados para exploração sob demanda.

REFERÊNCIAS

- 1 Terasic, Inc. **DE1-SoC Board**: user manual. Taiwan, 2014.
- 2 MILLER, F. P.; VANDOME, A. F.; MCBREWSTER, J. **Parallel ATA**. [S.l.]: AlphaScript Publishing, 2009. 108 p. ISBN 6130070896.
- 3 KISH, L. B. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. **Physics Letters A**, v. 305, n. 3-4, p. 144-149, 2002.
- 4 DENNIS, J. B. Data flow supercomputers. **IEEE Computer**, v. 13, n. 11, p. 48-56, 1980.
- 5 WATSON, I.; GURD, J. R. A practical dataflow computer. **IEEE Computer**, v. 15, n. 2, p. 51-58, 1982.
- 6 YUBA, T. *et al.* SIGMA-1: A dataflow computer for scientific computations. **Computer Physics Communications**, v. 37, n. 1, p. 141 - 148, 1985. ISSN 0010-4655.
- 7 ARVIND; CULLER, D. E. Dataflow architectures. **Annual Review of Computer Science**, v. 1, p. 225-253, 1986. Disponível em: <https://doi.org/10.1146/annurev.cs.01.060186.001301>. Acesso em: 12 out.21.
- 8 MARKOV, I. L. Limits on fundamental limits to computation. **Nature**, v. 512, n. 7513, p. 147-157, 2014.
- 9 WALDROP, M. The chips are down for Moore's law. **Nature News**, v. 530, n. 7589, p. 144, 2016.
- 10 FAXÉN, K.-F. *et al.* **Multicore computing—the state of the art**. 2008. Disponível em: <https://www.diva-portal.org/smash/get/diva2:503169/FULLTEXT01.pdf>. Acesso em: 28 set.21.
- 11 SANCHEZ, D.; KOZYRAKIS, C. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. **ACM SIGARCH Computer Architecture News**, Association for Computing Machinery, v. 41, n. 3, p. 475-486, June 2013. ISSN 0163-5964.
- 12 DONGARRA, J.; LUSZCZEK, P. **TOP500**. Boston, MA: Springer, 2011. Disponível em: https://doi.org/10.1007/978-0-387-09766-4_157. Acesso em: 25 set.21.
- 13 VEEN, A. H. Dataflow machine architecture. **ACM Computing Surveys**, Association for Computing Machinery, v. 18, n. 4, p. 365 - 396, Dec 1986. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/27633.28055>. Acesso em: 25 set.21.
- 14 GURD, J. R.; KIRKHAM, C. C.; WATSON, I. The manchester prototype dataflow computer. **Communications of the ACM**, v. 28, n. 1, p. 34-52, 1985.
- 15 MAGNA, P. **Proposta e simulação de uma arquitetura a fluxo de dados de segunda geração**. 1997. Tese (Doutorado em Ciências) — Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 1997.

- 16 KIRKHAM, C. C. **The Manchester prototype dataflow system: basic programming**. 6th. ed. Manchester: UMC-DF-BPM, 1987.
- 17 KUON, I.; TESSIER, R.; ROSE, J. **FPGA Architecture: survey and challenges**. Hanover, MA: Now Publishers Inc, 2008. ISBN 9781601981264. Disponível em: <https://books.google.com.br/books?id=AdK2OWDP7L0C>. Acesso em: 25 set.21.
- 18 SILVA JUNIOR, J. T. **PIFLOW - projeto, simulação e implementação de um protótipo dataflow em FPGA**. 2016. Dissertação (Mestrado) — Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2016.
- 19 BOUTROS, A.; BETZ, V. FPGA architecture: principles and progression. **IEEE Circuits and Systems Magazine**, v. 21, n. 2, p. 4–29, 2021. Disponível em: <https://doi.org/10.1109/MCAS.2021.3071607>. Acesso em: 29 jan.22.
- 20 ALTERA INC. **Altera: FPGA Architecture White paper**. San Jose, CA, 2006. 14 p. Disponível em: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf>. Acesso em: 25 set.21.
- 21 MASSOUMI, M. M. **Encyclopedia of life support systems (EOLSS): Computer science and engineering**. EOLSS Publishers, 2009. v. 1. ISBN 978-1-84826-227-0. Disponível em: <https://www.eolss.net/Sample-Chapters/C15/E6-45-02-11.pdf>. Acesso em: 14 out.21.
- 22 NAGEL, L. W.; MCANDREW, C. C. Is SPICE good enough for tomorrow's analog? *In: IEEE Bipolar/BiCMOS circuits and technology meeting (BCTM)*. Austin, TX: IEEE, 2010. p. 106–112. Disponível em: <https://doi.org/10.1109/BIPOL.2010.5668096>. Acesso em: 13 out.21.
- 23 IEEE COMPUTER SOCIETY. **IEEE Standards 1076-1987 VHDL: language reference manual interpretations**. New York: IEEE, 1992. ISBN 1559371811.
- 24 IEEE COMPUTER SOCIETY. **IEEE Standards 1364-1995 standard hardware description language based on the Verilog(R) hardware description language**. New York: IEEE, 1996. 688 p.
- 25 IEEE COMPUTER SOCIETY. **IEEE standard for SystemVerilog: unified hardware design, specification and verification language**. New York: IEEE, 2005. 648 p.
- 26 BLACK, D. C. *et al.* **SystemC: from the ground up**. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 0387699570.
- 27 NIKHIL, R. S.; ARVIND. What is Bluespec? **SIGDA Newsletter**, v. 39, n. 1, p. 1–1, 2009.
- 28 NIKHIL, R. Bluespec SystemVerilog: efficient, correct RTL from high level specifications. *In: INTERNATIONAL CONFERENCE ON FORMAL METHODS AND MODELS FOR CO-DESIGN. MEMOCODE '04*, 2004, Waltham. **Proceedings[...]**. Waltham: IEEE Computer Society, 2004.
- 29 TERESE. **Term rewriting systems**. Cambridge: Cambridge University Press, 2003. v. 55. (Cambridge Tracts in Theoretical Computer Science, v. 55). Disponível em: <https://www.bibsonomy.org/bibtex/28f7a18abdbb5478d1277c0164b2fa115/emanuel>. Acesso em: 25 set.21.

- 30 SINGH, D.; SHUAIBU, A. M.; IBRAHIM, A. M. An overview of term rewriting systems. **African Journal of Mathematics and Computer Science Research**, Academic Journals, v. 5, n. 9, p. 153–157, 2012.
- 31 DAVE, N. H. **Designing a processor in Bluespec**. 2005. Dissertação (Mestrado) — Massachusetts Institute of Technology, Massachusetts, 2005.
- 32 NIKHIL, R. **Bluespec SystemVerilog training**. 2012. Disponível em: http://www.demosondemand.com/dod/proddemos/vendors/pd_bluespec.aspx. Acesso em: 10 jul.21.
- 33 TRELEAVEN, P. C.; BROWNBRIDGE, D. R.; HOPKINS, R. P. Data-driven and demand-driven computer architecture. **ACM Computing Surveys**, Association for Computing Machinery, v. 14, n. 1, p. 93–143, Mar 1982. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/356869.356873>. Acesso em: 25 set.21.
- 34 DENNIS, J. B. First version of a data flow procedure language. **Programming Symposium**, Springer-Verlag, v. 19, p. 362–376, 1974.
- 35 ACKERMAN, W.; DENNIS, J. **VAL – a value-oriented algorithmic language**: preliminary reference manual. USA, 1979. Disponível em: <https://doi.org/10.1145/357153.357157>. Acesso em: 14 out.21.
- 36 BURGER, D. *et al.* Scaling to the end of silicon with edge architectures. **Computer**, v. 37, n. 7, p. 44–55, 2004.
- 37 GEBHART, M. *et al.* An evaluation of the trips computer system. **SIGARCH Computer Architecture News**, Association for Computing Machinery, New York, NY, USA, v. 37, n. 1, p. 1–12, Mar 2009. ISSN 0163-5964. Disponível em: <https://doi.org/10.1145/2528521.1508246>. Acesso em: 28 jan.22.
- 38 FLYNN, M. Dataflow supercomputing. *In: International conference on field programmable logic and applications*, **22**. Oslo: IEEE, 2012. p. 1–5. ISBN 978-1-4673-2257-7. Disponível em: <https://doi.org/10.1109/FPL.2012.6339172>. Acesso em: 28 jan.22.
- 39 PELL, O.; AVERBUKH, V. Maximum performance computing with dataflow engines. **Computing in Science & Engineering**, IEEE Computer Society, Los Alamitos, CA, USA, v. 14, n. 04, p. 98–103, Jul 2012. ISSN 1558-366X.
- 40 GIORGI, R. *et al.* TERAFLUX: Harnessing dataflow in next generation teradevices. **Journal of Microprocessors and Microsystems**, v. 38, n. 8, p. 976–990, 2014.
- 41 KHAN, B. *et al.* Architectural support for task scheduling: hardware scheduling for dataflow on NUMA systems. **The Journal of Supercomputing**, v. 71, n. 6, p. 2309–2338, 2015.
- 42 SWANSON, S. *et al.* The wavescalar architecture. **ACM Transactions on Computer Systems**, New York, NY, USA, v. 25, n. 2, May 2007. ISSN 0734-2071. Disponível em: <https://doi.org/10.1145/1233307.1233308>. Acesso em: 28 jan.22.
- 43 SARGEANT, J.; RUGGIERO, C. A. Control of parallelism in the manchester dataflow machine. **Lecture Notes in Computer Science**, v. 274, p. 1–15, 1987.

- 44 GURD, J. R. The manchester dataflow machine. **Computer Physics Communications**, v. 37, n. 1, p. 49–62, 1985.
- 45 GURD, J.; KIRKHAM, C. C.; BÖHM, W. The manchester dataflow computing system. *In*: DONGARRA, J. J. (ed.). **Experimental parallel computing architectures**. Amsterdam: North Holland, 1987.
- 46 BLUESPEC, INC. **Bluespec SystemVerilog**: reference guide. Massachusetts, 2012. Revision: 30 July 2014. Disponível em: http://csg.csail.mit.edu/6.S078/6_S078_2012/www/resources/reference-guide.pdf. Acesso em: 28 set.21.
- 47 BARAHONA, P. M. C. C.; GURD, J. R. Processor allocation in a multi-ring dataflow machine. **Journal of Parallel and Distributed Computing**, v. 3, p. 305–327, 1986.
- 48 BARAHONA, P. M. C. C. **Performance evaluation of a multi-ring dataflow machine**. 1984. Dissertação (Mestrado) — University of Manchester, Manchester, 1984.
- 49 BÖHM, A. P. W.; TEO, Y. M. Resource management in a multi-ring dataflow machine. *In*: INTERNATIONAL CONFERENCE ON PARALLEL ARCHITECTURES, 1988, UMIST, Manchester, United Kingdom. **Proceedings[...]**. New York: Cambridge University Press, 1988. p. 566–577. ISBN 0521371775.
- 50 ROSENBAND, D. L. The ephemeral history register: flexible scheduling for rule-base designs. *In*: INTERNATIONAL CONFERENCE OF FORMAL METHODS AND MODELS FOR CODESIGN, 4., 2004. **Proceedings[...]**. Massachusetts: Massachusetts Institute Technology, 2004. p. 189–198.
- 51 BOHM, A. P. W.; GURD, J. R. Iterative instructions in the manchester dataflow computer. **IEEE Transactions on Parallel and Distributed Systems**, v. 1, n. 2, p. 129–139, Apr. 1990. ISSN 1045-9219.
- 52 SILVA JUNIOR, J. T.; MATIAS, P.; RUGGIERO, C. A. π FlowMR: um protótipo dataflow escalável, implementado em um cluster de FPGAs de baixo custo. *In*: SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 2018. **Anais[...]**. São Paulo: Sociedade Brasileira de Computação, 2018.
- 53 BÖHM, A. P. W.; SARGEANT, J. **Efficient code generation for SISAL**. Manchester: University of Manchester, 1985. Technical Report UMCS-85-10-2.
- 54 IEEE STD 802.3-2002. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. **IEEE Std 802.3-2002**, New York: IEEE, Dec 2005. Disponível em: <https://doi.org/10.1109/IEEESTD.2002.94131>. Acesso em: 18 out.21.
- 55 MCLEAN, P. T. **Information technology - AT attachment with packet interface - 6 (ATA/ATAPI-6)**. Longmont, Colorado, 2002. 482 p. Disponível em: <http://www.t13.org/Documents/UploadedDocuments/project/d1410r3b-ATA-ATAPI-6.pdf>. Acesso em: 06 ago.20.

- 56 MCLEAN, P. T. **Information technology - AT attachment with packet interface - 5 (ATA/ATAPI-5)**. Longmont, Colorado, 2000. 383 p. Disponível em: <http://www.t13.org/documents/UploadedDocuments/project/d1321r3-ATA-ATAPI-5.pdf>. Acesso em: 06 ago.20.
- 57 WILLIAMS, S.; BAXTER, M. Icarus Verilog: Open-source verilog more than a year later. **Linux Journal**, Belltown Media, Houston, TX, v. 2002, n. 99, p. 3, July 2002. ISSN 1075-3583. Disponível em: <http://dl.acm.org/citation.cfm?id=513581.513584>. Acesso em: 25 set.21.
- 58 BLUESPEC, INC. **PAClib**: reference guide. Framingham, Massachusetts, 2014. Version 1.4 (May 15, 2014).
- 59 Terasic, INC. **DE1-SoC**: user manual. East Dist: Hsinchu City, 2016. Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=836&PartNo=4>. Acesso em: 18 out.21.
- 60 Terasic, INC. **DE10-Nano**: user manual. East Dist: Hsinchu City, 2016. Disponível em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=1046&PartNo=4>. Acesso em: 18 out.21.
- 61 SHARP, J. A. **Dataflow computing**: theory and practice. New York: Ablex Publishing Company, 1992.
- 62 JINKS, P. J. **A dataflow system simulator**. 1977. Dissertação (Mestrado) — Manchester University, Manchester, 1977.
- 63 GURD, J. R.; KIRKHAM, C. C. Dataflow: achievements and prospects. **Information Processing**, v. 86, p. 61–68, 1986.
- 64 NUSSBAUM, D.; AGARWAL, A. Scalability of parallel machines. **Communication of the ACM**, New York, NY, USA, v. 34, n. 3, p. 57–61, Mar 1991. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/102868.102871>. Acesso em: 25 set.21.
- 65 KUMAR, V. P.; GUPTA, A. Analyzing scalability of parallel algorithms and architectures. **Journal of Parallel and Distributed Computing**, v. 22, n. 3, p. 379–391, 1994.
- 66 DAVID, M. N.; FRANK, H. W. Problem size, parallel architecture, and optimal speedup. **Journal of Parallel and Distributed Computing**, v. 5, n. 4, p. 404 – 420, 1988. ISSN 0743-7315. Disponível em: <http://www.sciencedirect.com/science/article/pii/0743731588900056>. Acesso em: 25 set.21.
- 67 KUMAR, V.; RAO, V. Parallel depth first search. part ii. analysis. **International Journal of Parallel Programming**, v. 16, p. 501–519, 01 1987.
- 68 GUSTAFSON, J. L. Reevaluating Amdahl's law. **Communications of the ACM**, Association for Computing Machinery, v. 31, n. 5, p. 532–533, May 1988. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/42411.42415>. Acesso em: 25 set.21.
- 69 SUN, X. H.; NI, L. M. Another view on parallel speedup. *In*: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 1990, New York, USA. **Proceedings[...]**. New York: IEEE Computer Society Press, 1990. p. 324–333. Disponível em: <https://doi.org/10.1109/SUPERC.1990.130037>. Acesso em: 14 out.21.

- 70 WORLEY, P. H. The effect of time constraints on scaled speedup. **SIAM Journal on Scientific and Statistical Computing**, v. 11, n. 5, p. 838–858, 1990. Disponível em: <https://doi.org/10.1137/0911050>. Acesso em: 25 set.21.
- 71 PALACHARLA, S.; JOUPPI, N. P.; SMITH, J. E. Complexity-effective superscalar processors. **SIGARCH Computer Architecture News**, Association for Computing Machinery, v. 25, n. 2, p. 206–218, May 1997. ISSN 0163-5964. Disponível em: <https://doi.org/10.1145/384286.264201>. Acesso em: 25 set.21.
- 72 BROWNE, L. M. H.; SARGEANT, J. **A dataflow simulator**. Manchester: University of Manchester, 2013. 49 p. Completion of Course Work (Computational Science).
- 73 ALTERA CORPORATION. **Cyclone V**: device overview. San Jose, CA, 2018. Disponível em: https://www.altera.com/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf. Acesso em: 25 set.21.
- 74 FERREIRA, F. **Um modelo de execução dirigido pelos dados em processadores multi-core**. 2020. Tese (Doutorado em Ciências) — Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2020.
- 75 VOLDER, J. E. The CORDIC trigonometric computing technique. **IRE Transactions on Electronic Computers**, EC-8, n. 3, p. 330–334, 1959.
- 76 WIKIPEDIA. **CORDIC**. 2021. Disponível em: <https://en.wikipedia.org/wiki/CORDIC>. Acesso em: 13 set.21.
- 77 LAKSHMI, B.; DHAR, A. S. CORDIC architectures: a survey. **VLSI Design**, Hindawi Limited, v. 2010, 2010. ISSN 1065-514X. Disponível em: <https://doi.org/10.1155/2010/794891>. Acesso em: 25 set.21.
- 78 DAVIS, K. A. **Computing sin & cos in hardware with synthesisable Verilog**. 2011. Disponível em: <https://kierdavis.com/cordic.html>. Acesso em: 19 set.21.
- 79 WALTERS, K.; WERNER, B. Introduction to schottky rectifiers. **MicroNote Series**, v. 401, 2016. Disponível em: <https://www.microsemi.com/sites/default/files/micnotes/401.pdf>. Acesso em: 25 set.21.
- 80 ORHANOVIC, N.; RAGHURAM, R.; MATSUI, N. Characterization of microstrip meanders in PCB interconnects. *In*: ELECTRONIC COMPONENTS AND TECHNOLOGY CONFERENCE, 50TH, 2000, Las Vegas. **Proceedings[...]**. Las Vegas, NV: IEEE, 2000. p. 508–512. Disponível em: <https://doi.org/10.1109/ECTC.2000.853205>. Acesso em: 14 out.21.

Apêndices

APÊNDICE A – CORDIC

A.1 Definições

O **CORDIC**, derivado da expressão em inglês *COordinate Rotation DIgital Computer*, que em tradução literal pode ser interpretado como um “Computador de Coordenadas de Rotação Digital”, trata de um algoritmo simples e eficiente que pode ser aplicado em sistemas computacionais para a realização do cálculo de diversas áreas da matemática, desde o cálculo de operações de divisão e multiplicação, até funções trigonométricas, tipicamente a partir da conversão de cada dígito de precisão da medida, a cada iteração de seu modelo de operação. Por este motivo, normalmente o **CORDIC** é tomado como um exemplo de algoritmo de dígito a dígito. Este algoritmo é utilizado para a realização destas operações matemáticas mais elaboradas, principalmente para a implementação em sistemas que não possuem multiplicadores, pelo fato que ele utiliza apenas de funções triviais para a realização dos cálculos, como somas, subtrações, deslocamento de bit e tabelas de conferência (LUT).^{75,76}

Métodos matemáticos semelhantes ao **CORDIC** remontam ao século 17, entretanto, devido ao momento histórico realmente, o **CORDIC**, proposto por **Volder** no ano de 1956, é mais otimizado para o uso em computadores formados por máquinas de estados finitos de baixa complexidade. Seu princípio de funcionamento é bastante simples, no presente projeto ele é utilizado no modo de rotação, como definido. O algoritmo opera sobre um vetor unitário, que é rotacionado quantas vezes forem necessárias (dependente exclusivamente sob a precisão desejada para o cálculo) a fim de se aproximar o máximo possível do ângulo que se espera encontrar, obtendo com isso as coordenadas espaciais deste novo vetor rotacionado, a partir de suas identidades trigonométricas.

A Figura 51 apresenta o princípio de funcionamento deste algoritmo, evoluindo sobre o total de quatro iterações, explorado apenas sobre o primeiro quadrante do eixo de coordenadas. Perceba que o vetor v_0 é o ponto de partida do algoritmo, encontrado sobre as coordenadas indicadas na expressão da Equação A.1, com ângulo de inclinação igual a zero. Na sequência este vetor é rotacionado por um ângulo definido previamente pelo algoritmo, dando origem ao vetor v_1 . A rotação promovida na primeira iteração observada ainda é menor do que o ângulo β que se deseja encontrar, dessa forma, é realizada uma nova rotação no mesmo sentido anterior, desta vez aplicado sobre o vetor v_1 , a partir de um deslocamento mais “preciso”, cuja magnitude também deve ser definida estaticamente de acordo com este algoritmo, com isso, dando origem ao vetor v_2 . Na sequência, uma nova rotação será realizada, perceba que ela é ainda mais “precisa” e é realizada no sentido contrário da rotação anterior, pelo fato que o ângulo β é menor do que o ângulo do vetor

v_2 . Neste momento, é encontrado o ângulo β que era procurado inicialmente. Perceba que a manutenção do registro do sistema de coordenadas bidimensional, em cada passo de iteratividade do algoritmo é suficiente para encontrar os valores do *seno* e do *coseno* referentes ao ângulo β , que tratam de uma grandeza trigonométrica que o algoritmo é capaz de fornecer e que foi objeto de sua exploração no presente projeto.

Figura 51 – Ilustração do processo de evolução do algoritmo **CORDIC** após a realização de quatro iterações.

Fonte: WIKIPEDIA⁷⁶

O **CORDIC** pode ser utilizado para encontrar diversos valores distintos, alterando principalmente os seus valores de entrada, seus parâmetros iniciais, além da definição do sentido de rotação no sistema de coordenadas utilizado.⁷⁷ É importante destacar que a demonstração a seguir considera que o ângulo que se deseja encontrar esteja definido em radianos. A fim de encontrar os valores do *seno* e do *coseno* de um ângulo β qualquer, os valores de y ou x do sistema de coordenadas do círculo unitário, relativos ao ângulo procurado, precisa ser identificado. O algoritmo **CORDIC** deve ser iniciado a partir do vetor v_0 definido na expressão da Equação A.1.

$$v_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (\text{A.1})$$

Na primeira iteração do algoritmo, este vetor deve ser rotacionado por um ângulo de 45° , no sentido anti-horário, a fim de encontrar o vetor v_1 . Em seguida, sucessivas iterações são realizadas para rotacionar os vetores subsequentes para uma ou outra direção, de acordo com o valor do ângulo procurado, a partir de valores cada vez menores, até o término da operação promovida pelo algoritmo. O tamanho de cada passo será definido pelo valor $\arctan(2^{-i})$, para $i = 1, 2, 3, \dots$. Na verdade o tamanho de cada passo tem um papel fundamental no processo de implementação do algoritmo, uma vez que estes

valores de referência tratam de uma potência de dois, e portanto operações de divisão ou multiplicação sobre eles podem ser realizados a partir do simples deslocamento de *bits*.

Formalmente, cada passo de iteração do algoritmo pode ser definido de acordo com a expressão da Equação A.2. Aplicando-se um operador de rotação sob o vetor corrente.

$$v_{i+1} = R_i v_i \quad (\text{A.2})$$

Um operador de rotação bastante útil neste contexto trata da matriz de rotação, como definida na expressão da Equação A.3.

$$R_i = \begin{bmatrix} \cos(\gamma_i) & -\sin(\gamma_i) \\ \sin(\gamma_i) & \cos(\gamma_i) \end{bmatrix} \quad (\text{A.3})$$

Utilizando-se de algumas identidades trigonométricas importantes, é possível indicar os valores de seno e cosseno de acordo com a expressão da Equação A.4.

$$\begin{aligned} \sin(\gamma_i) &= \frac{\tan(\gamma_i)}{\sqrt{1 + \tan^2(\gamma_i)}} \\ \cos(\gamma_i) &= \frac{1}{\sqrt{1 + \tan^2(\gamma_i)}} \end{aligned} \quad (\text{A.4})$$

Com isso, a partir destas identidades, é possível reescrever a expressão referente à matriz de rotação conforme a expressão definida na Equação A.5.

$$R_i = \frac{1}{\sqrt{1 + \tan^2(\gamma_i)}} \begin{bmatrix} 1 & -\tan(\gamma_i) \\ \tan(\gamma_i) & 1 \end{bmatrix} \quad (\text{A.5})$$

E portanto, de forma equivalente, com base na matriz de rotação encontrada, é possível determinar cada passo de evolução do algoritmo, a fim de identificar as posições dos vetores subsequentes em cada passo de sua evolução, dentro do círculo unitário, como definido na Equação A.6.

$$v_{i+1} = \frac{1}{\sqrt{1 + \tan^2(\gamma_i)}} \begin{bmatrix} 1 & -\tan(\gamma_i) \\ \tan(\gamma_i) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (\text{A.6})$$

Aqui, os valores de x_i e y_i referem-se aos componentes coordenados do vetor v_i , ou seja, o vetor localizado no passo anterior. Na sequência, para ângulos suficientemente pequenos, é possível restringir os valores das rotações de forma que $\tan(\gamma_i) = \pm 2^{-i}$. Dessa forma, a multiplicação pela tangente pode ser substituída pela divisão por uma potência de dois, e como destacado anteriormente, esta operação pode ser realizada de forma lógica e eficiente, a partir do simples deslocamento de *bits*, transformando o cálculo de uma

rotação vetorial em uma sequência de somas e deslocamentos de *bits* apenas. Com isso, a expressão toma a forma definida na Equação A.7.

$$v_{i+1} = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (\text{A.7})$$

Onde o valor de σ será utilizado apenas para determinar o sentido da operação de rotação, de forma que se o valor de γ_i for positivo, então $\sigma_i = +1$ e caso contrário $\sigma_i = -1$. Além disso, o valor de K_i é definido de acordo com a expressão da Equação A.8.

$$K_i = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (\text{A.8})$$

Os valores intermediário como definido em K_i podem ser removidos do processo iterativo do algoritmo. Afinal, este resultado pode ser aproximado para um valor constante, atuando exclusivamente como um fator de escala para a determinação do posicionamento do vetor v_i . Note que seu valor pode ser encontrado conhecendo a quantidade de iterações que devem ser realizadas pelo algoritmo, de acordo com a expressão definida na Equação A.9.

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (\text{A.9})$$

Esta aproximação pode ser tomada, pois as variações de ângulo realizadas em cada passo deste algoritmo vão se tornando cada vez menores, de forma que as contribuições intermediárias deste valor constante também o são. Perceba que este processo trata de uma aproximação apenas, uma vez que seus valores intermediários iriam adicionar contribuições diferentes em cada passo de iteração, porém, tratando-o como um valor constante, é removido este processo de multiplicação em cada iteração, garantindo um melhor desempenho ao algoritmo. Com isso, este valor pode ser calculado com antecedência, sendo mantido em uma tabela de conferência como um valor constante, dependente exclusivamente da quantidade de iterações n que será utilizado pelo algoritmo. Note que seu valor pode ser aproximado pelo limite indicado na expressão da Equação A.10, quando n tende ao infinito.

$$K = \lim_{n \rightarrow \infty} K(n) = \lim_{n \rightarrow \infty} \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (\text{A.10})$$

$$K \approx 0.6072529350088812561694$$

Após uma quantidade suficiente de iterações o ângulo encontrado por este algoritmo tende a ser muito próximo do ângulo β que se deseja encontrar. Para a grande maioria

das aplicações, após a realização de 40 iterações ($n = 40$) é possível obter uma acurácia no resultado até a décima casa decimal de precisão.

Por fim, é necessário apenas identificar o sentido de rotação de cada iteração, essa capacidade é alcançada a partir do operador σ destacado na expressão da Equação A.7. Para isso, é necessário manter o registro do quanto o vetor já foi rotacionado até a iteração corrente, e subtrair este registro do ângulo que se deseja encontrar. A partir deste resultado, se β_{i+1} é positivo, então a rotação deve ser realizada no sentido horário, com $\sigma = +1$, caso contrário, $\sigma = -1$ e a rotação será realizada no sentido anti-horário.

$$\begin{aligned}\beta_0 &= \beta \\ \beta_{i+1} &= \beta_i - \sigma_i \gamma_i \\ \gamma_i &= \arctan(2^{-i})\end{aligned}\tag{A.11}$$

Como descrito anteriormente, ao término deste processo, de acordo com os passos definidos pelo algoritmo o eixo y do vetor, no sistema de coordenadas, é suficiente para determinar o valor do *seno* do ângulo β , bem como o eixo x , que define o *coseno* deste ângulo.

Com isso, desmembrando a Equação A.7, é possível obter os componentes de cada eixo coordenado, conforme determinado na Equação A.12, que tratam dos valores de *seno* e *coseno*, respectivamente, em cada passo do algoritmo.

$$\begin{aligned}y_{i+1} &= y_i + x_i \sigma_i 2^{-i} \\ x_{i+1} &= x_i - y_i \sigma_i 2^{-i}\end{aligned}\tag{A.12}$$

A.2 Implementação

Esta sequência de operações pode ser implementada facilmente em linguagens de programação e mesmo a partir de linguagens de descrição de hardware, como descrito a seguir em pseudo-linguagem, a partir do código extraído da referência,⁷⁸ que foi tomado como base para a implementação do algoritmo no presente projeto.

```

1 Beta = [
2     atan 2^0,
3     atan 2^1,
4     ...
5     atan 2^(N-1)
6 ]
7
8 K = 1
9 for i in 0 .. N-1 {
```

```
10     K = K * cos(Beta[i])
11 }
12
13 function cordic(theta) {
14     cos = K
15     sin = 0
16     phi = theta
17
18     for i in 0 .. N-1 {
19         if phi > 0 {
20             dir = 1
21         } else {
22             dir = -1
23         }
24
25         cos = cos - (dir * (sin >> i))
26         sin = sin + (dir * (cos >> i))
27         phi = phi - (dir * Beta[i])
28     }
29
30     return cos, sin
31 }
```

Algoritmo 1 – Algoritmo para implementação do CORDIC, descrito em pseudo-linguagem, para a determinação do seno e cosseno de um ângulo indicado como parâmetro de entrada do algoritmo, como definido na referência.⁷⁸

Este algoritmo é suficiente para implementar exatamente o processo descrito na seção anterior. Das linhas 1-6 é indicada a definição da tabela de conferência que será utilizada para indicar as rotações a serem realizadas no vetor de referência para o cálculo do valor constante K do algoritmo, como definido na Equação A.8 e para uso no ângulo residual, calculado em cada iteração do algoritmo, utilizado para determinar a direção de rotação do próximo passo.

Das linhas 8-11 é definido o valor “constante” de K. E finalmente nas linhas 13-31 é determinado todo o código de implementação do algoritmo do **CORDIC** descrito anteriormente. Especificamente nas linhas 25-27 estão definidos os valores das Equações A.12 e A.11.

APÊNDICE B – CIRCUITO DE PROTEÇÃO

Para a operação da máquina proposta por este projeto foi necessária a construção de uma placa de circuito impresso de suporte aos *kits* de desenvolvimento DE10 Nano, a fim de garantir a integridade de seus pinos de comunicação de propósito geral. Este projeto excepcional se fez necessário devido a ausência deste suporte, presente de forma nativa em outros *kits* de desenvolvimento da fabricante, mas que foram omitidos no modelo utilizado neste projeto, provavelmente em busca de economia no processo de fabricação. Este circuito é importante pois previne a passagem de qualquer eventual corrente excessiva a partir do pinos de comunicação da **FPGA**, que podem ocorrer principalmente na variação de corrente durante o envio de um sinal para cada um destes pinos.

Este recurso foi utilizado para a implementação de um protocolo de comunicação paralelo, em um modo de transmissão *half-duplex*, como descrito em detalhes na Seção 5.1. Por este motivo, a probabilidade na ocorrência desta variação inesperada na corrente, durante a transmissão de sinais, é alta e suficiente para danificar definitivamente os pinos de comunicação da **FPGA**, o que poderia inviabilizar a aplicação do método proposto por este projeto.

B.1 Projeto

A visão geral do projeto implementado para as placas de circuito impresso construídas é mostrada em detalhes na Figura 52. Toda esta solução foi desenvolvida a partir da ferramenta **Autodesk Eagle**, uma aplicação destinada para o projeto e *layout* de placas de circuitos impressos. Este diagrama esquemático foi construído com base no *datasheet* da fabricante, referente ao *kit* de desenvolvimento DE1-SoC, também utilizado em outras versões deste projeto, para a implementação de uma máquina utilizada para a composição da dissertação de mestrado do presente autor. Este outro *kit* de desenvolvimento possui características muito semelhantes à DE10 Nano, tratando-se de outro modelo de entrada da fabricante. Sua principal limitação é referente à quantidade de recursos lógicos presentes em sua **FPGA**, em comparação com a **FPGA** presente na DE10 Nano. Os componentes eletrônicos, bem como suas versões específicas também foram obtidos com base neste modelo, a partir da análise no seus números de referência, impressos nos próprios componentes e descrições presentes no *datasheet*.

O circuito é relativamente simples, principalmente ao analisar cada pino de comunicação, com seu diagrama esquemático mostrado na Figura 28. Diferentemente da estratégia adotada nos *kits* de desenvolvimento que implementam este circuito, aqui, foi utilizado um banco de resistores em cada lado da comunicação. Esta estratégia foi proposta por tratar

Figura 52 – Diagrama esquemático utilizado para a elaboração do projeto do circuito de proteção dos pinos de entrada e saída de propósito geral do *kit* de desenvolvimento **DE10 Nano**. Elaborado a partir do software *Eagle 9.6.2*. Construído inspirado pelo *datasheet* da fabricante.¹

Fonte: Elaborada pelo autor.

de uma comunicação bidirecional, com sinais podendo ser transmitidos por ambos os lados deste circuito de proteção. Estes resistores tem a finalidade de adicionar um controle para a corrente encaminhada aos diodos *schottky*, que serão os verdadeiros responsáveis pela proteção sobre os sinais que passarão por este circuito. Este componente é composto por uma junção formada entre um semicondutor e um metal. A principal finalidade destes diodos é de limitar a tensão que passa entre seus terminais, se sua corrente for muito alta, com um tempo de resposta bastante rápido,⁷⁹ como pode ser visto na Figura 53.

Na Figura 52, os componentes identificados como *INPUT* e *OUTPUT* referem-se aos conectores que irão se comunicar com os *kits* de desenvolvimento e com o cabo de transmissão, respectivamente. Eles são formados por duas colunas de 20 pinos com especificações bastante semelhantes ao padrão **ATA/ATAPI-6**,⁵⁵ tomado como fonte de inspiração para a realização da comunicação neste projeto, como definido em detalhes na Figura 25. Cada pino do conector *INPUT* é conectado aos terminais dos bancos de

Figura 53 – Gráfico de especificação sobre o princípio de funcionamento do diodo *schottky*, capaz de limitar a tensão para valores altos de corrente, com tempo de resposta muito rápido.

Fonte: WALTERS⁷⁹

resistores $BRN[j]$ equivalente, com $j = 1..9$, a partir das vias de transmissão identificadas como $IN_D[i]$, com $i = 1..36$, como pode ser visto na figura. Perceba que os bancos de resistores restantes foram omitidos da figura para fins de apresentação apenas, é definido o total de 9 bancos de resistores neste projeto. O nome $BRN[i]$ foi adotado por tratar dos resistores que estarão dispostos no lado de baixo da placa. Cada resistor destes bancos são conectados a um diodo *schottky* **BAT54S** exclusivo $D[i]$, como definido na figura, a partir da via nomeada como $MID_D[i]$. Cada uma destas vias de comunicação é conectada aos bancos de resistores $TRN[i]$, que por sua vez são conectados aos pinos de comunicação do conector *OUTPUT*, através das vias $OUT_D[i]$.

Um dos circuitos mostrados no canto inferior direito da imagem refere-se ao sistema de operação de um **LED**, alimentado pelo pino 29 dos módulos de **GPIO** dos *kits* DE10 Nano, utilizado como fonte de alimentação de 3.3V por todo o circuito. O pino 11 deste módulo foi inativado por oferecer uma fonte de alimentação de 5V que não foi necessária para a operação dos recursos utilizados. Além disso, os pinos 12 e 30 do *kit* são utilizados em conjunto, como o sistema de aterramento deste circuito. O **LED** tem a finalidade exclusiva de testar o correto funcionamento da placa de circuito impresso, diante de sua ligação no *kit* de desenvolvimento. Ele é conectado a um resistor de 120Ω , para controlar a corrente que passará por este componente. Abaixo deste circuito pode-se verificar os

capacitores $C[k]$, com $k = 1..4$, que têm a finalidade de proteger este circuito contra uma queda eventual e inesperada no seu sistema de alimentação.

Figura 54 – Disposição do circuito e componentes distribuídos no lado superior do circuito impresso construído a partir do diagrama esquemático mostrado na Figura 52.

Fonte: Elaborada pelo autor.

A partir deste planejamento, foi iniciado o projeto da placa de circuito impresso, como pode ser visto nas Figuras 54 e 55. O principal desafio neste processo foi alcançar uma disposição dos componentes que fosse suficiente para que as vias de transmissão dos sinais tivessem a mesma distância entre os pinos dos conectores de comunicação. Esta disposição foi procurada a fim de garantir que estes sinais estivessem o mais sincronizados quanto possível, afinal, diferente do padrão **ATA/ATAPI-6**, que possui uma frequência de operação padrão de 125MHz, era esperada uma frequência de 250MHz para a operação deste sistema de comunicação. Entretanto, não existe uma ferramenta para a disposição automática das vias de transmissão no **Eagle**, para atingir esta precisão. Para isto, foram analisadas em uma planilha eletrônica extraordinária, todas as distâncias entre os pinos de entrada e saída do circuito e partir da ferramenta de *meandro** do **Eagle**, foi possível obter uma diferença máxima de 0,87mm, com distância máxima entre estes pontos de 41,25mm, dentre todos os pinos de comunicação dos módulos de entrada e de saída da placa.

* Vias de transmissão prolongadas intencionalmente utilizadas no processo de projeto de um circuito impresso com a finalidade de promover um atraso na propagação do sinal.⁸⁰

Figura 55 – Disposição do circuito e componentes distribuídos no lado inferior do circuito impresso construído a partir do diagrama esquemático mostrado na Figura 52.

Fonte: Elaborada pelo autor.

É possível notar pelas Figuras 54 e 55 que foram utilizadas diversas furações para interligar os planos de terra de ambos os lados da placa, posicionadas entre os componentes que formam o circuito. O espaçamento entre os bancos de resistores foram necessários devido a disposição dos pinos padrões do *kit* de desenvolvimento, que utiliza o sistema de alimentação e de terra em posições simétricas no conector de 2 – 20 pinos do **GPIO**. Este posicionamento permitiu a simetria para a disposição dos resistores e diodos, bem como a disposição das furações para interligar os planos de terra, porém, dificultou o planejamento para garantir a distância exata entre os pinos de entrada e saída dos conectores da placa. Foi buscada esta mesma simetria para a disposição dos capacitores e circuito para o funcionamento do LED, como podem ser vistos na Figura 54.

B.2 Implementação

As placas foram impressas e montadas a partir dos serviços da empresa chinesa **PCBWay**. A lista de componentes *SMDs* utilizados para a elaboração das placas estão dispostas na Tabela 11 e as fotos do lado de cima e de baixo de uma das placas montadas são mostradas nas Figuras 57 e 56, respectivamente. As placas ficaram com dimensões aproximadas de $6.2cm \times 5,8cm$. Foram incluídas no pedido o total de 5 placas já montadas, e 5 placas apenas impressas, sem a realização da montagem, a fim de lidar com eventuais

falhas que pudessem ser encontradas a posteriori.

Tabela 11 – Lista de componentes eletrônicos utilizados para a montagem do circuito impresso construído para proteção dos pinos de **GPIO** do *kit* de desenvolvimento DE10 Nano.

Qtt	Designator	Value	Footprint	Manufacturer	Manuf. Code
18	T/BRNj	22R0	CAY16	BOURNS	CAY10-220J4LF
2	C1, C3	100nF	C0805	KEMET	C104K3RACTU
2	C2, C4	10nF	C0805	KEMET	C103K3RACTU
36	Di	BAT54S	SOT23	NEXPERIA	BAT54S, 215
1	INPUT	FHEAD	FE20-2	SAMTEC	SSQ-120-01-L-D
1	LED1	PWR	LED0805	MULTICOMP	OVS-0801
1	OUTPUT	MHEAD	MA20-2	SAMTEC	HTSS-120-01-L-D
1	R1	1200	R0805	BOURNS	FX-1200ELF

Fonte: Elaborada pelo autor.

Figura 56 – Foto ilustrativa do lado inferior da placa de circuito impresso de proteção para os pinos de comunicação de propósito geral dos *kits* de desenvolvimento Terasic DE10 Nano.

Fonte: Elaborada pelo autor.

Durante a realização dos primeiros testes de encaixe destas placas nos *kits* de desenvolvimento, foi identificado um erro de projeto, para a utilização deste conjunto experimental. Os conectores destinados para ligação deste *kit* com os pinos de propósito geral do **Arduino**, são muito próximos aos seus conectores de **GPIO**, de forma que não era possível acomodar a placa de proteção de forma imediata sobre o *kit*. Para resolver este problema, foram adquiridos alguns conectores para montagem em placa de circuito

Figura 57 – Foto ilustrativa do lado superior da placa de circuito impresso de proteção para os pinos de comunicação de propósito geral dos *kits* de desenvolvimento Terasic DE10 Nano.

Fonte: Elaborada pelo autor.

impresso retos, como mostrado na Figura 58, que passou a ser utilizado em conjunto à placa construída, como um extensor. A versão reta deste conector foi utilizada para garantir as distâncias equivalentes entre os sinais.

Com isso, a partir deste aparato experimental, foi possível a plena capacidade para utilização desta placa de expansão, que tem a finalidade de garantir a proteção de todos os pinos de comunicação da **FPGA** contra eventuais flutuações na corrente. É interessante destacar que este projeto extraordinário pode ser explorado para o desenvolvimento de qualquer solução que utilize estes pinos de comunicação, em *kits* de desenvolvimento equivalentes à DE10 Nano.

Um questionamento sempre recorrente sobre a elaboração deste projeto foi a solução de cabeamento a ser utilizado para a comunicação entre os *kits* de desenvolvimento a partir dos módulos de **GPIO**. Isso porque a distribuição de seus pinos não possuem equivalência com qualquer solução comercial disponível no mercado, apesar de sua semelhança física com o padrão **IDE**. Para isto, foram adquiridos cabos **IDE** convencionais, desenvolvidos para uso de acordo com as especificações do padrão **ATA/ATAPI6**, que conta com vias de transporte conectadas ao aterramento deste padrão de comunicação, entre todas as vias de transmissão dos sinais efetivos, com a finalidade de atenuar a possibilidade de interferência entre os sinais transportados pelos cabos que compõem este padrão.

Figura 58 – Foto ilustrativa do extensor utilizado para corrigir a falha de projeto encontrada durante a realização dos testes preliminares, para uso do circuito de proteção sob os *kits* de desenvolvimento Terasic DE10 Nano.

Fonte: Elaborada pelo autor.

O formato de montagem destes conectores são conhecidos na literatura, porém, apenas um pino de aterramento é coincidente com a distribuição dos pinos na **GPIO** da DE10 Nano. Portanto, foi necessário realizar a desmontagem completa destes cabos **IDE** convencionais, isolando cada componente de sua composição. O sistema de aterramento deste padrão não é definido no seu manual e varia de acordo com a fabricante do conector. Nos modelos adquiridos para a elaboração deste projeto, existe uma placa de metal única, que promove o contato com as vias de aterramento deste padrão. Nesta placa de metal existem alguns cortes de alta precisão que permitem o deslocamento de pequenas estrias nos locais que devem fazer contato com os pinos de aterramento do padrão **ATA**, que conta com um total de 7 pinos para esta finalidade, sendo eles os pinos: 2, 19, 22, 24, 26, 30 e 40. Para uso neste projeto, apenas os pinos 12 e 30 são conectados ao sistema de aterramento do *kit* de desenvolvimento. Desta forma, todas as estrias foram retraídas, com exceção do pino 30 que é coincidente, permitindo que estes outros pinos pudessem ser utilizados para propósitos gerais no sistema de comunicação construído.

Na Figura 59 são destacados os dois únicos contatos que foram mantidos com o sistema de aterramento. É possível notar na figura que para que seja possível o uso deste sistema, é necessário que haja uma fenda no material de composição do conector, capaz de garantir a passagem destas estrias da placa de aterramento aos pinos equivalentes apenas. Os outros pinos mantém-se isolados desta placa com a finalidade de evitar a ocorrência de qualquer tipo de interferência. Dessa forma, para a conexão do pino 12 do *kit* de desenvolvimento a esta placa de aterramento, foi necessário criar uma fenda artificial, que

Figura 59 – Foto ilustrativa dos conectores construídos para a ligação dos kits de desenvolvimento a partir dos pinos de comunicação de propósito geral dos *kits* de desenvolvimento Terasic DE10 Nano.

Fonte: Elaborada pelo autor.

foi realizada a partir de uma fresadora de precisão. Com isso, para realizar o contato da placa de aterramento ao pino 12, foi adicionado uma pequena alavanca na fenda adicionada, formada por uma parte do material da placa de aterramento, removida de outras placas, que acabaram sendo partidas no processo desmontagem destes conectores. Com isso, a partir desta solução, foi possível a montagem de conectores adaptados para o padrão de comunicação proposto por este projeto, a partir do uso de cabos **IDE** convencionais.

Estes conectores adaptados foram montados novamente nos cabos originais, mas

Figura 60 – Foto ilustrativa do cabo construído para a comunicação entre os *kits* de desenvolvimento Terasic DE10 Nano.

Fonte: Elaborada pelo autor.

que foram cortados para atingir o tamanho de apenas 15cm, dimensão sugerida para garantir a capacidade destes cabos em operar na frequência proposta para este sistema de comunicação, a fim de evitar a ocorrência de interferências que poderiam surgir para tamanhos de cabos maiores. Uma foto ilustrativa deste cabo, montado para a operação deste sistema, pode ser visto na Figura 60. Pode-se perceber que ele segue exatamente os mesmos preceitos do padrão **ATA/ATAPI6**, mas conta com todas as modificações de adaptação descritas neste apêndice.

A partir do uso de todo este aparato experimental, passou a ser possível a comunicação entre as **FPGAs** utilizadas para armazenar a máquina criada para a composição desta tese. Na Figura 61 pode ser vista uma pequena parte deste sistema, para a realização da comunicação entre apenas dois *kits* DE10 Nano. Foi adotada a convenção de que apenas o módulo **GPIO_0**, na composição destes *kits*, seria utilizado como o mestre da comunicação, como descrito em detalhes na Seção 5.1, e desta forma, apenas neste módulo que será utilizada a placa de expansão de proteção dos pinos. Perceba que apesar desta convenção todos os sinais trocados em ambos os sentidos da comunicação, passarão necessariamente pelos diodos *schottky* de proteção, garantindo uma capacidade plena para a utilização deste sistema de comunicação, bidirecional, em modo de operação *half-duplex*, sem a possibilidade de gerar prejuízos a qualquer pino dos módulos de **GPIO** dos *kits* de desenvolvimento utilizados.

Figura 61 – Foto ilustrativa de parte do aparato experimental, com destaque para a ligação entre os *kits* de desenvolvimento Terasic DE10 Nano utilizadas para a elaboração deste projeto, com as placas de proteção e cabos de comunicação construídas.

Fonte: Elaborada pelo autor.

Anexos

ANEXO A – PROGRAMAS

A.1 SISAL

```

1 export Main
2 function Fact(x, y: integer returns integer)
3     if x <= y then y
4     else Fact(x, ((x+y)/2) + 1) + Fact((x+y)/2, y)
5     end if
6 end function
7
8 function Main(returns integer)
9     Fact(100, 1)
10 end function

```

Algoritmo 1 – Algoritmo em alto nível, descrito em linguagem Sisal, para a implementação de um programa para o cálculo do fatorial de um valor definido dentro do programa principal, implementado para realizar o cálculo da operação de forma duplamente recursiva. O programa foi alterado para realizar o cálculo da soma de Gauss, a fim de permitir a mesma complexidade para o cálculo de valores maiores, sem a necessidade de implementação de soluções para a representação de números muito grandes.

```

1 export Main
2 function F(x: integer returns integer)
3     x*x + 6*x + 10
4 end function
5
6 function Area(l, r, depth: integer returns integer)
7     let
8         mid := (l+r)/2;
9     in
10         if depth <= 1 then
11             (r-l) * F(mid)
12         else
13             Area(l, mid, depth-1) + Area(mid, r, depth-1)
14         end if
15     end let
16 end function
17

```

```

18 function Main(a, b, depth: integer returns integer)
19     Area(a, b, depth)
20 end function

```

Algoritmo 2 – Algoritmo em alto nível, descrito em linguagem Sisal, para a implementação de um programa destinado para o cálculo de uma função de integração, como definida na expressão da Equação 6.4, a partir do método de implementação do trapézio, também implementado para se expandir de forma duplamente recursiva.

```

1 export Main
2 function Trap (returns real)
3     for initial
4         int := 0.0;
5         y   := 0.0;
6         x   := 0.02
7     while x < 1.0
8     repeat
9         int := 0.01 * (old y + y);
10        y   := old x * old x;
11        x   := old x + 0.02
12    returns
13        value of sum int
14    end for
15 end function
16
17 function Main(returns real)
18     Trap
19 end function

```

Algoritmo 3 – Algoritmo em alto nível, descrito em linguagem Sisal, para a implementação de um programa destinado para o cálculo de uma função de integração, utilizando o método dos retângulos. Contrário ao método do programa anterior, este realiza o cálculo numérico a partir de um laço, ao invés da recursividade. Programa retirado do artigo de GURD.¹⁴

```

1 export Main
2 function Fib (n: integer returns integer)
3     if n <= 1 then n
4     else Fib(n-1) + Fib(n-2)
5     end if
6 end function
7

```

```

8 function Main
9     Fib(15)
10 end function

```

Algoritmo 4 – Algoritmo em alto nível, descrito em linguagem Sisal, para a implementação de um programa destinado para o cálculo de um número qualquer da sequência de Fibonacci, indicado como parâmetro de entrada da função, implementado de forma duplamente recursiva, inspirado pelo programa definido em BROWNE.⁷²

A.2 Linguagem Montadora

```

1      0 ONKILO 0LBY
2      ;!$F FAT.rel
3      1 OLSWA1 2RBY1 4RBY
4      1 1RSCD0 0LBY
5      1 2LSYN1 3LBYI1000
6      1 3NDUP1 5LBY1 7LEW
7      1 4LSYN1 8LEWI1
8      1 5NGAN1 7REW1 6LBY
9      1 6NDUP1 8REW1 9LBY
10     1 7LSAN1 10LEW1 11LEW
11     1 8LSAN1 10REW1 14LEW
12     1 9LPRP1 1REWD0 2LBY
13     1 10NCLI1 11REW1 14REW
14     1 11LBRR1 12LBY1 13LBY
15     1 12NDUP1 17LEW1 20LBY
16     1 13NDUP1 22LEW1 25LEW
17     1 14DBRR1 15LBY1 1LEW
18     1 15NDUP1 17REW1 16LBY
19     1 16NDUP1 25REW1 31LEW
20     1 17NADI1 18LBY
21     1 18LDMD1 19LBYI2
22     1 19NADI1 23LEWI1
23     1 20NGAN1 22REW1 21LBY
24     1 21NDUP1 23REW1 24LBY
25     1 22LSAN1 10LEW1 11LEW
26     1 23LSAN1 10REW1 14LEW
27     1 24LPRP1 1REWD0 1LEW
28     1 25NADI1 26LBY
29     1 26LDMD1 27LBYI2
30     1 27NDUP1 28LBY1 30LEW

```

```

31      1 28NGAN1 30REW1 29LBY
32      1 29NDUP1 31REW1 32LBY
33      1 30LSAN1 10LEW1 11LEW
34      1 31LSAN1 10REW1 14LEW
35      1 32LPRP1 1REWDO 1REW
36      0 1NADI1 1LEW
37      0 2ROPT0 0LBYG0

```

Algoritmo 5 – Programa em linguagem montadora (grafo a fluxo de dados) para a composição do programa definido no Algoritmo 1, destinado à realização do calculo do fatorial, modificado para a realização da soma de *Gauss*.

```

1      0 ONKILO 0LBY
2      ;!$F BIN.rel
3      1 5NKILO 0LBY
4      1 6RSCD0 0LBY
5      1 7NGAN1 8LBY1 9LBY
6      1 8NDUP1 10REW1 11REW
7      1 9NDUP1 12REW1 13LBY
8      1 10LSAN1 14LEW1 3LEW
9      1 11LSAN1 14REW1 2LEW
10     1 12LSAN1 16LBY1 1LEW
11     1 13LPRP1 6REWDO 2LBY
12     1 14NADI1 15LBY
13     1 15LDMD1 4LEWI2
14     1 16NCLI1 0LBYY1
15     1 0NTUP1 1REW04
16     1 1LBRR1 22LBY1 30LBY
17     1 2DBRR1 35LEW1 38LEW
18     1 3DBRR1 17LBY1 38REW
19     1 17NDUP1 23LBY1 26LEW
20     1 4DBRR1 18LBY1 20LBY
21     1 18NDUP1 27LEW1 19LBY
22     1 19NDUP1 31LBY1 34LEW
23     1 20NDUP1 39LEW1 21LBY
24     1 21NDUP1 39REW1 40RBY
25     1 22NSBI1 28LEWI1
26     1 23NGAN1 24LBY1 25LBY
27     1 24NDUP1 26REW1 27REW
28     1 25NDUP1 28REW1 29LBY
29     1 26LSAN1 14LEW1 3LEW
30     1 27LSAN1 14REW1 2LEW
31     1 28LSAN1 16LBY1 1LEW

```



```

32      1 29LPRP1 6REWD0 1LEW
33      1 30NSBI1 36LEWI1
34      1 31NGAN1 32LBY1 33LBY
35      1 32NDUP1 34REW1 35REW
36      1 33NDUP1 36REW1 37LBY
37      1 34LSAN1 14LEW1 3LEW
38      1 35LSAN1 14REW1 2LEW
39      1 36LSAN1 16LBY1 1LEW
40      1 37LPRP1 6REWD0 1REW
41      0 1NADI1 6LEW
42      1 38NSBI1 43LEW
43      1 39NMLI1 41LEW
44      1 40NMLI1 41REWI6
45      1 41NADI1 42LBY
46      1 42NADI1 43REWI10
47      1 43NMLI1 6LEW
48      1 44NDUP1 45LBY
49      1 45NDUP1 7LBY1 10LEW
50      1 46NDUP1 11LEW
51      1 47NDUP1 12LEW
52      0 2ROPT0 0LBYG0

```

Algoritmo 6 – Programa em linguagem montadora (grafo a fluxo de dados) para a composição do programa definido no Algoritmo 2, destinado à realização do calculo da função de integração definida na Equação 6.4, operando sobre valores inteiros apenas, nos intervalos definidos como parâmetros de entrada do programa.

```

1      0 ONKILO 0LBY
2      ;!$F BIN.rel
3      1 5NKILO 0LBY
4      1 6RSCD0 0LBY
5      1 7NGAN1 8LBY1 9LBY
6      1 8NDUP1 10REW1 11REW
7      1 9NDUP1 12REW1 13LBY
8      1 10LSAN1 14LEW1 3LEW
9      1 11LSAN1 14REW1 2LEW
10     1 12LSAN1 16LBY1 1LEW
11     1 13LPRP1 6REWD0 2LBY
12     1 14NADR1 15LBY
13     1 15NMLR1 4LEWR0.5
14     1 16NCLI1 0LBYY1
15     1 0NTUP1 1REW04

```

```
16      1 1LBRR1 22LBY1 30LBY
17      1 2DBRR1 35LEW1 38LEW
18      1 3DBRR1 17LBY1 38REW
19      1 17NDUP1 23LBY1 26LEW
20      1 4DBRR1 18LBY1 20LBY
21      1 18NDUP1 27LEW1 19LBY
22      1 19NDUP1 31LBY1 34LEW
23      1 20NDUP1 39LEW1 21LBY
24      1 21NDUP1 39REW1 48LBY
25      1 22NSBI1 28LEWI1
26      1 23NGAN1 24LBY1 25LBY
27      1 24NDUP1 26REW1 27REW
28      1 25NDUP1 28REW1 29LBY
29      1 26LSAN1 14LEW1 3LEW
30      1 27LSAN1 14REW1 2LEW
31      1 28LSAN1 16LBY1 1LEW
32      1 29LPRP1 6REWDO 1LEW
33      1 30NSBI1 36LEWI1
34      1 31NGAN1 32LBY1 33LBY
35      1 32NDUP1 34REW1 35REW
36      1 33NDUP1 36REW1 37LBY
37      1 34LSAN1 14LEW1 3LEW
38      1 35LSAN1 14REW1 2LEW
39      1 36LSAN1 16LBY1 1LEW
40      1 37LPRP1 6REWDO 1REW
41      0 1NADR1 6LEW
42      1 38NSBR1 43LEW
43      1 39NMLR1 41LEW
44      1 40NMLR1 41REWR6.0
45      1 41NADR1 42LBY
46      1 42NADR1 43REWR10.0
47      1 43NMLR1 6LEW
48      1 44NDUP1 45LBY
49      1 45NDUP1 7LBY1 10LEW
50      1 46NDUP1 11LEW
51      1 47NDUP1 12LEW
52      1 48NDUP1 49LBY1 50LBY
53      1 49NSIN1 51LEW
54      1 50NCOS1 51REW
55      1 51NMLR1 40RBY
```

56 0 2ROPT0 0LBYG0

Algoritmo 7 – Programa em linguagem montadora (grafo a fluxo de dados) para a composição do programa definido no Algoritmo 2, destinado à realização do calculo da função de integração operando sobre valores em ponto flutuante com o uso de funções trigonométricas, como definido na Equação 6.5, nos intervalos definidos como parâmetros de entrada do programa.

```

1      0 ONKILO 0LBY
2      0 1ROPT0 0LBYG0
3      1 0NCGR1 1LBYR1.0
4      1 1NDUP1 2REW1 3LBY
5      1 2DBRR1 16LBY1 14LEW
6      1 3NDUP1 4REW1 5REW
7      1 4LBRR1 10LEW
8      1 5LBRR1 6LBY1 7LBY
9      1 6NDUP1 8LEW1 8REW
10     1 7NADR1 9LBYR0.02
11     1 8NMLR1 10REW1 13LBY
12     1 9LADL1 11LBYI1
13     1 10NADR1 12RBY
14     1 11NDUP1 0RBY1 5LBY
15     1 12NMLR1 14REWRO.01
16     1 13LADL1 4LBYI1
17     1 14NADR1 15LBY
18     1 15LADL1 2LBYI1
19     1 16LSILO 1LBY00
20     1 17LSYN1 2LEW
21     1 18LSYN1 4LEW
22     1 19LSYN1 0RBY
23     1 20LSYN1 5LEW

```

Algoritmo 8 – Programa em linguagem montadora (grafo a fluxo de dados) para a composição do programa definido no Algoritmo 3, destinado à realização do calculo da função de integração utilizando o método do retângulo, implementado sob a realização de um laço, inspirado pelo código disposto em GURD¹⁴

```

1      0 ONKILO 0LBY
2      0 1ROPT0 0LBYG0
3      0 2LSANO 1LBY
4      1 0LSYN1 33LBYR19.0
5      1 1NGAN1 2LBY

```

```

6      1 2NDUP1 6LBY1 3LBY
7      1 3NDUP1 5REW1 4RBY
8      1 4LSAN1 11REWDO 2LEW
9      1 5LSAN1 7LBY
10     1 6LSWA0 2REW
11     1 7NDUP1 8LBY1 9LEW
12     1 8NCGR1 9REWR2.0
13     1 9DBRR1 10LBY1 12LBY
14     1 10LSYN1 11LEWR1.0
15     1 11LSDS0 0LBY
16     1 12NDUP1 22LBY1 23LBY
17     1 13NDUP1 14LBY1 18LEW
18     1 14NGAN1 15LBY
19     1 15NDUP1 19LBY1 16LBY
20     1 16NDUP1 18REW1 17RBY
21     1 17LSAN1 11REWD1 20LEW
22     1 18LSAN1 7LBY
23     1 19LSWA1 20REW
24     1 20LSAN1 21LEW
25     1 21NADR1 11LEW
26     1 22NSBR1 13LBYS1.0
27     1 23NSBR1 24LBYS2.0
28     1 24NDUP1 25LBY1 29LEW
29     1 25NGAN1 26LBY
30     1 26NDUP1 30LBY1 27LBY
31     1 27NDUP1 29REW1 28RBY
32     1 28LSAN1 11REWD1 31LEW
33     1 29LSAN1 7LBY
34     1 30LSWA1 31REW
35     1 31LSAN1 21REW
36     1 32LSWA1 0RBY
37     1 33NSBR1 34LBYS1.0
38     1 34NDUP1 1LBY1 5LEW

```

Algoritmo 9 – Programa em linguagem montadora (grafo a fluxo de dados) para a composição do programa definido no Algoritmo 4, destinado à realização do cálculo de um número na sequência de *Fibonacci*, com limite definido como parâmetro de entrada da função, implementado de forma duplamente recursiva, inspirado pelo programa obtido em BROWNE.⁷²

A.3 C++

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 long unsigned int f(long unsigned int x) {
6     return x*x+6*x+10;
7 }
8
9 long unsigned int area(long unsigned int l, long unsigned int r,
10     long unsigned int depth) {
11     if (depth <= 1) {
12         long unsigned int mid = (r-l)/2;
13         return (r-l)*f(l+mid);
14     } else {
15         long unsigned int mid1 = (r-l)/3;
16         long unsigned int mid2 = mid1*2;
17
18         long unsigned int fL,fC,fR;
19
20         #pragma omp task shared(fL)
21         fL=area(l, l+mid1, depth-1);
22
23         #pragma omp task shared(fC)
24         fC=area(l+mid1, l+mid2, depth-1);
25
26         #pragma omp task shared(fR)
27         fR=area(l+mid2, r, depth-1);
28
29         #pragma omp taskwait
30         return (fL+fC+fR);
31     }
32 }
33
34 int main (int argc, char* argv[]) {
35     long unsigned int a, b, depth,R;
36     int Nthreads;
37
38     a = 0;
39     b = 1024;
40     depth = atoi(argv[1]);
41
42     Nthreads = atoi(argv[2]);
```

```

42  omp_set_num_threads(Nthreads);
43  #pragma omp parallel
44  {
45      #pragma omp single nowait
46      {
47          R=area(a,b,depth);
48      }
49  }
50
51  printf("%lu\n", R);
52  return 0;
53 }

```

Algoritmo 10 – Programa descrito em linguagem C++, destinado para realizar o cálculo numérico da função de integração definida na Equação 6.4, a partir do método do trapézio, operando sobre valores inteiros apenas, utilizado para a elaboração dos gráficos dispostos na Figura 50, utilizando o *OpenMP* para a exploração de paralelismo em arquiteturas convencionais.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <omp.h>
5
6  float f(float x) {
7      return x*x+6*cos(x)*sin(x)+10;
8  }
9
10 float area(float l, float r, unsigned long int depth) {
11     float mid = (l+r)*0.5;
12
13     if (depth <= 1) {
14         return (r-l)*f(mid);
15     } else {
16         float fL, fR;
17
18         #pragma omp task shared(fL)
19         fL=area(l,mid,depth-1);
20
21         #pragma omp task shared(fR)
22         fR=area(mid,r,depth-1);
23
24         #pragma omp taskwait

```

```
25     return (fL+fR);
26 }
27 }
28
29 int main (int argc, char* argv[]) {
30     float a, b, R;
31     unsigned long int depth;
32     int Nthreads;
33
34     a = 0;
35     b = 1;
36     depth = atoi(argv[1]);
37
38     Nthreads = atoi(argv[2]);
39     omp_set_num_threads(Nthreads);
40     #pragma omp parallel
41     {
42         #pragma omp single nowait
43         {
44             R=area(a,b,depth);
45         }
46     }
47
48     printf("%f\n", R);
49     return 0;
50 }
```

Algoritmo 11 – Programa descrito em linguagem C++, destinado para realizar o cálculo numérico da função de integração definida na Equação 6.5, a partir do método do trapézio, operando sob valores em ponto flutuante com o uso de operações trigonométricas, utilizado para a elaboração dos gráficos dispostos na Figura 50, utilizando o *OpenMP* para a exploração de paralelismo em arquiteturas convencionais.