UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

"ESTUDO DE DESEMPENHO DE REDES
DE INTERCONEXÃO PARA A ARQUITETURA
A FLUXO DE DADOS PROTO-WOLF"

João Angelo Martini

*Dissertação apresentada ao Instituto
de Física de São Carlos, Universidade
de São Paulo, para obtenção do Título
de Mestre em Ciências Física Apli-
cada - Subárea Física Computacional*

ORIENTADOR: *Prof. Dr. Álvaro Garcia Neto*

DEPARTAMENTO DE FÍSICA E INFORMÁTICA

SÃO CARLOS

1994

142

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE JOAO ANGELO MARTINI APRESENTADA AO
INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 21/10/1994

COMISSÃO JULGADORA:

Prof. Dr. Roberto Gomes Neto

Prof. Dr. Jan Frens Willem Blarte

Prof. Dr. Jose Maria Leite

“O Senhor é o meu pastor,
nada me faltará.”
Salmo 23

“À meu pai querido Angelo Martini,
que a bondade do Senhor nos permita
o reencontro na Pátria espiritual.
À minha mãe Helenice Maenaka
Martini, grande incentivadora, pelo
amor e confiança.
À minha esposa Vera, pelo amor,
paciência, compreensão e incentivo.
À pequena Mariane, pela alegria que
trouxe ao nosso lar. ”

Agradecimentos

Ao Prof. Dr. Álvaro Garcia Neto, pela amizade e incentivo, pelo apoio inestimável, pela orientação no desenvolvimento deste trabalho.

Ao Grupo de Instrumentação e Informática do DFI (Departamento de Física e Informática).

Aos professores Luciano da Fontoura Costa, Carlos Antônio Ruggiero e Edson dos Santos Moreira cujos ensinamentos ofereceram-me ótimas oportunidades de aprendizado.

Ao CNPq pelo apoio financeiro.

Ao Cláudio e Valdir pelo auxílio na utilização do Vax e das Sun's.

Ao técnico Lírio O. B. de Almeida pela amizade e espírito de colaboração.

Aos técnicos do Laboratório de Instrumentação Eletrônica (LIE) pela amizade e suporte técnico.

Às secretárias Lia e Maria Helena pela atenção e competente trabalho de secretaria.

Ao amigo Luiz H. P. de Godoy da Coordenadoria de Divulgação Científica e Cultural (CDCC) pelo apoio, compreensão e incentivo.

À Wladerez A. G. Caiado da seção de pós-graduação, pela amizade e colaboração.

Aos professores do Departamento de Engenharia Elétrica da EESC, Adilson Gonzaga e Jerson Vargas pelo incentivo e amizade.

Aos amigos Nelson Corona Jr., Patrícia Magna, Giulio P. Frigério, Regina F. Eto e Márcio A. Marques companheiros de trabalho e de cafezinho, pela amizade e incentivo.

Ao João E. M. Perea Martins, amigo e incentivador, pelo apoio constante durante o desenvolvimento deste trabalho.

Ao amigo Marcos Antônio Cavenaghi pelo companheirismo e amizade, pelo auxílio inestimável nas dúvidas com o simulador SAW e pelo exemplo de dedicação ao trabalho.

Ao amigo e irmão espiritual Osvaldo Luis Asato cuja amizade e apoio sempre presentes nos momentos difíceis muito contribuíram para superar os obstáculos encontrados no desenvolvimento deste trabalho.

Aos meus familiares que me incentivaram e apoiaram nos momentos críticos.

A todos que direta ou indiretamente colaboraram no desenvolvimento deste trabalho.

Conteúdo

1	Arquiteturas	1
1.1	Introdução	1
1.1.1	Flynn	2
1.1.2	Treleaven	4
1.1.3	Feng	5
1.1.4	Händler	7
1.1.5	Shore	8
1.2	Arquiteturas Dataflow	13
1.2.1	MTTDA	15
1.2.2	Monsoon	16
1.2.3	VIM	18
1.2.4	Epsilon-1	19
1.2.5	Epsilon-2	19
1.2.6	SIGMA-1	21
1.2.7	EM-4	22
1.2.8	MMDM	23
1.3	Conclusões	23
2	Redes de Interconexão	25
2.1	Introdução	25
2.2	Comunicação em Arquiteturas Paralelas	26
2.3	Topologias	28
2.3.1	Elementos de Chaveamento	30
2.3.2	Categorias de Topologias	31

2.3.2.1	Topologias Estáticas	31
2.3.2.2	Topologias Dinâmicas	33
2.3.3	Escolha de Topologia	33
2.4	Decisões de Projeto	34
2.5	Classificação das Redes de Interconexão	36
2.6	Descrições	37
2.6.1	Arranjo Linear	37
2.6.2	Anel	38
2.6.3	Estrela	38
2.6.4	Malha	39
2.6.5	Árvore	39
2.6.6	n-Cubo	39
2.6.7	Barramento Compartilhado	40
2.6.8	Crossbar	40
2.6.9	Redes de Interconexão Multi-Estágio	41
2.7	Permutações	43
2.7.1	Perfect-Shuffle	44
2.7.2	Exchange	45
2.7.3	Butterfly	45
2.7.4	Shift	46
2.7.5	Identidade	48
2.8	Aplicação de Permutações	49
2.8.1	Omega	49
2.8.2	Banyan	49
2.8.3	n-Cubo	52
2.9	Rede de Interconexão GSN	52
2.9.1	Padrão de Interconexão Generalized Shuffle	53
2.9.2	Projeto de uma GSN	54
2.9.3	Equivalências	55
2.10	Conclusões	55

3	Arquitetura Proto-WOLF	57
3.1	Introdução	57
3.2	Filosofia do Proto-WOLF	57
3.2.1	Problemas conhecidos	58
3.2.2	Alvos do Proto-WOLF	59
3.3	Descrição Funcional	60
3.3.1	Tipos de Fichas	60
3.3.2	Unidades	62
3.4	Requisitos das Chaves de Conexão	66
3.4.1	Características da Chave de Coleta	68
3.4.2	Características da Chave de Distribuição	69
3.5	Conclusões	69
4	Implementação e Análise das Redes	71
4.1	Introdução	71
4.2	Simulador SAW	71
4.2.1	Descrição Funcional do Simulador SAW	71
4.3	Definição das Redes de Conexão	73
4.4	Candidatas às Redes de Conexão	74
4.5	Implementação Ideal das Chaves de Distribuição e de Coleta	74
4.5.1	Cross1	75
4.5.2	Cross2	75
4.6	Avaliação Estática da Chave de Coleta	76
4.6.1	Modelo Lógico	77
4.6.2	Implementação da Chave de Coleta	78
4.6.2.1	Crossbar 8x2	78
4.6.2.2	Rede 8x2 GSN-derivada	79
4.6.2.3	Rede 8x2 Delta-derivada	80
4.6.3	Expansibilidade	81
4.7	Avaliação Estática da Chave de Distribuição	82
4.7.1	Modelo Lógico	82

4.7.2	Implementação	83
4.8	Avaliação Dinâmica	84
4.9	Resultados	86
4.9.1	Incremento	88
4.9.2	Ackermann	91
4.9.3	Integração Binária	94
4.9.4	Multiplicação de Matrizes	97
4.10	Conclusões	100
5	Conclusões e Trabalhos Futuros	103
5.1	Considerações Finais	103
6	Bibliografia	105

Lista de Figuras

1.1	SISD	2
1.2	SIMD	3
1.3	MISD	3
1.4	MIMD	4
1.5	Máquina I	9
1.6	Máquina II	10
1.7	Máquina III	10
1.8	Máquina IV	11
1.9	Máquina V	12
1.10	Máquina VI	12
1.11	Grafo para $a = (b + 1) * (b - c)$	14
1.12	MTTDA (<i>MIT Tagged-Token Dataflow Architecture</i>)	16
1.13	Arquitetura Monsoon	17
1.14	Arquitetura VIM	18
1.15	Arquitetura Epsilon-2	20
1.16	Arquitetura SIGMA-1	21
1.17	Arquitetura EM-4	22
1.18	Arquitetura MMDM	24
2.1	Sistema Multiprocessador	27
2.2	Sistema Multicomputador	27
2.3	Conexão Total	29
2.4	Elementos de Chaveamento	31
2.5	Topologias Estáticas	32

2.6	Topologias Dinâmicas	33
2.7	Classificação de Redes de Interconexão	38
2.8	Controle do Elemento de Chaveamento	42
2.9	Permutação <i>Perfect-Shuffle</i>	44
2.10	Interligação dos módulos após aplicação da <i>Perfect-Shuffle</i>	45
2.11	Permutação <i>Exchange</i>	46
2.12	Interligação dos módulos após aplicação da <i>Exchange</i>	46
2.13	Permutação <i>Butterfly</i>	47
2.14	Interligação dos módulos após aplicação da <i>Butterfly</i>	47
2.15	Permutação <i>Shift</i>	48
2.16	Interligação dos módulos após aplicação da <i>Shift</i>	48
2.17	Rede Omega para N=8 módulos	50
2.18	Rede Banyan para N=8 módulos	51
2.19	Rede n-Cubo para N=8 módulos	52
2.20	<i>Generalized Shuffle</i> de M=4x2	55
3.1	Código seqüencial	59
3.2	Arquitetura Proto-WOLF	63
4.1	Arquitetura simulada pelo SAW	72
4.2	Chave de Distribuição - Cross1	75
4.3	Chave de Coleta - Cross2	76
4.4	Chave de Coleta	77
4.5	Chave de Coleta - Crossbar 8x2	78
4.6	Chave de Coleta - GSN-derivada 8x2	80
4.7	Chave de Coleta - Delta-derivada 8x2	81
4.8	Chave de Distribuição	82
4.9	Chave de Distribuição - Demultiplexador	83
4.10	Programa Incremento	84
4.11	Programa Ackermann	85
4.12	Programa Integração Binária	86
4.13	Programa Multiplicação de Matrizes	87

4.14	Processo de Compilação de Programa em SISAL	88
4.15	Ocupação da Memória de Dados para Incremento (Cross1-Gsn)	89
4.16	Ocupação da Fila de Fichas para Incremento (Cross1-Gsn)	90
4.17	Ocupação da Memória de Dados para Incremento (Demultiplexador-Gsn) .	90
4.18	Ocupação da Fila de Fichas para Incremento (Demultiplexador-Gsn) . . .	91
4.19	Ocupação da Memória de Dados para Ackermann (Cross1-Gsn)	92
4.20	Ocupação da Fila de Fichas para Ackermann (Cross1-Gsn)	93
4.21	Ocupação da Memória de Dados para Ackermann (Demultiplexador-Crossbar)	93
4.22	Ocupação da Fila de Fichas para Ackermann (Demultiplexador-Crossbar) .	94
4.23	Ocupação da Memória de Dados para Integração Binária (Cross1-Delta) . .	95
4.24	Ocupação da Fila de Fichas para Integração Binária (Cross1-Delta)	96
4.25	Ocupação da Memória de Dados para Integração Binária (Demultiplexa- dor-Gsn)	96
4.26	Ocupação da Fila de Fichas para Integração Binária (Demultiplexador-Gsn)	97
4.27	Ocupação da Memória de Dados para Multiplicação de Matrizes (Cross1-Gsn)	98
4.28	Ocupação da Fila de Fichas para Multiplicação de Matrizes (Cross1-Gsn) .	99
4.29	Ocupação da Memória de Dados para Multiplicação de Matrizes (Demul- tiplexador-Cross2)	99
4.30	Ocupação da Fila de Fichas para Multiplicação de Matrizes (Demultiple- xador-Cross2)	100

Lista de Tabelas

1.1	Classificação de Treleaven	5
2.1	Características Ideais	30
2.2	Características Operacionais	36
4.1	Resultados para Incremento	89
4.2	Parâmetros para Incremento	89
4.3	Resultados para Ackermann	91
4.4	Parâmetros para Ackermann	92
4.5	Resultados para Integração Binária	94
4.6	Parâmetros para Integração Binária	95
4.7	Resultados para Multiplicação de Matrizes	97
4.8	Parâmetros para Multiplicação de Matrizes	98

Resumo

Este trabalho estuda as redes de interconexão da arquitetura a fluxo de dados Proto-WOLF, descreve suas implementações através do simulador da arquitetura (SAW - Simulador da Arquitetura Proto-WOLF) e identifica qual tipo de chave é mais adequada.

A fim de contextualizar o trabalho, são apresentados os fundamentos de redes de interconexão e as topologias mais amplamente utilizadas em arquiteturas MIMD, também são dadas as descrições de algumas das mais conhecidas máquinas a fluxo de dados.

São dadas as descrições do estágio atual do simulador SAW e da arquitetura Proto-WOLF a fim de definir as características das chaves de interconexão mais adequadas à arquitetura. Os resultados da implementação dessas redes são apresentados e analisados, verificando-se que a combinação Cross1-Gsn apresenta o melhor desempenho.

Abstract

This work presents the Proto-WOLF interconnection networks and describes the implementations in the simulator SAW and identifying the best network configuration. A survey of interconnection networks and a description of some dataflow machines is presented in order to contextualize the work.

The current version of the simulator is presented in order to define the features of the interconnection networks that best fit the Proto-WOLF architecture. Simulation results are presented and analysed to validate the work, and indicate that the Cross1-Gsn combination has the best characteristics.

Capítulo 1

Arquiteturas

1.1 Introdução

Esse capítulo descreve alguns dos principais esquemas de classificação de arquiteturas de computadores e discute os conceitos fundamentais do modelo de fluxo de dados. A fim de situar as arquiteturas baseadas no paradigma de fluxo de dados no contexto da pesquisa em arquiteturas paralelas são dadas as descrições básicas de algumas das mais conhecidas máquinas a fluxo de dados. Entre os principais esquemas de classificação estão:

- Flynn [Fly72]
- Treleaven [AG89]
- Feng [HB84]
- Händlers [Qui87, HB84]
- Shore [HJ88]

Cada esquema de classificação é baseado na estrutura ou comportamento das arquiteturas.

1.1.1 Flynn

Provavelmente a classificação mais amplamente conhecida seja a sugerida por M. J. Flynn. A taxonomia de Flynn tem como base os conceitos de fluxo de instruções e fluxo de dados. Um fluxo de instruções é uma seqüência de instruções realizada por um computador; um fluxo de dados é uma seqüência de dados usada para executar um fluxo de instruções. A classificação de Flynn identifica quatro categorias de arquiteturas:

SISD (Single Instruction Single Data) São as máquinas seqüenciais em que uma única instrução opera sobre um único dado. A arquitetura convencional de von Neumann se enquadra nessa categoria.

Figura 1.1: SISD

SIMD (Single Instruction Multiple Data) Corresponde às arquiteturas de arranjos de processadores, múltiplos processadores realizando a mesma instrução sobre dados distintos. Máquinas SIMD em geral possuem vários processadores supervisionados pela mesma unidade de controle. Todos os processadores recebem a mesma instrução, que é emitida pela unidade de controle, e a executa sobre dados distintos. Há também uma rede de interconexão que possibilita a comunicação entre processadores ou entre processadores e memórias. Algumas arquiteturas SIMD são:

- CM-200 da Thinking Machines Corporation [RO93]
- MP-1 da MasPar Computer Corporation [RO93]
- DAP (*Distributed Array Processor*) [HJ88]

Figura 1.2: SIMD

MISD (Multiple Instruction Single Data) Essa categoria considera que múltiplos processadores aplicam diferentes instruções sobre um único dado e seus resultados. Alguns especialistas consideram os processadores vetoriais *pipelined* como integrantes dessa categoria e não como parte da categoria SIMD [AG89].

Figura 1.3: MISD

MIMD (Multiple Instruction Multiple Data) Nessa categoria múltiplos processadores supervisionados por suas correspondentes unidades de controle operam sobre dados distintos simultaneamente. A maioria dos sistemas multiprocessadores e multicomputadores podem ser classificados nessa categoria.

Exemplos de arquiteturas MIMD:

- KSR1 da Kendell Square Research [RO93]

- CM-5 da Thinking Machines Corporation [HT93]
- Intel Paragon [CPHL94]

Figura 1.4: MIMD

A classificação de Flynn é bastante ampla e caracteriza as arquiteturas paralelas apenas nas categorias SIMD e MIMD, não apresentando maiores distinções entre as máquinas qualificadas no mesmo grupo. É uma taxonomia que apesar de ser bastante genérica é também muito simples e conhecida. Para efeitos de estudos e comparações este trabalho levará em consideração a taxonomia de Flynn.

1.1.2 Treleaven

Treleaven subdivide as arquiteturas MIMD de Flynn de acordo com os mecanismos de dados e de controle utilizados [AG89]. O mecanismo de dados foi dividido em dois tipos: o de memória compartilhada e o de passagem de mensagens. O mecanismo de memória compartilhada está mais próximo ao modelo de von Neumann que o de passagem de mensagens.

Em relação ao mecanismo de controle Treleaven considera desde arquiteturas com controle forte como por exemplo os multiprocessadores convencionais até aquelas com controle fraco como as arquiteturas de fluxo de dados. Nas máquinas com controle forte há um controle explícito da execução das instruções, enquanto que, no outro extremo encontram-se as máquinas cuja execução é dirigida pelos dados sem um mecanismo que seqüencie a execução.

As máquinas como as convencionais de von Neumann são dirigidas pelo controle, por isso, têm o mecanismo de controle denominado de *control-driven*, já as máquinas de fluxo de dados são dirigidas pelos dados e seu mecanismo de controle é chamado de *data-driven*. Treleaven ainda acrescenta entre o *control-driven* e o *data-driven* os controles *demand-driven* (tipo de controle das máquinas de redução) e o *pattern-driven* (lógica). A tabela a seguir resume a classificação de Treleaven.

Tabela 1.1: Classificação de Treleaven

Mecanismo de Controle	Mecanismo de Dados	
	Memória Compartilhada	Memória Privada
control driven	von Neumann	comunicação de processos
pattern driven	logic	actors
demand driven	graph reduction	string reduction
data driven	dataflow I-structure	dataflow tokens

1.1.3 Feng

A classificação de Tsé-yun Feng é orientada a performance, baseada no grau de paralelismo da máquina [HB84, Ski88]. De acordo com Feng, o paralelismo de uma máquina é descrito em função do número de *bits* que podem ser processados simultaneamente e é dado em termos de comprimento de palavra (*n*) e comprimento de *bit-slice*¹ (*m*). Esses comprimentos correspondem ao número de *bits* contidos numa palavra ou num *bit-slice*.

A partir do par (*n,m*) podemos obter o grau máximo de paralelismo (*P*) de uma máquina que é dado por: $P = n * m$. Como exemplo temos o TI-ASC¹, que é um sistema com palavra de comprimento 64 e 4 *pipelines* aritméticos, cada um com 8 estágios. Daí, temos $8 * 4 = 32$ *bits* por cada *bit-slice* nos quatro *pipes*, e podemos representar a máquina pelo par (64,32). Feng descreve quatro categorias de arquiteturas:

WSBS (Word-Serial Bit-Serial) Nessa categoria apenas um *bit* é processado por vez (*n=m=1*) e ela também é chamada de processamento *bit-serial*. Como exemplo

¹Um *bit-slice* é um conjunto de *bits*, sendo cada um correspondente à mesma posição vertical de cada uma das palavras

dessa categoria temos o sistema MINIMA (1,1) [HB84].

WPBS (Word-Parallel Bit-Serial) Essa categoria processa m *bit-slices* ($n=1, m>1$) por vez, também é chamada de processamento *bis* (*bit-slice*). Alguns exemplos de arquiteturas que se enquadram nessa categoria são:

- STARAN (Goodyear Aerospace) (1,256) [HB84]
- MPP (Goodyear Aerospace) (1,16384) [HB84]
- DAP (ICL,England) (1,4096) [HJ88]

WSBP (Word-Serial Bit-Parallel) Essa categoria processa uma palavra de n *bits* por vez, também é chamada de processamento *word-slice* ($n>1, m=1$) e corresponde a maioria dos computadores existentes. Exemplos de arquiteturas que pertencem a essa categoria são:

- CDC 6600 (60,1) [HB84]
- VAX 11/780 (16/32,1) [Yue88]
- Burrough 7700 (48,1) [HB84]

WPBP (Word-Parallel Bit-Parallel) Nessa categoria um arranjo de $n * m$ *bits* é processado por vez ($n>1, m>1$), é também chamada de processamento paralelo e é a mais rápida das categorias de Feng. Exemplos representativos dessa categoria são:

- ILLIAC IV (64,64) [IT89]
- TI-ASC (64,32) [HB84]
- C.mmp (16,16) [IT89]

Apesar de permitir comparações em termos de performance entre as diversas arquiteturas, a classificação de Feng não delimita com clareza as distinções entre as arquiteturas.

1.1.4 Händler

A classificação de Wolfgang Händler [Qui87, HB84] utiliza uma notação com três pares de inteiros para descrever o grau de paralelismo e *pipelining* que ocorrem em 3 níveis de um sistema:

- **PCU** *Processor Control Unit*
- **ALU** *Arithmetic Logic Unit*
- **BCL** *Bit-Level Circuit*

PCU corresponde a um processador ou unidade central de processamento, a ALU corresponde a uma unidade funcional ou a um elemento de processamento (EP) de um arranjo de processadores e o BCL corresponde à lógica necessária para realizar operações de 1 bit na ALU. Portanto, de acordo com Händler um computador (C) pode ser descrito seguinte maneira:

$$T(C) = \langle K \times K', D \times D', W \times W' \rangle$$

onde [HB84]:

K é o número de processadores (PCUs) dentro do computador

K' é o número de PCUs que podem ser *pipelined*

D é o número de ALUs ou (EPs) sob o controle de um PCU

D' é o número de ALUs que podem ser *pipelined*

W é o comprimento de palavra de uma ALU ou de um EP

W' é o número de estágios de *pipeline* em todas ALUs ou em um EP

Quando a notação acima apresentar o segundo elemento K', D' ou W' igual a 1, podemos omiti-lo. Os operadores x e + podem ser utilizados com as seguintes finalidades: unir descrições de diferentes tipos de processadores em um único sistema (computador) e separar os diferentes modos de operação do sistema (os casos em que o sistema pode ser usado de diversas maneiras), respectivamente. Alguns exemplos dessa taxonomia são:

- TI-ASC (*Texas Instruments Advanced Scientific Computer*) tem um controlador que controla 4 *pipelines* aritméticos cada um com comprimento de palavra de 64 bits e 8 estágios. Portanto, $T(\text{ASC}) = \langle 1 \times 1, 4 \times 1, 64 \times 8 \rangle = \langle 1, 4, 64 \times 8 \rangle$.
- CDC 6600 tem uma única unidade central de processamento (CPU). Uma ALU com 10 unidades funcionais ligadas em *pipeline*. O comprimento de palavra é de 60 bits. Além disso, o CDC 6600 tem 10 processadores periféricos de Entrada/Saída (E/S) que podem operar em paralelo. Cada processador de E/S contém uma ALU com comprimento de palavra de 12 bits. Portanto, $T(\text{CDC 6600}) = T(\text{processador central}) \times T(\text{processador E/S})$, ou seja, $T(\text{CDC 6600}) = \langle 1, 1 \times 10, 60 \rangle \times \langle 10, 1, 12 \rangle$.
- C.mmp pode ser usado em diversos modos. Ele consiste de 16 minicomputadores PDP-11 de comprimento de palavra de 16 bits. Normalmente, ele opera no modo MIMD, mas teoricamente pode operar no modo SIMD e também no modo MISD [HB84]. Portanto, temos a seguinte classificação dividida em 3 partes:

$$T(\text{C.mmp}) = \langle 16, 1, 16 \rangle + \langle 1 \times 16, 1, 16 \rangle + \langle 1, 16, 16 \rangle$$

A classificação de Händler descreve bem máquinas convencionais vetorizadas, mas não é muito clara em relação a arquiteturas multiprocessadoras mais recentes [Ski88].

1.1.5 Shore

A taxonomia de Shore é baseada na organização das estruturas do computador. Sua classificação identifica seis tipos diferentes de máquinas [HJ88] :

Máquina I Essa categoria corresponde a arquitetura convencional de von Neumann com uma única Unidade de Controle (UC), Unidade de Processamento (UP), Memória de Instruções (MI) e Memória de Dados (MD). Com uma única leitura da MD são fornecidos todos os *bits* de uma palavra para processamento em paralelo na UP. Podem existir múltiplas unidades funcionais na UP que podem estar em *pipeline*, daí a categoria englobar computadores *pipelined* escalares e computadores *pipelined* vetoriais. Exemplos:

- CDC 7600 - escalar [HJ88]

- CRAY-1 - vetorial [IT89, Qui87]

Figura 1.5: Máquina I

Máquina II É semelhante à máquina I, mas apresenta diferenças em relação à leitura da MD. Na máquina I uma leitura da MD produz todos os *bits* de uma palavra, enquanto que, na máquina II ocorre a busca de um *bit-slice* de todas as palavras na memória.

A UP é organizada de modo a efetuar suas ações no modo *bit-serial*. Se considerarmos a memória como sendo um arranjo bi-dimensional de *bits* onde uma palavra é armazenada em cada linha, a máquina I realiza uma leitura de *slice* horizontal, ao passo que, a máquina II lê um *slice* vertical. Exemplos:

- ICL DAP [HJ88]
- STARAN [HB84]

Máquina III Essa classe de máquinas é uma combinação das máquinas I e II, e apresenta uma memória bi-dimensional que possibilita a leitura tanto de palavras como *bit-slices*, e duas UPs sendo uma horizontal para processar palavras e outra vertical para tratar *bit-slices*. Exemplo:

- OMEN-60 da Sanders Associates [HB84]

Figura 1.6: Máquina II

Figura 1.7: Máquina III

Máquina IV Essa máquina possui uma única UC que emite instruções a um conjunto de Elementos de Processamento (PEs). Os PEs são obtidos a partir das UPs e MDs da máquina I, e somente se comunicam através da UC e não diretamente entre si. Exemplo:

- PEPE [HB84, IT89]

Figura 1.8: Máquina IV

Máquina V Essa máquina é bastante semelhante à máquina IV, mas seus PEs possuem conexões aos vizinhos mais próximos podendo assim endereçar palavras tanto em sua memória como na de seus vizinhos imediatos. Exemplo:

- ILLIAC IV [IT89, HJ88]

Máquina VI As máquinas das classes I a V apresentam as MDs separadas das UPs, enquanto a máquina VI também denominada LIMA (*Logic-in-Memory Array*) distribui a lógica do processador por toda a memória. Exemplos:

- Variam de simples memórias associativas a complexos processadores associativos [HJ88]

Figura 1.9: Máquina V

Figura 1.10: Máquina VI

Figura 1.9: Máquina V

Figura 1.10: Máquina VI

Na classificação de Shore as máquinas das classes II a V são subdivisões da classe SIMD de Flynn e a máquina I corresponde à classe SISD. Essa taxonomia também não apresenta distinções claras para os computadores vetoriais *pipelined*, os quais são agrupados juntamente com computadores escalares não *pipelined*.

1.2 Arquiteturas Dataflow

A pesquisa em fluxo de dados originou-se na década de 1960 e é atribuída ao Prof. Jack B. Dennis do MIT [HPV83]. Projetos pioneiros surgiram em centros de pesquisa como MIT e Universidade de Stanford nos Estados Unidos, CERT-ONERA na França, Universidade de Manchester na Inglaterra, NTT e ETL no Japão entre outros. De acordo com a taxonomia proposta por Flynn a arquitetura baseada no paradigma de fluxo de dados se enquadra na categoria MIMD.

O princípio fundamental dessa arquitetura consiste em que uma instrução pode ser executada a partir do instante em que seus operandos estejam disponíveis, não existindo controle centralizado da execução do programa. O programa é representado por um grafo que indica as dependências de dados entre as instruções. No grafo cada instrução é representada por um nó e os arcos que ligam os nós representam os caminhos que os dados percorrem até a instrução subsequente. Quando os dados de um nó (instrução) estão disponíveis o nó é então habilitado e pode ser executado, essa é a regra de disparo.

A figura 1.11 ilustra um grafo para o trecho de programa $a = (b + 1) * (b - c)$. No grafo acima percebe-se a possibilidade de execução concorrente das instruções de soma e subtração pois não há dependência de dados nesse caso. Dessa forma a arquitetura de fluxo de dados expõe o paralelismo dos algoritmos automaticamente, sem obrigar o programador a detectá-lo e explorá-lo. Os dados são conduzidos por fichas (pacotes) que se movem através dos arcos entre os nós. Como os nós podem possuir mais de uma entrada, sua execução só é disparada após a chegada de todas as fichas nas correspondentes entradas. No exemplo da figura 1.11, o nó que executa a operação de subtração é um nó de duas entradas, e somente é executado a partir da chegada das fichas correspondentes aos dados b e c em suas entradas. Supondo que apenas a ficha do dado b esteja presente

Figura 1.11: Grafo para $a = (b + 1) * (b - c)$

na sua respectiva entrada, então, essa ficha deve esperar pelo seu par correspondente, a ficha relativa ao dado c , que é denominada ficha parceira. A partir da chegada da ficha parceira, as fichas são emparelhadas e o nó pode ser executado.

Dependendo do modo como as fichas são tratadas, os computadores de fluxo de dados podem ser divididos em: modelo estático e modelo dinâmico [GB90].

O grupo do Prof. Jack B. Dennis no MIT pioneiro na pesquisa em fluxo de dados desenvolveu um modelo de computação estático. No modelo estático apenas uma ficha por vez pode ocupar determinado arco. Isso gera uma regra de disparo estática em que uma instrução é habilitada se uma ficha de dado está presente em cada um de seus arcos de entrada e nenhuma ficha está presente em qualquer um de seus arcos de saída. Portanto, o grafo contém fichas de controle além das fichas de dado, ambas contribuindo para habilitar as instruções. Essas fichas de controle agem com sinais de *acknowledge* (reconhecimento) quando as fichas de dados são removidas dos arcos de saída.

Outro grupo, o do Prof. Arvind também do MIT desenvolveu um modelo dinâmico. Em máquinas dinâmicas as fichas são rotuladas para permitir que múltiplas fichas apa-

reçam simultaneamente em qualquer arco de entrada de um nó. O rótulo essencialmente identifica o contexto em que a ficha foi gerada, fichas pertencentes a diferentes instâncias são distingüidas através desse mecanismo que é denominado de *cor*. Através do mecanismo de *cor* é possível tratar eficientemente código reentrante, possibilitando maior exploração de paralelismo. Não são necessárias fichas de controle para sinalizar (reconhecer) a transferência de fichas de dados entre as instruções. Entretanto, o emparelhamento (*matching*) de fichas destinadas a instruções que requerem mais de um operando se torna mais complexo devido a existência desse rótulo. Para que as fichas destinadas à mesma instrução sejam emparelhadas, elas devem possuir a mesma *cor*, ou seja, pertencerem à mesma instância. Alguns modelos de arquiteturas de fluxo de dados são:

1.2.1 MTTDA

A MTTDA (*MIT Tagged-Token Dataflow Architecture*) é uma arquitetura proposta pelo grupo do Prof. Arvind do MIT. Essa máquina possui 64 elementos de processamento (PE) idênticos e módulos de memória chamados *I-structures* interconectados por uma rede n-cúbica de chaveamento por pacotes [AN90]. Cada PE é um processador dataflow e cada unidade *I-structure* possui um controlador cuja função é coordenar leituras e escritas na mesma. Um único PE e uma única *I-structure* formam um computador dataflow completo.

Os módulos (nós) *I-structures* usam conceito de rotulação (*tagging*) a fim de possibilitar que mais de uma instanciação de uma rotina compartilhem o mesmo grafo. Esses nós também suportam leitura antes da escrita.

As leituras e escritas seguem o seguinte critério. Existe uma área de armazenamento de dados onde cada posição possui bits extras que especificam um entre seus três estados: *present*, *absent* e *waiting*. Ocorre uma inicialização de todas as posições da *I-structure* para *absent* quando ela é alocada. Para efetuar uma leitura, a ficha de leitura deve carregar o endereço da posição a ser lida e o rótulo para a instrução que está esperando pelo valor. Caso o estado da posição a ser acessada seja *present*, então o dado é lido e enviado à instrução. No entanto, se o estado for *absent* ou *waiting* a leitura é indeferida e o rótulo é armazenado na posição designada.

A escrita é realizada a partir de uma ficha de escrita que deve conter o endereço da

Figura 1.12: MTTDA (*MIT Tagged-Token Dataflow Architecture*)

posição onde será efetuada a operação de escrita e o dado correspondente à posição. Se o estado da posição for *absent* o valor é escrito e o estado é alterado para *present*. Mas, se o estado for *waiting*, então o valor é escrito na posição endereçada, o estado é modificado para *present* e o valor é emitido também para todos os endereços (destinos) armazenados naquela posição. Se o estado já for *present*, então ocorreu um erro, pois já existe um valor na posição. Para a programação dessa máquina utiliza-se a linguagem denominada *Id* que foi desenvolvida também no MIT e possui propriedades adequadas à arquitetura da máquina.

1.2.2 Monsoon

A arquitetura da máquina Monsoon foi proposta por Papadopoulos e Culler [Pap88]. Essa arquitetura constitui uma evolução da arquitetura MTTDA. Em vista da dificuldade de implementação da função de emparelhamento de fichas (*tokens*) na MTTDA, que requer a verificação por parte da ficha que está em busca da parceira de todas as fichas que

estão armazenadas esperando pelo emparelhamento, Papadopoulos e Culler propuseram o conceito de memória de fichas explícita (ETS).

A idéia do conceito ETS vêm da maneira como os grafos são mapeados nas máquinas dataflow multiprocessadoras. Nessas máquinas os rótulos incluem um número para o PE, isto é, apresentam um campo de PE e há uma seção separada de *waiting-matching* para cada processador.

Figura 1.13: Arquitetura Monsoon

Os rótulos de entrada correspondentes à mesma ativação de uma instrução devem ser idênticos. Como a parceira de uma ficha tem o rótulo idêntico conseqüentemente deve ter também o campo de PE idêntico. As fichas com o mesmo campo de PE são levadas à seção *waiting-matching* do PE correspondente. Dessa forma, a verificação da ficha que busca a parceira se restringe apenas às fichas que se encontram no mesmo PE. Assim, o

rótulo identifica explicitamente um processador.

O conceito ETS ameniza algumas dificuldades encontradas na MTTDA como por exemplo o gerenciamento de memória associativa, permitindo que a *waiting-matching store* seja uma memória rápida e diretamente endereçável.

1.2.3 VIM

Outro grupo do MIT sob o comando do Prof. Jack B. Dennis realiza pesquisa baseada no modelo de fluxo de dados estático. O projeto se destina à construção de uma máquina voltada para computação numérica e com performance de 1 GFLOP.

A intenção do grupo é mostrar a viabilidade do dataflow estático para computação numérica intensa. Essa arquitetura é denominada VIM e consiste de rede de roteamento, blocos de celas (CB), unidades funcionais (FU) e memórias de matrizes (AM) [HY+88]. A estrutura da máquina é mostrada a seguir.

Figura 1.14: Arquitetura VIM

Os grafos são armazenados nos blocos de celas (CB), os quais também armazenam

operações e operandos, endereços de destino dos nós além de reconhecerem as instruções que estão habilitadas para a execução. As unidades funcionais (FU) são responsáveis pelo processamento dos dados e as memórias de matrizes (AM) armazenam estruturas de dados. A máquina utiliza a linguagem de programação VIM-VAL.

1.2.4 Epsilon-1

Primeiro processador dataflow do Sandia National Laboratories o Epsilon-1 apresentava performance comparável à dos mini-supercomputadores contemporâneos [GDHH89]. A arquitetura do Epsilon-1 executa emparelhamento direto e portanto, elimina a necessidade de *matching stores* associativas. O protótipo do Epsilon-1 foi projetado levando-se em consideração escalabilidade e simplicidade de *design*.

A filosofia do projeto se ateve a alguns princípios RISC tais como hardware de controle simples, execução de instruções em um único ciclo (quando possível) e modos de combinação de instruções simples em instruções mais complexas.

1.2.5 Epsilon-2

A arquitetura do processador Epsilon-2 se baseia nos conceitos do modelo Epsilon-1 preservando as características fundamentais do modelo antecessor tais como emparelhamento direto, processamento *pipeline* e características RISC [GH90].

Porém, a arquitetura do processador Epsilon-2 abandona o modelo dataflow estático em favor do modelo dinâmico para dar suporte a recursividade e *looping*.

Essa máquina implementa um modelo computacional híbrido o qual combina o paralelismo de granularidade fina do dataflow com característica seqüencial de von Neumann. Os PEs do Epsilon-2 são baseados nos modelos do Epsilon-1 e do Monsoon. Epsilon-2 pode executar código compilado a partir das linguagens Id, SISAL e Fortran.

Figura 1.15: Arquitetura Epsilon-2

1.2.6 SIGMA-1

O projeto SIGMA-1 foi iniciado em 1982 no Electrotechnical Laboratory (ETL) no Japão pelo grupo de Toshitsugu Yuba [HNS+87]. O principal objetivo desse projeto era identificar e estabelecer a base tecnológica para supercomputadores dataflow altamente paralelos. Um sistema SIGMA-1 com 128 elementos de processamento síncronos tem estado operacional desde janeiro de 1988 [YSY+90].

Figura 1.16: Arquitetura SIGMA-1

Sua velocidade máxima é de 423 MFLOPS e a velocidade efetiva medida para um programa de integração numérica foi de 170 MFLOPS em 1988. Além dos 128 PEs o SIGMA-1 consiste também de 128 elementos de estrutura (SE), 32 redes de comunicação locais e uma rede global de comunicação conforme mostrado na figura 1.16. Os elementos de processamento e os elementos de estrutura são divididos em 32 grupos conectados via uma rede hierárquica de dois níveis. Cada um desses 32 grupos consiste de 4 PEs e 4 SEs conectados através de uma chave *crossbar* 10x10. Todos os grupos são conectados por meio da rede global de comunicação. A máquina SIGMA-1 usa a linguagem de programação DFC-2 (Dataflow C).

1.2.7 EM-4

Em 1986 o ETL iniciou o projeto do EM-4 que em contraste com a máquina SIGMA-1 constitui uma arquitetura dirigida para processamento simbólico. O elemento de processamento do EM-4 consiste de um processador denominado EMC-R que é integrado num único CI-RISC e um módulo de memória. Um protótipo do sistema EM-4 com 80 EMC-Rs está operacional desde abril de 1990 [SKY93].

Os elementos de processamento estão divididos em 16 grupos, cada um com 5 EMC-Rs. O pico de performance desse protótipo é de 1 GIPS. A estrutura que conecta esses elementos de processamento é uma rede de interconexão multi-estágio omega consistindo de unidades de chaveamento com controle independente (controle descentralizado).

Figura 1.17: Arquitetura EM-4

1.2.8 MMDM

O grupo de John R. Gurd e Ian Watson da Universidade de Manchester iniciou em 1975 o projeto de uma máquina a fluxo de dados dinâmica rotulada por fichas (*tagged-token*) denominada MMDM (*Manchester Multi-Ring Dataflow Machine*). A MMDM possui uma estrutura em anel onde estão interligadas uma chave de entrada e saída e quatro unidades (*Token Queue*, *Matching Unit*, *Node Store* e *Functional Unit*) formando um *pipeline* de quatro estágios [GKW85, HY+88].

A chave (*Switch*) faz a conexão do hospedeiro com o anel coordenando a entrada e a saída de fichas no anel. A unidade que faz o armazenamento de fichas com a finalidade de controlar o tráfego delas no anel é denominada de *Token Queue* e usa o conceito FIFO (*First In First Out*). A unidade de emparelhamento (*Matching Unit*) efetua a regra de disparo da máquina verificando o destino e o rótulo das fichas a serem emparelhadas.

A unidade de instrução (*Node Store*) armazena o código do programa em execução. É nessa unidade que as fichas emparelhadas e as fichas destinadas a instruções de uma só entrada vão buscar a instrução à qual estão endereçadas. Da *Node Store* as fichas e seus correspondentes dados são enviados para a unidade de processamento (*Functional Unit*) para serem executados.

A MMDM utiliza a linguagem de programação funcional SISAL (*Streams and Iteration in Single Assignment Language*) que é resultante do esforço conjunto entre Universidade de Manchester, Universidade do Colorado, Lawrence Livermore National Laboratory e DEC (Digital Equipment Corporation) [Mac85].

1.3 Conclusões

Em geral, as classificações de arquiteturas de computadores são um tanto quanto pobres, muitas vezes não cobrem em detalhes as diferenças fundamentais entre as diversas arquiteturas enquadradas dentro de uma mesma categoria. Além disso, elas são relativamente antigas, portanto, não consideram características híbridas apresentadas por um grande número de arquiteturas paralelas modernas. Daí, a dificuldade em enquadrar com

Figura 1.18: Arquitetura MMDM

precisão tais arquiteturas no perfil de alguma das taxonomias. Apesar disso, as taxonomias são importantes por diversas razões como:

- Permitem entender os trabalhos já realizados. Devido ao crescimento de sistemas com diferentes tipos de paralelismo, ainda não está completamente claro quais arquiteturas apresentam as melhores perspectivas para o futuro.
- Revelam possíveis configurações que ainda não tenham sido pensadas. Como os sistemas existentes já estão classificados, as lacunas nas classificações podem sugerir outras possibilidades.
- Permitem a elaboração de modelos para comparações de performance.

As arquiteturas a fluxo de dados surgiram como uma alternativa às limitações apresentadas pelas máquinas convencionais baseadas no modelo de von Neumann. As máquinas baseadas no paradigma de fluxo de dados tratam alguns problemas típicos de arquiteturas paralelas como alocação de processos e distribuição de carga autônomos, linearidade de velocidade de processamento quanto ao número de processadores, contenção de memória e latência de acesso de maneira elegante e eficiente [GNC'92]. Além disso, arquiteturas dataflow expõem o paralelismo intrínseco dos algoritmos automaticamente, sem obrigar o programador a detectá-lo e explorá-lo, e são mais fáceis de programar que outras arquiteturas MIMD.

Capítulo 2

Redes de Interconexão

2.1 Introdução

É crescente o número de aplicações que exigem maior velocidade e potência computacional. Aplicações das mais diversas áreas como previsões meteorológicas, modelagem mecânica, estudo de fluido dinâmico, processamento de imagens, visão computacional, simulação de circuitos eletrônicos, estudos sísmicos, estudo de aerodinâmica de aviões e mísseis, inteligência artificial, física molecular entre outras apresentam requisitos computacionais que consomem facilmente os recursos computacionais proporcionados pelas máquinas convencionais (modelo de von Neumann).

Daí, a motivação e necessidade de desenvolvimento de pesquisas em busca de soluções capazes de tratar tais aplicações, as quais apresentam dois fatores básicos dominantes que são: grande volume de informações a serem processadas e resposta em tempo real. Soluções que satisfaçam os requisitos citados acima abrem campo para um desafio que é o de gerar sistemas computacionais de alta velocidade e performance a custos reduzidos [WHJ90]. Uma das maneiras de se tentar vencer esse desafio é realizar atualizações nas arquiteturas existentes através do uso de tecnologias mais rápidas, o que tem proporcionado uma boa relação custo/desempenho devido principalmente ao rápido crescimento da área de dispositivos lógicos. Mas, devido a fatores físicos limitantes como por exemplo a velocidade da luz, que impulsionam rapidamente a indústria de dispositivos semicondutores para seu limite de tecnologia de fabricação, há necessidade de se buscar meios alternativos

para superar o desafio de melhorar a relação custo/desempenho sem depender unicamente de avanços da tecnologia de dispositivos lógicos [PB92].

Uma outra maneira de endereçar aquele desafio é através de mudanças mais profundas na arquitetura. Uma tentativa de solução que tem sido bastante pesquisada baseia-se em arquiteturas paralelas, principalmente na categoria MIMD [Rya91]. A idéia é bastante simples, deseja-se obter ganho de velocidade e potência computacional através do paralelismo. Conforme [HJA90] um sistema de processamento paralelo pode ser definido como um conjunto de recursos computacionais (CPUs, memórias, sistema operacional, etc) que exploram a concorrência de um processo de computação. Para efetivamente se explorar o paralelismo o trabalho a ser computado pelo sistema deve ser dividido em tarefas menores a fim de possibilitar a execução concorrente. Essas tarefas precisam ser coordenadas através de algum meio de comunicação. As redes de interconexão são mecanismos de comunicação que possibilitam coordenar a execução concorrente dessas tarefas. Portanto, seu papel é de extrema relevância no projeto de arquiteturas paralelas. Esse capítulo discute as características fundamentais das redes de interconexão tais como estruturas de interligação, elementos de chaveamento, parâmetros de projeto, permutações e classificação.

2.2 Comunicação em Arquiteturas Paralelas

De acordo com Duncan [Dun90] a taxonomia originalmente proposta por Flynn pode ainda ser subdividida de forma a considerar principalmente os modelos baseados na categoria MIMD. Duncan faz subdivisões que definem com maior precisão as arquiteturas de memória compartilhada, memória distribuída e arquiteturas baseadas no paradigma de fluxo de dados (dataflow). A partir dessas subdivisões da categoria MIMD pode-se destacar as formas de comunicação dos sistemas fortemente acoplados (ou sistemas de memória compartilhada) e dos sistemas fracamente acoplados (ou sistemas de memória distribuída). Os primeiros também são denominados de sistemas multiprocessadores e os últimos de sistemas multicomputadores. Conforme mostrado na figura 2.1, nos sistemas multiprocessadores todos os elementos de processamento acessam uma memória global (a

memória compartilhada) [Bhu87].

Figura 2.1: Sistema Multiprocessador

É através dessa memória compartilhada que são realizadas a comunicação e sincronização dos elementos de processamento.

Já nos sistemas multicomputadores cada elemento de processamento possui uma memória local que não pode ser acessada diretamente pelos outros processadores.

Figura 2.2: Sistema Multicomputador

E a comunicação e sincronização são efetuadas por meio de uma rede que interliga processadores a processadores. Ambos os sistemas multiprocessadores e multicomputadores necessitam de um meio de interligação que possibilite a comunicação ou entre processadores e memórias ou entre processadores.

De acordo com [Bhu87] uma rede de interconexão é uma conexão complexa de chaves e *links* que permite a comunicação de dados entre processadores e memórias em um sistema multiprocessador ou entre os processadores em um sistema multicomputador. A escolha entre qual dos dois tipos de sistemas a ser utilizado numa determinada arquitetura, em geral, depende da aplicação endereçada, sendo que os sistemas multiprocessadores são mais adequados às aplicações que exigem muita comunicação entre os elementos de processamento devido a memória ser global o que facilita a troca de dados; e os sistemas multicomputadores apresentam a propriedade de maior expansibilidade por possuírem memória distribuída o que facilita o acréscimo de elementos de processamento. As arquiteturas de memória compartilhada em geral utilizam como meio de interconexão barramentos compartilhados, chaves *crossbar* e redes de interconexão multi-estágio. Enquanto que as arquiteturas de memória distribuída fazem uso de diferentes tipos de estruturas de interconexão tais como: anel, malha, árvore, hipercubo e outras. A escolha do tipo de interconexão tem papel importante no desempenho global desses sistemas e leva em consideração fatores como a relação custo/desempenho, tipo de aplicação, modelo da arquitetura entre outros.

2.3 Topologias

Os módulos que constituem um sistema de processamento paralelo podem ser interligados de diversas maneiras. A estrutura de interligação varia conforme a topologia escolhida para a rede de interconexão. A decisão da escolha da topologia tem papel fundamental sobre os custos de comunicação e a performance de um sistema. Como consequência de uma escolha pouco criteriosa o custo dos módulos do sistema pode ser superado pelo custo da rede de interconexão. Portanto, um dos requisitos básicos de projeto de redes de interconexão é que o custo do hardware da rede corresponda a uma pequena parcela do custo total do sistema.

Uma estrutura de interconexão ideal *conexão total* é mostrada na figura 2.3. Essa rede apresenta a melhor estrutura de intercomunicação, visto que, ela permite que um módulo se comunique com qualquer outro mesmo que estejam ocorrendo outras comunicações

simultâneas. Portanto, ela possibilita que se efetue o número máximo de transações (comunicações). Na estrutura de *conexão total* constituída de N módulos existe uma conexão direta entre cada um dos N módulos com os demais. Isso resulta em $N*(N-1)/2$ conexões, pois cada módulo possui ligações com os $(N-1)$ módulos restantes. Portanto, o custo de interconexão para N módulos cresce com o fator de N^2 . Esse custo quadrático se torna um fator agravante à medida que N cresce, pois o custo da rede extrapola a expectativa de ser apenas uma fração do custo total do projeto. Além dessa séria desvantagem apresentada por essa estrutura outro item coopera para tornar a rede de *conexão total* inviável para sistemas grandes, o número de conexões para cada módulo também cresce com o aumento do sistema o que torna cada módulo mais complexo.

Figura 2.3: Conexão Total

Relegando os inconvenientes de complexidade e custo apresentados pela rede de *conexão total* que tornam-na altamente proibitiva para sistemas com grande número de módulos (isto é, para N grande), ela ainda possui características bastante desejáveis para uma rede de interconexão [AG89]. Como essa estrutura interliga cada módulo do sistema a todos os outros restantes (conexão todos-a-todos) ela possibilita uma **Conectividade** máxima, ou seja, permite o máximo grau de conexão do sistema. A conectividade de um sistema de processamento paralelo corresponde ao número de vizinhos imediatos que cada módulo (também chamado de nó) de processamento possui. Portanto, para se aproximar da característica ideal a conectividade de uma rede de interconexão deve ser a maior possível.

É de se esperar também que a rede de *conexão total* realize o maior número de

transações (comunicações) possível, já que existe a possibilidade de todos os módulos se comunicarem simultaneamente. A taxa de transações efetuadas pela rede é denominada **Largura de Banda** ou **Throughput**. A largura de banda portanto, corresponde ao volume de tráfego de mensagens que a rede de interconexão pode transmitir, e conclui-se disto que ela deva ser a mais alta possível.

Cada transação gasta um certo tempo para ser realizada, este tempo é denominado de **Latência de Comunicação** e corresponde ao tempo de trânsito de uma mensagem até atingir o módulo de destino. A latência de comunicação tende a crescer com N , pois as distâncias físicas crescem conforme aumenta-se o número de módulos. A latência de comunicação deve ser mínima. A tabela a seguir resume as características ideais de uma rede de interconexão.

Tabela 2.1: Características Ideais

Característica	Comportamento
Largura de Banda	Máxima
Conectividade	Máxima
Latência	Mínima
Custo de Hardware	Mínimo

2.3.1 Elementos de Chaveamento

As transações realizadas pela rede de interconexão são feitas através de elementos que roteiam a informação por meio de *links* desde o nó fonte até o nó de destino. Os elementos que efetuam o roteamento podem ser constituídos desde simples dispositivos capazes de apenas identificar o endereço da mensagem recebida e transmiti-la para o próximo elemento até dispositivos mais complexos como processadores e memórias. Genericamente esses elementos de roteamento que formam a rede podem ser denominados de **Elementos de Chaveamento** e estão representados na figura 2.4.

(A) - ELEMENTO DE CHAVEAMENTO SIMPLES

(B) - ELEMENTO DE CHAVEAMENTO COMPLEXO

Figura 2.4: Elementos de Chaveamento

2.3.2 Categorias de Topologias

De uma maneira mais formal, uma rede de interconexão pode ser descrita por um grafo onde os nós representam os elementos de chaveamento e os arcos representam os *links* de comunicação [Fen81, HB84]. As topologias podem ser divididas em duas categorias: estáticas e dinâmicas. Como descrito em [IT89] as conexões de entrada e saída de uma rede podem ser ou físicas ou virtuais.

2.3.2.1 Topologias Estáticas

Quando os *links* interconectam exatamente dois nós e são dedicados, então as conexões representam uma realização física da função de roteamento da rede. E barramentos dedicados não podem ser reconfigurados de forma a efetuarem conexões a outros processadores diretamente. Esse tipo de estrutura é denominada de topologia estática. A figura 2.5 apresenta alguns exemplos de topologias estáticas.

Figura 2.5: Topologias Estáticas

2.3.2.2 Topologias Dinâmicas

Por outro lado, uma rede com topologia dinâmica permite que os *links* sejam re-configurados suportando então conexões virtuais, isto é, os nós entre *links* podem ser compartilhados por várias conexões de entrada e saída desde que apenas uma delas esteja ativa a cada vez. A figura 2.6 apresenta alguns exemplos de topologias dinâmicas.

Figura 2.6: Topologias Dinâmicas

2.3.3 Escolha de Topologia

Basicamente a escolha da topologia deve satisfazer os requisitos intrínsecos de comunicação da aplicação específica endereçada. Portanto, a escolha pode ser bastante satisfatória quando se conhece de antemão o padrão de comunicação exigido, mas nem sempre as necessidades de comunicação de um sistema são previsíveis.

Numa máquina de propósito geral é difícil prever as necessidades de comunicação devido à generalidade das aplicações endereçadas. Como em uma rede de topologia estática a conectividade entre os nós é definida pela presença de *links* físicos, então a escolha da topologia depende sensivelmente do padrão de comunicação esperado [IT89].

Enquanto isso, devido à característica de reconfigurabilidade as topologias dinâmicas são mais adequadas a uma classe mais ampla de problemas, pois suportam padrões de comunicação arbitrários, o que tornam-nas mais convenientes às arquiteturas de propósito geral. Nesse sentido, as redes com topologias estáticas são deficitárias devido à sua falta de flexibilidade em vista da necessidade de se antever os requisitos do padrão de comunicação para que sejam estabelecidas as ligações físicas.

2.4 Decisões de Projeto

Além da escolha da topologia outras três decisões de projeto de redes de interconexão devem ser tomadas. As decisões são feitas entre:

- Modos de Operação
- Estratégias de Controle
- Métodos de Chaveamento

Essas decisões formam um conjunto denominado de características operacionais e são responsáveis pelos diferentes comportamentos do sistema.

Modos de Operação O modo de operação de uma rede trata da maneira como os dados são transferidos e existem dois tipos básicos: modo síncrono e modo assíncrono. O modo síncrono é caracterizado pela existência de um relógio (*clock*) global que emite sinais para todos os componentes do sistema a fim de sincronizá-los.

Esse método de operação em geral é empregado em estruturas de interconexão de sistemas SIMD pois tipicamente a movimentação dos dados ocorre de modo simultâneo. Entretanto, a estratégia de operação assíncrona trabalha sem a presença de um *clock* global. O modo assíncrono é normalmente empregado em estruturas

de interconexão de arquiteturas MIMD pois os PEs (elementos de processamento) em geral operam independentemente.

Estratégias de Controle A estratégia de controle corresponde à maneira como são gerados os sinais de controle que coordenam as funções de roteamento da rede. O controle pode ser centralizado ou distribuído (descentralizado). No tipo centralizado, os sinais de controle são gerados por uma única unidade de controle, o que pode causar um *engarrafamento* e degradar a performance do sistema além de ter efeito sobre a confiabilidade do sistema.

Por outro lado, os sinais de controle podem ser originados localmente aos componentes do sistema sem depender de um controlador global complexo. Esse esquema é denominado de controle distribuído pois um pequeno controlador mais simplificado é associado a cada elemento da estrutura de interconexão (elemento de chaveamento). Arquiteturas SIMD normalmente utilizam controle centralizado, ao passo que, arquiteturas MIMD geralmente empregam controle distribuído [IT89].

Métodos de Chaveamento O método de chaveamento diz respeito a utilização física das chaves na realização de uma função de roteamento. Há duas principais metodologias de chaveamento: chaveamento por circuito e chaveamento por pacotes. No chaveamento por circuito um caminho físico entre fonte e destino é estabelecido antes de se iniciar a comunicação, havendo portanto, um atraso (*delay*) decorrente do tempo de estabelecimento desse caminho. O caminho é mantido durante toda a transmissão. Esse tipo de chaveamento é mais adequado para mensagens longas e se ajusta bem a arquiteturas SIMD com controle centralizado. Apresenta *overhead* de controle relativamente baixo e requer elementos de chaveamento mais simples.

Já no método de chaveamento por pacotes a mensagem é dividida em pacotes pequenos para ser emitida por meio da rede através dos elementos de chaveamento que os recebem e os enviam a um novo elemento num estágio subsequente. Nesse caso, não há o atraso inicial existente no chaveamento por circuito para estabelecimento do caminho físico antes do início da transmissão, mas existe um atraso randômico que depende do tráfego presente na rede. A estratégia de chaveamento por pacotes é freqüentemente utilizada em sistemas MIMD e normalmente com

controle distribuído. E é mais eficiente quando se trabalha com mensagens curtas. Os elementos de chaveamento são relativamente mais complexos nesse caso. Há necessidade de mecanismos de tratamento de conflitos pois como os pacotes são auto-rodeantes existe possibilidade de ocorrência de colisão.

Tabela 2.2: Características Operacionais

Topologias	Estáticas
	Dinâmicas
Modos de Operação	Síncrono
	Assíncrono
Estratégias de Controle	Centralizado
	Distribuído
Métodos de Chaveamento	Circuito
	Pacote

2.5 Classificação das Redes de Interconexão

As redes de interconexão podem ser classificadas com base em suas características operacionais (modos de operação, estratégias de controle e métodos de chaveamento). Para uma determinada topologia escolhida existem oito categorias distintas para as redes de interconexão. A classificação é estruturada de acordo com os seguintes passos:

- 1. Divisão dos métodos de chaveamento:
 - a) Chaveamento por circuito (**C**)
 - b) Chaveamento por pacotes (**P**)
- 2. Divisão dos modos de operação:
 - a) Síncrono (**S**)
 - b) Assíncrono (**A**)

3. Divisão das estratégias de controle:

- a) Centralizado (**C**)
- b) Distribuído (**D**)

Portanto, as categorias resultantes são as seguintes:

- 1. **PSC**
- 2. **PSD**
- 3. **PAC**
- 4. **PAD**
- 5. **CSC**
- 6. **CSD**
- 7. **CAC**
- 8. **CAD**

A figura 2.7 resume o esquema de classificação das redes de interconexão.

2.6 Descrições

A seguir será feita a descrição de algumas estruturas de interconexão apresentadas anteriormente nas figuras 2.5 e 2.6. Considera-se N o número de nós da rede de interconexão.

2.6.1 Arranjo Linear

O arranjo linear é a mais simples das topologias estáticas apresentadas e pode ser considerado como uma malha uni-dimensional. É usado em algumas arquiteturas *pipeline*. Os nós interiores possuem duas conexões, enquanto que, os extremos têm apenas uma.

Figura 2.7: Classificação de Redes de Interconexão

2.6.2 Anel

Através da ligação dos nós das duas extremidades do arranjo linear obtém-se uma estrutura bi-dimensional denominada anel. Nessa topologia todos os nós têm grau 2, ou seja, cada nó tem duas conexões e o *Throughput*, o comprimento médio do caminho entre dois nós quaisquer e o custo são todos proporcionais a N [IT89]. Essa estrutura de interconexão é conveniente apenas para um pequeno número de nós e para algoritmos que não exijam muita comunicação.

2.6.3 Estrela

A estrela também é uma topologia bi-dimensional que apresenta um nó central responsável pela conexão de todos os outros nós. Essa configuração de interligação tem como fatores característicos o custo proporcional a N , a distância máxima entre nós que é 1, o número máximo de conexões paralelas que também é 1 e o grau de conexão do nó central que é $N - 1$. A topologia em estrela é inadequada para sistemas com grande número de

nós.

2.6.4 Malha

A rede com topologia em malha também é uma rede bi-dimensional bastante adequada a algoritmos que exploram comunicação altamente localizada entre nós. Essa estrutura é bastante conveniente para aplicações que envolvem algoritmos matriciais devido à correspondência topológica entre malhas e algoritmos orientados a matrizes. O número de transações possíveis em paralelo é proporcional a N e a distância máxima entre dois nós quaisquer é $\sqrt{N}$.

2.6.5 Árvore

As topologias estruturadas em árvore apresentam características que são favoráveis a aplicações que podem ser decompostas em hierarquias de atividades. As arquiteturas em árvore são convenientes para algoritmos de *searching* e *sorting*, algoritmos de processamento de imagens e paradigmas de programação dataflow [IT89, Dun90]. Uma das características fundamentais dessa topologia é o fato de que a distância entre dois nós quaisquer nunca é superior a $2(\log_2(N + 1) - 1)$.

2.6.6 n-Cubo

A topologia n-cubo binária também chamada de hipercubo é constituída de $N = 2^n$ nós e cada nó tem grau $n = \log_2 N$ resultando numa estrutura n-dimensional. Os nós são arranjados de forma a receberem uma identificação binária que varia de 0 a $N - 1$ e os nós adjacentes recebem valores que diferem em um único *bit*. A distância máxima entre dois nós corresponde ao diâmetro da topologia, ou seja, é $n = \log_2 N$. Um $(n + 1)$ -cubo binário é formado a partir de dois n-cubos e da conexão dos nós correspondentes. Conforme aumenta-se a dimensão de um n-cubo binário dobra-se o número de nós e acrescenta-se 1 ao número de conexões de cada nó. A topologia hipercubo tem sido

bastante pesquisada em vista da busca por arquiteturas que suportem os requisitos de aplicações tri-dimensionais [Dun90, Zha91, AP89, SG94, MB94].

2.6.7 Barramento Compartilhado

O barramento compartilhado constitui-se num meio *broadcast* que interconecta N módulos através de um único caminho de comunicação. Esse esquema de interconexão permite que cada módulo transmita dados para cada um dos $N-1$ módulos remanescentes.

Em face desse tipo de comunicação envolver normalmente um par de nós e do compartilhamento do meio de transmissão, há necessidade de um protocolo que administre a comunicação de modo que todos os módulos com exceção do receptor do dado ignorem os valores difundidos no barramento. O barramento é a estrutura de interconexão menos complexa e mais fácil de reconfigurar, mas também apresenta baixa performance. Ele pode realizar apenas uma transação por vez. Quando vários pares de módulos necessitam se comunicar simultaneamente deve haver um mecanismo de controle, isto é, um árbitro que resolva as disputas pelo barramento.

Por exemplo, num sistema de memória compartilhada, o barramento não permite que mais de um processador acesse a memória num dado instante. A consequência direta desse fato é que quando há um grande número de processadores decorre um longo tempo de espera pela posse do barramento. A latência de comunicação tende a aumentar proporcionalmente a N , visto que as distâncias físicas aumentam com o número de módulos tornando assim maior o tempo gasto por cada transação. Outro fator agravante decorre do fato de que a capacidade total de comunicação do barramento, isto é, o seu *Throughput* decresce conforme aumenta-se o número de módulos. Esses fatores tornam o barramento compartilhado um *gargalo* para sistemas com grande número de módulos.

2.6.8 Crossbar

Assim como o barramento compartilhado, a rede de interconexão *crossbar* é freqüentemente utilizada em sistemas de memória compartilhada. A rede de interconexão *crossbar* pode ser representada como um arranjo bi-dimensional de chaves, onde cada chave tem

a capacidade de interligar um módulo que se encontra conectado a um barramento horizontal a outro módulo que esteja conectado a um barramento vertical.

Esse tipo de estrutura se aproxima bastante do modelo ideal que é capaz de realizar conexão total entre todos os módulos simultaneamente. No entanto, uma rede *crossbar* suporta todas as possíveis conexões simultâneas entre os módulos desde que sejam distintas. Isso resulta em conflito quando dois ou mais módulos necessitam se comunicar com o mesmo módulo de destino. Como os módulos nessa topologia são conectados a barramentos horizontais e verticais, o comprimento de cada barramento geralmente cresce com o número de módulos, isto implica em maior tempo de comunicação resultando numa latência N . O *throughput* da rede é constante pois ela pode realizar N transações simultâneas.

Infelizmente o custo de uma rede *crossbar* que conecta N módulos do barramento horizontal a N módulos do barramento vertical é N^2 devido a necessidade de se utilizar N^2 chaves. Isso inviabiliza a escolha dessa topologia para sistemas com grande número de módulos. Apesar de que esse custo quadrático se restringe às chaves e fios, já que o custo por módulo para interfacear as linhas dos barramentos é constante [WHJ90].

2.6.9 Redes de Interconexão Multi-Estágio

Como visto anteriormente, uma topologia baseada no barramento compartilhado é bastante atrativa em termos de custo, mas deixa muito a desejar quanto ao desempenho que é fraco, pois não permite que mais de um par de módulos se comunique a cada vez. No entanto, apesar de apresentar características próximas daquelas da rede ideal, a rede *crossbar* possui custo muito elevado para grandes sistemas, o que a torna inviável em termos de escalabilidade. Essas duas topologias situam-se em extremos opostos do espectro de custo/desempenho. Entre esses dois extremos encontram-se as redes multi-estágio que apresentam um razoável equilíbrio entre custo e desempenho, e essa é a principal vantagem dessas redes. Elas consistem de dois ou mais estágios de elementos de chaveamento que definem caminhos de comunicação entre pares de módulos de entrada e saída.

A classe das redes multi-estágio apresenta um grande grupo de redes que são baseadas nas permutações *shuffle* e *exchange* (explicadas na seção 2.7). Uma rede multi-estágio

$N \times N$ conecta N módulos de entrada a outros N módulos de saída, como por exemplo, N processadores a N módulos de memória. Em geral, essas topologias possibilitam uma ampla variedade de mapeamentos simultâneos de módulos de entrada para módulos de saída utilizando $\log_2 N$ estágios, sendo cada estágio constituído de $N/2$ chaves *crossbar* 2×2 , considerando N uma potência de 2. Os elementos de chaveamento da rede são formados por essas chaves *crossbar* as quais possuem duas entradas e duas saídas. Por meio de um dígito de controle C é possível fazer a conexão de uma entrada a uma saída.

O algoritmo de roteamento baseia-se no *bit* de controle C fornecido pela entrada. Quando $C = 0$ a entrada é conectada à saída superior e quando $C = 1$ a entrada é conectada à saída inferior. Dessa maneira, o módulo (origem) que deseja efetuar comunicação com outro (destino) deve especificar o endereço de destino o qual contém um dígito de controle correspondente a cada estágio de chaves. Assim, a chave do i -ésimo estágio verifica o i -ésimo *bit* do endereço de destino contando a partir do dígito mais significativo para determinar se a entrada será conectada à saída superior ou inferior.

Figura 2.8: Controle do Elemento de Chaveamento

As redes multi-estágio são bastante apropriadas para sistemas multiprocessadores devido a essa característica de auto-roteamento, visto que não há necessidade de um controlador central. Uma rede multi-estágio pode ser bloqueante, não-bloqueante ou *rearranjável*. Uma rede é dita bloqueante se conexões simultâneas entre dois ou mais pares de módulos podem provocar disputas pelo uso dos *links* de comunicação da rede. Um exemplo de rede bloqueante é a rede Omega que não permite que determinadas combinações de conexões de entrada-saída sejam realizadas devido aos bloqueios entre os caminhos conflitantes. As redes não-bloqueantes são aquelas capazes de realizar todas as possíveis conexões sem bloqueio, ou seja, sem causar conflitos. Um exemplo de rede não-bloqueante

é a rede Clos. Já as redes *rearranjáveis* não-bloqueantes podem realizar todas as conexões possíveis entre entradas e saídas rearranjando as conexões existentes, de forma que sempre seja possível estabelecer uma conexão para um novo par de entrada-saída. Um exemplo de rede *rearranjável* não-bloqueante é a rede Benes. De acordo com [IT89], em princípio todas as redes multi-estágio com $\log N$ estágios são redes bloqueantes.

Alternativamente às redes multi-estágio, podem ser empregadas as redes de estágio único que realizam a mesma tarefa de uma rede multi-estágio com $\log_2 N$ estágios através de $\log_2 N$ passos, usando um único estágio que é equivalente a apenas um dos estágios de uma rede multi-estágio. Essa rede de estágio único evidentemente é $\log_2 N$ vezes mais barata e apesar de manter a mesma latência que a multi-estágio ela apresenta uma largura de banda $\log_2 N$ vezes menor porque todos os $\log_2 N$ passos devem ser completados antes de se iniciar nova comunicação. Enquanto que, uma rede multi-estágio pode efetuar nova comunicação a cada vez que a mensagem é transmitida para o próximo estágio. A maioria das redes multi-estágio são semelhantes, o que as diferencia é o padrão de interconexão entre seus estágios. Os padrões de comunicação utilizados nessas redes serão mais detalhados na seção 2.7.

2.7 Permutações

Uma rede de interconexão pode ser vista como uma estrutura que conecta seus módulos de entrada aos módulos de saída utilizando um conjunto de funções de roteamento ou funções de interconexão que atuam sobre os endereços dos módulos. Conforme descrito em [Sie85], cada função de roteamento é uma bijeção sobre o conjunto de endereços de entrada-saída.

Para um sistema com N módulos o conjunto de endereços E é constituído pelos inteiros de 0 a $N - 1$, ou seja, $E = \{0, 1, \dots, N - 1\}$. Uma função de roteamento f é uma função de endereços de módulos para endereços de módulos, isto é, $f : E \rightarrow E$, o que equivale a uma permutação, ou seja, uma bijeção de um conjunto nele mesmo. Assim, uma função de roteamento f permuta (mapeia) os endereços $0, 1, \dots, N - 1$ em $f(0), f(1), \dots, f(N - 1)$. A seguir serão descritas algumas das permutações mais empre-

gadas em redes de interconexão. Para efeito de definição das permutações considera-se x como sendo a representação binária do endereço de um módulo. Temos então que:

$$x = \{a_n, a_{n-1}, \dots, a_2, a_1\} = a_n * 2^{n-1} + a_{n-1} * 2^{n-2} + \dots + a_1$$

2.7.1 Perfect-Shuffle

Dividindo-se o conjunto ordenado de endereços dos módulos fonte em dois subconjuntos de mesmo tamanho e entrelaçando-os obtemos um mapeamento do conjunto de endereços dos módulos fonte para o conjunto de endereços dos módulos destino. Isso corresponde a uma permutação *perfect-shuffle* que pode ser descrita como uma rotação circular de uma posição à esquerda dos *bits* de endereço do módulo fonte. Assim, a permutação *perfect-shuffle* é definida como:

$$PS(x) = \{a_{n-1}, a_{n-2}, \dots, a_1, a_n\}$$

A figura 2.9 representa a permutação *perfect-shuffle* para $N = 8$ módulos.

Figura 2.9: Permutação *Perfect-Shuffle*

Uma outra forma de visualização que descreve com maior clareza o efeito de se aplicar a permutação *perfect-shuffle* mostrando a interligação entre os módulos é apresentada na figura 2.10. O inverso da permutação *shuffle* também é bastante utilizado, e pode ser definido como uma rotação circular de x à direita. Podemos obter ainda as permutações

super-shuffle (PS^i) e *sub-shuffle* (PS_i) a partir da rotação de uma determinada parte de x , respectivamente sobre os i bits mais significativos e sobre os i bits menos significativos de x .

Temos então:

$$PS^i(a_n, a_{n-1}, \dots, a_2, a_1) = \{a_{n-1}, \dots, a_{n-i+1}, a_n, a_{n-i}, \dots, a_1\}$$

e

$$PS_i(a_n, a_{n-1}, \dots, a_2, a_1) = \{a_n, \dots, a_{i+1}, a_{i-1}, \dots, a_1, a_i\}$$

Figura 2.10: Interligação dos módulos após aplicação da *Perfect-Shuffle*

2.7.2 Exchange

A permutação *exchange* interliga os módulos cujos endereços diferem apenas no i -ésimo bit. Ela pode ser definida como: $EX_i(x) = \{a_n, a_{n-1}, \dots, \bar{a}_i, \dots, a_1\}$

A figura 2.11 representa uma permutação *exchange* $EX_1(x)$ para $N = 8$ módulos.

Uma outra representação para a mesma permutação pode ser vista na figura 2.12.

2.7.3 Butterfly

A aplicação da permutação *butterfly* ocasiona a troca dos bits mais e menos significativos do endereço do módulo. A definição da permutação *butterfly* é dada a seguir:

Figura 2.11: Permutação *Exchange*

Figura 2.12: Interligação dos módulos após aplicação da *Exchange*

$$BF(x) = \{a_1, a_{n-1}, \dots, a_2, a_n\}$$

A permutação *butterfly* aplicada a $N = 8$ módulos pode ser vista na figura 2.13.

A representação do efeito da aplicação da permutação sobre os módulos é vista na figura 2.14.

Podemos também definir as permutações *super-butterfly* (BF^i) e *sub-butterfly* (BF_i). A permutação *super-butterfly* é dada por:

$$BF^i(a_n, a_{n-1}, \dots, a_2, a_1) = \{a_i, a_{n-1}, \dots, a_{i+1}, a_n, a_{i-1}, \dots, a_1\}$$

e a permutação *sub-butterfly* é dada por:

$$BF_i(a_n, a_{n-1}, \dots, a_2, a_1) = \{a_n, \dots, a_{i+1}, a_1, a_{i-1}, \dots, a_i\}$$

2.7.4 Shift

Na permutação *shift* há uma função aritmética que soma 1 ao valor numérico do endereço do módulo diferentemente das permutações anteriores em que uma função manipulava os *bits* para permutar o endereço. A permutação *shift* é definida como:

Figura 2.13: Permutação *Butterfly*

Figura 2.14: Interligação dos módulos após aplicação da *Butterfly*

$$SH(x) = |x + 1|_N$$

Uma representação da permutação *shift* para $N = 8$ módulos pode ser vista na figura 2.15. Outra forma de visualizar a permutação anterior é dada na figura 2.16.

Figura 2.15: Permutação *Shift*

Figura 2.16: Interligação dos módulos após aplicação da *Shift*

2.7.5 Identidade

A permutação identidade quando aplicada não causa nenhuma alteração nos endereços dos módulos, ela é usada apenas para definir permutações inversas.

Pode ser definida como:

$$I(x) = x$$

2.8 Aplicação de Permutações

Pode ser verificado a partir da observação das figuras representativas das permutações da seção 2.7 que a aplicação de apenas uma permutação não proporciona muita flexibilidade em termos de interconexões arbitrárias de entrada-saída. Mas, combinando-se algumas seqüências de permutações podemos obter resultados interessantes que possibilitam a construção de redes de interconexão multi-estágio.

Os elementos de chaveamento que formam uma rede de interconexão multi-estágio podem realizar permutação que pode ser definida da seguinte forma:

$$EC_i(x) = \{EX_i \mid I_i\}(x)$$

Através da combinação das permutações realizadas em cada estágio pelos elementos de chaveamento e pela seqüência intercalada de outras permutações podemos construir diversas redes multi-estágio tais como: Omega, Banyan, n-Cubo e outras. A seguir serão descritos alguns exemplos de construção dessas redes.

2.8.1 Omega

A rede Omega [Law75] com n estágios é definida como:

$$\text{Omega}_n = (PS_n EC_1)^n$$

A rede omega para $N = 8$ módulos é representada na figura 2.17.

2.8.2 Banyan

A rede Banyan [GL73] com n estágios é definida como:

$$\text{Banyan}_n = EC_1 BF_2 EC_1 BF_3 \dots BF_n EC_1$$

A rede Banyan para $N = 8$ módulos é representada na figura 2.18.

Figura 2.17: Rede Omega para $N=8$ módulos

Figura 2.18: Rede Banyan para $N=8$ módulos

2.8.3 n-Cubo

A rede Cubo Binário Indireto [PI77] com n estágios é definida como:

$$n - \text{Cubo} = EC_1 BF_2 EC_1 BF_3 \dots BF_n EC_1 PS_n^{-1}$$

A rede 3-Cubo para $N = 2^n = 8$ módulos é representada na figura 2.19.

Figura 2.19: Rede n-Cubo para N=8 módulos

2.9 Rede de Interconexão GSN

A rede de interconexão GSN (*Generalized Shuffle Network*) proposta por Bhuyan e Agrawal [BA83] é uma rede auto-roteante baseada no padrão de interconexão *Generalized Shuffle* (descrito na seção 2.9.1) e permite conectar qualquer número de M módulos de entrada a qualquer número de N módulos de saída.

2.9.1 Padrão de Interconexão Generalized Shuffle

O padrão de interconexão da rede GSN é baseado num sistema numérico que pode ser definido de acordo com [BA83] da seguinte forma:

Considere o número decimal M sendo representado como o produto de r fatores, assim, $M = m_1 * m_2 * \dots * m_r$. Na faixa de números compreendidos entre 0 e $M - 1$, cada número X é representado como um r -tuple $(x_1 x_2 \dots x_i \dots x_r)$ para:

$$0 \leq x_i \leq (m_i - 1)$$

onde x_1 é o dígito mais significativo e x_r é o dígito menos significativo. Então,

$$X = \sum_{i=1}^r x_i * w_i$$

onde w_i é um peso associado a x_i e é definido como

$$w_i = \frac{M}{m_1 m_2 \dots m_i}$$

para todo $1 \leq i \leq r$. A fim de especificar as raízes envolvidas, um número X poderá ser representado como $(X)_{m_1, m_2, \dots, m_r}$. A partir das descrições acima o *generalized shuffle* de um número como $X = (x_1 x_2 \dots x_r)_{m_1, m_2, \dots, m_r}$ é dado por:

$$S_g(X) = (x_2 x_3 \dots x_r x_1)_{m_2, m_3, \dots, m_r, m_1}.$$

Podemos também definir um m -shuffle de um inteiro X , $S_m(X)$ como:

$$S_m(X) = \begin{cases} mX \bmod (M - 1) & \text{para } 0 \leq X < M - 1 \\ X & \text{para } X = M - 1 \end{cases}$$

O m -shuffle pode ser representado da seguinte maneira:

Considerando-se um conjunto de cartas de baralho sobre o qual vamos aplicar o m -shuffle. Divide-se uma pilha de M cartas em m pilhas iguais de $\frac{M}{m}$ cartas cada uma. Toma-se a 1ª carta da 1ª pilha, a 1ª carta da 2ª pilha e assim por diante até formar uma nova pilha de m cartas. Passa-se agora a tomar as 2ªs cartas até formar uma nova 2ª pilha e prossegue até $\frac{M}{m}$ novas pilhas de m cartas cada uma. As novas pilhas são então colocadas umas sobre as outras de modo que a primeira pilha formada seja colocada no topo e a última na base.

Para exemplificar as definições dadas acima considere os seguintes dados:

$$M = 8 = 4 \times 2$$

Então, $m_1 = 4$, $m_2 = 2$ e $r = 2$

$$w_1 = \frac{M}{m_1} = \frac{8}{4} = 2$$

$$w_2 = \frac{M}{m_1 m_2} = \frac{8}{4 \times 2} = 1$$

daí para $X = \sum_{i=1}^r x_i * w_i$ de 0 a $M - 1 = 7$ temos:

$$0_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (00)_{4,2}$$

$$1_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (01)_{4,2}$$

$$2_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (10)_{4,2}$$

$$3_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (11)_{4,2}$$

$$4_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (20)_{4,2}$$

$$5_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (21)_{4,2}$$

$$6_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (30)_{4,2}$$

$$7_{10} = x_1 w_1 + x_2 w_2 = (x_1 x_2)_{m_1, m_2} = (31)_{4,2}$$

A partir da determinação dos números X podemos aplicar um *m-shuffle* a fim de prosseguir com o exemplo. Para um 4-*shuffle* de $M = 8$ entradas temos: 8 cartas divididas em 4 pilhas iguais de 2 cartas cada uma. A figura 2.20 representa o exemplo descrito acima.

2.9.2 Projeto de uma GSN

Para se projetar uma GSN $M \times N$, isto é, de M entradas e N saídas, utiliza-se como elemento de chaveamento uma chave *crossbar* controlada pelo correspondente dígito do endereço de destino. Para uma GSN com r estágios, os valores de M e N devem ser fatorados em r fatores, ou seja, $M = m_1 * m_2 * \dots * m_r$ e $N = n_1 * n_2 * \dots * n_r$. E os estágios são construídos de $\frac{M n_1 n_2 \dots n_{i-1}}{m_1 m_2 \dots m_i}$ chaves *crossbar* com tamanho $(m_i \times n_i)$ para

Figura 2.20: *Generalized Shuffle* de $M=4 \times 2$

$1 \leq i \leq r$, sendo que as chaves no i -ésimo estágio são controladas pelo i -ésimo dígito do endereço de destino. Além disso, o i -ésimo estágio é precedido por um m_i -shuffle.

2.9.3 Equivalências

De acordo com [BA83] as redes de interconexão multi-estágio como por exemplo a Omega, a n-Cubo binário indireto, a Delta e outras redes que usam chaves *crossbar* 2×2 são ou parte ou topologicamente equivalentes à GSN com $M = N = 2^n$. Uma GSN com M ou N sendo número primo equivale a uma *crossbar* $M \times N$ com $r = 1$ e, para $N = 1$ a GSN $M \times 1$ corresponde a um barramento compartilhado.

2.10 Conclusões

Arquiteturas paralelas constituem uma alternativa para a obtenção de sistemas computacionais de alta performance. Elas requerem um meio de comunicação capaz de proporcionar a interligação entre os módulos computacionais para execução paralela, daí a

importância das redes de interconexão.

No projeto de redes de interconexão devem ser considerados os requisitos de comunicação apresentados pelo sistema para se definir as características operacionais da rede e sua topologia. Em geral, topologias estáticas são mais convenientes às arquiteturas de propósito específico devido a possibilidade de se conhecer previamente o padrão de comunicação da computação em questão, enquanto topologias dinâmicas são mais adequadas às arquiteturas de propósito geral devido a propriedade de reconfigurabilidade.

Uma importante decisão recai sobre os fatores custo e desempenho, ou seja, tenta-se satisfazer as necessidades de comunicação do sistema a um custo razoável, tomando-se o cuidado de manter um bom compromisso entre custo e desempenho.

Capítulo 3

Arquitetura Proto-WOLF

3.1 Introdução

Este capítulo discute as características da arquitetura Proto-WOLF [Cav92, GNC92, GNR92]. A descrição funcional da arquitetura proporciona uma visão global do papel desempenhado por cada uma das unidades que a compõem, com ênfase nos requisitos de comunicação das chaves de interconexão.

3.2 Filosofia do Proto-WOLF

As limitações do modelo de von Neumann e a necessidade crescente por sistemas computacionais de alto desempenho conduziram à proposta do projeto do Proto-WOLF, cuja finalidade é implementar e estudar as características de um supercomputador de alto desempenho baseado no paradigma de fluxo de dados. A arquitetura do Proto-WOLF é uma proposta do Grupo de Instrumentação e Informática da USP (GII/USP) e do Grupo de Fluxo de Dados da UNICAMP. Essa arquitetura é baseada no modelo de fluxo de dados dinâmico com granularidade variável e deriva da Máquina a Fluxo de Dados de Manchester (*MMDM - Manchester Multiring Dataflow Machine*) [GWG80, GWK85, GKB87, Wat84], e leva em consideração as experiências adquiridas pelos pesquisadores do GII junto ao grupo do Professor Gurd em Manchester. A arquitetura Proto-WOLF procura solucionar alguns dos problemas detectados no projeto da *MMDM* e também se utiliza de resultados

de estudos realizados sobre a *MMDM*. Alguns desses problemas são descritos em [GNC92] e estão resumidos na seção seguinte.

3.2.1 Problemas conhecidos

Com a implementação da *MMDM* foi possível formar uma base maior de conhecimento sobre processamento a fluxo de dados e detectar problemas tais como:

- A *MMDM* apresenta um *pipeline* circular que aumenta a latência e influi na comunicação entre as unidades. O *pipeline* possui também maior número de estágios que o necessário, o que contribui para degradar a performance do sistema.
- A granularidade da *MMDM* é fina e fixa, o que dificulta a investigação de uma granularidade adequada dependente das características do algoritmo.
- Os rótulos das fichas apresentam redundâncias implicando numa necessidade de maior memória para emparelhamento. As fichas possuem, além do dado, campos de controle. Cerca de 30% da ficha contém informação processada, o restante se destina ao controle [Mag92].
- Arquiteturas a fluxo de dados puras não tratam eficientemente código estritamente seqüencial. A figura 3.1 apresenta um trecho de código estritamente seqüencial, onde a dependência de dados de uma instrução em relação à outra é direta, isto é, os dados necessários à execução de uma instrução dependem única e exclusivamente da execução da instrução anterior.

Nesse caso a ficha com o resultado da instrução 1 deve circular pela máquina a fim de possibilitar o início da execução da instrução 2, o mesmo acontecendo para as instruções 3 e 4. Isso leva a sobrecarga desnecessária da Unidade de Emparelhamento, além da latência provocada pela circulação da ficha pelo *pipeline*.

- Programas que geram alto grau de paralelismo conduzem a uma elevada utilização de memória, como por exemplo, o elevado número de instanciações simultâneas de uma função que pode provocar saturação da máquina. Assim, a ocupação de

Figura 3.1: Código seqüencial

memória torna-se um ponto crítico afetando o desempenho da máquina devido ao excesso de fichas geradas.

- O comprimento da palavra é fixo de 24 bits, não sendo adequado nem para computação simbólica para a qual é muito grande, nem para computação numérica para a qual é muito pequeno.

3.2.2 Alvos do Proto-WOLF

A arquitetura Proto-WOLF trata os problemas encontrados no projeto da *MMDM* através das seguintes características:

- A unidade de Fila de Fichas descrita na seção 3.3 é colocada fora do *pipeline* diminuindo assim, a latência.
- A granularidade é variável, podendo ser fina, cadeia de instruções e grossa.
- O Proto-WOLF possui capacidade de executar código como processador a fluxo de dados ou como processador seqüencial. Há capacidade para tratamento de código estritamente seqüencial pelas Memórias de Instruções (acopladas às Unidades Funcionais como mostra a seção 3.3) que são capazes de gerar cadeias de instruções.

- O tratamento de dados estruturados é realizado por um processador vetorial, o que permite explorar melhor o paralelismo derivado de regularidades na estrutura de dados.
- O excesso de paralelismo é gerenciado por uma Unidade de Controle de Paralelismo descrita na seção 3.3 que evita a saturação da máquina.
- O comprimento de palavra é variável de 32 a 96 *bits*.
- O Proto-WOLF apresenta Unidade de Emparelhamento capaz de efetuar endereçamento direto, eliminando a necessidade de memória associativa.

3.3 Descrição Funcional

A figura 3.2 mostra o esquema geral do Proto-WOLF. As unidades que constituem a arquitetura se comunicam através da troca de fichas, que representam os dados e são divididas em campos. Os campos pertencentes a cada ficha contêm todas as informações referentes ao dado correspondente. O comprimento de cada ficha é variável e depende de seu tipo. Os tipos de fichas são os seguintes:

3.3.1 Tipos de Fichas

Tipo PA (Ficha Par) As fichas do tipo Par são geradas a partir da Unidade de Memória de Dados e contêm os seguintes campos:

- Ficha - Indica o tipo da ficha (PA)
- Dado0 - Contém o operando esquerdo de uma operação
- Tipo0 - Contém o tipo do dado que está no campo Dado0
- Dado1 - Contém o operando direito de uma operação
- Tipo1 - Contém o tipo do dado que está no campo Dado1
- Nó - Contém o endereço para o qual os dados da ficha se destinam

- Na - Indica a instanciação da ficha
- Ix - Indica o índice da ficha

Tipo PR (Ficha Programa) As fichas do tipo Programa são inseridas na máquina via mecanismo de Entrada-Saída e contêm os seguintes campos:

- Ficha - Indica o tipo da ficha (PR)
- Cod.oper - Contém o código de operação, isto é, corresponde a uma instrução do programa
- Nó - Contém o endereço onde a ficha PR será armazenada
- Nó_dest0 - Contém o endereço de destino do primeiro resultado gerado pela execução da ficha executável
- F_emp0 - Contém a função de emparelhamento do primeiro resultado gerado pela execução da ficha executável
- Nó_dest1 - Contém o endereço de destino do segundo resultado gerado pela execução da ficha executável
- F_emp1 - Contém a função de emparelhamento do segundo resultado gerado pela execução da ficha executável

Tipo EX (Ficha Executável) As fichas do tipo Executável são criadas a partir da Memória de Instruções e contêm os seguintes campos:

- Ficha - Indica o tipo da ficha (EX)
- Os campos Dado0, Tipo0, Dado1, Tipo1, Na e Ix são idênticos aos utilizados nas fichas do tipo PA, e os campos Cod.oper., Nó_dest0, F_emp0, Nó_dest1 e F_emp1 são idênticos aos das fichas do tipo PR.

Outras As fichas dos tipos NA (Nome de Ativação), CD (Cor-Destino), DE (Destino), BO (Booleana), IN (Inteiro), RE (Real), QQ (Ponteiro), ZZ (Fim de *Stream*) têm estruturas semelhantes, apresentando diferenças apenas nos campos de Ficha e Dado.

São criadas nas Unidades Funcionais ou então inseridas via mecanismo de Entrada-Saída, exceto as fichas do tipo QQ que são geradas na Unidade de Processador Vetorial e Memória Vetorial. Essas fichas têm os campos Ficha, Dado0, No, F_emp0, Na e Ix, já descritos anteriormente.

A ficha do tipo QQ é um ponteiro para o início da estrutura armazenada na Memória de Estruturas e contém além do endereço inicial da estrutura o correspondente tamanho da mesma. O tipo de ficha ZZ é usado no mecanismo que verifica se uma operação sobre um vetor de dados terminou, isto é, se tal operação foi aplicada em todos os elementos do vetor de dados. Seu campo F_emp0 contém a função de emparelhamento. Tal função não se refere nesse caso (como nos casos anteriores) ao campo Dado0 e sim ao campo Ficha, visto que o mecanismo de controle que usa esse tipo de ficha, não necessita de um valor para o campo Dado0.

Tipo SA (Ficha Saída) As fichas do tipo Saída são produzidas nas Unidades Funcionais e possuem os seguintes campos:

- Ficha - Tipo de ficha (SA)
- Dado0 - Contém um dado que será enviado à saída como resultado de um programa
- Tipo0 - Contém o tipo do dado que está no campo Dado0
- Dado - Contém o valor de *Stream* que indica o resultado da saída. Se um programa gera saídas múltiplas, esse campo identifica cada uma.
- Na e Ix - Idênticos aos anteriores.

3.3.2 Unidades

A arquitetura Proto-WOLF é composta por 10 unidades:

Mecanismo de Entrada e Saída (E/S) O mecanismo de E/S é responsável pela inserção e extração de fichas na máquina. As fichas são inseridas na máquina pela

Figura 3.2: Arquitetura Proto-WOLF

via de Entrada, conectada à Chave de Coleta. Durante o processamento, a via Saída, através das fichas do tipo SA, apresenta o(s) resultado(s) de um programa. É através dessas vias que se faz a conexão da máquina ao exterior.

Unidade Chave de Coleta (CC) A Chave de Coleta (CC) reúne as fichas recebidas da via de Entrada e as transmite às unidades seguintes no *pipeline*. A CC possui duas saídas separadas em direção à Unidade de Emparelhamento (UE): uma delas transmite fichas para a Memória de Dados (MD) e a outra transmite endereços para a UE. Caso não haja possibilidade de efetuar a transmissão devido a indisponibilidade dessas unidades, a CC envia as fichas para a Fila de Fichas (FF).

Unidade Fila de Fichas (FF) Armazena as fichas que não puderam ser enviadas à UE e as distribui posteriormente, quando a UE estiver disponível. A FF é uma fila do tipo FIFO e responde pelo controle de fluxo de fichas na máquina evitando que os anéis¹ sejam sobrecarregados.

¹Cada anel é formado por uma Memória de Instruções (MI) e uma Unidade Funcional (UF).

Unidade de Emparelhamento (UE) Realiza o emparelhamento de operandos. Ela usa o endereço recebido da CC para procurar a ficha parceira (seção 1.2). Caso não encontre a ficha parceira, então, gera um sinal de escrita para armazenar o dado cujo par não foi localizado. Assim, essa ficha fica armazenada esperando pelo par correspondente. Mas, se o par for encontrado, a UE gera um sinal de leitura para que a Memória de Dados (MD) exponha o dado procurado. Assim, a UE efetua o emparelhamento combinando a saída da MD de modo a formar um outro tipo de ficha composta pelo par de operandos das duas fichas de mesma cor² (seção 1.2) dirigidas para um mesmo nó ou por uma ficha *BYPASS* combinada com uma ficha *null*. A nova ficha formada a partir desse emparelhamento é denominada de ficha par (PA).

Unidade de Memória de Dados (MD) Quando a UE não localiza a ficha parceira na unidade de Memória de Dados (MD), o dado é armazenado na MD e fica à espera do parceiro correspondente (ficha de mesma cor dirigida ao mesmo nó). Quando o parceiro chega, a MD apresenta o dado armazenado, possibilitando o emparelhamento e portanto a formação da ficha par (PA).

Unidade Chave de Distribuição (CD) Recebe as fichas emparelhadas na MD e as distribui para os anéis, para a unidade de Processador Vetorial e Memória Vetorial (PVMV) ou para a Unidade de Controle de Paralelismo (UCP), dependendo do tipo da ficha.

Unidade de Memória de Instruções (MI) A Memória de Instruções contém todas as informações sobre o nó para o qual a ficha par (PA) se destina. Essas informações dizem respeito por exemplo à operação a ser efetuada sobre os dados da ficha par bem como o destino do(s) resultado(s) ou ainda ao segundo operando da instrução para uma ficha *BYPASS*. A partir das informações do nó e da ficha par a MI produz uma ficha executável (EX) que contém todas as informações necessárias à execução do nó. A MI envia então a ficha executável à sua correspondente Unidade Funcional (UF).

²A cor corresponde aos campos Ix e Na e identifica cada instanciação na re-utilização recursiva dos grafos.

Unidade Funcional (UF) Responsável pela execução da ficha executável (EX) recebida da MI, realizando a operação representada pelo nó no grafo sobre os operandos correspondentes. A UF pode gerar dois tipos de fichas de resultados, *livres* e *encadeados*. Os resultados *livres* são enviados à CC para continuarem a circular pela máquina, enquanto os resultados *encadeados* continuam na UF. Os nós podem fazer parte de uma cadeia e essa informação deve estar armazenada na MI para que ela possa utilizar o endereço de destino da ficha executável como endereço de busca para formar a próxima ficha executável da cadeia. Assim, os resultados da ficha executável anterior que acaba de ser processada pela UF serão utilizados como operandos da ficha executável subsequente.

Unidade de Memória de Rascunho (MR) A Memória de Rascunho (MR) armazena resultados intermediários, isto é, os resultados *encadeados* produzidos pela UF. Esses resultados devem ser armazenados para servirem de operandos às próximas instruções da cadeia.

Unidade Processador Vetorial e Memória Vetorial (PVMV) É responsável pelo tratamento de dados estruturados. Possui unidades de Memórias de Estruturas (ME) que podem guardar os pedidos de leitura de dados ainda não produzidos, possibilitando, dessa forma, a execução de operações de leitura e escrita em ordem inversa. Outra característica das MEs é a alocação de espaço sob demanda e liberação automática de espaço. Como essa unidade processa e armazena estruturas de dados, não há necessidade que esses dados circulem pelo *pipeline*, evitando assim, tráfego desnecessário. Além disso, é muito mais adequado utilizar uma unidade dedicada a processamento de estruturas de dados para realizar essas tarefas, pois o processamento é feito de modo mais eficiente, e também diminuindo a necessidade de índices nas fichas, e conseqüentemente, reduzindo o uso de memória.

Unidade de Controle de Paralelismo (UCP) Um dos maiores problemas encontrados em arquiteturas a fluxo de dados é a geração excessiva de paralelismo devido a possibilidade de existência simultânea de diversas instâncias de uma mesma instrução. A Unidade de Controle de Paralelismo (UCP) desempenha um papel importante na arquitetura, pois possui um mecanismo de controle que restringe o número

de instanciações distintas de uma mesma instrução. O método de *Throttle* [Rug87] foi desenvolvido para a máquina de Manchester por Ruggiero e faz o gerenciamento de paralelismo através do controle da ativação de funções, de acordo com o nível de atividade da máquina [Mag92]. Quando esse nível está elevado, os pedidos de ativação de função são suspensos até o término da execução de algumas funções. Caso o nível esteja baixo, o pedido é atendido. A ocupação da unidade de Fila de Fichas indica o nível de atividade da máquina. Nível alto significa que as Unidades Funcionais não estão conseguindo consumir as fichas na mesma proporção em que estas estão sendo geradas, daí a suspensão das ativações. As ativações são liberadas quando a ocupação da unidade de Fila de Fichas cai para o nível baixo.

3.4 Requisitos das Chaves de Conexão

A função das chaves de Coleta e de Distribuição é transferir fichas provenientes das unidades conectadas em suas entradas às unidades conectadas em suas saídas. Diferentemente dos sistemas multiprocessadores e multicomputadores, a arquitetura Proto-WOLF requer comunicação unidirecional, isto é, as unidades interconectadas às entradas das redes somente enviam fichas e não requerem respostas das unidades conectadas nas saídas como ocorre por exemplo em sistemas multiprocessadores, onde os processadores ligados às entradas da rede de conexão requerem respostas das unidades de memória conectadas nas saídas. Devido a essa característica de comunicação unidirecional o fator velocidade de transmissão é menos crítico que no caso dos multiprocessadores.

A arquitetura do Proto-WOLF é de uso geral, sua finalidade é computar aplicações de diversas áreas. Devido a isso, a estrutura de comunicação da arquitetura não deve estar atrelada ao padrão de comunicação de uma aplicação particular. Esses fatores influem na escolha da topologia das redes de interconexão. O princípio básico para a escolha da topologia é a condição de que ela satisfaça as necessidades intrínsecas de comunicação da aplicação endereçada pelo sistema. Para uma boa escolha há necessidade de se conhecer previamente o padrão de comunicação exigido pela aplicação, o que na maioria das vezes não é possível prever *a priori*. Além disso, uma máquina de propósito geral endereça

uma ampla variedade de aplicações, o que dificulta a tarefa de se fazer prognósticos dos requisitos de comunicação exigidos.

Nas redes estáticas a estrutura da computação é completamente determinada antes da execução, nesse caso a escolha da topologia depende do padrão de comunicação esperado. Levando-se em consideração esses fatores, conclui-se que a escolha da categoria estática é em geral mais conveniente às arquiteturas de propósito específico, cujo padrão de comunicação pode geralmente ser conhecido de antemão.

Baseado no fato da máquina ser de propósito geral, a topologia dinâmica pode satisfazer bem às necessidades de comunicação do sistema devido à sua característica de reconfigurabilidade que permite atender a padrões arbitrários de comunicação, presentes quando se efetua computações sobre uma ampla variedade de aplicações.

Como descrito anteriormente, os dados são transmitidos através de fichas e as unidades que constituem a arquitetura se comunicam por meio de troca de fichas como em arquiteturas de passagem de mensagens (pacotes de informações). As fichas podem ser vistas como pacotes que contêm as diversas informações necessárias à execução do programa, assim, as redes de interconexão devem transferir pacotes de informações.

Os anéis entre as chaves têm a capacidade de realizar suas atividades independentemente e em paralelo. Sendo assim, a transferência de fichas através das redes deve ser realizada de forma assíncrona, para suprir os anéis que estiverem disponíveis para iniciar o processamento.

Apesar da arquitetura não impor restrições quanto a estratégia de controle das redes, é mais adequado, por questões de simplicidade e confiabilidade do sistema, adotar a estratégia de controle distribuído. Isso evita o problema de se utilizar uma única unidade geradora de sinais de controle que pode levar a um *engarrafamento* e comprometer o sistema em termos de performance e confiabilidade, além do que a estratégia de controle distribuído se adequa melhor a sistemas MIMD, enquanto que, a estratégia de controle centralizado é mais conveniente para sistemas SIMD [IT89].

Com base nesses requisitos essenciais podemos tomar as decisões de projeto para as redes de interconexão da arquitetura Proto-WOLF:

- Método de Chaveamento: Por Pacotes

- Modo de Operação: Assíncrono
- Estratégia de Controle: Distribuído

3.4.1 Características da Chave de Coleta

Às entradas da Chave de Coleta estão conectadas as seguintes unidades: unidade de Entrada-Saída (E/S), unidade de Fila de Fichas (FF), Unidade de Processador Vetorial e Memória Vetorial (PVMV), Unidade de Controle de Paralelismo (UCP) e os anéis (Memórias de Instruções (MI) e Unidades Funcionais (UF)).

Nas saídas da chave encontram-se conectadas as seguintes unidades: Unidade de Entrada-Saída (E/S), Unidade de Fila de Fichas (FF), Unidade de Emparelhamento (UE) e Memória de Dados (MD).

As funções da Chave de Coleta são: coletar as fichas provenientes das diferentes unidades conectadas em suas entradas, detectar o tipo de cada ficha recebida e determinar a saída para a qual a ficha será enviada e realizar a transmissão.

A Chave de Coleta pode coletar todos os tipos de fichas citados na seção 3.3 exceto os tipos PA e EX, que são produzidos e consumidos antes de atingirem a Chave de Coleta. A Chave de Coleta pode enviar também todos os tipos de fichas com exceção dos tipos PA e EX pelos mesmos motivos citados acima.

A Chave de Coleta é desbalanceada, com uma topologia assimétrica, isto é, o número de entradas da chave é diferente do número de saídas. Esse desbalanceamento é devido à estrutura da arquitetura que possui apenas uma Memória de Dados. Assim, a Chave de Coleta representa um *gargalo* por onde a maioria das fichas que circulam pela máquina tentam passar. Comparando-se aos sistemas multiprocessadores, onde em geral, as redes de conexão são simétricas possibilitando um grau maior de comunicação entre os módulos de entrada-saída, a Chave de Coleta permite um grau menor de transações devido ao número de saídas ser menor que o de entradas. Se houvesse mais Memórias de Dados seria possível tornar a Chave de Coleta mais eficiente, de modo a explorar mais paralelismo de transações.

3.4.2 Características da Chave de Distribuição

A Chave de Distribuição pode receber fichas de dois tipos, PA (Par) produzidas a partir da Memória de Dados (MD) e fichas do tipo EX (Executável) criadas pela Memória de Instruções (MI). As saídas da Chave de Distribuição estão conectadas às seguintes unidades: Unidade de Processador Vetorial e Memória Vetorial (PVMV), Unidade de Controle de Paralelismo (UCP) e aos anéis (Memórias de Instruções e Unidades Funcionais).

Assim como a Chave de Coleta a Chave de Distribuição também é topologicamente assimétrica, apresentando maior número de saídas que de entradas. Essa característica seria desejável em sistemas multiprocessadores, em que, por exemplo, às entradas estariam conectados processadores e às saídas unidades de memórias, nesse caso, o número de comunicações poderia ser bastante elevado, acarretando numa utilização eficiente da rede de conexão. Mas, a Chave de Distribuição possui somente uma entrada através da qual tenta *suprir* os anéis. Se fosse equilibrado o número de entradas e saídas nesse caso, os anéis poderiam ser explorados mais eficientemente.

3.5 Conclusões

A arquitetura Proto-WOLF apresenta características que buscam solucionar alguns dos problemas encontrados em arquiteturas baseadas no paradigma de fluxo de dados puro, tais como geração excessiva de paralelismo, processamento de dados estruturados, processamento de código estritamente seqüencial, elevado uso de recursos de memória, granularidade fixa. Em relação a *MMDM*, ela incorpora algumas características e aperfeiçoa outras, como a Unidade de Controle de Paralelismo e a Unidade de Fila de Fichas. A Unidade de Controle de Paralelismo proporciona à máquina a capacidade de tratamento a um dos problemas de sobrecarga encontrados em arquiteturas a fluxo de dados que é o excesso de paralelismo. Essa unidade possui um eficiente mecanismo para controlar e evitar exposição de paralelismo excessivo, o que resulta em diminuição do uso excessivo de memória. A Fila de Fichas é externa ao *pipeline*, o que implica numa diminuição da latência, visto que, não há necessidade das fichas obrigatoriamente circularem por ela.

A Unidade de Processador Vetorial e Memória Vetorial permite diminuir o *overhead* nos anéis (MI/UF), pois evita que os dados estruturados tenham de circular pelo *pipeline* para serem processados. Devido a implementação das Unidades de Controle de Paralelismo e de Processador Vetorial e Memória Vetorial há um ganho substancial em eficiência na utilização de memória pelo sistema.

A granularidade do Proto-WOLF é variável, possibilitando a execução tanto como processador a fluxo de dados como processador seqüencial. Essa característica soluciona o problema de tratamento de código estritamente seqüencial conforme descrito na seção 3.2.

Uma outra característica relevante é a do emparelhamento direto através da Unidade de Emparelhamento. A Unidade de Emparelhamento é independente da Memória de Dados e pode fazer o endereçamento direto a partir do endereço recebido da Chave de Coleta. Isso evita o uso de memória associativa que é cara e de difícil implementação.

Um dos problemas encontrados em arquiteturas a fluxo de dados dinâmicas por rotação de fichas é a necessidade de utilização de diversos campos extras, além daquele do próprio dado. Isso resulta em *overhead* devido a redundâncias em diversos campos das fichas, aumentando de forma desnecessária o espaço de emparelhamento. Através do trabalho proposto por Magna [Mag92] é possível eliminar tais redundâncias, permitindo assim, uma implementação da máquina de forma mais simples e eficiente.

Capítulo 4

Implementação e Análise das Redes

4.1 Introdução

Este capítulo exercita as chaves de interconexão propostas para a arquitetura e identifica o impacto causado no sistema pelas diferentes topologias. Identifica também a rede mais adequada à arquitetura Proto-WOLF.

4.2 Simulador SAW

O simulador SAW (Simulador da Arquitetura Proto-WOLF) é descrito na figura 4.1. No estágio atual ele ainda não implementa totalmente a arquitetura Proto-WOLF descrita no capítulo 3.

4.2.1 Descrição Funcional do Simulador SAW

Através das vias de Entrada-Saída são introduzidas as fichas que circularão pelo simulador. As fichas passam pela Chave de Coleta, que determina o destino das mesmas, se vão para a Memória de Dados para efetuar o emparelhamento ou se vão para a Fila de Fichas no caso de a Memória de Dados não estar disponível. A Chave de Coleta identifica também os dados destinados à saída. Quando uma ficha contendo um dado de saída chega, proveniente das Unidades Funcionais, a Chave de Coleta a envia para a saída, não

Figura 4.1: Arquitetura simulada pelo SAW

permitindo que ela entre na Fila de Fichas. Como geralmente o tráfego de fichas de saída é menor que a quantidade de fichas destinadas à Memória de Dados, evita-se o *overhead* de armazenamento na Fila de Fichas. Se a via de Entrada-Saída já possuir um dado no *buffer* de entrada, então, a Chave de Coleta bloqueia a transmissão de dados destinados à saída até que o *buffer* seja liberado.

As fichas não destinadas à saída são enviadas à Memória de Dados (quando está disponível) para emparelhamento. O mecanismo de emparelhamento é semelhante ao descrito no capítulo 3 (seção 3.3.2). O emparelhamento é realizado pela Memória de Dados, que é capaz de verificar pelo endereço da ficha de entrada (campos Nome de Ativação, Índice e Nó), se a ficha parceira está ou não armazenada. Se estiver, então, a Memória de Dados procura a parceira e efetua o emparelhamento. Se não estiver, a Memória de Dados armazena a ficha.

A Memória de Instruções localizada paralela à Memória de Dados pode gerar uma ficha executável caso os dados correspondam à uma instrução de controle da Memória de Estruturas. Se tais dados não corresponderem a uma instrução destinada a realizar alguma operação na Memória de Estruturas, a ficha par é transmitida intacta à Chave de Distribuição, para consulta à Memória de Instruções interna.

A Chave de Distribuição recebe a ficha vinda da Memória de Instruções e a envia para a unidade de destino de acordo com o tipo da ficha e a disponibilidade das unidades

subseqüentes interconectadas às suas saídas.

A Memória de Estruturas armazena dados estruturados que vão ser processados nas Unidades Funcionais. Ela evita que dados estruturados fiquem circulando pelo *pipeline* desnecessariamente. A Memória de Instruções imediatamente anterior à Chave de Distribuição gera fichas executáveis que servem para controlar as ações da Memória de Estruturas, como por exemplo, alocação de espaço, liberação de espaço, escrita ou leitura.

As fichas par são enviadas às Memórias de Instruções localizadas entre as Chaves de Coleta e de Distribuição, que formam as fichas do tipo executável, as transmitem para as respectivas Unidades Funcionais onde são processadas. Os resultados produzidos pelas Unidades Funcionais são enviados à Chave de Coleta, fechando assim o ciclo das fichas.

4.3 Definição das Redes de Conexão

Baseado nos requisitos de comunicação da arquitetura, descritos na seção 3.4 do capítulo 3 e na descrição da implementação atual do simulador SAW (seção 4.2.1), podemos definir as redes de conexão mais adequadas à arquitetura Proto-WOLF.

Considerando-se que o simulador MR [BT88] (simulador da *MMDM*) adotava entre 4 a 5 processadores por anel, apesar de terem ocorridas mudanças tecnológicas desde então, mantemos neste trabalho a mesma razão, cinco anéis, para efeito de comparações com os trabalhos já desenvolvidos em Manchester. Outros fatores que influíram na escolha do número de anéis foram a redução do número de variáveis, o que simplifica a simulação, e a preocupação inicial em pesquisar o impacto das topologias sobre o sistema, tratando a expansão do número de anéis num trabalho futuro.

A partir daí, podemos então, definir o número de conexões exigidas de cada uma das chaves. Para a Chave de Coleta temos oito entradas correspondendo às seguintes unidades:

- Unidade de Fila de Fichas
- Unidade de Entrada-Saída
- Unidade de Memória de Estruturas

- 5 Anéis (MI/UF)

e três saídas que são:

- Unidade de Fila de Fichas
- Unidade de Entrada-Saída
- Unidade de Memória de Dados

Já a Chave de Distribuição possui uma entrada vinda da Memória de Instruções e seis saídas, correspondentes aos cinco anéis e uma saída para a Memória de Estruturas.

4.4 Candidatas às Redes de Conexão

Baseado nos parâmetros apresentados, já pudemos descartar as redes com topologia estática, e restringimos ainda mais as alternativas de escolha. Dentre as diversas topologias dinâmicas as candidatas mais prováveis são as chaves *crossbar*, as redes Multi-estágio e as redes GSN porque apresentam características capazes de satisfazer os requisitos de comunicação apresentados.

As redes Multi-estágio e as redes GSN utilizam como unidade básica para efetuar o roteamento chaves *crossbar*. Devido a isso, para efeito comparativo em termos de custo, utilizaremos o modelo de custo proposto por Bhuyan e Agrawal [BA83] que é um método simples e considera uma chave *crossbar* $m \times n$ (m entradas e n saídas) como tendo um custo de $m * n$ unidades, as quais correspondem ao número de pontos que efetuam as interligações. Assim, definidas as redes que serão simuladas, poderemos compará-las em termos de performance e também de custo.

4.5 Implementação Ideal das Chaves de Distribuição e de Coleta

As redes Cross1 e Cross2 correspondem às Chaves de Distribuição e de Coleta respectivamente, implementadas na versão anterior do simulador, e são descritas nas seções 4.5.1

e 4.5.2. Essas implementações correspondem aos modelos ideais, visto que a primeira versão do simulador era mais simplificada e não considerava o desempenho e custo das redes.

4.5.1 Cross1

A Chave de Distribuição implementada através da rede Cross1 está representada na figura 4.2. A rede Cross1 identifica o tipo da ficha através de seu endereço, caso ela seja do tipo EX a chave Cross1 a transmite para a Memória de Estruturas se o *buffer* desta estiver livre, caso contrário a ficha é bloqueada na entrada. No entanto, se o tipo for PA, a ficha é enviada à primeira Unidade Funcional livre a partir do topo, ou seja, menor endereço tem prioridade maior. Se todas as Unidades Funcionais estiverem ocupadas, então, a ficha é bloqueada. Seu custo é de 6 unidades.

Figura 4.2: Chave de Distribuição - Cross1

4.5.2 Cross2

A chave Cross2 identifica o tipo da ficha através de seu endereço e a envia para a saída correspondente desde que esteja disponível. Somente as Unidades Funcionais enviam fichas para a saída e os pontos que realizam essa interligação juntamente com o ponto que conecta a Fila de Fichas à Memória de Dados possuem o mesmo custo daqueles das redes propostas para a Chave de Coleta na seção 4.6.

Entretanto, os pontos que interligam as entradas Memória de Estruturas, Entrada-Saída e Unidades Funcionais às saídas Fila de Fichas e Memória de Dados são mais complexos, pois, realizam re-roteamento, ou seja, quando a entrada é destinada à Memória de Dados e esta não se encontra disponível; é realizado então um re-roteamento para a Fila de Fichas caso ela esteja disponível, se não, a ficha é bloqueada. A figura 4.3 representa a chave Cross2.

Figura 4.3: Chave de Coleta - Cross2

Para efeito de comparação iremos considerar o custo de tais pontos como sendo de 2 unidades, daí, temos o custo total da chave Cross2 que é igual a 34 unidades. As fichas são enviadas de acordo com o campo tempo, ou seja, aquela que possuir menor tempo tem prioridade para ser transmitida. Em caso de conflito, as fichas são enviadas obedecendo a ordem crescente de endereços das portas de entrada da rede. As fichas do tipo SA não são enviadas à Fila de Fichas; quando o *buffer* de entrada da Unidade de Entrada-Saída está ocupado, elas ficam bloqueadas. A Fila de Fichas somente envia fichas para a Memória de Dados para evitar que as fichas fiquem retornando a ela quando a Memória de Dados não se encontra disponível.

4.6 Avaliação Estática da Chave de Coleta

Faremos a seguir uma análise das redes propostas considerando-as à parte do sistema. Estudaremos o modelo lógico, a implementação e a expansibilidade de cada uma.

4.6.1 Modelo Lógico

A partir da figura 4.4 podemos descrever as implementações da Chave de Coleta. Essa figura apresenta, além da chave propriamente dita, uma outra unidade denominada Arbitrador, cuja função básica é coordenar a transmissão de fichas destinadas à unidade de Memória de Dados. O mecanismo de coordenação do Arbitrador verifica se os *buffers* de entrada da unidade de Memória de Dados e da unidade de Fila de Fichas estão livres ou ocupados. Sempre que possível, o Arbitrador tenta enviar fichas à unidade de Memória de Dados, ou seja, nos casos onde: os *buffers* tanto da unidade de Memória de Dados quanto da unidade de Fila de Fichas estão vazios; somente o *buffer* da unidade de Memória de Dados está vazio.

Figura 4.4: Chave de Coleta

A ficha só é enviada à unidade de Fila de Fichas quando o *buffer* da unidade de Memória de Dados estiver ocupado e logicamente o *buffer* da unidade de Fila de Fichas estiver vazio. Caso os dois *buffers* estejam ocupados a ficha é bloqueada. Através do uso do Arbitrador a chave pode ser implementada de forma mais simples, pois não há necessidade de os elementos de chaveamento tomarem decisões baseados na disponibilidade ou não da unidade de Memória de Dados. Desse modo, os elementos de chaveamento somente cumprem sua tarefa de roteamento utilizando o endereço de destino da ficha

a ser transmitida, sem necessidade de lógica extra para detectar a ocupação da unidade de Fila de Fichas e da unidade de Memória de Dados. Além disso, é possível reduzir o número de saídas da rede de conexão de 3 para 2, contribuindo assim para a redução do custo da rede.

4.6.2 Implementação da Chave de Coleta

Esta seção descreve as três implementações da Chave de Coleta utilizando as seguintes estruturas de interconexão:

- Chave *Crossbar* 8x2
- Rede 8x2 derivada de uma rede GSN 8x4
- Rede 8x2 derivada de uma rede multi-estágio Delta 9x4

4.6.2.1 Crossbar 8x2

A figura 4.5 representa a Chave de Coleta implementada através de uma chave *crossbar* 8x2.

Figura 4.5: Chave de Coleta - Crossbar 8x2

Através dos pontos da figura podemos verificar o custo da chave que é de 13 unidades. As entradas da rede provenientes da Fila de Fichas, via de Entrada e Memória de Estruturas não possuem interligações com a via Saída porque não enviam fichas do tipo SA, somente transmitem fichas endereçadas à Memória de Dados. A rede *crossbar* identifica o tipo da ficha que chega através de seu endereço, podendo assim, enviá-la para a saída correspondente, desde que esta esteja disponível; a ficha a ser enviada é aquela que apresenta o campo tempo menor. Quando a ficha é do tipo saída, a chave verifica se o *buffer* de entrada da unidade de Entrada-Saída está livre, se estiver, a chave transmite a ficha, se não, a ficha fica bloqueada na entrada da chave até que haja disponibilidade da correspondente saída.

No caso da ficha ser endereçada à Memória de Dados, a chave verifica se o *buffer* de entrada do Arbitrador está ocupado ou não. Quando esse *buffer* está ocupado a ficha é bloqueada na entrada da chave e espera até que o *buffer* esteja desocupado. Quando a ficha é enviada ao Arbitrador este verifica a possibilidade de enviá-la à Memória de Dados ou à Fila de Fichas conforme descrito na seção 4.6. A ficha que é enviada à Fila de Fichas fica armazenada por um ciclo para evitar de continuar circulando pela chave desnecessariamente enquanto o *buffer* da Memória de Dados estiver ocupado. Na ocorrência de conflitos, quando duas ou mais fichas são dirigidas ao mesmo destino, a política de resolução de conflito da chave determina que será transmitida a ficha cuja porta de entrada na rede tenha o endereço de menor ordem, ou seja, a transmissão nesse caso obedece a ordem crescente de endereços, sendo que a porta com endereço menor é a do topo da rede.

4.6.2.2 Rede 8x2 GSN-derivada

A Chave de Coleta implementada a partir de uma derivação da rede GSN 8x4 está representada na figura 4.6 e possui um custo de 12 unidades. Essa implementação apresenta um estágio a mais que aquela baseada na chave *crossbar* (seção 4.6.2.1) que é caracterizada por um custo 8,3% maior. A rede GSN-derivada também identifica a ficha através de seu endereço e efetua a transmissão utilizando a mesma política de resolução de conflito. Os elementos de chaveamento do primeiro estágio somente enviam as

fichas para o estágio posterior se os respectivos *buffers* de entrada do elemento de chaveamento do segundo estágio estiverem livres. Caso estejam ocupados, as fichas permanecem bloqueadas nas entradas da rede. No segundo estágio as saídas são tratadas como no caso anterior, se a ficha é do tipo saída ela é enviada à unidade de Entrada-Saída desde que haja disponibilidade (*buffer* de entrada desocupado), caso contrário, ela é bloqueada na entrada do elemento de chaveamento até liberação do *buffer*. As fichas destinadas à Memória de Dados são enviadas ao Arbitrador que realiza a transmissão do mesmo modo descrito anteriormente, se a ficha não puder ser enviada ao Arbitrador, então ela permanece bloqueada na entrada do elemento de chaveamento.

Figura 4.6: Chave de Coleta - GSN-derivada 8x2

4.6.2.3 Rede 8x2 Delta-derivada

A figura 4.7 mostra a implementação da Chave de Coleta utilizando uma rede multi-estágio. Pode-se notar que, essa implementação apresenta o mesmo custo daquela referente à *crossbar* 8x2 (13 unidades), portanto, 8,3% mais cara que a implementação com a rede derivada da GSN e com a desvantagem de possuir dois estágios. A transmissão de fichas

através dessa rede segue o mesmo princípio das anteriores mantendo a mesma política de resolução de conflitos e o mesmo modelo de bloqueios no caso de *buffers* ocupados.

Figura 4.7: Chave de Coleta - Delta-derivada 8x2

4.6.3 Expansibilidade

Levando-se em consideração a possibilidade de se expandir o sistema através do aumento do número de anéis podemos analisar as mesmas redes das seções anteriores em termos de custo, para 13 anéis.

Crossbar 16x2 O custo para implementar a Chave de Coleta utilizando uma chave *crossbar* 16x2 é de 29 unidades.

Rede 16x2 GSN-derivada A implementação da Chave de Coleta através de uma rede derivada a partir de uma GSN 16x4 é de 20 unidades, portanto, a chave *crossbar* apresenta um custo 45% maior.

Rede 16x2 Delta-derivada A implementação baseada numa rede derivada da rede Delta 16x4 possui custo de 24 unidades, sendo então, 20% mais cara que a im-

plementação baseada na rede GSN 16x4 e 17,2% mais barata que a implementação referente à chave *crossbar* 16x2.

4.7 Avaliação Estática da Chave de Distribuição

Também será feita a descrição do modelo lógico e implementação da Chave de Distribuição.

4.7.1 Modelo Lógico

A Chave de Distribuição apresentada na figura 4.8 tem a particularidade de tratar somente os tipos de fichas par (PA) e executável (EX), o que simplifica seu mecanismo de controle.

Figura 4.8: Chave de Distribuição

O tipo EX deve ser enviado somente à unidade de Memória de Estruturas e o tipo PA somente aos anéis. Devido a essa característica, podemos separar a Chave de Distribuição em dois blocos, a fim de simplificá-la. A chave toma, então, o seguinte formato descrito na figura 4.9.

Figura 4.9: Chave de Distribuição - Demultiplexador

4.7.2 Implementação

A implementação da Chave de Distribuição denominada Demultiplexador apresenta um elemento de chaveamento semelhante ao Arbitrador da Chave de Coleta e um Distribuidor que tem a função de detectar qual dos anéis está habilitado a receber a ficha par. O elemento de chaveamento identifica o tipo da ficha recebida e a envia para uma das duas saídas de acordo com o tipo. As unidades de Memórias de Instruções que se encontram disponíveis podem receber a ficha do Distribuidor. No caso de conflito, quando duas ou mais Memórias de Instruções estão habilitadas a receberem fichas, escolhe-se pela ordem crescente de endereços das Memórias de Instruções, ou seja, aquela que possuir o menor endereço receberá a ficha (o endereço menor é o do topo). Quando a ficha é do tipo EX o elemento de chaveamento verifica se o *buffer* de entrada da Memória de Estruturas está livre para então efetuar a transmissão, no caso do *buffer* estar ocupado, a ficha permanece bloqueada na entrada da chave.

Quando o elemento de chaveamento detecta ficha do tipo PA, ele verifica se o *buffer* de entrada do Distribuidor está ocupado ou não. Se estiver ocupado, a ficha fica bloqueada na entrada da chave, caso contrário é enviada ao Distribuidor para posterior transmissão às Memórias de Instruções.

4.8 Avaliação Dinâmica

A avaliação dinâmica da implementação das redes de interconexão da arquitetura Proto-WOLF consiste na análise do impacto causado pela inserção dessas redes ao sistema. Essa análise será feita através do uso de alguns *benchmarks* empregados no estudo de arquiteturas paralelas. Esses *benchmarks* são programas de comportamento conhecido [CDJ84, GKW85, GNC92] que procuram esgotar os recursos computacionais disponíveis na máquina em avaliação, isto é, eles tentam explorar características que levam a máquina a trabalhar em situações de limite. Os programas foram codificados em linguagem SISAL.

Para efeito de validação serão utilizados os seguintes *benchmarks*: Incremento, Função de Ackermann, Integração Binária e Multiplicação de Matrizes.

Incremento Esse é o mais simples dos *benchmarks* utilizados nessa validação. Sua função é efetuar o incremento seqüencial de uma variável. O código em SISAL é mostrado na figura 4.10.

```
export main
  function main(returns integer)
    for i in 1,100
      e := i
    returns value of sum e
  end for
end function
```

Figura 4.10: Programa Incremento

Função de Ackermann O programa de cálculo da Função de Ackermann explora características de recursividade, eficiência de chamadas de procedimentos. Por isso, é amplamente utilizado na validação de arquiteturas paralelas.

O número de chamadas do algoritmo para efetuar o cálculo de Ackermann(3,n) é dado por: $(128 * 4^n - 120 * 2^n + 9 * n + 37)/3$. E o resultado é dado por: $Ackermann(3, n) - 1 = 2^{n+3} - 4$. O código em SISAL é mostrado na figura 4.11.

```

export main
function Ack(m,n:integer returns integer)
  if m = 0 then
    n + 1
  elseif n = 0 then
    Ack(m-1,1)
  else
    Ack(m-1,Ack(m,n-1))
  end if
end function

```

Figura 4.11: Programa Ackermann

Integração Binária É um *benchmark* bastante utilizado por apresentar alto grau de recursividade. Efetua o cálculo da área de uma função por integração binária utilizando a regra do trapézio. A função $y = x^2 - 6x - 10$ será utilizada e os parâmetros de entrada entre os quais será integrada são L e R. O código em SISAL é mostrado na figura 4.12.

Multiplicação de Matrizes É um *benchmark* que explora as características da Unidade de Memória de Estruturas. O código SISAL é mostrado na figura 4.13.

A figura 4.14 mostra o processo de compilação de um programa em SISAL. Esse processo é dividido em várias etapas:

- O programa em SISAL é convertido para o formato intermediário IF1, que é uma representação gráfica de SISAL e independe da máquina.
- A partir do formato IF1 é gerado um novo formato denominado IDC que contém uma mistura de nós primitivos e macro-nós.
- O formato IDC é transformado num formato de baixo nível denominado IGF onde existem apenas nós primitivos.
- O otimizador transforma o formato IGF produzindo um código relocável.
- O *Linker* produz três arquivos: .COD contém o programa, .DAT contém os dados de entrada e o .SEG contém uma tabela de segmentos usada na carga do arquivo de programa.

```

export main
function Ack(m,n:integer returns integer)
  if m = 0 then
    n + 1
  elseif n = 0 then
    Ack(m-1,1)
  else
    Ack(m-1,Ack(m,n-1))
  end if
end function

```

Figura 4.11: Programa Ackermann

Integração Binária É um *benchmark* bastante utilizado por apresentar alto grau de recursividade. Efetua o cálculo da área de uma função por integração binária utilizando a regra do trapézio. A função $y = x^2 - 6x - 10$ será utilizada e os parâmetros de entrada entre os quais será integrada são L e R. O código em SISAL é mostrado na figura 4.12.

Multiplicação de Matrizes É um *benchmark* que explora as características da Unidade de Memória de Estruturas. O código SISAL é mostrado na figura 4.13.

A figura 4.14 mostra o processo de compilação de um programa em SISAL. Esse processo é dividido em várias etapas:

- O programa em SISAL é convertido para o formato intermediário IF1, que é uma representação gráfica de SISAL e independe da máquina.
- A partir do formato IF1 é gerado um novo formato denominado IDC que contém uma mistura de nós primitivos e macro-nós.
- O formato IDC é transformado num formato de baixo nível denominado IGF onde existem apenas nós primitivos.
- O otimizador transforma o formato IGF produzindo um código relocável.
- O *Linker* produz três arquivos: .COD contém o programa, .DAT contém os dados de entrada e o .SEG contém uma tabela de segmentos usada na carga do arquivo de programa.

```

export main
function F(x:integer returns integer)
  x * x - 6 * x - 10
end function
function area(L,R,depth:integer returns integer)
  let
    mid := (L+R)/2;
  in
    if depth <= 1 then
      (R-L) * F(mid)
    else
      area(L,mid,depth-1)+area(mid,R,depth-1)
    end if
  end let
end function
function main(A,B,depth:integer returns integer)
  area(A,B,depth)
end function

```

Figura 4.12: Programa Integração Binária

4.9 Resultados

Os resultados do comportamento das redes obtidos pela execução dos *benchmarks* da seção 4.8 serão analisados através dos seguintes parâmetros: T (tempo de execução do *benchmark*) e O (ocupação média da Fila de Fichas). O tempo de execução e a ocupação média indicam a eficiência da rede de interconexão, pois ela pode melhorar ou degradar a performance da máquina conforme esses parâmetros sejam menores ou maiores. Tempo de execução elevado indica que a rede provoca atrasos indesejáveis na transmissão das fichas, e ocupação elevada acarreta maior utilização de memória para armazenamento de fichas que não puderam ser transmitidas devido à ineficiência da rede.

Essas grandezas portanto, refletem o desempenho das redes, e os valores ideais correspondem ao menor tempo e a menor ocupação média. Será considerado também o custo de implementação (C) conforme descrito na seção 4.4. Os resultados serão apresentados para as diversas combinações possíveis dos pares **Chave de Distribuição-Chave de Coleta**.


```

export Main
type Vector = array[real];
type Matrix = array[Vector]
function Fill(N1,N2:integer;R:real returns Matrix)
let
Row := array_fill(1,N2,R)
in array_fill(1,N1,Row)
end let
end function

function matmult(A,B:Matrix;N1,N2,M:integer returns Matrix)
for I in 1,N1
Row :=
for J in 1,N2
El :=
for K in 1,M
R := A[I,K] * B[K,J]
returns value of sum R
end for
returns array of El
end for
returns array of Row
end for
end function

function Main(returns real)
let
M1 := Fill(12,12,0.5);
M2 := Fill(12,12,1.2);
M3 := Matmult(M1,M2,12,12,12)
in M3[12,12]
end let
end function

```

Figura 4.13: Programa Multiplicação de Matrizes

Figura 4.14: Processo de Compilação de Programa em SISAL

4.9.1 Incremento

O resultado para Incremento de 1 a 100 é 5050, que corresponde ao mesmo resultado apresentado pelo simulador no estágio anterior a implementação das novas redes. A tabela 4.1 apresenta os resultados referentes ao tempo de execução, ao tempo normalizado e percentual, custos das redes e ocupação média da Fila de Fichas para o programa Incremento. A tabela 4.2 apresenta os parâmetros T, C, O e M, sendo que os três primeiros indicam em ordem crescente os índices de Tempo, Custo e Ocupação média, isto é, quanto maiores esses parâmetros mais baixo é o desempenho da rede; o parâmetro M corresponde à média dos três anteriores e serve para comparação entre as redes em termos de custo e desempenho. Aquela que apresenta menor média M corresponde ao melhor compromisso de performance e custo para o referido *benchmark*. As curvas das figuras 4.15 e 4.16 mostram a ocupação da Memória de Dados e da Fila de Fichas respectivamente, para a melhor combinação de redes para o programa Incremento, e as curvas das figuras 4.17 e 4.18 mostram as ocupações para a pior combinação.

Tabela 4.1: Resultados para Incremento

Redes	Tempo	Normal.	Perc.	Custo	Ocupação
Cross1-Crossbar	1502	1,3	27,5%	19	0,07
Cross1-Cross2	1178	1,0		40	0,09
Cross1-Delta	1716	1,5	45,7%	19	0
Cross1-Gsn	1715	1,5	45,6%	18	0
Demultiplexador-Crossbar	1715	1,5	45,6%	20	0,07
Demultiplexador-Cross2	1390	1,2	18%	41	0,17
Demultiplexador-Delta	1925	1,6	63,4%	20	0
Demultiplexador-Gsn	1936	1,6	64,3%	19	0,18

Tabela 4.2: Parâmetros para Incremento

Redes	T	C	O	M
Cross1-Crossbar	3	2	2	2,3
Cross1-Cross2	1	4	3	2,7
Cross1-Delta	4	2	1	2,3
Cross1-Gsn	4	1	1	2,0
Demultiplexador-Crossbar	4	3	2	3,0
Demultiplexador-Cross2	2	5	4	3,7
Demultiplexador-Delta	5	3	1	3,0
Demultiplexador-Gsn	5	2	5	4,0

Figura 4.15: Ocupação da Memória de Dados para Incremento (Cross1-Gsn)

Figura 4.16: Ocupação da Fila de Fichas para Incremento (Cross1-Gsn)

Figura 4.17: Ocupação da Memória de Dados para Incremento (Demultiplexador-Gsn)

Figura 4.18: Ocupação da Fila de Fichas para Incremento (Demultiplexador-Gsn)

4.9.2 Ackermann

O resultado obtido para Ackermann(3,3) é 61, o que também corresponde ao resultado obtido anteriormente pelo simulador sem a presença das redes propostas. As tabelas 4.3 e 4.4 apresentam os resultados e os parâmetros para a função de Ackermann. As curvas das figuras 4.19 e 4.20 mostram a ocupação da Memória de Dados e da Fila de Fichas respectivamente, para a melhor combinação de redes para o programa Ackermann, e as curvas das figuras 4.21 e 4.22 mostram as ocupações para a pior combinação.

Tabela 4.3: Resultados para Ackermann

Redes	Tempo	Normal.	Perc.	Custo	Ocupação
Cross1-Crossbar	218202	1,6	58,8%	19	0,18
Cross1-Cross2	137403	1,0		40	0,12
Cross1-Delta	260604	1,9	89,7%	19	0,09
Cross1-Gsn	273400	2,0	99,0%	18	0,00
Demultiplexador-Crossbar	246802	1,8	79,6%	20	0,45
Demultiplexador-Cross2	154802	1,1	12,7%	41	0,23
Demultiplexador-Delta	275605	2,0	100,6%	20	0,00
Demultiplexador-Gsn	284804	2,1	107,3%	19	0,01

Tabela 4.4: Parâmetros para Ackermann

Redes	T	C	O	M
Cross1-Crossbar	3	2	5	3,3
Cross1-Cross2	1	4	4	3,0
Cross1-Delta	5	2	3	3,3
Cross1-Gsn	6	1	1	2,7
Demultiplexador-Crossbar	4	3	7	4,7
Demultiplexador-Cross2	2	5	6	4,3
Demultiplexador-Delta	6	3	1	3,3
Demultiplexador-Gsn	7	2	2	3,7

Figura 4.19: Ocupação da Memória de Dados para Ackermann (Cross1-Gsn)

Figura 4.20: Ocupação da Fila de Fichas para Ackermann (Cross1-Gsn)

Figura 4.21: Ocupação da Memória de Dados para Ackermann (Demultiplexador-Crossbar)

Figura 4.22: Ocupação da Fila de Fichas para Ackermann (Demultiplexador-Crossbar)

4.9.3 Integração Binária

O resultado para Integração Binária no intervalo de 0 a 100 para largura do trapézio igual a 10 é 297650, o que também concorda com o resultado anterior. As tabelas 4.5 e 4.6 apresentam os resultados e os parâmetros para o programa Integração Binária. As curvas das figuras 4.23 e 4.24 mostram a ocupação da Memória de Dados e da Fila de Fichas respectivamente, para a melhor combinação de redes para o programa Integração Binária, e as curvas das figuras 4.25 e 4.26 mostram as ocupações para a pior combinação.

Tabela 4.5: Resultados para Integração Binária

Redes	Tempo	Normal.	Perc.	Custo	Ocupação
Cross1-Crossbar	136001	2,4	142,8%	19	587,29
Cross1-Cross2	57000	1,0	1,8%	40	802,67
Cross1-Delta	116001	2,1	107,1%	19	438,26
Cross1-Gsn	180000	3,2	221,4%	18	534,93
Demultiplexador-Crossbar	136000	2,4	142,8%	20	638,07
Demultiplexador-Cross2	56004	1,0		41	831,91
Demultiplexador-Delta	114004	2,0	103,6%	20	490,25
Demultiplexador-Gsn	184003	3,3	228,6%	19	590,21

Tabela 4.6: Parâmetros para Integração Binária

Redes	T	C	O	M
Cross1-Crossbar	5	2	4	3,7
Cross1-Cross2	2	4	6	4,0
Cross1-Delta	4	2	1	2,3
Cross1-Gsn	6	1	3	3,3
Demultiplexador-Crossbar	5	3	5	4,3
Demultiplexador-Cross2	1	5	7	4,3
Demultiplexador-Delta	3	3	2	2,7
Demultiplexador-Gsn	7	2	4	4,3

Figura 4.23: Ocupação da Memória de Dados para Integração Binária (Cross1-Delta)

Figura 4.24: Ocupação da Fila de Fichas para Integração Binária (Cross1-Delta)

Figura 4.25: Ocupação da Memória de Dados para Integração Binária (Demultiplexador-Gsn)

Figura 4.26: Ocupação da Fila de Fichas para Integração Binária (Demultiplexador-Gsn)

4.9.4 Multiplicação de Matrizes

O resultado obtido para Multiplicação de Matrizes foi 7,2 e confere com o resultado da versão anterior do simulador. As tabelas 4.7 e 4.8 apresentam os resultados e os parâmetros para o programa Multiplicação de Matrizes. As curvas das figuras 4.27 e 4.28 mostram a ocupação da Memória de Dados e da Fila de Fichas respectivamente, para a melhor combinação de redes para o programa Multiplicação de Matrizes, e as curvas das figuras 4.29 e 4.30 mostram as ocupações para a pior combinação.

Tabela 4.7: Resultados para Multiplicação de Matrizes

Redes	Tempo	Normal.	Perc.	Custo	Ocupação
Cross1-Crossbar	89002	1,3	29%	19	1209,24
Cross1-Cross2	69002	1,0		40	1409,87
Cross1-Delta	84539	1,2	22,5%	19	1209,91
Cross1-Gsn	82008	1,2	18,8%	18	1361,09
Demultiplexador-Crossbar	93004	1,3	34,8%	20	1123,95
Demultiplexador-Cross2	70002	1,0	1,4%	41	1725,13
Demultiplexador-Delta	87002	1,3	26,1%	20	1151,94
Demultiplexador-Gsn	84011	1,2	21,8%	19	1176,04

Tabela 4.8: Parâmetros para Multiplicação de Matrizes

Redes	T	C	O	M
Cross1-Crossbar	6	2	4	4,0
Cross1-Cross2	1	4	6	3,7
Cross1-Delta	4	2	4	3,3
Cross1-Gsn	3	1	5	3,0
Demultiplexador-Crossbar	7	3	1	3,7
Demultiplexador-Cross2	2	5	7	4,7
Demultiplexador-Delta	5	3	2	3,3
Demultiplexador-Gsn	4	2	3	3,0

Figura 4.27: Ocupação da Memória de Dados para Multiplicação de Matrizes (Cross1-Gsn)

Figura 4.28: Ocupação da Fila de Fichas para Multiplicação de Matrizes (Cross1-Gsn)

Figura 4.29: Ocupação da Memória de Dados para Multiplicação de Matrizes (Demultiplexador-Cross2)

Figura 4.30: Ocupação da Fila de Fichas para Multiplicação de Matrizes (Demultiplexador-Cross2)

4.10 Conclusões

Os resultados apresentados pelas tabelas 4.1 e 4.2 mostram que para o *benchmark* Incremento a combinação das redes Cross1-Gsn foi a que apresentou a melhor média (2,0). O par Cross1-Gsn apresentou a melhor relação entre custo e desempenho na execução do programa Incremento. O tempo de execução ficou pouco acima da média, contudo, a ocupação média da Fila de Fichas corresponde ao valor considerado ideal assim como o custo que é o mais baixo entre todas as combinações.

Apesar do par Demultiplexador-Gsn apresentar um custo próximo do valor mais baixo, seus índices de tempo e ocupação média são os mais elevados entre todas as combinações, o que torna esse par a pior combinação para a execução do *benchmark* Incremento.

Para o *benchmark* Ackermann os resultados de desempenho das redes são mostrados pelas tabelas 4.3 e 4.4. Para esse *benchmark* a combinação que apresentou o melhor compromisso entre custo e desempenho foi novamente o par Cross1-Gsn (2,7). Apesar de executar o programa com um tempo razoável, pouco acima da média, os parâmetros de custo e ocupação média possuem valores ideais, o que leva a relação de custo e desempenho desse par ao melhor valor entre todas as combinações.

O pior caso para o *benchmark* Ackermann foi apresentado pelo par Demultiplexador-Crossbar que apresentou a mais alta relação de custo e desempenho (4,7). Esse par executou o programa num tempo médio, mas além do custo estar pouco acima da média sua ocupação média foi a maior entre todas, e a combinação desses índices levou esse par a apresentar a pior média.

As tabelas 4.5 e 4.6 mostram os resultados dos desempenhos das combinações de redes na execução do *benchmark* Integração Binária. A melhor relação de custo e desempenho foi apresentada pelo par Cross1-Delta (2,3) cujo tempo de execução ficou pouco acima da média. O custo dessa combinação é considerado bom, pois, está próximo do custo mais baixo e a ocupação média é a ideal.

As piores relações de custo e desempenho (4,3) foram apresentadas pelas seguintes combinações:

- Demultiplexador-Crossbar
- Demultiplexador-Cross2
- Demultiplexador-Gsn

O par Demultiplexador-Crossbar executou num tempo relativamente alto, além de apresentar um custo pouco acima da média e uma ocupação média também relativamente alta.

O par Demultiplexador-Cross2 apesar de executar no menor tempo entre todas as combinações, apresentou o maior custo e a maior ocupação média.

O pior tempo de execução foi o do par Demultiplexador-Gsn, que, no entanto, apresentou um bom índice de custo, próximo ao mais baixo, e a ocupação média ficou pouco acima da média.

Os resultados de desempenho das combinações de redes para Multiplicação de Matrizes mostrados pelas tabelas 4.7 e 4.8 indicam as combinações Cross1-Gsn e Demultiplexador-Gsn como as melhores, e o par Demultiplexador-Cross2 como o pior caso.

O par Cross1-Gsn (3,0) executou o programa num tempo considerado bom, pouco abaixo da média, e apresentou o custo mais baixo entre as combinações, mas sua ocupação média foi relativamente alta.

Já o par Demultiplexador-Gsn (3,0) apesar de executar num tempo pouco acima da média apresentou um custo próximo do ideal e teve uma ocupação média considerada boa, pouco abaixo da média.

O par com a pior relação entre custo e desempenho foi Demultiplexador-Cross2 (4,7) que apesar de executar num tempo muito bom, próximo do mais baixo entre as combinações, apresentou o maior custo e a maior ocupação média entre todos os pares.

Os quatro *benchmarks* utilizados exploram características diferentes da arquitetura, por isso proporcionam uma visão geral do comportamento da máquina. Devido a característica de propósito geral apresentada pela arquitetura, o desempenho da máquina deve ser considerado em relação ao maior número de aplicações, baseado nesse fato podemos concluir que a combinação de redes Cross1-Gsn apresenta-se a mais adequada à arquitetura no estágio atual de implementação do simulador, visto que esse par de redes obteve a melhor média para três dos quatro *benchmarks* utilizados.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Considerações Finais

Esse trabalho possibilitou maior entendimento do mecanismo de comunicações em arquiteturas paralelas, no modelo a fluxo de dados. Foram aprendidos os conceitos básicos de redes de interconexão, o que habilita o desenvolvimento de novos trabalhos correlatos.

Os resultados apresentados permitiram concluir o primeiro estudo de desempenho de redes de interconexão para a arquitetura Proto-WOLF utilizando o simulador SAW. Conclui-se que a combinação das redes Cross1-Gsn é a mais adequada à arquitetura Proto-WOLF, pois, apesar dos pares Demultiplexador-Gsn e Cross1-Delta terem obtido também a melhor média em um dos *benchmarks*, para a maioria dos programas executados os melhores índices foram apresentados pelo par Cross1-Gsn.

Como próxima etapa de trabalho será realizado um estudo sobre redes de interconexão explorando características da arquitetura Proto-WOLF ainda não avaliadas, tais como características das redes na presença da Unidade de Controle de Paralelismo e da Unidade de Processamento Vetorial.

O desenvolvimento desse trabalho irá abranger as seguintes etapas:

- Avaliação do comportamento das redes na presença da Unidade de Processamento Vetorial através do uso de vários *benchmarks* que exploram as características de

uma unidade para tratamento de estruturas de dados. Espera-se que essa unidade diminua o *overhead* nos anéis (MI/UF), visto que os dados estruturados não precisarão circular pelo *pipeline* para serem processados.

- Avaliação do comportamento das redes de conexão na presença da Unidade de Controle de Paralelismo (*Throttle*), que é um mecanismo eficiente para controlar e evitar exposição de paralelismo excessivo. Através do controle de paralelismo pode-se diminuir o uso excessivo de memória.
- Avaliação do comportamento das redes introduzindo a redução da Chave de Emparelhamento proposta por [Mag92], que pode eliminar redundâncias em diversos campos das fichas, diminuindo assim, o espaço necessário para emparelhamento.
- Avaliação do impacto causado pela variação do número de processadores (variação do número de anéis).
- Identificar qual o modelo de rede mais apropriado em todas essas situações. Através da análise do comportamento das redes de interconexão na presença de cada uma dessas unidades pode-se chegar à definição das redes mais adequadas à arquitetura Proto-WOLF.

Com esse trabalho será possível então, conhecer todas as características introduzidas pela arquitetura Proto-WOLF, além de estudar alguns dos problemas mais comumente encontrados em fluxo de dados como granularidade variável, processamento de estruturas de dados e excesso de paralelismo.

Bibliografia

- [AG89] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 390 Bridge Parkway, Redwood City, CA 94065, 1989.
- [AN90] Arvind and R. S. Nikhil. Executing a Program on the MIT Tagged-Token Dataflow Architecture. *IEEE Transaction on Computers*, 39(3):300–318, March 1990.
- [AP89] Seth Abraham and Krishnan Padmanabhan. Performance of the Direct Binary n-Cube Network for Multiprocessors. *IEEE Transactions on Computers*, 38(7):1000–1011, July 1989.
- [BA83] L. N. Bhuyan and D. P. Agrawal. Design and Performance of Generalized Interconnection Networks. *IEEE Transactions on Computers*, C-32(12):1081–1090, December 1983.
- [Bhu87] Laxmi N. Bhuyan. Interconnection Networks for Parallel and Distributed Processing. *IEEE Computer*, pages 9–12, June 1987.
- [BT88] A. P. W. Bohn and Y. M. Teo. The Heterogeneous Multi-Ring Dataflow Machine Simulator, 1988.
- [Cav92] Marcos Antônio Cavenaghi. Implementação de um Simulador para a Arquitetura Fluxo de Dados WOLF. Dissertação de Mestrado, Departamento de Física e Ciências dos Materiais, Instituto de Física e Química de São Carlos, Universidade de São Paulo, Av. Dr. Carlos Botelho, 1465, 1992.

- [CDJ84] Fred B. Chambers, David A. Duce, and Gillian P. Jones, editors. *Distributed Computing*. 20. Academic Press, Inc., Orlando, Florida 32887, United States of America, first edition, 1984.
- [CPHL94] W. J. Camp, S. J. Plimpton, B. A. Hendrickson, and R. W. Leland. Massively Parallel Methods for Engineering and Science Problems. *Communications of the ACM*, 37(4):31–41, April 1994.
- [Dun90] Ralph Duncan. A Survey of Parallel Computer Architectures. *IEEE Computer*, 23(2):5–16, February 1990.
- [Fen81] T. Y. Feng. A Survey of Interconnection Networks. *IEEE Computer*, pages 12–27, December 1981.
- [Fly72] M. J. Flynn. Some Computer Organization and their Effectiveness. *IEEE Transactions on Computers*, C-21:948–960, 1972.
- [GB90] Dipak Ghosal and L. N. Bhuyan. Performance Evaluation of a Dataflow Architecture. *IEEE Transactions on Computers*, 39(5):615–627, May 1990.
- [GDHH89] V. G. Grafe, G. S. Davidson, J. E. Hoch, and V. P. Holmes. The Epsilon Dataflow Processor. In *Proc. 16th International Symposium on Computer Architecture*, 1989.
- [GH90] V. G. Grafe and J. E. Hoch. The Epsilon-2 Multiprocessor System. *Journal of Parallel and Distributed Computing*, 10(4):309–318, December 1990.
- [GKB87] J. R. Gurd, C. C. Kirkham, and W. Böhm. *The Manchester Dataflow Computing System*, volume 1 of *Special Topics in Supercomputing*. North-Holland, January 1987.
- [GKW85] J. R. Gurd, C. C. Kirkham, and I. Watson. The Manchester Prototype Dataflow Computer. *Communications of the ACM*, 28(1):34–52, January 1985.

- [GL73] L. Rodney Goke and G. J. Lipovski. Banyan Networks for Partitioning Multiprocessor Systems. In *The Proceedings of the Annual Symposium on Computer Architecture*, 1973.
- [GNC92] Álvaro Garcia Neto and Marcos Antônio Cavenaghi. Primeiros Resultados da Proto-Arquitetura a Fluxo de Dados Wolf. In *IV Simpósio Brasileiro de Arquitetura de Computadores – Processamento de Alto Desempenho*, 1992.
- [GNR92] Álvaro Garcia Neto and C. A. Ruggiero. The Proto-Architecture of WOLF, a Dataflow Supercomputer. In *XVIII Conferencia Latinoamericana de Informatica*, 1992.
- [GWG80] J. R. Gurd, I. Watson, and J. R. W. Glauert. A Multilayered Dataflow Computer Architecture. Internal Report, Department of Computer Science, University of Manchester, March 1980.
- [GWK85] J. R. Gurd, I. Watson, and C. C. Kirkham. The Manchester Prototype Dataflow Computer. *Communications of the ACM*, 28(1), January 1985.
- [HB84] Kai Hwang and Fayé A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Series in Computer Organization and Architecture. McGraw-Hill, Inc., United States of America, first edition, 1984.
- [HJ88] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2*. IOP, second edition, 1988.
- [HJA90] A. Harissis, C. Jam, and A. Ambler. Analysis and Design of Circuit Switching Interconnection Networks Using 4x4 Nodes. In G. Goos and J. Hartmanis, editors, *CONPAR 90 - VAPP IV Joint International Conference on Vector and Parallel Processing*, 1990.
- [HNS⁺87] K. Hiraki, K. Nishida, S. Sekiguchi, T. Shimada, and T. Yuba. The SIGMA-1 Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems. *Journal of Information Processing*, 10(4):219–226, 1987.
- [HPV83] C. Halatsis, G. Philokyprou, and C. Verkerk. Prospects of Data Flow Computing in High Energy Physics, September 1983.

- [HT93] Daniel W. Hillis and Lewis W. Tucker. The CM-5 Connection Machine: A Scalable Supercomputer. *Communications of the ACM*, 36(11):31–40, November 1993.
- [HY+88] Jayantha Herath, Yoshinori Yamaguchi, et al. Dataflow Computing Models, Languages, and Machines for Intelligence Computations. *IEEE Transactions on Software Engineering*, 14(12):1805–1828, December 1988.
- [IT89] Roland N. Ibbett and N. P. Topham. *Architecture of High Performance Computers*, volume 2 of *Macmillan Computer Science Series*. Macmillan Education LTD., London, England, first edition, 1989.
- [Law75] Duncan H. Lawrie. Acces and Alignment of Data in an Array Processor. *IEEE Transactions on Computers*, C-24(12):1145–1155, December 1975.
- [Mac85] J. R. MacGraw. *SISAL: Streams and Iteration in Single Assignment Language, Language Reference Manual, Version 1.2*. Lawrence Livermore National Laboratory, March 1985.
- [Mag92] Patrícia Magna. Redução dos Bits de Endereçamento da Máquina de Fluxo de Dados de Manchester. Dissertação de Mestrado, Departamento de Física e Ciências dos Materiais, Instituto de Física e Química de São Carlos, Universidade de São Paulo, Av. Dr. Carlos Botelho, 1465, 1992.
- [MB94] Qutaibah M. Malluhi and Magdy A. Bayoumi. The Hierarchical Hypercube: A New Interconnection Topology for Massively Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):17–30, January 1994.
- [Pap88] Gregory Michael Papadopoulos. *Implementation of a General Purpose Dataflow Multiprocessor*. PhD thesis, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge MA 02139, August 1988.
- [PB92] Dick Pountain and John Bryan. All Systems Go. *BYTE*, 17(8):112–136, August 1992.
- [PI77] Marshall C. Pease III. The Indirect Binary n-Cube Microprocessor Array. *IEEE Transactions on Computers*, C-26(5):458–473, May 1977.

- [Qui87] Michael Jay Quinn. *Designing Efficient Algorithms for Parallel Computers*. Series in Supercomputing and Artificial Intelligence. McGraw-Hill, United States of America, 1987.
- [RO93] Gowri Ramanathan and Joel Oren. Survey of Commercial Parallel Machines. *Computer Architecture News*, 21(3):13–33, June 1993.
- [Rug87] C. A. Ruggiero. Throttle Mechanisms for the Manchester Dataflow Machine. Technical Report Series UMCS-87-8-1, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, England, July 1987.
- [Rya91] Bob Ryan. Multiprocessor Surf's Up. *BYTE*, pages 199–206, June 1991.
- [SG94] Steven L. Scott and James R. Goodman. The Impact of Pipelined Channels on k-ary n-Cube Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):2–16, January 1994.
- [Sie85] Howard Jay Siegel. *Interconnection Networks for Large-Scale Parallel Processing*. The Lexington Books Series in Computer Science. Lexington Books, United States of America, first edition, 1985.
- [Ski88] David B. Skillicorn. A Taxonomy for Computer Architectures. *Computer*, 21(11):46–57, November 1988.
- [SKY93] Shuichi Sakai, Yuetsu Kodama, and Yoshinori Yamaguchi. Design and implementation of a circular omega network in the EM-4. *Parallel Computing*, 19(2):125–142, February 1993.
- [Wat84] Ian Watson. *Architecture and Performance*, chapter 2. Number 20 in APIC Studies in Data Processing. Academic Press, Inc., Orlando, Florida 32887, United States of America, first edition, 1984.
- [WHJ90] Stephen A. Ward and Robert H. Halstead Jr. *Computation Structures*. The MIT Electrical Engineering and Computer Science Series. The MIT Press, Cambridge, Massachusetts, 1990.

- [YSY⁺90] T. Yuba, T. Shimada, Y. Yamaguchi, K. Hiraki, and S. Sakai. Dataflow Computer Development in Japan. *ACM*, pages 140–147, 1990.
- [Yue88] C. K. Yuen. *Essential Concepts of Computer Architecture for programmers and computer users*. Addison-Wesley Publishing Company, Singapore, first edition, 1988.
- [Zha91] Xiaodong Zhang. System Effects of Interprocessor Communications Latency in Multicomputers. *IEEE Micro*, 11(2):12–15, 52–55, April 1991.