JONAS GOMES FILHO

**MAPEAMENTO E POSICIONAMENTO DE MÓDULOS PROCESSANTES EM
SISTEMAS DINAMICAMENTE RECONFIGURÁVEIS BASEADOS EM REDES
INTRACHIP**

São Paulo

2015

JONAS GOMES FILHO

**MAPEAMENTO E POSICIONAMENTO DE MÓDULOS PROCESSANTES EM
SISTEMAS DINAMICAMENTE RECONFIGURÁVEIS BASEADOS EM REDES
INTRACHIP**

Tese apresentada à Escola Politécnica
da Universidade de São Paulo para a
obtenção do Título de Doutor em
Ciências.

São Paulo

2015

JONAS GOMES FILHO

**MAPEAMENTO E POSICIONAMENTO DE MÓDULOS PROCESSANTES EM
SISTEMAS DINAMICAMENTE RECONFIGURÁVEIS BASEADOS EM REDES
INTRACHIP**

Tese apresentada à Escola Politécnica
da Universidade de São Paulo para a
obtenção do Título de Doutor em
Ciências.

Área de concentração:
Engenharia Elétrica; Microeletrônica

Orientador:

Prof. Dr. Wang Jiang Chau

São Paulo

2015

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de fevereiro de 2015.

Assinatura do autor _____

Assinatura do orientador _____

Catálogo-na-publicação

Gomes Filho, Jonas

Mapeamento e posicionamento de módulos processantes em sistemas dinamicamente reconfiguráveis baseados em redes intrachip / J. Gomes Filho. – versão corr. -- São Paulo, 2015. 130 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Microeletrônica 2.Redes intrachip 3.Arquitetura reconfigurável 4.Mapeamento I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

DEDICATÓRIA

Aos meus pais, à minha irmã e à minha amada esposa, Flávia.

AGRADECIMENTOS

Ao prof. Dr. Wang Jiang Chau, um excelente orientador, que se tornou também um amigo. Deu-me a oportunidade de realizar o mestrado e doutorado e me fez crescer profissionalmente. Com muita paciência e dedicação me mostrou o caminho correto a seguir, sempre com discussões de altíssimo nível.

Aos meus professores do IFSP, Ricardo Pires, Alexandre Brincalepe Campo, Omar Alves e Luiz Carlos Moreira por terem acreditado no meu potencial e por me recomendarem para o curso de mestrado na POLI-USP. Ao professor Sérgio Tosin, que me apresentou ao mundo da eletrônica. Ao Professor Rolando Antonio Thimmig que além de ser grande amigo, me ajudou com ricas discussões em várias questões sobre o doutorado.

Ao Dr. Marius Strum e aos demais colegas do GSEIS, Mário Raffo, Raul Acosta, Joel Muñoz, Martha Johanna Sepúlveda, Christopher Castillo, Gustavo Patino e Carlos Castro Marquez. Também em memória de Edgar Romero, que infelizmente nos deixou, porém continua conosco em nossas memórias e corações.

A meus pais Jonas Gomes e Cristiane Gomes e minha irmã, Ana Carolina Gomes, que me apoiaram em cada decisão e ajudaram em cada passo de minha vida, que culminaram no meu casamento, e agora neste doutorado. Tudo que conquistei em minha vida devo a eles.

Aos meus amigos, Felipe Gambier Souza, José Antonio de Melo Neto, Raniel Victor Machado, Fellipe Cerqueira, Raí Soares e Leandro César Bezerra que estão juntos comigo em toda minha caminhada e me ajudaram de diversas maneiras.

À Francisca, Miguel, Dálete e Priscila, que se tornaram minha segunda família e me deram muita força, apoio e alegria durante estes anos.

À minha esposa Flávia, que trouxe muita alegria e beleza para minha vida, transformando-a completamente. Me apoiou nos momentos difíceis, sempre me dando a força necessária para enfrentar os problemas. Esta tese é dedicada a ela, assim como todas as minhas conquistas.

Ao CNPq, pelo apoio financeiro para a realização deste trabalho.

"Esta é a marca de um homem realmente admirável: firmeza e perseverança quando se vê diante de problemas."

Ludwig van Beethoven

RESUMO

Sistemas Dinamicamente Reconfiguráveis (SDRs) tem sido aceitos como alternativa importante para diminuir os custos de circuitos digitais. Porém, eles adicionam novas dimensões no projeto de Sistemas sobre Silício (System-On-Chip, SoC). Apesar de novas metodologias terem sido propostas por fabricantes de FPGA para lidar com a complexidade deste tipo de circuitos, as soluções ainda são muito específicas.

Considerando-se que o uso de recursos de comunicação robustos em SoCs complexos atuais é generalizado, os meios de comunicação estruturados, como rede Intrachips (Network-On-Chip, NoCs), foram incluídas em sistemas dinamicamente reconfiguráveis, gerando-se arquiteturas de SDRs baseadas em NoCs, ou de SDR-NoCs.

Arquiteturas de SDR-NoCs podem ser simples ou complexas. As arquiteturas de SDR-NoCs simples são aquelas com topologias regulares e diretas e Módulos Processantes (MPs) homogêneos. As arquiteturas de SDR-NoCs complexas são aquelas com topologias irregulares e indiretas com MPs heterogêneos.

O mapeamento é a fase no fluxo de projeto do SoC que visa encontrar a melhor localização das unidades de processamento da aplicação junto à topologia da NoC, de tal forma que as métricas de interesse podem ser otimizadas. O problema do posicionamento lida com a alocação otimizada de recursos (*cores*) dentro do dispositivo reconfigurável.

No mapeamento de SDR-NoCs, a capacidade de reconfiguração no tempo acrescenta uma nova dimensão ao problema de mapeamento, uma vez que diferentes *cores* são atribuídos ao mesmo roteador, mas estão presentes no dispositivo em momentos distintos. Para arquiteturas de SDR-NoCs complexas, o problema de mapeamento está fortemente associado ao problema do posicionamento e convém tratá-los em conjunto. Até o presente momento, o problema de mapeamento e posicionamento para SDR-NoCs não tem sido tratados adequadamente.

Neste trabalho são apresentadas soluções para o mapeamento e/ou posicionamento de MPs para arquiteturas SDR-NoCs tanto simples quanto

complexas. Primeiramente, uma estratégia de mapeamento é proposta para arquiteturas simples, de uma forma que torna possível a utilização de estratégias de mapeamento clássicas anteriores (sem reconfiguração) para SDRs. Os resultados mostram a redução de até 38%, no atraso médio da NoC e de até 41% de economia de energia comparando a melhor solução com a média de soluções aleatórias.

Em uma segunda fase, o problema de mapeamento e posicionamento são tratados em conjunto para arquiteturas SDR-NoCs complexas: uma formalização do problema é proposta e um algoritmo exato, semi-exaustivo, é implementado e utilizado para a sua análise. Devido à alta complexidade do problema, um segundo algoritmo genético (*Genetic Algorithm*, GA) foi implementado para que casos maiores possam ser resolvidos. Vários tipos de *crossover* e metodologias de GAs são comparadas para se obter a melhor solução. Os resultados mostram que a melhor solução GA obteve, em média, custos de comunicação com 4% de penalidade quando comparada com a melhor solução, sendo que o algoritmo apresenta bons tempos de execução.

Palavras-chave: Redes Intrachip. Sistemas Dinamicamente Reconfiguráveis. Mapeamento. Posicionamento.

ABSTRACT

Dynamic Reconfigurable Systems (DRSs) have been accepted as an important alternative for lowering costs of digital circuits. However, they add new dimensions to the system-on-chip (SoC) design space. Although new methodologies have been proposed by Field Programmable Gate Arrays (FPGAs) manufacturers to deal with the increased design complexity in this class of circuits, solutions to the algorithmic and block level design are still very ad-hoc.

Considering the generalized use of robust communication resources in current complex SoCs, structured communication means, as network-on-chips (NoCs), have been included in dynamic reconfigurable systems generating DRSs based on NoCs, or DRS-NoCs, under different architectures.

DRS-NoC architectures can be simple or complex. Simple DRS-NoCs architectures refer to regular and direct NoC topologies, with homogeneous Processing Modules (PMs). Complex DRS-NoCs architectures refer to irregular and undirected NoC topologies, with heterogeneous MPs.

Mapping is the step in the SoC design flow which aims to find the best topological location for the application processing units onto the NoC topology, such that the metrics of interest can be greatly optimized. The placement problem deals with the optimized allocation of resources (cores) inside the reconfigurable device.

In DRS-NoCs mapping, the on-going reconfiguration capability adds a new dimension to the mapping problem, since different cores are assigned to the same router, but being present in the logic fabric in separate moments. Furthermore, in complex DRS-NoC architectures the mapping problem is strongly associated with the placement one, and they should be dealt concurrently. To the date, the mapping and placement problems have not been properly addressed for those kind of architectures.

In this work solutions are presented for hardware core placement and/or mapping for both simple and complex DRS-NoC architectures. Firstly, a mapping strategy is proposed for simple architectures, in a way that makes it possible to use previous classic mapping strategies (without reconfiguration) for DRSs. Results show

reductions up to 38% on the average NoC delay and up to 41% of energy saving when comparing the best solution with average random solutions.

In the second phase, the mapping and placement problems are dealt concurrently for DRS-NoC complex architectures: the problem formalization is proposed and for its analysis, an exact, and semi-exhaustive, algorithm is implemented and applied. Due to the high complexity associated to the problem, an Genetic Algorithm (GA) was implemented to deal with larger cases. Several GAs crossovers and methodologies are compared for obtaining the best solution. Results show that best GA solution obtained, in average, communication costs with 4% of penalty when compared with best solution. In addition, the algorithm presents low execution times.

Keywords: Networks-on-Chip. Dynamic Reconfigurable Systems. Mapping. Placement.

LISTA DE ILUSTRAÇÕES

Figura 1 - Topologias: (a) Grelha 2D; (b) Toróide 2D; (c) Cubo 3D; (d) Cubo 4D; (e) Totalmente Conectada.	3
Figura 2 - Particionamento temporal em projetos de DRFPGAs.	5
Figura 3 - Arquitetura DyNoC.	7
Figura 4 - A importância da posição dos elementos processantes em uma NoC.	9
Figura 5 - A importância do posicionamento em um SDR implementado em NoC. ...	10
Figura 6 - Extensão do mapeamento para arquiteturas mais complexas.	11
Figura 7 - Exemplo de posicionamento.	17
Figura 8 - Exemplo de posicionamento ineficiente.	17
Figura 9 - Rede intrachip 3x3.	18
Figura 10 - Estrutura genérica de um roteador.	19
Figura 11 - (a) Topologia irregular (b) Topologia indireta.	20
Figura 12 - NoC com MPs (a) heterogênea (b) homogênea.	22
Figura 13 - Exemplos de grafos de aplicação (a); da NoC (b); e da NoC mapeada (c).	24
Figura 14 - Reconfiguração Dinâmica Parcial.	27
Figura 15 - Reconfiguração Estática (a) X Reconfiguração Dinâmica (b)	28
Figura 16 - Exemplo de particionamento de tarefas.	30
Figura 17 - Exemplo de escalonamento.	31
Figura 18 - Esquema de particionamento para uma arquitetura SDR-NoC.	32
Figura 19 - Arquiteturas (a) SDR-NoC simples. (b) SDR-NoC complexa.	33
Figura 20 - (a) Arquitetura DyNoC. (b) Estrutura do roteador.	34
Figura 21 - Alocação de blocos reconfiguráveis na DyNoC.	35
Figura 22 - Roteamento surrounding XY.	36
Figura 23 - Estrutura de uma unidade de comunicação.	37
Figura 24 - Vizinhança entre UCs e UCgs.	38
Figura 25 - Etapas de implementação da CuNoC.	38
Figura 26 - Bloqueio de roteamento na CuNoC.	39
Figura 27 - Árvore de busca para o PCV.	41
Figura 28 - <i>Crossover</i> simples.	47

Figura 29 - Exemplo de mutação.	48
Figura 30 - Exemplo de três indivíduos durante a evolução do algoritmo.	48
Figura 31 - Exemplo de <i>crossover</i> durante a execução do algoritmo.....	49
Figura 32 - Exemplo de mutação durante a execução do algoritmo.	49
Figura 33 - Tarefas específicas do algoritmo do DynoPlace. (RAFFO, 2010).....	54
Figura 34 - Grafos de comunicação de (a) MPEG-2 (b) H.263 (c) MPEG-4.....	62
Figura 35 - Mapeamento estático para (a) e (b) MPEG-2 (c) e (d) H.263 (e) e (f) MPEG-4.....	62
Figura 36 - Processo de mapeamento (BERETTA et al, 2011).....	63
Figura 37 - Mapeamento dinâmico para (a) MPEG-2 e H.263 (b) MPEG-4.	64
Figura 38 - Exemplo de fusão de MPs.	67
Figura 39 - Exemplo de fusão de MPs com conflito de arcos.....	68
Figura 40 - Aplicação VOPD (a) Original (b) Fundida	71
Figura 41 - Aplicação MP4 (a) Original (b) Fundida	71
Figura 42 - Aplicação MMS (a) Original (b) Fundida	72
Figura 43 - Árvore de busca para uma NoC 2x2.	74
Figura 44 - Pseudocódigo do algoritmo B&B.....	77
Figura 45 - Automação para simulação de mapeamentos aleatórios.....	81
Figura 46 - Resultado do mapeamento da aplicação VOPD.	82
Figura 47 - Resultado do mapeamento da aplicação MP4.....	82
Figura 48 - Resultado do mapeamento da aplicação MMS.....	83
Figura 49 - Modelo de SDR baseado em NoC com tamanho 7x7.....	87
Figura 50 - Definição de região.	89
Figura 51 - Uma região $\Phi(ci)$ e uma região ocupada $Ro(ci)$	91
Figura 52 - EAPCG de uma aplicação sintética.	92
Figura 53 - Mapeamento mínimo para: (a) cenário 1, (b) cenário 2, (c) cenário 3. ...	93
Figura 54 - Exemplo de codificação de mapeamento.	95
Figura 55 - Pseudocódigo do algoritmo genético.	99
Figura 56 - Ordered Crossover.....	102
Figura 57 - Non-wrapping ordered crossover.....	103
Figura 58 - Partially matched crossover.	104
Figura 59 - Média do custo vs gerações: (a) Alpha (b) Beta (c) Gamma	107

LISTA DE TABELAS

Tabela 1 - Largura x altura de peças de posicionamento.....	17
Tabela 2 - Distância entre cidades do PCV.....	40
Tabela 3 - Codificação binária para GA	44
Tabela 4 - Codificação de um cromossomo.	45
Tabela 5 - Decodificação de um cromossomo inválido.	45
Tabela 6 - Comparação dos trabalhos correlatos.....	65
Tabela 7 - Tempos de execução do algoritmo B&B	78
Tabela 8 - Resultados de simulação para NoC ideal.	84
Tabela 9 - Resultados de simulação para NoC real.	85
Tabela 10 - Número de MPs nas aplicações sintéticas.....	94
Tabela 11 - Resultados do algoritmo semi-exaustivo.....	97
Tabela 12 - Custos obtidos pelo GA.....	108
Tabela 13 - Comparação dos resultados com os melhores mapeamentos.....	110
Tabela 14 - Tempos de execução das simulações em segundos.	111
Tabela 15 - Tamanho dos <i>cores</i> na aplicação Alpha.	125
Tabela 16 - Tabela de comunicação da aplicação Alpha.	126
Tabela 17 - Tamanho dos <i>cores</i> na aplicação Beta.	127
Tabela 18 - Tabela de comunicação da aplicação Beta.....	128
Tabela 19 - Tamanho dos <i>cores</i> na aplicação Gamma.	129
Tabela 20 - Tabela de comunicação da aplicação Gamma.....	130

LISTA DE ABREVIATURAS E SIGLAS

AGA	<i>Adaptative Genetic Algorithm</i>
APCG	<i>Application characterization graph</i>
ASIC	<i>Application Specific Integrated Circuit</i>
B&B	<i>Branch and Bound</i>
BF	<i>Best Fit</i>
BN	<i>Best Neighbor</i>
CAV	Compactação de Área Vazia
CVV	Compactação de Volume Vazio
CT	<i>Cutting Tree</i>
CuNoC	<i>Comunication unit Network on Chip</i>
DRFPGA	<i>Dynamically Reconfigurable Field Programmable Gate Arrays</i>
DyNoC	<i>Dynamic Network-on-Chip</i>
EAPCG	<i>Extended Application characterization graph</i>
EP	Elemento Processante
EPROM	<i>Erasable Programmable Read Only Memory</i>
FCFS	<i>First Come First Served</i>
FD	<i>Force Directed</i>
FIFO	<i>First In First Out</i>
FLIT	<i>Flow control unit</i>
FPGA	<i>Field Programmable Gate Array</i>
GA	<i>Genetic Algorithm</i>
GI	<i>Greedy Incremental</i>

GPP	<i>General-Purpose Processors</i>
GSEIS	Grupo de Sistemas Eletrônicos Integrados e Software Aplicado
IP	<i>Intellectual Property Core</i>
LBC	<i>Lower Bound Cost</i>
LCF	<i>Largest Communication First</i>
LRS	<i>Least Recently Served</i>
MACL	<i>Minimum Average Channel Load</i>
MER	<i>Maximum Empty Rectangle</i>
MMCL	<i>Minimum Maximum Channel Load</i>
MMS	<i>Multiple Multimedia System</i>
MP	Módulo Processante
MPSoC	<i>Multi-Processor System-on-Chip</i>
NoC	<i>Network-on-Chip</i>
Noxim	<i>Network-on-chip Simulator</i>
NWOX	<i>Non-Wrapping Ordered Crossover</i>
OX	<i>Ordered Crossover</i>
PCV	Problema do Caixeiro Viajante
PL	<i>Path Load</i>
PMX	<i>Partially Matched Crossover</i>
R3TOS	<i>Reliable Reconfigurable Real-Time Operating System</i>
ROG	<i>Random Offspring Generation</i>
RTL	<i>Register-Transfer level</i>
RTR	<i>Run-Time Reconfiguration</i>
SA	<i>Simulated Annealing</i>
SAF	<i>Store-And-Forward</i>
SAT	<i>Satisfiability Boolean Algorithm</i>

SDR	Sistema Dinamicamente Reconfigurável
SDR-NoC	Sistema Dinamicamente Reconfigurável baseado em NoC
SoC	<i>System on Chip</i>
SPR	Sistema Parcialmente Reconfigurável
SXY	Algoritmo de roteamento Surrounding XY
TLM	<i>Transaction-Level Modeling</i>
TGFF	<i>Task Graphs for Free</i>
TS	<i>Tabu Search</i>
UBC	<i>Upper Bound Cost</i>
UC	Unidade de Comunicação
UCg	Unidade de Comunicação Gerenciadora de Caminho
VCT	<i>Virtual-Cut-Trough</i>
VOPD	<i>Video Object Plane Decoder</i>
WF	<i>Worst Fit</i>
XY	Algoritmo de roteamento XY

SUMÁRIO

1. INTRODUÇÃO.....	1
1.1. CONTEXTUALIZAÇÃO	1
1.1.1. REDES INTRACHIP	1
1.1.2. FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)	3
1.1.3. SISTEMAS DINAMICAMENTE RECONFIGURÁVEIS (SDRs).....	6
1.2. MOTIVAÇÃO	8
1.3. JUSTIFICATIVA	12
1.4. OBJETIVOS.....	13
1.4.1. OBJETIVOS ESPECÍFICOS NO ASPECTO TEÓRICO.....	13
1.4.2. OBJETIVOS ESPECÍFICOS NO MAPEAMENTO DE ARQUITETURAS SDR-NOCS SIMPLES	13
1.4.3. OBJETIVOS ESPECÍFICOS NO MAPEAMENTO DE ARQUITETURAS SDR-NOCS COMPLEXAS.....	14
1.5. CONTRIBUIÇÕES.....	15
1.6. ORGANIZAÇÃO DO TEXTO	15
2. BASES TEÓRICAS	16
2.1. O PROBLEMA GERAL DE POSICIONAMENTO	16
2.2. REDES INTRACHIP	18
2.2.1. HOMOGENEIDADE DOS MÓDULOS PROCESSANTES.....	22
2.3. MAPEAMENTO DE TAREFAS EM REDES INTRACHIP.....	23
2.4. RECONFIGURAÇÃO DINÂMICA	26
2.5. PARTICIONAMENTO E ESCALONAMENTO DE PROCESSOS EM SDRs.....	30
2.6. SDRs BASEADOS EM REDES INTRACHIP	32
2.6.1. ARQUITETURA DyNOC	34
2.6.2. ARQUITETURA CuNOC	36
2.7. ALGORITMOS	40
2.7.1. BRANCH AND BOUND	40
2.7.2. ALGORITMOS GENÉTICOS	43
2.7.2.1. Codificação e geração de população aleatória	44
2.7.2.2. Seleção dos indivíduos mais aptos.....	46
2.7.2.3. Crossover e Mutação	46

2.7.2.4. Evolução do algoritmo	48
2.7.2.5. Técnicas de variação da população.....	50
2.7.2.6. Benefícios e limitações.....	52
3. TRABALHOS CORRELATOS	53
3.1. POSICIONAMENTO DE MÓDULOS EM SDRs	53
3.2. MAPEAMENTO DE TAREFAS EM NOCs	55
3.2.1. MAPEAMENTO ESTÁTICO	55
3.2.2. MAPEAMENTO DINÂMICO	57
3.3. MAPEAMENTO PARA SDRs BASEADOS EM NOCs	60
3.4. COMPARAÇÃO COM OS TRABALHOS DE MAPEAMENTO CORRELATOS	64
4. MAPEAMENTO PARA ARQUITETURAS SDR-NOc SIMPLES.....	66
4.1. ESTRATÉGIA DE MAPEAMENTO PARA ARQUITETURA SDR-NOc SIMPLES	66
4.1.1. FUSÃO DE MÓDULOS COM ORIGENS (DESTINOS) INDEPENDENTES	67
4.1.2. FUSÃO DE MÓDULOS COM ORIGENS (DESTINOS) COMUNS	68
4.2. APLICAÇÃO DA ESTRATÉGIA.....	70
4.3. ALGORITMO PARA VALIDAÇÃO DA ESTRATÉGIA	73
4.3.1. CÁLCULO DOS CUSTOS DE COMUNICAÇÃO	73
4.3.2. O ALGORITMO.....	77
4.4. SIMULAÇÃO DAS APLICAÇÕES PARA ARQUITETURAS SDR-NOc SIMPLES.....	79
4.4.1. PROCESSO DE SIMULAÇÃO	80
4.5. RESULTADOS EXPERIMENTAIS	81
5. MAPEAMENTO PARA ARQUITETURAS SDR-NOCS COMPLEXAS	86
5.1. FORMULAÇÃO DO PROBLEMA.....	86
5.2. APLICAÇÕES SINTÉTICAS	93
5.3. ALGORITMOS PARA RESOLUÇÃO DE MAPEAMENTO E POSICIONAMENTO.....	95
5.3.1. CODIFICAÇÃO DAS POSSÍVEIS SOLUÇÕES.....	95
5.3.2. RESOLUÇÃO COM ALGORITMO SEMIEXAUSTIVO.....	96
5.3.3. RESULTADOS DO ALGORITMO SEMI-EXAUSTIVO.....	97
5.3.4. RESOLUÇÃO COM ALGORITMOS GENÉTICOS.....	98

5.3.4.1. Estrutura geral do algoritmo genético	98
5.3.4.2. Tipos de cruzamento (<i>crossover</i>) utilizados.....	101
5.3.4.3. Técnicas de variação da população.....	105
5.3.5. ESCOLHA DOS PARÂMETROS DE SIMULAÇÃO	106
5.3.6. SIMULAÇÃO E RESULTADOS.....	108
6. CONCLUSÕES E TRABALHOS FUTUROS.....	112
6.1. CONTRIBUIÇÕES.....	112
6.2. CONCLUSÕES.....	114
6.3. TRABALHOS FUTUROS.....	115
REFERÊNCIAS.....	117
APÊNDICE A - DADOS DAS APLICAÇÕES SINTÉTICAS	125

1. INTRODUÇÃO

Serão apresentados neste capítulo os distintos aspectos que justificam a utilização tanto dos sistemas dinamicamente reconfiguráveis (SDRs), quanto das redes intrachip neste contexto. A partir destes princípios, serão apresentados argumentos que motivam a exploração do problema e o desenvolvimento de algoritmos para mapeamento e posicionamento de módulos processantes em SDRs baseados em redes intrachip. Também serão apresentados os objetivos do presente trabalho, as contribuições originais da tese e a organização dos capítulos subsequentes.

1.1. Contextualização

1.1.1. Redes Intrachip

A indústria da microeletrônica está continuamente em evolução. A capacidade e a complexidade dos circuitos integrados têm avançado em um ritmo maior que a produtividade de projeto que as equipes de engenharia têm mantido. Este problema tem sido evidenciado por diversos relatórios, como o documento publicado pela Associação das Indústrias de Semicondutores (ITRS, 2012) em um trabalho que mostra que o número de transistores por CI crescem em média 58% ao ano, enquanto a produtividade, representada por transistores por funcionário-mês, cresce, em média, apenas 21% ao ano.

Uma solução apontada para reduzir a crise de projeto é o emprego de sistemas sobre silício (*System on Chip*, SoC) que consistem na integração de diversos blocos complexos em um único chip. Essa estratégia permite o reuso de núcleos de hardware já testados e pré-validados conhecidos como *Intellectual Property Cores* (IPs ou, simplesmente, *cores*), que reduzem significativamente o ciclo de projeto.

Devido ao número crescente de componentes e o aumento do desempenho dos mesmos, a quantidade e o fluxo de informações dentro de um chip está se

tornando crítica e o projeto da estrutura de comunicação passa a ter um papel fundamental na definição de área, desempenho e consumo de energia de todo o sistema (MARCULESCU et al., 2009). O projeto de comunicação intrachip está baseado em técnicas tradicionais baseadas em interconexões globais, tais como barramentos e comunicação ponto-a-ponto. Diversos trabalhos vêm demonstrando que esse tipo de estrutura de comunicação não é escalável o suficiente para as arquiteturas atuais que englobam muitos *cores* (CORNELIUS; TIMMERMAN, 2006; MARCULESCU et al., 2009). Além disso, com a escalada da tecnologia as interconexões globais podem causar erros de sincronização, *delays* imprevisíveis e alto consumo de potência (MARCULESCU et al., 2009).

Quando o número de *cores* presentes em um sistema é elevado, redes intrachip (Network on Chip, NoC) (BENINI; DE MICHELI, 2002; KUMAR et al., 2002) tornam-se atrativas; elas apresentam uma estrutura composta basicamente por um conjunto de roteadores e canais de comunicação que interconectam núcleos (i.e., memórias, processadores, dispositivos de entrada ou saída, etc.) dentro de um circuito integrado. Estes núcleos são comumente denominados elementos processantes (EPs). As NoCs aplicam, em nível intrachip, importantes conceitos herdados da área de redes de computadores tais como topologia e mecanismos de comunicação. A topologia estabelece a forma de interconexão entre os roteadores na rede. Pode-se citar como exemplos de topologias estáticas: grelha, toróide 2D, cubo 3D, cubo 4D e totalmente conectada; estas topologias estão ilustradas na Figura 1 (RAFFO, 2010), onde os círculos laranjas representam os roteadores e as linhas entre estes representam as conexões. Nas arquiteturas tradicionais de NoC, a cada roteador tem-se um EP associado, portanto, cada nó da rede consiste de conjunto roteador e EP, caso conhecido como NoC .

As arquiteturas baseadas em NoC tornam-se uma alternativa relevante frente às limitações impostas pelas arquiteturas de comunicação tradicionais, apresentando alta escalabilidade, melhor previsibilidade, menor consumo de potência, além de fornecer suporte adequado a tarefas executadas paralelamente em um mesmo chip (MARCULESCU et al., 2009).

Figura 1 - Topologias: (a) Grelha 2D; (b) Toróide 2D; (c) Cubo 3D; (d) Cubo 4D; (e) Totalmente Conectada.

Fonte: (RAFFO, 2010)

1.1.2. Field Programmable Gate Arrays (FPGAs)

Até a década de 1990, o cenário para implementação de algoritmos de uso específico estava polarizada em dois tipos de dispositivos: os processadores de propósito geral (*general-purpose processors*, GPPs) e os circuitos integrados de aplicação específica (*application specific integrated circuits*, ASICs). O primeiro, baseado na arquitetura Von Neumann, possui grande flexibilidade de aplicação, além de permitir edições e modificações de projeto; entretanto, devido à natureza sequencial e às características multiuso dos GPPs, estes apresentam desempenho computacional muito abaixo das soluções em hardware dedicado. Os ASICs por outro lado, apresentam alta eficiência computacional, porém pouca flexibilidade e nenhum tipo de alteração em campo. Com o advento dos arranjos de portas programáveis em campo (*field programmable gate arrays*, FPGAs), encontrou-se

uma solução de compromisso, obtendo parte da flexibilidade dos GPPs e parte do alto desempenho computacional típico dos ASICs.

A principal característica que distingue os FPGAs de outras soluções é a sua configurabilidade. Existem FPGAs baseados em tecnologia *antifuse* que são configuráveis apenas uma única vez. Todavia, a adoção da tecnologia EPROM e SRAM permitiu a reconfiguração dos FPGAs (i.e., um número ilimitado de configurações em um mesmo dispositivo), bastando para isso carregar um novo *bitstream* na memória de configuração interna.

As soluções com processadores de uso geral continuam sendo a alternativa mais flexível, pois admitem novas aplicações e modificações de projeto através de alterações no software, porém sob penalidade de desempenho e maior consumo comparado aos ASICs. Para tarefas específicas em que o desempenho do hardware é uma questão crucial, os ASICs ainda são a solução mais recomendada, entretanto, o projeto com ASICs deve ser justificado por um alto volume de produção para compensar os investimentos iniciais do projeto e dos altos custos de fabricação. Os FPGAs, por outro lado, tendem a possuir um ciclo de projeto menor, possibilitando alterações do projeto, com custo menor, nas fases mais avançadas e com o desempenho relativo aumentando e aproximando-se cada vez mais dos ASICs. Por estes motivos, os FPGAs vêm tomando cada vez mais espaço de mercado dos ASICs (HENKEL; PARAMESWARAN, 2007); além disso, os FPGAs não são somente interessantes para a substituição dos ASICs no uso final, mas também para a prototipagem destes.

O desenvolvimento de FPGAs com programação baseada na tecnologia SRAM levou ao surgimento de uma estrutura que suporta uma forma especial de operação na qual podem ser feitas alterações em determinadas áreas do dispositivo, sem alterar as demais (XILINX, 2014). Este tipo de reconfiguração, denominada reconfiguração dinâmica parcial, possibilitou a implementação de vários subcircuits que podem ser mapeados sucessivamente em uma estrutura de hardware, comum a todos eles, aumentando assim a densidade lógica de um chip (i.e., permitindo maior alocação de recursos por área disponível no dispositivo) (LIU; WONG, 1999).

As tecnologias de reconfiguração parcial permitiram a implementação de um circuito em partições sucessivas, temporalmente excludentes, em um mesmo dispositivo, sem interrupção de seu funcionamento. Assim, uma dimensão temporal

foi adicionada à visão meramente espacial utilizada no projeto de ASICs. Os FPGAs com reconfiguração parcial que suportam esse tipo de operação serão definidos neste trabalho como FPGAs dinamicamente reconfiguráveis (Dynamically Reconfigurable FPGAs, DRFPGAs) (ZHANG; NG, 2000).

A Figura 2 (LYSAGHT; DUNLOP, 1993) mostra o deslocamento temporal da alocação de recursos em um DRFPGA, onde temos 11 tarefas executadas em três fases. Na primeira fase executamos as tarefas de 1 a 6. Na segunda fase as tarefas 2 e 6 mantêm seus posicionamentos as tarefas 7 e 8 são posicionadas no espaço de trabalho. Na última fase, além de manter as tarefas 2, 6 e 7, as tarefas 9, 10 e 11 são posicionadas e executadas.

Figura 2 - Particionamento temporal em projetos de DRFPGAs.

Fonte: (LYSAGHT; DUNLOP, 1993)

Note que se torna necessário considerar aspectos, tais como dependência de dados na sequencia das partições, via de dados adaptada às várias partições, projeto do controlador de partições, entre outros. Conclui-se que a atenção deve incluir o domínio do tempo. Para projetos que envolvam arquiteturas dinamicamente reconfiguráveis é necessária uma estratégia de partição bidimensional com um domínio espacial e outro temporal, sendo necessários novos métodos de projeto.

1.1.3. Sistemas Dinamicamente Reconfiguráveis (SDRs)

O desenvolvimento de DRFPGAs permitiu o desenvolvimento de Sistemas Dinamicamente Reconfiguráveis (SDRs), levando à melhor utilização da área do FPGA, aumento da densidade lógica do chip e à redução de consumo de potência, fator importante no desenvolvimento de sistemas embutidos, quando comparado aos FPGAs usados de forma tradicional.

Algumas implementações em reconfiguração dinâmica têm sido anunciadas em várias aplicações computacionais, trazendo ganhos em área e consumo de potência. Incluem-se nesta lista, aplicações em telecomunicações (ESQUIAGOLA, 2006), aplicações médicas, (XIE et al, 2014), sistemas automotivos (OHKAWA et al., 2014), arquitetura de computadores (MAJER et al., 2007; LU et al., 2008), processamento digital de sinais (SAHA; AMITABHA, 2013), entre outros.

Nos últimos anos, uma das questões que mais tem gerado interesse no que se refere às metodologias de projeto utilizando DRFPGAs é a definição de um meio de comunicação estruturado para tratar da transferência de dados entre as partições de um SDR. Os métodos de interconexões globais, baseados em barramentos, não são se adaptam facilmente a projetos de SDRs, pois são estruturas fixas e unidimensionais, que têm como princípio que a origem e o destino da comunicação serão imutáveis. Já as NoCs são bem mais flexíveis, baseando-se em um sistema que se estende da mesma maneira em duas dimensões e aproveitando a capacidade dos novos DRFPGAs para reconfigurar uma bloco específico do dispositivo.

O primeiro trabalho a implementar um SDR com infra-estrutura NoC, que será denominado de Sistema Dinamicamente Reconfigurável baseado em Rede Intrachip (SDR-NoC), foi desenvolvido por Bobda (BOBDA et al., 2005). O sistema proposto, denominado DyNoC (*Dynamic Network on Chip*), é ilustrado na Figura 3, mostrando uma DyNoC 7x7, em uma topologia do tipo grelha, onde cada roteador (representado por círculos) possui cinco canais de comunicação: quatro para conexão com outros roteadores e um canal de comunicação com um módulo processante, MP. Os MPs da figura podem ser de tamanhos unitários (blocos azuis)

ou múltiplos (demais *cores*); todos podem ser substituídos por outros blocos reconfiguráveis, que são alocados de maneira livre dentro da rede de roteadores, sempre tendo como roteador associado aquele ao seu canto superior direito. Pela proposta dos autores, os roteadores "cobertos" pelos MPs maiores não são desconfigurados, mas apenas desativados.

Figura 3 - Arquitetura DyNoC.

Fonte: (BOBDA, 2005)

Uma alternativa interessante foi proposta por Jovanovic (JOVANOVIC et al., 2007, 2008) propondo uma arquitetura onde os roteadores possuem apenas quatro portas (leste, oeste, norte e sul), sendo os blocos alocados no lugar dos próprios roteadores. Esta solução é especialmente interessante para o projeto de SDRs visando famílias mais recentes de DRFPGAs, que permitem a reconfiguração parcial de pequenos retangulares do dispositivo, pois a NoC proposta é totalmente 2D, com grande liberdade para alocação dos blocos reconfiguráveis e uma estrutura de comunicação bastante adaptativa. Outras propostas se seguiram (JOVANOVIC; TANOUGAST; WEBER, 2008; RANA et al., 2010; KILLIAN et al., 2012), porém, essencialmente, são variações das arquiteturas SDR anteriores.

1.2. Motivação

Um dos aspectos importantes do projeto de NoCs é o mapeamento das tarefas da(s) aplicação(ões) a ser(em) implementada(s), que, em uma NoC tradicional, consiste na escolha da melhor posição em uma NoC para que uma dada tarefa seja alocada e processada (CARVALHO, 2009). Esta escolha pode aumentar ou diminuir dramaticamente o desempenho da comunicação no dispositivo. Para ilustrar este conceito, considere-se uma NoC 3x3, mostrada na Figura 4, na qual estão alocadas as tarefas A, B, C, ..., F, de uma dada aplicação. Tal alocação significa que uma dada tarefa está sendo computada junto a um roteador em determinada posição na topologia da NoC; pode tratar-se de tarefas implementadas como programa em um GPP (sistema multiprocessado) ou como núcleos específico de hardware. Estes núcleos são conhecidos na literatura pelo seu nome em inglês, *core*; também como elementos processantes (EPs) ou, ainda, como módulos processantes (MPs) termo que será o mais adotado neste trabalho.

Considere-se que a tarefa J é a tarefa mais comunicante, trocando dados com as tarefas A, B, E e G, em uma alta taxa de transmissão/recepção de dados; por outro lado, a tarefa C é a menos comunicante, comunicando-se apenas com a tarefa A, com pequena taxa de transmissão/recepção de dados. Fica perfeitamente claro que a Figura 4(a) possui um mapeamento muito mais eficiente do que o da Figura 4(b), pois alocando a tarefa J no canto superior esquerdo, diminuem-se os canais de comunicação, e um caminho maior é necessário para um pacote de dados atingir seu destino. Assim, para a tarefa J comunicar-se com a tarefa E, por exemplo, ela deve percorrer o caminhos pelos roteadores junto às tarefas TA, TI, TE ou TB, TF, TE, saturando as vias de dados entre os roteadores. Ademais, é mais eficiente alocar TA e TC em nós vizinhos como na Figura 4(a) do que afastados como na Figura 4(b).

O mapeamento é dito estático quando definido em tempo de síntese ou projeto (CARVALHO, 2009). Este tipo de mapeamento já foi explorado por diversos trabalhos (HU; MARCULESCU, 2003; MURALI; DE MICHELI, 2004; BERTOZZI et al., 2005; HU, MARCULESCU, 2005; MURALI et al., 2006; MARCULESCU et al., 2009). A literatura também define o conceito de mapeamento dinâmico, onde as

posições das tarefas são determinadas em tempo de execução, baseados em parâmetros como carga do sistema, congestionamento da rede, etc. Em geral, tais estratégias estão associadas à distribuição de tarefas junto a GPPs conectados aos roteadores. Como trabalhos importantes de mapeamento dinâmico pode-se citar (KIM et al., 2005; KIM et al., 2007; CHOU; MARCULESCU, 2007; CARVALHO; CALAZANS; MORAES, 2007; CARVALHO, 2009).

Figura 4 - A importância da posição dos elementos processantes em uma NoC.

Fonte: Autor.

Quando a rede intrachip está suportando um SDR, o problema torna-se mais complexo, pois as tarefas alternam-se durante a execução da aplicação. Para ilustrar a influência da reconfiguração dinâmica sobre o processo de mapeamento, vamos analisar um exemplo alternativo de NoC 3x3, agora com um SDR, onde temos sete tarefas fixas (TT a TZ) e quatro tarefas reconfiguráveis (R1 a R4), das quais duas estão sempre alocadas a duas posições específicas na rede. Em um dado momento, o sistema se reconfigura e as tarefas R1 e R3 dão lugar às tarefas R2 e R4. Em SDRs, a alocação de tarefas está associada apenas à implementação em MPs, diferente de sistemas multiprocessados. As tarefas trocam grande volume de dados entre si no padrão descrito abaixo.

R1 ↔ TV,TX

R2 ↔ TY,TZ

R3 ↔ TY

R4 ↔ TX

A Figura 5(a) mostra um bom posicionamento para as tarefas R1 e R3, porém, quando estas dão lugar as tarefas R2 e R4 a comunicação se mostra ineficiente. Já a Figura 5(b) mostra o caso oposto, onde R2 e R4 possuem bom posicionamento, mas quando se transformam novamente em R1 e R3, os canais ficam mais congestionados. Um mapeamento eficiente para qualquer uma das configurações é mostrado na Figura 5(c), onde para ambos os cenários o posicionamento é satisfatório.

Figura 5 - A importância do posicionamento em um SDR implementado em NoC.

Fonte: Autor.

O estudo dos problemas de mapeamento será detalhado no capítulo 2. De acordo com a classificação criada nesta tese, o problema ilustrado na Figura 5 considera uma arquitetura SDR-NoC simples, composto de NoCs diretas e regulares com MPs homogêneos. As situações mais complexas de SDRs com interfaces de comunicação do tipo NoC, como as propostas por (BOBDA, 2005) e (JOVANOVIC, 2007), que tratam de NoC indiretas e irregulares, serão classificadas como arquiteturas SDR-NoC complexas.

Para o entendimento de arquiteturas SDR-NoC complexas, será usado como exemplo uma arquitetura similar à proposta por Jovanovic. A arquitetura é composta de diversos roteadores, que podem ser trocados, a qualquer momento, por MPs de qualquer tamanho e que entrem em qualquer posição. Estão alocadas, neste exemplo, oito tarefas estáticas (de TA a TH) e duas tarefas dinâmicas (R1 e R2) como mostra a Figura 6, onde os blocos cinzas são tarefas estáticas, os blocos coloridos são tarefas dinâmicas e os círculos são roteadores (as letras indicam a

qual bloco os roteadores estão associados). O sistema funciona de forma que no primeiro momento executa-se a tarefa R1 e posteriormente a tarefa R2 seja executada onde o módulo R1 foi inicialmente alocado. Também será considerado que as tarefas R1 se comunica com TD e TA, e a tarefa R2 se comunique com TD e TE.

Figura 6 - Extensão do mapeamento para arquiteturas mais complexas.

Fonte: Autor.

O mapeamento é uma tarefa mais complexa agora, uma vez que os roteadores associados ao EPs, podem trocar de posição durante uma reconfiguração. Portanto, além das métricas de comunicação, largura de banda ou de vizinhança, há um fator adicional: a ocupação de espaços na NoC em determinado tempo. Desta forma, a tarefa de posicionamento deve ser desenvolvida em paralelo, de tal maneira que aborde todos os aspectos desta estrutura mais complexa.

A Figura 6(a) exhibe um mapeamento ruim, pois além de R1 estar muito longe de TA (tendo que percorrer cinco roteadores), com esta disposição, a tarefa R2 não poderá ser alocada em lugar algum (a tarefa R2 é quatro vezes maior que a tarefa

R1 e não pode ser alocada no mesmo espaço de R1) e a reconfiguração não poderá ser executada. A Figura 6(b) mostra um mapeamento em que as duas tarefas podem ser executadas em um mesmo espaço e que, além disso, aproxima as duas tarefas reconfiguráveis de TA, TD e TE, melhorando o consumo de energia e a performance da rede.

Assim sendo, o problema deixa de ser apenas de mapeamento, e passa a ser um problema de mapeamento e posicionamento, já que agora, um não pode ser dissociado do outro.

Até o momento, o mapeamento para trabalhos de SDRs baseados em NoCs, como a DyNoC e a CuNoC, ainda não foram desenvolvidos ou tratados com a devida atenção. O mapeamento para SDRs foi abordado por Beretta (BERETTA et al., 2011), porém, a arquitetura utilizada é diferente das descritas anteriormente. Os autores definem um conjunto de *slots* reconfiguráveis (são ligados por uma interconexão geral que pode ser um *bus* ou uma NoC) onde diversas tarefas podem ser alocadas. É importante, portanto, o desenvolvimento de novos métodos de mapeamento para SDRs baseados em NoC.

1.3. Justificativa

O Grupo de Projeto de Sistemas Eletrônicos Integrados e Software Aplicado (GSEIS) do Laboratório de Microeletrônica (LME) da Universidade de São Paulo (USP) tem como seu interesse principal de investigação as metodologias de projeto de sistemas digitais. O grupo tem desenvolvido importantes trabalhos na área de reconfiguração dinâmica (ESQUIAGOLA, 2006; GONZALEZ, 2006; KOJIMA, 2007; RAFFO, 2010; RAFFO et al., 2010; GOMES FILHO, 2010; GOMES FILHO et al., 2010) e também contribuído para área de desenvolvimento das NoCs, explorando modelos de alto nível de abstração (SEPÚLVEDA, 2006), Serviço de Qualidade de Segurança (SEPÚLVEDA et al., 2010, SEPÚLVEDA, 2011), assim como no aproveitamento da estrutura das redes intrachips para a comunicação entre módulos fixos e reconfiguráveis (RAFFO 2010, RAFFO et al., 2010). Este trabalho vem dar continuidade à linha de trabalho do GSEIS e apresentar um desenvolvimento unindo as duas linhas de pesquisa.

1.4. Objetivos

Este trabalho tem como objetivo geral contribuir no amadurecimento das técnicas de projeto de sistemas dinamicamente reconfiguráveis, assim como nas técnicas de mapeamento e posicionamento de NoCs. Com o desenvolvimento de novas metodologias, pretende-se aumentar a viabilidade de projetos de SDRs, assim como a eficiência da comunicação dentro do sistema.

Mais especificamente, o trabalho tem como objetivo desenvolver três áreas dentro do estado da arte: aspectos teóricos de SDR-NoCs, mapeamento de arquiteturas SDR-NoCs simples e mapeamento e posicionamento de arquiteturas SDR-NoCs complexas.

1.4.1. Objetivos específicos no aspecto teórico

- Definir métricas e considerações para a construção de um arcabouço teórico para caracterizar e classificar as arquiteturas SDR-NoCs em dois tipos: arquiteturas simples e arquiteturas complexas.
- Formalizar o problema geral de mapeamento e posicionamento de arquiteturas SDRs complexas baseadas em NoCs.

1.4.2. Objetivos específicos no mapeamento de arquiteturas SDR-NoCs simples

- Propor e construir uma estratégia de mapeamento para arquiteturas SDR-NoC simples, tendo como ponto de partida as partições fixas e reconfiguráveis, o grafo da aplicação e a caracterização da NoC.
- Provar a eficiência da estratégia acima através da comparação de seus resultados obtidos por mapeamentos aleatórios e controlados com resultados ótimos (obtido através de um algoritmo *branch-and-bound*).
- Confirmar os resultados utilizando uma simulação de baixo nível e recolher dados de latência, *delay*, e consumo de potência da NoC.

1.4.3. Objetivos específicos no mapeamento de arquiteturas SDR-NoCs complexas

Assim como para o mapeamento em arquiteturas SDR-NoCs simples, o mapeamento em arquiteturas SDR-NoCs complexas ainda não foi devidamente estudado. A natureza do problema de mapeamento é naturalmente explosiva (MARCULESCU, 2009), e, como o número de fatores e elementos de uma arquitetura SDR-NoC complexa é muito superior a de uma NoC simples, deve-se buscar um algoritmo que responda bem ao problema, com uma evolução o mais linear possível.

Algoritmos genéticos (GOLDBERG, 1989) têm sido utilizados para uma série de problemas de otimização com excelentes resultados. Inclusive para problemas de mapeamento (LEI; KUMAR, 2003; RANA et al. 2009; BERETTA et al. 2011).

Assim sendo, os objetivos de mapeamento foram:

- Buscar uma primeira solução exata (solução ótima) para o problema de mapeamento e posicionamento de arquitetura SDR-NoCs complexas, com objetivo de utilizá-las como base de comparação com resultados posteriores, já que não existem outros trabalhos para comparação.
- Adaptar o problema desenvolvido teoricamente para a solução com algoritmos genéticos (codificação, posicionamento, etc.)
- Construir um algoritmo genético para solucionar o problema para uma grande variedade de casos, com boa otimização em um tempo rápido de execução.
- Testar o algoritmo genético com diferentes métricas e configurações em diversas aplicações para a validação do método.

1.5. Contribuições

A contribuição principal do presente trabalho de doutorado será a obtenção de estratégias e algoritmos de mapeamento e posicionamento em NoCs baseadas em SDR. Tais estratégias e algoritmos irão contribuir para o ciclo de projeto de SDRs que utilizam como estrutura de comunicação as NoCs.

Até o momento, encontram-se poucos trabalhos que tratam este problema em arquiteturas SDR-NoCs simples, e não se encontra nenhum trabalho na literatura que tenha abordado o problema em arquiteturas SDR-NoCs complexas. Portanto, o trabalho terá uma contribuição original para o estado da arte. Mais especificamente, as contribuições originais serão:

- Construção de um arcabouço teórico como suporte à tese.
- Definição formal do problema geral de mapeamento de arquiteturas SDR-NoCs complexas.
- Desenvolvimento e validação de uma estratégia de mapeamento para arquiteturas SDR-NoCs simples.
- Desenvolvimento e validação de um algoritmo genético rápido e eficiente para o mapeamento e posicionamento em arquiteturas SDR-NoCs complexas.

1.6. Organização do texto

Este trabalho está organizado em seis capítulos. O capítulo dois mostra os aspectos teóricos que serviram como base deste trabalho, dividido em sete subseções: problema geral de posicionamento; redes intrachips; mapeamento de redes intrachips; reconfiguração dinâmica, particionamento e escalonamento de processos em SDRs, SDRs baseadas em redes intrachips e algoritmos. O terceiro capítulo lista alguns trabalhos correlatos que contribuem com o entendimento da proposta. O quarto capítulo apresenta uma estratégia de mapeamento para sistemas SDR-NoCs simples, enquanto o quinto capítulo apresenta a formulação e a solução de mapeamento e posicionamento de arquiteturas SDR-NoCs complexas, ambos com resultados e discussões. Finalmente, o sexto capítulo apresenta as conclusões deste trabalho.

2. BASES TEÓRICAS

Neste capítulo serão abordados fundamentos teóricos importantes para o desenvolvimento deste trabalho, focando principalmente nas bases da metodologia proposta: o problema geral de posicionamento, a teoria das redes intrachip, os princípios da reconfiguração dinâmica em FPGAs, o conceito de mapeamento de tarefas em NoCs e a teoria básica dos principais algoritmos utilizados na tese.

2.1. O problema geral de posicionamento

O problema geral de posicionamento pode ser descrito basicamente como a alocação de vários itens retangulares em um determinado espaço bidimensional de entrada, ocupando-o de acordo com um objetivo. Em aplicações industriais e logísticas, o objetivo é, geralmente, ocupar o espaço consumindo a menor área possível (LODI; MARTELLO; MONACI, 2002).

Em FPGAs, o posicionamento de módulos é importante para diminuir o número de interconexões e aumentar o desempenho (CHEN; CONG; PAN, 2006). Em DRPGAs, há motivações adicionais para o posicionamento, como, por exemplo, o aproveitamento de recursos do dispositivo e o uso de *bus-macros* (RAFFO, 2010).

Formalmente, o problema pode ser descrito da seguinte maneira: dados n itens retangulares $j \in J = \{1, \dots, n\}$, cada um definido por uma largura w_j e uma altura h_j , deve-se colocá-los em um espaço de entrada que possui uma largura W e uma altura H , com as seguintes restrições: 1) o posicionamento não pode violar os limites do espaço de entrada. 2) o posicionamento de um retângulo não pode violar o espaço de outro retângulo, ou seja, dois retângulos não podem ocupar o mesmo espaço.

Um exemplo do problema é ilustrado na Figura 7(a), onde existem sete peças de diversas dimensões, cujos tamanhos, em unidades arbitrárias, estão descritos na Tabela 1.

Tabela 1 - Largura x altura de peças de posicionamento.

Peça	Peça 1	Peça 2	Peça 3	Peça 4	Peça 5	Peça 6	Peça 7
Tamanho	1 x 4	4 x 1	1 x 3	4 x 4	1 x 1	1 x 3	5 x 1

Fonte: Autor

Dadas as dimensões das peças retangulares, deve-se posicioná-las em um espaço de entrada (W, H) de 8 x 8, ocupando a área de maneira eficiente, como a solução apresentada na Figura 7(b).

Figura 7 - Exemplo de posicionamento.

Fonte: Autor

Nota-se que o posicionamento adotado utilizou uma área compactada de tamanho 36 dentro de um espaço disponível de 64. Uma disposição fragmentada poderia impedir o posicionamento de algumas peças como no exemplo da Figura 8, onde não sobra espaço para alocação da peça 4.

Figura 8 - Exemplo de posicionamento ineficiente.

Fonte: Autor.

2.2. Redes intrachip

As redes intrachip surgiram nos anos 90 como alternativa aos sistemas de interconexão global em SoCs tais como barramentos e conexão ponto-a-ponto, trazendo vantagens como escalabilidade, flexibilidade e seu grande potencial para trabalhar com paralelismo de tarefas.

A Figura 9 ilustra a estrutura básica de uma NoC 3x3 com topologia do tipo grelha, onde cada roteador é ligado aos vizinhos imediatos. Cada roteador é também conectado a um módulo processante (MP), que pode ser um GPP, memória, acelerador, IP em geral, etc.

Figura 9 - Rede intrachip 3x3.

Fonte: Autor

Os roteadores são estruturas básicas de uma NoC. Eles transmitem as informações disponíveis dos canais de entrada para os canais de saída, através de portas de comunicação. A estrutura genérica de um roteador é composta por estrutura de chaveamento, controle de chaveamento e elementos de armazenamento temporário de informações entre os canais (*buffers*). Esta estrutura é mostrada na Figura 10. Apesar de não estar explícita na figura, a forma mais comum para roteadores é que eles tenham quatro portas (norte, sul, leste e oeste) para conexão com outros roteadores e um quinto porto para conexão específica com a MP a eles associados.

Figura 10 - Estrutura genérica de um roteador

Fonte: Autor.

As características mais importantes das NoCs são a topologia, os mecanismos de roteamento, os algoritmos de roteamento, o controle de fluxo e as políticas de arbitragem.

A topologia, como já mencionado no Capítulo 1, é a forma de interconexão entre os roteadores da rede. Há duas formas comuns de classificar as topologias:

- **Regulares / Irregulares**

Topologias regulares possuem conexões diretas entre os roteadores, e os roteadores, por sua vez, possuem conexões diretas com os MPs, dentro de alguma regra estabelecida. A rede da Figura 9 é um exemplo clássico de topologia regular, outros exemplos podem ser observados na Figura 1. Topologias irregulares, por sua vez, não apresentam regras de conexão; incluem-se casos em que roteadores não possuem necessariamente conexão com todos os seus vizinhos imediatos. A Figura 11(a) (BJERREGAARD; MAHADEVAN, 2006) mostra um exemplo de NoC irregular.

- **Diretas / Indiretas**

As topologias diretas são aquelas em que cada roteador da rede conecta-se diretamente a um determinado MP através de ligações ponto-a-ponto, como ilustrado na Figura 9. As topologias indiretas são aquelas em que a conexão entre roteadores e MPs não são fixas, sendo possível mais de um caminho entre roteadores e MPs. Um exemplo deste tipo de topologia pode ser observado na Figura 11(b) (RAFFO, 2010) onde os MPs são alocados em volta de uma matriz de

chaveamento, conhecida como *crossbar*. Nota-se que para cada conexão, há vários caminhos possíveis.

Figura 11 - (a) Topologia irregular (b) Topologia indireta.

Fonte: (RAFFO, 2010)

Para que um pacote saia de um determinado nó da rede e atinja o nó de destino, são necessários os mecanismos de roteamento. Os mecanismos de roteamento definem a forma em que os dados trafegam dentro da rede. Geralmente cabeçalhos dos pacotes incluem a identificação dos destinos, para que sejam processados nos roteadores. Os principais métodos conhecidos são o *store-and-forward* (SAF), *virtual-cut-trought* (VCT) e *wormhole*. No método SAF, os pacotes são armazenados em *buffers* dentro dos roteadores, e o cabeçalho é analisado para determinar por qual canal o pacote será enviado. O SAF gera assim uma alta latência para a rede, além de que, o tamanho dos buffers precisa necessariamente ter o tamanho máximo de um pacote. O VCT apresenta uma evolução do SAF, adicionando um recurso com o qual, só se envia o pacote para o próximo roteador quando há disponibilidade de receber todo o pacote. Assim o VCT diminui a latência do sistema, porém, caso a rede esteja sobrecarregada o latência volta a ser tão alta quanto a do sistema SAF. Finalmente, o sistema *wormhole* divide o pacote original em pacotes menores, denominados unidades de controle de fluxo (*flow control units*, *flits*) que representam a menor unidade de controle de fluxo. Os *flits* são enviados individualmente. Assim, o impacto direto deste método é a redução dos *buffers*, além da redução de latência, fato de alta relevância, tendo em vista que a maioria das

NoCs possui limitação de recursos de armazenagem e requerimentos rígidos de latência (MARCULESCU et al, 2009).

Juntamente com os mecanismos de roteamento, os algoritmos de roteamento são fundamentais para o funcionamento da NoC. Eles são responsáveis pela definição da direção e sentido que o pacote deve tomar ao chegar em um roteador; as decisões de envio e recebimento de pacotes existem de tal forma que sejam minimizados/eliminados efeitos que paralisam a comunicação, tais como *deadlocks* e *livelocks*.

Os algoritmos de roteamento podem ser locais ou distribuídos. O roteamento é local quando a decisão do caminho que o pacote deve percorrer é tomada no próprio MP, incluindo-se a informação no pacote. O roteamento é distribuído quando o pacote é controlado diretamente pelo roteador que determina se o pacote será enviado para o MP associado ou se será enviado para outro roteador.

Outro atributo dos algoritmos de roteamento é a sua capacidade de adaptação. Os algoritmos determinísticos não possuem capacidade de adaptação; dados uma origem e um destino, com o algoritmo determinístico um pacote sempre percorrerá o mesmo caminho, não importando o estado de ocupação da rede. Os algoritmos adaptativos, por outro lado, podem percorrer um caminho alternativo quando um recurso dentro de sua rota estiver ocupado. Os algoritmos adaptativos distribuem o tráfego de modo mais homogêneo e diminuem a latência da rede, porém, são mais complexos e utilizam mais recursos de hardware para sua implementação. Como exemplos de algoritmos de roteamento determinístico pode-se citar os clássicos XY, *e-cube*, etc. (DUATO et al., 2003). Entre os adaptativos, encontram-se os algoritmos *dynamic adaptive switching* (HU; MARCULESCU, 2004), *turn model routing* (DALLY; TOWLES, 2004), *surrounding XY* (BOBDA, 2005), entre outros.

O controle de fluxo lida com a alocação de recursos necessários para que os pacotes trafeguem pela rede (e.g. *buffers* e canais). Quando o pacote compete por estes recursos, o controle de fluxo deve tomar uma série de decisões para que a comunicação continue sendo estabelecida na NoC. O controle de fluxo pode descartar o pacote, desviá-lo do caminho original, bloquear recursos, etc. Os controles de fluxo mais comuns são os do tipo *handshake* ou em sistemas baseados em créditos.

A política de arbitragem lida com conflitos pelo uso de recursos, e.g. quando dois ou mais pacotes competem por um buffer ou canal de saída de um roteador. A arbitragem pode ser centralizada (quando há um comando central que controla tanto a arbitragem quanto o algoritmo de roteamento) ou distribuída (onde há um módulo arbitrador para cada saída do roteador). Assim a política de arbitragem pode ser baseada em uma série de prioridades, podendo-se citar dentre elas: prioridade estática, prioridade dinâmica, *first-come-first-served* (FCFS), *least recently served* (LRS), etc. (DALLY; TOWLES, 2004).

2.2.1. Homogeneidade dos Módulos Processantes

Apesar de não ser aspecto de uma NoC, e sim da arquitetura de um SOC, a homogeneidade dos MPs pode influenciar significativamente o projeto de uma NoC por causa dos aspectos de posicionamento. A arquitetura dos elementos processantes de uma NoC pode ser definida como homogênea ou heterogênea (BJERREGAARD; MAHADEVAN, 2006).

A Figura 12 compara as duas possibilidades de homogeneidade. A Figura 12(a) mostra uma arquitetura heterogênea onde os MPs possuem diferentes tamanhos. A compreensão fica mais clara quando comparada com a arquitetura homogênea ilustrada na Figura 12(b). Uma NoC com arquitetura homogênea é geralmente reservada para módulos do mesmo tamanho, enquanto que uma NoC com arquitetura heterogênea é desenvolvida para módulos de diversos tamanhos e gera um cuidado especial em relação ao posicionamento destes módulos.

Figura 12 - NoC com MPs (a) heterogênea (b) homogênea.

Fonte: (BJERREGAARD; MAHADEVAN, 2006)

2.3. Mapeamento de tarefas em redes Intrachip

O processo de mapeamento em uma rede intrachip tem como premissa de que a sua topologia já foi definida e a aplicação está descrita como um conjunto de tarefas concorrentes que já foram definidas para serem executadas em uma lista de núcleos (MPs), seja como software ou hardware. O mapeamento é o processo de decisão de como alocar topologicamente os MPs junto aos nós da rede NoC de uma forma na qual as métricas de interesse sejam otimizadas. O problema de mapeamento é um problema de otimização combinatória similar à atribuição quadrática (quadratic assignment problem) que é um problema do tipo NP, ou seja, para os quais não se tem soluções exatas e genéricas que possam ser obtidas em tempo polinomial¹, sendo assim necessário evitar a análise exaustiva e desenvolver métodos e heurísticas específicos que contornem o problema.

Será apresentado a seguir um modelo matemático para mapeamento de MPs em NoCs, baseado nos trabalhos de (MURALI; DE MICHELI, 2004; MARCULESCU et al., 2009), bastante aceito na comunidade de pesquisa em NoCs. Assumindo-se que a aplicação é descrita através de blocos funcionais que se comunicam, a referida comunicação será representada por um grafo de caracterização de aplicação (*application characterization graph*, APCG) $G(C, A)$ onde cada vértice $c_i \in C$ representa um bloco funcional, também denominado usualmente de elemento processantes, MP, ou um *core*, e cada arco direcionado $a_{i,j} \in A$ representa a comunicação entre os vértices c_i e c_j . Para cada arco, o parâmetro $b(a_{i,j})$ indica a largura de banda de comunicação entre c_i e c_j , ou seja, a taxa de dados transmitidos. Um exemplo de grafo de aplicação pode ser observado na Figura 13(a), onde o volume de comunicação entre os núcleos é dado em *Megabits* por segundo. O volume pode descrever taxas máximas ou médias de acordo com a conveniência do usuário (MURALI; DE MICHELI, 2004).

A arquitetura da NoC será representado por um grafo direcionado da topologia $T(R, Ch)$, onde cada vértice $r_i \in R$ representa um roteador, e cada arco

¹ Soluções exaustivas para casos pequenos (e.g. uma NoC 2x2) podem ser possíveis, enquanto que para NoCs com maiores dimensões o tempo de resolução do problema é inaceitável.

direcionado $ch_{i,j} \in Ch$ representa o canal (*channel*) de comunicação entre os vértices r_i e r_j . Um exemplo de grafo de NoC 3x3 está representado na Figura 13(b). Para cada arco, o parâmetro $bw(ch_{i,j})$ indica a largura de banda de comunicação entre r_i e r_j . A largura de banda é a quantidade máxima de taxa de transmissão de dados que um canal da NoC suporta. Por exemplo, se na Figura 13(b) o canal entre r_1 e r_2 suporta 2MB/s e pacotes são transferidos entre os MPs associados aos roteadores r_1 e r_2 a 1,5 MB/s e o mesmo ocorre na comunicação entre MPs associados a r_1 e r_3 , a taxa total de transmissão de o pacotes será de 3MB/s e estará acima da largura de banda suportada.

Figura 13 - Exemplos de grafos de aplicação (a); da NoC (b); e da NoC mapeada (c).

Fonte: Autor.

Dadas as definições iniciais, o mapeamento é matematicamente definido pela função Ω , ilustrada na equação (1), e condicionada por (2) e (3).

$$\Omega: C \rightarrow R \quad (1)$$

$$r_j, \forall c_i \in C, \exists r_j \in R \quad (2)$$

$$|C| \leq |R| \quad (3)$$

A equação (1) associa cada MP C a um roteador R . A condição (2) determina que para cada MP c_i , existe um roteador associado r_j . Finalmente, a condição (3) mostra que o número de roteadores deve ser igual ou maior ao número de MPs, do contrário, os MPs adicionais não seriam mapeados. O mapeamento também deve obedecer às restrições de largura de banda da arquitetura, como descrito nas equações:

$$\forall ch_{i,j}, bw(ch_{i,j}) \geq \sum_{a_{k,l}} x_{i,j} \quad (4)$$

$$x_{i,j} = \begin{cases} b(a_{k,l}), & \text{se } ch_{i,j} \in Path(\Omega(c_i), \Omega(c_j)) \\ 0, & \text{do contrário} \end{cases} \quad (5)$$

As equações (4) e (5) garantem que para canal $x_{i,j}$ pertencentes ao caminho de comunicação, a capacidade largura de banda da NoC não será violada.

Assim, o mapeamento pode ser realizado segundo uma série de critérios de otimização, separados em duas categorias: métricas de desempenho (latência dos pacotes, largura de banda, qualidade do serviço, etc.), ou por métricas de custo (consumo de energia, ocupação de área, temperatura, etc.). Independentemente do critério adotado, o mapeamento pode ser obtido minimizando uma função custo. Serão utilizados dois exemplos de função custo; no primeiro exemplo, o objetivo do mapeamento é a redução do consumo de energia na comunicação, como descrito na equação (6) (MARCULESCU et al., 2009). No segundo, pela equação (7) (MURALI; de MICHELI, 2004) o objetivo é minimizar o custo de comunicação.

$$\min \left\{ \sum_{\forall a_{i,j}} b(a_{i,j}) \cdot E_{\text{bit}}(\Omega(c_i), \Omega(c_j)) \right\} \quad (6)$$

$$\min \left\{ \sum_{\forall a_{i,j}} b(a_{i,j}) \cdot dist(\Omega(c_i), \Omega(c_j)) \right\} \quad (7)$$

onde $E_{\text{bit}}(\Omega(c_i), \Omega(c_j))$ é a energia necessária para enviar 1 bit de informação a partir de onde o MP c_i foi mapeado até o nó onde o MP c_j foi mapeado, assim como $\text{dist}(\Omega(c_i), \Omega(c_j))$ é a distância em *hops* de dois módulos mapeados. Um exemplo de mapeamento é ilustrado na Figura 13(c), já com todos MPs associados aos roteadores.

2.4. Reconfiguração Dinâmica

A principal característica que distingue os FPGAs das soluções ASIC e GPP é a sua configurabilidade ou programabilidade. A evolução dos FPGAs levou ao surgimento de uma série de implementações com estratégias diferentes de configuração.

A área de pesquisa de arquiteturas reconfiguráveis data do início dos anos 90 e está ainda em constante desenvolvimento. Após anos de pesquisa e desenvolvimento, ainda não existe uma terminologia unificada totalmente aceita neste campo. Muitos trabalhos têm utilizado termos como configuração, reconfiguração, reconfiguração estática, reconfiguração dinâmica, reconfiguração parcial, entre muitos outros, de maneiras distintas com significados distintos, gerando conflitos de interpretação. Serão definidos nesta seção os tipos de reconfiguração e as respectivas terminologias associadas, que serão adotados neste texto.

Uma classificação interessante, baseada na configurabilidade dos dispositivos foi proposta no trabalho de Lysaght e Dunlop (1993) e depois atualizado por Zhang e Ng, 2000. Por definição, todos os FPGAs são programáveis (ou configuráveis); existem aqueles FPGAs que, particularmente, são enquadrados apenas nesta categoria, que são aqueles baseados em tecnologia *antifuse*². Com o desenvolvimento de FPGAs com tecnologia E2PROM, FLASH e SRAM, ficou estabelecida a categoria dos dispositivos que podem ser reconfigurados

² Esta tecnologia permite apenas uma configuração (i.e., uma vez configurado o circuito permanecerá naquele FPGA).

indefinidamente³, porém mantendo essa configuração durante toda a execução. Para a implementação de cada nova reconfiguração, o dispositivo deve ser retirado de sua operação. Esse conceito de reconfigurabilidade ficou posteriormente conhecido como reconfiguração estática (ZHANG; NG, 2000).

A tecnologia SRAM também gerou outra possibilidade de dispositivos: uma reconfiguração seletiva (ou reconfiguração parcial), onde é possível reconfigurar uma área do FPGA sem modificar os circuitos programados nas áreas restantes. A rigor, do ponto de vista formal, a reconfiguração parcial poderia ser plenamente do tipo estático. Na prática, o dispositivos comerciais com reconfiguração parcial realizam-na em tempo de execução (ou reconfiguração dinâmica), onde não é necessário interromper o funcionamento do FPGA, para carregar um novo *bitstream*, ou seja, o FPGA é reconfigurado em uma área específica, enquanto outras áreas do FPGA continuam ativas, executando outras operações.

A Figura 14 detalha o mecanismo da reconfiguração dinâmica parcial ilustrando a atualização seletiva das células lógicas de um FPGA e os recursos de roteamento a ela associados, enquanto que outras áreas do FPGA continuam ativas, sem interrupção.

Figura 14 - Reconfiguração Dinâmica Parcial.

Fonte: (LYSAGHT; STOCKWOOD, 1996).

³ A tecnologia E2PROM possui um limite de ciclos de reconfiguração.

O conceito da diferença entre reconfiguração estática e reconfiguração dinâmica adotado neste trabalho será ilustrado na Figura 15. Considere três tarefas a serem executadas “TR1”, “TR2” e “TR3”. Estas tarefas são independentes entre si. No caso de reconfiguração estática Figura 15(a), o FPGA configuraria a tarefa 1 e depois esta seria executada. Após a execução o FPGA baixaria um *bitstream* que reconfiguraria toda a sua lógica interna e implementaria a tarefa 2. Em seguida o mesmo procedimento seria executado para a reconfiguração e execução da tarefa 3.

Figura 15 - Reconfiguração Estática (a) X Reconfiguração Dinâmica (b)

Fonte: Autor.

Para a exposição do comportamento da reconfiguração dinâmica, vamos considerar a Figura 15(b), onde há duas partições reservadas internamente no FPGA, de forma que uma fique em modo de execução enquanto a outra fica em modo de reconfiguração. Na primeira partição seria configurada a tarefa 1 e em seguida esta seria executada ao mesmo tempo em que a tarefa 2 seria reconfigurada. Após o término da reconfiguração da tarefa 2 esta seria executada ao mesmo tempo em que a tarefa 3 seria reconfigurada. Por fim a tarefa 3 seria executada. Deste modo a reconfiguração é feita *on-line* e de modo parcial.

Pode-se entender que um dispositivo FPGA com reconfiguração dinâmica deve ser capaz de realizar reconfiguração parcial, mas um FPGA com tal capacidade não precisa funcionar necessariamente com reconfiguração dinâmica.

Ou seja, a reconfiguração parcial é uma característica do dispositivo, enquanto a reconfiguração dinâmica é uma característica de operação do dispositivo.

A estratégia de reconfiguração dinâmica tem o potencial de otimizar o uso de área, pois, reconfigurando repetidas vezes uma mesma partição do FPGA, a quantidade de recursos utilizados (para processamento real) supera a quantidade de recursos existentes. A remoção de um módulo que não tem papel ativo por algum tempo em uma computação, permitirá disponibilizar a área correspondente para novas funções. Cada mudança de blocos no dispositivo cria um contexto de reconfiguração, que corresponde à uma microarquitetura específica que realiza uma determinada computação em um intervalo de tempo.

Ao contrário da arquitetura clássica Von Neumann, o FPGA permite a exploração da simultaneidade, sendo possível realizar várias tarefas em paralelo. Isso pode ser explorado na reconfiguração dinâmica de forma a mascarar o tempo de reconfiguração, reduzindo o tempo total de resposta, como se pode observar na Figura 15.

No presente trabalho os seguintes termos serão definidos:

- **Reconfiguração Estática** – Reconfiguração que é carregada no FPGA e nele permanece por toda a duração da aplicação. O termo reconfiguração total será considerado sinônimo.
- **Reconfiguração Parcial** – Reconfiguração seletiva das células lógicas de uma FPGA e dos recursos de roteamento a elas associados, enquanto outras áreas do FPGA continuam ativas ou não.
- **Reconfiguração Dinâmica** – Reconfiguração em tempo de execução (*run-time*) do FPGA, das células lógicas e dos recursos de roteamento a elas associados, enquanto outras áreas do FPGA continuam ativas. Serão considerados sinônimos: reconfiguração parcial dinâmica; *run-time Reconfiguration* – RTR (LYSAGHT; DUNLOP, 1993), *on-line reconfiguration*, *logic caching* (ZHANG; NG, 2000).
- **Sistema Dinamicamente Reconfigurável⁴ – SDR** – Sistema implementado em FPGA, ou não, que inclua reconfiguração dinâmica. Será considerado

⁴ Termo também adotado por (SOARES, 2005), (ESKINAZI, 2005), (RAFFO, 2010).

termo equivalente a Sistema Parcialmente Reconfigurável – SPR (HORTA, 2002).

- **FPGA Dinamicamente Reconfigurável – DRFPGA** – FPGA que inclui infraestrutura de reconfiguração parcial dinâmica.

2.5. Particionamento e Escalonamento de processos em SDRs

O particionamento é um aspecto muito importante no fluxo de projeto de SDRs. Basicamente, a tarefa de particionamento consiste na decisão de quais tarefas serão alocadas em uma partição do sistema reconfigurável. Já o escalonamento define a sequência em que as tarefas serão executadas e as dependências entre elas, o que pode determinar quais tarefas podem ser executadas em paralelo e quais devem esperar a resolução de outras para iniciar suas operações. Na Figura 16 Bobda (2007) mostra dois exemplos de particionamento.

Figura 16 - Exemplo de particionamento de tarefas.

Fonte: (BOBDA, 2007).

Um APCG é dado na Figura 16(a), e a partir dele são criadas duas partições (P0 e P1). A Figura 16(b) mostra uma abordagem onde há seis arcos entre as partições. Já a Figura 16(c), com um rearranjo dos vértices, possui apenas duas conexões entre as partições tornando-a muito mais eficiente em termos de comunicação, dada pela quantidade de dados a serem armazenados e transferidos.

Um exemplo de escalonamento do tipo *Forced-Directed List Scheduling* (PAULIN; KNIGHT, 1989) é mostrado na Figura 17 para operações, porém, seria válido para tarefas também. O escalonamento da Figura 17 é baseado em uma lista de prioridades, onde as tarefas prioritárias são alocadas primeiro. Em cada linha, é delimitado o tempo de execução das tarefas. Tarefas alocadas paralelamente podem ser executadas simultaneamente.

Figura 17 - Exemplo de escalonamento.

Fonte: (BOBDA, 2007)

Em (BOBDA, 2007) são relacionados estes e outros algoritmos de particionamento e escalonamento voltados à reconfiguração dinâmica. Juntas, as tarefas de particionamento e escalonamento definem quais tarefas operarão em blocos fixos e quais em reconfiguráveis, o que pode impactar muito significativamente na arquitetura e no desempenho do SDR.

2.6. SDRs baseados em redes intrachip

O foco do presente trabalho é em SDRs baseadas em redes intrachips. Este tipo de arquitetura será denominado SDR-NoC. Nesta seção serão caracterizadas e classificadas as arquiteturas SDR-NoCs. Até o momento, não se encontra este tipo de caracterização na literatura, sendo uma contribuição original deste trabalho.

Um SDR-NoC pode ser interpretado como um tipo de aplicação em forma de processos interligados como demonstrado na Figura 16(c). Esta interligação ocorre por meio de roteadores, que, por sua vez, estão interligados entre si, formando uma rede de roteadores. Estes roteadores seguem as diretrizes apresentadas na seção 2.2. O esquema de particionamento desta proposta é ilustrado na Figura 18:

Figura 18 - Esquema de particionamento para uma arquitetura SDR-NoC.

Fonte: (BOBDA, 2007), Autor.

Nota-se que os blocos podem representar tanto um processo como uma roteador. Neste contexto, existem duas possibilidades: a) Os roteadores não são incluídos no processo de reconfiguração: Neste caso, se o ponto de partida é uma topologia de rede intrachip regular, todos os contextos ou cenários serão regulares; b) Os roteadores são incluídos no processo de reconfiguração: Neste caso, a reconfiguração levará a cenários com topologias irregulares. Em relação ao tamanho dos MPs eles podem ser, como descrito na seção 2.2.1: a) homogêneos ou b) heterogêneos.

Assim, as arquiteturas SDR-NoC serão classificadas em dois tipos:

Arquitetura SDR-NoC Simples

- NoC regular.
- Topologia direta.
- MPs homogêneos.
- Roteadores não incluídos no processo de reconfiguração.

Arquitetura SDR-NoC Complexa

- NoC irregular
- Topologia indireta
- MPs heterogêneos.
- Roteadores incluídos no processo de reconfiguração.

A Figura 19 ilustra os dois tipos de arquitetura SDR-NoC.

Fonte: Autor.

A Figura 19(a) mostra uma arquitetura SDR-NoC simples com todas as características que a define, significando que a mudança de cenário ocorre com as

trocas de MPs apenas (em branco). Já a Figura 19(b) mostra uma arquitetura SDR-NoC complexa, na qual os retângulos cinzas indicam o posicionamento de módulos heterogêneos que levam às propriedades descritas e que, na mudança de cenários, podem ser trocadas junto a roteadores, por outros roteadores e MPs.

Nas seções a seguir serão apresentadas duas das principais arquiteturas SDR-NoCs propostas na literatura. Existem outras arquiteturas que são, basicamente, variações das duas aqui apresentadas.

2.6.1. Arquitetura DyNoC

Bobda e outros (BOBDA et al., 2005) propuseram um sistema para implementação de SDRs em uma NoC com topologia do tipo malha. A arquitetura da rede pode ser observada na Figura 20(a). A rede é composta por roteadores (representados por círculos) interconectados entre si por quatro portas: leste, oeste, norte e sul e mais uma porta para a conexão com o EP (elementos processantes representados por quadrados azuis). Cada porta possui uma FIFO com capacidade de 32 bits de dados e 4 bits de endereçamento (a NoC proposta tem tamanho 4x4). O roteador também conta com um controlador central para o roteamento. A estrutura do roteador pode ser observada na Figura 20(b):

Figura 20 - (a) Arquitetura DyNoC. (b) Estrutura do roteador.

Fonte: (BOBDA et al., 2005)

Pode-se observar que, caso todos os EPs tivesse tamanhos iguais e fossem simplesmente substituídos por outros EPs com o mesmo tamanho, a DyNoC poderia ser classificada, de acordo com a seção anterior, como uma arquitetura SDR-NoC simples.

A estratégia apresentada por Bobda (2005), no entanto, é tratar de casos de arquiteturas SDR-NoC complexas, alocando módulos reconfiguráveis sobre a área antes ocupada por roteadores. A Figura 21 ilustra a alocação de quatro blocos reconfiguráveis na DyNoc (observe-se que trata-se da Figura 3 a qual aqui replicamos para facilitar a leitura). Quando o módulo é alocado, os roteadores que estavam relacionados aos EPs locais são desativados e os roteadores na vizinhança desativam as portas que antes estavam conectadas com eles. Cada módulo reconfigurável é associado ao roteador que está ao seu respectivo canto superior direito (roteadores nos mesmos EPs dos módulos na Figura 21).

Os módulos são divididos nos EPs da área ocupada (e.g. o módulo C4 ocupa os seis EPs do canto superior esquerdo da Figura 21) e a comunicação interna destes EPs é feita através de barramentos internos. Este processo caracteriza uma arquitetura SDR-NoC complexa.

Figura 21 - Alocação de blocos reconfiguráveis na DyNoC.

Fonte: (BOBDA et al., 2005)

A comunicação é feita através de pacotes e o algoritmo de roteamento é uma variação adaptativa do algoritmo XY, denominada *surrounding XY* (SXY), que detecta quando um roteador está ativado e contorna o módulo alocado para chegar ao destino. A Figura 22 ilustra o processo de roteamento quando há um obstáculo entre a origem (à direita) e o destino (à esquerda). No caso, dois caminhos podem ser utilizados para contornar o obstáculo: caminho A e caminho B. Sendo o caminho A o mais curto, este é selecionado.

Figura 22 - Roteamento *surrounding XY*.

Fonte: (BOBDA et al., 2005)

2.6.2. Arquitetura CuNoC

Posteriormente, Jovanovic (JOVANOVIC et al., 2007, 2008) desenvolveu uma arquitetura SDR-NoC complexa, denominada *Communication unit Network on Chip* (CuNoC). A arquitetura proposta substitui os roteadores tradicionais por duas novas estruturas denominadas unidade de comunicação (UC) e unidade de comunicação gerenciadora de caminho (UCg).

As UCs, diferentemente dos roteadores tradicionais, possuem apenas quatro portas: leste, oeste, norte e sul, dispensando a quinta porta que tradicionalmente faz a comunicação com o EP. A UC possui uma unidade de roteamento e arbitragem,

que armazena os dados de entrada em um buffer central e direciona o pacote para uma saída específica. Uma série de multiplexadores e demultiplexadores completam o chaveamento interno da UC. A Figura 23 mostra em detalhes a estrutura da UC.

Figura 23 - Estrutura de uma unidade de comunicação.

Fonte: (JOVANOVIĆ et al., 2007)

Assim como no trabalho de Bobda (2005), o algoritmo SXY é utilizado como forma adaptativa de roteamento dentro da rede. O mecanismo de roteamento adotado foi o SAF. Como as UCs possuem apenas um *buffer* central (ao contrário de roteadores tradicionais, que possuem *buffers* em cada canal), é utilizada uma política de prioridade de curva a direita para o roteamento do pacote (por exemplo, se um pacote entrar pela porta do sul, a prioridade é que o pacote seja roteado a leste).

No caso de congestionamento das UCs, as UCgs são utilizadas para transferir os dados a uma UC livre. As UCgs possuem o dobro de *buffers* das UCs, podendo armazenar o dobro de pacotes e executar as transferências. Para isso cada saída da UC é conectada a uma UCg e vice-versa, em um arranjo ilustrado na Figura 24.

Figura 24 - Vizinhaça entre UCs e UCgs.

Fonte: (JOVANOVIĆ et al., 2007)

A abordagem do trabalho consiste em três etapas: a) a definição de uma área reconfigurável; b) implementação de uma rede com UCs e UCgs; c) substituição, através da reconfiguração, das UCs e UCgs pelos módulos reconfiguráveis, utilizando os chaveadores vizinhos para a comunicação. Estas etapas estão representadas na Figura 25. Quando se insere um módulo reconfigurável, deve-se ter uma estimativa da área a ser ocupada. Caso o tamanho do módulo seja menor ou igual à área de uma UC, basta substituí-los; se o módulo possui tamanho superior a UC, então o módulo deve substituir a área ocupada por duas ou mais UCs. A ocupação da CuNoC é ilustrada na terceira etapa da Figura 25.

Figura 25 - Etapas de implementação da CuNoC.

Fonte: (JOVANOVIĆ et al., 2007)

A proposta de Jovanovic (2007) é mais eficiente do que a de Bobda (2005) no que diz respeito ao quesito de comunicação estruturada para SDRs. No entanto, a arquitetura inclui vários problemas complexos que devem ser considerados pelo projetista do SDR. A ativação e desativação dos roteadores e a mudança dos espaços na NoC podem causar problemas como o bloqueio de roteamento. A Figura 26 mostra um exemplo deste tipo de bloqueio. Na alocação à esquerda da figura, foram alocados quatro EPs em volta do EP "A", de tal forma que não há UCs disponíveis para que o pacote chegue ao referido EP. Na alocação ilustrada à direita da figura, o arranjo dos EPs deixa rotas de UCs que conseguem alcançar o módulo anteriormente bloqueado.

Figura 26 - Bloqueio de roteamento na CuNoC.

Fonte: (JOVANOVIĆ et al., 2007)

2.7. Algoritmos

A proposta desta seção é apresentar uma breve teoria de alguns algoritmos que serão utilizados com mais ênfase neste trabalho. No capítulo 4 um algoritmo *Branch and Bound* será utilizado, enquanto que algoritmos genéticos serão utilizados no capítulo 5.

2.7.1. Branch and Bound

Branch and Bound (B&B) é um modo estruturado de busca inteligente de soluções de problemas de otimização (LAWLER; WOOD, 1966). Para melhor compreensão, o algoritmo B&B será utilizado para resolver o clássico problema do caixeiro viajante (PCV), como o exemplo ilustrado no livro de (BRASSARD; BRATLEY, 1988). O PCV a ser estudado possui cinco cidades com suas respectivas distâncias representadas na Tabela 2. Como se sabe, o PCV consiste em achar o trajeto mais curto para sair de uma determinada cidade, percorrer todas as outras cidades e retornar a cidade de origem.

Tabela 2 - Distância entre cidades do PCV.

Cidades	1	2	3	4	5
1	0	14	4	10	20
2	14	0	7	8	7
3	4	5	0	7	16
4	11	7	9	0	2
5	18	7	17	4	0

Fonte: Autor.

A representação de um problema para o algoritmo B&B consiste em uma árvore de busca com três tipos de nós: nó raiz, nó interno e nó folha. O nó raiz consiste no primeiro nó que dá origem à busca. Os nós internos são soluções

incompletas que representam a ramificação da árvore. Por fim, os nós folhas são representações das soluções completas.

A Figura 27 ilustra o conceito dos nós e a representação da árvore de busca. Do nó raiz se tem a primeira ramificação que é a partida da cidade 1. Como possíveis destinos, existem as cidades 2, 3, 4 e 5, que geram quatro ramificações. Os nós internos mostram a rota incompleta (de 1 para 2, de 1 para 3, etc.). Tomando por exemplo⁵ o nó "1,3" as possíveis ramificações são viagens para as cidades 2, 4 e 5. Finalmente, a quarta linha de ramificações gera soluções completas (nós-folha) que podem ter o custo de viagem avaliado com precisão.

Figura 27 - Árvore de busca para o PCV.

Fonte: (BRASSARD; BRATLEY, 1988)

A proposta do algoritmo B&B é fazer uma busca seletiva dentro da árvore. A cada nó, um limite (*bound*) é calculado de acordo com os possíveis valores dos

⁵ Por limitação de espaço, optou-se por não ilustrar completamente a árvore de busca.

custos que o problema pode atingir futuramente na árvore de busca. Se um *bound* mostra que todas as futuras ramificações daquele nó são necessariamente piores do que a melhor solução encontrada até o momento, então a exploração destas ramificações é desnecessária, i.e., o nó pode ser podado, e suas ramificações descartadas (BRASSARD; BRATLEY, 1988).

O cálculo dos *bounds* deve ser feito com uma métrica que garanta que as ramificações terão, no mínimo, aquele determinado valor. Aqui pode-se adotar a seguinte métrica: a metade da distância entre as cidades i e j é contada quando parte-se de i e a outra metade é somada quando chega-se em j (BRASSARD; BRATLEY, 1988). Por exemplo, deixar o nó 1, custará pelo menos 2, pois é a metade do menor valor de partida ($14/2$, $4/2$, $10/2$ e $20/2$). Analogamente, visitar a cidade 2 custará pelo menos 6 (mínimo de $5/2$ na chegada e mínimo de $7/2$ para a partida).

Para calcular uma viagem completa, é necessário contar a partida de uma cidade, uma passagem em todas as outras cidades e o retorno para a primeira cidade. Como exemplo, o *bound* do primeiro nó pode ser calculado com a seguinte soma: $2+6+4+3+3+2=20$. Note que esse cálculo não implica que exista uma solução com custo 20, apenas é uma métrica para garantir que os nós ramificados custarão no mínimo 20.

A medida que as ramificações são construídas, *bounds* mais detalhados podem ser calculados, e.g., o nó "1,2" pode ter a primeira viagem calculada com exatidão: da cidade 1 para a 2, temos um custo de 14. A viagem restante pode ser estimada da forma anterior: $7/2+11/2+3+3+2=17$. Daí conclui-se que o *bound* deste nó possui um valor $14+17=31$.

Agora será analisada a Figura 27 com mais atenção. Olhando a segunda geração de nós internos, conclui-se que o nó mais interessante é o "1,3" com custo 24. A escolha do nó de menor custo (*bound*) é um dos aspectos fundamentais do algoritmo B&B. Refere-se ao caminho mais promissor, com probabilidade de levar à solução final de menor custo. Entre as ramificações do nó "1,3", o menor *bound* é também 24, através do nó "1,3,2". Finalmente encontra-se o nó-folha com custo real de 31, que passa a ser o critério de poda. Se é fato que os nós "1,3,5", "1,4,2" e "1,4,3" terão, no mínimo, valores maiores que 31, não serão necessárias explorações adicionais nos mesmos.

Contudo, se forem explorados os nós adicionais, chegar-se-á a conclusão que a melhor solução consiste no nó-folha "1,4,5,2,3,1" com custo final 30.

O PCV é reconhecidamente um problema NP-completo e é evidente que, dependendo dos dados a solução só será dada em tempo exponencial no número de cidades consideradas. O algoritmo B&B tem a sua eficiência baseada na quantidade de caminhos que podem ser desconsiderados (ramos da árvore podados) durante a análise da função-custo (*bound*) nos diversos níveis de decisão da árvore. Esta capacidade depende do problema a ser resolvido e da modelagem realizada para a função-custo. A função-custo ideal, usada como *bound*, deve ser a mais próxima possível do valor real (para o corte de alternativas) e, preferencialmente, ter uma variação razoavelmente constante pelos diversos níveis de busca (para uma boa escolha de solução mais promissora).

2.7.2. Algoritmos genéticos

Algoritmos genéticos (Genetic Algorithms, GAs) são uma família de algoritmos que imitam os mecanismos da biologia evolucionária para otimização de problemas. São métodos robustos que podem lidar com uma grande variedade de problemas. Um GA clássico é composto de quatro etapas principais (GOLDBERG, 1989):

1. Geração de uma população aleatória.
2. Seleção dos indivíduos mais aptos.
3. Cruzamento (*crossover*) dos indivíduos selecionados.
4. Mutação.

Os passos 2, 3 e 4 são repetidos até formar uma nova população do mesmo tamanho da população inicial e a população antiga é substituída (eliminada). Cada nova população compõe uma geração. A cada nova geração a população é refinada, afim de atingir um objetivo.

Para adaptar um problema a um GA, é necessário primeiramente codificá-lo. A menor unidade de informação é chamada de gene, e uma solução, formada por um conjunto de genes, é chamada de cromossomo.

Para uma melhor compreensão de cada etapa, será utilizado um exemplo para ilustração dos processos. O exemplo foi baseado em um estudo de caso retirado de um texto informal⁶. Ele consiste de uma expressão matemática (E) utilizando cinco números inteiros entre 0 e 9 e operações aritméticas básicas entre eles, como por exemplo: $E=4+3-5*2/1$. O objetivo é procurar uma expressão que resulte em um número alvo. Tome-se como exemplo o número alvo 18. Algumas das possíveis expressões para atingir o número alvo seriam $E=3*9/1+0-3=18$, $E=9+9-2+2*1=18$ entre outros.

2.7.2.1. Codificação e geração de população aleatória

Para codificar este caso pode-se escolher uma representação binária para cada possibilidade, representada na tabela abaixo, onde a coluna da esquerda representa a codificação binária para sua interpretação na coluna da direita.

Tabela 3 - Codificação binária para GA

Gene	Interpretação
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	+
1 0 1 1	-
1 1 0 0	*
1 1 0 1	/

Fonte: Autor.

⁶ Texto disponível em <http://www.ai-junkie.com/ga/intro/gat1.html> (acesso em junho de 2014).

A Tabela 4 mostra como uma expressão matemática seria codificada e representada por um cromossomo:

Tabela 4 - Codificação de um cromossomo.

Expressão	2	+	3	-	0	/	1	*	5
Cromossomo	0010	1010	0011	1011	0000	1101	0001	1100	0101

Assim, a expressão matemática $2+3-0/1*5$ é representada pelo cromossomo 001010100011101100001101000111000101.

O processo de decodificação funciona, obviamente, de forma invertida. Porém é importante observar que alguns cromossomos podem gerar casos inválidos, como, por exemplo, no cromossomo ilustrado abaixo:

Tabela 5 - Decodificação de um cromossomo inválido.

Cromossomo	0001	1011	0111	1010	0011	0100	0001	1100	1111
Expressão	1	-	7	+	3	4	1	*	N/A

No cromossomo acima existem duas irregularidades: a primeira é o número quatro entre os números três e um. O esperado é que nesta posição exista um operador e não um número; a segunda irregularidade é o último gene do cromossomo que possui um código não aplicável. Durante a execução de um algoritmo genético, os cromossomos inválidos podem ser ignorados.

Depois de definida a codificação, uma população pode ser gerada, neste caso, simplesmente criando sequências aleatórias de zeros e uns no limite dos 36 caracteres que determinam o tamanho de um cromossomo. Os casos inválidos podem ser excluídos. A população inicial é definida de acordo com a necessidade. Quanto maior o número de indivíduos de uma população, também será maior o espaço de buscas, fazendo com que o esforço computacional também aumente significativamente.

2.7.2.2. Seleção dos Indivíduos mais aptos

Primeiramente, para todos os n indivíduos de uma população de tamanho N , um peso é atribuído. Quanto maior este peso, maior a chance de este indivíduo sobreviver. A função que calcula este peso é conhecida como *fitness function*. No exemplo atual, a função *fitness* deve aumentar a medida que o resultado da expressão se aproximar do número alvo, portanto a equação adequada seria:

$$fitness(n) = \frac{1}{|E - Na|} \quad (8)$$

onde Na é o número alvo da busca. Quando $E = Na$, o número alvo é encontrado.

Há vários tipos de seleção na literatura. Uma listagem das metodologias clássicas de seleção pode ser encontrada em (GOLDBERG, 1989; DAVIS, 1991). Cada tipo de seleção é adequado ao tipo de problema. Neste estudo de caso, uma seleção adequada seria a seleção por giro de roleta ou *roulette wheel selection*, na qual, para cada cromossomo n a probabilidade de ser selecionado é proporcional ao seu valor *fitness*:

$$P(n) = \frac{fitness(n)}{\sum_{m=1}^N fitness(n)} \quad (9)$$

2.7.2.3. Crossover e Mutação

Após a seleção de dois cromossomos, eles geram um novo cromossomo (filho) através de um processo conhecido como *crossover*. Este filho, possui características dos dois cromossomos que o originaram (pais). Novamente, existem vários tipos de *crossover* na literatura que serão discutidos oportunamente. Entre estes podem-se citar *one-point crossover*, *two-point crossover*, *cut and splice*, *uniform crossover*, *ordered crossover*, *partially matched crossover*, entre outros (GOLDBERG, 1989; DAVIS, 1991).

A probabilidade de *crossover* é arbitrária e fica tipicamente entre 0,5 e 1 (SRINIVAS; PATNAIK, 1994).

Aqui, aplica-se a técnica mais básica de *crossover* onde os cromossomos simplesmente doam parte de seu código genético aos seus filhos. Este processo é ilustrado na Figura 28.

Figura 28 - Crossover simples.

1 0 0 1	1 0 1 1	0 0 1 0	1 1 0 0	0 1 1 0	1 0 1 0	0 0 0 0	1 0 1 0	0 0 1 1
9	-	2	*	6	+	0	+	3

(a)

0 1 1 0	1 0 1 0	0 1 0 1	1 1 0 0	0 1 0 0	1 1 0 1	0 0 1 0	1 0 1 0	0 0 0 1
6	+	5	*	4	/	2	+	1

(b)

1 0 0 1	1 0 1 1	0 0 1 0	1 1 0 0	0 1 0 0	1 1 0 1	0 0 1 0	1 0 1 0	0 0 0 1
9	-	2	*	4	/	2	+	1

(c)

Fonte: autor.

A Figura 28(a) e (b) mostra os dois cromossomos selecionados para o *crossover*, onde os bits sombreados representam a parte do código genético que será doada ao filho. O resultado do *crossover* é ilustrado na Figura 28(c), onde o filho é gerado com características misturadas dos pais.

A mutação é um processo mais raro. A probabilidade de mutação também é arbitrária e varia entre 0,001 e 0,05, tipicamente (SRINIVAS; PATNAIK, 1994). O processo de mutação aplicado no exemplo poderia ser representado pela simples inversão de um bit, como mostra a Figura 29:

Figura 29 - Exemplo de mutação.

1 0 0 1	1 0 1 1	0 0 1 0	1 1 0 0	0 1 1 0	1 0 1 0	0 0 0 0	1 0 1 0	0 0 1 1
9	-	2	*	6	+	0	+	3

(a)

1 0 0 1	1 0 1 1	0 0 1 0	1 1 0 1	0 1 1 0	1 0 1 0	0 0 0 0	1 0 1 0	0 0 1 1
9	-	2	/	6	+	0	+	3

(b)

Fonte: autor.

Note-se que a simples inversão de um bit, muda a operação de multiplicação para divisão. A mutação é um recurso que evita que o algoritmo fique preso em mínimos locais, variando o espaço de procura.

2.7.2.4. Evolução do algoritmo

Como exemplo de evolução do exemplo citado, será analisada uma busca por indivíduo cuja expressão resulte em 41. Suponha-se que em determinado momento da execução do GA existam três elementos entre a população com as características ilustradas na Figura 30:

Figura 30 - Exemplo de três indivíduos durante a evolução do algoritmo.

0 0 0 0	1 1 0 1	0 0 1 1	1 0 1 1	0 0 1 1	1 0 1 0	1 0 0 0	1 1 0 0	0 1 0 1
0	/	3	-	3	+	8	*	5

(a)

0 0 0 1	1 1 0 0	0 1 0 0	1 0 1 1	0 0 1 0	1 0 1 0	0 1 1 1	1 1 0 0	0 1 1 0
1	*	0	+	2	+	7	*	6

(b)

0 1 1 1	1 1 0 0	1 0 0 1	1 0 1 0	0 1 1 0	1 1 0 0	1 0 0 0	1 0 1 1	0 0 0 0
7	*	9	+	6	*	8	-	0

(c)

Fonte: autor.

Assim o indivíduo (a) possui resultado 38 e *fitness* 0,333 (calculado de acordo com a equação (8)); o indivíduo (b) possui resultado 44 e *fitness* 0,333; já o indivíduo (c) possui resultado 111 e *fitness* 0,01428. Nesse contexto, a probabilidade de seleção dos dois primeiros indivíduos entre os componentes da população é muito maior do que do terceiro; apesar da probabilidade ser menor, o GA permite que o terceiro seja ainda assim escolhido. Sendo os dois primeiros cromossomos os selecionados, o cruzamento iria prosseguir como ilustrado na Figura 31. Gerando um filho com resultado 43 e *fitness* 0,5, sendo a probabilidade deste ser selecionado maior do que a probabilidade dos pais serem selecionados.

Figura 31 - Exemplo de *crossover* durante a execução do algoritmo.

0 0 0 0	1 1 0 1	0 0 1 1	1 0 1 1	0 0 1 1	1 0 1 0	1 0 0 0	1 1 0 0	0 1 0 1
0	/	3	-	3	+	8	*	5

(a)

0 0 0 1	1 1 0 0	0 1 0 0	1 0 1 1	0 0 1 0	1 0 1 0	0 1 1 1	1 1 0 0	0 1 1 0
1	*	0	+	2	+	7	*	6

(b)

0 0 0 1	1 1 0 0	0 1 0 0	1 0 1 1	0 0 1 1	1 0 1 0	1 0 0 0	1 1 0 0	0 1 0 1
1	*	0	+	3	+	8	*	5

(c)

Fonte: autor.

Suponha-se que o filho foi selecionado em uma próxima geração para o processo de mutação, executado como na Figura 32:

Figura 32 - Exemplo de mutação durante a execução do algoritmo.

0 0 0 1	1 1 0 0	0 1 0 0	1 0 1 1	0 0 1 1	1 0 1 0	1 0 0 0	1 1 0 0	0 1 0 1
1	*	0	+	3	+	8	*	5

(a)

0 0 0 1	1 1 0 0	0 1 0 0	1 0 1 1	0 0 0 1	1 0 1 0	1 0 0 0	1 1 0 0	0 1 0 1
1	*	0	+	1	+	8	*	5

(b)

Fonte: autor.

A mutação mudaria um gene do cromossomo, que levaria a um dos possíveis indivíduos que satisfazem o objetivo da busca. Note-se que esta suposição é somente uma possibilidade de resultado dentre inúmeras para este exemplo.

A finalização do algoritmo pode ser definida quando se define um alvo, como o exemplo do número 41 exibido nesta seção, ou quando se limita a busca a um tempo ou a um número de gerações.

2.7.2.5. Técnicas de variação da população

Um conhecido problema dos GAs, é a convergência prematura para mínimos locais (HOLLAND, 1992). Para lidar com este problema, várias metodologias foram propostas para forçar a variação da população, mantendo-a diversa e saudável. Dentre muitas técnicas, foram selecionadas duas técnicas de variação de população bastante aceitas e que serão utilizadas neste trabalho: a geração aleatória de descendentes e o algoritmo genético adaptativo (*Adaptative Genetic Algorithm* - AGA).

A. Geração aleatória de descendentes (ROG)

Um dos fatores que leva à convergência prematura a mínimos locais é o grande número de indivíduos compartilhando o mesmo material genético. Caso isto ocorra, é grande a probabilidade da operação de *crossover* ter como entrada dois indivíduos idênticos, tornando o processo de reprodução em uma simples clonagem, e, conseqüentemente, em um processo ineficaz.

Para lidar com este problema, foi proposta a técnica de geração aleatória de descendentes (*Random Offspring Generation*- ROG) (ROCHA; NEVES, 1999), na qual antes de cada processo de cruzamento os códigos genéticos são verificados. Caso os pais sejam idênticos, um dos dois filhos é gerado de maneira aleatória.

É importante notar que o mecanismo utilizado na técnica ROG é diferente do mecanismo da mutação. Enquanto que, no processo de mutação, muda-se apenas um gene do cromossomo, no procedimento do ROG, cria um novo indivíduo com o cromossomo obtido inteiramente de forma aleatória.

B. Algoritmo Genético Adaptativo (AGA)

Uma das soluções mais bem-sucedidas para lidar com a convergência prematura para mínimos locais foi o algoritmo genético adaptativo (Adaptative Genetic Algorithm- AGA), inicialmente proposto por (SRINIVAS; PATNAIK, 1994). O AGA não possui probabilidades de mutação e de *crossover* fixas; estas probabilidades alteram-se de acordo com a evolução do algoritmo.

Uma boa evolução de um GA tem que possuir duas características: convergência para um mínimo (local ou global) e exploração ampla no espaço de busca. O equilíbrio entre estas duas características são controlados pela probabilidade de execução do *crossover* ($P(\text{crossover})$) e probabilidade de mutação ($P(\text{mutação})$). Para adaptar a probabilidade a situação momentânea da convergência, (SRINIVAS; PATNAIK, 1994) propuseram probabilidades que variam de acordo com os valores de *fitness*.

$$P(\text{crossover}) = k_1 \cdot \frac{f_{\max} - f'}{f_{\max} - \bar{f}} \quad (10)$$

$$P(\text{mutação}) = k_2 \cdot \frac{f_{\max} - f}{f_{\max} - \bar{f}} \quad (11)$$

onde $f_{\max}$ é o maior *fitness* da população, f' é o maior *fitness* entre os dois cromossomos que irão cruzar, $\bar{f}$ é a média do *fitness* de toda a população e f é o valor *fitness* do cromossomo que irá entrar em processo de mutação. k_1 e k_2 são parâmetros de controle do AGA, que devem variar entre 0 e 1.

Além disso, para não exceder 100% de probabilidade, as funções devem obedecer às seguintes restrições:

$$P(crossover) = k_3 \quad se \quad f' < \bar{f} \quad (12)$$

$$P(mutação) = k_4 \quad se \quad f < \bar{f} \quad (13)$$

onde k_3 e k_4 também são parâmetros de controle.

2.7.2.6. Benefícios e limitações

O uso de GAs é capaz de otimizar uma grande variedade de problemas, incluindo multidimensionais, não-lineares, não-contínuos e não-paramétricos (GOLDBERG, 1989, DAVIS, 1991). A eficiência e rapidez do GA depende fundamentalmente do número de elementos de uma geração, do número de gerações e da escolha da função *fitness*.

O uso de GAs também acelera consideravelmente o tempo de busca de uma solução, sendo que o crescimento do tempo de execução em relação ao tamanho do problema não é explosivo, sendo adequado para tratar problemas com dependência a instâncias em ordem exponencial.

Em algumas situações, a população pode convergir para um único tipo de solução, tornando os cromossomos muito parecidos, e, conseqüentemente, focando a otimização em um mínimo local. Como descrito na seção anterior, as técnicas de variação de população podem atenuar o problema de convergência prematura para um mínimo local.

Não há garantia absoluta de que um algoritmo genético irá achar um mínimo global. Quanto maior o tamanho do problema e o número de variáveis, menor a chance deste mínimo ser obtido.

3. TRABALHOS CORRELATOS

É objetivo deste capítulo apresentar os principais trabalhos correlatos a esta tese, evidenciando o estado da arte em relação a esta pesquisa. Primeiramente, serão abordados os trabalhos realizados na área de posicionamento. No campo de mapeamento, serão analisados tanto os trabalhos sobre mapeamento estático (quando definido em tempo de projeto), quanto os sobre mapeamento dinâmico (quando definido em tempo de execução), que podem, por sua vez, serem planos (sem reconfiguração dinâmica) ou com reconfiguração dinâmica.

3.1. Posicionamento de módulos em SDRs

Não existem muitas ferramentas de posicionamento para SDRs descritos na literatura. Um dos primeiros trabalhos que considerou o problema de posicionamento em SDRs foi apresentado por (AHMADINIA et al., 2004) para FPGAs com reconfiguração em tempo real (RTR), onde o ponto de partida é um esquema de processos particionados e escalonados. O sistema de posicionamento é composto de três partes: uma base de dados dos módulos, um gerenciador de solicitações de posicionamentos e o posicionador. Quando um novo módulo vai entrar em operação (em tempo de execução), um pedido é acionado no gerenciador, e o algoritmo analisa os espaços já alocados para determinar onde posicionar o novo módulo. O critério para a escolha do local de posicionamento é a otimização da comunicação entre os módulos. O trabalho foi atualizado em 2007 (AHMADINIA et al., 2007), onde foi incluído um método de controle para os espaços livres através de técnicas de geometria computacional.

No trabalho de (MORANDI et al., 2008) foi proposto um algoritmo B&B para posicionamento em tempo real em FPGAs, tendo novamente como alvo os SDRs. O ponto de partida, assim como no trabalho anterior, é um esquema de processos particionados e escalonados. Na codificação do algoritmo, cada nó representa uma região ocupada do dispositivo e o nó folha representa o retângulo vazio máximo

(*maximum empty rectangle* - MER), buscando uma solução otimizada. Uma vez achado o MER, os módulos mais comunicantes são posicionados nos espaços vazios com a menor distância de Manhattan para minimizar os custos de roteamento.

O trabalho de Mario Raffo (RAFFO, 2010) propõe um método de posicionamento para arquiteturas SDR-NoCs em FPGAs, considerando MPs heterogêneos e em tempo de compilação. Um algoritmo de posicionamento denominado *DynoPlace* foi proposto, que leva em conta os aspectos reais das famílias de DRFPGAs, tornando possível automatizar este processo na metodologia de projeto. A metodologia foi proposta para utilização em tempo de compilação e é realizada em cinco etapas, como na Figura 33.

Figura 33 - Tarefas específicas do algoritmo do DynoPlace. (RAFFO, 2010)

Fonte: (RAFFO, 2010)

No pré-processamento, as dimensões dos módulos são processadas e ordenadas. Na segunda etapa, o autor utiliza um algoritmo de escalonamento em lista (*list scheduling*) em que os módulos de mesma altura são posicionados sequencialmente nas filas, de baixo para cima e da esquerda para direita, começando pelos maiores módulos reconfiguráveis e seus módulos adjacentes e posicionando os módulos menores posteriormente. As três últimas etapas fazem as adaptações necessárias para se adaptar à arquitetura da DRFPGA específica. O trabalho não trata o problema de mapeamento e não foca na regularidade da rede, alocando MPs em um lado do dispositivo e os roteadores em outro.

No trabalho de (ITURBE et al., 2013) foi desenvolvido um sistema operacional confiável para reconfiguração em tempo real em FPGAs sem NoC, *Reliable Reconfigurable Real-Time Operating System* - R3TOS. Tendo como princípio as partições da aplicação, o R3TOS executa escalonamento e posicionamento de tarefas utilizando uma série de métricas. Os algoritmos de posicionamento são

denominados Compactação de Área Vazia (CAV) e Compactação de Volume Vazio (CVV). Os algoritmos focam em preservar o MER intacto para uso futuro no maior prazo possível, tentando posicionar as menores tarefas nas menores áreas vazias possíveis. A principal diferença entre a CAV e a CVV é que a última faz uma análise da linha do tempo e previne fragmentações futuras, mantendo os módulos unidos e, conseqüentemente, minimizando os custos de comunicação.

Embora estes trabalhos tenham realizado explorações complexas no assunto de posicionamento em SDR, os mesmos não podem ser aplicados em ambientes baseados em NoC (com exceção de (RAFFO, 2010)). Com exceção do R3TOS, os algoritmos de posicionamento consideram apenas o cenário instantâneo para o posicionamento, ignorando o cenário geral de todas as configurações.

3.2. Mapeamento de tarefas em NoCs

3.2.1. Mapeamento Estático

A forma mais tradicional de mapeamento em NoCs é para sistemas planos (sem reconfiguração dinâmica) e na modalidade estática. Em um mapeamento estático, a definição da associação de tarefas a roteadores é realizada em uma etapa anterior à da operação real do sistema. O problema do mapeamento foi discutido primeiramente em um trabalho de 2003 (HU; MARCULESCU, 2003), onde um algoritmo do tipo *branch and bound* (B&B), foi utilizado para encontrar uma solução otimizada. O algoritmo foi aplicado para mapear uma série de MPs em uma NoC do tipo malha, de forma que o consumo de energia total da comunicação seja minimizada. Ao mesmo tempo, o desempenho é otimizado, com o sistema de comunicação resultante respeitando os limites de largura de banda. Quando comparado com soluções *ad-hoc* a economia de energia foi em torno de 60%.

Lei e Kumar (LEI; KUMAR, 2003) apresentam um algoritmo genético para a solução do mapeamento. A aplicação é descrita como um grafo de tarefas obtido com o software *Task Graphs for Free* (TGFF) (DICK; RHODES; WOLF, 1998). A arquitetura alvo é uma NoC com topologia malha que interconecta um conjunto heterogêneo de MPs. O objetivo do mapeamento, neste caso, é minimizar o tempo

total de execução da aplicação. Para tanto, os autores inseriram modelos de atraso de comunicação na estimativa do tempo de execução.

O algoritmo de mapeamento NMAP (MURALI; DE MICHELI, 2004) utiliza um método analítico de mapeamento. Primeiramente, um mapeamento inicial é feito, aproximando as tarefas com maior requerimento de comunicação aos roteadores com maior número de vizinhos. Depois o algoritmo iterativamente procura melhorar a solução inicial com a troca de posição entre pares de vértices. A cada troca, o desempenho da rede é avaliada e caso o resultado seja melhor que os anteriores, a modificação é mantida.

Além de considerar a largura de banda como restrição o trabalho de Srinivasan e Chatha (SRINIVASAN; CHATHA, 2005) consideram também a latência da rede como limitante ao mapeamento. A NoC estudada foi do tipo malha e cada roteador possui atributos específicos como largura de banda máxima suportada e energia consumida em cada porta. A solução implementada é um algoritmo conhecido como *cutting tree* (CT), onde o grafo da aplicação é recursivamente dividido em um plano horizontal e outro vertical, até que a partição gere um vértice para cada nó da NoC. Foram realizados vários experimentos com aplicações de multimídia e os requisitos foram atendidos. A complexidade computacional do algoritmo é considerada relativamente baixa.

Marcon e outros (MARCON et al., 2007) aplicaram e compararam vários métodos para mapeamento visando a minimização de energia em NoCs com topologia do tipo malha. O método *Simulated Annealing* (SA) é uma abordagem estocástica que consiste em uma técnica de busca local probabilística (KIRKPATRICK; GELATT; VECCHI, 1983). O algoritmo possui dois laços aninhados: um laço externo que implementa uma técnica de busca global enquanto um laço interno busca um refinamento local. O laço interno otimiza mapeamentos obtidos do laço externo, trocando sucessivamente pares aleatórios de módulos (MPs) e armazenando o mapeamento com melhor resultado obtido.

O método *Tabu Search* (TS) (GLOVER, 1990) também é uma abordagem estocástica. Assim como o SA, o algoritmo possui dois laços aninhados. O laço interno troca pares de módulos aleatoriamente, procurando por uma posição que resulte em menos consumo de energia. Uma lista com todas as trocas distintas e possíveis de pares otimiza a busca, e.g. se existe na lista um par (A,B), não constará

na lista o par (B,A). O processo de busca acessa a lista de trocas de forma aleatória. Se uma troca economiza mais energia do que as anteriores, essa troca é armazenada. Quando o laço interno é encerrado e uma troca que economiza energia é encontrada, essa troca é armazenada em uma lista-tabu e removida da lista de trocas. O mapeamento então incorpora essa troca e passa ao próximo laço interno. Outros trabalhos também utilizam o *Tabu Search* como método de mapeamento (MURALI; BENINI; DE MICHELI, 2005; MANOLACHE; ELES; PENG, 2005).

O método *Largest Communication First* (LCF) é um método proposto pelos autores onde os MPs mais comunicantes são mapeados primeiro. Primeiramente, a importância de cada MP é calculada equilibrando a quantidade de comunicações com o volume das comunicações. Em seguida os módulos são mapeados do centro para as bordas de acordo com a importância de cada MP.

Finalmente, o método *Greedy Incremental* (GI) também possui dois laços aninhados. Dado um mapeamento inicial gerado randomicamente, o laço externo fixa pivôs para serem avaliados. O primeiro pivô é a posição equivalente ao MP posicionado no canto superior direito da NoC e a última está localizado no canto inferior direito. O pivô a ser analisado é incrementado a cada iteração do laço externo. O laço interno define qual troca de posição dos MPs leva a melhor economia de energia, trocando a posição de todos os módulos nas posições restantes. Estas posições incluem somente aquelas que ainda não foram escolhidas pelo laço externo como pivôs. Quando o laço interno é finalizado, a melhor alternativa de mapeamento em questão de economia de energia é armazenada e se uma troca de par está levando a melhores resultados, o algoritmo faz a troca permanentemente. Na próxima iteração, o algoritmo desconsidera os pivôs analisados anteriormente.

3.2.2. Mapeamento dinâmico

O mapeamento dinâmico é executado em tempo de execução e se faz necessário quando as tarefas a serem mapeadas são definidas em tempo de execução. Estes casos acontecem, por exemplo, em aplicações de rede e multimídia

que demandam tipos diferentes de processamento de acordo com a aplicação. Diferentemente do objetivo do presente trabalho, que trata SDRs, aqui o foco é, em geral, mapear tarefas de software em diferentes elementos processantes de um MPSoC. É importante considerar estes trabalhos, apesar de não tratarem de SDRs, porque o problema de mapear um número de tarefas maior que um número de recursos é similar ao problema a ser estudado nos capítulos posteriores.

Inicialmente, algoritmos de migração de tarefas (NOLLET et al., 2005; BERTOZZI et al., 2006) foram sugeridos para otimizar o desempenho destas aplicações. Esses algoritmos visam realocar dada tarefa quando gargalos são identificados, distribuindo uniformemente a carga de processamento entre os MPs do sistema. Posteriormente, foi proposto o método de mapeamento dinâmico, que, diferentemente do método de mapeamento estático e da migração de tarefas, pode inserir novas tarefas no sistema em tempo real (CARVALHO; CALAZANS; MORAES, 2007).

Ngouanga e outros (NGOUANGA et al., 2006) propuseram um mapeamento em um MPSoC uma NoC com topologia malha com MPs homogêneas, similar à estrutura Hermes apresentada em (MORAES et al., 2004), considerando aplicações em tempo real. Nenhuma informação é conhecida a priori sobre o mapeamento, o qual é executado de acordo com parâmetros de localidade tais como compartilhamento de canais de comunicação. Os MPs foram divididos em três categorias: mestre, escravos, e interface de entrada e saída. Ao MP mestre é atribuída a gerência do mapeamento, que foi testada com dois algoritmos implementados em software: SA e *Force Directed* (FD). O FD é um algoritmo inspirado em um fenômeno físico onde o equilíbrio do sistema é atingido quando a distância entre os pesos (análogos as tarefas) conectados por cordas (análogas aos arcos entre as tarefas) é mínimo. Assim, o algoritmo seleciona a posição para uma nova tarefa de acordo com uma "força de atração" proporcional ao volume de comunicação e a distância entre as tarefas. O tempo de execução do algoritmo FD é, em média, uma ordem de grandeza menor do que o tempo de execução do algoritmo SA, apesar do caminho entre duas tarefas ser bem similar nos dois algoritmos. As análises mostram que quanto maior o número de MPs na rede, melhor tende a ser o resultado dos algoritmos quando comparados com mapeamentos aleatórios.

A proposta de Wronski (WRONSKI; BRIÃO; WAGNER, 2006) é utilizar algoritmos do tipo *bin-packing* (GAREY; GRAHAM; ULLMAN, 1973) para o mapeamento dinâmico de tarefas. Aqui o contexto é a alocação de tarefas de software em MPSoCs. Dois algoritmos são avaliados: *Best Fit* (BF) e *Worst Fit* (WF). O primeiro método aloca uma nova tarefa no menor espaço remanescente, enquanto que a estratégia *Worst Fit* aloca a tarefa no maior espaço remanescente. Uma ferramenta desenvolvida em SystemC simula o comportamento dinâmico de um MPSoC e com o auxílio da biblioteca Orion (WANG et al., 2002), estima a energia consumida na comunicação. Cada tarefa é representada por seu pior tempo de execução e por seu número de chaveamentos. Os experimentos mostraram que duas alocações diferentes que possuem desempenho de tempo similar, podem resultar em grandes variações de consumo de energia.

Chou e Marculescu (CHOU; MARCULESCU, 2007) implementaram uma estratégia incremental para o mapeamento dinâmico de tarefas em MPSoCs em uma NoC homogênea. Os processadores são interconectados por uma NoC com topologia malha e roteamento XY. A arquitetura possui caminhos distintos para o fluxo de dados e fluxo de controle. Um processador gerente recebe informações de várias aplicações a serem executadas e tem a função de mapeá-las em regiões da NoC em que os MPs fiquem próximos e possam se comunicar mais facilmente, evitando a fragmentação do sistema. Experimentos foram realizados em um MPSoC com dimensões de 7x7 com tarefas sintéticas geradas pelo software TGFF. Foram gerados mapeamentos exaustivos e estes foram comparados com a proposta do trabalho; o método proposto apresentou penalidade de 21% no consumo de energia em relação aos exaustivos, no entanto, o tempo de mapeamento do algoritmo proposto foi da ordem de dezenas de microssegundos enquanto que no método exaustivo o tempo de mapeamento foi da ordem de dezenas de horas. Quando comparado com mapeamentos aleatórios, o método apresentou 45% de redução no consumo de energia.

O trabalho de Carvalho (2009) apresenta heurísticas para mapeamento em tempo real para MPSoCs heterogêneas. As tarefas são mapeadas em tempo de execução, sob demanda da aplicação. Os algoritmos implementados baseiam-se na ocupação dos canais da NoC para decidir a melhor posição para determinada tarefa. Foram propostas quatro heurísticas, denominadas *congestion-aware*, cujo intuito é a

redução do nível de congestionamento da NoC, evitando a perda de desempenho nas aplicações. A heurística *Minimum Maximum Channel Load* (MMCL) emprega como função custo a ocupação máxima em todos os canais da NoC, enquanto a *Minimum Average Channel Load* (MACL) avalia a ocupação média para todos os canais. *Path Load* (PL) e *Best Neighbor* (BN) são heurísticas que avaliam a ocupação apenas considerando os canais que pertencem ao caminho de comunicação entre a tarefa mestre, que é a solicitante, e a tarefa escrava, que é a solicitada. Os mapeamentos propostos tiveram redução em torno de 90% dos parâmetros de congestionamento, 31% da ocupação dos canais da NoC e 15% da latência dos pacotes.

Os trabalhos aqui descritos apresentaram vários métodos de mapeamento. Com exceção de (CARVALHO, 2009), todos os trabalhos focam em MPSoCs baseadas em NoCs homogêneas, que não são aplicáveis no caso de arquiteturas SDR-NoC complexas. Nenhum dos casos apresentados considera a reconfiguração dinâmica, o que impossibilita a aplicação dos métodos para arquiteturas SDR-NoCs tanto simples quanto complexas.

3.3. Mapeamento para SDRs baseados em NoCs

Até o momento, pouquíssimos trabalhos apresentaram soluções para mapeamento em SDRs baseados em NoCs. Entre eles pode-se citar (RANA et al., 2009; BERETTA et al., 2010, BERETTA et al., 2011).

O trabalho de (RANA et al., 2009) propõe uma metodologia de mapeamento que utiliza uma arquitetura diferente das vistas anteriormente. Diferentemente das propostas da DyNoC (BOBDA et al., 2005), CuNoC (JOVANOVIC, 2007), etc., a rede não altera sua topologia nem alterna o funcionamento de roteadores. O que alterna entre um contexto e outro são MPs (*cores*) dentro de *slots*. O espaço do FPGA (ou outro dispositivo reconfigurável) é dividido em grandes *slots* reconfiguráveis, que podem abrigar um ou mais MPs dentro de sua área. No trabalho foi proposta uma metodologia de mapeamento em tempo de compilação (estático).

Na proposta de RANA e outros, o SDR é multiaplicação. O processo de mapeamento tem como ponto de partida os grafos de comunicação das aplicações e foi dividido, basicamente, em três fases: pré-processamento, particionamento e o mapeamento em si. A ideia básica das fases de pré-processamento e particionamento é dividir a solução em duas partes: mapeamento básico e configurações específicas. O mapeamento base inclui o subconjunto dos MPs compartilhado pela maioria das aplicações a serem mapeadas. As configurações específicas são um conjunto de configurações que não podem ser reutilizadas a partir do mapeamento base.

Após a divisão da solução entre mapeamento básico e configurações específicas, o mapeamento é executado da seguinte forma: primeiramente um algoritmo genético é utilizado para alocar os MPs do mapeamento base. Após a definição do mapeamento base, um segundo mapeamento é executado para alocar os módulos pertencentes às configurações específicas. Este algoritmo utiliza uma combinação linear entre tamanho dos *slots*, número de vezes que o *slot* é reconfigurado e custo de comunicação para escolher o *slot* mais adequado para cada módulo a ser alocado.

O foco do trabalho é no mapeamento de diferentes aplicações no mesmo dispositivo com reuso de MPs. Toma-se, por exemplo, as aplicações MPEG-2 e H.263, cujos grafos estão representados na Figura 34: os MPs CPU, DCT, IDCT, Coder, memória, Q e motion estimation são compartilhados por ambas as aplicações. Os autores tomam proveito, para reutilizar os MPs compartilhados por diferentes aplicações em diversos mapeamentos.

A metodologia de (RANA et al., 2009) foi implementada e testada no trabalho de (BERETTA et al, 2011), cuja a aplicação testada é ilustrada na Figura 34. A figura mostra o grafo de comunicação de três aplicações que serão mapeadas em um mesmo dispositivo, onde os MPs são divididos em grandes e pequenos e a comunicação entre eles é classificada como crítica (muito comunicativa) ou não-crítica (pouco comunicativa).

A

Figura 35 (BERETTA et al, 2011) ilustra o resultado do mapeamento estático ao longo do tempo. Os *slots* verdes representam o mapeamento base, enquanto os

amarelos representam as configurações específicas. Seis mapeamentos são mostrados: (a) e (b) ilustram a configuração para execução da aplicação MPEG-2, (c) e (d) para execução da aplicação H.263 e, finalmente, (e) e (f) para execução da aplicação MPEG-4. A sequência de figuras ilustra a execução sequencial das três tarefas no dispositivo. As subfiguras (a), (c) e (e) representam o primeiro cenário de configuração enquanto as subfiguras (b), (d) e (f) mostram o segundo cenário de configuração necessários para utilizar cada aplicação.

Figura 34 - Grafos de comunicação de (a) MPEG-2 (b) H.263 (c) MPEG-4.

Fonte: (BERETTA et al, 2011).

Figura 35 - Mapeamento estático para (a) e (b) MPEG-2 (c) e (d) H.263 (e) e (f) MPEG-4

Fonte: (BERETTA et al, 2011).

O trabalho de (BERETTA et al, 2011) utiliza a arquitetura e a metodologia de mapeamento estático proposto por (RANA et al., 2009), além de propor novas metodologias de mapeamento em tempo de execução (dinâmico). A processo de mapeamento é ilustrado na Figura 36, para os dois casos. No caso estático, o mapeamento de Rana é utilizado. No caso dinâmico foram propostas três diferentes

soluções: o reuso dos MPs existentes, algoritmo de satisfatibilidade booliana (SAT) (MOSKEWICZ et al., 2001) ou caso geral. O caso geral só é utilizado se novos MPs forem necessários durante a execução dos aplicativos.

Figura 36 - Processo de mapeamento (BERETTA et al, 2011).

Fonte: (BERETTA et al, 2011).

O mapeamento em tempo real, pode ser realizado em três processos, como já ilustrado na Figura 36. Se for possível reutilizar os módulos já prontos, uma análise da situação é feita com uma métrica imediatista (*greedy*), que pode ou não achar uma boa solução, realocando a posição dos *slots* através da realocação dos *bitstreams*. Caso seja necessário uma solução com maior qualidade, um algoritmo SAT, bem mais complexo e demorado, analisa os módulos e suas posições e acha uma solução completa. Caso seja(m) necessário(s) alocar novo(s) módulo(s), um terceiro algoritmo analisa como alocar os novos elementos na arquitetura, gerando uma pontuação para cada área do dispositivo, decidindo assim em qual posição descarregar o(s) novo(s) módulo(s).

A Figura 37 mostra um exemplo de mapeamento dinâmico onde os módulos MPEG-2 e H.263 são mapeados estaticamente e a aplicação MPEG-4 é adicionada em tempo de execução. A Figura 37(a) mostra o mapeamento base gerado para as aplicações MPEG-2 e H.263: todos os módulos necessários para o funcionamento destas duas aplicações estão presentes, não sendo necessária nenhuma reconfiguração. Na Figura 37(b) o mapeador dinâmico reusa quatro *slots* da configuração base (1, 2, 3 e 5) e os módulos necessários para o funcionamento do MPEG-4 são alocados nos *slots* 4 e 6.

Figura 37 - Mapeamento dinâmico para (a) MPEG-2 e H.263 (b) MPEG-4.

Fonte: (BERETTA et al, 2011).

Apesar destes trabalhos abordarem o problema de mapeamento em SDRs, estes possuem uma abordagem multitarefas, utilizando uma arquitetura específica, que não pode ser aplicadas nas arquiteturas NoC-SDRs complexas apresentadas na seção 2.6.

3.4. Comparação com os trabalhos de mapeamento correlatos

A Tabela 6 apresenta um quadro comparativo dos trabalhos investigados. A primeira coluna indica a referência e o ano do trabalho estudado; a segunda coluna informa em qual tipo de arquitetura de NoC o mapeamento é baseado; a terceira coluna descreve o tipo de configuração do sistema, ou seja, se é plano ou dinamicamente reconfigurável; a quarta coluna informa se o mapeamento proposto é estático ou dinâmico; a quinta coluna relaciona os algoritmos utilizados em cada

trabalho para realizar a tarefa de mapeamento; finalmente, a última coluna diz quais são as métricas que definem a otimização alvo do algoritmo de mapeamento. As duas últimas linhas da tabela apresentam a proposta deste trabalho de doutorado, a serem detalhadas nos capítulos 4 e 5.

Tabela 6 - Comparação dos trabalhos correlatos.

Trabalho	Arquitetura	Configuração	Mapeamento	Algoritmo	Métricas
(HU; MARCULESCU, 2003)	Clássica Regular	Plana	Estático	B&B	Consumo de energia.
(LEI; KUMAR, 2003)	Clássica Regular	Plana	Estático	Genético	Tempo de execução
(MURALI; DE MICHELI, 2004)	Clássica Regular	Plana	Estático	Analítico	Largura de banda
(SRINIVASAN; CHATHA, 2005)	Clássica Regular	Plana	Estático	CT	Largura de banda e latência
(MARCON et al., 2007)	Clássica Regular	Plana	Estático	SA, TS LCF e GI	Consumo de energia e tempo de execução
(NGOUANGA et al., 2006)	Clássica Regular	Plana	Dinâmico	SA e FD	Tempo de comunicação
(WRONSKI et al., 2006)	Clássica Regular	Plana	Dinâmico	BF e WF	Consumo de energia e deadlines
(CHOU; MARCULESCU, 2007)	Clássica Regular	Plana	Dinâmico	Analítico	Consumo de energia e fragmentação
(CARVALHO, 2009)	Clássica Regular	Plana	Dinâmico	MMCL, PL BN e MACL	Ocupação da NoC
(RANA et al., 2009)	SDR-NoC Simples	Reconfiguração Dinâmica	Estático	Genético	Largura de banda e reuso de MPs
(BERETTA et al., 2011)	SDR-NoC Simples	Reconfiguração Dinâmica	Estático e Dinâmico	Genético, SAT e analítico	Largura de banda e reuso de MPs
Mapeamento para SDRs Simples (Capítulo 4)	SDR-NoC Simples	Reconfiguração Dinâmica	Estático	B&B	Largura de banda
Mapeamento para SDRs Complexos (Capítulo 5)	SDR - NoCs complexas	Reconfiguração Dinâmica	Estático	Genético	Largura de banda

Fonte: Autor.

4. MAPEAMENTO PARA ARQUITETURAS SDR-NoC SIMPLES

Será apresentado primeiramente o estudo para a solução do problema de mapeamento em uma arquitetura SDR-NoC simples. Como definido nos capítulos 1 e 2, uma arquitetura SDR-NoC simples é um SDR estruturado em uma NoC regular, com topologia direta e MPs homogêneos.

Não é objetivo do presente trabalho definir como é feito o particionamento nem o escalonamento das partições reconfiguráveis. Parte-se do princípio que estas decisões já estarão definidas pelo projetista. O objetivo geral é propor uma estratégia de mapeamento para arquiteturas SDR-NoC simples.

Para provar a eficiência da estratégia, um algoritmo *branch-and-bound* (B&B) será desenvolvido para buscar a solução ótima. Uma vez que este algoritmo é semiexaustivo, ele só é aplicável ao número relativamente reduzido de MPs. Porém, uma vez demonstrada a eficiência da estratégia, será possível seguir trabalhos clássicos de mapeamento⁷, usando-se os algoritmos de mapeamentos planos ali propostos (sem reconfiguração) para SDRs.

4.1. Estratégia de mapeamento para arquitetura SDR-NoC simples

O modelo tem como ponto de partida o APCG com as partições fixas e reconfiguráveis já definidas. Tomando-se como base o modelo clássico apresentado na seção 2.3, uma estratégia para o mapeamento de blocos reconfiguráveis será apresentada.

Neste novo modelo, a restrição apresentada pela equação (3) (pág. 24), estabelecendo que o número de MPs a serem mapeados deve ser igual ou inferior ao número de roteadores disponíveis, será alterada. Em se tratando um sistema reconfigurável, é permitido o número de MPs superar o número de posições disponíveis para mapeamento, ou seja, $|C| > |R|$ (e.g. dado um APCG com 12 MPs

⁷ Como aqueles apresentados na seção 3.2.

e uma NoC 3x3, seis MPs podem ser fundidos reduzindo o número de blocos fixos e reconfiguráveis para nova, tornando o mapeamento possível).

Pela nova técnica de mapeamento, resolve-se o problema da restrição citada acima fundindo-se no APCG módulos que se reconfiguram entre si em um único módulo, fazendo com que o número de módulos efetivos a serem tratados diminua. A estratégia consiste em trazer para o módulo fundido toda a comunicação envolvida nos dois módulos que anteriormente estavam separados. Há duas situações possíveis: a) os módulos possuem origens (destinos) independentes; b) as origens (destinos) são comuns.

4.1.1. Fusão de módulos com origens (destinos) independentes

Tome-se como exemplo a Figura 38(a). Dos sete módulos existentes, é definido pelo projetista que os MPs R1 e R2 são reconfiguráveis entre si compartilhando o mesmo espaço físico. A Figura 38(b) ilustra o resultado da estratégia de fusão, na qual os módulos são simplesmente unidos, mantendo seus arcos originais. O exemplo evidencia que a independência de origens não afeta a taxa de comunicação a ser registrado no APCG: os módulos V e X do exemplo devem ser mapeados junto a diferentes roteadores, portanto, os arcos relativos à comunicação permanecem inalterados.

A análise para destinos independentes é similar, evidenciado pelo exemplo da Figura 38.

Figura 38 - Exemplo de fusão de MPs.

4.1.2. Fusão de módulos com origens (destinos) comuns

O caso da Figura 39(a) refere-se a uma situação mais complexa, pois a fusão implica na superposição de arcos, ou seja, das correspondentes taxas de comunicação. Tal condição ocorre uma vez que o mapeamento dos módulos reconfiguráveis fundidos, junto ao mesmo roteador (ocupando o mesmo espaço físico), implica que os pacotes associados a dois arcos originariamente independentes, passam a ocupar a mesma rota dentro de uma NoC. Esta condição será denominada conflito. É o caso de R1 e R2 da figura que, ao serem fundidos, devem ter os seus arcos de entrada e de saída tratados conjuntamente. Haverá um caso conflitante sempre que:

- 1) Dois módulos reconfiguráveis, c_i e c_j a serem fundidos, recebem dados $a_{k,i}$ e $a_{k,j}$ de um terceiro módulo k .
- 2) Dois módulos reconfiguráveis, c_i e c_j a serem fundidos, enviam dados $a_{i,k}$ e $a_{j,k}$ a um terceiro módulo k .

Figura 39 - Exemplo de fusão de MPs com conflito de arcos.

Fonte: Autor.

Nos casos onde há conflitos, um novo arco deve ser gerado para substituir os arcos conflitantes. O valor de largura de banda composta deve ser definido como a média ponderada dos dois arcos anteriores em relação ao tempo em que estejam

em operação. Um módulo reconfigurável só entra em operação quando uma outra deixa de operar. Baseado no exemplo da Figura 39, os novos arcos gerados na Figura 39(b) teriam as seguintes médias compostas, para os arcos à entrada e à saída, respectivamente:

$$b(a_{X,R12}) = \frac{b(a_{X,R1}) \times t_{R1} + b(a_{X,R2}) \times t_{R2}}{t_{R1} + t_{R2}} \quad (14)$$

$$b(a_{R12,Y}) = \frac{b(a_{R1,Y}) \times t_{R1} + b(a_{R2,Y}) \times t_{R2}}{t_{R1} + t_{R2}} \quad (15)$$

onde t_R é o tempo que uma tarefa reconfigurável leva para ser executada.

As equações (14) e (15) expressam o fato de que uma vez que módulos fundidos operam sequencialmente, a taxa de dados combinada irá depender do tempo em que cada módulo opera ativamente. As expressões podem ser facilmente estendidas para o caso de mais módulos reconfiguráveis para o mesmo espaço.

Apesar das métricas acima funcionarem adequadamente para propósitos de mapeamento, elas não são apropriadas para avaliar se os requerimentos de largura de banda máxima da arquitetura estão sendo respeitados. Para exemplificar este conceito, vamos supor que os canais da NoC do exemplo da Figura 39(b) suportem 100 Mb/s, e que os antigos arcos, agora unidos, possuíam a seguinte característica: $b(a_{X,R1}) = 200\text{Mb/s}$ com R1 sendo executado por 1s; $b(a_{X,R2}) = 5\text{Mb/s}$ com R2 sendo executado por 9s. Aplicando a fórmula (14), o valor obtido para a largura de banda composta seria de 24,5 Mb/s, o que poderia levar à conclusão errônea de que a fusão dos arcos não viola as restrições de largura de banda máxima, o que de fato ocorre individualmente para R1 com a taxa de 200 Mb/s. Neste caso, deve-se considerar o máximo do valor dos arcos, e não sua média ponderada. Tendo novamente como referência a Figura 39, os parâmetros de requerimento de largura de banda para a entrada e saída, após a fusão, seriam definidos como:

$$bw_req(a_{X,R12}) = \max(b(a_{X,R1}), b(a_{X,R2})) \quad (16)$$

$$bw_req(a_{R12,Y}) = \max(b(a_{R1,Y}), b(a_{R2,Y})) \quad (17)$$

As equações (16) e (17) funcionam para qualquer número de módulos reconfiguráveis. Com essa alteração, a equação (5) (pág. 25) do modelo clássico será substituída pela equação (18):

$$x_{i,j} = \begin{cases} bw_req(a_{k,l}), & \text{se } ch_{i,j} \in Path(\Omega(c_i), \Omega(c_j)) \\ 0, & \text{do contrário} \end{cases} \quad (18)$$

4.2. Aplicação da estratégia

Para validação da estratégia apresentada na seção anterior, serão adotadas três aplicações reais, que foram utilizadas em trabalhos clássicos anteriores: um decodificador VOPD (do inglês, Video Object Plane Decoder ou VOPD) e um decodificador MPEG4 (MP4), obtidos de (MURALI; De MICHELI, 2004; BERTOZZI et al., 2005) e um sistema multimídia (MultiMedia System - MMS), adaptado de (HU, MARCULESCU, 2005).

O VOPD é um *benchmark* clássico que possui 12 MPs, como apresentado na Figura 40(a). A Figura 40(b) mostra o processo de fusão, onde, nesta tese, seis módulos foram escolhidos para ocuparem três partições reconfiguráveis (representadas pelos retângulos cinzas), tornando possível o mapeamento desta rede em uma NoC 3x3. Neste caso específico, conflito de arcos como os ilustrados na Figura 39(b) não ocorrem, portanto, não é necessária a criação de novos arcos.

Figura 40 - Aplicação VOPD (a) Original (b) Fundida

Fonte: (MURALI; de MICHELI, 2004), Autor.

O grafo de caracterização original da aplicação MP4 é mostrado na figura Figura 41(a). O processo de fusão aconteceu da seguinte forma: o módulo MED_CPU foi unido ao RAST, o módulo AU foi fundido com VU, e, finalmente, os módulos RISC e IDCT também foram fundidos, como ilustrado na Figura 41(b). Aqui, o principal critério de escolha dos módulos para as partições reconfiguráveis foi evitar a fusão de dois módulos que se comunicam diretamente. Neste caso, seria necessário executar a computação do primeiro MP, salvar o resultado em uma memória externa, reconfigurar e então executar a segunda etapa do processamento.

Figura 41 - Aplicação MP4 (a) Original (b) Fundida

Fonte: (MURALI; de MICHELI, 2004), Autor.

O processo de fusão resultou em oito novos arcos gerados por conflitos. Estes arcos podem ser observados na Figura 41(b). Todos os novos arcos foram calculados de acordo com as equações (14) a (18). As métricas de tempo utilizadas nas equações serão explicadas mais adiante.

A aplicação MMS é um sistema integrado de áudio e vídeo. Originalmente, ele possui 16 MPs com oito processadores digitais de sinais (Digital Signal Processor - DSP), um CPU, três memórias e quatro ASICs. A arquitetura completa é mostrada na Figura 42(a) enquanto a Figura 42(b) mostra o processo de fusão, onde os módulos DSP foram reduzidos para seis e os quatro módulos ASICs foram agrupados em duas partições reconfiguráveis. Assim, é possível alocar a mesma aplicação em uma NoC 4x3 (ou 3x4), onde antes era necessário uma NoC 4x4. O processo de fusão resultou em dois novos arcos (em vermelho na Figura 42(b)), onde as equações (14) a (18) foram novamente aplicadas.

Figura 42 - Aplicação MMS (a) Original (b) Fundida

Fonte: (MURALI; de MICHELI, 2004), Autor.

Os dados relativos à troca de dados destas aplicações são abertos à comunidade científica, todavia, as descrições comportamentais dos sistemas não estão disponíveis. Portanto o sistema será testado em três condições pré-estabelecidas para o tempo t_R : primeiramente, os tempos de execução de cada configuração foram definidos como sendo o mesmo, resultando em um peso de 50%-50% para os novos arcos. Adicionalmente, foram criados dois casos utilizando o princípio de Pareto, definindo o tempo de execução da primeira configuração em 20% e o segundo em 80% e vice-versa.

Apesar de todos os exemplos citados possuírem apenas dois cenários, a estratégia é válida para n cenários, bastando para isso adaptar as equações (14) a (18) para n .

4.3. Algoritmo para validação da estratégia

4.3.1. Cálculo dos custos de comunicação

A abordagem baseia-se no algoritmo *branch-and-bound* (B&B), apresentado na seção 2.7.1, utilizando-se de aspectos do algoritmo descrito em (HU; MARCULESCU, 2003). A Figura 43 é utilizada para entender como o algoritmo proposto funciona. A figura apresenta um exemplo de árvore de busca para mapear uma aplicação com quatro MPs (0,1,2,3) em uma NoC 2x2. A figura não mostra todos os nós por razões de limitação de espaço.

Na Figura 43, cada nó pode pertencer a três categorias: nó raiz, nó interno e nó folha. O nó raiz é um nó vazio onde nenhum MP é mapeado. Os nós internos representam mapeamentos parciais onde os MPs já mapeados são marcados. O nó 21xx, por exemplo, representa um mapeamento parcial onde os MPs 2 e 1 foram mapeados nos roteadores 0 e 1 respectivamente, enquanto os MPs 0 e 3 não foram mapeados ainda. Um nó folha representa um mapeamento completo.

Figura 43 - Árvore de busca para uma NoC 2x2.

Fonte: (HU; MARCULESCU, 2003)

O objetivo selecionado é diminuir o custo de comunicação (tráfego de dados), que pode ser definido pela função-custo descrita na equação (7) (pág. 25):

Cada nó possui seu próprio custo. Tomando novamente como exemplo o nó 21xx, seu custo específico seria definido pelos MPs já mapeados, calculados como $b(a_{2,1}) \times dist(0,1) + b(a_{1,2}) \times dist(1,0)$. De acordo com a teoria de algoritmos B&B nenhum nó descendente irá possuir custo menor do que o custo dos nós de sua linhagem anterior.

Sendo Λ o conjunto de todos os vértices que já foram mapeados, um nó é considerado legal se obedecer à seguinte condição:

$$\forall ch_{i,j}, bw(ch_{i,j}) \geq \sum_{a_{k,l}, c_i c_j \in \Lambda} x_{i,j} \quad (19)$$

Esta condição garante que o nó não viola a capacidade de largura de banda dos canais da rede. Assim, se o nó não obedecer à equação (19), ele é ilegal, assim como todos os seus descendentes, podendo ser podado da árvore de busca.

O custo limite superior (*Upper Bound Cost* - UBC) de um nó é definido como um valor não menor do que o custo mínimo dos seus nós folhas descendentes. Já o custo limite inferior (*Lower Bound Cost* - LBC) de um nó é definido como o menor custo que seus nós folhas descendentes podem alcançar. Ou seja, se um nó possui um valor determinado, então seus nós folhas descendentes terão, no mínimo, este valor. Achar o mapeamento ótimo consiste basicamente em achar o nó folha legal que possui o menor custo.

Na fase *branch* do algoritmo, um novo nó é gerado, alocando um MP não mapeado em um roteador não ocupado. Na fase *bound* cada nó gerado é avaliado para saber se pode gerar nós folhas melhores. Um nó pode ser podado se seu LBC for maior do que o menor UBC achado durante a busca.

Para determinar o UBC será usada a métrica adotada por (HU; MARCULESCU, 2003), onde os MPs ainda não mapeados são mapeados utilizando uma metodologia imediatista (*greedy*). Em cada passo do cálculo do UBC, o MP ainda não mapeado com maior demanda de comunicação é selecionado e sua localização topológica (x, y) é definida pelas seguintes equações:

$$x = \frac{\sum_{c_i \in \Lambda} (b(a_{k,i}) + b(a_{i,k})) \times c_i^x}{\sum_{c_i \in \Lambda} (b(a_{k,i}) + b(a_{i,k}))} \quad (20)$$

$$y = \frac{\sum_{c_i \in \Lambda} (b(a_{k,i}) + b(a_{i,k})) \times c_i^y}{\sum_{c_i \in \Lambda} (b(a_{k,i}) + b(a_{i,k}))} \quad (21)$$

onde c_i^x e c_i^y representam as posições x e y , respectivamente, do mapeamento do MP c_i . A seleção de MPs com maior demanda de comunicação se justifica pela necessidade de um UBC com custo baixo. Mapeando-se prioritariamente os MPs mais comunicantes, pode-se diminuir a distância destes com os módulos já posicionados e, conseqüentemente, o custo de comunicação (HU; MARCULESCU, 2003). Após a geração deste mapeamento, o valor UBC é calculado de acordo com a equação (4), (pág. 25).

O LBC foi determinado também de forma similar ao do trabalho de (HU; MARCULESCU, 2003), onde o cálculo pode ser dividido em três componentes:

$$LBC = C_{m,m} + C_{u,u} + C_{m,u} \quad (22)$$

onde $C_{m,m}$ é o custo de comunicação entre os MPs já mapeados, $C_{u,u}$ é o custo entre os MPs ainda não mapeados e $C_{m,u}$ é o custo de comunicação entre os MPs já mapeados e os MPs ainda não mapeados. O cálculo de $C_{m,m}$ é exato, pois a localização dos MPs mapeados é conhecida. Sendo $\bar{\Lambda}$ o conjunto dos MPs não mapeados e $\bar{\Phi}$ o conjunto dos roteadores não ocupados, o cálculo de $C_{u,u}$ é definido como:

$$C_{u,u} = \frac{1}{2} \times \sum_{\forall c_i, c_j \in \bar{\Lambda}} b(a_{i,j}) \times \min_{\forall r_m, r_n \in \bar{\Phi}} (dist(m, n)) \quad (23)$$

Por sua vez, o custo $C_{m,u}$ através da equação (24):

$$\begin{aligned} C_{m,u} = & \sum_{\forall c_i \in \Lambda} \sum_{\forall c_j \in \bar{\Lambda}} b(a_{i,j}) \times \min_{\forall r_k \in \bar{\Phi}} (dist(\Omega(c_i), k)) \\ & + \sum_{\forall c_i \in \bar{\Lambda}} \sum_{\forall c_j \in \Lambda} b(a_{i,j}) \times \min_{\forall r_k \in \bar{\Phi}} (dist(k, \Omega(c_j))) \end{aligned} \quad (24)$$

Justificativas detalhadas de cálculos dos componentes de LBC podem ser vistas no trabalho de (HU; MARCULESCU, 2003).

4.3.2. O algoritmo

A Figura 44 mostra o pseudocódigo do algoritmo completo. Primeiramente, um nó raiz é gerado (linha 1). A linha 3 indica que uma fila é formada tendo como o seu primeiro elemento o nó raiz; enquanto houver elementos nesta fila, o laço da linha 4 ficará em execução. Um novo laço interno é percorrido (linha 6): para cada roteador desocupado, um novo nó é criado. Todo Nó deve possuir os seguintes elementos: valores de UBC e LBC, e o próprio mapeamento como demonstrado no exemplo da Figura 43. Durante as iterações, o algoritmo avalia os custos e, de acordo com os valores de LBC e UBC e da legalidade da situação, os nós continuam sendo avaliados ou são podados. Em cada iteração, as instruções das linhas 10 a 12 avaliam, caso o nó seja do tipo folha, se o mapeamento ótimo foi encontrado.

Figura 44 - Pseudocódigo do algoritmo B&B.

Algoritmo Branch-and-Bound para mapeamento de MPs em NoCs	
1	Nó raiz = novo nó(NULO)
2	Min_UBC = $+\infty$, melhor_custo = $+\infty$
3	FILA.inserir(Nó raiz)
4	ENQUANTO (Fila.elementos != 0) {
5	Nó atual = FILA.próximo()
6	PARA cada roteador desocupado r_i {
7	Criar Nó n_{novo}
8	SE ($n_{novo}.LBC > \text{min_UBC} \ \& \ n_{novo}.\acute{e}_legal$) continue
9	SE ($n_{novo}.Nó_folha$) {
10	SE ($n_{novo}.Custo < \text{melhor_custo}$) {
11	Melhor_custo = $n_{novo}.Custo$
12	Melhor_mapeamento = n_{novo} }}
13	DO CONTRÁRIO {
14	SE ($n_{novo}.UBC < \text{min_UBC}$) {
15	min_UBC = $n_{novo}.UBC$
16	FILA.inserir(n_{novo}) }}
17	}

Fonte: Autor.

O algoritmo foi testado com as aplicações indicadas anteriormente e os tempos de execução são apresentados na Tabela 7. A primeira coluna apresenta o tipo de aplicação mapeada de acordo com a variação de valores dos arcos, como explicado na seção 4.2; a aplicação VOPD não apresenta conflito de arcos, como indicado na Figura 40, portanto não há casos diferenciados de acordo com o tempo de reconfiguração tr . A segunda coluna relaciona o número de MPs mapeados, assim como o aspecto geométrico da NoC; a terceira coluna mostra a capacidade da largura de banda, escolhida arbitrariamente. Finalmente, a última coluna mostra o tempo de execução do algoritmo em segundos até a obtenção do mapeamento ótimo.

O algoritmo é lento, pois deve-se percorrer toda a árvore de busca com o objetivo de se achar o melhor mapeamento. É importante lembrar, porém, que o objetivo deste algoritmo é achar a melhor solução (e não a mais rápida) para validar a estratégia; algoritmos mais rápidos podem ser aplicados em NoCs de grandes dimensões.

Como o algoritmo é explosivo em relação ao número de MPs, o aumento do tempo de execução do mapeamento do caso com 9 MPs para o caso de 12 MPs é notável. Porém, nota-se também que o processo de poda do algoritmo é eficiente, pois apesar do aumento do tempo de execução na ordem de 120 vezes, pode-se estimar que o número de possibilidades de mapeamento aumenta em 1320 vezes.

Tabela 7 - Tempos de execução do algoritmo B&B

Aplicação	Número de MPs	Capacid. Bw.	Tempo de execução (s)
VOPD	9 (3x3)	1 GB/s	75
MP4 50%-50%	9 (3x3)	2 GB/s	55
MP4 20%-80%	9 (3x3)	2 GB/s	58
MP4 80%-20%	9 (3x3)	2 GB/s	54
MMS 50%-50%	12 (4x3)	0,2 GB/s	7258
MMS 20%-80%	12 (4x3)	0,2 GB/s	7247
MMS 80%-20%	12 (4x3)	0,2 GB/s	7251

Fonte: Autor.

4.4. Simulação das aplicações para arquiteturas SDR-NoC simples

Para o teste da efetividade das funções-custos do algoritmo de mapeamento ótimo proposto na seção anterior, foi definida uma estratégia de uso plataforma *Network-on-chip Simulator* (Noxim) (FAZZINO; PALESI; PATTI, 2014). Esta forma de análise foi escolhida devido à indisponibilidade das aplicações para a realização de uma simulação RTL ou mesmo em alto nível.

O Noxim é um simulador estruturado em systemC, que é uma linguagem de descrição de sistemas baseado em C++. Neste simulador, o usuário tem a disposição uma NoC regular com topologia tipo grelha e pode definir vários parâmetros, entre eles: tamanho da rede, tamanho do *buffer* do roteador, algoritmo de roteamento, tipo de tráfego de dados, etc. O simulador permite coletar vários dados para avaliação da rede, tais como pacotes recebidos e enviados, atrasos nos roteadores (*delays*), consumo de energia, entre outros.

Uma das razões para a escolha do Noxim é a possibilidade de se fazer alterações ou inclusões na ferramenta através de modificações do código-fonte. Este foi alterado para suportar reconfiguração dinâmica e foram inseridos geradores de tráfego para corresponder ao fluxo de dados dos APCGs. Tome-se, por exemplo, o grafo da Figura 40(b). A simulação é dividida em duas etapas reconfiguráveis: na primeira, os roteadores enviam e recebem dados dos módulos VLD, RLD e Ins; na segunda etapa, os dados enviados e recebidos são relativos aos MPs Ups, Pad e VopR. Outras combinações podem ser definidas.

Foram criados dois cenários para simulação, descritos a seguir:

1) NoC ideal - Neste caso, os parâmetros da rede gerada no Noxim são virtualmente infinitos. Isso quer dizer que a rede suportaria qualquer valor de largura de banda e o *buffer* dos roteadores poderiam armazenar qualquer quantidade de *flits*.

2) NoC real - Neste segundo caso, foram definidos parâmetros de uma NoC real, com *buffers* suportando 4 *flits* e largura de banda limitada, com valores que serão detalhados na Tabela 9.

Nos cenários acima, para provar que a estratégia de mapeamento é válida, dois casos de mapeamento foram comparados.

a) Mapeamento mínimo - obtido pelo resultado ótimo do algoritmo B&B.

b) Mapeamentos aleatórios - casos gerados com disposição aleatória dos MPs.

A ideia é verificar o quão melhor é o resultado obtido com algoritmo de mapeamento ótimo por B&B em comparação com o mapeamento médio obtido com os casos gerados aleatoriamente. Para o mapeamento aleatório, nos casos onde as NoCs possuem tamanho 3x3 (VOPD e MP4), foram geradas amostras de 0,5% do universo de todos os mapeamentos possíveis. No caso do MMS foram geradas amostras de $5 \times 10^{-5}\%$ de todos os espaços de amostra. Este número de amostras é bastante significativo para analisar o impacto de mapeamento. O número pequeno de amostragem se justifica pela inviabilidade de testar uma população significativa do universo das possibilidades de mapeamento. Em uma NoC com 12 roteadores, por exemplo, existem mais de $4,79 \times 10^8$ possibilidades.

4.4.1. Processo de simulação

Para gerar o alto número de simulações dos mapeamentos aleatórios, foi necessário desenvolver um processo de automação das simulações. A Figura 45 mostra tal processo, onde um programa desenvolvido em MatLab® (MATLAB, 2014) gera uma rede pronta para simulação no Noxim, tendo como parâmetros de entrada os grafos de aplicação e parâmetros de tempo de simulação e tempos de reconfiguração.

O programa gera uma série de arquivos em C++ que serão executados no Noxim. Um *script* para Linux armazena os resultados em um relatório e como uma base de dados para que os dados específicos possam ser extraídos e analisados por um outro *script*. Por exemplo, ao analisar que o atraso da rede tende a aumentar para valores muito acima da média, pode-se concluir que os limites de largura de banda da rede tenham sido violados.

Figura 45 - Automação para simulação de mapeamentos aleatórios

Fonte: Autor.

4.5. Resultados experimentais

As figuras a seguir, mostram o mapeamento ótimo obtido pelo algoritmo B&B. Para melhor visualização e compreensão do mapeamento, cada figura mostra o APCG e o resultado do mapeamento. As APCGs das aplicações são replicadas nas figuras para facilitar o entendimento.

A Figura 46(b) mostra, para a aplicação VOPD, o melhor mapeamento obtido pelo algoritmo B&B. Cumpre lembrar que para este caso, não há arcos compartilhados, não existindo, portanto as variações 50%-50%, 20%-80% ou 80%-20%.

Figura 46 - Resultado do mapeamento da aplicação VOPD.

Fonte: Autor.

No entanto, as variações de arco tiveram efeito na aplicação MP4. Aqui nota-se a diferença na variação de Pareto para o peso dos arcos. O melhor mapeamento obtido para variação 20%-80% é ilustrado na Figura 47(b), enquanto o melhor resultado para os casos 50%-50% e 80%-20% é mostrado na Figura 47(c).

Figura 47 - Resultado do mapeamento da aplicação MP4.

Fonte: Autor.

em torno de 27% para a VOPD e a MP4, chegando a 41% no caso da MMS. Não houve diferenças notáveis entre as diferentes aplicações.

Tabela 8 - Resultados de simulação para NoC ideal.

Aplicação	Melhor Mapeamento			Amostras Aleatórias		
	Atr. Médio	Atr. Máx.	Ener. (mJ)	Atr. Médio	Atr. Máx.	Ener. (mJ)
VOPD	6,03	142	19,76	7,91	145,29	27,18
MP4 50%-50%	7,67	140	47,53	9,55	169,16	64,93
MP4 20%-80%	7,55	146	48,13	9,83	173,59	70,50
MP4 80%-20%	7,37	153	42,91	9,41	169,16	61,23
MMS 50%-50%	6,36	121	4,31	9,36	139,57	6,96
MMS 20%-80%	6,50	110	4,60	9,40	139,89	7,11
MMS 80%-20%	6,20	124	4,01	9,28	139,78	6,80

Fonte: Autor.

A Tabela 9 mostra os resultados para uma NoC real, com uma estrutura similar à da tabela anterior, com as seguintes diferenças: a segunda coluna mostra a capacidade de largura de banda escolhida arbitrariamente e utilizada no algoritmo B&B para ilustrar os efeitos nesse quesito e a última coluna mostra a porcentagem de casos de mapeamento aleatório que violam a capacidade de largura de banda.

No caso da aplicação MP4 na NoC real, os valores de atraso (médio e máximo) do mapeamento ótimo são muito próximos das médias dos casos aleatórios; isto deve-se à quantidade elevada de dados sendo roteados, o que mantém os *buffers* sempre cheios, ou seja, o melhor mapeamento não apresenta os seus efeitos em caso de redes congestionadas. Entretanto, o mapeamento é essencial para respeitar a capacidade da largura de banda da rede: até 61% dos mapeamentos aleatórios violaram a restrição de 2GB/s. Além disso, isso prova a eficiência da estratégia em relação às equações (16), (17) e (18), pois as simulações comprovam que o caso ótimo B&B realmente não ultrapassa a largura de banda máxima.

Tabela 9 - Resultados de simulação para NoC real.

Aplicação	Melhor Mapeamento				Amostras Aleatórias			
	Capacid. Bw.	Atraso Médio	Atraso Máx.	Ener. (mJ)	Atraso Médio	Atraso Máx.	Ener. (mJ)	% de Violação
VOPD	1 GB/s	13,81	561	20,41	20,22	829,83	27,93	18%
MP4 50%-50%	2 GB/s	60,64	1429	51,42	62,38	1571,9	69,36	59%
MP4 20%-80%	2 GB/s	62,91	1558	52,74	66,17	1481,9	75,04	61%
MP4 80%-20%	2 GB/s	44,48	1873	46,46	49,48	1666,5	63,79	57%
MMS 50%-50%	0,2 GB/s	23,24	537	4,57	38,02	1065,2	7,26	32%
MMS 20%-80%	0,2 GB/s	23,88	539	4,68	38,88	1066,6	7,32	33%
MMS 80%-20%	0,2 GB/s	23,20	532	4,40	38,15	1063,2	6,97	32%

Fonte: Autor.

Nos outros casos para os quais exigiu-se menos da capacidade da NoC, a efetividade do algoritmo de mapeamento ótimo fica evidenciada pela redução média de 31% de atraso médio no VOPD e de 38% no MMS. Também é importante notar que a economia de energia para os casos reais são boas (entre 27% e 37%). Não houve diferenças notáveis entre as diferentes aplicações.

Os resultados anteriores com o algoritmo B&B indicam que a obtenção de mapeamento ótimo em reconfiguração dinâmica é importante, trazendo evidentes benefícios para um melhor tráfego e para uma NoC mais rápida. O algoritmo foi importante para evidenciar este aspecto, porém, por gerar o mapeamento ótimo, a sua utilização é inviável para grande quantidade de MPs em NoCs maiores. Para um número elevado de MPs ou para outras funções-custos, pode-se adotar outras estratégias nos trabalhos clássicos de mapeamento, com algoritmos mais rápidos mas que geram mapeamentos não-ótimos.

5. MAPEAMENTO PARA ARQUITETURAS SDR-NoCs COMPLEXAS

Neste capítulo será detalhada a solução desenvolvida para arquiteturas SDR-NoCs complexas. Como definido nos capítulos 1 e 2, uma arquitetura SDR-NoC complexa é um SDR estruturado em uma NoC que pode ser regular ou irregular, com topologia direta ou indireta, com MPs tanto homogêneos quanto heterogêneos.

Na primeira seção o problema de mapeamento em arquiteturas SDR-NoCs complexas será definido formalmente. Nas seções posteriores, será exibida primeiramente a solução com um algoritmo semiexaustivo e posteriormente, com a utilização de algoritmos genéticos.

5.1. Formulação do problema

Podendo os MPs da rede ser heterogêneos, a resolução do problema de mapeamento implica em também tratar do posicionamento na rede intrachip e, em última instância, no tecido lógico do FPGA de base. A formulação matemática formal de mapeamento e posicionamento terá como modelo inicial a rede DyNoC, porém o objetivo principal é fazer um modelo o mais geral possível para que este seja facilmente estendido a outras arquiteturas.

Como primeira generalização, será considerado que todos os roteadores conectar-se-ão a seu respectivo MP (daqui em diante denominados *slots*) a partir do canto superior direito deste; assim, tem-se um modelo como o representado na Figura 49.

Um dispositivo reconfigurável possui $m \times n$ *slots* de alocação, onde m e n representam respectivamente, os números de colunas e linhas de uma rede intrachip direta e regular. O grupo P indica um conjunto de posições no dispositivo:

$$P = \{p_k | k = 1, 2, \dots, m \times n\}, \quad p_k = (x_p, y_p) \quad (25)$$

Uma posição refere-se ao espaço tanto de um potencial MP assim como do roteador naquela posição. Um espaço de entrada 7x7 pode ser observado na Figura 49, onde o canto inferior esquerdo é a posição $p_k = (0,0)$ e o canto superior direito é a posição $p_k = (6,6)$.

Figura 49 - Modelo de SDR baseado em NoC com tamanho 7x7

Fonte: Autor.

Um cenário é a arquitetura definida por uma configuração em um dado momento. Cada reconfiguração é representada por um cenário diferente. Cada cenário s pertence ao conjunto SC de todos os cenários.

O modelo clássico de grafo de caracterização de aplicação APCG apresentado no capítulo 2 e adotado integralmente no capítulo 4, é insuficiente para descrever as características físicas dos MPs, necessárias à resolução do problema de posicionamento. Definimos o grafo estendido de caracterização de aplicação (*Extended Application Characterization Graph*, EAPCG) como o APCG com dados geométricos dos MPs. O EAPCG é definido como $G(C, A)$ onde cada MP (ou *core*) $c_i \in C$ é um vértice com pesos (h_i, w_i) , onde h_i e w_i representam, respectivamente altura e largura do MP em termos de *slots*. O conjunto C é subdividido em dois subconjuntos:

- MPs Fixos - estão presentes em todos os cenários de configuração. São representados pelos MPs $Fc_i \in FC$.
- MPs Reconfiguráveis - não estão presentes em todos os cenários. Estando presentes no cenário s , são representados pelos MPs $Rc_i(s) \in RC(s)$.

Como no APCG tradicional, cada arco $a_{i,j} \in A$ representa a comunicação entre os MPs (cores) c_i e c_j . Para cada arco, o parâmetro $b(a_{i,j})$ indica a largura de banda de comunicação entre os MPs c_i e c_j em Mbits/s.

Diferente do mapeamento clássico (de MP para roteador), o mapeamento em arquiteturas SDR-NoCs complexas é definido como:

$$\Omega: C \rightarrow P \quad (26)$$

associando cada MP c_i a uma posição de referência $p_k = (x_k, y_k)$, com $\Omega(c_i) = (x_i, y_i)$. Em paralelo, pode-se usar a função.

$$\Phi: C \rightarrow R \quad (27)$$

onde R é o conjunto de todas as regiões retangulares definidas por duas posições p_j e p_k , nas diagonais opostas. $\Phi(c_i) = r_i \in R$ é um retângulo que possui origem na parte inferior à esquerda em $(x_i - h_i + 1, y_i - w_i + 1)$ e término na parte superior à direita em (x_i, y_i) , sendo que $\Omega(c_i) = (x_i, y_i)$. A Figura 50 ilustra estes conceitos; o MP c_1 (em cinza) tem o mapeamento de posição $\Omega(c_1) = (6,6)$, enquanto aponta para a região R_1 da figura (alaranjado). Como c_1 tem associado a ele um peso ($h=2, w=3$), tem-se que R_1 é definido pelas posições $(6,6)$ e $(5,4)$.

Figura 50 - Definição de região.

Fonte: Autor.

Cada cenário $s = FC \cup RC$ possui um mapeamento específico $\Omega\Omega(s)$, que corresponde à união de todos os mapeamentos $\Phi(c_i)$, com $c_i \in s$. O posicionamento dos módulos fixos deve ser o mesmo para todos os cenários s . Cada mapeamento possui as seguintes restrições.

$$\forall i \neq j, \Phi(c_i) \cap \Phi(c_j) = \emptyset \quad (28)$$

$$\Phi(c_i) \subseteq P \quad (29)$$

A equação (28) garante que nenhum MP irá ocupar o espaço de outro, enquanto a equação (29) assegura que o posicionamento estará contido em P , ou seja, não irá extrapolar o espaço de entrada.

Uma arquitetura de dispositivo é um grafo específico para cada cenário s , $Arch(\Omega\Omega(s)) = (T, Ch)$, onde cada roteador $t_i \in T$ é um vértice e cada arco direcionado $ch_{i,j} \in Ch$ representa o canal (*channel*) de comunicação entre os vértices t_i e t_j . Para cada arco, o parâmetro $bw(ch_{i,j})$ indica a largura de banda de comunicação entre t_i e t_j . $Path(i, j)$ representa o conjunto de canais que fazem o

caminho desde a origem i até o destino j e $hops(i,j)$ representa o número de roteadores no qual a informação passa desde a origem i até o destino j .

Os roteadores podem ser ativos ou inativos. A definição de um roteador inativo j é:

$$\forall i, \begin{cases} bw(ch_{i,j}) = 0 \\ bw(ch_{j,i}) = 0 \end{cases} \quad (30)$$

Ou seja, um roteador inativo não possui conexões com qualquer outro roteador. Se um roteador possuir ao menos uma conexão, ele é considerado ativo.

Os roteadores estarão posicionados no espaço P , sendo que todos os MPs c_i comunicam-se através de um roteador associado na mesma posição de alocação:

$$\forall c_i \in C, \exists t_j \in T \rightarrow p(c_i) = p(t_j) \quad (31)$$

Pode-se observar na Figura 50 que, por exemplo, o roteador $t1$ está ativo, na posição (6,6), conectado ao $c1$ na mesma posição. De outro lado, $t2$ está na posição (5,5) e inativo. Já o roteador $t3$, em (5,6), está ativo e comunica-se apenas com os roteadores da posição (6,6) e (4,6).

Uma região ocupada $Ro(c_i) \subseteq \Phi(c_i)$ é uma região que não permite que roteadores estejam ativos. Esta região existe se a altura e largura do MP forem maiores do que 1 e é delimitada por um retângulo com origem na parte inferior à esquerda em $(x_i - h_i + 1, y_i - w_i + 1)$ e término na parte superior à direita em $(x_i - 1, y_i - 1)$. A Figura 51 mostra uma região associada a um MP (externo em cinza) e a região ocupada correspondente (interno em vermelho).

Figura 51 - Uma região $\Phi(c_i)$ e uma região ocupada $Ro(c_i)$.

Fonte: Autor.

Os roteadores pertencentes a uma região comum funcionam normalmente, enquanto que os roteadores pertencentes a uma região ocupada são desativados.

Assim como ocorre no mapeamento clássico, outra restrição para o mapeamento é de que em cada cenário s , a comunicação entre os módulos não pode ultrapassar a capacidade de largura de banda dos canais:

$$\forall ch_{i,j} \in Arch(\Omega(s)), bw(ch_{i,j}) \geq \sum_{a_{i,j} \in \Omega(s)} x_{i,j} \quad (32)$$

onde

$$x_{i,j} = \begin{cases} b(a_{i,j}), & \text{se } ch_{i,j} \in Path(p(c_i), p(c_j)) \\ 0, & \text{do contrário} \end{cases} \quad (33)$$

O objetivo do mapeamento é posicionar os MPs de forma a minimizar ou maximizar um parâmetro escolhido pelo projetista, ou seja, neste caso, minimizar o custo de comunicação representado pela equação (34):

$$\sum_{a_{i,j} \in \Omega(s)} b(a_{i,j}) \times hops(p(c_i), p(c_j)) \quad (34)$$

O problema de minimização será ilustrado a seguir. O EAPCG de uma aplicação sintética é ilustrado na Figura 52. Esta aplicação possui treze MPs, sendo seis MPs fixos (Fc1-Fc6) e sete MPs reconfiguráveis distribuídos em três cenários s_1, s_2 e s_3 . Por exemplo, no cenário 2, existem dois RCs denominados Rc1 e Rc2 de tamanhos (2,2) e (1,3), respectivamente. O peso dos arcos, que representam a comunicação entre os MPs, está representada em vermelho nos arcos (Mb/s). Já o tamanho dos MPs está representado como o peso nos vértices (largura, altura).

Figura 52 - EAPCG de uma aplicação sintética.

Fonte: Autor.

A Figura 53 ilustra um mapeamento com mínimo custo para o EAPCG da Figura 52. Os roteadores associados aos MPs estão em preto. Novamente, pode-se observar, como exemplo, que no cenário 2, Rc1 e Rc2 foram alocados para as posições (5,4) e (3,4), ocupando as respectivas regiões associadas segundo os seus tamanhos de (2,2) e (1,3).

Figura 53 - Mapeamento mínimo para: (a) cenário 1, (b) cenário 2, (c) cenário 3.

Fonte: Autor.

Note-se que os MPs com comunicação crítica, e.g. do Fc1 para o Fc4, com 50 Mb/s, estão próximos um do outro. Isto também pode ser observado quando o MP Rc2(2) é alocado no segundo cenário: ele fica a apenas dois *hops* de distância do Fc4, sendo que eles enviam e recebem 18 Mb/s um para o outro. Vizinhos similares podem ser observadas nos resultados do mapeamento.

5.2. Aplicações sintéticas

A seção anterior traz o modelo de representação de aplicações para arquiteturas SDR-NoCs complexas, além dos critérios para mapeamento e posicionamento ótimo, baseados em quantidade de tráfego pela rede intrachip. Para a obtenção do mínimo, implementações diferentes de algoritmo genético foram desenvolvidas e comparadas com um algoritmo de otimização semiexaustivo.

Dada a dificuldade de se encontrar exemplos de aplicação reais na literatura ou disponíveis de forma aberta que fossem adequados a SDRs, nove aplicações sintéticas foram propostas. As nove aplicações são baseadas em três arranjos de MPs denominados Alpha, Beta e Gamma. Os arranjos são detalhados na Tabela 10:

Tabela 10 - Número de MPs nas aplicações sintéticas.

Aplicação	MPs Fixos	MPs Reconfiguráveis				
		Cenário 1	Cenário 2	Cenário 3	Cenário 4	Cenário 5
Alpha	6	2	2	3	N/A	N/A
Beta	4	3	5	2	4	N/A
Gamma	7	3	5	5	3	3

Fonte: Autor.

A partir dessas arranjos básicos, foram desenvolvidas três variações de tamanhos de MPs para cada um, com o objetivo de dar mais ou menos flexibilidade para o processo de posicionamento:

1 - MPs pequenos.

Nesta variação os MPs posicionados nunca irão violar as restrições de tamanho do SDR-NoC, dados pela equação (29).

2 - MPs variados.

Nesta condição MPs pequenos e grandes se alternam, sendo que em algumas condições, o posicionamento irá violar o tamanho do SDR-NoC.

3 - MPs Grandes.

Neste caso, os MPs quase sempre irão violar o espaço de entrada do SDR-NoC, reservando poucas soluções possíveis.

A Figura 52 mostra o EAPCG para a aplicação Alpha com MPs grandes, o que pode ser verificado pelos seus tamanhos descritos nos nós do grafo. A Figura 53 já mostra o resultado do mapeamento e posicionamento mínimo. Devido a limitações de espaço, não é possível detalhar o EAPCG das outras aplicações, porém, informações detalhadas destas aplicações estão descritas no Apêndice A desta tese.

5.3. Algoritmos para resolução de mapeamento e posicionamento

5.3.1. Codificação das possíveis soluções

Para a codificação das soluções intermediárias ou final, foi desenvolvida uma representação simples. Parte-se da premissa de que MPs compactados levam, geralmente, a um melhor custo de comunicação. Tal premissa é razoável, uma vez que o afastamento de MPs apenas causaria a existência de mais roteadores entre eles. Portanto, não serão considerados espaços vazios entre MPs, o que torna o processo de computação mais simples.

Utilizando-se deste artifício, limita-se o número de soluções possíveis, porém, torna a complexidade do problema proporcional ao fatorial ao número de módulos da aplicação. Utilizando-se uma abordagem mais generalista e considerando os espaços vazios, o problema seria proporcional ao fatorial do número de *slots* da arquitetura, o que seria completamente intratável.

A Figura 54 mostra a codificação da solução final para a aplicação com treze MPs e três cenários, cujo mapeamento e posicionamento foi ilustrado na Figura 53. A solução é codificada como uma sequência de MPs, onde os primeiros oito elementos representam o conjunto dos seis MPs fixos e dois reconfiguráveis, que compõe o primeiro cenário. Nas posições 9 e 10 estão os elementos reconfiguráveis que compõe o segundo cenário e as últimas três posições compõe o terceiro cenário.

Figura 54 - Exemplo de codificação de mapeamento.

Fc1	Fc4	Fc6	Fc3	Fc5	Rc1(1)	Fc2	Rc2(1)	Rc2(2)	Rc1(2)	Rc3(3)	Rc2(3)	Rc1(3)
FC U RC(1)								RC(2)		RC(3)		

Fonte: Autor.

A posição dos MPs dentro da codificação representam a posição física relativa do MP. O posicionamento no espaço de entrada é feito na seguinte sequência:

- 1 - De cima para baixo.
- 2 - Da esquerda para a direita.

Entende-se, então, que o preenchimento dos espaços é feito da seguinte maneira para os MPs na sequência de codificação: 1) para cada MP, buscar a coluna mais à esquerda que tenha *slot* vazio; 2) ao encontrá-lo, posicionar o MP com o canto esquerdo superior no *slot* superior livre; 3) caso não seja possível, devido a conflitos, passar para a linha seguinte. A codificação da solução pode ser entendida observando-se o mapeamento final da Figura 53. O primeiro cenário é mapeado como na Figura 53(a). Os MPs Fc1, Fc4 e Fc6 foram alinhados à coluna 0; a seguir Fc3 é posicionado na linha superior, seguido de Fc5. Depois, Rc1 é alocado no espaço possível junto à coluna 3; finalmente Fc2 e Rc2 são alocados nos espaços disponíveis. No segundo cenário os MPs reconfiguráveis do primeiro cenário são removidos e os MPs Rc1(2) e Rc2(2) são alocados no lugar, como na Figura 53(b), utilizando-se as prioridades de posicionamento. A Figura 53(c) mostra o mesmo processo para os últimos três MPs.

5.3.2. Resolução com algoritmo semiexaustivo

De forma similar ao capítulo 4, será aplicado primeiramente um algoritmo de referência para achar os mapeamentos mínimos e depois usar esta referência para comparar com soluções algorítmicas mais abrangentes.

A proposta do algoritmo semiexaustivo é bem simples: procurar em todas as permutações possíveis do vetor de forma sequencial; por exemplo, para o vetor da Figura 54 seriam testadas todas as disposições de posições entre os módulos, de Fc1 a Rc2(2). Para cada disposição, seriam testadas também as duas variações possíveis do segundo cenário, com as seis variações do terceiro. Quando não é possível mapear todos os módulos, devido à extrapolação do limites físicos do dispositivo, o algoritmo poda tal permutação. Para este procedimento, foi utilizada uma variação do algoritmo B&B apresentado na seção adaptando-se a codificação para aquela apresentada na Figura 54 e acrescentando-se as restrições de violação do espaço de entrada e conflitos de posicionamento (Eqs. (28) e (29)) como critério de poda. Apesar do algoritmo ser semiexaustivo, o mínimo global é sempre obtido.

De acordo com a codificação de entrada, o algoritmo posiciona cada módulo e emula o algoritmo SXY (Bobda, 2005) em cada fase para calcular o custo de

mapeamento de acordo com a Eq. (34) (cumpre lembrar que é necessário emular o SXY para calcular a distância entre os pontos de partida e chegada de um pacote na arquitetura considerada). Ao final da execução, o mapeamento mínimo é obtido.

5.3.3. Resultados do algoritmo semi-exaustivo

A Tabela 11 apresenta os resultados das simulações realizadas com o algoritmo semi-exaustivo. A primeira coluna mostra os três tipos de aplicações, e suas respectivas variações são exibidas na segunda coluna. O melhor custo obtido pelo algoritmo é mostrado na terceira coluna e o tempo de execução do algoritmo é apresentado na quarta coluna. A quinta coluna mostra a porcentagem de casos que excederam o espaço de entrada e, conseqüentemente, não puderam ser mapeados.

Tabela 11 - Resultados do algoritmo semi-exaustivo.

Aplicação	MPs	Melhor Custo	Tempo de execução	% de Violação
ALPHA	Pequenos	877	197 s	0%
	Variados	1317	89 s	71,3%
	Grandes	1355	60 s	94,8%
BETA	Pequenos	2364	6552 s	0%
	Variados	2518	4486 s	15,3%
	Grandes	2659	698 s	91,4%
GAMMA	Pequenos	N/A	$> 8,6 \times 10^4$ s	N/A
	Variados	N/A	$> 8,6 \times 10^4$ s	N/A
	Grandes	N/A	$> 8,6 \times 10^4$ s	N/A

Fonte: Autor.

O algoritmo funcionou bem para o primeiro caso, com tempos de execução aceitáveis. Para casos de MPs de tamanho grande, os tempos de execução ficaram na mesma ordem de grandeza, enquanto a quantidade de casos em que houve violação do espaço de entrada aumentou, como esperado. Comparando a violação

de casos com o tempo de execução, nota-se que o processo de poda acelera significativamente o processo de mapeamento.

O algoritmo também apresentou boas respostas para a aplicação Beta, porém o tempo de execução aumentou significativamente, quando comparado com a primeira aplicação.

A explosão nos casos de busca do problema torna-se clara quando a aplicação Gamma é analisada. Mesmo após 24 horas de simulação, o algoritmo não conseguiu concluir a análise dos mapeamentos.

Observando-se os dados das aplicações da Tabela 10 (pág. 94), fica evidenciado que o tamanho da rede intrachip e a quantidade de MPs reconfiguráveis e cenários causam um aumento no esforço algorítmico para a obtenção da solução do mapeamento e posicionamento. Portanto, para casos reais, onde muitas aplicações ultrapassam 16 MPs, nota-se a necessidade de um algoritmo que possa resolver o problema em tempo polinomial.

5.3.4. Resolução com algoritmos genéticos

Para lidar com o problema apresentado na última seção, o algoritmo genético (GA) foi escolhido. Os GAs têm sido utilizados com sucesso para tratar uma série de problemas de otimização, tendo inclusive sido aplicados em mapeamento de MPs, como apresentado anteriormente no capítulo 3.

5.3.4.1. Estrutura geral do algoritmo genético

Nesta seção serão apresentadas a codificação e o pseudocódigo do GA geral que serão utilizados no resto do capítulo.

Para a codificação do cromossomo, será utilizada a representação apresentada na Figura 54. Desta forma, cada MP irá representar um gene e um conjunto de MPs ordenados de acordo com os cenários de configuração representará um cromossomo.

O pseudocódigo será descrito detalhadamente na Figura 55. O algoritmo será testado posteriormente com modificações nos tipos de *crossover*, ou com técnicas de variação de população de população, como as descritas na seção 2.7.2.5, porém a estrutura geral, apresentada na Figura 55, será a referência.

Figura 55 - Pseudocódigo do algoritmo genético.

Estrutura geral do algoritmo genético	
1	Melhor_solução = $+\infty$
2	População = gerar_população_randômica(TP)
3	$\forall$população, <i>fitness</i> = 1 / custo de comunicação
4	ENQUANTO geração $\leq$ NG faça
5	ENQUANTO $i \leq$ TP faça
6	Cr_sel1, Cr_sel2 = roleta(população, fitness)
7	SE P(crossover) então
8	Nova_geração(i) = crossover(Cr_sel1, Cr_sel2)
9	Nova_geração(i+1) = crossover(Cr_sel2, Cr_sel1)
10	Do contrário
11	Nova_geração(i) = Cr_sel1
12	Nova_geração(i+1) = Cr_sel2
13	FIM SE
14	SE P(mutação) então
15	Nova_geração(i) = mutação(Cr_sel1)
16	Nova_geração(i+1) = mutação(Cr_sel2)
17	FIM SE
18	PARA i, i+1
19	SE Nova_geração(i).foi_mapeado então
20	Novo_fitness(i) = 1 / custo de comunicação
21	Do contrário
22	Novo_fitness(i) = 0
23	FIM SE
24	FIM PARA
25	FIM ENQUANTO
26	População = nova_população
27	Fitness = novo_fitness
28	SE mais_apto(geração) < melhor_solução então
29	Melhor_solução = mais_apto(geração)
30	FIM SE
31	Geração++
32	FIM ENQUANTO

Fonte: Autor.

Inicialmente, um conjunto de soluções é gerado de forma aleatória (linha 2). O parâmetro "tamanho da população" (TP), definido pelo usuário, define o número de indivíduos da população. Na linha 3 o valor *fitness* de cada indivíduo é calculado como o inverso do custo de comunicação, representado pela equação (34) (pág. 92).

O algoritmo é composto principalmente de dois laços: o primeiro refere-se ao refinamento da população, que é limitada pelo número de gerações (NG). Em cada geração, um segundo laço executa seleção, otimização e mutação para todos os elementos da população.

A processo de seleção sempre será executado pelo método da roleta⁸ (linha 6), que é um método de seleção eficaz (GOLDBERG, 1989) e se mostrou eficiente na maioria dos casos testados do presente trabalho. A roleta irá selecionar dois dos cromossomos (Cr_sel1 e Cr_sel2) com uma probabilidade proporcional ao seu valor *fitness*.

O *crossover* tem uma probabilidade de execução $P(\text{crossover})$, tipicamente entre 0,5 e 1. Neste trabalho serão testados diferentes tipos de *crossover*, procedimentos que serão descritos nas próximas seções. Depois do *crossover*, o algoritmo executa o processo de mutação na linha 14 de acordo com sua probabilidade (normalmente menor que 0,05), gerando dois cromossomos mutantes.

No laço da linha 18, o mapeamento é verificado, i.e., se o resultado é válido, não violando o espaço de entrada. Caso o mapeamento seja válido, o valor *fitness* é calculado, caso contrário o valor é anulado.

Ao final de cada iteração, a população antiga é eliminada, dando lugar a nova população. A linha 29 indica que o algoritmo guarda o melhor indivíduo da população atual caso este seja mais apto do que a melhor solução achada até o momento.

⁸ Método previamente explicado na seção 2.7.2.2 desta tese.

5.3.4.2. Tipos de cruzamento (*crossover*) utilizados

Nesta seção, serão apresentados os operadores de *crossover* utilizados. Para a codificação do problema apresentado na seção 5.1, são adequados os *crossovers* de permutação (CICIRELLO, 2006), que mudam as posições dos parentes sem repetir os genes.

Os operadores de *crossover* são normalmente aplicados no cromossomo inteiro. Porém, como a codificação adotada divide os cromossomos em partes correspondentes a cada cenário de configuração, será realizada uma adaptação onde o *crossover* será executado em várias partes do mesmo cromossomo. Estas adaptações serão detalhadas nas seções a seguir.

A. Ordered Crossover (OX)

O primeiro *crossover* adotado neste trabalho é o clássico *ordered crossover*, OX (GOLDBERG, 1989). A Figura 56 exemplifica o processo de cruzamento em um mapeamento com três cenários, onde o primeiro cromossomo é o doador e o segundo é o receptor. As três faixas (azul, verde e amarela) mostram a separação dos cenários. O OX será aplicado em cada faixa de cenário de forma independente.

A Figura 56(a) mostra o primeiro passo, onde dois cromossomos são selecionados. O primeiro cromossomo doa um subconjunto (representado em cinza) de cada cenário para o receptor na mesma posição⁹. No segundo passo, o receptor elimina do seu próprio código os MPs correspondentes aos doados, como na Figura 56(b). Os MPs restantes no receptor são deslocados (processo *shift*) circularmente à esquerda dentro de cada cenário (e.g. o primeiro cenário irá executar deslocamentos somente nos oito primeiros *slots*) como ilustrado nas Figura 56(c) e (d). O resultado final do OX é mostrado na Figura 56(d).

⁹ Os genes a serem doados foram escolhidos de forma arbitrária.

Figura 56 - Ordered Crossover.

Fonte: Autor.

B. Non-Wrapping Ordered Crossover (NWOX)

O OX apresenta uma característica de deslocamento que pode atrapalhar na identidade dos pais: o *shift* circular desloca os genes da extrema esquerda para a extrema direita. Para preservar as posições absolutas na operação de cruzamento, o *Non-Wrapping Ordered Crossover*, NWOX (CICIRELLO, 2006), foi proposto, tentando evitar o problema do deslocamento que ocorre no OX.

Figura 57 - Non-wrapping ordered crossover.

Fonte: Autor.

As Figura 57(a) e (b) apresentam situações idênticas às vistas no caso do OX, sendo que na figura (b), estabelece-se a permanência e a ordem de Fc6, Fc3, Rc2(1) e Fc5. A diferença entre o OX e o NWOX fica clara no terceiro passo, onde o espaço para recepção de novos MPs é alocado entre os genes ordenados, como ilustrado na Figura 57(c).

Note-se que a ordem relativa do receptor é preservada. Ao invés de executar o *shift* e deslocar circularmente os módulos Fc6 e Fc3 para o lado direito do cromossomo, como no OX, estes são mantidos mais próximos às suas posições originais. Dessa forma, o resultado final do NWOX fica sendo o ilustrado na Figura 57(d).

C. Partially Matched Crossover (PMX)

O *Partially Matched Crossover*, PMX (GOLDBERG, 1989) utiliza um princípio de troca de posições dentro do cromossomo.

A Figura 58(a) apresenta o doador e o receptor, respectivamente, o superior e inferior. Na Figura 58(b) tem-se os primeiro processo de *swapping*, onde a posição do gene receptor troca a posição dos genes de forma a haver correspondência às mesmas posições do doador na área sombreada. Como o gene Fc4 foi trocado para uma região fora da zona sombreada, a sua posição deve ser alterada novamente, com uma nova troca, até que a região fique idêntica a do doador, como ilustrado na Figura 58(c).

Figura 58 - Partially matched crossover.

Fonte: Autor.

5.3.4.3. Técnicas de variação da população

Como discutido na seção 2.7.2.5, é possível que problemas a serem resolvidos com GAs apresentem convergência prematura para mínimos locais (HOLLAND, 1992). Como forma de lidar com este problema, para cada tipo de *crossover* a ser utilizado no algoritmo, serão aplicadas as duas técnicas de variação de população apresentados naquela seção: a geração aleatória de descendentes (ROG) e o algoritmo genético adaptativo (AGA).

A. Geração aleatória de descendentes (ROG)

Um grande número de indivíduos compartilhando o mesmo material genético pode levar à convergência prematura a mínimos locais. Há alta probabilidade de a operação de *crossover* ter como entrada dois indivíduos idênticos, que torna o processo de reprodução ineficaz.

Será adotada a técnica de geração aleatória de dependentes (*Random Offspring Generation* - ROG) (ROCHA; NEVES, 1999), apresentada na seção 2.7.2.5. É verificado, antes de cada processo de cruzamento, se os códigos genéticos dos pais são idênticos; caso positivo, um dos dois filhos é gerado de maneira aleatória.

B. Algoritmo Genético Adaptativo (AGA)

Outra solução bem-sucedida para lidar com a convergência prematura para mínimos locais é o AGA, inicialmente proposto por (SRINIVAS; PATNAIK, 1994), o qual posteriormente descrito na seção 2.7.2.5. O AGA não possui probabilidades de mutação e de *crossover* fixas; estas probabilidades são alteradas de acordo com a execução do algoritmo, na busca de um equilíbrio entre a convergência para um mínimo (local ou global) e exploração ampla no espaço de busca.

Como indicado nas equações (10) a (13) (pág. 51), os valores de $P(\text{crossover})$ e $P(\text{mutação})$ aumentam caso o *fitness* dos cromossomo de cruzamento ou mutação forem baixos, modulados pelos parâmetros k_1 , k_2 , k_3 e k_4 .

5.3.5. Escolha dos parâmetros de simulação

Nesta seção serão justificadas as escolhas dos parâmetros de simulação utilizados. Os principais parâmetros que devem ser definidos para a execução da GA são: o número de gerações, o tamanho da população, a probabilidade de *crossover* e a probabilidade de mutação. Para a AGA, deve-se definir os parâmetros de controle k_1 , k_2 , k_3 e k_4 .

Em relação às probabilidades, foram estudados diversos trabalhos clássicos que sugerem valores altos para a probabilidade de *crossover* e valores baixos para a probabilidade de mutação, como já descrito anteriormente na seção 2.7.2.3. Experimentalmente, foram alcançados melhores resultados com uma probabilidade de *crossover* de 0,7 e uma probabilidade de mutação de 0,02.

Para definir o número de gerações das simulações, foram feitas dez simulações com a variação pequena de MPs de cada uma das aplicações, utilizando o GA clássico e *crossover* OX¹⁰ e as probabilidades definidas no parágrafo anterior. Depois foram levantados gráficos com o custo médio de comunicação da população em cada geração. Os resultados estão representados na Figura 59.

É notório que a queda do custo é acentuada até a 100ª geração na maioria dos casos. Entre a 100ª e a 300ª geração há uma queda menos acentuada. Em todos os casos há uma tendência de estabilização na 300ª geração. Deve-se notar que o valor indica apenas a média de custo das gerações, sendo que o custo mínimo não necessariamente será encontrado nas últimas gerações.

Tomando-se em conta estes dados optou-se por utilizar o número de 300 gerações para os experimentos. A probabilidade de encontrar uma solução melhor

¹⁰ As avaliações podem ser estendidas aos outros casos.

crece com o número de gerações, porém, a partir deste limite, pouca variação ocorre, enquanto o esforço computacional aumenta significativamente.

Em relação ao tamanho da população, foi determinado, através de diversos experimentos, que uma população de 100 indivíduos é uma solução de compromisso entre diversidade da população e esforço computacional.

Os parâmetros de controle da técnica AGA foram determinados seguindo-se a sugestão dos seus autores (SRINIVAS; PATNAIK, 1994) com $k_1 = k_3 = 1$ e $k_2 = k_4 = 0,5$.

Figura 59 - Média do custo vs gerações: (a) Alpha (b) Beta (c) Gamma

(a)

(b)

(c)

Fonte: Autor.

5.3.6. Simulação e resultados

Para a simulação, foram utilizados todos os recursos descritos neste capítulo. Para cada tipo de *crossover* foram testados três tipos de GA, o normal e com as técnicas AGA e ROG de variação das populações, totalizando 81 casos de simulação. Para cada caso de simulação, foram feitas 10 simulações para obtenção do valor médio, uma vez que todas as simulações são diferentes, pois a população inicial é gerada aleatoriamente.

Os algoritmos foram simulados na plataforma MATLAB®, onde os códigos do GA foram desenvolvidos. A simulação do algoritmo de roteamento SXY para o cálculo do custo de mapeamento foi também realizada na mesma plataforma.

A Tabela 12 mostra os custos obtidos pelo GA, computados através da equação (34), em cada um dos casos. A primeira coluna mostra os três tipos de aplicações, enquanto suas respectivas variações são exibidas na segunda coluna. As colunas 3, 4 e 5 mostram os custos obtidos para o caso de *crossover* NWOX com a GA normal, e com as variações de população AGA e ROG, respectivamente; pela mesma sequência, nas colunas 6, 7 e 8 são apresentados os custos obtidos para o *crossover* PMX e o mesmo processo é repetido para as últimas 3 colunas em relação ao *crossover* OX.

Tabela 12 - Custos obtidos pelo GA.

Aplicação	MPs	NWOX			PMX			OX		
		Normal	AGA	ROG	Normal	AGA	ROG	Normal	AGA	ROG
ALPHA	Pequenos	1011,4	907,5	924,6	1003,6	900,7	968,8	974,0	903,0	942,9
	Variados	1477,1	1417,8	1520,6	1559,1	1386,8	1456,0	1530,5	1424,0	1557,0
	Grandes	1686,5	1630,1	1537,7	1697,5	1512,7	1636,5	1645,7	1594,3	1537,8
BETA	Pequenos	2576,0	2485,0	2953,3	2503,8	2395,0	2796,1	2569,0	2377,8	2644,3
	Variados	2550,7	2532,1	2827,1	2541,9	2533,3	2815,7	2607,8	2535,1	2764,4
	Grandes	2820,3	2771,8	3107,2	2729,9	2760,2	3148,2	2899,0	2797,5	3027,3
GAMMA	Pequenos	3932,7	3660,9	4407,8	3970,8	3679,6	4535,6	3926,2	3657,4	4386,7
	Variados	4072,8	3931,6	4565,6	4102,0	3888,6	4435,4	4141,2	3964,8	4463,6
	Grandes	4268,2	4109,2	4783,7	4159,2	3974,0	4822,2	4223,1	4049,6	4814,6

Fonte: Autor.

Para facilitar a visualização dos resultados, estes foram dispostos em uma escala de cor em cada linha, onde o verde mais escuro indica o melhor resultado e o vermelho mais forte, o pior.

Pode-se observar que o GA, sem a técnica de variação de população apresenta resultados de moderados a ruins quando comparados aos outros. Isso é explicado pela convergência prematura da população a um mínimo local.

A técnica ROG mostra alguns resultados bons para aplicação Alpha e resultados ruins para aplicações maiores. Foi possível notar, durante as simulações que o fato de a técnica impor a geração aleatória de descendentes causou forte variação na população de forma que se mostrou inadequado para este problema específico de mapeamento, impedindo a convergência para um mínimo global nas aplicações maiores.

Os melhores resultados foram obtidos com a técnica AGA de variação de população. Observou-se, durante os experimentos, que a sua mecânica de variação de população adapta-se melhor ao problema de mapeamento.

Quanto aos tipos de *crossover*, o PMX apresentou melhor resultados na maioria dos casos quando comparados aos outros. A melhor solução apresentada foi, portanto, PMX com variação de população AGA.

A Tabela 13 compara os resultados obtidos pelo GA, com os resultados ótimos do algoritmo semiexaustivo. A porcentagem indica o quanto o custo ficou acima da solução mínima. Este índice será denominado penalidade. A aplicação Gamma (destacada em laranja), foi comparada com o melhor resultado obtido com o próprio algoritmo genético, pois não foi possível obter o melhor custo com o algoritmo semiexaustivo (como visto na Tabela 11).

A melhor solução (PMX-AGA) apresentou penalidades entre 0,6% e 11,6%, chegando a uma penalidade média de 4%. Outros tipos de *crossover* com variação de população apresentaram resultados similares: o NWOX-AGA alcançou entre 0,6% e 20,3% de penalidade, com um valor médio de 6,6% e o OX-AGA apresentou entre 0,6% e 17,7% de penalidade com média de 5,6%.

Tabela 13 - Comparação dos resultados com os melhores mapeamentos.

Aplicação	MPs	NWOX			PMX			OX		
		Normal	AGA	ROG	Normal	AGA	ROG	Normal	AGA	ROG
ALPHA	Pequenos	15,3%	3,5%	5,4%	14,4%	2,7%	10,5%	11,1%	3,0%	7,5%
	Variados	12,2%	7,7%	15,5%	18,4%	5,3%	10,6%	16,2%	8,1%	18,2%
	Grandes	24,5%	20,3%	13,5%	25,3%	11,6%	20,8%	21,5%	17,7%	13,5%
BETA	Pequenos	9,0%	5,1%	24,9%	5,9%	1,3%	18,3%	8,7%	0,6%	11,9%
	Variados	1,3%	0,6%	12,3%	0,9%	0,6%	11,8%	3,6%	0,7%	9,8%
	Grandes	4,4%	2,6%	15,0%	1,0%	2,2%	16,5%	7,3%	3,5%	12,0%
GAMMA	Pequenos	7,5%	0,1%	20,5%	8,6%	0,6%	24,0%	7,3%	0,0%	19,9%
	Variados	4,7%	1,1%	17,4%	5,5%	0,0%	14,1%	6,5%	2,0%	14,8%
	Grandes	7,4%	3,4%	20,4%	4,7%	0,0%	21,3%	6,3%	1,9%	21,2%

Fonte: Autor.

Os mapeamentos da aplicação Alpha com MPs grandes apresentaram os piores resultados. Como a ocupação do SDR-NoC foi a maior entre as aplicações sintéticas, isso levou a uma maior dificuldade de convergência do GA para um mínimo global, sendo que um número muito grande de mapeamentos eram inválidos.

Os tempos médios de execução dos GAs é apresentado na Tabela 14, também obtidos em uma média de dez simulações. As simulações foram realizadas em um computador com processador Intel® Core® I7 - 3770 com 8GB de RAM. Os resultados mostram que o tempo de execução do GA evolui linearmente em relação ao número de MPs, resolvendo o problema da explosão apresentado na seção 5.3.3. Utilizando os parâmetros de simulação definidos na seção 5.3.5, os tempos variaram entre 40 segundos na aplicação Alpha e 113 segundos na aplicação Gamma.

Houve poucas diferenças, em termos de tempo de execução dos algoritmos, entre os tipos de *crossover*, sendo que o OX apresenta resultados ligeiramente mais lentos do que os outros tipos.

Tabela 14 - Tempos de execução das simulações em segundos.

Aplicação	MPs	NWOX			PMX			OX		
		Normal	AGA	ROG	Normal	AGA	ROG	Normal	AGA	ROG
ALPHA	Pequenos	43	42	42	42	40	40	46	46	45
	Variados	43	41	42	40	40	42	48	45	43
	Grandes	43	42	41	41	41	41	48	43	43
BETA	Pequenos	64	64	62	61	63	61	73	62	63
	Variados	63	62	62	62	64	62	73	62	62
	Grandes	63	61	61	59	61	62	73	62	63
GAMMA	Pequenos	95	95	96	94	96	95	108	96	94
	Variados	101	100	99	103	101	98	113	100	102
	Grandes	99	98	99	95	95	97	109	97	96

Fonte: Autor.

6. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma série de contribuições para mapeamento e posicionamento de SDRs baseados em redes intrachip. A seguir estas contribuições serão detalhadas, assim como as conclusões obtidas.

6.1. Contribuições

Entende-se que o trabalho traz contribuições no desenvolvimento no projeto de SDRs, assim como para métodos de mapeamento e posicionamento em arquiteturas SDR-NoCs. Métodos inéditos de mapeamento e posicionamento foram propostos tanto para arquiteturas SDR-NoCs simples como complexas.

As principais contribuições serão listadas a seguir:

- Desenvolvimento de um arcabouço teórico para arquiteturas SDR-NoCs. A classificação como simples ou complexa foi proposta na seção 2.6, de acordo com métricas de regularidade das topologias, da conexão dos roteadores e da homogeneidade dos elementos processantes.
- Formalização do problema de mapeamento e posicionamento de arquiteturas SDR-NoCs complexas, detalhados na seção 5.1. Foi proposto um novo modelo estendido de aplicação denominado EAPCG. O processo de mapeamento e posicionamento, assim como suas restrições, foram devidamente modelados utilizando-se ferramentas matemáticas.
- Construção de uma estratégia de mapeamento para arquiteturas SDR-NoCs simples que tornou possível a utilização de algoritmos clássicos de mapeamento para SDRs. A técnica de fusão de MPs nos APCGs,

apresentada no capítulo 4, mostrou-se válida quando a melhor solução, obtida com o B&B, foi comparada com a média de mapeamentos aleatórios tanto em alto nível quanto em baixo nível de simulação utilizando o Noxim.

- Desenvolvimento de um algoritmo genético para solucionar o problema de mapeamento em arquiteturas SDR-NoCs complexas apresentado integralmente no capítulo 5. Uma nova codificação que abrange tanto o posicionamento quanto o mapeamento foi criada. O algoritmo foi testado em uma grande variedade de aplicações sintéticas e validado com vários tipos de *crossover* e técnicas de variação da população.

O desenvolvimento do trabalho gerou duas publicações em periódicos internacionais. O estudo em SDR-NoCs simples foi publicado no *Latin American Symposium on Circuits and Systems* (LASCAS 2013) realizado em Cusco no Perú:

- GOMES FILHO, J.; STRUM, M.; WANG J. C. A strategy for mapping reconfigurable cores in NoCs. In: *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*. IEEE, 2013. **Proceedings...** p. 1-4.

A formalização do problema para arquiteturas SDR-NoCs complexas e a solução utilizando um algoritmo semi-exaustivo foram publicadas no RECONFIG, congresso sobre reconfiguração, realizado em 2013, no México.

- GOMES FILHO, J. WANG J. C. Exploring the Problems of Placement and Mapping in NoC-based Reconfigurable Systems. In: *RECONFIG 2013, IEEE, 2013, Proceedings...* p. 1-4.

6.2. Conclusões

Após a realização desta tese, foi possível se debruçar sobre a metodologia utilizada e os resultados obtidos, concluindo-se o seguinte:

- A estratégia de mapeamento para SDR-NoCs simples mostrou-se eficiente quando a melhor solução de mapeamento foi comparada com a média de soluções aleatórias e controladas. Quando a comparação foi feita em nível TLM, notou-se redução de até 38% de atraso médio na comunicação e de até 41% de consumo de energia da NoC. Demonstrada a eficiência desta estratégia, pode-se aplicá-la em outros algoritmos clássicos de mapeamento.
- A codificação do problema de mapeamento e posicionamento da arquitetura SDR-NoC complexa criada se mostrou eficiente. Através desta, tornou-se possível codificar ao mesmo tempo: mapeamento, posicionamento e cenários fixos e reconfiguráveis, sendo possível assim, modelar todos os aspectos de uma arquitetura SDR-NoC complexa.
- A comparação entre a melhor solução de mapeamento e posicionamento, obtida com o algoritmo exato, com a média de mapeamentos aleatórios e controlados, comprovou a importância do problema.
- A solução com algoritmos genéticos apresentou excelentes resultados, muito próximos do mínimo global, principalmente utilizando-se do *crossover* PMX (GOLDBERG, 1989) e da técnica de variação de população AGA (SRINIVAS; PATNAIK, 1994). Estas combinações obtiveram custos de comunicação, em média, 4% acima da melhor

solução possível, considerando a codificação compacta adotada na seção 5.3.1.

- A solução com algoritmos genéticos apresentou, além da eficiência qualitativa das soluções, tempos de execução baixos, ficando entre 41s e 113s, para os exemplos propostos. Mais importante do que os valores baixos de tempo de execução, foi a evolução linear em relação ao crescimento do problema. Assim, torna-se possível tratar um problema de grande complexidade.
- A solução utilizando o algoritmo genético clássico mostra convergência e chega a resultados medianos devido à convergência prematura para mínimos locais.
- A solução que utiliza a técnica de geração aleatória de descendentes ROG, (ROCHA; NEVES, 1999), resultou em uma população diversa demais, interferindo na convergência do algoritmo para um mínimo, tanto local quanto global.

6.3. Trabalhos futuros

Como propostas de trabalhos futuros, sugere-se:

- Adaptar a metodologia de mapeamento e posicionamento de arquiteturas SDR-NoC complexas às outras arquiteturas onde há problemas de bloqueio, e outros aspectos, nas propostas de (JOVANOVIĆ; TANOUGAST; WEBER, 2008; RANA et al., 2010; KILLIAN, 2012).

- Otimizar a metodologia de mapeamento e posicionamento de arquiteturas SDR-NoC complexas para minimizar outros parâmetros além do custo de comunicação. Entre os parâmetros a serem minimizados poder-se-ia considerar: latência e ciclos de reconfiguração dinâmica e potência consumida.
- Testar a metodologia de mapeamento e posicionamento de arquiteturas SDR-NoC complexas em sistemas de simulação TLM ou RTL.
- Ampliar os mapeamentos SDR-NoC estáticos propostos nesta tese, para mapeamentos SDR-NoC dinâmicos (em tempo de execução).

REFERÊNCIAS

AHMADINIA A.; BOBDA, C.; BEDNARA, M.; TEICH J. A new approach for on-line placement on reconfigurable devices. In: 18th International Parallel and Distributed Processing Symposium, **Anais...** 2004, p. 134-140.

AHMADINIA, A.; BOBDA, C.; FEKETE, S. P.; TEICH, J.; VAN DER VEEN, J. C. Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices. **IEEE Transactions on Computers**, v. 56, n. 5, p. 673-680, 2007.

BENINI, L. DE MICHELI, G. Networks on Chips: a new soc paradigm, **IEEE Computer**, p. 70-78, Janeiro, 2002.

BERETTA, I.; RANA, V.; ATIENZA, D.; SCIUTO, D. Run-time mapping of applications on FPGA-based reconfigurable systems, ISCAS, 2010, **Proceedings...** p. 3329–3332, 2010.

BERETTA, I.; RANA, V.; ATIENZA, D.; SCIUTO, D. A mapping flow for dynamically reconfigurable multi-core system-on-chip design, **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 30, n.8, p.1211-1224, Agosto, 2011.

BERTOZZI, D.; JALABERT, A.; SRINIVASAN MURALI; TAMHANKAR, R.; STERGIOU, S.; BENINI, L.; DE MICHELI, G.; NoC synthesis flow for customized domain specific multiprocessor systems-on-chip, **Parallel and Distributed Systems, IEEE Transactions on**, v.16, n.2, p. 113- 129, Fevereiro, 2005.

BERTOZZI, S.; ACQUAVIVA, A.; BERTOZZI, D.; POGGIALI, A. Supporting task migration in multi-processor systems-on-chip: a feasibility study. In: Design, Automation and Test in Europe (DATE), **Proceedings...** Munich, Alemanha. Março, 2006. p.15-20.

BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of network-on-chip. **ACM Computing Surveys (CSUR)**, v. 38, n. 1, p. 1, 2006.

BOBDA, C.; AHMADINIA, A.; MAJER, M.; TEICH, J.; FEKETE, S.; VAN DER VEEN, J.; DyNoC: A dynamic infrastructure for communication in dynamically reconfigurable devices, **Proceedings...** FPL 2005. p. 153- 158, 24-26 Agosto, 2005

BOBDA, C. **Introduction to reconfigurable computing**. Dordrecht: Springer, 2007.

BRASSARD, G.; BRATLEY, P. **Algorithmics: theory & practice**. Prentice-Hall, Inc., 1988.

CARVALHO, E.; CALAZANS, N.; MORAES, F. Heuristics for dynamic task mapping in noc-based heterogeneous mpsoCs. In: Annual IEEE/IFIP International Workshop on Rapid Systems Prototyping (RSP), **Anais...** Porto Alegre, Brasil. Maio, 2007. p.34-40.

CARVALHO, E. L. **Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC**. 155 p. + apêndice. Tese (Doutorado) - Pontifícia Universidade Católica do Rio Grande do Sul. Rio Grande do Sul, 2009.

CHEN, D.; CONG, J.; PAN, P. FPGA design automation: A survey. Foundations and trends in electronic design automation, **Proceedings...** v. 1, n. 3, p. 139-169, 2006.

CHOU, C-L.; MARCULESCU, R. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In: IEEE/ACM/IFIP international conference on 145 Hardware/software codesign and system synthesis (CODES+ISSS), **Proceedings...** Salzburg, Austria. Outubro, 2007. p.161-166.

CICIRELLO, V. A. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In: 8th annual conference on Genetic and evolutionary computation, **Proceedings...** ACM, 2006. p. 1125-1132.

CORNELIUS, C.; TIMMERMAN, D. Development and operation of networks-on-chip. In: IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC). **Proceedings...** Nice, França. Outubro, 2006. p.19-23.

DALLY, W. J.; TOWLES, B. P. **Principles and practices of interconnection networks**. Elsevier, 2004.

DAVIS, L. D. **Handbook of genetic algorithms**. [S.l.]: Van Nostrand Reinhold, 1991.

DICK, R.; RHODES, D.; WOLF, W. **TGFF: Task graphs for free**. In: International Workshop on Hardware/Software Co-design (CODES/CASHE). Washington, EUA. Março, 1998. **Proceedings...** p.97-101.

DUATO, J.; YALAMANCHILI, S.; N.I., Lionel M. **Interconnection networks: An engineering approach**. Morgan Kaufmann, 2003.

ESKINAZI, R. **Uma Metodologia Para Escalonamento De Tarefas De Tempo Real Em Arquiteturas Dinamicamente Reconfiguráveis**, Tese (Doutorado) - Universidade Federal de Pernambuco, Recife, 2005.

ESQUIAGOLA, J. **Reconfiguração dinâmica de um controlador Bluetooth banda base**, Dissertação (Mestrado) - Escola politécnica, Universidade de São Paulo. São Paulo, 2006.

FAZZINO F.; PALESI M.; PATTI D.; **Noxim: network-on-chip simulator**, <http://www.noxim.org> (Acesso em Julho de 2014).

GAREY, M.R.; GRAHAM, R.L.; ULLMAN, J.D. An Analysis of Some Packing Algorithms. **Combinatorial Algorithms**. Junho, 1973. p.39-47.

GLOVER, F. Tabu search: a tutorial. **Interfaces**, v.20, n.4, p.74–94, Julho, 1990.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization & Machine Learning**. Addison-Wesley, 1989.

GONZALEZ, J.A.Q. **Uma Metodologia de Projetos para Circuitos com Reconfiguração Dinâmica de Hardware Aplicada a Support Vector Machine**, 2006, Tese (Doutorado) - Escola politécnica, Universidade de São Paulo, São Paulo, 2006.

GOMES FILHO, J. **Aplicação de Técnicas de Reconfiguração Dinâmica a Projeto de Máquina de Vetor Suporte (SVM)**. 2010, 89p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2010.

GOMES FILHO, J.; RAFFO, M.; STRUM, M.; WANG JIANG CHAU; A general-purpose dynamically reconfigurable SVM, Programmable Logic Conference (SPL), 2010 VI Southern, **Proceedings...** p.107-112, 24-26 Março, 2010.

GOMES FILHO, J.; STRUM, M.; WANG J. C. A strategy for mapping reconfigurable cores in NoCs. In: Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on. IEEE, 2013. **Proceedings...** p. 1-4.

GOMES FILHO, J. WANG J. C. Exploring the Problems of Placement and Mapping in NoC-based Reconfigurable Systems. In: RECONFIG 2013, IEEE, 2013, **Proceedings...** p. 1-4.

HENKEL, J.; PARAMESWARAN, S. (Ed.). **Basics of reconfigurable computing**. Dordrecht: Springer, 2007. 525 p.

HOLLAND J. Reproductive plans and genetic operators, em **Adaptation in Natural and Artificial Systems**, ch. 6. Cambridge, MA: MIT Press, pp. 89–118, 1992.

HORTA, E. L. **Comutador ATM reconfigurável dinamicamente através de hardware**. 2002. 94p. + 2 apêndices. Tese (Doutorado) – Escola politécnica, Universidade de São Paulo, São Paulo, 2002.

HU, J., MARCULESCU, R. Energy-aware mapping for tile-based NOC architectures under performance constraints, Asia South Pacific Design Automation Conference, Kitakyushu, Japan, **Proceedings...** p. 233–239. Janeiro de 2003.

HU, J.; MARCULESCU, R. DyAD: smart routing for networks-on-chip. In: 41st annual Design Automation Conference, **Proceedings...** ACM, 2004. p. 260-263.

HU, J., MARCULESCU, R. Energy- and performance-aware mapping for regular NoC architectures, **IEEE Trans. Comput.-Aided Design Integr. Circuits Systems**, v. 24, n. 4, p. 551–562, Abril, 2005.

ITRS - International Technology Roadmap for Semiconductors. **2012 International Technology Roadmap for Semiconductors Design Report**. Disponível em <http://www.itrs.net/Links/2012Winter/1205%20Presentation/DesignSD_12052012.pdf> (acesso em Julho de 2014).

ITURBE, X.; BENKRID, K.; HONG, C.; EBRAHIM, A.; ARSLAN, T.; MARTINEZ, I. Runtime Scheduling, Allocation, and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence, **International Journal of Reconfigurable Computing (IJRC)**, v. 2013, Article ID905057, 2013.

JOVANOVIC, S.; TANOUGAST, C.; WEBER, S.; BOBDA, C. CuNoc: A scalable dynamic NoC for dynamically reconfigurable FPGAs. In: 17th International Conference on Field Programmable Logic and Applications (FPL), **Proceedings...** 2007, p. 753-756.

JOVANIVIC, S. et al. CuNoC: A dynamic scalable communication struture for dynamically reconfigurable FPGAs. In: **Microprocessors and Microsystems**, v. 33, n. 1, p. 24-36, 2008.

JOVANOVIC, S.; TANOUGAST, C.; WEBER, S. A New High-Performance Scalable Dynamic Interconnection for FPGA-based Reconfigurable Systems. International Conference on Application-Specific Systems Architectures and Processors, ASAP, **Proceedings...** p. 61-66, 2008.

KILLIAN, C.; TANOUGAST, C.; MONTEIRO, F.; DANDACHE, A. (2012). A new efficient and reliable dynamically reconfigurable network-on-chip. **Journal of Electrical and Computer Engineering**, 2012.

KIM, J.; SIEGEL, H.; MACIEJEWSKI, A.; EIGENMANN, R. Dynamic Mapping in Energy Con-strained Heterogeneous Computing Systems. In: International Parallel and Distributed Processing Symposium (IPDPS), **Proceedings...** Colorado, EUA. Abril, 2005. p.64-73.

KIM, J.; SHIVLE, S.; SIEGEL, H.; MACIEJEWSKI, A.; BRAUN, T.; SCHNEIDER, M.; TIDEMAN, S.; CHITTA, R.; DILMAGHANI, R.; JOSHI, R. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. **Journal of Parallel and Distributed Computing**, v.67 n. 2 Fevereiro, 2007. p.154-169.

KIRKPATRICK S., GELATT C., VECCHI M.: Optimization by simulated annealing, **Science**, 220, (4598), pp. 671–680, 1983.

KOJIMA, L **Metodologia de Projeto de Sistemas Dinamicamente**

Reconfiguráveis. Dissertação (Mestrado) - Escola politécnica, Universidade de São Paulo. São Paulo, 2007.

KUMAR, S.; JANTSCH, A.; SOININEN, J.-P.; FORSELL, M.; MILLBERG, M.; OBERG, J.; TIENSYRJA, K.; HEMANI, A.; A network on chip architecture and design methodology, *VLSI, 2002. Proceedings...* IEEE Computer Society Annual Symposium on, p.105-112.

LAWLER, E. L.; WOOD, D. E. Branch-and-bound methods: A survey. **Operations research**, v. 14, n. 4, p. 699-719, 1966.

LEI, T.; KUMAR, S. A Two-step Genetic Algorithm for Mapping Task Graphs to Network on Chip Architecture. In: Euromicro Symposium on Digital System Design: Architectures, Methods and Tools (DSD), **Proceedings...** Turquia. Setembro, 2003. pp.180-187.

LIU, H.; WONG, D. F. Circuit partitioning for dynamically reconfigurable FPGAs. ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, Monterey, CA, **Proceedings...** p. 187-194, Fevereiro, 1999.

LODI, A.; MARTELLO, S.; MONACI, M. Two-dimensional packing problems: A survey. **European Journal of Operational Research**, v. 141, n. 2, p. 241-252, 2002.

LU, S. L.; YIANNACOURAS, P.; SUH, T.; KASSA, R.; KONOW, M. A Desktop Computer with a Reconfigurable Pentium® **ACM Transactions on Reconfigurable Technology and Systems (TRETs)**, v. 1, n. 1, Março, 2008.

LYSAGHT, P.; DUNLOP, J. Dynamic reconfiguration of field programmable gate arrays. In: The 1993 International Workshop on Field-Programmable Logic and Applications, **Proceedings...** 1993, p. 82-94.

LYSAGHT, P.; STOCKWOOD, J. A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays. In: **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 4, n. 3, p. 381-390, 1996.

MAJER, M.; TEICH, J.; AHMADINIA, A.; BOBDA, C. The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based computer. **Journal of VLSI Signal Processing**. v. 47, n. 1, 2007.

MANOLACHE, S.; ELES, P.; PENG, Z. Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC. In: ACM/IEEE Design Automation Conference (DAC). **Proceedings...** CA, EUA. Junho, 2005. pp.266-269.

MARCON, C.; MORENO, E.; CALAZANS, N.; MORAES, F. Evaluation of Algorithms for Low Energy Mapping onto NoCs. In: IEEE International Symposium on Circuits and Systems (ISCAS). **Proceedings...** Louisiana, EUA. Maio, 2007. pp.389-392.

MARCULESCU, R.; OGRAS, U.Y.; LI-SHIUAN PEH; JERGER, N.E.; HOSKOTE, Y.; Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives, **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, Janeiro, 2009, v.28, n.1, p.3-21.

MATLAB, <<http://www.mathworks.com/products/matlab/>> (Acesso em Julho de 2014). 2014.

MORAES, F.; CALAZANS, N.; MELLO, A.; MÖLLER, L.; OST, L. Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. **Integration, the VLSI Journal**, v..38 n.1. Outubro, 2004. p.69-93.

MORANDI M.; NOVATI, SANTAMBROGIO M. D.; SCIUTO, D. Core allocation and relocation management for a self dynamically reconfigurable architecture, in IEEE Computer Society Annual Symposium on VLSI: Trends in VLSI Technology and Design (ISVLSI '08), **Proceedings...** p. 286–291, Abril, 2008.

MOSKEWICZ, M. W.; MADIGAN, C. F.; ZHAO, Y.; ZHANG, F.; MALIK, S.; Chaff: Engineering an efficient SAT solver, in Design Automation Conference, **Proceedings...** 2001, p. 530–535.

MURALI, S.; DE MICHELI, G.; Bandwidth-constrained mapping of cores onto NoC architectures, Design, Automation and Test in Europe Conference and Exhibition, 2004. **Proceedings...** v.2, p. 896- 901 16-20 Fevereiro, 2004.

MURALI, S.; BENINI, L.; DE MICHELI, G. Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees. In: Asia and South Pacific Design Automation Conference (ASP-DAC). **Proceedings...** Shanghai, China. Janeiro, 2005. pp.27-32.

MURALI, S. COENEN, M. RADULESCU, A. GOOSSENS, K. DE MICHELI, G. A methodology for mapping multiple use-cases onto networks on chips Design Automation and Test Europe Conference, **Proceedings...** Março 2006, p. 118–123.

NGOUANGAL, A.; SASSATELLI, G.; TORRES, L.; GIL, T.; SOARES, A.; SUSIN, A. A contextual resources use: a proof of concept through the APACHES platform. In: IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS), **Proceedings...** Praga, República Tcheca. Abril, 2006. p.42-47.

NOLLET, V.; AVASARE, P.; MIGNOLET, J.; VERKEST, D. Low Cost Task Migration Initiation in a Heterogeneous MP-SoC. In: Design, Automation and Test in Europe (DATE), **Proceedings...** Munich, Alemanha. Março, 2005. p.252-253.

OHKAWA, T.; UETAKE, D.; YOKOTA, T.; OOTSU, K.; BABA, T. Reconfigurable and hardwired ORB engine on FPGA by Java-to-HDL synthesizer for realtime application. **ACM SIGARCH Computer Architecture News** v. 41 n. 5 (2014): p. 77-82.

PAULIN, P.; KNIGHT, J. Force-directed scheduling for the behavioral synthesis of asic's. **IEEE Transactions on CAD** v. 6, n. 8, p. 661–679, 1989.

PIONTECK, T.; KOCH, R.; ALBRECHT, C. Applying partial reconfiguration to networks on chips. In: 16th International Conference on Field Programmable Logic and Applications (FPL), **Proceedings...** 2006, p. 155-160.

RANA, V.; MURALI, S.; ATIENZA, D.; SANTAMBROGIO, M. D.; BENINI, L.; SCIUTO, D. Minimization of the reconfiguration latency for the mapping of applications on FPGA-based systems, CODES-ISSS, **Proceedings...** 2009, p.120–130.

RANA, V.; ATIENZA, D.; SANTAMBROGIO M. D.; SCIUTO, D.; De MICHELI, G. A Reconfigurable Network-on-Chip Architecture for Optimal Multi-Processor SoC Communication. VLSI-SoC: Design Methodologies for SoC and SiP, **Anais...** p.232–250 v. 313. 2010

RAFFO, M. **Desenvolvimento de um sistema dinamicamente reconfigurável baseado em redes intrachip e ferramenta para posicionamento de módulos**, 94p. + 3 Apêndices. Dissertação (Mestrado) - Escola politécnica, Universidade de São Paulo. São Paulo, 2010.

RAFFO, M.; GOMES FILHO, J.; STRUM, M.; WANG JIANG CHAU; A placement tool for a NOC-based dynamically reconfigurable system, *Programmable Logic Conference (SPL), 2010 VI Southern*, **Proceedings...** p.47-52, 24-26 Março, 2010.

ROCHA, M.; NEVES J. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. **Multiple Approaches to Intelligent Systems**. Springer Berlin Heidelberg, 127-136. 1999.

SAHA, A.; AMITABHA S., Reconfigurable DSP Processor-A New Computing Platform for Software Radio Applications. Software Engineering (WCSE), 2013 Fourth World Congress on. IEEE, **Proceedings...** 2013.

SEPÚLVEDA, M.J. **Estimativa de desempenho de uma NoC a partir de seu modelo em SystemC - TLM**. 2006. 172 p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2006.

SEPÚLVEDA, M.J.; PIRES, R. ; STRUM, M.; WANG J.C. Implementation of QoSS (Quality-of-Security Service) for NoC-Based SoC Protection. **Lecture Notes in Computer Science**, v. 6340, p. 187-201, 2010.

SEPÚLVEDA, M. J. **Projeto de estruturas de comunicação Intrachip baseadas em NoC que implementam serviços QoS e segurança.** 172p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2011.

SRINIVAS, M.; PATNAIK, LALIT M. Adaptive probabilities of crossover and mutation in genetic algorithms. **Systems, Man and Cybernetics, IEEE Transactions on**, v. 24, n. 4, p. 656-667, 1994.

SRINIVASAN, K.; CHATHA, K. A technique for low energy mapping and routing in network-on-chip architectures. In: International Symposium on Low Power Electronics and Design (ISLPED). **Proceedings...** CA, EUA. Agosto, 2005. p.387-392.

WANG, H-S; ZHU, X.; PEH, L-S; MALIK, S. Orion: A Power-Performance Simulator for Interconnection Networks. IEEE/ACM International Symposium on Microarchitecture (MICRO'02), 2002, **Proceedings...** p. 294.

WRONSKI, F.; BRIÃO, E.; WAGNER, F. Evaluating Energy-aware Task Allocation Strategies for MPSoCs. In: IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES), **Proceedings...** Braga, Portugal. Outubro, 2006. p. 215-224.

XIE, J.; CHEN, X.; ZHOU, B.; HU, X.; TAN, X.; REN, R.; XIA, S. A reconfigurable wireless health monitoring system with undecimated wavelet transform implemented. In: Electronics, Computer and Applications, 2014 IEEE Workshop. **Proceedings...** p. 848-851. Maio, 2014.

XILINX. Disponível em: <<http://www.xilinx.com>> Acesso em: julho de 2014.

ZHANG, X.; NG, K. W. A review of high-level synthesis for dynamically reconfigurable FPGAs. In: **Microprocessors and Microsystems**, 2000, p. 199-211.

APÊNDICE A - DADOS DAS APLICAÇÕES SINTÉTICAS

Este apêndice mostra os detalhes das aplicações sintéticas utilizadas nos testes do capítulo 5. As Tabelas 15, 17 e 19 mostram o tamanho dos *cores* em largura (w) e altura (h) para as aplicações Alpha, Beta e Gamma, respectivamente. Já as tabelas 16, 18 e 20 detalham a comunicação entre os *cores* das mesmas aplicações, na seguinte forma: as colunas 1 e 4 contém os *cores* de origem, enquanto que nas colunas 2 e 5 encontram-se os *cores* de destino correspondentes; as colunas 3 e 6 apresentam os valores de taxa de transmissão.

Tabela 15 - Tamanho dos *cores* na aplicação Alpha.

Alpha	Pequeno		Variado		Grande	
MP	w_i	h_i	w_i	h_i	w_i	h_i
Fc1	2	2	1	1	2	2
Fc2	2	2	2	5	2	3
Fc3	2	2	1	1	2	2
Fc4	1	2	3	2	3	2
Fc5	1	1	3	3	3	1
Fc6	1	1	1	1	2	4
Rc1(1)	1	1	1	1	3	3
Rc2(1)	2	1	2	5	4	1
Rc1(2)	2	2	3	3	2	2
Rc2(2)	1	1	1	1	3	1
Rc1(3)	1	1	3	3	3	3
Rc2(3)	2	1	2	1	2	1
Rc3(3)	1	2	1	1	1	2

Fonte: Autor.

Tabela 16 - Tabela de comunicação da aplicação Alpha.

Do MP	Para o MP	Mbit/s
Fc1	Fc2	4
Fc1	Fc4	50
Fc1	Rc1(2)	12
Fc2	Fc3	6
Fc3	Fc6	10
Fc3	Rc1(3)	13
Fc4	Rc1(1)	5
Fc4	Fc5	30
Fc4	Rc2(2)	18
Rc1(1)	Rc2(1)	20
Rc2(1)	Rc1(1)	20
Fc5	Rc1(1)	10
Fc5	Fc3	7
Fc5	Fc6	11
Rc1(2)	Fc2	15
Rc2(2)	Fc4	18
Rc1(3)	Rc2(3)	14
Rc2(3)	Rc3(3)	14
Rc3(3)	Fc6	15

Fonte: Autor.

Tabela 17 - Tamanho dos cores na aplicação Beta.

Beta	Pequeno		Variado		Grande	
MP	w_i	h_i	w_i	h_i	w_i	h_i
Fc1	2	3	1	1	2	3
Fc2	3	2	3	2	3	2
Fc3	1	1	1	2	2	2
Fc4	1	1	3	3	1	1
Rc1(1)	1	1	1	1	1	1
Rc2(1)	2	2	3	1	1	1
Rc3(1)	2	2	1	2	2	1
Rc1(2)	2	2	3	2	2	2
Rc2(2)	2	2	1	1	2	2
Rc3(2)	2	2	1	2	2	2
Rc4(2)	2	2	3	3	3	3
Rc5(2)	2	2	1	1	2	2
Rc1(3)	2	2	3	2	2	2
Rc2(3)	2	2	1	1	2	3
Rc1(4)	2	2	3	3	2	2
Rc2(4)	2	2	1	1	2	4
Rc3(4)	2	2	3	2	2	2
Rc4(4)	1	1	1	1	1	1

Fonte: Autor.

Tabela 18 - Tabela de comunicação da aplicação Beta.

Do MP	Para o MP	Mbit/s
Fc1	Fc2	12
Fc2	Fc1	23
Fc2	Fc3	32
Fc3	Fc1	22
Fc4	Fc3	33
Fc3	Rc1(1)	5
Rc2(1)	Fc2	3
Rc1(1)	Rc3(1)	100
Fc1	Rc1(2)	23
Rc2(2)	Rc3(2)	55
Fc3	Rc2(2)	8
Rc2(2)	Rc3(2)	15
Rc3(2)	Rc4(2)	16
Rc4(2)	Rc5(2)	17
Rc5(2)	Fc4	18
Fc3	Rc1(3)	50
Rc1(3)	Rc2(3)	12
Rc2(3)	Fc3	25
Fc1	Rc1(4)	22
Rc1(4)	Rc2(4)	44
Rc2(4)	Rc3(4)	33
Rc3(4)	Rc4(4)	34
Rc4(4)	Fc2	55

Fonte: Autor.

Tabela 19 - Tamanho dos *cores* na aplicação Gamma.

Gamma	Pequeno		Variado		Grande	
MP	w_i	h_i	w_i	h_i	w_i	h_i
Fc1	2	2	2	1	2	2
Fc2	2	2	1	2	2	2
Fc3	1	1	1	1	1	1
Fc4	1	1	1	3	1	1
Fc5	1	1	3	1	1	1
Fc6	2	2	1	2	2	2
Fc7	2	1	2	1	3	1
Rc1(1)	2	2	1	1	2	2
Rc2(1)	1	2	1	3	1	2
Rc3(1)	1	1	1	4	1	1
Rc1(2)	1	1	4	1	2	2
Rc2(2)	2	1	2	3	2	1
Rc3(2)	1	2	1	3	1	2
Rc4(2)	1	1	1	1	3	2
Rc5(2)	2	2	1	1	2	2
Rc1(3)	2	1	2	1	2	1
Rc2(3)	1	2	2	3	2	2
Rc3(3)	2	2	1	1	2	2
Rc4(3)	1	2	3	1	1	3
Rc5(3)	1	2	3	1	3	1
Rc1(4)	1	1	3	1	3	2
Rc2(4)	2	2	3	2	2	2
Rc3(4)	1	2	1	1	2	3
Rc1(5)	2	1	3	2	2	3
Rc2(5)	2	2	1	1	2	2
Rc3(5)	1	1	3	1	3	3

Fonte: Autor.

Tabela 20 - Tabela de comunicação da aplicação Gamma.

Do MP	Para o MP	Mbit/s	Do MP	Para o MP	Mbit/s
Fc1	Fc2	12	Rc2(2)	Rc4(2)	20
Fc1	Rc1(2)	22	Fc1	Rc1(3)	11
Fc2	Fc1	23	Fc3	Rc2(3)	21
Fc2	Fc3	32	Fc5	Rc3(3)	35
Fc3	Fc1	22	Fc7	Rc4(3)	16
Fc4	Fc3	33	Rc1(3)	Rc5(3)	15
Fc3	Fc5	15	Rc2(3)	Rc3(4)	26
Fc6	Fc2	33	Fc4	Rc3(4)	16
Fc5	Fc7	100	Fc2	Rc1(4)	28
Fc1	Rc1(1)	23	Fc4	Rc2(4)	19
Rc2(1)	Fc4	55	Fc6	Rc3(4)	10
Fc3	Rc2(1)	8	Rc1(4)	Rc3(4)	22
Fc2	Rc2(2)	20	Rc2(4)	Rc1(4)	18
Fc3	Rc3(2)	10	Fc3	Rc1(5)	6
Fc4	Rc4(2)	15	Fc5	Rc2(5)	18
Fc5	Rc5(2)	32	Fc7	Rc3(5)	8
Rc1(2)	Rc5(2)	26	Rc3(5)	Rc1(5)	36

Fonte: Autor.