Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

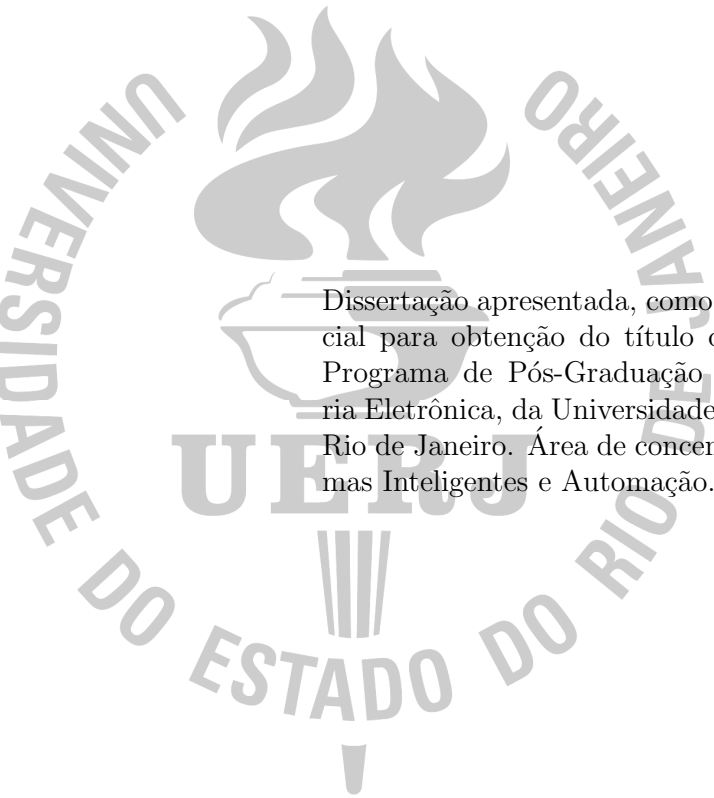
Marcus Vinícius Carvalho da Silva

Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos

Rio de Janeiro
2009

Marcus Vinícius Carvalho da Silva

Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof^a. Dr^a. Nadia Nedjah
Co-orientadora: Prof^a. Dr^a. Luiza de Macedo Mourelle

Rio de Janeiro
2009

CATALOGAÇÃO NA FONTE
UERJ/REDE SIRIUS/CTC/B

S663 Silva, Marcus Vinícius Carvalho da.
Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos/Marcus Vinícius Carvalho da Silva. – 2009.
178 f. : il.

Orientadora: Nadia Nedjah.
Co-orientadora: Luiza de Macedo Mourelle.
Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia: f. 154 – 163.

1. Redes embutidas – Teses. 2. Alocação de IPs. 3. Mapeamento de IPs. 4. Otimização multiobjetivo. I. Nedjah, Nadia. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 004:530.145

Assinatura _____ Data _____

Marcus Vinícius Carvalho da Silva

Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em

Banca Examinadora:

Prof^a. Dr^a. Nadia Nedjah (Orientadora)
Faculdade de Engenharia, UERJ

Prof^a. Dr^a. Luiza de Macedo Mourelle (Co-orientadora)
Faculdade de Engenharia, UERJ

Prof. Dr. Heitor Silvério Lopes
Programa de Engenharia Elétrica e Informática Industrial, UTFPR

Prof^a. Dr^a. Priscila Machado Vieira Lima
Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ

Rio de Janeiro
2009

DEDICATÓRIA

À minha mãe pelo exemplo de perseverança.

AGRADECIMENTOS

A Deus por me dar a capacidade de escolher e por me guiar em minhas escolhas;

Às mulheres da minha vida, Maria Brígida, Celia Regina, Tessi Carvalho e Renata Correa;

À minha família pelo conforto, suporte e carinho recebido por todos estes anos;

Aos meus queridos amigos do *Bonde* e aos queridos *Amigos de Uma Vida Toda*, pela compreensão e apoio durante este período de grande ausência;

Às minhas orientadoras, professora Nadia Nedjah e professora Luiza de Macedo Mourelle, pela orientação que me foi dada quando estava diante de tantos caminhos, me auxiliando para que eu não me perdesse;

Aos colegas do mestrado Rodrigo Martins da Silva, pelo apoio; Marcos Paulo M. Araujo, pela motivação e Sergio Oliveira Costa, pelos momentos de descontração;

Aos funcionários e professores do Programa de Pós-graduação em Engenharia Eletrônica - PEL, pela ajuda e ensinamentos;

À Universidade do Estado do Rio de Janeiro - UERJ por ter me recebido novamente, desta vez na qualidade de mestrando.

RESUMO

SILVA, Marcus Vinícius Carvalho da. *Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos*. 2009. 178f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

No projeto de redes embutidas baseado em plataforma, a alocação e o mapeamento de IPs são duas etapas típicas e de alta complexidade. Ferramentas de auxílio são essenciais para o desenvolvimento de projetos baseados em plataforma. A alocação de IPs consiste em alocar os IPs adequados para executarem as tarefas de uma aplicação específica. Baseado em características da aplicação e em características dos IPs disponíveis, deve-se escolher o melhor conjunto de IPs capaz de executar a aplicação. O mapeamento de IPs consiste na organização do espaço físico que os IPs ocuparão da rede embutida. Baseado em características de comunicação entre IPs e da plataforma de destino, deve-se escolher o melhor mapeamento possível. Estas etapas de projeto são consideradas problemas de alta complexidade e difíceis de serem resolvidos até mesmo por ferramentas computacionais. Nesta dissertação, o melhor conjunto de IPs e o melhor mapeamento serão aqueles que ocupam a menor área, apresentam o menor consumo de energia e executam a aplicação específica no menor tempo possível. Para encontrar soluções que atinjam estes objetivos, é proposta uma alocação evolutiva de IPs e um mapeamento evolutivo de IPs, utilizando algoritmos evolucionários multiobjetivos. Estes algoritmos representam uma técnica estocástica de busca e otimização voltada para problemas de otimização multiobjetivo, como é o caso dos problemas de alocação e mapeamento de IPs. Os algoritmos utilizados são o NSGA-II e o microGA. Na etapa de alocação evolutiva de IPs, estes algoritmos recebem os dados de IPs, obtidos de um repositório, e os dados das tarefas da aplicação, obtidos do grafo de tarefas da aplicação. A combinação destes dados forma indivíduos que representam soluções para o problema e serão submetidos a operadores genéticos. Na etapa de mapeamento evolutivo de IPs, os algoritmos recebem as alocações de IPs, obtidas da etapa anterior, e dados da plataforma da rede embutida, onde as alocações serão mapeadas. Para preservar características dos indivíduos adquiridas durante a etapa de alocação, é proposto um operador genético de recombinação por deslocamento, inspirado no mecanismo biológico de partenogênese, e um operador genético de mutação interna. Fatores de impacto no desempenho da rede embutida são analisados e utilizados para avaliar as soluções obtidas por ambos os algoritmos. Os resultados obtidos mostram-se competitivos com os de outra ferramenta de mapeamento. Uma comparação de desempenho entre as implementações do NSGA-II e o microGA é realizada com base na quantidade de alocações e mapeamentos ótimos obtidos e os tempos de busca de cada algoritmo.

Palavras-chave: redes embutidas, alocação de IPs, mapeamento de IPs, otimização multiobjetivo, algoritmos genéticos.

ABSTRACT

Two typical steps on a platform-based NoC project are the IP assignment and the IP mapping. These are highly complex steps and the use of platform-based EDA tools is essential in such cases. The IP assignment step is when the best set of IPs is assigned of executing a given application. The selection of the best set of IPs, capable to execute the given application, is driven by applications features and by IPs features. The IP mapping step is when the selected IPs are disposed in the NoC platform. Considering the communication among IPs and the target platform, the best mapping is chosen. These stages of design problems are considered highly complex and difficult to be solved even by computational tools. In this dissertation, the best set of IPs and the best mapping will be those that occupy the smallest area, have the lowest energy consumption and perform a specific application in the shortest time possible. To find solutions that achieve these goals, we propose an evolutionary IP assignment and an evolutionary IP mapping, using multi-objective evolutionary algorithms. These algorithms are based on a stochastic technique of search and optimization aimed to multi-objective optimization problems, such as the assignment and mapping problems. The chosen algorithms are the NSGA-II and microGA. In the evolutionary IP assignment step, these algorithms receive the IP data, obtained from a repository, and the data about the application tasks, obtained from the application's task graph. From the combination of these data, arise individuals that represent solutions to the problem and will be subjected to genetic operators. In the evolutionary IP mapping step, these algorithms receive the assignment of IPs obtained from the previous step and data from the network platform wherein the assignments will be mapped. To preserve the characteristics of the solutions acquired during the stage of assignment, a shift crossover, based on the biological mechanism of parthenogenesis, and an inner mutation operators are proposed. Factors that have an impact on the performance of the embedded network are analyzed and used to evaluate the solutions obtained by both algorithms. The obtained results are competitive with the results obtained by existing mapping tool. A comparison of performance between the implementations of NSGA-II and microGA is performed based on the amount of optimal assignments and mappings obtained and the search time spent by each algorithm.

Keywords: network-on-chip, IP assignment, IP mapping, multi-objective optimization, genetic algorithms.

LISTA DE FIGURAS

1 Arquitetura básica de um SoC 10
2 Fluxo típico de projeto de sistemas embutidos 12
3 Arquitetura básica de um SoC integrada à diferentes bibliotecas 15
4 Ilustração da metodologia baseada em plataforma 16
5 Plataforma SoC para uma câmera digital 17
6 Camadas de um projeto baseado em plataforma 21
7 NoC de tipo malha com nove recursos 23
8 Arquitetura interna de um *switch* 24
9 Exemplos de topologias de redes diretas 27
10 Exemplos de topologias de redes indiretas 28
11 Fluxo típico de projeto de sistemas embutidos para plataforma NoC 31
12 Fluxo proposto de projeto de sistemas embutidos para plataforma NoC 32

13 Mapeamento de um POM 35
14 Transformação Polinomial de dois problemas 36
15 Exemplos de fronteiras ótimas de Pareto convexas 38
16 Exemplos de fronteiras ótimas de Pareto côncavas 39
17 Exemplo de fronteira ótimas de Pareto convexa discreta 40
18 Ilustração do conceito de $\varepsilon-$dominância 41
19 Classificação das técnicas de busca e otimização a um objetivo 42

20 Estrutura de dados e terminologia de um AE 49
21 Seleção pelo giro da roleta 52
22 Recombinação em um ponto 52
23 Mutação binária 53
24 Etapas de um AE 54
25 Etapas de um AEM 55
26 Classificação das técnicas de otimização multiobjetivo 56
27 Desempenho do método da soma ponderada com dois objetivos 57
28 Ilustração do desempenho do método da soma ponderada quando a fronteira ótimas de Pareto é côncava 58
29 Ilustração do desempenho do método de restrição ε quando a fronteira ótimas de Pareto possui região côncava 59
30 Ilustração da otimização multiobjetivo baseada nas teorias de Nash 62
31 Processo de classificação do MOGA 64
32 Distância de aglomeração 73
33 População aleatória, memória populacional e população inicial 77
34 Resultados obtidos com o NSGA-II e microGA para os *benchmarks*: *SCH* e *FON* 80
35 Resultados obtidos com o NSGA-II e microGA para os *benchmarks*: KUR e *ZDT4* 81

36 Exemplo de grafo de tarefas 87

37 Impacto de diferentes alocações de IPs 89
38 Dinâmica do processo de alocação evolutiva de IPs 91
39 Representação em XML do repositório de IPs 93
40 Representação em XML do grafo de tarefas 93
41 Codificação do cromossomo para a alocação de IPs 94
42 Ilustração do funcionamento do operador de mutação de alocações 96
43 Representação dos valores médios dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S 104
44 Número de soluções não dominadas encontradas pelo NSGA-II e microGA, para as aplicações do E3S, e seus respectivos tempos de busca 105
45 Comparação de desempenho entre o NSGA-II e o microGA de acordo com os dados da Tabela 13 106
46 Representação das fronteiras ótimas de Pareto para as aplicações 3, 5, 10 e 12 . 107
47 CGA da aplicação *auto-indust-tg2* obtido pela alocação evolutiva usando NSGA-II 108
48 CGA da aplicação *auto-indust-tg2* obtido pela alocação evolutiva usando microGA 108

49 Impacto de diferentes mapeamentos de IPs 114
50 Rota do nó (1,0) até o nó (2,2), de acordo com o algoritmo XY 117
51 Ilustração dos atrasos básicos da rede: t_R, t_l e t_L 120
52 Rota para enviar o sétimo pacote que do recurso r_3 até o recurso r_2 indicando as contribuições dos *switches* e canais 121
53 Dinâmica do processo de mapeamento evolutivo de IPs 121
54 Codificação do cromossomo para o mapeamento de IPs 122
55 Método de recombinação por deslocamento 124
56 Método de mutação interna 124
57 Impacto do mapeamento de tarefas paralelas no tempo de execução da aplicação 127
58 Representação dos valores médios dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S 138
59 Representação dos valores mínimos dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S 139
60 Número de mapeamentos não dominados encontrados pelo NSGA-II e microGA, para as aplicações do E3S, e seus respectivos tempos de busca 140
61 Implementação generalizada da aplicação *SegImag* 142
62 Exemplo de uma imagem com 640×480 *pixels* segmentada em quatro partes . 144
63 GT da aplicação *SegImag* para 4 segmentos de imagem 145
64 Alocações da aplicação *SegImag* que deram origem aos mapeamentos de consumo mínimo de energia usando o NSGA-II e microGA 147
65 Comparação de mapeamentos entre NSGA-II e microGA com o CAFES 147
66 Ilustração dos mapeamentos encontrados pelo NSGA-II, microGA e CAFES para a aplicação *SegImag* 148

LISTA DE TABELAS

1 Diferenças entre núcleos de *software*, *firmware* e *hardware* 14

2 POMs utilizados como *benchmarks* 80

3 Ilustração da explosão exponencial de número de alocações factíveis 87
4 Impacto das alocações ilustradas na Figura 37 88
5 Objetivos concorrentes e colaborativos 97
6 Características das aplicações do repositório do E3S e da complexidade dos problemas de alocação correspondentes 99
7 Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o NSGA-II para as aplicações do E3S 100
8 Mínimos dos objetivos para as soluções não dominadas, encontradas pelo NSGA-II 101
9 Amostra de alocações não dominadas, encontradas pelo NSGA-II, apresentando valores mínimos nos objetivos, conforme listado na Tabela 8 102
10 Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o microGA para as aplicações do E3S 103
11 Amostra de alocações não dominadas, encontradas pelo microGA, apresentando valores mínimos nos objetivos, conforme listado na Tabela 8 103
12 Tempo gasto aproximadamente pelos dois algoritmos na geração de uma única alocação 105
13 Desempenho do NSGA-II e do microGA em relação à quantidade de soluções encontradas e o tempo de busca aproximado para cada solução 106

14 Ilustração da explosão exponencial de número de mapeamentos factíveis para uma alocação 113
15 Número total de mapeamentos possíveis para a aplicação e alocações mostradas na Figura 37 do Capítulo 4 113
16 Impacto dos mapeamentos ilustrados na Figura 49 115
17 Complexidade dos problemas de mapeamento para as aplicações do repositório do E3S 132
18 Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de recursos mapeados com o NSGA-II para as aplicações do E3S 133
19 Mínimos dos objetivos para as soluções não dominadas, encontradas pelo NSGA-II 134
20 Amostra de mapeamentos não dominados, encontrados pelo NSGA-II, apresentando valores mínimos nos objetivos, conforme listado na Tabela 19 135
21 Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o microGA para as aplicações do E3S 136
22 Mínimos dos objetivos para as soluções não dominadas, encontradas pelo microGA 136
23 Amostra de mapeamentos não dominados, encontrados pelo microGA, apresentando valores mínimos nos objetivos, conforme listado na Tabela 19 137

24 Tempo gasto aproximadamente pelos dois algoritmos na geração de um único mapeamento não dominado . . . 140
25 Desempenho do NSGA-II e do microGA em relação à quantidade de soluções encontradas e o tempo de busca aproximado para cada solução . . . 141
26 Tamanhos de mensagens para uma imagem 640 × 480 dividida em quatro segmentos . . . 143
27 Fluxo de tarefas nos PAs, no PC e na ME . . . 143
28 Resultados obtidos pelo NSGA-II e pelo microGA para a aplicação SegImag . . . 146
29 Mapeamentos obtidos pelo NSGA-II, microGA e CAFES a partir de alocações não dominadas . . . 146
30 Características dos mapeamentos obtidos para a aplicação SegImag . . . 148

31 Lista dos processadores do repositório . . . 164
32 Lista das tarefas usadas no repositório . . . 165
33 Lista das tarefas usadas no repositório – (Continuação) . . . 166
34 Lista de tarefas por processador . . . 167
35 Lista de processador por tarefa . . . 168
36 Lista de processador por tarefa – Continuação . . . 169
37 Características de IPs para uso no exemplo ilustrativo da Figura 37 . . . 170
38 Características de IPs para uso no exemplo ilustrativo da Figura 37 (Continuação) 170
39 Características de IPs para uso no exemplo ilustrativo da Figura 37 (Continuação) 170

LISTA DE ALGORITMOS

1 Algoritmo Evolucionário 50
2 Algoritmo de Nível de Goldberg 63
3 Torneio por seleção do NPGA 66
4 SPEA 67
5 Poda através de *clusterização* 68
6 Ordenação Rápida pela Dominância 71
7 Cálculo da distância de aglomeração 74
8 NSGA-II 75
9 microGA 79
10 $f_1(Q)$ – MesmaFonteDifAlvos 129
11 $f_2(Q)$ – DifFonteMesmoAlvo 129

SUMÁRIO

INTRODUÇÃO 1

1 REDES EMBUTIDAS **8**
1.1 Sistemas Embutidos 9
1.1.1 Projeto de sistemas embutidos 11
1.1.2 Blocos de propriedade intelectual 13
1.1.3 Metodologia baseada em plataforma 16
1.1.4 Particionamento 18
1.1.4.1 Particionamento estrutural 18
1.1.4.2 Particionamento funcional 19
1.1.5 Plataforma de *hardware* 20
1.1.6 Plataforma de *software* 20
1.2 Redes Embutidas 21
1.2.1 Arquitetura interna 22
1.2.2 Modelo de referência OSI 25
1.2.3 Topologias 27
1.3 Metodologia de Síntese para Plataforma NoC 29
1.4 Considerações Finais do Capítulo 31

2 OTIMIZAÇÃO MULTIOBJETIVO **33**
2.1 Conceitos Básicos de Otimização Multiobjetivo 33
2.1.1 Problema de otimização multiobjetivo 34
2.1.2 Complexidade 36
2.1.3 Conceitos de Pareto 37
2.2 Métodos de Busca e Otimização 41
2.2.1 Taxonomia 41
2.2.2 Estratégias de otimização multiobjetivo 45
2.3 Considerações Finais do Capítulo 47

3 OTIMIZAÇÃO EVOLUCIONÁRIA MULTIOBJETIVO **48**
3.1 Algoritmos Evolucionários 48
3.1.1 Definição formal de AEs 50
3.1.2 Operadores genéticos 51
3.2 Algoritmos Evolucionários Multiobjetivo 52
3.2.1 Métodos baseados na seleção por agregação 55
3.2.1.1 Soma ponderada 56
3.2.1.2 Restrição ε 57
3.2.1.3 Programação por meta 58
3.2.2 Métodos baseados na seleção por critério 60
3.2.2.1 VEGA – *Vector Evaluated Genetic Algorithm* 60
3.2.2.2 Otimização lexicográfica 61
3.2.2.3 Teoria dos jogos – Equilíbrio de Nash e Jogo Não Cooperativo 61

3.2.3 Métodos baseados na seleção por dominância . . . 63
3.2.3.1 MOGA – *Multiple Objective Genetic Algorithms* . . . 63
3.2.3.2 NPGA – *Niched Pareto Genetic Algorithms* . . . 65
3.2.3.3 SPEA – *Strength Pareto Evolutionary Algorithms* . . . 66
3.2.3.4 NSGA e NSGA-II – *Non-Dominated Sorting Genetic Algorithms* . . . 69
3.2.3.5 MicroGA – *Micro Genetic Algorithm* . . . 75
3.3 Seleção de Métodos para Implementação . . . 78
3.4 Considerações Finais do Capítulo . . . 81

4 ALOCAÇÃO EVOLUTIVA DE IPS EM PLATAFORMA NOC . . . 83
4.1 O Problema de Alocação de IPs . . . 84
4.1.1 Repositório de IPs . . . 85
4.1.2 Grafo de tarefas . . . 85
4.1.3 Complexidade do problema e impacto da alocação de IPs . . . 86
4.1.4 Grafo de caracterização de aplicação . . . 90
4.2 Alocação Evolutiva de IPs . . . 90
4.2.1 Codificação . . . 91
4.2.1.1 Codificação do repositório . . . 91
4.2.1.2 Codificação de grafos de tarefas . . . 92
4.2.1.3 Codificação dos indivíduos . . . 94
4.2.2 Operadores genéticos . . . 94
4.2.3 Avaliação de aptidão . . . 95
4.2.3.1 Área . . . 97
4.2.3.2 Tempo de execução . . . 97
4.2.3.3 Consumo de energia . . . 98
4.3 Resultados de Experimentais . . . 98
4.3.1 Resultados do NSGA-II . . . 99
4.3.2 Resultados do micro-GA . . . 100
4.3.3 Comparação dos resultados . . . 102
4.4 Considerações Finais do Capítulo . . . 107

5 MAPEAMENTO EVOLUTIVO EM PLATAFORMA NOC . . . 110
5.1 O Problema de Mapeamento de IPs . . . 111
5.1.1 Complexidade do problema de mapeamento de IPs . . . 112
5.1.2 Impacto do mapeamento de IPs . . . 113
5.2 Modelo de energia . . . 114
5.3 Modelo de Tempo de Comunicação . . . 116
5.3.1 Roteamento e chaveamento . . . 116
5.3.1.1 Algoritmos de roteamento . . . 116
5.3.1.2 Técnicas de chaveamento . . . 118
5.4 Mapeamento Evolutivo de IPs . . . 120
5.4.1 Codificação . . . 122
5.4.2 Operadores genéticos . . . 123
5.4.3 Avaliação de aptidão . . . 124
5.4.3.1 Área . . . 125
5.4.3.2 Tempo de execução . . . 126
5.4.3.3 Consumo de energia . . . 129
5.4.3.4 Restrição quanto a formação de *hot spots* . . . 131
5.5 Resultados Experimentais . . . 131
5.5.1 Resultados do NSGA-II . . . 132

5.5.2 Resultados do microGA . . . 133
5.5.3 Comparação dos resultados . . . 135
5.5.3.1 NSGA-II *vs.* microGA . . . 135
5.5.3.2 Aplicação de segmentação de imagem . . . 139
5.5.3.3 NSGA-II e microGA *vs.* CAFES . . . 144
5.6 Considerações Finais do Capítulo . . . 149

6 CONCLUSÕES E TRABALHOS FUTUROS . . . 150
6.1 Considerações Finais . . . 150
6.2 Trabalhos Futuros . . . 153

REFERÊNCIAS . . . 154

APÊNDICE A – Processadores e Tarefas do Repositório . . . 164

APÊNDICE B – Aplicações de Referência . . . 171

INTRODUÇÃO

A PROFECIA feita por Gordon E. Moore, em 1965, previa um aumento exponencial da quantidade de componentes em um circuito integrado, o que acarretaria na redução do custo de manufatura dos processadores e na duplicação anual [1] da capacidade de processamento dos mesmos (MOORE, 1965). Com o cumprimento deste fato nos anos seguintes, a profecia ganhou *status* de lei, se tornando conhecida como a *Lei de Moore*. Embora a larga escala de integração traga benefícios para a indústria de componentes eletrônicos, esta também traz consigo novos desafios de projeto.

Os desafios dizem respeito ao *layout* dos circuitos, aos aspectos elétricos que variam com o grau de integração de componentes e com prazos cada vez mais rigorosos estabelecidos pelo mercado. Para auxiliar o projetista a vencer estes desafios, são utilizadas ferramentas computacionais de auxílio a projeto, conhecidas como EDA (*Electronic Design Automation*). O objetivo destas ferramentas é automatizar etapas do projeto de circuitos integrados de modo a otimizar a implementação final a partir de uma descrição abstrata especificada pelo projetista (EDWARDS *et al.*, 1997).

A complexidade das ferramentas de EDA aumenta juntamente com a complexidade dos circuitos integrados. Para acelerar o trabalho do projetista, estas ferramentas precisam lidar com descrições cujo grau de abstração é cada vez mais alto, mas os algoritmos de otimização utilizados não são tão eficientes para lidar com os desafios atuais (OGRAS; HU; MARCULESCU, 2005). Neste contexto, surgiram novos algoritmos para aumentar a eficiência das ferramentas de EDA, juntamente com novos paradigmas na concepção de circuitos integrados.

Um paradigma que é muito explorado é a concepção de que um sistema computacional, constituído de um processador, uma memória, interfaces de entrada e saída e interconexões entre estes componentes, pode ser construído inteiramente em um circuito integrado. Este paradigma levou a concepção dos sistemas embutidos, mais conhecidos como SoCs (*Systems-on-Chip*). Além dos componentes já mencionados, um SoC também pode incluir componentes

[1] Originalmente, em 1965, Moore previu uma duplicação anual da capacidade de processamento, sendo que mais tarde, ele reconsiderou este intervalo, afirmando que seria de 18 meses.

dedicados que auxiliam o processador na aceleração de algumas tarefas críticas para os sistema como um todo. Estes componentes são chamados de *núcleos*, *blocos de propriedade intelectual* ou simplesmente IPs (*Intellectual Property*).

Um IP é um componente normalmente projetado para realizar funções específicas, como por exemplo, processamento digital de sinais, operações em ponto flutuante e criptografia de dados. Um IP também pode implementar componentes como um processador ou uma memória. Uma das vantagens principais na utilização de SoCs é a combinação da implementação em *software*, tendo como elemento central o processador, e da implementação em *hardware*, caracterizada pela utilização de IPs específicos. A facilidade na alteração de *softwares* e a possibilidade de reaproveitar e combinar diferentes IPs, facilita o projeto e reduz o tempo de lançamento de um produto no mercado (BENINI; MICHELI, 2002).

Em consonância com a previsão da Lei de Moore, a capacidade de combinar diferentes IPs em SoCs aumentou consideravelmente. Com o aumento do número de IPs utilizados em um único sistema, surge o problema de comunicação entre estes. A comunicação entre IPs pode ser feita basicamente através de ligações ponto-a-ponto ou através de um barramento de comunicação compartilhado. A eficiência destes métodos de interconexão de IPs diminui de acordo com o aumento da quantidade dos IPs interconectados. Com o objetivo de melhorar o desempenho da comunicação em SoCs, surgiu o paradigma de *redes embutidas*, ou simplesmente NoCs (*Network-on-Chips*).

Redes embutidas são consideradas um avanço natural dos sistemas embutidos (KUMAR *et al.*, 2002). NoCs e SoCs compartilham aspectos comuns importantes, como a reutilização de componentes existentes, o que facilita e acelera o projeto de novos componentes e dispositivos eletrônico. NoCs podem ser formadas por IPs interconectados ou até mesmo por SoCs interconectadas. Os componentes de uma NoC são implementados em recursos. Estes recursos compartilham uma plataforma de interconexão que é responsável pela comunicação entre os recursos. A comunicação é estabelecida através de *switches* que são responsáveis pelo envio e recebimento de dados dos recursos.

A arquitetura de rede de uma NoC se assemelha à arquitetura de rede *Ethernet* (METCALFE; BOGGS, 1976), amplamente utilizada em redes locais atualmente. Os recursos se comunicam através de mensagens, formadas por pacotes, que são encaminhadas de acordo com a técnica de chaveamento e o algoritmo de roteamento implementados nos *switches* da rede. Este tipo de arquitetura torna a rede escalável e reutilizável.

Tanto SoCs, quanto NoCs, são implementados para executar uma determinada aplica-

ção. Uma aplicação pode ser entendida como um conjunto de tarefas que devem ser executadas respeitando-se determinadas dependências de dados e controle. Uma maneira de expressar as características de uma aplicação é através de um grafo de tarefas (GT) orientado e acíclico. Neste tipo de grafo, uma tarefa da aplicação é representada como um vértice e as suas dependências como arestas orientadas. Uma das maneiras de descrever um sistema, de forma abstrata, para uma ferramenta de EDA é através do GT da aplicação a ser implementada.

A ferramenta de EDA, usando a descrição abstrata fornecida pelo projetista, deve ser capaz de reduzir o nível de abstração da descrição através de sucessivas etapas de otimização até que seja alcançada uma implementação final de acordo com as especificações de projeto. Entre as etapas de otimização, existem algumas que são classificadas como problemas complexos a serem resolvidos na área de projeto de NoCs. Dois destes problemas são representados pelas etapas de alocação de IPs e do mapeamento de IPs (OGRAS; HU; MARCULESCU, 2005).

O problema de alocação de IPs consiste em reduzir a abstração do GT de modo que cada tarefa ou conjunto de tarefas, sejam associadas a um IP adequado. Após a alocação de IPs, o grafo de tarefas é chamado de grafo de caracterização de aplicação (GCA), pois a partir deste, algumas características menos abstratas sobre a implementação da aplicação podem ser determinadas. As diferentes combinações de IPs determinam estas características de implementação, como o tempo de execução, a área ocupada no circuito integrado e o consumo de energia.

A complexidade deste problema varia de acordo com as características do GT e a quantidade de IPs que podem ser selecionados para executar as tarefas. As ferramentas de EDA devem auxiliar o projetista na identificação de IPs que podem ser atribuídos a mais de uma tarefa, de modo a reduzir a área ocupada, o consumo de energia e o tempo de execução, sem que o desempenho da aplicação seja comprometido. As características obtidas a partir do GCA são utilizadas para guiar a etapa de mapeamento.

O problema de mapeamento de IPs consiste em reduzir a abstração do GCA de modo que cada IP alocado seja mapeado em um recurso da NoC. A complexidade deste problema varia de acordo com a quantidade de IPs a serem mapeados e o número de recursos disponíveis na plataforma NoC. Por exemplo, para mapear um GCA com 9 IPs em uma NoC com 9 recursos, existem 9! = 362.880 mapeamentos possíveis. Adicionando apenas um IP e um recurso, a possibilidade de mapeamentos aumenta para 10! (3.628.800). Levando em conta a Lei de Moore, é fácil perceber que o problema de mapeamento de IPs tende a piorar com o tempo.

O mapeamento influencia diretamente nas características de operação da NoC. Em geral, recursos que trocam mais mensagens devem ficar mais próximos para reduzir o atraso e consumo de energia devidos à comunicação. Identificar estes recursos e encontrar o mapeamento ideal para eles, é um problema de otimização de difícil solução. Além de reduzir o consumo de energia, o projetista também precisa considerar o tempo gasto em comunicação entre os recursos e a área final ocupada pelos IPs, *switches* e pelos canais de comunicação necessários.

Os problemas de alocação e mapeamento de IPs são problemas de busca e otimização que além de possuírem alta complexidade, devem avaliar diferentes objetivos para cada solução encontrada. Problemas deste tipo são chamados de problemas de *otimização multiobjetivos* (POMs). Normalmente estes problemas apresentam objetivos que são conflitantes, isto é, a otimização de um objetivo implica na degradação do outro. Nestes casos, não é possível obter apenas uma solução ótima, mas sim um conjunto de soluções ótimas que representam um balanço entre os objetivos em questão.

Um dos métodos de identificar e classificar as soluções ótimas de um POM é o método de dominância (PARETO, 1896). Neste método são identificadas as soluções que não são piores do que nenhuma outra solução em relação a todos os objetivos. Estas soluções são ditas não dominadas por nenhuma outra solução encontrada. Do conjunto de soluções não dominadas, o projetista da NoC pode selecionar uma alocação e/ou mapeamento que atenda a critérios específicos, sabendo que não existe nenhuma outra solução melhor considerando os mesmos critérios.

Ferramentas de EDA que não consideram a avaliação de múltiplos objetivos nos problemas de alocação e mapeamento de IPs, tendem a ser mais rápidas na solução do problema mas por outro lado, podem encontrar soluções que sejam ótimas em relação a um objetivo e péssimas em relação a outro. Algoritmos clássicos de busca e otimização, baseados em métodos determinísticos, que consideram os múltiplos objetivos de um problema, consomem um tempo de busca excessivo e, na maioria das vezes, inviável para encontrar soluções ótimas. Esta ineficiência dos algoritmos determinísticos de busca e otimização está relacionada com o espaço de busca dos POMs, que normalmente variam exponencialmente (FOGEL; OWENS; WALSH, 1966) (GAREY; JOHNSON, 1979).

Para lidar com este tipo de problemas, foram desenvolvidos algoritmos de busca e otimização capazes de tratar em tempo polinomial, problemas cujas entradas variam exponencialmente. Estes algoritmos são baseados em técnicas estocásticas de busca, ao invés de técnicas determinísticas. São baseados em heurísticas ou metaheurísticas, como é o caso dos

algoritmos que utilizam a técnica de computação evolucionária.

A computação evolucionária compreende técnicas de otimização que simulam os processos evolucionários naturais. O campo de computação evolucionária engloba técnicas como os algoritmos genéticos, estratégias evolucionárias e programação evolucionária, sendo que estes três também são chamados de algoritmos evolucionários. Estas técnicas são baseadas na teoria evolucionista de Charles Darwin que explica a evolução das espécies através da seleção natural e da sobrevivência do mais apto (GOLDBERG, 1989a). A vantagem destas técnicas em relação às técnicas classificadas como deterministas é a capacidade de resolver em tempo polinomial problemas de alta complexidade (VELDHUIZEN, 1999).

Esta dissertação aborda a otimização das etapas de alocação e mapeamento de IPs no projeto de uma NoC. Como descritas anteriormente, estas etapas se tratam de POMs de alta complexidade e portanto, técnicas determinísticas de busca e otimização podem consumir muito tempo computacional na busca de soluções ótimas. Para lidar com estes problemas, serão utilizados algoritmos genéticos, que baseados na teoria da evolução das espécies, buscam por soluções ótimas a partir de sucessivas etapas evolutivas.

Os algoritmos genéticos utilizados tratam-se de dois algoritmos genéticos multiobjetivos, que são específicos para a solução de POMs. Um deles é o NSGA-II (*Nondominated Sorting Genetic Algorithm - II*) (DEB *et al.*, 2002) e o microGA (*Micro Genetic Algorithm*) (COELLO; PULIDO, 2001). Estes algoritmos foram escolhidos porque apresentaram bons desempenho na resolução de POMs que são comumente utilizados como *benchmarks* e possuem poucos parâmetros de configuração. Ambos os algoritmos selecionam e classificam as soluções utilizando o conceito de dominância, e dessa forma, o conjunto de soluções não dominadas, de acordo com os objetivos de interesse, é apresentado ao projetista da NoC no final do processo evolutivo; este então deverá selecionar a solução que melhor lhe atende.

Para resolver um POM utilizando algoritmos genéticos, é preciso representar o problema na forma de um cromossomo. Este cromossomo será tratado pelo algoritmo como um indivíduo e operações genéticas de recombinação e mutação serão realizadas para simular o processo de evolução ao longo de gerações. A representação das soluções dos problemas em questão, na forma de indivíduos, é um trabalho que requer percepção do problema. A representação pode influenciar, de modo positivo ou negativo, no desempenho e na eficácia do algoritmo genético. Primeiramente será abordado o problema de alocação de IPs a partir de um GT e em seguida será abordado o problema de mapeamento de IPs em uma NoC.

Nesta dissertação, serão apresentadas as representações utilizadas no problema de aloca-

ção de IPs (SILVA; NEDJAH; MOURELLE, 2009a) (SILVA; NEDJAH; MOURELLE, 2009c) e aquelas utilizadas no problema de mapeamento de IPs (SILVA; NEDJAH; MOURELLE, 2009b). Também serão apresentados dois operadores genéticos desenvolvidos para as operações de recombinação e de mutação na etapa de otimização do mapeamento de IPs. O operador de recombinação por deslocamento, inspirado no processo biológico de partenogênese (WATTS *et al.*, 2006), e o operador de mutação interna (SILVA; NEDJAH; MOURELLE, 2009b), apresentam a característica de alterar a solução quanto ao mapeamento mas preservando a alocação obtida na etapa de alocação de IPs.

Esta dissertação contribui para a área de engenharia eletrônica através da apresentação dos resultados obtidos a partir de métodos não determinísticos e multiobjetivos, aplicados em duas etapas típicas de projetos de NoCs. Os resultados apontam que estas técnicas não determinísticas podem ser utilizadas com sucesso para auxiliar no projeto de NoCs. A abordagem multiobjetivo deve ser considerada para que aspectos importantes do projeto não sejam negligenciados. O conteúdo desta dissertação está organizado em seis capítulos e dois apêndices, cujos conteúdos são descritos a seguir.

O Capítulo 1 apresenta a arquitetura de sistemas embutidos (SoCs) e o método de projeto destes sistemas que utilizam *hardware* e *software* em conjunto. A metodologia de projeto baseada em plataforma e o fluxo típico de etapas que fazem parte deste tipo de projeto. As etapas do projeto que formam o escopo desta dissertação são destacadas. Apresenta os blocos de propriedade intelectuais (IPs) e a arquitetura de comunicação de uma rede embutida (NoC). Faz uma adaptação do modelo OSI desenvolvido para redes de computadores, identificando os elementos da NoC nas camadas correspondentes. São apresentadas topologias de rede que são utilizadas na implementação de NoCs e é apresentada a metodologia de síntese de uma plataforma NoC.

O Capítulo 2 introduz os conceitos básicos de otimização de problemas multiobjetivos e define formalmente um POM. Descreve a complexidade destes problemas e apresenta o conceito de dominância introduzido por Pareto (PARETO, 1896). Os métodos de busca e otimização mais utilizados são apresentados e são divididos de acordo com a técnica utilizada por cada um.

O Capítulo 3 apresenta métodos evolucionários para a solução de POMs, com destaque para os algoritmos evolucionários multiobjetivos. Previamente, os algoritmos evolucionários para problemas de objetivo único são descritos e definidos formalmente. Em seguida, os algoritmos evolucionários multiobjetivos são apresentados e são divididos de acordo com o método de seleção e classificação utilizado por cada um. Por fim, os algoritmos escolhidos para tra-

tarem dos problemas de alocação de IPs e de mapeamento de IPs são apresentados com mais detalhes.

O Capítulo 4 detalha o problema de alocação de IPs, apresenta o repositório de IPs que será utilizado e define o modelo de grafo de tarefas que será utilizado; apresenta a complexidade deste problema e o impacto causado por diferentes alocações. É apresentado o grafo de caracterização de aplicação que é resultado da alocação de IPs. A alocação evolutiva de IPs é detalhada e as codificações do repositório, do grafo de tarefas e dos indivíduos são apresentadas. Os operadores genéticos utilizados são definidos e explicados. Os objetivos de interesse são escolhidos de acordo com as características do problema e as funções de avaliação de cada objetivo são definidas. Os resultados experimentais obtidos pelas implementações do NSGA-II e do microGA são expostos e por fim são comparados.

O Capítulo 5 detalha o problema de mapeamento de IPs, detalha a complexidade deste problema e ilustra o impacto causado por diferentes mapeamentos. É apresentado um modelo de energia utilizado para computar o consumo de energia da plataforma de comunicação de uma NoC e também é apresentado um modelo de tempo de comunicação, que é utilizado para computar o tempo gasto pelos elementos da plataforma de comunicação. Algoritmos de roteamento e de chaveamento são apresentados. O processo de mapeamento evolutivo de IPs é detalhado e a codificação dos indivíduos e os operadores genéticos são apresentados. As funções de avaliação de cada objetivo de interesse são detalhadas, assim como a restrição utilizada nesta etapa. Uma aplicação de segmentação de imagem, que será submetida ao processo evolutivo de mapeamento de IPs, é apresentada juntamente com seu grafo de tarefas. Os resultados experimentais obtidos pelas implementações do NSGA-II e do microGA são expostos e por fim são comparados entre si e com uma ferramenta de mapeamento de IPs que utiliza um método estocástico de otimização chamado de recozimento simulado (RUSSEL; NORVIG, 1995).

O Capítulo 6 apresenta um apanhado do trabalho desenvolvido e uma análise dos resultados obtidos. Além disso, são determinadas direções que apontam para trabalhos futuros que dão continuidade a esta pesquisa.

Complementando os dados descritos nos capítulos mencionados, esta dissertação possui dois apêndices. O Apêndice A apresenta a lista dos processadores do repositório utilizado, assim como as tarefas que cada processador pode executar. O Apêndice B apresenta as aplicações de referência e que foram utilizadas como *benchmarks*. Estas aplicações estão divididas em cinco conjuntos e a característica de cada conjunto de aplicações é explicada.

Capítulo 1

REDES EMBUTIDAS

O AVANÇO das tecnologias de fabricação de circuitos integrados (CIs) viabilizou a integração em alta escala de vários elementos de *hardware* por CI, possibilitando a implementação de sistemas computacionais completos em um único componente. Esse tipo de implementação é geralmente chamada de *sistema embutido*, *sistema embarcado* ou SoC de *System-on-Chip*. Com a crescente necessidade por mobilidade, estes sistemas ficaram cada vez menores e mais complexos, passando a executar tarefas antes executadas apenas por grandes computadores. Seguindo o exemplo de grandes computadores, estes sistemas embutidos passaram a se comunicar usando redes estruturadas de interconexão implementadas em *hardware* dentro do mesmo CI, juntamente com as funcionalidades do sistema embutido. Essas redes de interconexão são comumente chamadas de *redes embutidas* ou NoCs de *Networks-on-Chip*. Este capítulo apresenta a evolução destes sistemas embutidos, como eles são constituídos e os desafios de projeto existentes.

É comum na literatura o termo sistema embutido ser utilizado para denominar um componente (CI) e também o dispositivo que contem o CI. Nesta dissertação, e em consonância com a maior parte da literatura existente, sistema embutido ou SoC, trata-se de um componente. Um dispositivo eletrônico que contém um SoC, pode ser um *dispositivo móvel* ou não, dependendo do seu tamanho físico. Uma definição mais detalhada de SoC será apresentada na seção seguinte.

A popularização de SoCs é impulsionada pelo crescente número de sistemas embutidos de alto desempenho lançados no mercado. É possível encontrar SoCs em dispositivos eletrônicos como telefones celulares, consoles de jogos, reprodutores de áudio e vídeo e outros aparelhos portáteis, mostrando que esta é uma tecnologia bem presente no cotidiano.

A motivação inicial para utilização de SoCs foi a necessidade de implementar um sistema computacional de uso específico em dispositivos que não são considerados computadores e

em geral possuem pouco espaço interno. Além da redução de espaço, também foi necessário reduzir o consumo de energia e o número de componentes susceptíveis a falhas. A concepção de projeto, utilizando SoCs, possui grande apelo para executar uma grande gama de aplicações, não somente em dispositivos móveis e de tamanho reduzido mas também em dispositivos de uso específico que requerem boa capacidade computacional (SIEWERT, 2005).

Com o aumento da escala de integração, aumenta a complexidade de projeto de CIs e aumenta a capacidade de implementar sistemas mais complexos através de um SoC. Ao mesmo tempo, o mercado exige que dispositivos sejam lançados cada vez mais rápido. No ano de 2000, acreditava-se que em pouco tempo, o prazo máximo entre as especificações iniciais e a implementação final de um produto seria de apenas 6 meses (KEUTZER *et al.*, 2000). Para atender a esta exigência é necessário uma metodologia de projeto que priorize a reutilização de componentes. No caso da implementação de SoCs, para reduzir o tempo de projeto e facilitar a reutilização, são utilizados blocos de propriedade intelectual.

Com a utilização de blocos de propriedade intelectual ou IPs, surge o problema de comunicação entre os blocos. No caso de projetos com poucos IPs, a comunicação entre eles pode ser feita ponto-a-ponto, mas em um projeto com muitos IPs, a comunicação torna-se um fator crítico de projeto. Como a previsão é que a quantidade de IPs por SoC aumente, é necessária uma arquitetura de comunicação eficiente e escalável. Desta necessidade surgiu a arquitetura de *redes embutidas* ou NoCs, que foi inspirada no modelo de redes de comunicação de computadores.

Este capítulo apresenta na Seção 1.1 o conceito de SoC, o fluxo típico de projeto de um SoC, os componentes que integram um SoC, a metodologia baseada em plataforma, a plataforma de *hardware* e a plataforma de *software*. A Seção 1.2 apresenta o conceito de NoC, a arquitetura interna de uma NoC, a divisão em camadas baseadas no Modelo de Referência OSI e diferentes topologias. A Seção 1.3 introduz a metodologia de síntese para plataforma NoC, o fluxo típico de projeto e o fluxo de projeto proposto que será utilizado nesta dissertação.

1.1 Sistemas Embutidos

Um sistema embutido ou SoC (*System-on-Chip*), é um computador programável de uso específico implementado em um circuito integrado (CI). Normalmente é utilizado em dispositivos eletrônicos de tamanho reduzido e que executam aplicações específicas (WOLF, 2001). Dispositivos de uso cotidiano como telefones celulares e aparelhos de fax, são alguns dos dispositivos que podem possuir um SoC como componente principal.

Um CI é considerado um SoC quando este é constituído por uma unidade central de processamento (ou CPU – *Central Processing Unit*), memória, componentes E/S e uma interconexão entre os três (SIEWERT, 2005), permitindo a execução de um programa. Um SoC pode também incluir alguns componentes cujas funcionalidades são fixas e que permitem acelerar a execução de algumas tarefas críticas para auxiliar a CPU em tempo real. A arquitetura da CPU utilizada é escolhida de acordo com as tarefas que serão executadas. A arquitetura SoC constitui a integração de todos os componentes de um computador em um único CI. Quanto menor for a dependência do SoC de circuitos adicionais como, memórias e controladores de E/S externos, mais "*on chip*"é considerado o sistema. A Figura 1 (MADISETTI; SHEN, 1997) mostra uma arquitetura típica de um SoC.

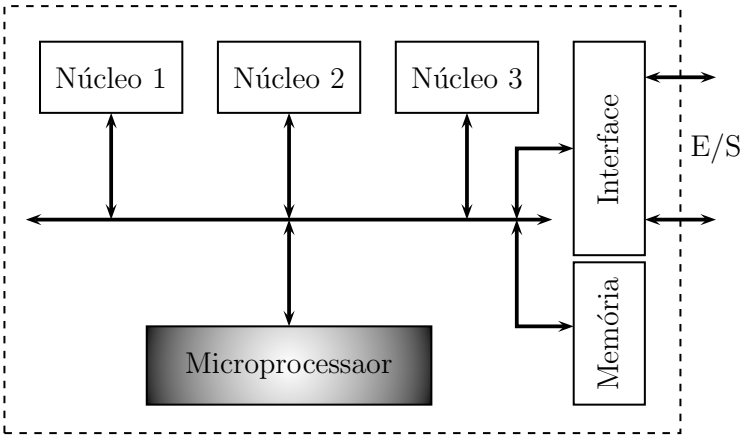


Figura 1: Arquitetura básica de um SoC

Um SoC com mais de uma unidade de processamento é chamado de MPSoC (*Multi-Processor* SoC). Um MPSoC não é apenas um CI que incluí múltiplos processadores, viabilizado pela crescente densidade de transistores das tecnologias de fabricação. Mais do que isso, MPSoCs são arquiteturas personalizadas, que equilibram as restrições da tecnologia e necessidades das aplicações, geralmente de um domínio específico (JERRAYA; TENHUNEN; WOLF, 2005).

Como consequência das restrições tecnológicas, os MPSoCs assumem um caráter altamente heterogêneo e acabam requerendo estruturas de comunicação elaboradas para atender as restrições de projeto. Duas são as estruturas de comunicação mais adotadas atualmente: estrutura de barramento centralizado e redes embutidas ou NoCs (BENINI; MICHELI, 2002).

O barramento centralizado é muito utilizada em arquiteturas de computadores, sendo natural seu uso em MPSoCs. Contudo, apresenta uma grande desvantagem em relação à arquitetura descentralizada, que é a falta de escalabilidade. A arquitetura NoC é baseada estabelece a separação entre computação e comunicação, tornando ambas mais eficientes.

1.1.1 Projeto de sistemas embutidos

O projeto de um sistema embutido é guiado por requisitos e restrições que determinam os elementos computacionais que farão parte da implementação do projeto. Entende-se como requisitos os objetivos que devem ser buscados em todas as etapas do projeto, como por exemplo, a minimização do consumo de energia e a redução da área ocupada. Entende-se como restrições as fronteiras que não podem ser ultrapassadas para que o projeto seja aceito, como por exemplo, o consumo máximo de energia e a máxima área ocupada.

Sistemas embutidos possuem partes de *software* e partes de *hardware*. Um grande número de pesquisadores se esforça para conceber métodos para automatizar as diversas atividades de projeto, permitindo assim uma distribuição mais adequada das funcionalidades do sistema entre componentes de *software* e componentes de *hardware*, de forma a atender aos requisitos e às restrições. Dentre as atividades de projeto de um sistema embutido, estão incluídas: especificação, escalonamento, alocação, síntese, mapeamento e validação (MARCON, 2005).

Um fluxo característico de projeto de sistemas embutidos está ilustrado na Figura 2. Nesta figura os retângulos arredondados de linha cheia representam descrições, a numeração de 1 a 4 representa diferentes estágios de descrições de *software* e de *hardware*, retângulos representam etapas do projetos que podem ser automatizadas com ferramentas computacionais que operam sobre as descrições, as arestas indicam o fluxo de dependências do projeto e os retângulos arredondados de linha tracejada representam os componentes onde serão implementadas as descrições finais. A cada etapa de descrição o projeto é validado para verificar se as especificações iniciais são atendidas.

Uma especificação é o nível mais alto de descrição de um sistema. A especificação pode ser descrita em uma linguagem de programação de propósito geral, como por exemplo, a linguagem C (KERNIGHAN; RITCHIE, 1988), em linguagem de descrição de *hardware*, como por exemplo, VHDL (IEEE, 2009) ou em uma linguagem que descreva *software* e *hardware* com características das linguagens anteriores, como por exemplo, SystemC (SYSTEMC, 2009) e HardwareC (GUPTA; CLAUDIONOR N.C.; MICHELI, 1994). As descrições dos componentes devem ser passíveis de tratamento automatizado ou semi-automatizado por métodos e ferramentas de níveis de abstração inferiores, ou seja, compilação para o *software* e síntese de alto nível e física para o *hardware*.

A etapa de particionamento consiste em dividir o sistema completo gerando subsistemas que podem ser implementados em componentes de *hardware* ou de *software*. Os subsistemas

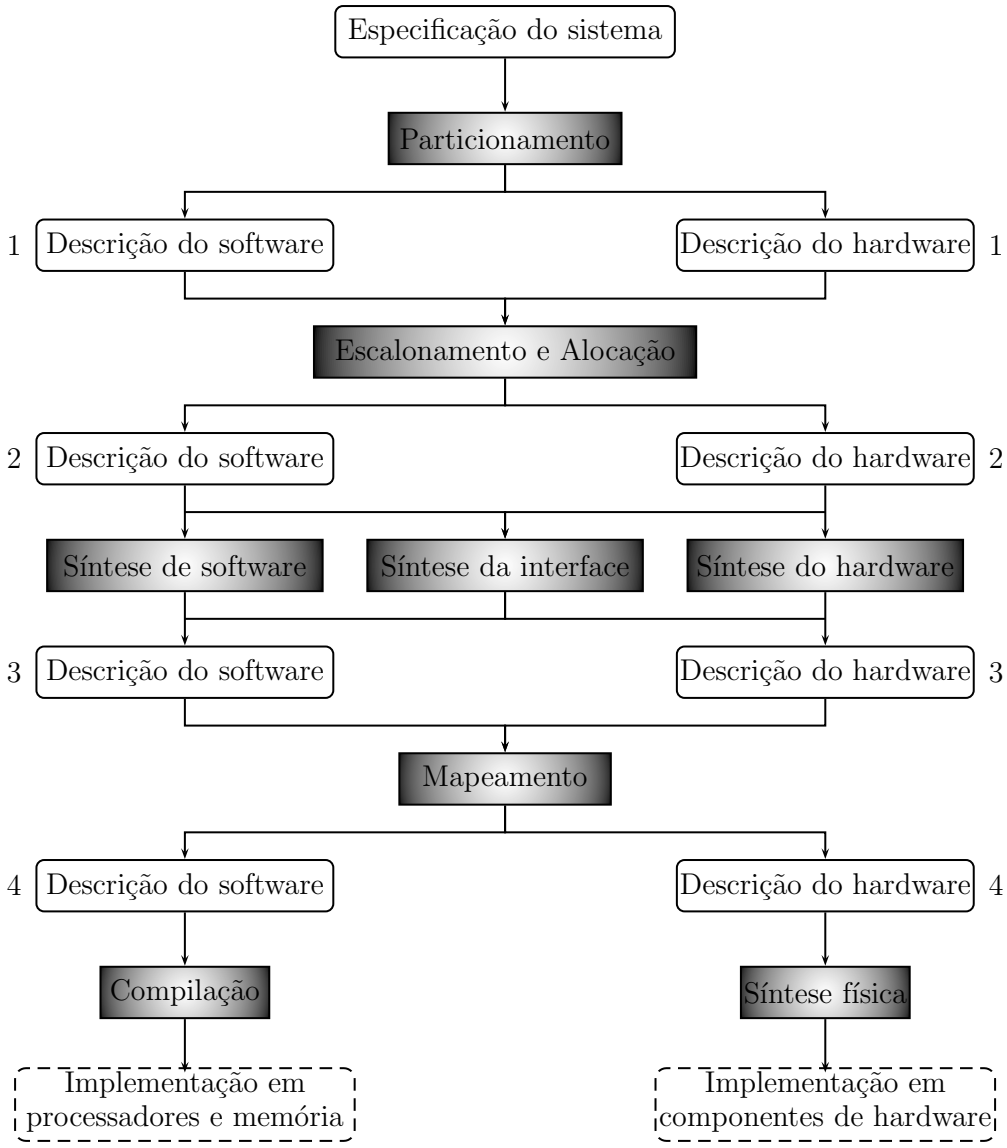


Figura 2: Fluxo típico de projeto de sistemas embutidos

de *hardware* formam a partição de *hardware* e os subsistemas de *software* formam a partição de *software*. O particionamento será visto com mais detalhes adiante.

O escalonamento é a etapa que define a ordem parcial de execução das tarefas da aplicação. A alocação é a atividade que direciona a implementação da aplicação aos recursos computacionais disponíveis, produzindo uma nova ordem de execução que complementa o escalonamento. Normalmente estas etapas são feitas em conjunto. As etapas de particionamento, escalonamento e alocação estão intimamente relacionadas mas devido à alta complexidade das mesmas, muitas ferramentas de apoio a projetos realizam estas etapas separadamente (MARCON, 2005). Nesta tese, quando nos depararmos com o problema de alocação, no Capítulo 4, assumiremos que o particionamento e o escalonamento já foram realizados.

A síntese é a etapa em que ocorre uma queda de nível de abstração das descrições. Uma descrição mais abstrata é traduzida em uma descrição menos abstrata até que seja alcançada a implementação física do sistema (EDWARDS *et al.*, 1997). A síntese de *software*, a partir das descrições dos subsistemas particionados, normalmente implica na transformação de estruturas concorrentes em um código sequencial. A síntese de *hardware*, por sua vez, tem o objetivo de extrair o máximo de paralelismo da descrição do sistema. A comunicação entre a implementação de *hardware* e a implementação de *software* só é possível através de interfaces de comunicação em ambas as implementações. Esta interfaces são geradas na etapa de síntese de interface.

O mapeamento de uma aplicação é uma função que associa subsistemas implementados em *hardware* e em *software* a pontos de acesso de uma arquitetura de comunicação. Através destes pontos de acesso os subsistemas irão se comunicar para executarem a aplicação do projeto. O mapeamento pode ser considerado uma síntese de alto nível que permite a validação do sistema antes da síntese física. No Capítulo 5 o problema do mapeamento será apresentado com mais detalhes e a proposta de solução para este problema será apresentada.

1.1.2 Blocos de propriedade intelectual

Normalmente, SoCs são projetados como circuitos integrados para aplicações específicas ASICs (*Application-Specific Integrated Circuits*). Para que um ASIC seja reconfigurável, são utilizados blocos de propriedade intelectual, comumente chamados de *IP Blocks* ou simplesmente *IPs*, (*Intellectual Properties*). Um *IP*, após ser devidamente testado, representa um componente que pode ser utilizado na construção de um sistema mais complexo para formar um SoC. Um *IP* pode especificar o projeto de uma memória, um processador, controlador de E/S, DSP (*Digital Signal Processor*) ou outros circuitos de uso específico em sistemas digitais. Cada *IP* quando implementado em um SoC passa a ser chamado de *núcleo* ou *core*.

Os blocos de *IP* recebem este nome porque o projetista é dono dos direitos de uso do componente, que é considerado uma propriedade intelectual. O conceito de propriedade intelectual é um conceito jurídico e aplicado em outras áreas além da eletrônica. O projetista pode optar por disponibilizar ou comercializar o código fonte do IP sob diferentes tipos de licenças. No lugar do código fonte, a distribuição pode ser feita através de código binário, não possibilitando modificações (SIEWERT, 2005). Dependendo das mudanças que podem ser feitas no *IP*, existem três formas de disponibilizar estes componentes (GUPTA; ZORIAN, 1997). São elas:

- *Núcleos de software*: São IPs descritos na forma de especificações elaboradas em uma linguagens de descrição de *hardware* (HDL - *Hardware Description Language*). Podem ser escritos de modo comportamental, utilizando todos os recursos de alto nível da linguagem para descrever o comportamento do componente, de modo de fluxo de dados, determinando as saídas de acordo com as entradas e/ou de modo estrutural, descrevendo a estrutura do componente em termos de outros componentes mais básicos como portas lógicas, *flip-flops* e suas interconexões.

- *Núcleos de firmware*: São IPs descrevendo a estrutura do componente, geralmente vêm na forma de uma *netlist* em nível de portas lógicas. Uma *netlist* contem informação sobre as conexões internas entre as portas lógicas que compõem o *IP*.

- *Núcleos de hardware*: São IPs descrevendo o *layout* para a implementação física na camada de silício do CI. Este tipo de núcleos é muito usado em dispositivos reconfiguráveis de tipo FPGAs (*Field Programmable Gate Arrays*) com uma granularidade grossa (PALMA *et al.*, 2002).

Núcleos de *hardware* são IPs que são geralmente de custo baixo pois não possibilitam nenhuma modificação. São o resultado da síntese, que consiste no mapeamento das portas e da *netlist* no *hardware* final que pode ser um FPGAs ou ASICs. Por outro lado, núcleos de *software* são disponibilizados através do respectivo código-fonte, dando ao projetista a possibilidade de incluir alterações e sintetizar para diferentes plataformas, aumentando assim o valor de comercialização. Entre o núcleo de *hardware* e o de *software* está o núcleo de *firmware*, onde o projetista não tem tanta liberdade como no caso de núcleos de *software* mas também não está tão restrito quanto no caso de núcleos de *hardware*. A Tabela 1 resume as diferenças de flexibilidade, custo e descrição dos três tipos de *IPs*.

Tabela 1: Diferenças entre núcleos de *software*, *firmware* e *hardware*

Categoria	**Flexibilidade**	**Custo**	**Descrição**
Núcleos de *software*	Alta	Alto	HDL
Núcleos de *firmware*	Moderada	Médio	Netlist
Núcleos de *hardware*	Nenhuma	Baixo	Layout

Além dos núcleos formados pela CPU, memória e controlador de E/S, um SoC também necessita de uma arquitetura de comunicação e uma interface com o mundo externo. Os *IPs* que formam os núcleos de um SoC podem ser novos ou herdados de projetos já existentes, e

também podem ser obtidos de uma ou mais bibliotecas. Na Figura 3, núcleos provenientes de bibliotecas diferentes podem apresentar problemas de comunicação interna ou de interface, o que dificulta a integração entre os núcleos e com o mundo externo. Neste caso, é necessário modificar o *IP*, o que só é possível inteiramente no caso de núcleos de *software* e parcialmente no caso de núcleos *firmware* e impossível no caso de núcleos de *hardware*.

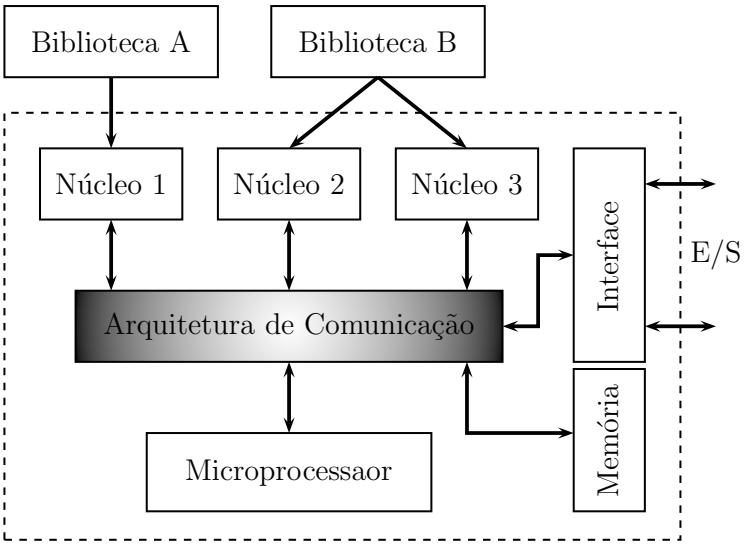


Figura 3: Arquitetura básica de um SoC integrada à diferentes bibliotecas

Uma grande variedade de *IPs* de diferentes tipos e funcionalidades está disponível para projetistas de SoCs. Uma grande vantagem é que estes IPs já testados e consolidados formam bibliotecas a disposição do projetista, que irá combina-los para formar o sistema desejado (GIZOPOULOS; PASCHALIS; ZORIAN, 2004). Uma desvantagem é que as bibliotecas de *IPs* disponíveis não para de crescer e cabe ao projetista escolher a melhor combinação de *IPs* para uma dada aplicação em curto prazo.

É importante ter em mente que o tempo de lançamento de um dispositivo eletrônico pode determinar o seu sucesso ou o fracasso. Um produto terá sucesso se for lançado no momento em que seus consumidores em potencial precisam usá-lo desde que, seu funcionamento atenda às necessidades. Um produto que atende a todas as necessidades esperadas não terá sucesso se não for lançado no momento em que seu público alvo o requer (GIZOPOULOS; PASCHALIS; ZORIAN, 2004).

Para facilitar o projeto de SoCs e reduzir o tempo de implementação, existem ferramentas de auxílio a projeto e bibliotecas com *IPs* baseados em interfaces e em arquiteturas de comunicação. Um projeto baseado em bibliotecas que contemplam componentes de *hardware*,

na forma de *IPs*, e componentes de software responsáveis pela comunicação dos *IPs*, é chamado de *projeto baseado em plataforma* ou PBD (*Platform-Based Design*) (VAHID; GIVARGIS, 2001)(KEUTZER *et al.*, 2000). A próxima seção apresenta esta metodologia aplicada em projetos de SoCs.

1.1.3 Metodologia baseada em plataforma

Uma plataforma é uma arquitetura de referência altamente parametrizável que contem componentes de *hardware* tais como processadores, memórias, controladores de E/S e *software* tais como *drivers*, protocolos de comunicação e aplicações. É desenvolvida para facilitar a implementação de um projeto de SoC (DEMMELER; GIUSTO, 2001). A Figura 4 ilustra a metodologia "*plug and play*"usada pela PBD, onde blocos de propriedade intelectual (IPs) são adicionados em uma plataforma.

Figura 4: Ilustração da metodologia baseada em plataforma

As plataformas variam de acordo com o tipo de dispositivo, como por exemplo: conversores de TV digital, câmeras digitais e telefones celulares (VAHID; GIVARGIS, 2001). Um dos objetivos da metodologia de projeto baseado em plataforma é minimizar o tempo e custo de implementação do projeto através da re-utilização da arquitetura e componentes de *hardware* e *software*. Os componentes da plataforma podem ser incluídos, removidos e até parametrizados de acordo com as necessidades da aplicação em desenvolvimento. A Figura 5 apresenta uma plataforma de *hardware* de um SoC para câmeras fotográficas digitais. Fazendo uma analogia com a Figura 4, o SoC da ilustrado na Figura 5 pode representar uma plataforma padrão para projetos de câmeras fotográficas digitais, onde o projetista escolheria os IPs de acordo com as especificações do projeto.

Um SoC baseado em plataforma é implementado removendo-se componentes desnecessários da arquitetura de referência e adicionando-se componentes necessários que originalmente

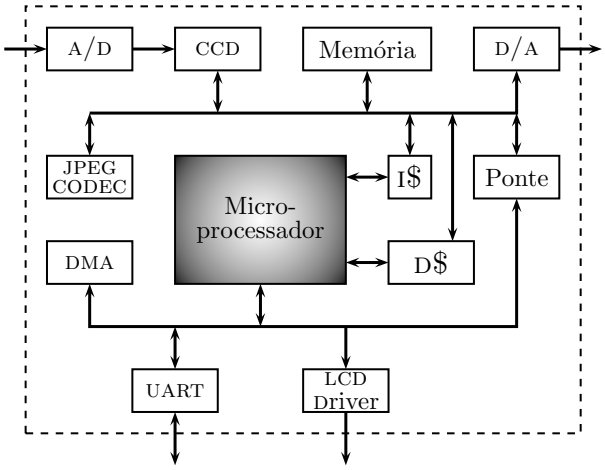


Figura 5: Plataforma SoC para uma câmera digital

não fazem parte da arquitetura. O custo inicial do desenvolvimento da plataforma é alto. No entanto, projetos subsequentes, desenvolvidos com base nessa plataforma, apresentarão um custo menor devida a re-utilização de componentes de *hardware* e de *software*. Além da redução de custo do projeto, o tempo de implementação do mesmo também será reduzido, acelerando a disponibilidade do respectivo produto no mercado.

No caso de uma implementação inicial, onde não há referencia anterior que possibilite o reuso de componentes, o tempo de seleção de componentes ainda é alto. Para a plataforma da Figura 5, alguns dos possíveis parâmetros a serem definidos são: capacidade da *cache* de dados, capacidade da *cache* de instruções, tamanho da linha da *cache* de dados, tamanho da linha da *cache* de instruções, resolução do *CODEC*, tamanho do bloco de dados de *DMA*, tamanho do *buffer* de *UART*, ciclo de trabalho do *LCD* e tamanho da memória. Analisar cada uma das possíveis combinações existentes pode ser extremamente demorado (VAHID; GIVARGIS, 2001).

Além de selecionar os componentes de *hardware* e *software* de um SoC, também é necessário selecionar o que será implementado em *hardware* e o que será programado em *software*. A partir das especificações de projeto é possível modelar o comportamento do sistema em execução. As tarefas executadas pelo SoC podem ser representadas por um grafo de tarefas em que cada nó do grafo pode ser um componente de *hardware* ou *software*. A divisão de tarefas que serão implementadas em *hardware* e tarefas que serão implementadas em *software* é geralmente identificada através de *particionamento*. Características e técnicas usadas para realizar esta etapa durante o projeto de um SoC são discutidas a seguir.

1.1.4 Particionamento

Um SoC é definido pelos seus diferentes *núcleos* ou *cores* e estes interagem para a realização de tarefas específicas. Para obter o funcionamento correto, o projetista do SoC precisa definir o conjunto de componentes que será utilizado na implementação e distribuir as tarefas que devem ser executadas pelo SoC entre esses componentes. O conjunto de componentes selecionados para integrar o SoC assim com a operação de seleção são chamados de *alocação* e o resultado da distribuição de tarefas é chamada de *partição* mas a operação de distribuição é chamada de *particionamento*. A alocação e o particionamento devem ser efetuados de modo que a implementação satisfaça restrições do projeto, tais como, custo, desempenho, tamanho e consumo de energia (GAJSKI *et al.*, 1994). Existem duas técnicas bem distintas de particionamento, denominadas de particionamento *estrutural* e particionamento *funcional*.

1.1.4.1 Particionamento estrutural

No particionamento estrutural, inicialmente é definida a estrutura de *hardware* do projeto, compreendendo componentes de *hardware* e suas conexões. Os componentes de *hardware* podem ser simples portas lógicas ou unidades complexas de cálculo e microprocessadores. Em seguida, a estrutura é particionada. O particionamento agrupa os componentes em grupos onde cada um representa um componente *macro* do sistema. O particionamento estrutural possui a vantagem de estimar diretamente o tamanho da implementação final do CI assim como a quantidade de pinos. O tamanho é estimado pela soma dos tamanhos dos componentes de cada grupo e o número de pinos é estimado pelo número de conexões entre um grupo e outro (GAJSKI *et al.*, 1994). As três maiores desvantagens do particionamento estrutural são:

- **Difícil balanço entre tamanho e desempenho** - Como a definição da estrutura é feita antes do particionamento, esta segunda etapa pode acabar anulando a primeira. No caso do projetista optar por uma estrutura de *hardware* mínima, na etapa de particionamento tarefas iguais e sequenciais podem ser atribuídas a grupos diferentes, gerando uma perda de desempenho. No caso contrário, onde o projetista opta por utilizar mais *hardware* e na etapa de particionamento o máximo de tarefas iguais e sequenciais é atribuída ao mesmo grupo, isto gera um desperdício de área.

- **Quantidade de componentes** - Os algoritmos utilizados para particionamento não estão evoluindo na mesma velocidade que os sistemas estão crescendo e a quantidade de componentes em um único CI aumenta. Sistemas muito grandes tendem a um particionamento ruim.

- **Soluções apenas de *hardware*** - O particionamento estrutural é limitado apenas à parte de *hardware* do projeto e não ao funcionamento do sistema por completo. Partes do sistema que podem ser implementadas em software, assim como o software que governa o funcionamento do sistema, são ignorados.

À medida que os projetos de SoCs se tornam mais complexos e os CIs acomodam cada vez mais transistores, as desvantagens do particionamento estrutural tornam-se mais evidentes (GAJSKI *et al.*, 1994).

1.1.4.2 Particionamento funcional

No particionamento funcional, inicialmente o sistema é dividido em funções indivisíveis, chamadas de *objetos funcionais*. Em seguida é feito o particionamento que consiste em implementar cada objeto funcional através de *hardware* ou *software*. As três maiores vantagens do particionamento funcional em relação ao particionamento estrutural são:

- **Balanço entre tamanho e desempenho** - Na etapa de implementação estrutural, subsequente ao particionamento funcional, a utilização dos componentes pode ser otimizada ao máximo, maximizando o desempenho e minimizando a área ocupada.

- **Quantidade de componentes** - Menor quantidade de componentes a ser particionada, já que a quantidade de objetos funcionais é menor do que a quantidade objetos em nível de *hardware*. Com menos objetos, o desempenho dos algoritmos de particionamento é melhor e o projetista pode tomar decisões com mais facilidade.

- **Solução *hardware/software*** - A maior vantagem do particionamento funcional é que ele permite um particionamento de *hardware* e *software*. Isso é possível porque os objetos particionados são funcionais e não estruturais. Objetos funcionais podem ser implementados em um processador na forma de um conjunto de instruções ou como componentes de *hardware*. Como a maioria dos sistemas baseados em SoCs possuem partes de *hardware* e *software*, a capacidade de particionar entre estas duas plataformas é indispensável para qualquer sistema de particionamento.

Como visto anteriormente, o projeto baseado em plataforma é composto de bibliotecas de *hardware*, na forma de *IPs*, e de *softwares*. Estas duas partes da plataforma, respectivamente, formam a *plataforma de hardware* e a *plataforma de software*. Estas duas plataformas podem ser implementadas separadamente nas ferramentas de projeto mais modernas, como EDK (*Embedded Development Kit*) da Xilinx.

1.1.5 Plataforma de *hardware*

Reutilização é a palavra chave para reduzir custos nos projetos de SoCs e dispositivos móveis em geral. Acredita-se que estes dispositivos representarão a maioria dos dispositivos eletrônicos em breve. Como os projetistas de SoCs implementam cada vez mais funções em software, é necessária uma arquitetura mínima de *hardware* que possa ser facilmente alterada e reutilizada por diferentes aplicações. Esta arquitetura mínima de *hardware* é chamada de *plataforma de hardware* (KEUTZER *et al.*, 2000).

As restrições impostas pela aplicação, em termos de desempenho influenciam na definição da plataforma de *hardware*. Para implementar um conjunto de funcionalidades, é necessário um microprocessador com um conjunto mínimo de microinstruções e uma memória com uma quantidade mínima de *bytes* (TANENBAUM, 2005). Como cada projeto é caracterizado por um conjunto diferente de funcionalidades, as restrições de cada projeto definem os IPs, que em conjunto com o microprocessador, serão responsáveis por tarefas específicas e garantirão o desempenho desejado. Além das restrições impostas pela aplicação também existem as próprias restrições de *hardware* em termos de custo, consumo de energia, área e outros. A intercessão entre as restrições do domínios da aplicação e as restrições do domínio de *hardware* definem a plataforma de *hardware* que será utilizada no produto final (KEUTZER *et al.*, 2000).

1.1.6 Plataforma de *software*

A plataforma de *hardware* por si só não é suficiente para atingir o nível de reutilização de software desejado. Para ser reaproveitável, a plataforma de *hardware* precisa ser abstrata para o *software* em execução. O *software* precisa "enxergar"uma interface de alto-nível acima da camada de *hardware*. Esta interface de alto nível é chamada de API (*Application Program Interface*). A API constitui a plataforma de *software* que encapsula a plataforma de *hardware* possibilitando a abstração desejada dos componentes de *hardware*.

A camada de *software* é constituída por *drivers* que controlam os dispositivos de E/S, por um protocolo de comunicação que controla a comunicação entre os componentes do SoC e um sistema operacional de tempo real, comumente chamado de RTOS (*Real Time Operating System*). Algumas literaturas denominam a plataforma de *software* como RTOS, incluindo os *drivers* de E/S e o protocolo de comunicação. A Figura 6 mostra a estrutura da camada de *software* e as demais camadas de um projeto baseado em plataforma (KEUTZER *et al.*, 2000).

Um nível acima da plataforma de *software* está o *software* programado para executar as funcionalidades do projeto, representando o nível mais alto de abstração. Atualizações do

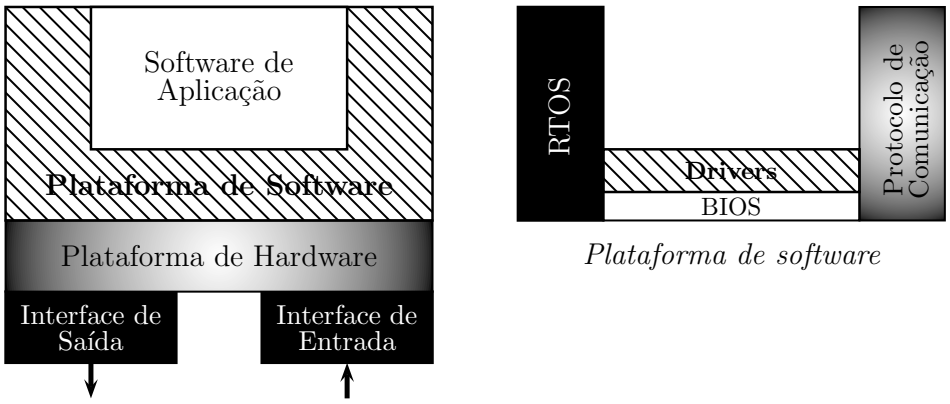


Figura 6: Camadas de um projeto baseado em plataforma

projeto podem ser feitas diretamente na camada mais alta, reduzindo consideravelmente o custo e tempo de lançamento de um novo produto no mercado. No entanto, ainda existe o entrave da compatibilidade. Os programas de alto nível devem ser desenvolvidos para uma plataforma de *software* ou um RTOS específico. Para maximizar a capacidade de reutilização de projetos é necessária a padronização da plataforma de *software*. Atualmente cada fabricante desenvolve o seu próprio RTOS ou então adota um sistema existente no mercado.

1.2 Redes Embutidas

No ano de 2002 pesquisadores do *ITRS* (*International Technology Roadmap for Semiconductors*) fizeram a previsão de que até o final da década, SoCs seriam fabricados com 4 bilhões de transistores na ordem de 50nm cada, operando abaixo de 1V e operando com uma frequência de 10Ghz. Talvez não cheguemos a tantos transistores em um CI e nem a frequência de 10Ghz até o final desta década, mas os números atuais já representam um desafio no projeto de SoCs. A grande quantidade de conexões dos componentes internos representa um fator limitador de desempenho e gera alto consumo de energia (BENINI; MICHELI, 2002).

Além da grande quantidade de conexões internas, existe o problema de sincronização, que será muito difícil, senão impossível, de ser resolvido à medida que a quantidade de componentes e a frequência de operação aumentam. Uma possível solução para o problema de sincronização consiste em utilizar diferentes sinais de *clocks*, gerando um sistema globalmente assíncrono e localmente síncrono. Na ausência de uma única frequência de operação de referência, um SoC se transforma em um sistema distribuído embutido em um único CI. O controle global do sistema se torna mais difícil porque é preciso monitorar cada um dos subsistemas formados. O controle de comunicação entre os múltiplos domínios de frequência de operação pode ser muito custoso e até mesmo inviável (BENINI; MICHELI, 2002).

A baixa tensão de alimentação, a grande quantidade de barramentos em diferentes frequências de operação e o reduzido tamanho dos CIs, pode tornar os dispositivos mais susceptíveis a interferência eletromagnética, *crosstalk* e a injeção de cargas induzidas (BENINI; MICHELI, 2002), ocasionando erros na transmissão de dados. A transmissão de dados digitais em tais condições seria não determinística e talvez impraticável.

O projeto de SoCs depende fortemente de uma plataforma modular, tanto para *software* quanto para *hardware*. O uso de métricas probabilísticas, como desempenho e consumo de energia, para a avaliação de projeto, leva a uma mudança de metodologia no projeto de SoCs. Com base no fato que a comunicação será o maior limitador no projeto de SoCs, uma nova metodologia de comunicação inspirada nas infra-estruturas de redes de computadores foi desenvolvida para os SoCs. Esta nova metodologia aproveita o conhecimento de engenharia consolidado no projeto de grandes redes de computadores e utiliza-o no projeto de SoCs, dando origem às redes embutidas, ou simplesmente NoCs (*Networks-on-Chip*).

A arquitetura NoC, além de eliminar os problemas de interferência citados anteriormente, pode ser empregada em SoCs com múltiplos processadores, os MPSoCs (PANDE *et al.*, 2005). A tendencia natural é que os projetos de SoCs utilizem múltiplos processadores, criando um ambiente de computação paralela dentro de um CI. Para viabilizar a comunicação entre os processadores e os demais dispositivos do CI, é desejável que a comunicação interna se assemelhe a comunicação de computadores por meio de uma rede de dados. Projetos baseados em arquitetura NoC atendem aos requisitos de comunicação impostos atualmente nos projetos de SoCs e MPSoCs.

Embora a arquitetura NoC esteja baseada na arquitetura de comunicação de computadores, um conjunto significativo de restrições separa estas duas arquiteturas. Em relação ao desempenho, o alto *throughput* e a baixa latência são características desejadas em ambos os domínios. Contudo, na perspectiva de NoCs, a energia dissipada para realizar a comunicação pode ser significativa comparada com a energia total do sistema. A área utilizada pelos componentes de comunicação é então uma outra característica importante que deve ser observada durante o projeto. Esta metodologia permite um alto nível de abstração do modelo de comunicação (PANDE *et al.*, 2005).

1.2.1 Arquitetura interna

Uma importante característica na metodologia baseada em NoC consiste na separação entre componentes de comunicação e aqueles de processamento. A comunicação entre os componen-

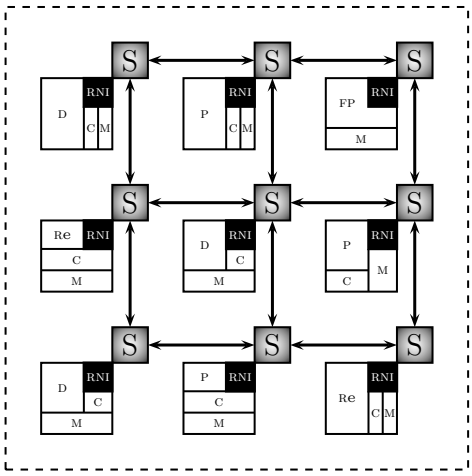


Figura 7: NoC de tipo malha com nove recursos

tes de processamento é responsabilidade da plataforma de rede enquanto que os componentes da plataforma de *hardware* ou de processamento ficam dedicados apenas para a execução das funcionalidades da aplicação propriamente ditas. Neste contexto, os elementos da plataforma de *hardware* são geralmente chamados de *recursos*. Cada recurso é um meio físico onde IPs podem ser implementados, gerando assim um *core* (ou núcleo) da rede. Os componentes da plataforma de rede são os *canais*, os *switches* e as *interfaces de rede*. Os canais são barramentos por onde os dados trafegam e permitem a interligação dos *switches*. Os *switches* são responsáveis pela lógica de roteamento e chaveamento da rede. As interfaces de rede ou RNIs (*Resource Network Interface*) implementam as ligações entre *switches* e recursos. Cada recurso deve possuir sua própria interface de rede para que possa se comunicar com o restante dos recursos da plataforma de *hardware*. A Figura 7 apresenta uma estrutura de NoC em malha (KUMAR *et al.*, 2002) denominada de CLICHÉ (*Chip-Level Integration of Communicating Heterogeneous Elements*), onde diferentes blocos IPs estão implementados nos recursos e as siglas utilizadas significam: C – memória *cache*, D – bloco DSP, FP – unidade de ponto flutuante, M – memória, P – processador, Re – bloco reconfigurável e RNI – interface de rede.

No caso de uma implementação em um CI, qualquer IP pode ser implementado em um recurso desde que o recurso disponha de área de *hardware* suficiente. No caso de implementações em *hardware* reconfigurável, como uma FPGA, a implementação do IP dependerá dos recursos básicos disponíveis na FPGA, como *flip-flops* e *lookup tables*. Qualquer recurso dotado de uma interface de rede, que esteja ligada a um *switch*, pode estabelecer comunicação com qualquer outro recurso da NoC que apresente as mesmas condições. A interface de rede identifica cada recurso com um endereço físico. Qualquer recurso conectado a rede é visto como um sistema embutido independente. A rede inteira pode ser considerada como um sistema distribuído,

onde os componentes da plataforma de rede disponibilizam serviços de comunicação (SOININEN; HEUSALA, 2003).

A arquitetura do *switch* depende do esquema de roteamento utilizado na NoC. As duas principais categorias de roteamento são: *determinística* e *adaptativa*. Algoritmos de roteamento determinísticos sempre traçam o mesmo caminho entre um recurso origem e um recurso destino. Algoritmos adaptativos utilizam a informação do trafego nos canais da rede para evitar congestionamento ou falha no envio de mensagens. Em uma NoC com roteamento determinístico, os *switches* são mais rápidos e mais compactos. No caso de roteamento adaptativo, o *switch* precisa de mais área de *hardware* para implementar o processamento necessário que permite avaliar as condições da rede (PANDE *et al.*, 2005). Outro fator de impacto no tamanho e desempenho dos *switches* é o tamanho dos *buffers* de entrada e saída. A Figura 8 mostra a arquitetura de um *switch* constituído basicamente de *buffers* de E/S, multiplexadores e de um bloco de lógica de roteamento. Este *switch* possui 5 pares de canais de E/S, o que possibilita o envio e recebimento de dados dos quatro *switches* vizinhos (Norte, Sul, Leste, Oeste) e do recurso local da plataforma ao qual o *switch* é conectado, simultaneamente, garantindo assim um alto *throughput* durante a comunicação.

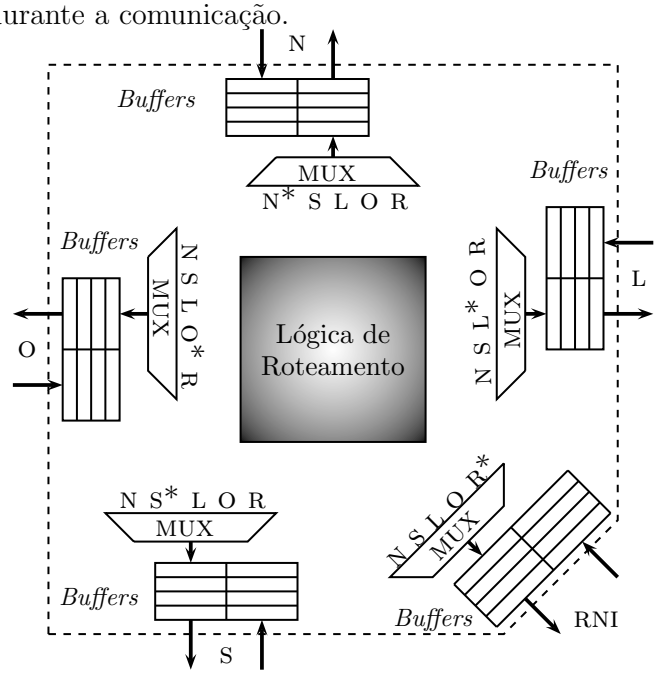


Figura 8: Arquitetura interna de um *switch*

A próxima seção apresenta o Modelo de Referência OSI adotado para identificar o papel exercido por cada elemento de rede em uma rede de computadores e em uma NoC.

1.2.2 Modelo de referência OSI

O Modelo de Referência OSI (*Open System Interconnection*) (DAY; ZIMMERMANN, 1983) foi criado para definir as etapas pelas quais um dado passa, em uma rede de comunicação de dados, desde a fonte até o destino. O modelo é organizado em sete camadas. Cada camada representa um elemento físico ou lógico da rede e possui funções próprias. A maneira como as funções de cada camada são executadas dependem dos protocolos da mesma. Como o nome diz, o Modelo de Referência OSI não se trata de um padrão, mas sim de uma referência que pode ser seguida ou não. Por ser um modelo largamente difundido e adotado por fabricantes e desenvolvedores de soluções para redes de comunicações, a sua adoção em NoCs contribui para a padronização de componentes, aplicações e também facilita o entendimento da arquitetura.

O modelo OSI também é chamado de *pilha* por dois motivos: primeiro porque as camadas estão empilhadas, de modo que as camadas superiores abstraem a implementação das camadas inferiores e segundo, porque o dado que sai de um recurso fonte da rede, localizado logicamente no topo de uma pilha de camadas, deve atravessar todas as camadas dessa pilha, se dirigir até a base da pilha de camadas no destino e subir até o recurso, também localizado logicamente no topo desta. Neste caminho, cada camada manipula o dado de uma maneira diferente. As camadas definidas no modelo OSI são (da base da pilha para o topo) (DAY; ZIMMERMANN, 1983):

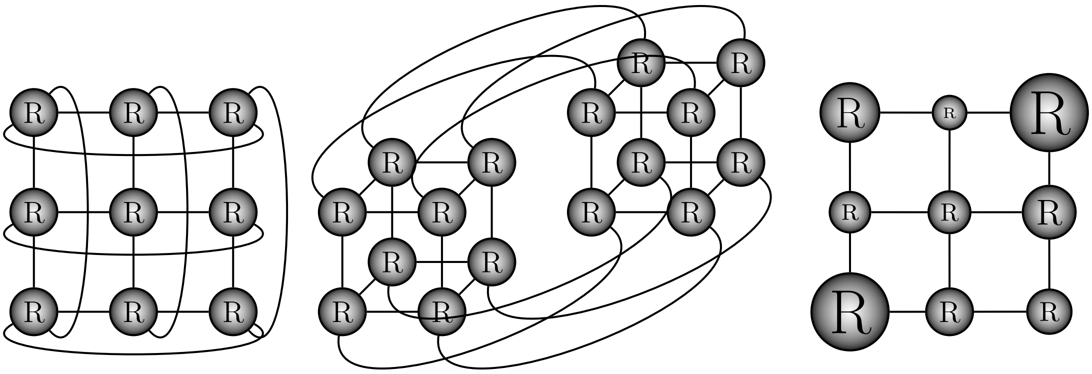
- **Camada física**: Abrange o meio físico, fios, conexões e sinais elétricos. A unidade de dados é o *bit*.
- **Camada de enlace**: Permite a transmissão de dados no meio físico e correção de erros. A unidade de dados é o *frame*.
- **Camada de rede**: Implementa o roteamento entre redes. A unidade de dados é o *pacote*.
- **Camada de transporte**: Estabelece a conexão fim-a-fim e é responsável pelo controle de fluxo de dados. A unidade de dados é o *segmento*.
- **Camada de sessão**: Estabelece, gerencia e encerra as conexões. A unidade de dados é o PDU (*Protocol Data Unit*).
- **Camada de apresentação**: Permite a criptografia ou decifragem das mensagens enviadas ou recebidas, respectivamente. A unidade de dados também é o PDU.
- **Camada de aplicação**: Interage com as aplicações do usuário. A unidade de dados também é o PDU.

Em plataformas NoCs, é possível utilizar quatro das setes camadas do modelo OSI para definir as etapas pelas quais os dados passam e para alocar os elementos físicos e lógicos da rede. Essas camadas e suas finalidades são descritas as seguir (da base da pilha para o topo) (KUMAR *et al.*, 2002) (SGROI *et al.*, 2001):

- **Camada física**: Abrange as características dos canais da NoC e toda a parte física e elétrica.
- **Camada de Enlace**: Implementa as comunicações recurso-*switch* e *switch-switch* a nível físico. A camada física e a camada de enlace são dependentes do meio físico.
- **Camada de rede** - Comunicação *switch-switch* a nível lógico. Fornece um endereço lógico para os *switches* abstraindo os aspectos físicos que são dependentes da tecnologia de fabricação.
- **Camada de transporte**: Permite abstrair completamente as especificidades da plataforma de *hardware* e é responsável pelo controle de fluxo de dados. É da sua responsabilidade também a decomposição da mensagens a ser enviada em pacotes e da recomposição da mensagem original a partir da sequência de pacotes recebidos. A interface de rede (RNI) implementa a camada de transporte.

Além destas quatro camadas, é possível implementar mais camadas acima da camada de transporte para aumentar o nível de abstração e facilitar a programação de aplicações mais complexas (KUMAR *et al.*, 2002). Fugindo um pouco do modelo OSI, Benini (BENINI; MICHELI, 2002) adicionou duas novas camadas acima da camada de transporte. A combinação destas duas é chamada de camada de *software*. Estas duas camadas são (de baixo para cima):

- **Camada de software de sistema**: Possibilita a comunicação de sistemas independentes no caso de cada elemento da NoC possuir um ambiente de comunicação próprio. No caso de uma NoC com múltiplos processadores e cada processador possuir comunicação própria com dispositivos locais, esta camada viabiliza a comunicação entre esses processadores, separando a comunicação interna da comunicação externa.
- **Camada de software de aplicação**: permite abstrair os detalhes de comunicação das camadas inferiores e dá suporte em linguagens de programação de alto nível. Além disso, possibilita maior portabilidade das aplicações do usuário, que passam a depender apenas da camada de *software* de sistema.



(a) Toróide (b) Hipercubo 4D (c) Malha irregular

Figura 9: Exemplos de topologias de redes diretas

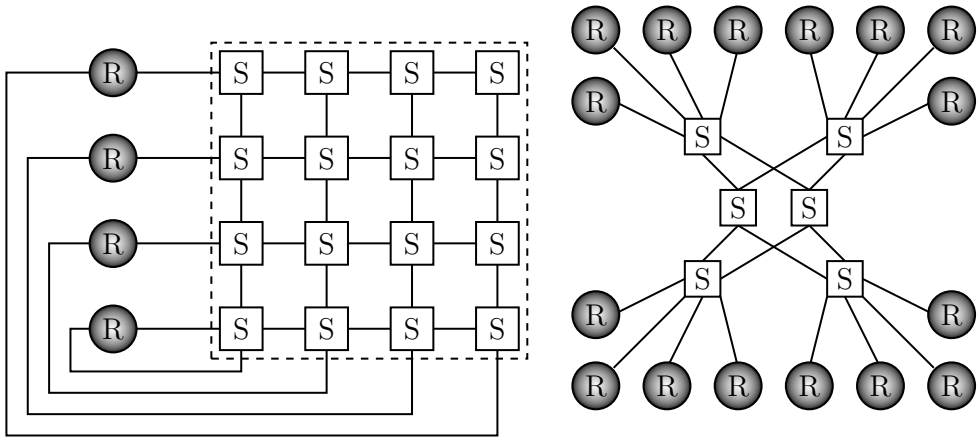
1.2.3 Topologias

A maneira como os *switches* de uma rede são interligados caracteriza a *topologia* da rede. A topologia pode ser representada por um grafo $G(S, C)$, onde S é o conjunto de *switches* e C é o conjunto de canais da rede. As duas principais classes de topologias utilizadas em NoCs permitem classificar as redes como *diretas* ou *indiretas* (ZEFERINO, 2003). A arquitetura dos elementos da rede, i.e. recursos e *switches*, em relação a tamanho e simetria, caracteriza a rede como *regular* ou *irregular*. Redes diretas e indiretas podem ser regulares ou irregulares.

Na topologia de rede direta, cada *switch* está associado a um recurso e este par recebe o nome de *nó*. Cada nó possui ligações ponto-a-ponto para um determinado número de nós vizinhos. A troca de mensagens entre nós não vizinhos deve passar por nós intermediários. Se uma mensagem recebida por um nó é destinada a outro nó dentro da rede, cabe à lógica de roteamento do *switch* do nó receptor decidir encaminhar a mensagem para um dos nós vizinhos mais próximos, para que a mesma avance em direção ao seu destino. Apenas os *switches* são envolvidos nesta comunicação, sem que haja intervenção dos recursos locais.

Uma rede direta ideal deve ser completamente conectada (ZEFERINO, 2003). Para que uma NoC seja diretamente conectada é necessário que cada nó possua no mínimo $N-1$ canais, onde N é o número de nós da rede. Isto torna o projeto proibitivo para uma rede com muitos nós além de apresentar pouca escalabilidade. Para contornar as dificuldades de implementação de uma rede direta ideal, na prática são utilizadas topologias diretas não ideais. Nestas topologias cada nó se comunica apenas com seus vizinhos mais próximos em um espaço n-dimensional. A Figura 9(a) e a Figura 9(b) apresentam duas topologias de redes diretas regulares e a Figura 9(c) apresenta uma topologia de rede direta irregular.

Nas redes indiretas, nem todos os *switches* estão associados a um único recurso. Neste



(a) Crossbar (b) Borboleta

Figura 10: Exemplos de topologias de redes indiretas

tipo de redes, os recursos estão ligados a uma rede de *switches*. Por sua vez, os *switches* estão ligados entre si e alguns estão ligados diretamente aos recursos formando nós. Duas topologias de redes indiretas se destacam: redes *crossbar* e redes multiestágio. Para conexões indiretas de N recursos, o *crossbar* é a topologia ideal em relação à disponibilidade de caminhos. Esta topologia utiliza um único elemento de roteamento que funciona como se fosse N^2 *switches* responsáveis pelo roteamento das mensagens dos N recursos da rede. A complexidade da rede é definida pelo elemento de roteamento, sendo essa igual a N^2. Isto torna o custo da rede *crossbar* proibitivo para interligar uma grande quantidade de recursos.

Redes indiretas com *switches* organizados em estágios são chamadas de redes multiestágio. Para que uma mensagem vá de um recurso a outro é preciso que ela atravesse os estágios de *switches* entre os recursos. Estas redes são caracterizadas pelo número de estágios incluídos e pela forma como estes são organizados. Assim como as redes diretas, as redes indiretas podem ser regulares e irregulares. Em redes regulares, os *switches* são idênticos, formando assim o que são chamados de *estágios regulares* (ZEFERINO, 2003). Em redes irregulares, os *switches* e recursos podem ser *ad hoc*. A Figura 10(a) mostra um rede *crossbar* com 4 recursos e a Figura 10(b) mostra uma rede multiestágio do tipo *borboleta*.

1.3 Metodologia de Síntese para Plataforma NoC

Tão importante quanto um bom projeto é uma boa metodologia de implementação do projeto e ferramentas de suporte, que traduzam as especificações do projeto em seu produto final, neste caso, uma plataforma NoC.

Como primeiro passo, deve-se identificar o escopo do projeto do sistema, suas funcionalidades principais junto com as restrições impostas. Esta análise deve gerar uma especificação do sistema, assim como uma lista de tarefas básicas provenientes da decomposição do sistema. É preciso uma avaliação sistemática que para que não ocorram ambiguidades na decomposição de tarefas. Em seguida, as tarefas identificadas devem ser organizadas em uma estrutura que contemple informações das tarefas e suas dependências. Esta estrutura pode ser um grafo de tarefas orientado e acíclico, que será definido na Seção 4.1.2 no Capítulo 4. Neste grafo serão identificadas as tarefas que devem ser executadas sequencialmente e aquelas que podem ser executadas paralelamente. A plataforma NoC favorece o paralelismo e explorar esta característica na fase inicial do projeto é fundamental. Mesmo que as tarefas ainda não estejam relacionadas a nenhum recurso ou IP específico e nem mapeadas na estrutura física da plataforma, este passo basicamente determina a granularidade e as unidades atômicas que eventualmente serão implementadas no final. Além disso, neste passo o modelo de comunicação básico entre as tarefas também é definido.

Mesmo o grafo de tarefas sendo uma representação de alto nível do projeto, este é importante para gerar o modelo preliminar para a plataforma e minimizar alguns problemas antes que um modelo menos abstrato seja requerido. Com o grafo de tarefas e as restrições identificadas é possível realizar a alocação, que consiste em atribuir IPs para executar as tarefas atômicas já identificadas. Esta atribuição pode ser feita usando uma biblioteca específica de IPs.

Após identificar os IPs, a partir da alocação, é preciso realizar o mapeamento, que consiste em mapear os IPs, nos recursos da plataforma, da melhor maneira possível. Estas etapas são condicionadas a muitas restrições, sendo que muitos fatores influenciam a otimização da plataforma, mesmo em uma abordagem de alto nível. Por exemplo, uma tarefa pode ser melhor executada por um processador do que por outro, dependendo das operações básicas necessárias e dos requerimentos de desempenho. Ao mesmo tempo é preciso levar em conta os requisitos de comunicação descritos no grafo de tarefas. Tarefas que trocam mensagens frequentemente devem ficar próximas fisicamente para reduzir o tempo de comunicação.

Após associar cada tarefa a um IP e mapear os IPs nos recursos, é hora de passar para

uma modelagem mais precisa do sistema. No caso de implementações em FPGAs, utiliza-se uma linguagem de descrição de *hardware*. Em um projeto baseado em plataforma, são utilizados os IPs disponíveis na biblioteca da plataforma. Também é necessário implementar a comunicação entre os elementos da rede, os *drivers* de comunicação dos dispositivos de E/S e a interface de software. Os componentes de software responsáveis pela comunicação, assim como os *drivers* e a interface de alto nível, normalmente fazem parte das bibliotecas das ferramentas mais atuais.

Os passos descritos anteriormente são independentes mas todos influenciam enormemente na obtenção de uma solução ótima. Na prática, são necessárias iterações entre estas etapas, isto é, dependendo do resultado obtido após algum passo, talvez seja necessário voltar aos passos anteriores e repetir o processo necessário. Mais importante do que executar todos estes passos é realizar uma avaliação contínua da implementação. A avaliação deve ser efetuada levando em conta os objetivos desejados, como, por exemplo, desempenho, consumo de energia, custo e quantidade de recursos utilizados. Para realizar estas avaliações são necessárias métricas precisas e fiéis capazes de avaliar o projeto desde a modelagem de alto nível até a implementação em *hardware*. É importante enfatizar a importância da avaliação em alto nível pois esta é mais rápida, mais barata e reduz significativamente o espaço de decisão para a implementação final.

A primeira fase da metodologia de síntese de uma plataforma NoC, que consiste na especificação funcional e na elaboração do grafo de tarefas, é de alto nível e independente da implementação física. Esta fase se torna cada vez mais importante à medida que a complexidade dos sistemas aumenta. As etapas finais, de modelagem mais precisa e dependentes da implementação física, são beneficiadas pelas ferramentas de projeto baseado em plataforma, que dispõem de bibliotecas que facilitam na síntese do circuito. O fluxo típico de projeto baseado em plataforma NoC, como descrito nesta seção está ilustrado na Figura 11.

Normalmente o projeto baseado em plataforma NoC é executado com base em diferentes requisitos como, consumo de energia, área de *hardware*, desempenho e custo. É razoável supor que muitas soluções atenderão a alguns requisitos e não atenderão a outros. Se todos os requisitos forem igualmente importantes, não é possível afirmar que uma solução que atende bem a um requisito A e mal a um requisito B, seja melhor do que uma solução que atenda bem ao requisito B e mal o requisito A, por exemplo. Nestes casos, trabalhar com um conjunto de soluções ao longo do projeto, pode ser mais eficiente do que trabalhar apenas uma solução. Para lidar com este tipo de problema, onde diferentes requisitos devem ser alcançados, propomos

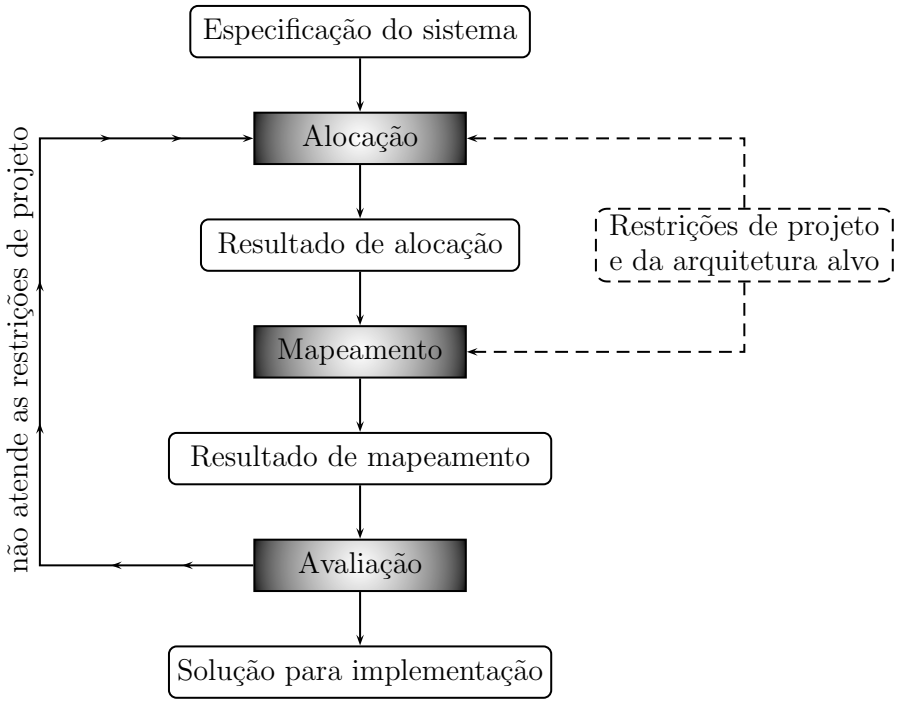


Figura 11: Fluxo típico de projeto de sistemas embutidos para plataforma NoC

inicialmente, que as soluções igualmente ótimas sejam aproveitadas durante a maior parte do projeto. Diferentes alocações serão aproveitadas para a etapa de mapeamento, que por sua vez, irá gerar diferentes mapeamentos. O fluxo de projeto proposto está ilustrado na Figura 12.

1.4 Considerações Finais do Capítulo

Um sistema embutido ou SoC é um computador de uso específico fabricado em um CI e utilizado em dispositivos que não são necessariamente considerados computadores. A maioria, mas não todos, possuem espaço interno limitado e restrições quanto ao consumo de energia. Estes sistemas são formados por blocos de propriedade intelectual ou IPs. Os IPs são componentes projetados em linguagem de descrição de *hardware*. Em projetos baseados em plataforma, os IPs podem ser adicionados e removidos da plataforma de acordo com as necessidades do projeto. Esta metodologia agiliza o projeto e possibilita a reutilização de IPs. Com o crescente capacidade de acomodar IPs em um SoC, surgiu o problema de comunicação entre IPs. Para contornar este problema foi introduzida a arquitetura de redes embutidas, que baseada em redes de computadores, utiliza uma arquitetura de comunicação independente. A arquitetura de comunicação é responsável pela comunicação da rede e possibilita maior escalabilidade. O

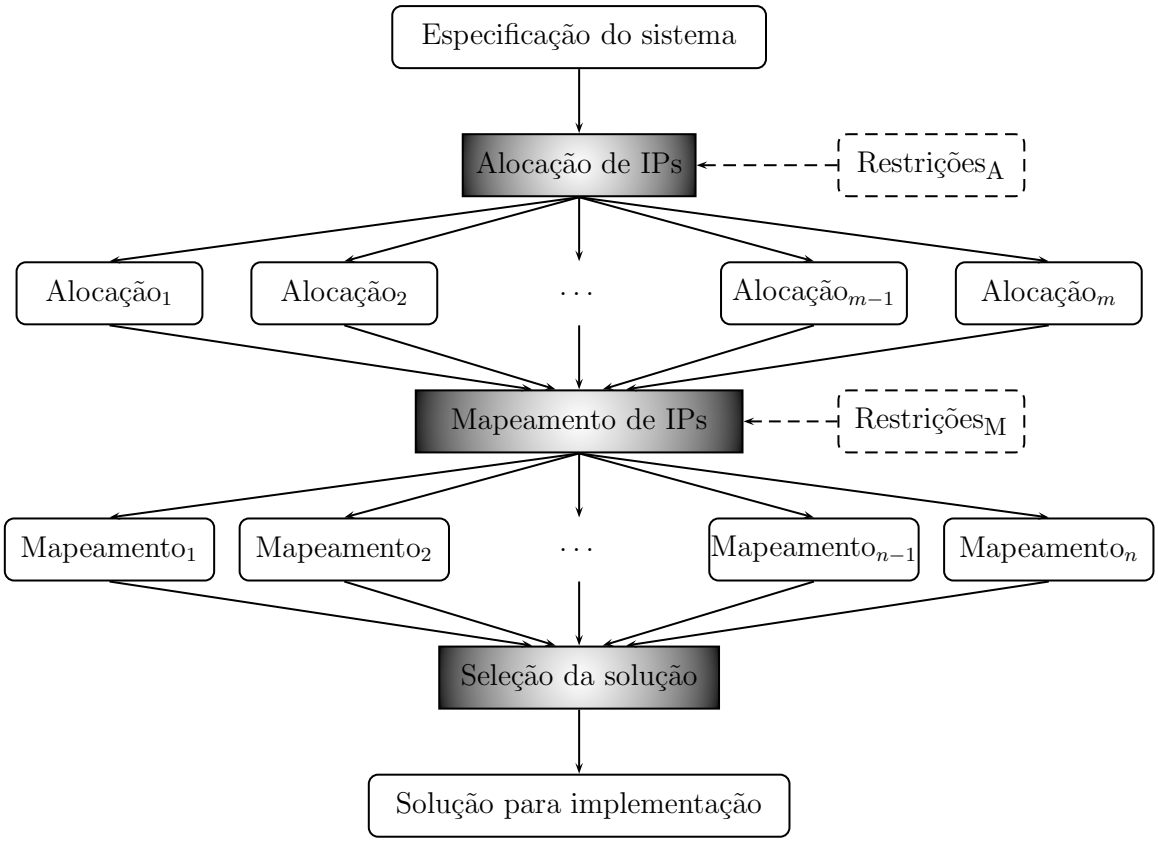


Figura 12: Fluxo proposto de projeto de sistemas embutidos para plataforma NoC

fluxo de projeto baseado em plataforma NoC começa com as especificações do projeto e com descrições abstratas. Ao longo do projeto o nível de abstração é reduzido e as descrições são validadas. A validação das descrições é feita em relação a diferentes critérios (ou objetivos). Para considerar o balanço entre os diferentes objetivos é proposto um modelo de projeto onde múltiplas descrições são avaliadas a cada etapa do projeto. O problema de otimização de projeto de uma plataforma NoC, é um tipo de problema de otimização multiobjetivo. No próximo capítulo será apresentado este tipo de problema de otimização.

Capítulo 2

OTIMIZAÇÃO MULTIOBJETIVO

A SOLUÇÃO de muitos problemas reais nem sempre é única. Muitos problemas cotidianos e de engenharia apresentam mais de uma solução ótima. Cada uma destas soluções apresenta um equilíbrio entre os aspectos distintos do problema. Por exemplo, considere o projeto de um *hardware* onde os objetivos são reduzir a área ocupada e o consumo de energia. Não é possível afirmar que uma solução com baixo consumo de energia que ocupa uma grande área é melhor do que uma solução com alto consumo de energia que ocupa uma pequena área. Ambas as soluções apresentam um balanço entre os objetivos de interesse. Problemas que apresentam este tipo de comportamento são chamados de problemas multiobjetivos. A solução de problemas de otimização multiobjetivos (POM), assim como a solução de vários problemas de otimização, é de difícil análise através de métodos determinísticos.

Este capítulo apresentará os conceitos relacionados à otimização multiobjetivo e os diferentes métodos de resolução, destacando os métodos não determinísticos. A Seção 2.1 introduz a otimização multiobjetivo, definindo formalmente um POM, apresenta a complexidade deste tipo de problema e introduz os conceitos de Pareto. A Seção 2.2 apresenta a taxonomia dos métodos de busca o otimização e diferentes estratégias adotadas para solucionar problemas de otimização multiobjetivo. A Seção 2.3 faz o fechamento deste capítulo com algumas considerações finais e introduz o assunto a ser abordado no próximo capítulo.

2.1 Conceitos Básicos de Otimização Multiobjetivo

A maioria dos problemas encontrados no mundo real requer a otimização de vários objetivos simultaneamente. Muitas vezes é possível combinar os objetivos utilizando uma função e tratar o problema como se fosse um problema de otimização de um único objetivo. No entanto, em muitos casos, a natureza do problema não permite tal abordagem e então cada objetivo deve ser otimizado separadamente de modo que a otimização de cada um resulte na solução do problema.

Problemas de otimização com um único objetivo possuem apenas uma solução ótima, que é traduzida no valor máximo ou mínimo do objetivo em questão. POMs possuem um conjunto de soluções que, quando avaliadas, formam vetores cujos componentes representam pontos no domínio espacial de objetivos. Características do problema farão com que uma solução representada por um vetor seja mais ou menos adequada que uma outra solução representada por outro. Em seguida, introduzimos algumas definições usadas no estudo de POMs.

2.1.1 Problema de otimização multiobjetivo

Seja $\Omega \subseteq \mathbb{R}^n$ o *espaço de busca* de dimensão n e $\Psi \subseteq \Omega$ o espaço de busca possível. Não havendo restrições, o espaço de busca possível é igual ao espaço de busca. Seja $\mathtt{x} = (x_1, x_2, \ldots, x_n) \in \Omega$, o *vetor de decisão*, onde cada $x_i, i = 1, \ldots, n$ representa uma *variável de decisão*. As variáveis de decisão x_i representam as variáveis de entrada que podem ser manipuladas com o intuito de resolver o POM. Para cada solução encontrada existe um vetor de decisão $\mathtt{x}$ associado.

Sejam $\Lambda \subseteq \mathbb{R}^k$, o *espaço de objetivos* de dimensão k e $\mathtt{f(x)} = (f_1(\mathtt{x}), f_2(\mathtt{x}), \ldots, f_k(\mathtt{x})) \in \Lambda$, o *vetor de objetivos*, onde cada $f_i : \Omega \rightarrow \Lambda, i = 1, \ldots k$ representa uma *função objetivo* e $\mathtt{x} \in \Omega$. A função objetivo mapeia um valor para um determinado objetivo do POM a partir de um vetor de decisão, enquanto que o vetor de objetivos mapeia um ponto no espaço de objetivos. O mapeamento do vetor de objetivos $\mathtt{f(x)}$ pode ser alterado devido à restrições representadas por inequações e equações (SRINIVAS; DEB, 1994).

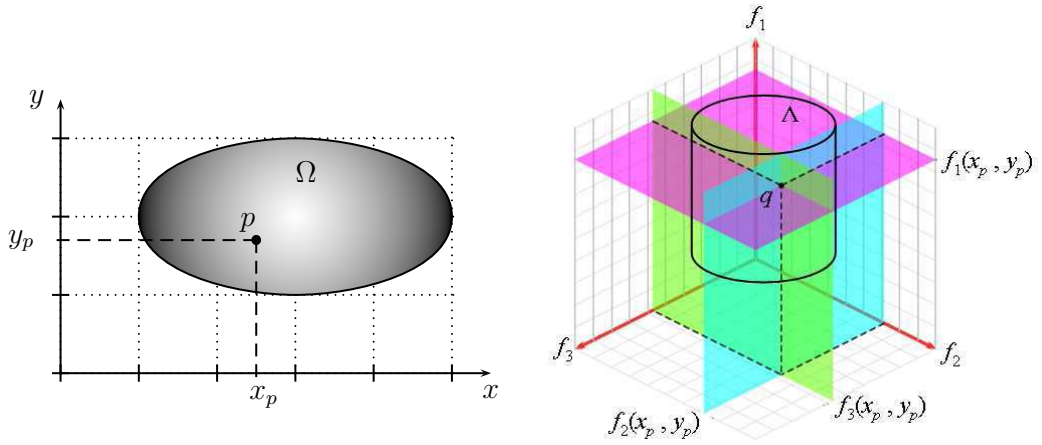
Definição 1 (Problema de Otimização Multiobjetivo). *Um POM de n variáveis de decisão, k funções objetivo cujos espaços de busca e objetivos são Ω e Λ, respectivamente, é definido pelo sistema:*

$$\begin{aligned} &\min_{\mathtt{x} \in \Omega} \quad \mathtt{f(x)}, \\ &\textit{sujeito à} \begin{cases} g_i(\mathtt{x}) \leq 0, & i = 1, \ldots, m_1 \\ h_i(\mathtt{x}) = 0, & i = 1, \ldots, m_2 \\ \mathtt{x} \in [\mathtt{x}_{min}, \mathtt{x}_{max}]^n \end{cases} \end{aligned} \tag{1}$$

Definição 2 (Mínimo Global). *Dada uma função $\mathtt{f} = (f_1(\mathtt{x}), f_2(\mathtt{x}) : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $\Omega \neq \emptyset$, o valor de $\mathtt{f}(\mathtt{x}_0) \in \Omega$ é dito um mínimo global se e somente se $\forall \mathtt{x} \in \Omega : f_i(\mathtt{x}_0) < f_i(\mathtt{x})$.*

Na Equação 1, a inequação $g_i \leq 0$ e a equação $h_i = 0$ representam as restrições do POM enquanto que $\mathtt{x} \in [\mathtt{x}_{min}, \mathtt{x}_{max}]^n$ representa os limites dos vetores de decisão no espaço de busca de dimensão n. Soluções possíveis são todas as soluções $\mathtt{x}^* = \mathtt{x} \in \Psi$.

A Figura 13 exemplifica o caso de um POM com 2 variáveis de decisão ($n = 2$) e 3 funções objetivo ($k = 3$). Note que as restrições impostas delimitam o espaço de objetivos, fazendo com que nem todos os mapeamentos sejam válidos, isto é, estejam compreendidos dentro do espaço válido de objetivos.



(a) Domínio dos critérios (b) Domínio dos objetivos

Figura 13: Mapeamento de um POM

Normalmente, POMs possuem objetivos conflitantes, isto é, a otimização de um objetivo resulta no detrimento de outro objetivo. Por exemplo, no caso da otimização de projeto de um microprocessador, enquanto que a velocidade deve ser maximizada, o consumo de energia deve ser minimizado. Ao aumentar a frequência de operação de um processador há um ganho de velocidade na execução das instruções mas por outro lado há um aumento no consumo de energia. Este é um caso real de um POM com objetivos conflitantes.

Para determinar a melhor ou pior solução em um problema com um único objetivo, basta ordenar as soluções de acordo com os valores mapeados através da função objetivo. A solução desejada será a primeira ou a última da lista ordenada, dependendo do tipo de ordenação (crescente ou decrescente) e do tipo de otimização (minimização ou maximização). No caso de POMs, para avaliar as soluções encontradas, a ordenação do espaço de objetivos deve ser parcial, isto é, a ordenação das soluções encontradas deve ser feita para cada objetivo, tendo por consequência, um grande tempo polinomial de busca em todo o espaço de objetivos. Encontrar o ótimo global em um POM se torna um problema de complexidade *NP*-completo (GAREY; JOHNSON, 1979), como especificado na próxima seção.

2.1.2 Complexidade

A complexidade computacional dos problemas está diretamente relacionada com o custo computacional (tempo) necessário para resolver o problema de acordo com o tamanho da representação digital do problema (instância do problema). Problemas de complexidade P são problemas que podem ser resolvidos por algoritmos determinísticos a um custo computacional que cresce polinomialmente de acordo com o tamanho da instância do problema. Problemas de complexidade *NP* são problemas que apresentam um aumento de custo computacional exponencial em relação ao tamanho da instância quando resolvidos por algoritmos determinísticos.

Problemas da classe *NP*-completo são problemas especiais da classe *NP*, onde cada problema de *NP* pode ser polinomialmente transformável em um problema *NP*-completo. Sendo assim, para um problema ser *NP*-completo, é necessário que:

- Prove-se que o problema está na classe *NP*.
- Prove-se que um problema *NP*-completo conhecido pode ser polinomialmente transformado para o problema em questão.

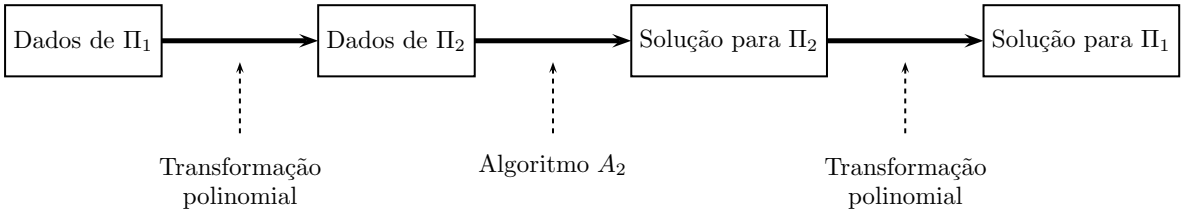


Figura 14: Transformação Polinomial de dois problemas

A Figura 14 mostra o esquema de uma transformação polinomial onde, dados dois problemas, Π_1 e Π_2, o algoritmo A_2 resolve o problema Π_2. Se for possível transformar Π_1 em Π_2 e a solução de Π_2 em solução de Π_1, então A_2 pode ser utilizado para resolver Π_1. Se pudermos realizar as transformações nos dois sentidos em tempo polinomial (não exponencial), então Π_1 é polinomialmente transformável em Π_2. Sabe-se que o problema de otimização combinacional é um problema no mínimo *NP*-completo (GAREY; JOHNSON, 1979). Os problemas de alocação e mapeamento são problemas de otimização combinacional. Além do fator combinacional, estes problemas são avaliados por diferentes objetivos, o que dificulta a busca por uma solução ótima global.

O conceito de ótimo global, para POMs, foi proposto inicialmente em 1881 por Francis Y. Edgeworth (EDGEWORTH, 1881), que definiu como ótimo o melhor balanço entre os múltiplos objetivos avaliados em um problema, de modo que a otimização de um objetivo não

deteriore significativamente os outros objetivos. Após 15 anos este conceito foi generalizado e estendido por Vilfredo Pareto (PARETO, 1896), dando o nome de solução ótima de Pareto a toda solução ótima de um POM. Os conceitos de Pareto são detalhados na próxima seção.

2.1.3 Conceitos de Pareto

Para identificar uma solução ótima em um conjunto de soluções com diferentes objetivos, Pareto definiu alguns conceitos. De acordo com a Definição 2, os conceitos definidos por Pareto são: Dominância, dominância fraca, solução ótima de Pareto, conjunto ótimo de Pareto e fronteira ótima de Pareto. A seguir, estão as definições destes conceitos introduzidos por Pareto.

Definição 3 (Dominância). *Diz-se que uma solução u, representada por um vetor $\mathtt{f}(\mathtt{x}) = [f_1(\mathtt{x}), \ldots, f_k(\mathtt{x})] \mid \mathtt{x} \in \Omega$, domina uma outra solução v, representada pelo vetor $\mathtt{f}(\mathtt{x}') = [f_1(\mathtt{x}'), \ldots, f_k(\mathtt{x}')] \mid \mathtt{x}' \in \Omega$, se e somente se u for parcialmente menor que v, isto é, $\forall i \in 1, \ldots, k : f_i(\mathtt{x}) \leq f_i(\mathtt{x}') \wedge \exists i \in 1, \ldots, k : f_i(\mathtt{x}) < f_i(\mathtt{x}')$. Essa relação será denotada por $u \prec v$.*

Definição 4 (Dominância fraca). *Diz-se que uma solução u, representada por um vetor $\mathtt{f}(\mathtt{x}) = [f_1(\mathtt{x}), \ldots, f_k(\mathtt{x})] \mid \mathtt{x} \in \Omega$, exerce dominância fraca sobre uma outra solução v, representada pelo vetor $\mathtt{f}(\mathtt{x}') = [f_1(\mathtt{x}'), \ldots, f_k(\mathtt{x}')] \mid \mathtt{x}' \in \Omega$, se e somente se u for parcialmente igual a v, isto é, $\forall i \in 1, \ldots, k : f_i(\mathtt{x}) \leq f_i(\mathtt{x}')$. Essa relação será denotada por $u \preceq v$.*

Note que a diferença entre a noção de dominância e dominância fraca é que a última não considera a condição na qual pelo menos um objetivo deve ser melhor.

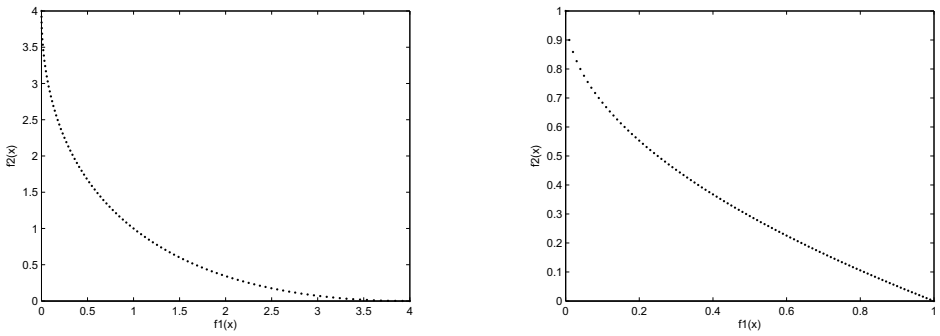
Definição 5 (Solução ótima de Pareto). *Diz-se que a solução u, representada por um vetor $\mathtt{f}(\mathtt{x}) = [f_1(\mathtt{x}), \ldots, f_k(\mathtt{x})] \mid \mathtt{x} \in \Omega$ é uma solução ótima de Pareto em relação ao conjunto Ω se e somente se $\nexists \mathtt{x}' \in \Omega \mid v = \mathtt{f}(\mathtt{x}') = [f_1(\mathtt{x}'), \ldots, f_k(\mathtt{x}')] \prec u = \mathtt{f}(\mathtt{x}) = [f_1(\mathtt{x}), \ldots, f_k(\mathtt{x})]$.*

Definição 6 (Conjunto ótimo de Pareto). *Para uma dada função de avaliação $\mathtt{f}(\mathtt{x})$ de um POM, o conjunto ótimo de Pareto ($\mathscr{P}^*$) é definido como:*

$$\mathscr{P}^* := \{\mathtt{x} \in \Omega \mid \nexists \mathtt{x}' \in \Omega : \mathtt{f}(\mathtt{x}') \prec \mathtt{f}(\mathtt{x})\}$$

O conjunto ótimo de Pareto contem o conjunto de soluções que possuem o melhor balanço entre os objetivos do POM. Os vetores objetivos que fazem parte do conjunto ótimo de Pareto formam uma fronteira definida a seguir.

Definição 7 (Fronteira ótima de Pareto). *Para uma dada função de avaliação $\mathtt{f}(\mathtt{x})$ de um POM e um conjunto ótimo de Pareto ($\mathscr{P}^*$), a fronteira ótima de Pareto ($\mathscr{F}\mathscr{P}^*$) é definida como:*



(a) Fronteira convexa uniforme (b) Fronteira convexa não uniforme

Figura 15: Exemplos de fronteiras ótimas de Pareto convexas

$$\mathscr{F}\mathscr{P}^* := \{\mathtt{u} = \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), \ldots, f_k(\mathtt{x})) \mid \mathtt{x} \in \mathscr{P}^*\}$$

A fronteira ótima de Pareto contem o conjunto de soluções correspondentes à variáveis de decisões que não são dominadas por nenhuma outra solução. Os seguintes exemplos ilustram diferentes tipos de fronteiras ótimas de Pareto para as seguintes funções:

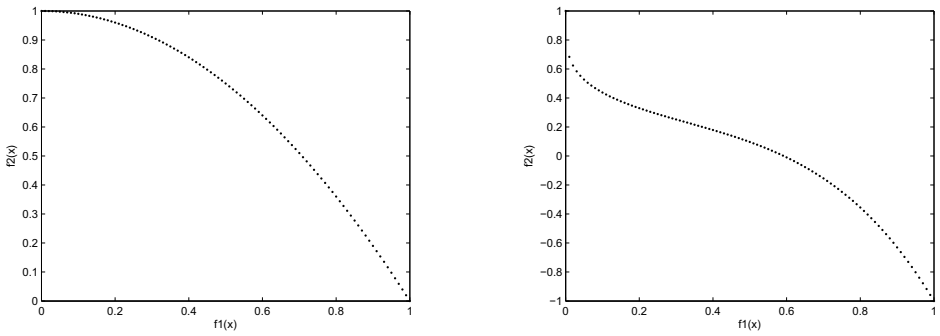
- A Figura 15(a) mostra a fronteira ótima de Pareto convexa e uniforme do problema multiobjetivo descrito na Equação 2.

$$\begin{aligned} &\min_{\mathtt{x}\in\Omega} \quad \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), f_2(\mathtt{x})), \\ &\text{onde} \quad \begin{cases} f_1(\mathtt{x}) &= \frac{1}{n}\sum_{j=0}^{n} x_j^2, \\ f_2(\mathtt{x}) &= \frac{1}{n}\sum_{j=0}^{n} (x_j - 2)^2. \end{cases} \end{aligned} \tag{2}$$

- A Figura 15(b) mostra a fronteira ótima de Pareto convexa e não uniforme do problema multiobjetivo descrito na Equação 3.

$$\begin{aligned} &\min_{\mathtt{x}\in\Omega} \quad \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), f_2(\mathtt{x})), \\ &\text{onde} \quad \begin{cases} f_1(\mathtt{x}) &= x_1, \\ f_2(\mathtt{x}) &= g(\mathtt{x})(1 - \sqrt{f_1(\mathtt{x})/g(\mathtt{x})}, \end{cases} \\ &\text{com} \quad g(\mathtt{x}) = 1 + \tfrac{9}{n-1}\sum_{j=2}^{n} x_j. \end{aligned} \tag{3}$$

- A Figura 16(a) representa a fronteira ótima de Pareto concava do problema multiobjetivo



(a) Fronteira côncava

(b) Fronteira parcialmente convexa e parcialmente côncava

Figura 16: Exemplos de fronteiras ótimas de Pareto côncavas

descrito na Equação 4.

$$\begin{aligned} &\min_{\mathtt{x}\in\Omega} \quad \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), f_2(\mathtt{x})), \\ &\text{onde} \quad \begin{cases} f_1(\mathtt{x}) &= x_1, \\ f_2(\mathtt{x}) &= g(\mathtt{x})(1-(f_1(\mathtt{x})/g(\mathtt{x}))^2), \end{cases} \\ &\text{com} \quad g(\mathtt{x}) = 1 + \tfrac{9}{n-1}\textstyle\sum_{j=2}^{n} x_j. \end{aligned} \tag{4}$$

- A Figura 16(b) mostra a fronteira ótima de Pareto que é parcialmente côncava e parcialmente convexa do problema multiobjetivo descrito na Equação 3.

$$\begin{aligned} &\min_{\mathtt{x}\in\Omega} \quad \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), f_2(\mathtt{x})), \\ &\text{onde} \quad \begin{cases} f_1(\mathtt{x}) &= x_1, \\ f_2(\mathtt{x}) &= g(\mathtt{x})(1-\sqrt[4]{f_1(\mathtt{x})/g(\mathtt{x})}-(f_1(\mathtt{x})/g(\mathtt{x}))^4), \end{cases} \\ &\text{com} \quad g(\mathtt{x}) = 1 + \tfrac{9}{n-1}\textstyle\sum_{j=2}^{n} x_j. \end{aligned} \tag{5}$$

- A Figura 17 representa a fronteira ótima de Pareto que é descontínua e convexa do problema multiobjetivo descrito na Equação 3.

$$\begin{aligned} &\min_{\mathtt{x}\in\Omega} \quad \mathtt{f}(\mathtt{x}) = (f_1(\mathtt{x}), f_2(\mathtt{x})), \\ &\text{onde} \quad \begin{cases} f_1(\mathtt{x}) &= x_1, \\ f_2(\mathtt{x}) &= g(\mathtt{x})(1-\sqrt{f_1(\mathtt{x})/g(\mathtt{x})}-(f_1(\mathtt{x})/g(\mathtt{x}))sin(10\pi f_1(\mathtt{x}))), \end{cases} \\ &\text{com} \quad g(\mathtt{x}) = 1 + \tfrac{9}{n-1}\textstyle\sum_{j=2}^{n} x_j. \end{aligned} \tag{6}$$

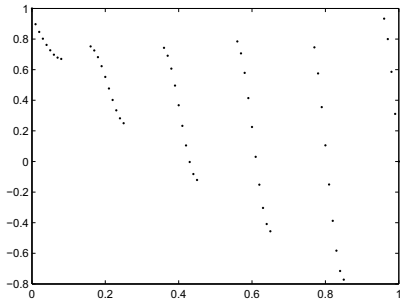


Figura 17: Exemplo de fronteira ótimas de Pareto convexa discreta

Nos quatro últimos exemplos, note as similaridades entre a Equação 3, Equação 4, Equação 5 e Equação 6, onde a somente a função objetivo f_2 é alterada.

Soluções ótimas de Pareto, também podem ser chamadas de soluções *não inferiores*, *admissíveis* ou *eficientes* (HORN, 1997) e seus respectivos vetores objetivos são chamados de *não dominados*. Juntas, formam o conjunto de todas as soluções cujos vetores correspondentes não são dominados por nenhum outro vetor do espaço de busca Ω. Quando mapeadas pelo vetor de objetivos $\mathtt{f}(\mathtt{x})$ e tendo seus valores representados graficamente no espaço de objetivos Λ, formam a fronteira ótima de Pareto. Como o conjunto ótimo de Pareto é um subconjunto de todas as possíveis soluções em Ω, a existência de um subconjunto dominado pelo conjunto ótimo de Pareto, forma uma fronteira adjacente à fronteira ótima de Pareto, com soluções dominadas pelas soluções ótimas de Pareto. Excluindo de Ω o conjunto ótimo de Pareto, o novo conjunto de soluções não dominadas se torna a fronteira adjacente à fronteira ótima de Pareto. Através da exclusão de conjuntos de soluções não dominadas é possível identificar diferentes fronteiras de Pareto (não ótimas) para um POM.

Encontrar a fronteira de Pareto-ótima pode ser extremamente difícil em casos onde existem muitas fronteiras próximas a fronteira ótima de Pareto. Esta tarefa pode ser reduzida à encontrar uma aproximação da fronteira ótima de Pareto, desde que (ENGELBRECHT, 2006):

- a distância separando esta fronteira da fronteira ótima de Pareto seja minimizada,
- o conjunto de soluções não dominadas seja o mais diverso possível, e
- as soluções ótimas de Pareto encontradas sejam preservadas.

Encontrar uma aproximação para a fronteira ótima de Pareto pode ser visto como um POM, onde o primeiro objetivo garante uma boa aproximação e o segundo objetivo garante boa exploração do espaço de objetivos. Alguns algoritmos de otimização multiobjetivo que permitem a aproximação da fronteira ótima de Pareto, utilizam os conceitos de ε-dominância

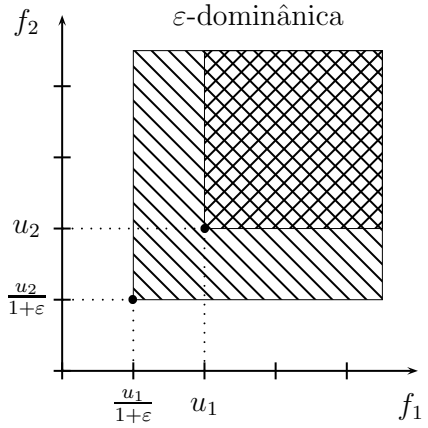


Figura 18: Ilustração do conceito de $\varepsilon-$dominância

e de conjunto ε-aproximação da fronteira ótima de Pareto (LAUMANNS *et al.*, 2002), definidos a seguir. A Figura 18 ilustra o conceito de $\varepsilon-$dominância.

Definição 8 ($\varepsilon-$Dominância). *Uma solução u representada por um vetor $\mathtt{f(x)}$, exerce ε-dominância sobre uma outra solução v representada por um outro vetor $\mathtt{f(x')}$, sendo $\varepsilon > 0$, se, e somente se: (i) $u_i/(1+\varepsilon) \leq v_i : \forall i \in \{1, \ldots, k\}$, e (ii) $\exists i \in \{1, \ldots, k\} : u_i/(1+\varepsilon) < v_i$. Essa relação é denotada por $u \prec_\varepsilon v$.*

Definição 9 (Conjunto $\varepsilon-$aproximação da fronteira ótima de Pareto). *Seja $\mathscr{S}$ um conjunto de soluções e $\varepsilon > 0$. Um conjunto é dito $\varepsilon-$aproximação da fronteira ótima de Pareto, $\mathscr{FP}^*_\varepsilon$, em relação ao conjunto $\mathscr{S}$, se e somente se: $\forall s \in \mathscr{S} : \exists p \in \mathscr{FP}^*_\varepsilon \mid p \prec_\varepsilon s$.*

2.2 Métodos de Busca e Otimização

Por muitos anos, algoritmos determinísticos foram largamente empregados devido ao largo embasamento teórico disponível para resolução de problemas. Um algoritmo determinístico funciona como uma função não caótica, isto é, se sabendo a entrada é possível determinar a saída. Seu comportamento é previsível e seus resultados podem ser comprovados matematicamente. Como visto na Seção 2.1.2, o custo computacional de problemas de complexidade *NP* cresce exponencialmente em relação à instância do problema. Para este tipo de problemas, há uma economia de custo computacional quando se utilizam técnicas não determinísticas, que podem ser baseadas em heurísticas ou inspiradas na natureza.

2.2.1 Taxonomia

Geralmente, técnicas de busca e otimização podem ser classificadas em três categorias: enumerativas, determinísticas e estocásticas (VELDHUIZEN, 1999). Note que uma busca enumerativa

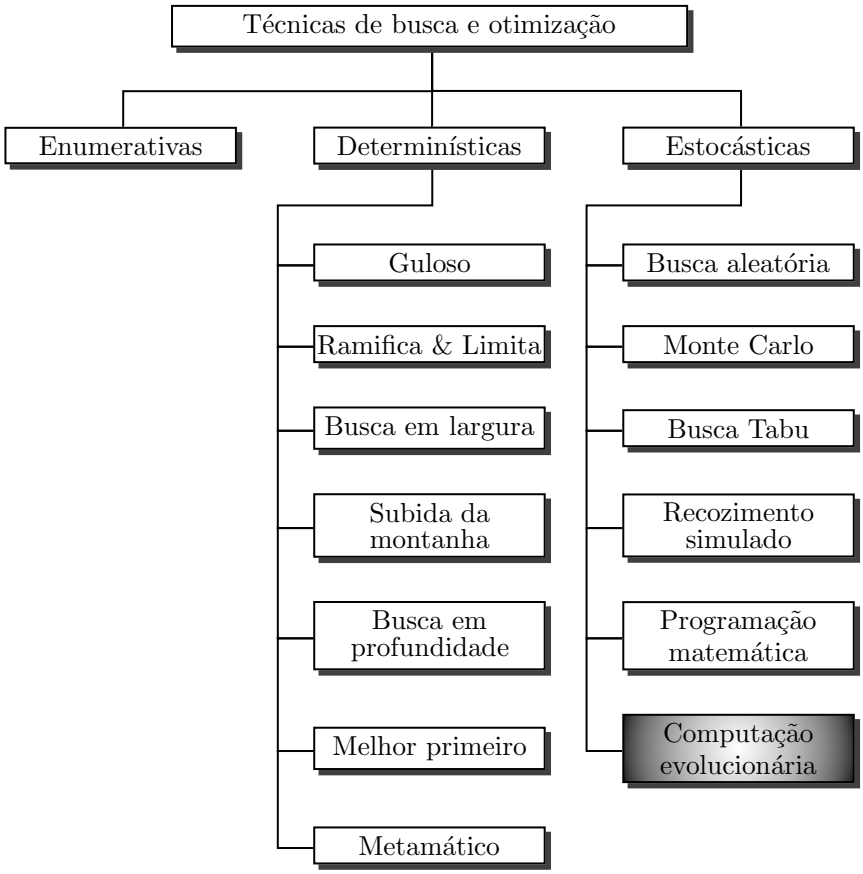


Figura 19: Classificação das técnicas de busca e otimização a um objetivo

é também determinística mas não faz uso de heurísticas. A Figura 19 mostra a taxonomia das técnicas de busca e otimização em geral.

As técnicas enumerativas são as mais simples para solucionar problemas de busca e otimização. Dentro de um espaço de busca finito cada solução possível é avaliada. É fácil perceber que esta técnica se torna ineficiente ou mesmo inviável à medida que o espaço de busca aumenta. Como muitos problemas reais possuem um grande espaço de busca, é preciso limitar este espaço de alguma maneira para reduzir o custo computacional demandado na solução do problema. O espaço de busca deve ser definido de modo que soluções aceitáveis possam ser encontradas em um tempo também aceitável. Algoritmos determinísticos tentam resolver este problema incorporando conhecimentos prévios (heurísticas), sobre o espaço de busca de modo a limitá-lo. Muitos destes algoritmos representam o problema utilizando estruturas de dados de árvores e/ou grafos.

Algoritmos do tipo *guloso* encontram ótimos locais assumindo que soluções parciais (soluções para uma parte do problema) ótimas locais são sempre parte da solução ótima global (HUSBANDS, 1992)(BRASSARD; BRATLEY, 1988). Em problemas que não seguem esta premissa,

este tipo de algoritmo não deve ser utilizado. O algoritmo de *subida da montanha* realiza a busca na direção de subida (ou descida) mais íngreme assumindo que assim chegará ao topo (ou fundo) mais rapidamente, escapando de ótimos locais. A direção é determinada a partir do cálculo do gradiente a partir do ponto atual. Este algoritmo tem melhor desempenho para funções unimodais, mas a presença de ótimos locais, platôs, ou cumes na superfície de busca, reduzem a eficiência do mesmo (RUSSEL; NORVIG, 1995). Métodos gulosos e de subida da montanha são ditos *irrevogáveis*. A busca é feita de modo que não é possível voltar atrás para examinar soluções encontradas anteriormente (PEARL, 1989).

O método *Ramifica e Limita* (*Branch and Bound*) precisa de um algoritmo com heurísticas específicas para limitar o espaço de busca (GAREY; JOHNSON, 1979) (PEARL, 1989). A partir de um ponto inicial é calculada uma fronteira (*bound*), limitando o espaço de busca. Dentro desta fronteira o ponto é avaliado se é *promissor* ou não, ou seja, se o ponto está de acordo com as heurísticas adotadas. O algoritmo cria uma ramificação (*branch*) para o ponto mais promissor dentro da fronteira e calcula uma nova fronteira para este novo ponto, reiniciando o procedimento (NEAPOLITAN; NAIMIPOUR, 1996).

A busca em profundidade é uma técnica utilizada na otimização de problemas que dependem de sucessivas tomadas de decisões, que são representadas através de uma *árvore de decisões*. A busca é feita a partir dos nós mais profundos da árvore de decisões e vai subindo até que a solução desejada seja encontrada. É dita uma técnica *cega* ou *desinformada* por não utilizar nenhuma heurística do problema. Na *busca em largura*, a cada iteração, os nós da árvore que estão na mesma profundidade (ou nível) são visitados (PEARL, 1989). Também é considerada uma técnica cega ou desinformada. A técnica *melhor primeiro* baseia a busca em uma heurística que atribui valores qualificando quão promissor é um nó em relação às heurísticas adotadas. Este valor pode expressar, por exemplo, o custo para alcançar o nó. Os nós mais promissores são examinados primeiro (PEARL, 1989). Finalmente, os métodos matemáticos, que utilizam ferramentas matemáticas como o cálculo numérico, o cálculo diferencial e integral e a álgebra linear, podem ser utilizados quando o espaço de busca apresenta continuidade (ANTON, 1988). Estes métodos requerem um custo elevado de processamento.

Todos os métodos determinísticos listados na Figura 19 conseguem resolver com sucesso uma grande variedade de problemas (BRASSARD; BRATLEY, 1988) (GOLDBERG, 1989a) (NEAPOLITAN; NAIMIPOUR, 1996). No entanto, a maioria dos POMs apresentam uma alta dimensionalidade e descontinuidade, são multimodais e apresentam alta complexidade. Esses métodos são ineficazes quando aplicados a problemas *NP*-completos de alta dimensionalidade

porque requerem conhecimento prévio para guiar a busca e limitar o respectivo espaço, que nestes casos, é excepcionalmente grande (FOGEL; OWENS; WALSH, 1966) (GAREY; JOHNSON, 1979) (GOLDBERG, 1989a). Problemas que apresentam uma ou mais das características acima, são chamados de problemas *irregulares* (VELDHUIZEN, 1999).

Como muitos POMs científicos e de engenharia são irregulares, técnicas de busca e otimização enumerativas e determinísticas são inviáveis. Métodos estocásticos de busca e otimização, tais como, recozimento simulado, Monte Carlo, busca Tabu e computação evolucionária (CE), foram desenvolvidos como alternativa para a resolução dos POMs irregulares (GOLDBERG, 1989a) (MICHALEWICZ, 1994). Métodos estocásticos requerem uma função que avalie as possíveis soluções encontradas e um mecanismo de codificação e decodificação entre os domínios do problema e algoritmo. Embora muitos algoritmos que implementam técnicas estocásticas encontrem soluções ótimas, muitas vezes é impossível provar que não há solução ótima melhor. Estes métodos geralmente encontram *boas* soluções para muitos problemas de otimização difíceis de serem resolvidos através de métodos determinísticos(GOLDBERG, 1989a)(HUSBANDS, 1992).

A *busca aleatória* é a técnica de busca estocástica mais simples que existe, de modo que ela simplesmente avalia um determinado número de soluções aleatórias. Uma variação da busca aleatória é a *caminhada aleatória*, onde a solução seguinte é escolhida aleatoriamente usando a última solução avaliada como ponto de partida. Assim como a técnica enumerativa, a busca aleatória é ineficiente para muitos POMs por não incorporar conhecimento sobre o domínio do problema.

A *técnica de recozimento simulado* (RUSSEL; NORVIG, 1995) é baseada no processo metalúrgico de aquecimento e resfriamento controlado de um material com a finalidade de aumentar sua resistência. Enquanto que a *técnica de subida da montanha* escolhe a melhor direção a partir de um ponto, a técnica de recozimento simulado faz uma escolha aleatória. Se a escolha adotada resultar em uma solução melhor que a atual, então esta recebe probabilidade $p = 1$, senão recebe uma probabilidade de $p < 1$. As escolhas estão atreladas com o parâmetro de temperatura T que vai sendo reduzindo ao longo do processo e deixando o mesmo menos aleatório (RUSSEL; NORVIG, 1995). No processo de fabricação do aço, se a temperatura é reduzida lentamente, este atinge a configuração de menor energia interna, resultando em maior resistência. A técnica de recozimento simulado faz uma analogia com o processo de fabricação do aço, tornando a busca menos aleatória no final, quando se aproxima do ótimo global.

Em geral, a *técnica de Monte Carlo* (ROBERT; CASELLA, 2005) envolve simulações a

partir de eventos estocásticos. Uma busca aleatória é feita de modo que a solução encontrada é inteiramente independente de decisões tomadas anteriormente e de decisões futuras (como ocorre na *técnica de subida da montanha* onde uma solução aponta a próxima direção). O resultado final é obtido a partir da análise dos resultados parciais obtidos previamente (SCHWEFEL, 1995).

A *busca tabu* (GLOVER; LAGUNA, 1997) é uma metaheurística desenvolvida para evitar que o processo de otimização fique preso em ótimos locais. As soluções encontradas e os caminhos que o algoritmo percorreu são armazenados, e esta informação é usada para restringir a escolha de novos caminhos e permitir a avaliação de novas soluções. A busca tabu é normalmente utilizada em conjunto com outro método de busca e otimização (SCHWEFEL, 1995).

A *programação matemática* (WHITE, 1973) consiste em um conjunto de técnicas que utilizam métodos determinísticos e estocásticos para solucionar POMs. Estas técnicas foram desenvolvidas por pesquisadores da comunidade de Pesquisa Operacional (RAVINDRAN; PHILIPS; SOLBERG, 1987) e o maior diferencial é que estas priorizam as restrições ao invés das funções objetivo durante o processo de otimização (SCHWEFEL, 1995).

A *computação evolucionária* é um método genérico utilizado que simula processos evolucionários naturais. O campo de computação evolucionária engloba as técnicas de *algoritmos genéticos*, *estratégias evolucionárias* e *programação evolucionária*, que também são conhecidos como *algoritmos evolucionários*. Estas técnicas são baseadas na evolução natural e na teoria de Charles Darwin da seleção natural ou sobrevivência do mais apto (GOLDBERG, 1989a). O que há em comum entre os algoritmos evolucionários é o processo de reprodução dos indivíduos, mutação, competição e seleção dos mais aptos (FOGEL, 1997). Em geral um algoritmo evolucionário possui uma população de soluções codificadas, chamadas de *indivíduos*. Os indivíduos são manipulados por uma sequência de operadores genéticos e avaliados através de uma função de *aptidão*. A aptidão de um indivíduo determina se o mesmo sobreviverá ou não na geração seguinte. Este assunto será abordado com mais detalhes na Seção 3.1 do Capítulo 3.

2.2.2 Estratégias de seleção para otimização multiobjetivo

Um POM requer duas fases para ser solucionado (LAUMANNS; RUDOLPH; SCHWEFEL, 1999): *(i)* o processo de otimização para encontrar as soluções ótimas de Pareto e *(ii)* o processo de seleção que permite escolher a solução mais adequada. O último processo é atribuído a um tomador de decisão ou especialista. Este deverá considerar os aspectos inerentes do problema

e adotar uma das três estratégias apresentadas a seguir:

- *A priori*: É uma estratégia do tipo *decide-então-busca*. O especialista combinará os objetivos em uma função de avaliação dando maior importância ao objetivo mais custoso. Aplica-se esta estratégia em casos que o tomador de decisão conhece bem o problema a ponto de distinguir o objetivo mais custoso. Havendo mudança de importância entre os objetivos é necessário repetir o processo de otimização.

- *A posteriori*: É uma estratégia do tipo *busca-então-decide* e consiste em encontrar o máximo de soluções não dominadas e então decidir por usar uma destas. Havendo mudança de prioridade entre os objetivos o processo de otimização não precisa ser repetido, basta que o especialista selecione outra solução que atenda às novas necessidades.

- *Progressivo*: Periodicamente, durante o processo de otimização, a importância dos objetivos pode ser modificada, de modo que no final do processo o balanço entre os objetivos seja o melhor possível.

Escolher uma solução de um POM baseando-se na otimização de apenas um objetivo, pode ignorar a existência de soluções melhores em relação aos demais objetivos. O conjunto ótimo de Pareto contêm estas soluções. Independente da estratégia adotada pelo especialista, as soluções escolhidas sempre fazem parte no conjunto ótimo de Pareto, que é representado pela fronteira ótima de Pareto. Identificar o conjunto de soluções ótimas de Pareto de um POM é fundamental para facilitar o trabalho do especialista (VELDHUIZEN, 1999).

A complexidade dos POMs e as lacunas existentes nos métodos determinísticos de busca e otimização, levaram a comunidade de Pesquisa Operacional (HILLIER; LIEBERMAN, 2005) à busca de novos métodos. Este novos métodos lineares ou não lineares, determinísticos ou estocásticos, podem ser agrupados como métodos de *programação matemática* (WHITE, 1973). Estes métodos se assemelham pelo fato de tratarem com prioridade as restrições dos problemas (SCHWEFEL, 1995). A *programação linear* é voltada para problemas onde a relação entre as funções objetivo e as restrições são lineares (HILLIER; LIEBERMAN, 2005). Inversamente, a *programação não linear* resolve alguns problemas não lineares, tendo necessariamente funções de restrições formando superfícies convexas (SCHWEFEL, 1995). Muitas restrições devem ser atendidas para viabilizar o uso da programação não linear. A maioria dos problemas científicos e de engenharia, podem ser modelados apenas por funções não lineares e, portanto, a programação não linear não resolve todos estes problemas (HILLIER; LIEBERMAN, 2005). Um outro método de programação matemática é a *programação estocástica* que é utilizada quando

parâmetros aleatórios e funções objetivo sujeitas à perturbações fazem parte da formulação do problema (SCHWEFEL, 1995).

2.3 Considerações Finais do Capítulo

Problemas de otimização multiobjetivo são problemas com dois ou mais objetivos de interesse a serem otimizados e sujeitos a uma ou mais restrições. Estes problemas podem apresentar múltiplos mínimos e máximos globais e os objetivos podem ser de natureza conflitante. Cada solução de um problema de otimização multiobjetivo recebe uma avaliação para cada objetivo. A avaliação de cada objetivo separadamente e o tamanho do espaço de busca, fazem com que este tipo de problema seja de alta complexidade. Para classificar as soluções de modo que o balanço entre os objetivos seja considerado, pode-se utilizar o conceito de dominância introduzido por Pareto. As técnicas de busca e otimização são classificadas como: enumerativas, determinísticas e estocásticas. As técnicas estocásticas empregam heurísticas e metaheurísticas com o objetivo de reduzir o tempo de computação necessário para resolver o problema. Os algoritmos evolucionários são uma metaheurística baseada na evolução das espécies. Estes algoritmos utilizam indivíduos que representam soluções para uma problema. Através de transformações ao longo de gerações, estes indivíduos são avaliados, sendo que os mais aptos sobrevivem. No capítulo seguinte serão apresentados os algoritmos evolucionários multiobjetivos.

Capítulo 3

OTIMIZAÇÃO EVOLUCIONÁRIA MULTIOBJETIVO

OS ALGORITMOS inspirados na teoria da evolução das espécies, de Charles Darwin (DARWIN, 1859), formam uma alternativa não determinística para solução de problemas de otimização. Estes algoritmos são chamados de algoritmos evolucionários (AEs). Ao contrário dos métodos determinísticos, estes algoritmos não são baseados em heurísticas e a prova de sua eficácia não é feita matematicamente mas sim experimentalmente. Uma classe especial de algoritmos evolucionários (AEs) é voltada para a solução de POMs. Dois destes algoritmos serão utilizados para otimizar as etapas de alocação e mapeamento de IPs em plataforma NoC. Como visto no Capítulo 1, estes são dois casos de POMs.

Este capítulo apresenta os AEs utilizados para resolver problemas de otimização a único ou múltiplos objetivos. A Seção 3.1 apresenta a definição de um AE, seus principais aspectos e os operadores genéticos. A Seção 3.2 apresenta os algoritmos evolucionários multiobjetivos classificados em três categorias. O critério de classificação usado está relacionado ao modo como as soluções encontradas são avaliadas de acordo com os seus objetivos. A Seção 3.3 seleciona dois algoritmos para evoluir alocação e mapeamento de IPs para a implementação eficiente de aplicações em paltaforma NoC, como visto no Capítulo 1. A Seção 3.4 encerra este capítulo com algumas considerações finais e introduz o assunto a ser abordado no próximo capítulo.

3.1 Algoritmos Evolucionários

Os termos usados em AEs, normalmente possuem significados análogos aos empregados no contexto biológico. Uma *estrutura* ou *indivíduo* é uma solução codificada para um problema. Normalmente, um indivíduo é representado por uma cadeia de *bits* que corresponde ao seu *genótipo*. Este genótipo quando decodificado revela as características do indivíduo e este con-

junto de características recebe o nome de *fenótipo*. O genótipo é composto de um ou mais *cromossomos* e cada cromossomo é composto de *genes* que recebem valores, chamados de *alelos*, de um determinado alfabeto genético. As representações mais utilizadas são: a binária, que representa os indivíduos como uma sequência de 0s ou 1s e a real, que representa os indivíduos como uma cadeia de valores reais. A posição do gene no cromossomo é identificada pelo *locus*. Um conjunto de indivíduos recebe o nome de *população*. Estes conceitos estão ilustrados na Figura 20 para as duas representações.

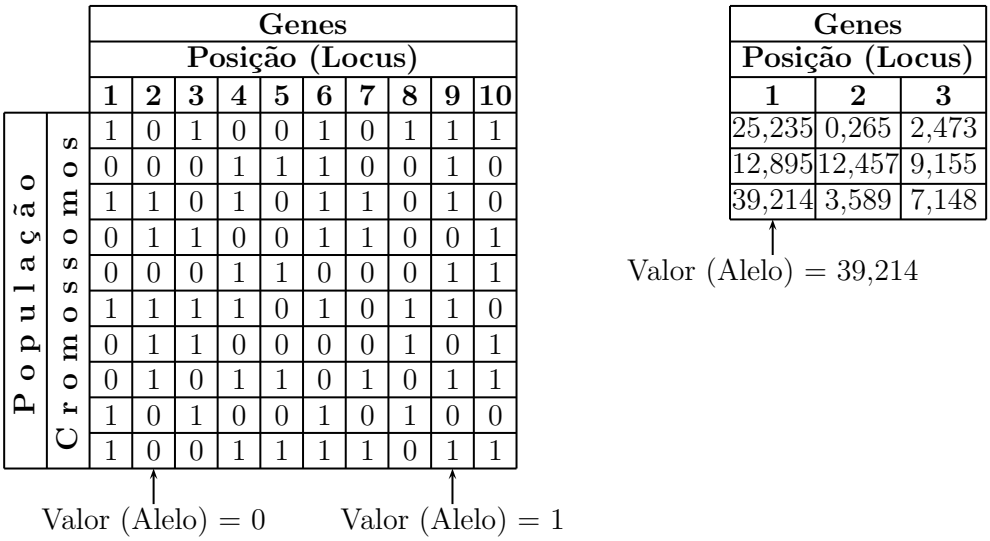


Figura 20: Estrutura de dados e terminologia de um AE

Segundo a teoria da evolução das espécies de Darwin (DARWIN, 1859), "o processo evolutivo caminha para melhorar os indivíduos". Ao longo das gerações, os genes dos indivíduos de uma população são modificados através de seleção natural ou através de mutação. O mecanismo de seleção natural pode ocorrer de dois modos. Um modo de seleção natural ocorre quando indivíduos mais aptos recombinam material genético com indivíduos do mesmo nível de aptidão, gerando, normalmente, indivíduos mais aptos. O outro modo de seleção natura é quando os indivíduos menos aptos não sobrevivem ao meio, restando cada vez mais indivíduos aptos para recombinação. A modificação dos genes através de mutação, ocorre através de uma variação genética promovida por fatores externos e resulta em um indivíduo cujas características não foram necessariamente herdadas de seus progenitores. Os mecanismos de seleção natural, que incluem a recombinação e a seleção dos mais aptos, e o mecanismo de mutação, são chamados de operadores genéticos. Transportando estes mecanismos naturais para um ambiente computacional e entendendo que os indivíduos são soluções de um problema, podemos dizer que o processo evolutivo caminha para melhorar as soluções de um problema.

3.1.1 Definição formal de AEs

Para definir formalmente um AE é possível representá-lo através de notação matemática, que pode ser utilizada para definir os conceitos utilizados por diferentes tipos de AEs, como o que será apresentado na Seção 3.2. Neste sentido, um AE está associado a um conjunto não vazio I chamado de *espaço de indivíduos* do AE. Cada indivíduo $\mathtt{i} \in I$ representa uma possível solução para o problema. Um indivíduo é normalmente representado por um vetor cuja dimensão é compatível com a dimensão do cromossomo. Em (BÄCK, 1996), um conjunto de $\mu \in \mathbb{N}$ indivíduos é denotado por I^μ e chamado de população. A transformação ocorrida no espaço de indivíduos (ou população) é denotada como: $T : I^\mu \to I^\mu$. A esta transformação T dá-se o nome de geração. Existem AEs cujas populações variam de tamanho na passagem de uma geração para outra. Para generalizar a transformação no espaço de indivíduos foi introduzida a definição: $T : I^\mu \to I^{\mu'}$, indicando que populações sucessoras podem variar de tamanho. De posse desta terminologia é possível definir um AE como (MERKLE; LAMONT, 1997):

Definição 10 (Algoritmo Evolucionário). *Seja I um conjunto não vazio chamado de espaço de indivíduos, $\mu^{(t)} \in \mathbb{N}$ com $t \in \mathbb{Z}^+$ chamado de tamanho da população de progenitores, $\mu'^{(t)} \in \mathbb{N}$ com $t \in \mathbb{Z}^+$ chamado de tamanho da população filha, $\Phi : I \to \mathbb{R}$ uma função de aptidão, $\iota : \cup_{i=1}^{\infty}(I^\mu)^i \to \{true, false\}$ o critério de parada, $\chi \in \{true, false\}$ o critério de seleção, r uma sequência de operadores de recombinação $r^{(t)}$, m uma sequência de operadores de mutação $m^{(t)}$, s uma sequência de operadores de seleção $s^{(t)}$, $\Theta_r^{(t)}$, $\Theta_m^{(t)}$ e $\Theta_s^{(t)}$, parâmetros de recombinação, mutação e seleção, respectivamente. O Algoritmo 1 é dito um algoritmo evolucionário.*

Algoritmo 1 Algoritmo Evolucionário

1: $t := 0$;
2: inicializa: $P(0) := \{\mathtt{i}_1(0), \mathtt{i}_2(0) \ldots, \mathtt{i}_\mu(0)\} \in I^{\mu^{(0)}}$;
3: **Enquanto** $(\iota(\{P(0), \ldots, P(t)\}) \neq true)$ **Faça**
4: recombina: $P'(t) := r^{(t)}_{\Theta_r^{(t)}}(P(t))$;
5: muta: $P''(t) := m^{(t)}_{\Theta_m^{(t)}}(P'(t))$;
6: seleciona:
7: **Se** χ **Então**
8: $P(t+1) := s^{(t)}_{(\Theta_s^{(t)}, \Phi)}(P''(t))$;
9: **Senão**
10: $P(t+1) := s^{(t)}_{(\Theta_s^{(t)}, \Phi)}(P''(t) \cup P(t))$;
11: **Fim Se**
12: $t := t + 1$;
13: **Fim Enquanto**

No Algoritmo 1, pode-se observar que no começo de tudo ($t := 0$), um AE é dotado de uma população inicial, $P(0)$, que pertence ao espaço de indivíduos inicial, $I^{\mu^{(0)}}$. Enquanto um critério de parada não for satisfeito, os operadores de recombinação, mutação e seleção são executados a cada geração t. A seleção pode ser feita considerando-se apenas o conjunto dos novos indivíduos ($P''(t)$) ou considerando os conjuntos dos novos indivíduos e seus progenitores ($P''(t) \cup P(t)$).

3.1.2 Operadores genéticos

Assim como ocorre na natureza, os operadores genéticos de um AE tem o objetivo de gerar indivíduos (soluções) cada vez mais aptos. Os três principais operadores genéticos utilizados em AEs são: *seleção*, *recombinação* (ou *reprodução*) e *mutação*. A seleção faz com que indivíduos mais aptos tenham maior chance de sobrevivência e apareçam na geração seguinte. A recombinação permite a troca do material genéticos entre os indivíduos selecionados pelo operador seleção. Espera-se que a troca de material genético entre indivíduos aptos gere indivíduos tão ou ainda mais aptos que seus progenitores. A mutação permite aumentar a diversidade da população fazendo com que o genótipo dos indivíduos gerados pelo operador recombinação, se diferencie levemente do genótipo de seus progenitores, possibilitando assim o surgimento de um indivíduo melhor e completamente diferente do restante da população. Existem diferentes técnicas para implementação desses operadores genéticos. Em seguida, serão introduzidas algumas dessas implementações.

A Figura 21 ilustra o método de seleção através do giro da roleta. Trata-se de um método simples de seleção que simula um jogo de roleta com divisões que podem ser irregulares. A probabilidade de seleção está diretamente relacionada com a proporção entre a aptidão de cada indivíduo e a aptidão média da população. Indivíduos mais aptos recebem mais divisões da roleta enquanto que indivíduos menos aptos recebem menos divisões. É importante ressaltar que os indivíduos mais aptos têm maior probabilidade de serem selecionados, o que não significa que serão sempre selecionados.

A Figura 22 mostra uma técnica simples de recombinação, onde um locus é selecionado para dois indivíduos progenitores e a partir deste locus, o material genético dos dois é trocado, formando então dois novos indivíduos filhos. A Figura 23 mostra o efeito do operador mutação que troca aleatoriamente o alelo de um ou mais genes. No caso de representação binária, o alelo é trocado de 0 para 1 e vice-versa. Existem diferentes operadores evolucionários tanto para representação binária quanto para representação real, cada um com suas características

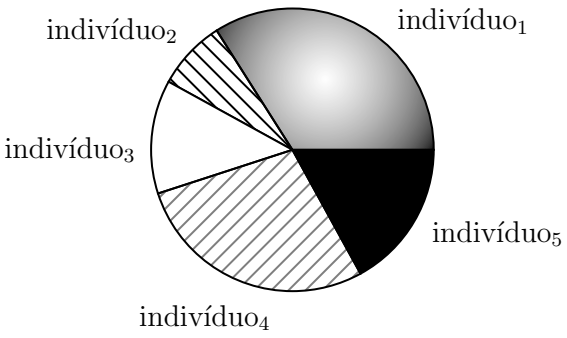


Figura 21: Seleção pelo giro da roleta

(BÄCK, 1996). Uma análise profunda sobre os principais operadores evolucionários está alem do escopo desta dissertação e pode ser encontrada em (BÄCK, 1996).

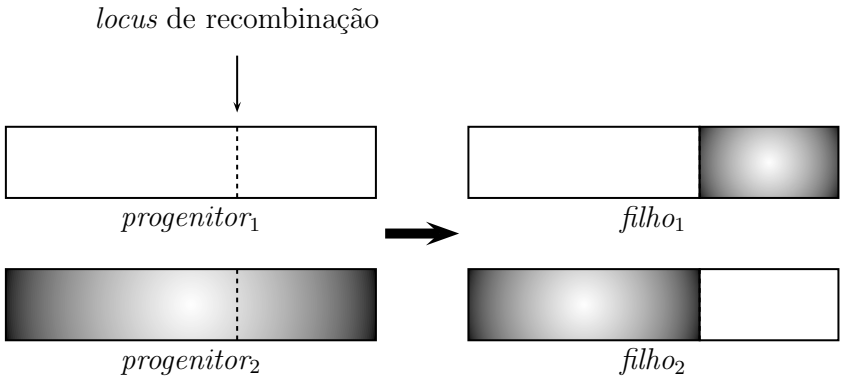


Figura 22: Recombinação em um ponto

Embora seja necessário muita criatividade na escolha da representação da solução do problema através de um cromossomo, algumas considerações importantes devem ser levadas em conta, como, por exemplo, a precisão utilizada na codificação. A escolha dos operadores evolucionários também deve ser analisada com cuidado para que seja escolhido o operador que mais se adeque à natureza do problema. Representações e operadores impróprios podem afetar significativamente a eficiência e até mesmo a eficácia de um AE (BÄCK, 1996). Mesmo não havendo uma combinação garantida entre representação e operadores, uma escolha criteriosa resulta em implementações mais eficientes e eficazes (FOGEL; GHOZEIL, 1997).

3.2 Algoritmos Evolucionários Multiobjetivo

Algoritmos evolucionários multiobjetivo (AEMs) são voltados para solucionar POMs levando em consideração o espaço de busca do problema e as restrições impostas pelo mesmo. Muitos

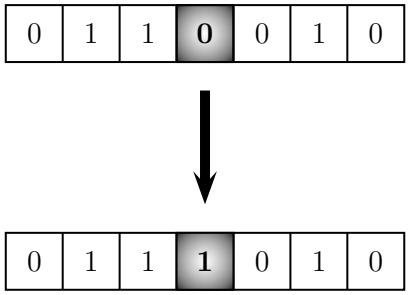


Figura 23: Mutação binária

dos métodos de otimização vistos na Seção 2.2, do Capítulo 2, foram desenvolvidos para busca em espaços extremamente grandes mas as técnicas tradicionais utilizadas para solução de POMs se limitavam à um espaço de busca restrito (HORN, 1997). Alguns métodos tradicionais se concentram na busca em si, enquanto que outros são centralizados na seleção das soluções encontradas. A utilização de AEMs se torna interessante para solucionar POMs porque tratam da busca e da seleção de soluções. O

A característica que permite a otimização simultânea de diferentes objetivos, pode ser utilizada para diferenciar um AEM de um AE e também para formular uma definição para um AEM. Decompondo um AEM e um AE em etapas, pouca diferença existe na sequência das mesmas que cada um realiza para simular o processo de evolução natural. A Definição 11, a Figura 24 e a Figura 25, demonstram a relação existente entre ambos os algoritmos.

Definição 11 (Algoritmo Evolucionário Multiobjetivo). *Seja* $\Phi : I \rightarrow \mathbb{R}^k \mid k \geq 2$*, a função de aptidão de um POM. Se esta função de aptidão for substituída pela função de aptidão do Algoritmo 1, então o AE de um único objetivo* ($k = 1$) *passa a ser um AEM* ($k \geq 2$)*.*

A Figura 24 e Figura 25 mostram as etapas de um AE e um AEM, respectivamente. A única diferença está na etapa de avaliação, onde um AEM precisa calcular k ($k \geq 2$) funções objetivo para determinar a aptidão de cada indivíduo. Alguns AEMs agregam os k objetivos encontrados para cada indivíduo em um único escalar através de uma avaliação combinada. Existem diferentes técnicas para agregar os valores dos objetivos de um indivíduo em um único valor e para estes algoritmos, esta etapa representa um custo computacional adicional. Os AEMs que não agregam os valores de aptidão, não possuem a etapa de avaliação combinada. Embora a estrutura de um AE e um AEM seja praticamente a mesma, isso não significa que as diferenças sejam insignificantes. Na etapa de avaliação é necessário realizar ordenação parcial no espaço de objetivos, tarefa esta, de alto custo computacional como visto no Capítulo 2.

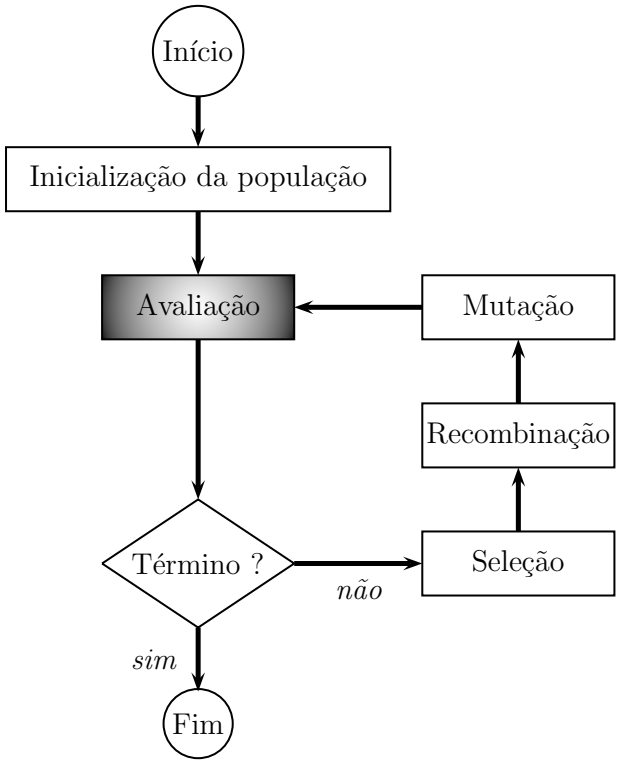


Figura 24: Etapas de um AE

Como visto anteriormente, um algoritmo genético (AG) é um tipo de AE (VELDHUIZEN, 1999). Analogamente, existem AEMs que são classificados como algoritmos genéticos multiobjetivo (AGM). O fator em comum entre estes algoritmos é o caráter evolucionário, que é baseado na sobrevivência do indivíduo mais apto. A característica particular dos AGs e dos AGMs é a representação dos indivíduos na forma de um cromossomo combinada com a utilização de operadores de recombinação e mutação.

Os principais AEMs podem ser classificados em três categorias, dependendo do método utilizado para selecionar e classificar os indivíduos durante o processo de otimização (HORN, 1997). A Figura 26 mostra as categorias e os principais algoritmos que fazem parte destas. Em seguida, serão detalhadas as características de cada uma das três categorias de AEM.

- *Seleção por agregação*: Os métodos classificados nesta categoria agregam as k funções objetivo de um POM em uma única função objetivo para determinar a aptidão do indivíduo. Esta técnica transforma um POM em um problema de otimização de um único objetivo. A dificuldade na utilização desta técnica está na determinação de uma função de agregação que abranja todas as características do problema adequadamente.

- *Seleção por critério*: Os métodos classificados nesta categoria determinam a aptidão de cada indivíduo avaliando cada função objetivo separadamente. Em um problema com k

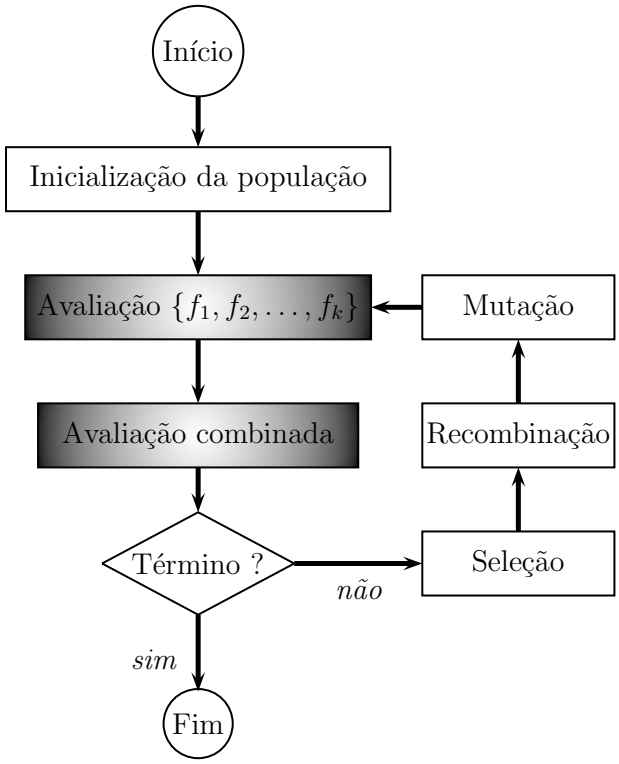


Figura 25: Etapas de um AEM

objetivos, cada indivíduo carregará consigo k valores de aptidão, um para cada objetivo. Esta técnica difere de um AG simples na quantidade de avaliações necessárias por solução. Sua maior limitação é que na maioria dos casos as soluções encontradas pertencem a um mínimo local e não à fronteira ótima de Pareto.

- *Seleção por dominância*: Os métodos classificados nesta categoria determinam a aptidão dos indivíduos usando o conceito de dominância apresentado na Seção 2.1.3 (Capítulo 2). Ao contrário da técnica de seleção por critério, na seleção por dominância os objetivos não são analisados isoladamente formando diferentes valores de aptidão. As soluções não dominadas da população são consideradas as mais aptas da população. Otimizações realizadas usando estas técnicas apresentam uma convergência rápida para a fronteira ótima de Pareto.

3.2.1 Métodos baseados na seleção por agregação

A principal característica destes métodos é a de transformar um POM em um problema de otimização de um único objetivo através da agregação dos objetivos. Em seguida, serão apresentados três métodos diferentes de agregação de objetivos.

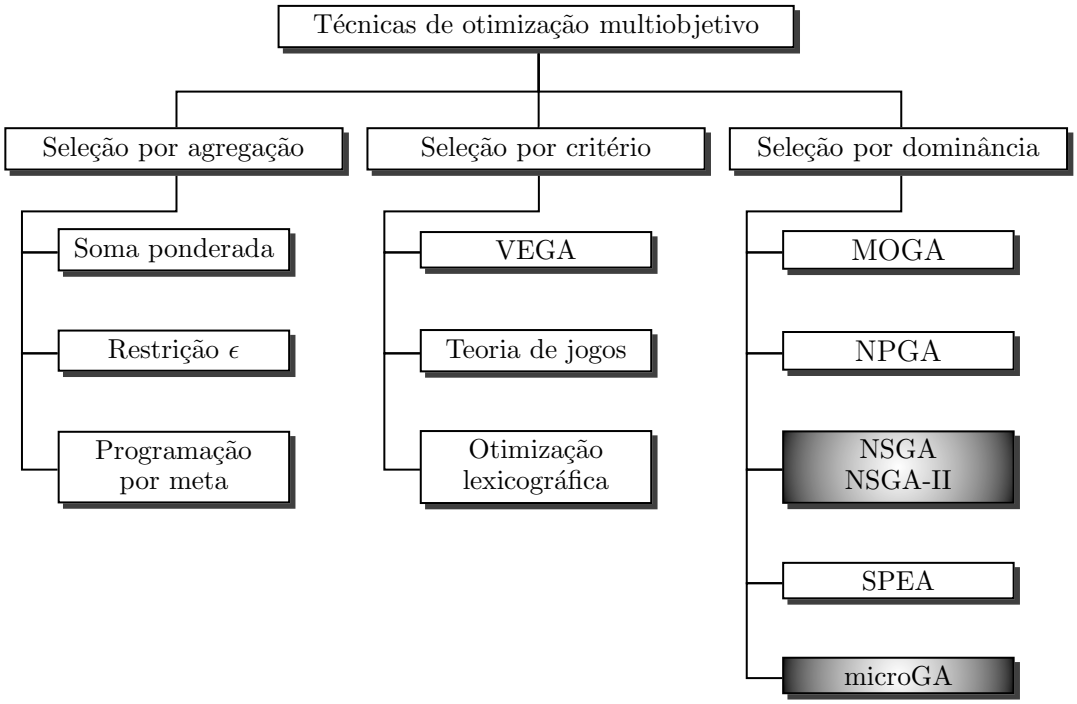


Figura 26: Classificação das técnicas de otimização multiobjetivo

3.2.1.1 Soma ponderada

Este é o método de resolução de POM mais usado. Este método agrega linearmente os objetivos em uma única função objetivo. A agregação é feita utilizando *pesos* que são atribuídos para cada objetivo. Um POM com k objetivos, $f_1, \ldots, f_k$, é transformado em um problema de otimização com único objetivo utilizando a Equação 7.

$$\min_{\mathbf{x}\in\Omega} \mathtt{f}(\mathtt{x}) = \sum_{i=1}^{k} w_i f_i(\mathtt{x}) \tag{7}$$

Os pesos w_i, também chamados de *fatores de importância*, são números reais não nulos e escolhidos de modo que $w_1 + w_2 + \ldots + w_k = 1$. Cada peso w_i representa a importância do objetivo f_i na classificação de cada solução.

Como exemplo, a soma ponderada, representada por Z, de um problema de otimização com dois objetivos, é dada por:

$$Z = w_1 f_1(\mathtt{x}) + w_2 f_2(\mathtt{x}) \tag{8}$$

A minimização da soma ponderada pode ser interpretada como achar o valor de Z que faz com que uma reta entre f_1 e f_2, com inclinação $-\frac{w_1}{w_2}$, tangencie o espaço de objetivos Λ, conforme mostra a Equação 9.

$$(8) \Rightarrow f_2(\mathtt{x}) = -\frac{w_1}{w_2} f_1(\mathtt{x}) + \frac{Z}{w_2} \tag{9}$$

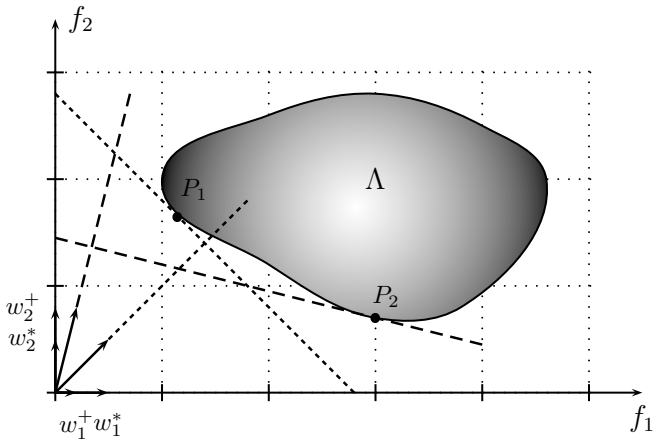


Figura 27: Desempenho do método da soma ponderada com dois objetivos

A Figura 27 ilustra duas retas, correspondentes a duas duplas distintas (w_1, w_2), tangenciando o espaço de objetivos Λ. Os pontos P_1 e P_2, nas tangentes, fazem parte do conjunto ótimo de Pareto do problema. Variando os pesos em um método de tentativa e erro é possível determinar outras soluções ótimas de Pareto e a forma da fronteira ótima de Pareto. Para um problema com dois objetivos esta tarefa apresenta um baixo custo computacional. Conforme o número de objetivos aumenta, a complexidade em achar a fronteira ótima de Pareto aumenta exponencialmente.

Uma grande vantagem deste método é a sua simplicidade na implementação. Uma desvantagem é a determinação dos pesos, que deve ser feita por um especialista do problema que possa avaliar a importância de cada objetivo ou através de um processo interativo na busca da melhor solução. No entanto, esta busca cria um novo problema de otimização a ser resolvido. Obter uma solução única para cada conjunto de pesos também é uma desvantagem.

A maior limitação da soma ponderada é que a fronteira ótima de Pareto deve ser convexa. Independente dos valores dos pesos, este método não é capaz de encontrar soluções em regiões côncavas da fronteira. A Figura 28 ilustra um problema com dois objetivos com uma acentuada região côncava. Mesmo variando as duplas de pesos não é possível traçar uma reta entre f_1 e f_2 que seja tangente ao espaço de objetivos Λ na região côncava da fronteira ótima de Pareto.

3.2.1.2 Restrição ε

Como visto na seção anterior, o método de soma ponderada não é capaz de identificar soluções Pareto-ótimas em regiões côncavas da fronteira ótima de Pareto. O método de *restrição* ε (RITZEL; EHEART; RANJITHAN, 1994) pode ser utilizado para identificar tais soluções. Este

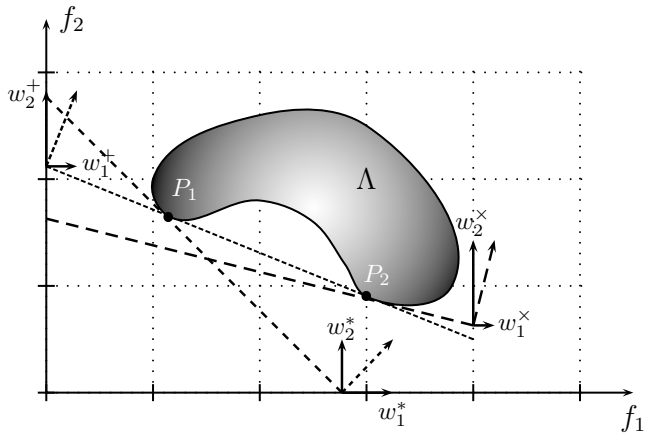


Figura 28: Ilustração do desempenho do método da soma ponderada quando a fronteira ótimas de Pareto é côncava

método transforma um POM de k objetivos e m restrições em um problema de otimização de um único objetivo sujeito a $m+k-1$ restrições. Esta transformação de objetivos em restrições é definida pela Equação 10.

$$
\begin{aligned}
&\min_{\mathtt{x}\in\Omega} && \mathtt{f}(\mathtt{x}),\\
&\text{sujeito à} && g_i(\mathtt{x}) \leq 0, \quad 1 \leq i \leq m\\
&&& \Downarrow\\
&\min_{\mathtt{x}\in\Omega} && f_h(\mathtt{x}),\\
&\text{sujeito à} && \begin{cases} g_i(\mathtt{x}) \leq 0, & 1 \leq i \leq m \\ f_j(\mathtt{x}) \leq \varepsilon_i, & 1 \leq j \leq k-1 | j \neq h \end{cases}
\end{aligned}
\tag{10}
$$

Para um POM com dois objetivos ($k = 2$) e uma fronteira ótima de Pareto com região côncava, o funcionamento do método restrição ε é demonstrado na Figura 29. Um valor ε é atribuído a um dos objetivos através de restrições enquanto que o outro objetivo é minimizado. O ponto P_3 pode ser encontrado fixando f_1 no intervalo $[\varepsilon_1, \varepsilon_1']$ e minimizando f_2. Variando as restrições e minimizando apenas um objetivo enquanto, que os outros objetivos estão fixados através das restrições no espaço de objetivos, é possível determinar a fronteira de Pareto-ótima até nos casos que esta apresenta regiões côncavas.

3.2.1.3 Programação por meta

Uma das maneiras de introduzir as preferencias do especialista é atribuir um valor alvo e pesos ou *prioridades* para cada objetivo. O método de soma ponderada atribui um fator de importância para cada objetivo enquanto que a *programação por meta* (CHARNES; COOPER,

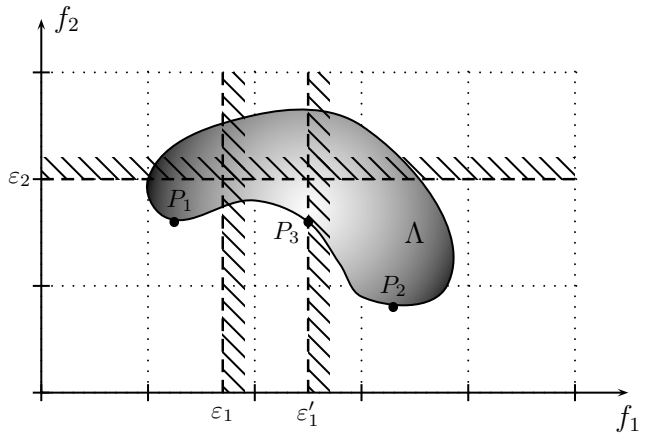


Figura 29: Ilustração do desempenho do método de restrição ε quando a fronteira ótimas de Pareto possui região côncava

1967) (IJIRI, 1965) atribui um valor alvo para cada objetivo. Estes valores são chamados de *níveis de aspiração*. O ponto correspondeste aos *níveis de aspiração* no espaço de objetivos é chamado de *meta*. A especificação dos *níveis de aspiração* não é uma tarefa fácil e requer um certo conhecimento do POM.

Para entender a essência do método de programação por meta, assume-se um problema de minimização de k objetivos $f_1, f_2, \ldots, f_k$ sujeito a m restrições. Seja, $\hat{\mathtt{f}}$ a *meta* com *níveis de aspiração* $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_k$. Então, o POM é transformado em um problema de otimização de um único objetivo cuja solução é obtida minimizando a distância entre o objetivo $\mathtt{f}$ e a *meta* $\hat{\mathtt{f}}$. Esta transformação é feita através da Equação 11, onde w_i é o peso que determina a importância do objetivo f_i no processo de otimização e δ é uma variável que armazena a menor distância encontrada entre $\mathtt{f}$ e $\hat{\mathtt{f}}$.

Assume-se que a melhor solução é aquela que estiver mais próxima da meta determinada. Assim como no método de *soma ponderada*, são necessários vários processos de otimização, variando as metas para obter as soluções da fronteira ótima de Pareto. O bom desempenho deste algoritmo para fronteiras de Pareto côncavas e convexas depende da escolha das metas (YANG, 2000).

$$
\begin{array}{ll}
\min\limits_{\mathtt{x}\in\Omega} & \mathtt{f}(\mathtt{x}), \\
\text{sujeito à} & g_i(\mathtt{x}) \leq 0, \quad 1 \leq i \leq m \\
& \Downarrow \\
\min & \delta, \\
\text{sujeito à} & \begin{cases} g_i(\mathtt{x}) \leq 0, & 1 \leq i \leq m \\ w_j|\hat{f}_j(\mathtt{x}) - f_j(\mathtt{x})| \leq \delta, & 1 \leq j \leq k \end{cases}
\end{array}
\tag{11}
$$

3.2.2 Métodos baseados na seleção por critério

A principal característica desta técnica de seleção e classificação de soluções é considerar cada objetivo individualmente. Nesta seção encontram-se três métodos baseados na *seleção por critério*.

3.2.2.1 VEGA – *Vector Evaluated Genetic Algorithm*

Este método, desenvolvido por David Schaffer (SCHAFFER, 1985), é uma extensão do programa GENESIS, de Grefenstette (GREFENSTETTE, 1984), que é baseado em um algoritmo genético padrão, seguindo as etapas da Figura 24. A diferença do VEGA para um algoritmo genético padrão é o método de seleção. Este operador genético foi modificado de modo que a cada geração um objetivo é avaliado e uma sub-população é gerada de acordo com o objetivo em questão. Por exemplo, dado um POM com k objetivos e uma população de N indivíduos, sendo N múltiplo de k, k sub-populações de tamanho N/k serão geradas. A seleção realizada para cada sub-população é referente ao objetivo relacionado. Deste modo, é possível que um mesmo indivíduo seja selecionado mais de uma vez se ele for o melhor em mais de um objetivo. Após a criação das k sub-populações, os N indivíduos selecionados são agrupados em uma ordem aleatória e as operações de recombinação e mutação são realizadas para formar os indivíduos da geração seguinte.

Schaffer observou que as soluções obtidas não eram dominadas localmente por estarem associadas a um único objetivo. É possível afirmar que uma solução dominada localmente é também dominada globalmente mas o contrário não pode ser afirmado (SCHAFFER, 1985), isto é, uma solução não dominada localmente, pode ser dominada globalmente. Em geral, as soluções obtidas com o VEGA não são dominadas localmente, logo, não são necessariamente soluções ótimas de Pareto. Como este método seleciona indivíduos a partir da análise de apenas um dos objetivos do POM, este gera um problema que em genética é chamado de

especiação. Os indivíduos selecionados são bons mas não são os melhores considerando-se todos os objetivos. Para reduzir o problema de *especiação*, Schaffer propôs duas heurísticas: a primeira diz respeito à seleção, onde indivíduos dominados são penalizados no intuito de serem menos selecionados; a segunda diz respeito à recombinação, onde, ao invés de utilizar um método aleatório de recombinação, um método mais criterioso seria utilizado entre indivíduos de diferentes sub-populações.

3.2.2.2 Otimização lexicográfica

Nesse método, os objetivos são classificados por ordem de importância. Os objetivos mais importantes devem ser otimizados antes dos objetivos menos importantes. Considerando um POM de k objetivos, o objetivo mais importante é dito ser um objetivo de *prioridade* 1 enquanto que o objetivo menos importante é dito ser um objetivo de *prioridade* k. A solução ótima é obtida minimizando-se cada função objetivo na ordem escolhida. A importância dos objetivos é definida *a priori* pelo especialista. Uma solução é *lexicograficamente* melhor do que outra de acordo com a Definição 12.

Definição 12. *Sejam* $\mathtt{x}_1 \in \Omega$ *e* $\mathtt{x}_2 \in \Omega$*, dois vetores de decisão de um POM com* k *objetivos* $f_1, f_2, \ldots, f_k$*. Seja* i *a prioridade do objetivo* f_i*. A solução* $\mathtt{x}_1$ *é dita lexicograficamente melhor do que a solução* $\mathtt{x}_2$ *se e somente se* $f_i(\mathtt{x}_1) < f_i(\mathtt{x}_2)$*, sendo* i *o menor índice em* $\{1, 2, \ldots, k\}$ *tal que* $f_i(\mathtt{x}_1) \neq f_i(\mathtt{x}_2)$*.*

A *otimização lexicográfica* funciona como uma ordenação alfabética: As primeiras letras de cada palavra são comparadas e em caso de empate, compara-se a letra seguinte e assim por diante. Se duas soluções, $\mathtt{x}_1$ e $\mathtt{x}_2$, forem iguais para todos os objetivos $f_1, f_2, \ldots, f_k$, então estas soluções não podem ser classificadas lexicograficamente. Em uma população de tamanho N, cada solução pode receber um *nível* de acordo com a posição ocupada após a otimização lexicográfica. Soluções equivalentes são associadas ao mesmo *nível*. A melhor solução recebe *nível* 1 e a pior solução recebe *nível* N no caso de haver apenas uma solução por *nível*. Com cada solução associada a um *nível* é possível utilizar o método da roleta de seleção, sendo que, quanto menor for o *nível* maior será a chance de seleção.

3.2.2.3 Teoria dos jogos – Equilíbrio de Nash e Jogo Não Cooperativo

Este método é inspirado em duas teorias aplicadas na área matemática chamada de *teoria dos jogos* (NEUMANN; MORGENSTERN, 1944). As teorias, *Equilíbrio de Nash* (NASH, 1950) e *Jogos Não Cooperativos* (NASH, 1951), foram desenvolvidas por John Nash que em 1994 ganhou o

prêmio Nobel em economia devido à sua contribuição na área de Teoria dos Jogos aplicada à economia.

O processo de otimização multiobjetivo baseado nas teorias de Nash é dito *não cooperativo* já que cada objetivo é otimizado separadamente. A ideia básica consiste em associar um jogador a um objetivo e cada jogador se encarrega de otimizar o objetivo correspondente. O objetivo do jogo é alcançar o dito *Equilíbrio de Nash*, que ocorre quando nenhum jogador é capaz de otimizar o seu respectivo objetivo.

Considere um POM com k objetivos $f_1, f_2, \ldots, f_k$. O AGM baseado nas teorias de Nash atribui a otimização de cada objetivo f_i para um $jogador_i$ e cada jogador lida com a sua própria população. O processo evolucionário é ilustrado na Figura 30. Trata-se de um AG paralelo (DORIGO; MANIEZZO, 1993) com múltiplos objetivos a serem otimizados. A cada geração t quando um $jogador_i$ completa o processo evolucionário com sua população, ele envia as melhores soluções f_i^t encontradas para todos os outros jogadores $jogador_j$, $j \in \{1, \ldots, k\} \setminus \{i\}$. Na geração seguinte, cada jogador irá otimizar as soluções recebidas dos demais jogadores e assim espera-se que todos os objetivos sejam otimizados paralelamente. O processo evolucionário termina quando o *equilíbrio de Nash* é alcançado.

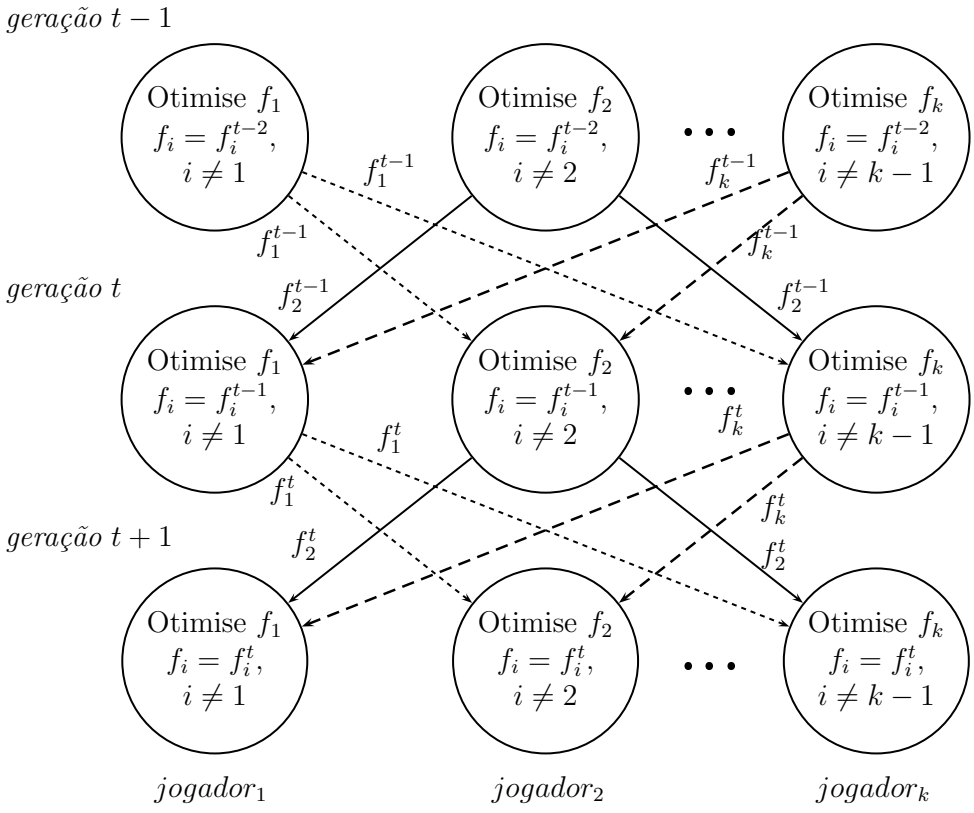


Figura 30: Ilustração da otimização multiobjetivo baseada nas teorias de Nash

3.2.3 Métodos baseados na seleção por dominância

Métodos baseados na *seleção por dominância* (ou *seleção de Pareto*) são os métodos de otimização multiobjetivo mais usados (VELDHUIZE; LAMONT, 1998). São capazes de encontrar um conjunto significativo de soluções ótimas de Pareto com relativa eficiência, independente da forma da fronteira ótima de Pareto. Em geral, os métodos que utilizam esta técnica, determinam a aptidão de cada indivíduo da população através da relação de dominância definida por Pareto (PARETO, 1896) e apresentada na seção 2.1.3 no Capítulo 1.

Nesta seção, são apresentados seis AGMs usados na otimização de problemas com múltiplos objetivos. Todos utilizam o conceito de dominância para a classificação e seleção de soluções. Suas maiores diferenças estão nos métodos de busca utilizados, na divisão da população e nos operadores genéticos empregados.

3.2.3.1 MOGA – *Multi-Objective Genetic Algorithm*

O primeiro AG a lidar com POM e utilizar o conceito de dominância de Pareto para seleção e classificação de soluções foi proposto por Goldberg (GOLDBERG, 1989a). Este algoritmo atribui *nível* 1 para todas as soluções não dominadas da população principal P de N indivíduos; em seguida, os indivíduo que receberam *nível* 1 são movidos para uma população auxiliar P' e os indivíduos não dominados considerando aqueles que restaram na população principal ($P \backslash P'$) recebem *nível* 2. Este processo é repetido até que a população principal fique vazia, o que significa que todos os indivíduos já possuem um *nível*. Este procedimento é descrito no Algoritmo 2, onde S_i representa uma solução i, $1 \leq i \leq N$, e $R(.)$ é o método que atribui um *nível* à uma dada solução.

Algoritmo 2 Algoritmo de Nível de Goldberg

```
1: nível := 1;
2: P' := 0;
3: Enquanto P ≠ 0 Faça
4:    Para cada {S_i, S_j ∈ P | j ≠ i ∧ ∄S_j ≺ S_i} Faça
5:       R[S_i] := nível;
6:       P' := P' ∪ {S_i};
7:    Fim Para
8:    P := P \ P';
9:    nível := nível + 1;
10: Fim Enquanto
11: Retorna R
```

O algoritmo genético de múltiplos objetivos ou simplesmente MOGA foi desenvolvido por Fonseca e Fleming (FONSECA; FLEMING, 1993). O processo de classificação do MOGA

se assemelha ao algoritmo proposto por em (GOLDBERG, 1989a), sendo que o *nível* de cada indivíduo depende do número de indivíduos que o dominam. Inicialmente, as soluções não dominadas recebem *nível* 1, assim como no Algoritmo 2. Em seguida, as soluções não dominadas no conjunto de soluções que não possuem *nível*, recebem um *nível* de acordo com a quantidade de soluções, com *nível*, que às dominam. O *nível* do indivíduo S_i é calculado conforme Equação 12 (FONSECA; FLEMING, 1993):

$$\mathcal{N}(S_i, t) = 1 + p_i^{(t)}, \tag{12}$$

onde, t é a geração atual e $p_i^{(t)}$ é a quantidade de indivíduos que dominam a solução S_i.

A Figura 31 ilustra o processo de classificação usado pelo MOGA para um POM com dois objetivos. Neste processo os indivíduos não dominados continuam recebendo o *nível* máximo (*nível* 1) enquanto que os demais indivíduos são penalizados de acordo com a região que ocupam no espaço de objetivos. As soluções marcadas com *nível* 2 e 3 receberiam *nível* 2, mas é possível observar que a solução de *nível* 3, está mais afastada da fronteira formada pelas soluções de *nível* 1 e é dominada por duas soluções, enquanto que a solução de *nível* 2 é dominada por apenas uma solução e está mais próxima da fronteira ótima de Pareto. A solução de *nível* 5 é dominada por quatro soluções, sendo uma de *nível* 2 e três de *nível* 1. Note que nem todos os *nível* serão representados em todas as gerações, como é o caso do *nível* 4 que não aparece na Figura 31.

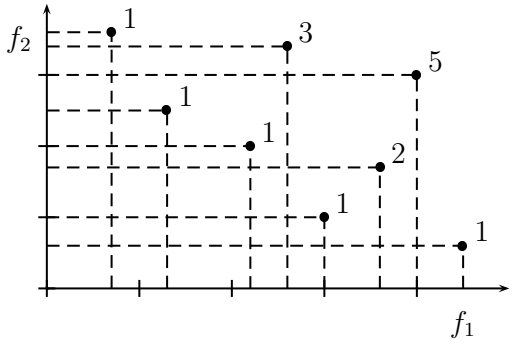


Figura 31: Processo de classificação do MOGA

A aptidão de um indivíduo não coincide com o *nível* correspondente. Esta aptidão é calculada da seguinte maneira:

- Ordenar os indivíduos da população de acordo com seus respectivos *níveis*.
- Calcular a aptidão de cada indivíduo usando uma função de interpolação, geralmente linear, assumindo que o melhor indivíduo é o de *nível* 1 e o pior é o de nível $n \leq N$.

- Calcular a média das aptidões dos indivíduos de mesmo *nível*, permitindo assim que todos estes tenham a mesma chance de serem selecionados.

Os indivíduos de mesmo *nível* podem causar grande pressão no AG fazendo com que o mesmo convirja prematuramente, isto é, a diversidade seja mínima antes que o conjunto ótimo de Pareto seja encontrado. Para remediar esta limitação, MOGA utiliza o método de formação de nichos através da *aptidão compartilhada* para manter a diversidade da população (FONSECA; FLEMING, 1993). Uma desvantagem deste método é a introdução de um novo parâmetro que determina o tamanho do nicho e tem grande influência na performance do algoritmo.

3.2.3.2 NPGA – *Niched Pareto Genetic Algorithms*

Assim como o MOGA, o algoritmo NPGA (HORN; NAFPLIOTIS; GOLDBERG, 1994) é baseado no conceito de dominância de Pareto e utiliza a técnica de formação de nichos através de *aptidão compartilhada* para manter a diversidade da população. Uma grande diferença entre o NPGA e o MOGA é o método de seleção utilizado. O NPGA utiliza a *seleção por torneio*. Enquanto que no método de seleção tradicional um indivíduo é escolhido de acordo com a probabilidade relacionada entre sua aptidão e a aptidão média da população, na *seleção por torneio*, um conjunto de indivíduos é escolhido aleatoriamente e os indivíduos disputam entre si como se fosse um torneiro, sendo que o indivíduo vencedor será selecionado. O conjunto de soluções é mais conhecido como *pool*. A *seleção por torneio* implementada no NPGA é ligeiramente diferente e funciona da seguinte forma: Dois indivíduos, candidatos à seleção, são escolhidos aleatoriamente e um conjunto de indivíduos para comparação também é escolhido aleatoriamente, formando o *pool*. Se um dos candidatos for dominado por algum indivíduo do *pool* e o outro não for, então o último será selecionado para reprodução. Se ambos forem dominados ou não, ocorre empate entre os candidatos. Neste caso, é utilizada a *contagem de nicho* que calcula a distância do indivíduo em relação a outros indivíduos do nicho. O indivíduo que obtiver a menor *contagem de nicho* vence (HORN; NAFPLIOTIS; GOLDBERG, 1994). O Algoritmo 3 descreve os detalhes deste processo de seleção, onde η representa a função de contagem de nicho.

O tamanho do *pool* tem grande influencia na convergência do NPGA (HORN; NAFPLIOTIS; GOLDBERG, 1994). Inicialmente, é comum definir este parâmetro em torno de 10% do tamanho da população do momento. Caso necessário, este valor deve ser ajustado para melhorar a convergência. Com um *pool* de tamanho 1, a *seleção por torneio* se torna um método

Algoritmo 3 Torneio por seleção do NPGA

Entrada. População P, pool de torneio T e fronteira de Pareto local $PF \in P$
Saída. Indivíduo selecionado SI

```
1: Selecionar aleatoriamente T indivíduos de P para formar o pool de torneio PT;
2: Selecionar aleatoriamente, de P, dois candidatos C1 e C2 para o torneio;
3: D_C1 := false;     /* C1 não é dominado */
4: D_C2 := false;     /* C2 não é dominado */
5: Para cada I ∈ PT Faça
6:    Se I ≺ C1 Então
7:       D_C1 := true;     /* C1 é dominado */
8:    Fim Se
9:    Se I ≺ C2 Então
10:      D_C2 := true;     /* C2 é dominado */
11:   Fim Se
12: Fim Para
13: Se D_C1 ∧ ¬D_C2 Então
14:   SI := C2;
15: Senão Se ¬D_C1 ∧ D_C2 Então
16:   SI := C1;
17: Senão Se η_C1(PF) < η_C2(PF) Então
18:   SI := C1;
19: Senão
20:   SI := C2;
21: Fim Se
22: Retorna SI;
```

de *seleção aleatória*. Com um *pool* de tamanho 2, temos um método de *seleção por torneio binário*.

O NPGA é mais eficiente do que o MOGA por não atribuir um *nível* para todos os indivíduos de uma só vez, mas apenas para uma parte da população que participa da *seleção por torneio* (COELLO, 2001). Outra vantagem deste algoritmo é que este produz fronteiras de Pareto com boa diversidade (COELLO, 1996). Como desvantagem destaca-se a necessidade de configurar o parâmetro de *aptidão compartilhada* para determinar o tamanho dos nichos e o parâmetro de tamanho do *pool* para a *seleção por torneio*.

3.2.3.3 SPEA – *Strength Pareto Evolutionary Algorithms*

O SPEA (ZITZLER; THIELE, 1999) é um algoritmo que utiliza o conceito de dominância não só para seleção e classificação de soluções mas também para manter a diversidade da população. Foi desenvolvido a partir de técnicas já consolidadas e utilizadas em outros AEs misturadas com técnicas inéditas na época. Assim como alguns de seus predecessores, o SPEA possui uma população externa para armazenar soluções não dominadas, a aptidão de cada indivíduo

é proporcional ao número de indivíduos que o dominam e utiliza a noção de nichos para manter a diversidade da população. O SPEA introduziu alguns aspectos inovadores, tais como:

- A combinação das três técnicas citadas anteriormente em um único algoritmo;
- A aptidão da população é baseada unicamente nos indivíduos da população externa.
- Participação de todos os indivíduos da população externa no processo de seleção.
- Novo método de formação de nichos baseado na dominância, sem a necessidade de um parâmetro de tamanho do nicho.

O Algoritmo 4 descreve detalhadamente o funcionamento do SPEA. A população externa, P', é mantida atualizada a cada geração com as soluções não dominadas. Quando a capacidade da população externa excede seu limite, um processo de poda através de *clusterização* é iniciado para reduzir o tamanho da mesma.

Algoritmo 4 SPEA

Entrada. Tamanho do pool de torneio $Tpool$, tamanho da população externa Tp' e número de gerações Ng
Saída. População externa P'

1: $Inicializa(P)$;
2: $geração := 1$;
3: $P' := \emptyset$;
4: **Enquanto** $geração \leq Ng$ **Faça**
5: $P' := \{s | \nexists r \in P,\ r \prec s\}$;
6: $P' := P' \setminus \{s | \exists r \in P',\ r \prec s\}$;
7: **Se** $\#P' > Tp'$ **Então**
8: $Poda(P')$;
9: **Fim Se**
10: $Aptidão(P, P')$;
11: $Mutação(Recombinação(Seleção(P, P', Tpool)))$;
12: $geração := geração + 1$;
13: **Fim Enquanto**
14: **Retorna** P';

No Algoritmo 4, a função $Inicializa(P)$ cria uma população inicial aleatória P. A cada geração as soluções não dominadas em P são copiadas para P' e as soluções dominadas em P' são eliminadas. Se o tamanho de P' exceder Tp', então P' é reduzida através do método $Poda(.)$. Este método realiza uma análise de *clusters* em P'. *Clusters* são regiões que podem ser comparadas a nichos, sendo que o tamanho dos *clusters* é dinâmico e não estático como o tamanho dos nichos.

O Algoritmo 5 descreve o procedimento de poda através de *clusterização* ou agrupamento. A variável Δ_{c_1,c_2} representa a distância entre os *clusters* c_1 e c_2, que é calculada a partir da distância média entre todos os pares de indivíduos dos dois *clusters*. A distância entre dois *clusters* é obtida usando a distância Euclidiana entre os mesmos. A redução da população externa é realizada através da seleção do indivíduo representativo de cada cluster. Este indivíduo é o indivíduo que está mais próximo a todos os indivíduos do *cluster*. Sua localização é chamada de *centroide* do *cluster*.

Algoritmo 5 Poda através de *clusterização*

Entrada. População externa P'
Saída. População externa podada P'

1: $C := \bigcup_{s \in P'} \{\{s\}\}$;
2: $P' := \emptyset$;
3: **Enquanto** $\#C > \#P'$ **Faça**
4: **Para cada** $\text{par}(c_1, c_2) \in C$ **Faça**
5: $\Delta_{c_1,c_2} := \frac{1}{|c_1| \times |c_2|} \times \sum_{(i_1 \in c_1, i_2 \in c_2)} \delta_{i_1,i_2}$;
6: **Fim Para**
7: $C = C \setminus \{c_1, c_2\} \cup \{c_1 \cup c_2\}$ onde $\Delta_{c_1,c_2} = \min_{(x,y) \in C} \Delta_{x,y}$;
8: **Fim Enquanto**
9: **Para cada** $c \in C$ **Faça**
10: $P' := P' \cup \{s\}$ onde $\frac{\sum_{x \in c} \delta_{x,i}}{|c|} = \min_{y \in c} \left(\frac{\sum_{z \in c} \delta_{y,z}}{|c|} \right)$;
11: **Fim Para**
12: **Retorna** P';

O modo como é calculada a aptidão dos indivíduos é uma das características do SPEA. Este processo é realizado em duas etapas: Primeiro os indivíduos da população externa P' são avaliados e em seguida, são avaliados os indivíduos da população P. Essa avaliação é executada da seguinte forma:

1. Cada indivíduo $i \in P'$ recebe um valor real $s_i \in [0, 1)$ chamado de *strength*. Este valor é proporcional ao número de indivíduos $j \in P$ que são *encobertos* por i, isto é, $i = j$ ou $i \prec j$. Seja n a quantidade de indivíduos em P que são encobertos por i e N o tamanho de P. O *strength* do indivíduo s_i é definido por $s_i = \frac{n}{N+1}$. A aptidão f_i do indivíduo $i \in P'$ é seu *strength*: $f_i = s_i$.

2. A aptidão de um indivíduo $j \in P$ é calculada a partir da soma dos *strengths* de toda a população externa de soluções não dominadas $i \in P'$ que *encobrem* j. A este total, uma unidade é adicionada para garantir que os membros de P' serão mais aptos do que os

membros de P. A aptidão dos membros de P é dada da seguinte forma: $f_j = 1 + \sum_{i=j \vee i \prec j} s_i$, onde $f_i \in [1, N)$.

Comparando a atribuição de aptidão do SPEA com o método de *aptidão compartilhada*, a principal diferença é que os *clusters* não são definidos em termos de distância (como são os nichos) mas em termos de dominância (ZITZLER; THIELE, 1999). Isto torna desnecessária o uso do parâmetro de distância como aquele utilizado no MOGA (FONSECA; FLEMING, 1993) e NPGA (HORN; NAFPLIOTIS; GOLDBERG, 1994). Porem, o tamanho da população externa vem a influenciar na capacidade de formar *clusters* (ZITZLER; THIELE, 1999).

3.2.3.4 NSGA e NSGA-II – *Non-Dominated Sorting Genetic Algorithms*

O NSGA (SRINIVAS; DEB, 1994) é um algoritmo que se baseia em preservar as soluções não dominadas ao longo das gerações e manter uma boa diversidade da população através da criação de nichos. A cada iteração todas as soluções não dominadas são identificadas. À estas soluções é atribuído o mesmo *nível* que é chamado de *aptidão fraca*. No processo de formação de nichos é utilizada a técnica de *aptidão compartilhada* para atribuir diferentes valores de aptidão para soluções de mesmo *nível*. Após a formação dos primeiros nichos, as soluções não dominadas são ignoradas temporariamente e as soluções restantes passam pelo mesmo processo até que toda a população tenha sido avaliada. Uma vez que o processo de avaliação é concluído, os operadores de seleção, recombinação e mutação são executados nesta ordem.

A eficiência do NSGA se deve à representação de múltiplos objetivos através do valor de *aptidão fraca*, que pode ser traduzido como o *nível* do indivíduo e pela ordenação usando o critério de não dominância. Um fator que pode comprometer a eficiência do NSGA é o parâmetro de compartilhamento que define o tamanho dos nichos. Este parâmetro deve ser configurado *a priori* e varia muito de acordo com o problema em questão. Para eliminar a dependência do parâmetro de compartilhamento, foi desenvolvido o NSGA-II. O NSGA-II foi baseado no NSGA e em outros AEMs que usam o conceito de dominância. A *aptidão fraca*, utilizada no NSGA e também no NSGA-II, é obtida através do método de *ordenação pela dominância*. Este método foi aprimorado no NSGA-II e recebeu o nome de *ordenação rápida pela dominância*. Em seguida será apresentado o NSGA-II juntamente com as suas diferenças em comparação aos outros AEMs.

Embora algoritmos anteriores ao NSGA-II tivessem a capacidade de encontrar soluções não dominadas em múltiplas fronteiras, pesquisadores perceberam a necessidade de introduzir operadores adicionais. O primeiro operador sugerido, foi o operador de *elitismo*. Este operador

tem a finalidade de preservar os melhores indivíduos encontrados durante a evolução de um AE fazendo com que os mesmos se mantenham presentes até o final do processo evolucionário. A introdução do elitismo ajuda os AEMs a alcançar a convergência (ZITZLER; DEB; THIELE, 2000). Um dos AEMs elitistas que serviram de base para a formulação dos operadores do NSGA-II foi o SPEA (ZITZLER; THIELE, 1998), apresentado na seção anterior.

A seguir será mostrado o mecanismo de ordenação rápida pela dominância, o mecanismo de preservação de diversidade, que ao contrário da técnica de formação de nichos, não requer parâmetros adicionais, e por último, o algoritmo principal com seu mecanismo de elitismo. Todos esses mecanismos fazem parte no NSGA-II.

Ordenação rápida pela dominância. Para identificar soluções pertencentes à primeira fronteira de soluções não dominadas em uma população de tamanho N, pode-se comparar cada solução com todas as outras soluções da população. Este procedimento requer $M * N$ comparações para cada solução, onde M é o número de objetivos. Continuando este processo para toda a população, a complexidade final é $O(MN^2)$. Neste ponto, todos os indivíduos da primeira fronteira já foram encontrados. Para encontrar os indivíduos da próxima fronteira, as soluções da primeira fronteira são retiradas temporariamente da população e o procedimento acima é repetido. No pior caso, a complexidade de encontrar a segunda fronteira também será $O(MN^2)$. Estendendo para as demais fronteiras, o pior caso ocorre quando existem N fronteiras com apenas uma solução em cada uma delas e então a complexidade total será $O(MN^3)$. A ordenação rápida pela dominância proposta por Deb *et al.* (DEB *et al.*, 2002) reduz a complexidade do algoritmo para $O(MN^2)$, como é mostrado no Algoritmo 6. Este algoritmo mostra que para cada solução, são calculados dois valores: o contador de dominância n_p, que representa o número de soluções que dominam a solução p e o conjunto de soluções dominadas pela solução p, denotado por S_p.

Todas as soluções da primeira fronteira terão contadores de dominância nulos. Para cada solução p com $n_p = 0$, os membros, q, de seus conjuntos S_p, terão os respectivos contadores de dominância decrementados $n_q = n_q - 1$. Os membros q que ficarem com contadores nulos serão agrupados em uma lista separada Q. Estes membros pertencem a segunda fronteira. O procedimento anterior é repetido para cada membro de Q e a terceira fronteira é identificada. Repete-se este processo até que todas as fronteiras sejam identificadas.

Para cada solução p da segunda fronteira em diante, o contador associado n_p pode valer no máximo $N - 1$. Então, cada solução p será visitada no máximo $N - 1$ vezes antes que n_p

Algoritmo 6 Ordenação Rápida pela Dominância

```
1: Para todo p ∈ P Faça
2:    S_p = 0; n_p = 0
3:    Para todo q ∈ P Faça
4:       Se p ≺ q Então
5:          S_p = S_p ∪ {q}
6:       Senão Se q ≺ p Então
7:          n_p = n_p + 1
8:       Fim Se
9:    Fim Para
10:   Se n_p = 0 Então
11:      p_nível = 1; F_1 = F_1 ∪ {p}
12:   Fim Se
13: Fim Para
14: i = 1
15: Enquanto F_i ≠ 0 Faça
16:   Q = 0
17:   Para todo p ∈ F_i Faça
18:      Para todo q ∈ S_p Faça
19:         n_q = n_q − 1
20:         Se n_q = 0 Então
21:            q_nível = i + 1; Q = Q ∪ {q}
22:         Fim Se
23:      Fim Para
24:      i = i + 1; F_i = Q
25:   Fim Para
26: Fim Enquanto
```

se torne 0. Neste ponto, a solução receberá um identificador de dominância denominado de *nível* e não será mais visitada. Existindo no máximo $N-1$ soluções p a partir da segunda fronteira, a complexidade total é $O(N^2)$. Em consequência, a complexidade total do algoritmo é $O(MN^2)$. Outra maneira de avaliar esta complexidade é considerar os passos do Algoritmo 6. Como cada indivíduo pode pertencer no máximo à uma fronteira, a iteração da linha 17 é executada exatamente N vezes; como cada indivíduo domina no máximo $N-1$ indivíduos, a iteração da linha 18 é executada no máximo $N-1$ vezes para cada indivíduo. Como cada verificação de dominância requer no máximo M comparações, a complexidade final é $O(MN^2)$. Embora a complexidade tenha sido reduzida de $O(MN^3)$ para $O(MN^2)$, o uso de memória aumentou de N para N^2 devido o uso do conjunto S_p.

Preservação de diversidade. Durante o processo de evolução de um AEM, é importante manter uma boa diversidade de soluções que fazem parte da fronteira de Pareto. A primeira versão do NSGA (SRINIVAS; DEB, 1994) utilizava uma função de partilha, que se mostrou bastante eficiente para manter uma boa diversidade, desde que seus parâmetros fossem bem

ajustados. Um desses parâmetros é o de compartilhamento, que determina o grau de compartilhamento a ser adotado no problema. Este parâmetro denota a maior distância a qual dois indivíduos compartilham suas aptidões. É um parâmetro normalmente estático e é configurado pelo usuário. Por ser um parâmetro de grande influência sobre a convergência do processo evolucionário e de difícil ajuste, Deb e Goldberg desenvolveram alguns passos para o seu ajuste (DEB; GOLDBERG, 1989). As duas maiores dificuldades em utilizar a função de partilha, são:

1. O desempenho da função de partilha para manter uma boa diversidade de soluções é fortemente dependente do valor escolhido do parâmetro de compartilhamento.

2. Como cada solução deve ser comparada com todas as outras soluções da população, a complexidade da função de partilha é $O(N^2)$.

No algoritmo NSGA-II, a função de partilha é substituída por uma comparação de densidade populacional ou *comparação por aglomeração*. Esta comparação por aglomeração elimina as duas dificuldades, citadas anteriormente, que estão relacionadas com a utilização da função de partilha. O novo método não requer nenhum parâmetro para manter a diversidade entre os indivíduos da população e não é necessário comparar todas as soluções entre si. Para entender como é feita a comparação por aglomeração é preciso entender como funcionam dois mecanismos deste método, que são a estimativa de densidade e o operador de comparação por aglomeração. O primeiro é responsável por calcular a densidade de soluções ao redor de uma solução, já o segundo, determina a dominância, ou não, a partir do *nível* e da densidade de soluções. A seguir, é apresentado o funcionamento destes mecanismos.

1. Estimativa de densidade: Para obter uma estimativa da densidade de soluções ao redor de uma solução particular, calcula-se a distância média entre dois pontos em ambos os lados do ponto particular, para todos os objetivos. O valor encontrado serve para estimar o perímetro do cuboide formado a partir dos pontos vizinhos e é chamado de *distância de aglomeração*. Na Figura 32, a distância de aglomeração da i-ésima solução é representada pela largura média, i_{dist} do cuboide formado pelas soluções vizinhas $i-1$ e $i+1$ na mesma fronteira. Os pontos preenchidos representam soluções na mesma fronteira.

   Para calcular a distância de aglomeração é preciso ordenar a população de acordo com um dos objetivos em ordem ascendente. A soluções i encontradas nos limites superior e inferior de cada objetivo, são associadas com a distância infinita ($i_{dist} = \infty$). Todas as outra soluções intermediárias têm como distância de aglomeração, o valor absoluto

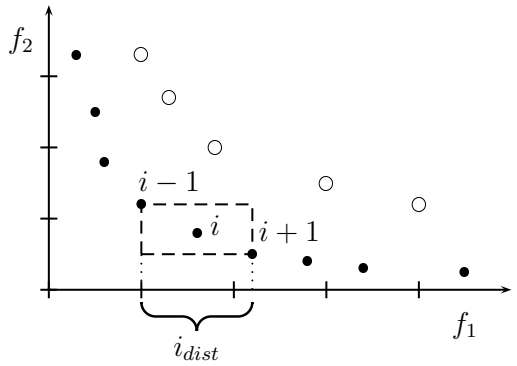


Figura 32: Distância de aglomeração

normalizado da diferença entre os valores do objetivo das soluções vizinhas. Esta distância é calculada para todos os objetivos e é obtida a partir do somatório das distâncias individuais para cada objetivo. Para evitar que soluções com objetivos discrepantes prejudiquem a estimativa, antes que a distância de aglomeração seja calculada, os objetivos são normalizados. O Algoritmo 7 mostra como é calculada a distância de aglomeração para cada solução $F_{(i)}$ do conjunto de soluções da mesma fronteira F. O valor de $F_{(i+1)_m}$ corresponde ao valor do m-ésimo objetivo do i-ésimo indivíduo do conjunto F. Os parâmetros f_m^{max} e f_m^{min} correspondem aos valores máximo e mínimo do m-ésimo objetivo. A complexidade deste algoritmo é determinada pelo algoritmo de ordenação utilizado. Algoritmos de ordenação ineficientes possuem complexidade $O(N^2)$ enquanto que algoritmos eficientes possuem complexidade $O(N\log(N))$(KNUTH, 1973). Utilizando um algoritmo de ordenação eficiente para ordenar M objetivos em uma população de N indivíduos, a complexidade final do cálculo da distância de aglomeração é $O(MN\log(N))$.

Após calcular a distância de aglomeração para todas as soluções do conjunto F é possível comparar duas soluções em função de suas proximidades com as outras soluções. Soluções com valores pequenos estão mais aglomeradas e portanto são considerada *mais parecidas*, resultando em baixa diversidade. O operador de comparação por aglomeração irá selecionar as soluções menos aglomeradas.

2. Operador de comparação por aglomeração: Este operador de comparação, representado por $\prec_{dist}$, guia o processo de seleção com o objetivo de formar uma fronteira de Pareto uniformemente distribuída. Assumindo que cada indivíduo da população possua um identificador de dominância representado pelo *nível* e pela distância de aglomeração, o operador de comparação por aglomeração assume que a solução i domina a solução j,

representado por $i \prec_{dist} j$, se:

$$i \prec_{dist} j \Longleftrightarrow i_{nível} < j_{nível} \vee ((i_{nível} = j_{nível}) \wedge (i_{dist} > j_{dist})) \quad (13)$$

Isto é, entre duas soluções de diferentes níveis de dominância, a solução escolhida será a de menor *nível*. Caso as soluções possuam o mesmo *nível* e portanto pertençam a mesma fronteira, a solução escolhida será a que estiver em uma região menos aglomerada.

Algoritmo 7 Cálculo da distância de aglomeração

```
1: n = |F|
2: Para todo F_(i) Faça
3:    F_(i)_dist = 0
4: Fim Para
5: Para todo m ∈ M Faça
6:    F = sort(F, m)
7:    F_(1)_dist = F_(n)_dist = ∞
8:    Para i = 2 a (n − 1) Faça
9:       F_(i)_dist = F_(i)_dist + (F_(i+1)_m − F_(i−1)_m)/(f_m^max − f_m^min)
10:   Fim Para
11: Fim Para
```

Computação principal: Inicialmente, uma população P_0 é criada aleatoriamente. A população é ordenada com base na dominância. A cada indivíduo é associado um nível de dominância (nível 1 é o maior, nível 2 é o segundo maior e assim por diante). Sem perda de generalidade, assume-se que as funções objetivo são de minimização. Os operadores de seleção, recombinação e mutação são usados para criar uma população filha Q_0 de tamanho N. O funcionamento do NSGA-II, para uma geração t, está descrito no Algoritmo 8 e comentado a seguir.

Primeiro, é formada uma população combinada $R_t = P_t \cup Q_t$. A população R_t tem tamanho $2N$. A população R_t é ordenada de acordo com o *nível* dos indivíduos. Como R_t é formada de indivíduos progenitores, P_t, e filhos, Q_t, o elitismo é assegurado. Agora, as soluções não dominadas de F_1, representam as melhores soluções da população combinada. Se o número de indivíduos em F_1 for menor do que N, todos os membros de F_1 serão selecionados para fazer parte da nova população P_{t+1}. Os membros restantes de P_{t+1} serão selecionados, de acordo com o *nível*, dos conjuntos F_i, tal que $i > 1$, até que P_{t+1} tenha N indivíduos. Suponhamos que o último conjunto que pode ser acomodado em P_{t+1} seja o conjunto F_n. Geralmente, o total de membros de F_1 a F_n, é maior do que N. Para selecionar exatamente os N melhores, os membros do último conjunto F_n são ordenados em ordem decrescente pelo operador de comparação por aglomeração ($\prec_{dist}$). Os membros com maior distância de aglomeração são selecionados até que P_{t+1} tenha N indivíduos. A nova população P_{t+1} de tamanho N é utilizada para seleção,

recombinação e mutação, dando origem assim à população Q_{t+1}, que também possui tamanho N. O processo de seleção é feito pelo operador de seleção por torneio e em caso de empate é utilizado o operador de comparação por aglomeração. Como este operador requer o *nível* e a distância de aglomeração de cada indivíduo, estas grandezas são calculadas desde a formação da população P_{t+1}.

Considerando a complexidade de uma interação do algoritmo, as operações básicas e seus piores casos são:

1. ordenação rápida pela dominância tem uma complexidade de $O(M(2N)^2)$;
2. determinar a distância de aglomeração tem uma complexidade de $O(M(2N)\log(2N))$;
3. ordenação pelo operador $\prec_{dist}$ tem uma complexidade de $O(2N\log(2N))$.

Portanto, a complexidade imposta pelo Algoritmo 8 é $O(MN^2)$, que é determinada pela operação de ordenação pela dominância. Analisando cuidadosamente, observa-se que não há necessidade de ordenar toda a população de tamanho $2N$. Assim que os N membros sejam encontrados, não há motivo para continuar com a ordenação; os demais serão automaticamente descartados.

Algoritmo 8 NSGA-II

1: $R_t = P_t \cup Q_t$
2: $F = \texttt{OrdenaçãoRápidaPorDominância}(R_t)$
3: $P_{t+1} = 0$
4: $i = 1$
5: **Enquanto** $|P_{t+1}| + |F_i| \leq N$ **Faça**
6: $\quad \texttt{CálculoDistânciaAglomeração}(F_i)$
7: $\quad P_{t+1} = P_{t+1} \cup F_i$
8: $\quad i = i + 1$
9: **Fim Enquanto**
10: Ordena$(F_i, \prec_{dist})$
11: $P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
12: $Q_{t+1} = \texttt{SeleçãoRecombinaçãoMutação}(P_{t+1})$
13: $t = t + 1$

3.2.3.5 MicroGA – *Micro Genetic Algorithm*

O microGA recebe este nome devido ao tamanho muito reduzido da população utilizada. O método surgiu a partir de resultados teóricos obtidos por Goldberg (GOLDBERG, 1989b), demonstrando que uma população de apenas três indivíduos é o suficiente para convergir, sem levar em conta o tamanho do cromossomo utilizado. Goldberg sugeriu iniciar o algoritmo com

uma pequena população gerada aleatoriamente; em seguida, utilizar os operadores de seleção, recombinação e mutação até alcançar o que é chamado de *convergência nominal*, que pode ser entendida como a perda total ou parcial de diversidade; então transferir os melhores indivíduos para a população da próxima geração e gerar os restantes aleatoriamente.

O primeiro registro de uma implementação de um microGA é de Krishnakumar (KRISHNAKUMAR, 1990), que utilizou uma população de cinco indivíduos, uma taxa de recombinação de 1 e taxa de mutação nula. Foi também utilizada uma estratégia elitista, copiando o melhor indivíduo encontrado para a população seguinte. Krishnakumar (KRISHNAKUMAR, 1990) comparou o microGA com um AG simples (com tamanho da população igual a 50, taxa de recombinação igual a 0,6 e taxa de mutação igual a 0,001). O microGA foi mais rápido e obteve melhores resultados. Após a implementação de Krishnakumar, outros pesquisadores desenvolveram diferentes implementações de um microGA (JOHNSON; ABUSHAGUR, 1995) (XIAO; YABE, 1998).

A primeira implementação de do microGA para solução de POM foi realizada por Coello e Pulido (COELLO; PULIDO, 2005). O microGA multiobjetivo herdou algumas características de outros algoritmos (JASZKIEWICZ, 2002). Uma delas é o uso de uma memória externa, que serve para inicializar uma pequena população. Coello e Pulido utilizaram duas memórias e as chamaram de *memória populacional* e *memória externa*. A primeira armazena um conjunto de indivíduos que participam do processo evolutivo, enquanto que a segunda, armazena os melhores indivíduos obtidos ao longo do processo evolutivo.

Inicialmente, é gerado um conjunto de soluções aleatórias que são armazenadas na memória populacional. Esta memória é dividida em duas partes, uma *renovável* e outra *não renovável*. A parte não renovável, como o nome sugere, nunca será alterada ao longo da execução do algoritmo e seu principal objetivo é manter a diversidade. A parte renovável será alterada a cada ciclo (geração) do microGA. A pequena população inicial do microGA é gerada de ambas as partes da memória populacional, conforme o esquema representado na Figura 33. O percentual de indivíduos provenientes de cada uma das partes da memória populacional é definido pelo usuário.

Durante cada ciclo, os operadores de elitismo, seleção, recombinação e mutação são executados. Ao final de cada ciclo é realizado o teste de convergência nominal. Considerando o tamanho reduzido da população, pode-se assumir que a convergência nominal será atingida a cada t ciclos. Coello e Pulido (COELLO; PULIDO, 2005) adotaram uma quantidade de dois a cinco ciclos no máximo. Esta suposição é perfeitamente aceitável e elimina o custo de comparar

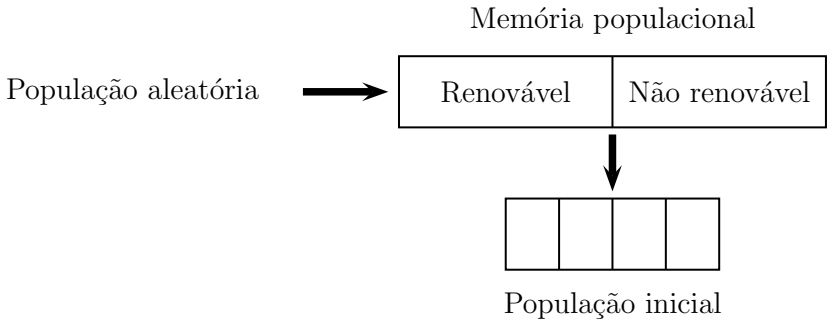


Figura 33: População aleatória, memória populacional e população inicial

os genes de todos os indivíduos para verificar a diversidade da população. Havendo convergência nominal, dois indivíduos são selecionados, caso haja apenas um indivíduo não dominado, este será o único selecionado. Os indivíduos selecionados são então comparados com todos os indivíduos da memória externa, inicialmente vazia. Caso estes não sejam dominados por nenhum indivíduo da memória externa, então serão acrescentados ao conteúdo da memória externa e os indivíduos que são dominados pelos novos indivíduos, serão excluídos da mesma. Estes novos indivíduos também serão comparados com outros dois (ou um) indivíduos, escolhidos aleatoriamente, da memória populacional. Se o indivíduo da memória populacional for dominado, então este será substituído pelo novo. O objetivo é fazer com que a memória populacional seja cada vez mais ocupada por indivíduos não dominados. Alguns destes indivíduos serão utilizados na população inicial de um novo ciclo.

O microGA utiliza três tipos de elitismo: *(i)* O primeiro consiste em guardar todos os indivíduos não dominados gerados a cada ciclo, com o propósito de não perder nenhuma informação valiosa originada no processo evolutivo; *(ii)* O segundo consiste em preencher a memória populacional com *soluções nominais*, i.e. soluções encontradas quando há convergência nominal, objetivando uma convergência gradual; *(iii)* O terceiro tipo de elitismo ocorre em intervalos determinados pelo usuário, chamados de *ciclos de recolocação*. Em cada ciclo de recolocação, certa quantidade de indivíduos, extraídos de todas as regiões da fronteira de Pareto construída até o momento, é utilizada para preencher a memória populacional. Dependendo do tamanho da parte renovável dessa memória, são selecionados indivíduos em número suficiente da fronteira de Pareto para permitir uma distribuição uniforme, alem de evitar que os indivíduos fiquem homogêneos. Este método visa alcançar uma rápida convergência para a fronteira de Pareto mantendo uma boa distribuição de soluções.

Para manter a diversidade, o microGA utiliza uma *grade adaptativa* semelhante àquela

proposta por Knowles e Corne (KNOWLES; CORNE, 2000). Esta grade funciona como uma matriz que cobre o espaço de soluções se adaptando ao tamanho deste espaço. Sempre que a memória externa atingir o seu limite, o espaço de busca coberto pelas soluções existentes, nesta memória, será dividido e cada solução receberá um conjunto de coordenadas baseadas na grade adaptativa. Então, cada nova solução não dominada será aceita, apenas se a posição geográfica no qual esta é mapeada, estiver menos aglomerada do que a posição mais aglomerado da grade. Uma nova solução não dominada que pertença a um local fora das fronteiras existentes, também pode ser adicionada, com o custo de recalcular toda a grade adaptativa novamente. Em resumo, os locais menos aglomerados possuem preferência para receber novas soluções da fronteira de Pareto, aprimorando assim a distribuição uniforme de soluções.

O cálculo da grade adaptativa requer dois parâmetros: o tamanho esperado da fronteira de Pareto e a quantidade de posições no qual o espaço de soluções de cada objetivo será dividido. O primeiro parâmetro é definido pelo tamanho da memória externa e portanto é configurado pelo usuário. O segundo parâmetro é mais difícil de ser determinado, porem, testes demonstraram que valores entre quinze e vinte produzem um bom resultado (KNOWLES; CORNE, 2000). Foi também mostrado que o algoritmo não é muito sensível ao valor atribuído a este parâmetro (COELLO; PULIDO, 2005). O cálculo da localização dos indivíduos é baseado no valor de seus objetivos. Este cálculo é de baixo custo computacional. Entretanto, quando um indivíduo está localizado fora das fronteiras existentes, é preciso realocar toda a população. Embora esta situação não ocorra com frequência, é possível determinar limites maiores prevendo esta situação.

A grade adaptativa também é utilizada para determinar os indivíduos que serão descartados quando a memória externa estiver cheia. Os indivíduos descartados serão aqueles localizados nas regiões de maior aglomeração. Através deste método é possível balancear a quantidade de soluções não dominadas, espalhando-as uniformemente ao longo da fronteira de Pareto. O funcionamento do microGA é descrito nos passos do Algoritmo 9.

3.3 Seleção de Métodos para Implementação

Para implementar a alocação evolutiva de IPs e o mapeamento evolutivo de IPs, como será explicado no Capítulo 4 e no Capítulo 5, respectivamente, foram escolhidos dois algoritmos genéticos multiobjetivos: o NSGA-II e o microGA. O NSGA-II foi escolhido devido aos bons resultados obtidos em *benchmarks* para POMs e por não depender da configuração de parâmetros críticos para o desempenho do algoritmo e de difícil ajuste (DEB *et al.*, 2002). O microGA

Algoritmo 9 microGA

1: $M_P = P$
2: $t = 0$
3: **Enquanto** $t < t.max$ **Faça**
4: $P_t = \{p_t^1, p_t^2, \ldots, p_t^n \mid p_t \in M_{PR} \subset M_P \vee p_t \in M_{PNR} \subset M_P\}$
5: **repeat**
6: $Q_t =$ `SeleçãoRecombinaçãoMutação`(P_t)
7: $\{q_t\} =$ `Não-Dominado`(Q_t)
8: $P_t = Q_t$
9: **until** convergência nominal
10: **Se** E não cheia **Então**
11: $E = E \cup \{p_t, q_t \mid p_t =$ `Não-Dominado`$(P_t) \wedge p_t \neq q_t\}$
12: **Senão**
13: `GradeAdaptativa`(E, p_t, q_t)
14: **Fim Se**
15: $M_{PR} = M_{PR} \cup \{p_t, q_t\}$
16: **Se** $i \setminus$ cicloDeRecolocação $= 0$ **Então**
17: `Recolocação`(E, M_{PR})
18: **Fim Se**
19: $t = t + 1$
20: **Fim Enquanto**

foi selecionado por também ter apresentado bons resultados em *benchmarks* para POMs (COELLO; PULIDO, 2001) e para poder comparar os resultados obtidos por ambos e o desempenho das duas técnicas empregadas nos dois algoritmos.

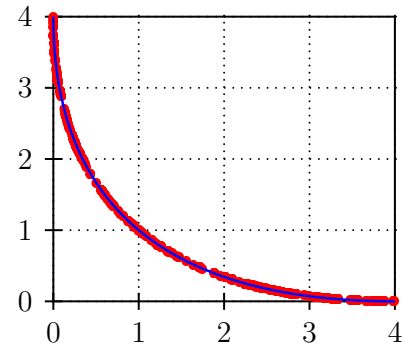
Para validar a implementação de ambos os algoritmos, foram utilizados POMs que são considerados como *benchmarks* nesta área. Os problemas multiobjetivos utilizados foram: *SCH*, utilizado no estudo de Schaffer (SCHAFFER, 1985), tratando-se de uma fronteira convexa, *FON*, utilizado no estudo de Fonseca e Fleming (FONSECA; FLEMING, 1998), tratando-se de uma fronteira côncava, *KUR*, utilizado no estudo e Kursawe (KURSAWE, 1991), tratando-se de uma fronteira com descontinuidades e *ZDT4* utilizado no estudo de Zitzler, Deb e Thiele (ZITZLER; DEB; THIELE, 2000), tratando-se de uma fronteira côncava. A Tabela 2, apresenta para cada problema, as funções objetivo, o número de variáveis de decisão n, os limites do espaço de busca, o intervalo onde as soluções ótimas se encontram e o tipo da fronteira de Pareto.

Os resultados obtidos com as implementações do NSGA-II e do microGA, para os POMs de teste descritos na Tabela 2, estão representados na Figura 34 para as funções *SCH* e *FON* e na Figura 35 para as funções KUR e *ZDT4*.

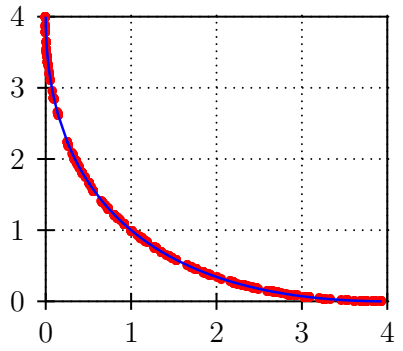
Ambos os algoritmos encontraram diversas soluções ótimas, formando a fronteira de Pareto dos problemas utilizados como *benchmark*. O NSGA-II, comparado com o microGA,

Tabela 2: POMs utilizados como *benchmarks*

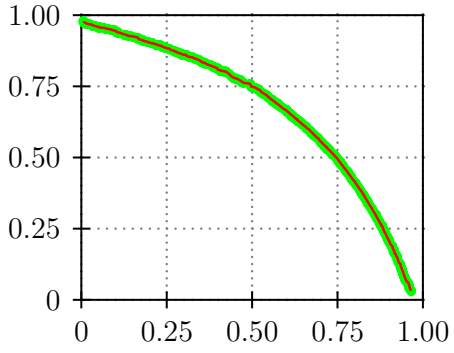
Problema	n	Limites	Funções objetivo	Intervalo ótimo
SCH	1	$[-10^3, 10^3]$	$f_1(\mathtt{x}) = x_i^2$ $f_2(\mathtt{x}) = (x_i - 2)^2$	$x \in [0, 2]$
FON	3	$[-4, 4]$	$f_1(\mathtt{x}) = 1 - exp(-\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{3}})^2)$ $f_2(\mathtt{x}) = 1 - exp(-\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{3}})^2)$	$x_1 = x_2 = x_3$ $\in [-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$
KUR	3	$[-5, 5]$	$f_1(\mathtt{x}) = \sum_{i=1}^{n-1} -\frac{exp(-0{,}2\sqrt{x_i^2+x_{i+1}^2})}{0{,}1}$ $f_2(\mathtt{x}) = \sum_{i=1}^{n}(\mid x_i \mid^{0{,}8} + sen(x_i^3))$	$x \in [-1, 1]$
ZDT4	10	$x_1 \in [0, 1]$ $x_i \in [-5, 5]$ $i \in 2, \ldots, n$	$f_1(\mathtt{x}) = x_1$ $f_2(\mathtt{x}) = g(\mathtt{x})\left(1 - \sqrt{x_1/g(\mathtt{x})}\right)$ $g(\mathtt{x}) = 1 + 10(n-1) +$ $\sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	$x_1 \in [0, 1]$ $x_i = 0$ $i \in 2, \ldots, n$



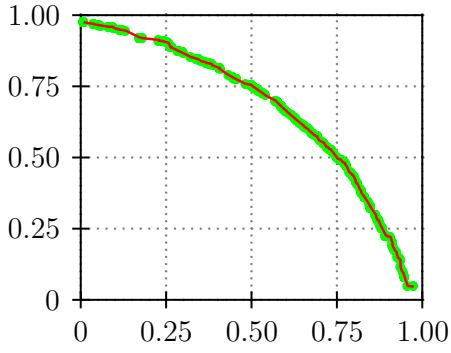
(a) $\mathscr{F}_{\mathscr{P}}$ encontrada pelo NSGA-II para *SCH*



(b) $\mathscr{F}\mathscr{P}$ encontrada pelo microGA para *SCH*



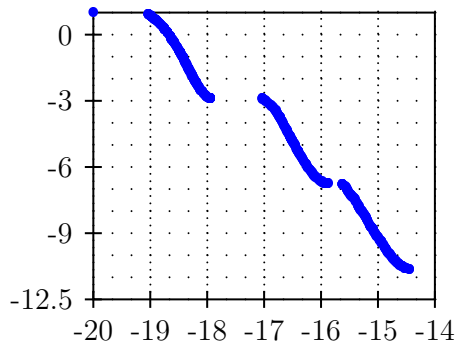
(c) $\mathscr{F}\mathscr{P}$ encontrada pelo NSGA-II para *FON*



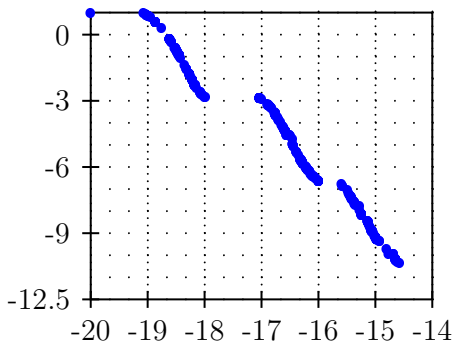
(d) $\mathscr{F}\mathscr{P}$ encontrada pelo microGA para *FON*

Figura 34: Resultados obtidos com o NSGA-II e microGA para os *benchmarks*: *SCH* e *FON*

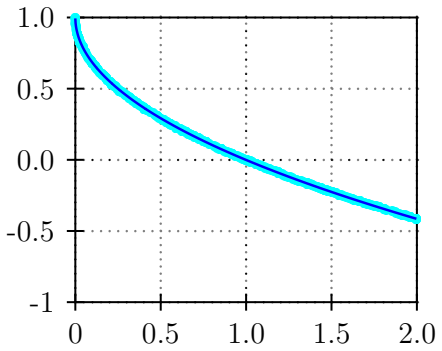
encontrou uma diversidade maior de pontos mas ambos encontraram uma boa quantidade de soluções ótimas.



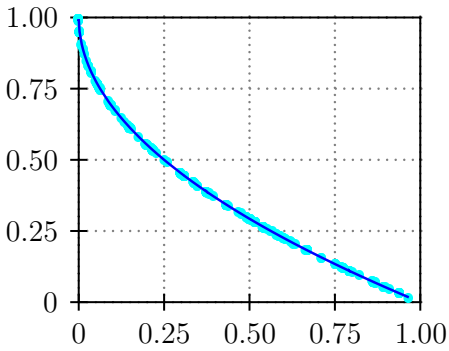
(a) $\mathscr{F}\mathscr{P}$ encontrada pelo NSGA-II para *KUR*



(b) $\mathscr{F}\mathscr{P}$ encontrada pelo microGA para *KUR*



(c) $\mathscr{F}\mathscr{P}$ encontrada pelo NSGA-II para *ZDT4*



(d) $\mathscr{F}\mathscr{P}$ encontrada pelo microGA para *ZDT4*

Figura 35: Resultados obtidos com o NSGA-II e microGA para os *benchmarks*: KUR e *ZDT4*

3.4 Considerações Finais do Capítulo

Algoritmos evolucionários são algoritmos inspirados na teoria evolucionária baseada na sobrevivência do mais apto e na evolução ocorrida para que o indivíduo se adapte ao meio. O funcionamento básico destes algoritmos consiste em realizar transformações em indivíduos, que representam soluções de um problema, de modo a simular um processo evolucionário. Este processo consiste em avaliação, seleção, recombinação e mutação das soluções. Uma classe especial de algoritmos evolucionários são os algoritmos genéticos. Estes algoritmos representam a solução de um problema através de um cromossomo e realizam operações de seleção, recombinação e mutação. Para tratar de problemas de otimização multiobjetivos, existem os algoritmos evolucionários multiobjetivos e nesta classe estão os algoritmos genéticos multiobjetivos. Diferentes técnicas de avaliação e seleção de soluções influenciam no desempenho destes algoritmos, que precisam avaliar múltiplos objetivos por indivíduo. O método de seleção por nível de dominância é um método que considera todos os objetivos de um indivíduo e seleciona

os indivíduos que apresentam o melhor balanço entre os objetivos. Os algoritmos genéticos multiobjetivos NSGA-II e microGA, são algoritmos com estratégias de busca e otimização diferentes e que apresentaram bons resultados experimentais. Estes dois algoritmos serão utilizados para implementação dos métodos propostos de alocação evolutiva de IPs e mapeamento evolutivo de IPs em plataforma NoC, como será apresentado nos capítulos seguintes.

Capítulo 4

ALOCAÇÃO EVOLUTIVA DE IPS EM PLATAFORMA NOC

O PROJETO de síntese de uma plataforma NoC utiliza uma metodologia que divide o processo em várias etapas. Cada etapa é caracterizada por um grau de abstração do *hardware* a ser implementado, sendo que cada uma possui um grau de abstração maior do que o da etapa subsequente. Ao longo desses etapas, diferentes modelos da plataforma são elaborados. Os modelos iniciais são funcionais e não especificam detalhes de *hardware* enquanto os modelos finais são mais estruturais e apresentam as características do *hardware* a ser implementado. Mais detalhes sobre essa metodologia se encontram na Seção 1.3 (do Capítulo 1).

Com base em modelos mais ou menos abstratos, são realizados processos de otimização com o objetivo de refinar cada vez mais as soluções intermediárias, para obter uma solução final, completa e ótima. A primeira otimização é realizada em termos de IPs a serem usados para implementar a aplicação na plataforma NoC projetada. Esta otimização é baseada em um modelo inicial que expressa a especificação da aplicação. Este modelo consiste em um grafo de tarefas, onde cada uma é executada por um IP da NoC. Os IPs são selecionados de um ou mais repositórios (ou bibliotecas) de IPs prontos para serem utilizados de acordo com as necessidades do projeto.

A Seção 4.1 descreve o problema *alocação de IPs*, que consiste em selecionar IPs para executar as tarefas de uma aplicação, definindo a noção de repositório de IPs, grafo de tarefas para a especificação de aplicação e grafo de caracterização de aplicação. Nesta seção, também é avaliada a complexidade do problema e o impacto de uma alocação específica no desempenho da implementação final. A Seção 4.2 apresenta uma proposta de solução para este problema através de métodos evolutivos, mostrando a codificação do conteúdo do repositório, do grafo de tarefas e dos indivíduos, para permitir um processo evolutivo eficaz e eficiente. Em seguida, são definidas as funções de avaliação dos objetivos de interesse para uma solução evoluída. A

Seção 4.3 apresenta os resultados obtidos de dois algoritmos evolucionários que são utilizados na implementação da alocação evolutiva de IPs e a comparação do desempenho desses algoritmos. A Seção 4.4 faz o fechamento deste capítulo com algumas considerações finais e introduz o assunto a ser abordado no próximo capítulo.

4.1 O Problema de Alocação de IPs

A metodologia de projeto baseada em plataforma é voltada para a reutilização de recursos existentes que foram elaborados em projetos anteriores. Por outro lado, uma outra forma de reutilizar recursos é feita quando um mesmo IP executa diferentes tarefas de uma aplicação. Isto é possível principalmente em projetos que utilizam múltiplos processadores, onde cada processador é capaz de executar de maneira eficiente várias tarefas da aplicação. Como cada processador possui características próprias, a combinação de diferentes processadores juntamente com o reaproveitamento de alguns para executar tarefas específicas, resulta em diferentes soluções para o problema de alocação.

O problema de alocação de IPs à tarefas é um problema de otimização combinatória (GAREY; JOHNSON, 1979). O objeto da otimização é a plataforma NoC a ser implementada. Otimizar neste contexto significa explorar diferentes combinações na busca por soluções que atendam a diferentes objetivos do projeto. Considerando a otimização destes diferentes objetivos, o problema de alocação de tarefas consiste então de um POM (problema de otimização multiobjetivo).

A busca de soluções válidas para o problema de alocação da IPs é feita a partir da seleção de IPs, levando em conta as características das tarefas, que são fornecidas na especificação da aplicação e as características dos IPs. Neste contexto, uma aplicação é geralmente especificada através de um *grafo de tarefas*, que será definido com detalhes na Seção 4.1.2. Os IPs são selecionados de uma biblioteca, também chamada de *repositório* onde cada IP disponível é especificado através das características próprias em termos de tempo de execução, consumo de energia, área, custo, entre outras. Note que quanto mais detalhado for o grafo de tarefas e mais extenso for o repositório de IPs disponíveis, mais complexo se torna o problema de otimização. A complexidade do problema aumenta em um fator exponencial, o que o caracteriza como um problema *NP*-completo. Mais detalhes sobre este assunto podem ser encontrados na Seção 2.1.2 (do Capítulo 2).

4.1.1 Repositório de IPs

Com o aumento da fabricação de dispositivos que utilizam SoCs, MPSoCs e NoCs, repositórios de IPs são fundamentais para acelerar os projetos destes dispositivos. Esta aceleração ocorre em virtude da reutilização de IPs previamente testados em projetos anteriores. Os IPs podem ser fornecidos na forma de núcleos de *software*, *firmware* e *hardware*, definidos na Seção 1.1.2 (do Capítulo 1). Na fase inicial de projeto, uma descrição detalhada das características dos IPs é suficiente para desenvolver um modelo lógico do projeto. Nesta dissertação, o repositório *E3S Benchmark Suite* (DICK, 2008) foi utilizado para validar o mérito das técnicas propostas. No entanto, qualquer outro repositório com as características necessárias para cada IP pode ser utilizado.

O E3S (DICK, 2008) é composto de 17 processadores utilizados para sistemas embutidos e 5 conjuntos de diferentes aplicações. Cada um dos processadores disponíveis pode executar algumas das 46 diferentes tarefas usadas nas aplicações especificadas. As tarefas são definidas por seu tempo de execução e consumo de energia quando executada por um processador específico. Os processadores são definidos através das respectivas frequência de operação, respectivos custos e tamanhos reais sem o invólucro (i.e. *die size*). O repositório foi codificado de modo que as tarefas representam IPs que estão associados a processadores. Esta codificação é apresentada na Seção 4.2.1. Os dados dos processadores e das tarefas foram obtidos junto aos fabricantes ou experimentalmente. Cada um dos 5 conjuntos de aplicações possui características próprias e representam um segmento específico. São denominados de *indústria automotiva*, *consumo*, *redes*, *escritório* e *telecomunicações*. Mais detalhes sobre as características de cada conjunto de aplicações e de suas respectivas aplicações, encontram-se no Apêndice B.

4.1.2 Grafo de tarefas

Dispositivos eletrônicos que utilizam NoCs são normalmente dispositivos voltados para um uso específico. A maneira como estes dispositivos funcionam é definida pelo *software* instalado e que será executado pelo *hardware* ou por um ASIC dedicado. Neste caso, o *hardware* formado pelos processadores e ASICs dispostos em uma plataforma NoC. Nem o *software* final e nem o *hardware* final são conhecidos a princípio. As funcionalidades da aplicação é especificada através das suas tarefas básicas e os fluxos de dados e controle entre estas e podem ser descritas através de um fluxograma. As operações desse fluxograma podem representar tarefas complexas ou operações básicas tal como uma adição de dois inteiros, dependendo do nível de abstração desejado. O fluxograma é geralmente chamado de *grafo de tarefas*. Nesta dissertação, o modelo

de grafo de tarefas utilizado está de acordo com a Definição 13.

Definição 13 (Grafo de Tarefas – GT). *Um grafo de tarefas, denotado por* $GT = G(T, D)$*, é um grafo orientado acíclico, onde cada nó representa um módulo de processamento da aplicação, referente à uma tarefa* $t_i \in T$ *e cada aresta orientada* $d_{i,j} \in D$*, entre duas tarefas* t_i *e* t_j*, caracteriza uma dependência de dados ou controle entre estas tarefas.*

Conforme a Definição 13, um grafo de tarefas é composto por nós (ou vértices) e caminhos orientados chamados de arestas (ou arcos). Os nós representam tarefas que em uma plataforma NoC serão associadas com elementos de processamento. Uma tarefa pode ser uma operação em ponto flutuante, uma codificação (ou decodificação) de áudio (ou vídeo), uma operação matricial, entre outras. Cada nó pode conter informações referentes à tarefa que este representa. Esta informação pode ser o custo computacional da tarefa ou qualquer outra característica da mesma, dependendo do modelo utilizado. Normalmente, os processos são iterativos onde a cada iteração, um nó consome dados de entrada, processa esses dados e por fim gera dados de saída. As arestas são representadas com setas que indicam a direção do fluxo de dados e/ou de controle e determinam a dependência causal entre os processos (MARSLAND; YANG, 1994). À estas arestas podem estar associados *pesos* que representam aspectos referentes à comunicação entre as tarefas, tais como, a largura de banda disponível ou quantidade de *bits* enviada ou recebida. Também representam as dependências das tarefas em relação ao término de outras tarefas. Uma tarefa representada por um nó n só pode ser executada se todas as tarefas representadas pelos nós com o qual nó n possui dependência causal estiverem concluídas e os dados de entrada estiverem disponíveis.

A Figura 36 mostra um grafo com 4 nós. Cada nó possui um identificador e está associado com um tipo de tarefa. Cada aresta também possui um identificador e possui um custo que é referente à comunicação de dados. Assumi-se que os nós sem predecessores recebem dados do ambiente externo. Do mesmo modo, assume-se que os nós sem sucessores geram dados de saída para o ambiente externo.

4.1.3 Complexidade do problema e impacto da alocação de IPs

Considere uma aplicação cujo o grafo inclui m tarefas, t_0, t_1, ..., t_{m-1} e um repositório de n IPs. Seja n_i o número de IPs que podem ser alocados à tarefa t_i. O número de diferentes alocações possíveis A pode ser calculado conforme Equação 14. Note que tem-se $1 \leq A \leq n^m$. O caso limite com $A = 1$ acontecem quando o repositório é formado de um único IP e este pode ser alocado à qualquer tarefa incluída no grafo da aplicação e o outro caso limite $A = n^m$

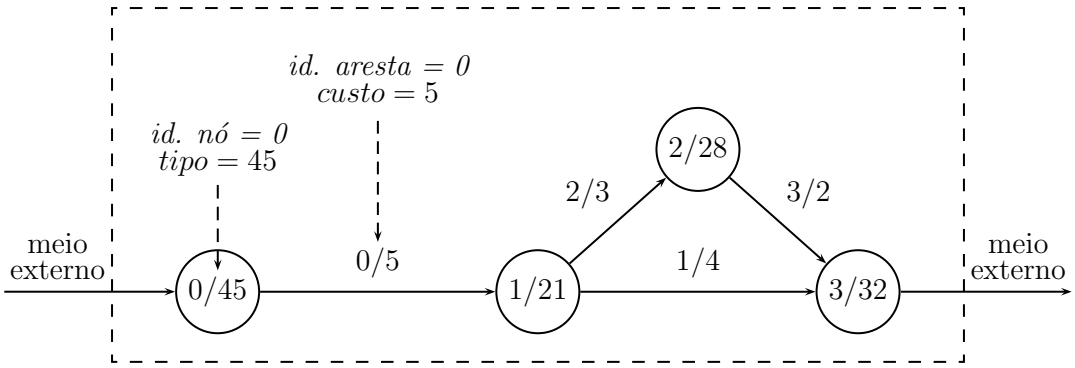


Figura 36: Exemplo de grafo de tarefas

Tabela 3: Ilustração da explosão exponencial de número de alocações factíveis

Número de	Número de IPs no repositório (n)				
tarefas (m)	2	3	4	5	16
2	4	9	16	25	256
3	8	27	64	125	4.096
4	16	81	256	625	65.536
5	32	243	1.024	3125	1.048.576
6	64	729	4.096	15.625	16.777.216
7	128	2.187	16.384	78125	268.435.456
8	256	6.561	65.536	390625	4.294.967.296
9	512	19.683	262.144	1953125	68.719.476.736
10	1.024	59.049	1.048.576	9.765.625	1,09951E+12

acontece quando o repositório abrange m IPs e todos estes podem ser alocados à qualquer tarefa do grafo. Para ilustração, o número de alocações factíveis neste caso, para alguns valores de n e m, é mostrado na Tabela 3.

$$A = n_0 \times n_1 \times \ldots n_{m-2} \times n_{m-1} \tag{14}$$

Em uma alocação, um IP é dito *dedicado* se e somente se, o processador que lhe associado é completamente dedicado à execução de uma única tarefa. Senão o IP é dito *compartilhado*, o que significa que o processador associado executa mais de uma tarefa da aplicação. Portanto, considerando o fato que um IP pode ser dedicado ou compartilhado, o número de IPs que podem ser associados à tarefa t_i é dobrado, passando ao valor $2 \times n_i$. Em consequência, o número total de alocações possíveis pode ser alterado, conforme mostra Equação 15. Observe que $A' = 2^m \times A$, e portanto $A' \gg A$.

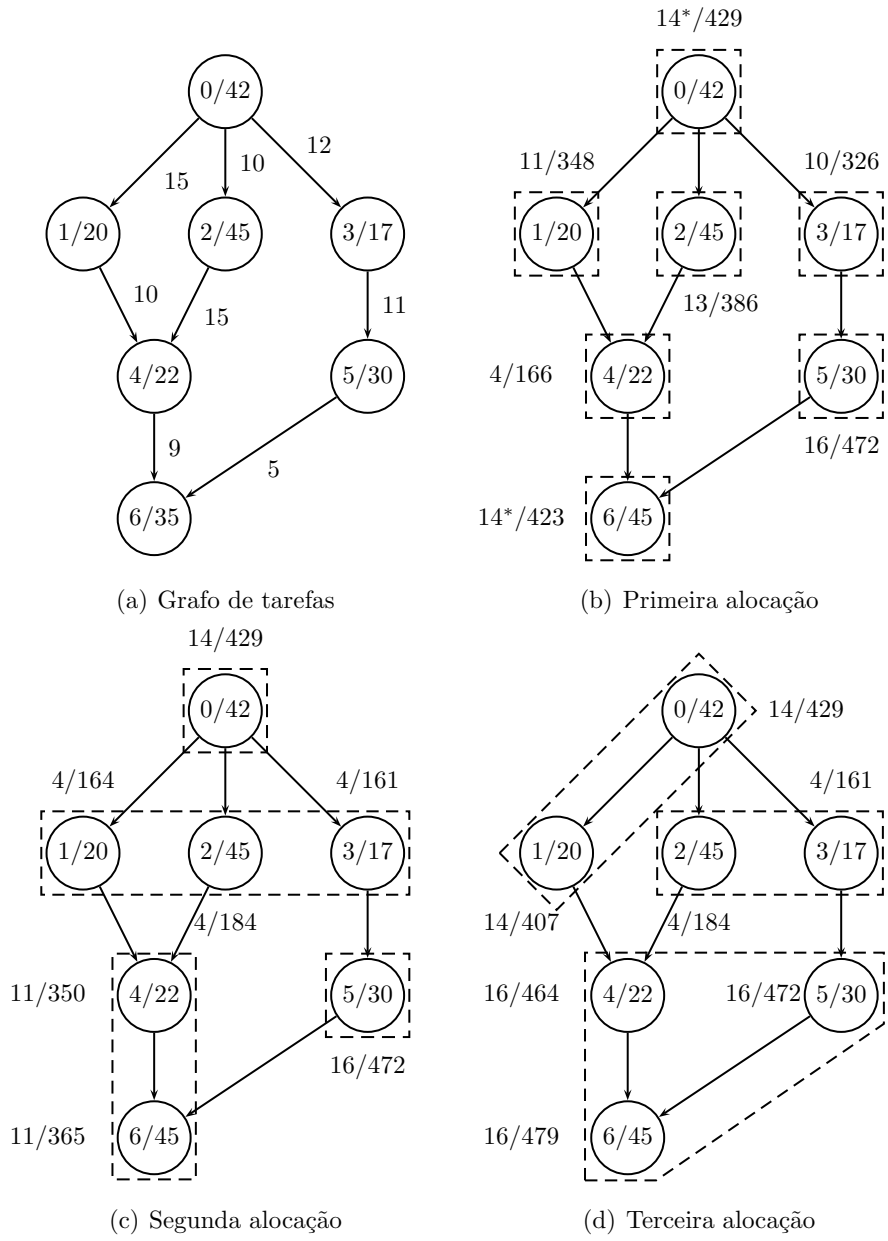
$$A' = 2^m \times n_0 \times n_1 \times \ldots n_{m-2} \times n_{m-1} \tag{15}$$

Tabela 4: Impacto das alocações ilustradas na Figura 37

No.	Alocação	#EPs	Tempo[(1)]	Área[(2)]	Consumo[(3)]	Custo[(4)]
1	[14*, 11, 13, 10, 4, 16, 14*]	7	32,02	151,86	26,07	417,90
2	[14, 4, 14, 14, 11, 16, 11]	4	31,93	128,68	53,64	318,90
3	[14, 14, 4, 4, 16, 16, 16]	3	35,26	36,78	41,80	266,80

[1]($\times 10^{-3}$s), [2]($\times 10^{-6}$m^2), [3](W), [4]($US)

A alocação de IPs é de suma importância para a implementação de uma aplicação em uma plataforma NoC. Para ilustrar isso, considere o grafo de tarefas da Figura 37(a). Este tem 7 tarefas no total, onde o número de IPs que podem ser alocados à cada uma dessas tarefas varia e pode ser encontrado na Tabla 35 e na Tabela 36 (do Apêndice A). Note que o número total de alocações correspondentes à este grafo de tarefa é de $A' = 2^7 \times 9 \times 12 \times 17 \times 12 \times 14 \times 14 \times 17 = 9.396.559.872$, i.e. mais do que 9 bilhões de combinações possíveis. As Figuras 37(b)–(d) apresentam três possíveis alocações de processadores, onde uns fazem o papel de IPs dedicados enquanto que outros são responsáveis por mais de uma tarefa. Nestas figuras, os dados de comunicação foram omitidos e os identificadores de processadores/IPs alocados foram introduzidos. Os valore obtidos a partir da avaliação das três alocações está representado na Tabela 4. A primeira alocação, na Figura 37(b), tenta minimizar o tempo de execução total da aplicação dedicando um processador a cada tarefa. Observe que as tarefas 0 e 6 serão executadas pelo mesmo tipo de processadores (i.e. 12), mas neste caso, dois processadores deste tipo serão utilizados, cada um dedicado à tarefa que executa. A segunda alocação, na Figura 37(b), usa menos processadores mas permite um tempo de execução menor do que aquele atingido pela primeira alocação. A área ocupada, o consumo de energia e o custo devido, também são menores se comparados com a primeira alocação. A terceira alocação, na Figura 37(d), permite um melhor balanço entre todos os objetivos, ocupando menos área, consumindo menos energia e impondo um custo menor comparado com as outras duas soluções. Contudo, o tempo de execução oferecido pela terceira alocação é maior em relação às demais alocações. Note que a primeira alocação é dominada pelas outras duas e a segunda e terceira alocações representam soluções não dominadas. O procedimento que permite calcular do consumo de energia, a área e o tempo de execução devidos a uma alocação específica, é detalhado na Seção 4.2.3.



(a) Grafo de tarefas

(b) Primeira alocação

(c) Segunda alocação

(d) Terceira alocação

Figura 37: Impacto de diferentes alocações de IPs

4.1.4 Grafo de caracterização de aplicação

Nesta etapa do projeto a otimização é feita baseando-se apenas em informações do grafo de tarefas e do repositório de IPs. As soluções obtidas representam os conjuntos de IPs que maximizam o desempenho da NoC sem considerar alguns aspectos físicos tais como, distância entre os recursos onde os IPs são mapeados na plataforma NoC e o impacto dessa localização na comunicação necessária à execução da aplicação implementada. Após a alocação de tarefas, o resultado obtido é chamado de *grafo de caracterização de aplicação.*

Definição 14 (Grafo de Caracterização de Aplicação). *Um grafo de caracterização de aplicação, denotado por* $GCA = G(I, D)$*, correspondente ao grafo de tarefa* $GT = (T, D)$*, é um grafo orientado, onde a cada nó* $i \in I$ *está associado com um nó* $t \in T$*, sendo que* i *representa o IP alocado para executar a tarefa do nó* t*.*

Note que o CGA não é nada mais do que o GT da aplicação juntamente com a seleção de IPs alocados às tarefas do GT. Estes IPs podem ser dedicados à uma única tarefa ou compartilhados por várias destas. As alocações esperadas, são na verdade as soluções de um POM. Como visto no Capítulo 2, as soluções ótimas de um POM formam uma fronteira ótima de Pareto e representam o melhor balanço entre os objetivos avaliados no processo de otimização. A próxima seção apresenta a utilização dos algoritmos evolucionários multiobjetivos NSGA-II e microGA para obter as soluções ótimas de Pareto na forma de alocações de IPs para uma dada aplicação. Detalhes dos algoritmos se encontram na Seção 3.2.3 (do Capítulo 3).

4.2 Alocação Evolutiva de IPs

Dois métodos evolutivos de otimização multiobjetivo foram escolhidos para resolver o problema de alocação de tarefas: NSGA-II e microGA que são mais especificamente, dois algoritmos genéticos multiobjetivos. Esta abordagem não determinística foi escolhida porque o tempo de resolução do problema em questão cresce exponencialmente de acordo com o tamanho da entrada, que é formada pelo grafo de tarefas e repositório de IPs. Esse impacto é detalhado na Seção 4.1.3.

O processo de alocação evolutiva de IPs consiste em aproveitar o caráter evolucionário dos AGMs para gerar soluções eficientes a partir do conjunto de todas as soluções possíveis. Neste intuíto, é preciso codificar concisa e objetivamente todos os elementos que caracterizam o problema e portanto são manipulados durante a resolução deste. Isso inclui o repositório de IPs que constitui o espaço de busca, o grafo de tarefas da aplicação a ser implementada

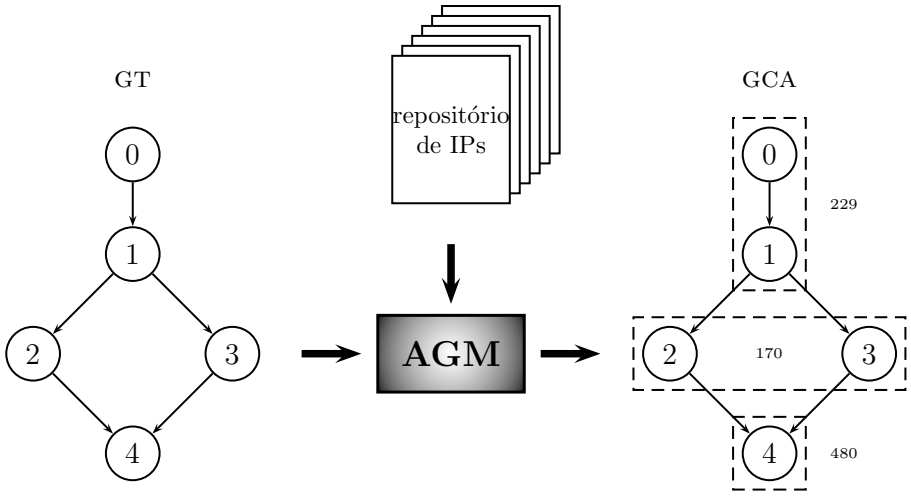


Figura 38: Dinâmica do processo de alocação evolutiva de IPs

em plataforma NoC e a solução, formando o indivíduo no processo evolutivo. O problema é codificado na forma de um cromossomo, para que este então possa ser avaliado e manipulado por operadores genéticos. Além de codificar o problema, é necessário definir funções eficientes para a avaliação de aptidão das soluções evoluídas, em termos dos objetivo selecionados. A Figura 38 ilustra a dinâmica deste processo.

4.2.1 Codificação

Nesta seção, serão apresentadas as diferentes codificações usadas para representar os objetos manipulados pelo processo evolutivo de alocação de IPs. Estes objetos são o repositório de IPs, o grafo de tarefas e os cromossomos representando as soluções do problema.

4.2.1.1 Codificação do repositório

Originalmente o E3S é disponibilizado como um conjunto de documentos de texto contendo as descrições dos processadores e suas tarefas, assim como as descrições dos grafos de tarefas. Para facilitar a integração do repositório com uma ferramenta computacional e para aumentar a legibilidade e compreensão do seu conteúdo, os documentos de texto originais foram re-estruturados utilizando a formatação de um documento XML (W3C, 2008).

Além de o formato XML aumentar a legibilidade dos dados, este também facilita a integração de dados entre diferentes ferramentas computacionais. Atualmente, não existe uma formatação padrão para a representação de repositórios de IPs no formato XML. Portanto, a estruturação foi feita de modo que facilitasse a integração com a ferramenta utilizada neste

trabalho e a divisão dos dados em elementos e atributos foi efetuada com base nos dados disponibilizados no E3S.

O E3S consiste de um conjunto de dados com informações de 17 processadores, que executam até 46 tarefas diferentes, e 5 conjuntos de aplicações, representadas como grafos de tarefas. Consequentemente, poderia ser visto como uma lista de IPs dedicados, isto é, IPs elaborados para executar uma tarefa única e específica. Como cada tarefa poderia ser executada exclusivamente por um processador, formando assim um IP dedicado, ou dividir o processador com outras tarefas, o repositório foi estruturado como uma lista de IPs, onde cada um é representado por uma *tag* <*ip*>. Os atributos desta *tag* são referentes às características da tarefa e do processador selecionado para executá-la. Sendo assim, cada *tag* <*ip*> possui um identificador único (de 0 a 479) e vários atributos que são divididos em duas partes: os atributos referente ao processador e aqueles referentes à tarefa. O primeiro conjunto de atributos define a frequência de operação do IP, seu custo e a área que este ocupa. O segundo conjunto de atributos define o tempo de execução do IP e seu consumo de energia. Note que a frequência de operação, o custo e área de um IP coincidem com a frequência de operação, o custo e área do processador que lhe é associado, respectivamente. Observe também que o tempo de execução de um IP e seu consumo de energia coincidem com o tempo e consumo correspondente à execução da tarefa que lhe é associada. Além disso, o identificador do processador e da tarefa são também incluídos como atributos de uma *tag* <ip> para identificar o processador de referência e a tarefa executada.

A Figura 39 mostra um extrato do documento XML utilizado para representar o repositório de IPs. Internamente o repositório é guardado na forma de tabela *hash*, que permite que este seja indexado diferentes chaves. No contexto desta etapa de alocação de IPs, o repositório pode ser indexado pelo identificador do IP e também pelo identificador do tipo da tarefa.

4.2.1.2 Codificação de grafos de tarefas

Assim como foi feito no caso do repositório de IPs, o grafo de tarefas foi também estruturado em um documento XML. Como um grafo de tarefas possui dois elementos principais, o nó e a aresta, estes formam as *tags* principais do documento XML. A Figura 40 mostra o documento XML utilizado para representar o grafo de tarefas da Figura 36. As *tags* <*node*> e <*edge*> representam os nós e as aresta do grafo de tarefas em questão, respectivamente. Os nós e as arestas possuem identificadores distintos. Um nó tem como atributo um identificador de tipo de tarefa e o nível (ou profundidade) no grafo. Seja t uma tarefa que depende de n outras tarefas t_1, t_2, ... e t_n. O nível associado à tarefa t é o calculado a partir do máximo dos níveis

```
<?xml version="1.0" encoding="UTF-8"?>
<repository>
  <ips>
    <ip id="204" type="2"   taskTime="9.2E-5" taskPower="2.0"
                 procID="6" maxFreq="2.66E8" price="65.0"  area="7.1823997E-6"/>
    <ip id="284" type="16" taskTime="0.0016" taskPower="1.2"
                 procID="8"  maxFreq="1.0E8"  price="16.0"  area="2.9929001E-6"/>
    <ip id="429" type="42" taskTime="0.031"  taskPower="2.5"
                 procID="14" maxFreq="1.67E8" price="30.0"  area="1.91844E-5"/>
    <ip id="466" type="45" taskTime="1.0E-5" taskPower="1.0"
                 procID="30" maxFreq="4.0E7"  price="45.0"  area="1.0000001E-6"/>
    <ip id="250" type="2"   taskTime="5.6E-4" taskPower="6.0"
                 procID="7" maxFreq="5.0E8"  price="210.0" area="9.604E-5"/>

    . . .
   </ips>
</repository>
```

Figura 39: Representação em XML do repositório de IPs

das tarefas t_1, t_2, ... e t_n incrementado de 1. Por exemplo, no grafo de tarefa apresentado na Figura 40, como a tarefa 3 depende das tarefas 1 e 2, esta terá o nível de $3 = 2+1$. Através deste identificador, é possível restringir os IPs do repositório que podem ser usados para implementar a tarefa representada por este nó. As arestas têm como atributos dois identificadores, sendo um do nó de origem e outro do nó destino. Um outro atributo das arestas representa o volume de dados a serem enviados, em número de *bits*.

```
<?xml version="1.0" encoding="UTF-8"?>
<taskgraph>
   <nodes>
      <node id="0" type="16" level="0"/>
      <node id="1" type="42" level="1"/>
      <node id="2" type="2"  level="2"/>
      <node id="3" type="45" level="3"/>
   </nodes>
   <edges>
      <edge id="0" src="0" tgt="1" volume="500"/>
      <edge id="1" src="1" tgt="3" volume="400"/>
      <edge id="2" src="1" tgt="2" volume="350"/>
      <edge id="3" src="2" tgt="3" volume="120"/>
   </edges>
</taskgraph>
```

Figura 40: Representação em XML do grafo de tarefas

4.2.1.3 Codificação dos indivíduos

Nesta etapa, uma solução, na forma de um cromossomo, deve associar um conjunto de IPs às tarefas do GT da aplicação. A Figura 41 ilustra a codificação do cromossomo utilizada. A posição do gene no cromossomo coincide com o identificador da tarefa no GT. Considerando uma aplicação de n tarefas, então, o gene na posição 0 do cromossomo corresponde à tarefa cujo identificador é 0, o gene 1 corresponde à tarefa cujo identificador é 1 e assim por diante, até o gene $n-1$ que corresponde à tarefa de identificador $n-1$. Cada gene carrega consigo três dados distintos. São eles: o tipo da tarefa correspondente, identificador do IP alocado e a informação sobre a sua dedicação.

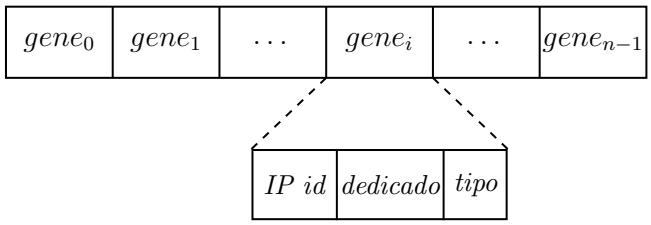


Figura 41: Codificação do cromossomo para a alocação de IPs

Os campos de dados que formam um gene são compatíveis com seus homólogos no repositório de IPs. Em um gene g, o *identificador de IP* (0 à 479) corresponde à um IP do repositório capaz de executar a tarefa associada à g e o *tipo de tarefa* identifica uma das 46 tarefas usadas no repositório. Se a tarefa requer um IP dedicado, o campo *dedicado* é igual a 1, caso contrário é igual a 0. Lembre que um cromossomo juntamente com o grafo de tarefas representa um GCA da aplicação em consideração. De posse de um gene formado somente do identificador do IP já é suficiente para acessar todos os respectivos atributos, inclusive o tipo de tarefa que este implementa. No entanto, o tipo de tarefa é também incluído no gene para permitir uma identificação direta deste campo que [e consultado frequentemente. Tendo o tipo de tarefa diretamente disponível no gene, um único acesso ao repositório é necessário ao invés de dois, sendo um usando o identificador do IP para obter o tipo de tarefa e o outro usando o tipo assim obtido para ter os identificadores de todos os IPs que podem ser usados para implementar aquele tipo de tarefa.

4.2.2 Operadores genéticos

O operador de seleção utilizado é o operador de seleção por torneio, que funciona da seguinte maneira: Uma quantidade t de indivíduos é selecionada aleatoriamente da população. Como

em um verdadeiro torneio, estes t indivíduos competem entre si, sendo que o mais apto vence o torneio. O indivíduo vencedor é então selecionado. Este processo é repetido até que a população de indivíduos selecionados atinja o tamanho desejado. O parâmetro t é chamado de *tamanho do pool*. Quando o torneio é realizado entre dois indivíduos, o método é chamado de *torneio binário*. Este operador de seleção foi escolhido porque a perda de diversidade da população pode ser controlada através do número de indivíduos do torneio (BLICKLE; THIELE, 1995).

Com a codificação de cromossomo utilizada, conforme mostra a Figura 41, é necessário tomar cuidado para que as operações de recombinação e mutação gerem sempre soluções factíveis. A recombinação deve ocorrer de modo que a ordem dos genes não seja alterada, tendo em vista que, alterando-se a ordem dos genes, altera-se a semântica do grafo de tarefas e portanto, tal alteração não é permitida. Foi utilizado o operador de recombinação em um único ponto, cujo funcionamento está ilustrado na Figura 22 (do Capítulo 3). Este operador seleciona aleatoriamente um *locus*, em dois indivíduos progenitores e então os genes dos dois indivíduos, a partir deste *locus*, são trocados, dando origem a dois indivíduos filhos, onde a ordem dos genes é mantida.

A mutação deve ocorrer de modo que apenas IPs válidos, isto é, capazes de executar o tipo de tarefa associada ao gene, sejam escolhidos. A Figura 42 exemplifica o funcionamento do operador de mutação quando aplicado à um cromossomo que representa o grafo de tarefas da Figura 40 usando o repositório descrito na Figura 39. Um gene do cromossomo é escolhido aleatoriamente e o IP alocado neste gene é substituído por outro IP. A escolha do IP de substituição é aleatória dentro de um conjunto de IPs permitidos. Restringindo os IPs possíveis de acordo com o tipo de tarefa garante que as soluções geradas pelo operador de mutação são todas válidas. O campo representando a dedicação (1) ou compartilhamento (0) do IP pode também ser alterado, como mostra a Figura 42 para o gene em destaque.

4.2.3 Avaliação de aptidão

Durante o processo evolutivo de alocações de IPs, é preciso avaliar a aptidão dos indivíduos na população do momento. A avaliação é feita em termos dos objetivos de interesse. No caso do problema de alocação de IPs, os objetivos principais são: reduzir a área real, o custo da implementação, seu consumo de energia e aumentar o desempenho desta como um todo. Nesta etapa, o desempenho da implementação é avaliado como o tempo total de execução de todas as tarefas da aplicação, ignorando o tempo que será despendido na comunicação entre tarefas.

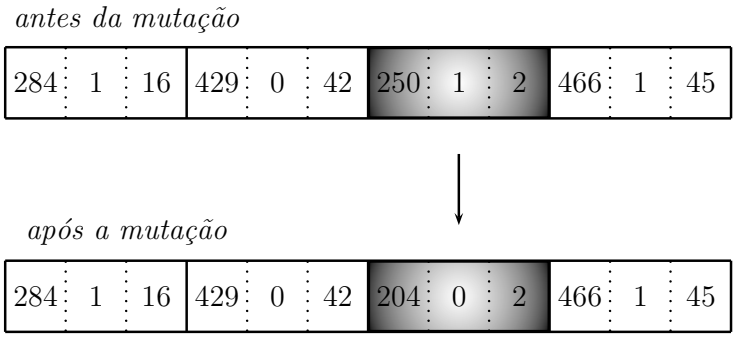


Figura 42: Ilustração do funcionamento do operador de mutação de alocações

Note que este tempo só pode ser estimado quando os IPs que implementam a aplicação foram selecionados e mapeados na infra-estrutura da plataforma NoC. A etapa de mapeamento será discutida no próximo capítulo.

Para reduzir a área real e o custo total, deve-se minimizar a quantidade de elementos processadores (EPs) utilizados, os tamanhos reais e o custo destes. Para aumentar o desempenho, deve-se minimizar o tempo de execução de cada tarefa que é inversamente proporcional à frequência de operação do processador usado. Para reduzir o consumo de energia deve-se minimizar o consumo de todos os processadores empregados.

Alguns objetivos considerados são *conflitantes* ou *concorrentes*, como é o caso da minimização do tempo de execução que está basicamente relacionada ao aumento da frequência de operação e este aumento, por sua vez, ocasiona um aumento no consumo de energia. Os objetivos não conflitantes são ditos *colaborativos*, como é o caso da redução de EPs que introduz uma redução da área real e portanto ocasiona uma redução do custo proporcional.

A avaliação de objetivos conflitantes é necessária para obter soluções que apresentam um bom balanço na eficiência, mas a avaliação dos objetivos colaborativos significa realizar mais trabalho sem necessidade. Deve-se então selecionar cuidadosamente o conjunto mínimo de objetivos a serem considerados durante o processo de otimização multiobjetivo. A Tabela 5 ajuda a escolher este conjunto para o problema de alocação de IPs.

Na Tabela 5, as setas na diagonal (i, i) indicam se o objetivo deve ser minimizado (seta para baixo) ou maximizado (seta para cima). As demais colunas da mesma linha i indicam o comportamento que deve ser imposto, i.e. minimização ou maximização, dos demais objetivos em virtude da otimização do objetivo referente à posição (i, i). Por exemplo, a linha 6 da Tabela 5 indica que a quantidade de EPs deve ser minimizada, gerando uma redução de área e custo. Pode-se observar que a área e o custo devem ser minimizados, logo, área, custo e

Tabela 5: Objetivos concorrentes e colaborativos

no.		Área	Custo	Frequência	Tempo	Consumo	#EPs
1	Área	↓	-	-	-	-	↓
2	Custo	-	↓	-	-	-	↓
3	Frequência	-	-	↑	↓	↑	-
4	Tempo	-	-	↑	↓	↑	-
5	Consumo	-	-	↓	↑	↓	-
6	#EPs	↓	↓	-	-	-	↓

quantidade de EPs são objetivos colaborativos. Por um outro lado, o tempo de execução e a frequência de operação são colaborativos e ambos são concorrentes com o consumo de energia. Para obter soluções que apresentam um balanço adequado, levando em consideração a relação entre os objetivos, conclui-se que seja necessário considerar o conjunto mínimo formado pela área real, o tempo de execução e consumo de energia. A avaliação destes três objetivos é feita com o intuito de minimizar todos. A seguir são apresentadas as funções objetivo utilizadas para a avaliação das soluções evoluídas.

4.2.3.1 Área

Para avaliar a área necessária é preciso somar as áreas de cada processador usado na alocação. O identificador de cada processador alocado é obtido visitando os genes da solução S. Agrupando os nós de mesmo processador compartilhado e dedicados é possível identificar quais são os processadores do repositório que fazem parte da solução. A Equação 16 é utilizada para calcular a área de uma solução S. A função $\mathcal{PE}(S)$ retorna o conjunto de processadores utilizados na alocação S que executam mais de uma tarefa, i.e. compartilhados. Neste caso o campo *dedicado* é 0. A notação $S[t]_{ip}$ indica o IP alocado à tarefa t na solução S.

$$\acute{A}rea(S) = \sum_{t \in GT} \acute{a}rea_{S[t]_{ip}} * S[t]_{dedicado} + \sum_{p \in \mathcal{PE}(S)} \acute{a}rea_p \tag{16}$$

4.2.3.2 Tempo de execução

Para avaliar o tempo de execução imposto por uma alocação S, é preciso percorrer todos os caminhos do GCA que esta representa e encontrar o caminho que impõe o maior atraso. Este caminho é denominado de *caminho crítico*. O atraso imposto pelo caminho crítico é computado usando o método recursivo de busca em profundidade. Neste caminho, quando tarefas paralelas são alocadas para serem executadas pelo mesmo processador compartilhado, não é possível que

elas sejam executadas simultaneamente, logo, estas devem ser escalonadas sequencialmente. A ordem de escalonamento usada é ditada pela ordem crescente do identificador de tarefa. Neste contexto, considere o caso em que t_1, t_2, ..., t_k, são k tarefas que podem ser executadas em paralelo, mas todas estão alocadas ao mesmo processador compartilhado. O tempo de execução associado a um caminho que passa por uma tarefa t_i, é acrescido da soma dos tempos de execução de todas as tarefas que são escalonadas antes de t_i. Estas tarefas são aquelas cujo identificador é menor do que o identificador da tarefa t_i.

A Equação 17 mostra os detalhes deste cálculo. Neste contexto, a função $\mathcal{C}(g)$ retorna todos os caminhos possíveis do grafo de tarefas g, a função $\mathcal{P}(t)$ retorna o conjunto de todas as tarefas no GCA que podem ser executadas em paralelo com a tarefa t e que estão associadas ao mesmo processador na solução S, a função $\mathcal{D}(t)$ informa todas as tarefas das quais a execução de t depende e que também estão alocadas ao mesmo processador em S. Note que o atributo *level* dos nós de um grafo de tarefas pode ser usado para determinar os membros do conjunto retornado pela função $\mathcal{P}(.)$.

$$Tempo(S) = \max_{M \in \mathcal{C}(GT)} \left(\sum_{t \in M} tempo_{S[t]_{ip}} + \begin{cases} 0 & \textbf{se} \quad S[t]_{dedicado} = 1 \\ & \textbf{ou} \quad \mathcal{P}_{(t)} = \mathcal{D}_{(t)} = \emptyset \\ \sum\limits_{\substack{t' \in \mathcal{P}_{(t)} \cup \mathcal{D}_{(t)} \\ t' < t}} tempo_{S[t']_{ip}} & \textbf{senão} \end{cases} \right) \tag{17}$$

4.2.3.3 Consumo de energia

Para avaliar o consumo de energia total de uma aplicação representada por um GT, os consumos de energia dos IPs referenciados no GCA correspondente, são somados. Na Equação 18, $consumo_{(t,p)}$ representa o consumo de energia necessário para executar uma tarefa de tipo t usando o processador de identificador p.

$$Consumo(S) = \sum_{t \in GT} consumo_{S[t]_{ip}} \tag{18}$$

4.3 Resultados Experimentais

Nesta seção são apresentados os resultados obtidos pelas implementações do método de alocação evolutiva de IPs em plataforma NoC utilizando os algoritmos NSGA-II e microGA. Em um primeiro momento, são apresentados os resultados de desempenho dos dois. Em seguida, é feita uma comparação das características das implementações decorrentes das alocações geradas pelo NSGA-II e microGA. Esta comparação tem como objetivo comprovar que as alocações evoluídas

podem de fato conduzir a implementações eficientes de aplicações baseadas em plataforma NoCs.

Para a realização das simulações foi utilizado um conjunto de aplicações obtido de (DICK, 2008). Estas aplicações são referências (*benchmarks*) para testes envolvendo implementações de sistemas embutidos. As especificações completas destas aplicações, e seus respectivos grafos de tarefas em XML, encontram-se no Apêndice B. A Tabela 6 apresenta os detalhes destas aplicações, que estão listadas em ordem alfabética. As aplicações são associadas com o número total de tarefas, arestas e alocações possíveis. Note que este último número é calculado como descrito na Seção 4.1.3 e permite caracterizar a complexidade do problema de alocação de IPS para a respectiva aplicação.

Tabela 6: Características das aplicações do repositório do E3S e da complexidade dos problemas de alocação correspondentes

ID	Aplicação	#Tarefas	#Arestas	#Possíveis alocações
1	*auto-indust-tg0*	6	4	1.183.744
2	*auto-indust-tg1*	4	3	18.496
3	*auto-indust-tg2*	9	9	606.076.928
4	*auto-indust-tg3*	5	4	8.704
5	*consumer-tg0*	7	8	2.247.264
6	*consumer-tg1*	7	5	176.868
7	*networking-tg1*	4	3	41.616
8	*networking-tg2*	4	3	41.616
9	*networking-tg3*	4	3	41.616
10	*office-tg0*	5	5	210.681
11	*telecom-tg0*	4	4	56.644
12	*telecom-tg1*	6	6	9.516.192
13	*telecom-tg2*	6	6	9.516.192
14	*telecom-tg3*	3	2	4.046
15	*telecom-tg4*	3	2	3.468
16	*telecom-tg5*	2	1	238

4.3.1 Resultados do NSGA-II

Para os resultados expostos nesta seção, a implementação do NSGA-II foi configurada com os seguintes parâmetros: uma população de 850 indivíduos, taxa de mutação de 0,01, taxa de recombinação de 0,8, um número de gerações de 100. O torneio de seleção foi realizado entre 3 indivíduos. Os valores adequados desses parâmetros foram identificados após várias simulações.

A Tabela 7 e a Tabela 8 apresentam os resultados obtidos pelo NSGA-II para as aplicações do E3S. A Tabela 7 mostra, para cada aplicação, o número total de soluções não domi-

Tabela 7: Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o NSGA-II para as aplicações do E3S

ID	# Soluções	Médias de			# Processadores		
		Consumo[1]	Área[2]	Tempo[3]	Mínimo	Máximo	Médio
1	33	4,724	3,636	0,022	3	4	3,63
2	9	2,441	2,869	0,047	2	3	2,77
3	621	8,937	11,68	2,0	4	6	5,39
4	11	4,152	2,775	0,039	2	4	2,9
5	157	6,089	10,49	123,0	3	5	4,16
6	67	3,428	7,878	47,2	2	3	2,7
7	35	2,185	11,11	2,0	2	3	2,34
8	39	2,028	12,07	2,0	2	3	2,3
9	37	1,774	12,35	4,0	2	3	2,32
10	159	4,376	15,76	446,0	2	5	3,04
11	17	2,364	2,342	0,358	2	2	2
12	266	3,414	9,498	1,0	2	4	3,6
13	254	3,146	10,31	1,0	2	4	3,55
14	7	2,413	1,679	0,165	1	2	1,57
15	11	1,931	15,06	0,68	1	2	1,72
16	11	1,875	1,679	0,785	1	2	1,72

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

nadas, as médias de consumo de energia, área de *hardware* e tempo de execução, obtidos para estas soluções. A Tabela 7 inclui também a quantidade média, máxima e mínima de elementos processadores que foram alocados. A Tabela 8 apresenta os valores mínimos obtidos para cada objetivo de interesse. Note que esses valores não provêm necessariamente da mesma solução, mas indicam os valores mínimos encontrados em soluções não dominadas.

A Tabela 9 expõe as alocações não dominadas, encontradas pelo NSGA-II, para uma amostra das aplicações do E3S. Entre as 16 aplicações usadas, foram selecionadas 6 dos diferentes domínios com diferentes complexidades. As listas mostram os números de IPs de acordo com a Tabela 35 e a Tabela 36 do Apêndice A. Os números de IPs com (*) indicam o uso dedicado destes. A presença do símbolo × em uma das últimas colunas indica que a alocação indicada permitiu alcançar o valor mínimo em área (A), consumo de energia (C) e/ou tempo (T).

4.3.2 Resultados do microGA

Para os resultados expostos nesta seção, a implementação do microGA foi configurada com os seguintes parâmetros: uma memória populacional de 1.000 indivíduos, sendo 70% renovável e 30% não renovável, memória externa de 200 indivíduos, população de 4 indivíduos, um número

Tabela 8: Mínimos dos objetivos para as soluções não dominadas, encontradas pelo NSGA-II

ID	Mínimos de		
	Consumo$^{(1)}$	Área$^{(2)}$	Tempo$^{(3)}$
1	4,15	3,0000003	0,02216
2	2,15	2,0000002	0,0475
3	7,15	6,0000006	0,84514
4	3,15	2,0000002	0,03955
5	0,525	5,4651998	57,34
6	0,375	2,4884	14,52
7	0,3	2,4884	0,282
8	0,3	2,4884	0,311
9	0,3	2,4884	0,39
10	0,375	2,0000002	4,23
11	0,3	2,0000002	0,03456
12	0,45	2,0000002	0,21976
13	0,45	2,0000002	0,21976
14	0,225	1,0000001	0,029
15	0,225	1,0000001	0,13
16	0,15	1,0000001	0,073

^{1}(W), ^{2}($\times 10^{-6}$m^2), ^{3}($\times 10^{-3}$s)

de gerações de 3.000, ciclos de recolocação realizados a cada 200 gerações, uma convergência nominal a cada 4 gerações e grade adaptativa com bisseção de 5. O método de seleção utilizado foi o torneio binário. Os valores adequados desses parâmetros foram identificados após várias simulações.

A Tabela 10 apresenta os resultados obtidos pelo microGA para as aplicações do E3S, sendo estes, o número total de soluções não dominadas e as médias de consumo de energia, área de *hardware* e tempo de execução obtidas para estas soluções. A Tabela 10 inclui também a quantidade média, máxima e mínima de elementos processadores que foram alocados. Os valores mínimos obtidos para cada objetivo de interesse para as soluções evoluídas pelo microGa são iguais àqueles alcançados pelo NSGA-II, exceto para o consumo de energia das aplicações 12 e 13 (i.e. *telecom-tg1* e *telecom-tg3*, que foram 0,3 e 0,225 ao invés de 0,45 e 0,224 no caso do NSGA-II.

Como foi feito para o NSGA-II, a Tabela 11 expõe as alocações não dominadas, encontradas pelo microGA, para uma amostra das aplicações do E3S. Entre as 16 aplicações usadas, foram selecionadas 6 dos diferentes domínios e com diferentes complexidades. Como antes, a presença do símbolo $\times$ em uma das últimas colunas indica que a alocação indicada permitiu alcançar o valor mínimo em área (A), consumo de energia (C) e/ou tempo (T).

Tabela 9: Amostra de alocações não dominadas, encontradas pelo NSGA-II, apresentando valores mínimos nos objetivos, conforme listado na Tabela 8

ID	Alocação não dominada	C	A	T
1	[201, 370, 371, 370* 382, 201]	×		×
	[369, 370, 371* 370, 382, 369]	×	×	
3	[462, 375* 380* 379, 376* 370, 383* 384, 462]	×		
	[369, 375* 380, 379* 376* 370* 383, 384, 369]		×	
	[462, 375* 380, 379* 6*, 370, 383* 384, 462]			×
5	[462, 456* 456* 456, 458* 454* 462]	×		
	[369, 456* 456, 456, 458* 454, 369]		×	
	[462, 241* 241, 241* 243, 239, 462]			×
7	[462, 434, 435, 462*]	×		
	[462, 434, 435, 369]		×	
	[462 345 346 462]			×
10	[462, 461* 462, 460* 459]	×		
	[201, 368, 201, 367, 366]		×	
	[462, 96, 462, 245, 244]			×
12	[462* 438* 441, 447, 444* 462]	×		
	[201, 185, 466, 472, 469, 201]		×	
	[462* 438, 441, 472, 469, 462]			×

4.3.3 Comparação dos resultados

As execuções do NSGA-II e microGA foram repetidas igualmente e as soluções não dominadas de cada execução foram aproveitadas. Cada evolução do microGA foi aproximadamente sete vezes mais rápida do que cada evolução do NSGA-II.

Os melhores resultados obtidos para os objetivos avaliados por ambos os algoritmos são praticamente iguais. Os resultados médios desses objetivos, no entanto apresentam diferenças, como pode ser observado nos histogramas da Figura 43, para os objetivos de interesse: consumo de energia, área e tempo de execução.

A Figura 44(a) permite comparar o número de soluções não dominadas encontradas por ambos os algoritmos, sendo que o microGA obteve mais soluções não dominadas que o NSGA-II para todas as aplicações do E3S. A Figura 44(b) mostra as diferenças entre o tempo gasto pelo NSGA-II e o microGA na busca das soluções com os menores valores em relação aos objetivos avaliados. Desses histogramas, percebe-se que o tempo de busca referente ao microGA foi menor do que aquele do NSGA-II para 12 das 16 aplicações e o microGA encontrou mais soluções do que NSGA-II em todos os casos. Isso demonstra o desempenho superior do microGA sobre o NSGA-II para o problema de alocação de IPs. Para consolidar esta observação, foram calculados os tempos aproximados, gastos por cada algoritmo, na geração de uma única solução, conforme mostra a Tabela 12. O fator de desempenho de cada algoritmo

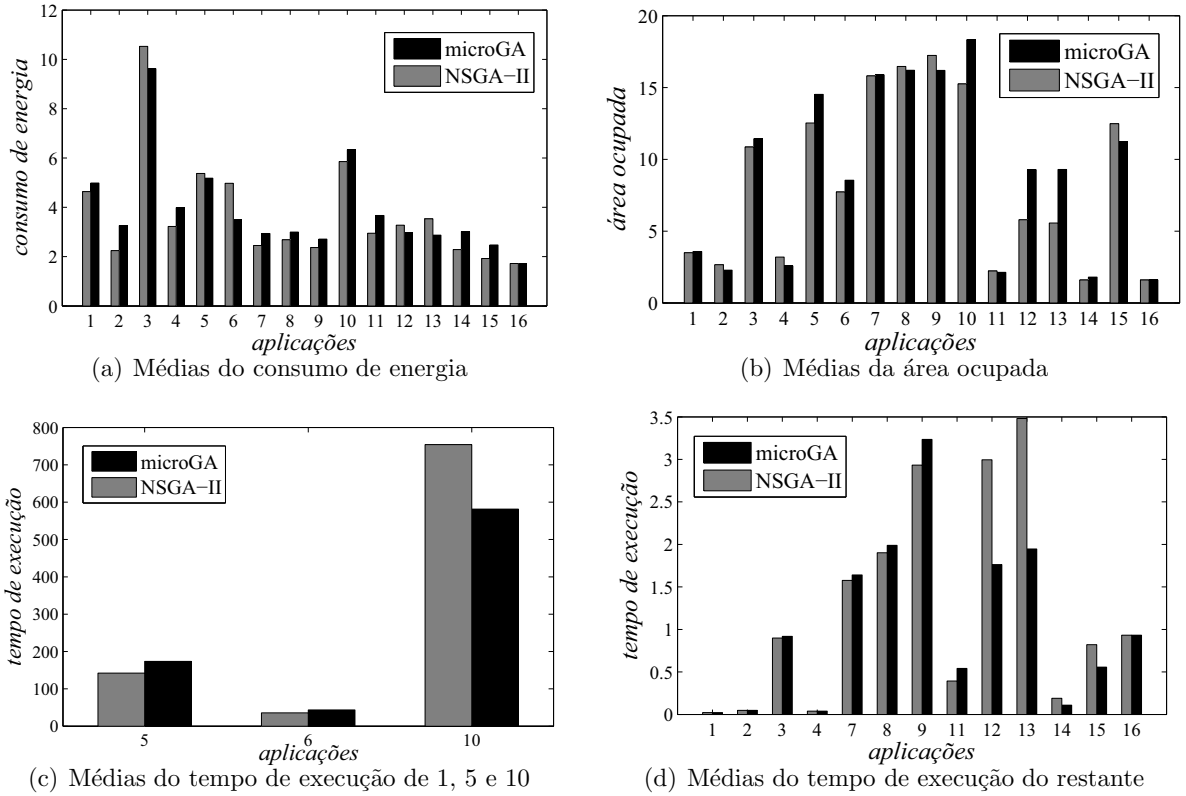
Tabela 10: Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o microGA para as aplicações do E3S

ID	# Soluções	Médias de			# Processadores		
		Consumo[(1)]	Área[(2)]	Tempo[(3)]	Mínimo	Máximo	Média
1	47	4,9842	3,5775	0,0222	3	4	3,66
2	17	3,2621	2,2824	0,0475	2	3	2,47
3	1435	9,6210	11,4307	0,9188	3	9	6,01
4	13	3,9842	2,5775	0,0396	2	3	2,7
5	1.050	5,1805	14,5164	173,5787	3	7	4,51
6	150	3,5021	8,5436	43,7476	2	5	2,96
7	95	2,9394	15,8808	1,6411	2	4	2,32
8	92	2,9983	16,1959	1,9888	2	4	2,35
9	103	2,7122	16,1860	3,2344	2	4	2,35
10	1010	6,3370	18,3256	581,5330	2	5	3,58
11	51	3,6680	2,1332	0,5421	2	2	2,00
12	1.169	2,9635	9,2769	1,7633	2	6	3,85
13	1.158	2,8722	9,2966	1,9458	2	6	3,84
14	20	3,0145	1,7977	0,1092	1	2	1,85
15	26	2,4738	11,2321	0,5567	1	2	1,88
16	11	1,7230	1,5954	0,9318	1	2	1,72

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

Tabela 11: Amostra de alocações não dominadas, encontradas pelo microGA, apresentando valores mínimos nos objetivos, conforme listado na Tabela 8

ID	Alocação não dominada	C	A	T
1	[462, 370, 371, 370, 382* 462]	×		×
	[201, 370* 371, 370, 382, 201]		×	×
3	[462, 375, 380* 379* 376* 370, 383* 384, 462]	×		
	[369, 375, 380* 379* 376, 370* 383, 384* 369]		×	
	[462, 375, 380* 379* 6*, 370, 383* 384, 462]			×
5	[462* 456* 456, 456* 458, 454, 462*]	×		
	[369, 456* 456, 456, 458* 454, 369]		×	
	[479, 241* 241, 241* 243, 239, 479]			×
7	[462* 434, 435, 462]	×		
	[462, 434, 435, 479*]		×	
	[462, 345, 346, 462]			×
10	[462* 461, 462, 460, 459*]	×		
	[386, 368, 386, 367, 366]		×	
	[462, 96, 462, 245, 244]			×
12	[462* 438, 441, 447* 444* 462]	×		
	[201, 185, 466, 472, 469, 201]		×	
	[462, 463, 441, 472, 469, 462]			×



(a) Médias do consumo de energia

(b) Médias da área ocupada

(c) Médias do tempo de execução de 1, 5 e 10

(d) Médias do tempo de execução do restante

Figura 43: Representação dos valores médios dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S

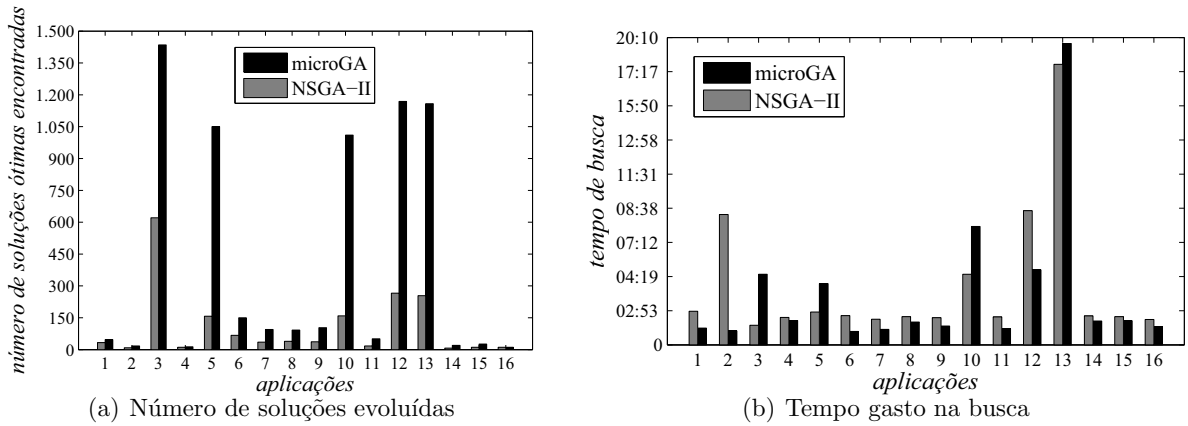
foi avaliado em relação ao tempo gasto na busca e à quantidade de soluções encontradas. Este fator de desempenho expressa a quantidade de soluções que um algoritmo é capaz de encontrar no tempo em que o outro algoritmo gasta para encontrar uma solução. A Tabela 13 apresenta os fatores de desempenho do NSGA-II e do microGA para cada uma das aplicações. A Figura 45 permite comparar esses fatores.

Muitas das soluções encontradas por ambos os algoritmos tratam-se de alocações diferentes que possuem a mesma avaliação de objetivos. Em relação ao conjunto de objetivos avaliados, estas soluções não apresentam diferença. Graficamente, estas soluções representam o mesmo ponto no espaço de objetivos, o que reduz a quantidade de pontos distintos. Este fato, aliado com o fato do espaço de busca representado pelo repositório ser discreto, dificulta a obtenção de uma fronteira de Pareto bem definida. A Figura 46 mostra a nuvem de pontos da fronteira ótima de Pareto para uma amostra de aplicações, no espaço de objetivos. As soluções não dominadas obtidas somente pelo NSGA-II, estão representadas por losangos vermelhos e somente pelo microGA, por pontos azuis. As soluções obtidas por ambos os algoritmos estão representadas por quadrados pretos.

Tabela 12: Tempo gasto aproximadamente pelos dois algoritmos na geração de uma única alocação

ID	#Alocações		Tempo de busca*		Tempo por solução*	
	NSGA-II	microGA	NSGA-II	microGA	NSGA-II	microGA
1	33	47	02:12,0	01:06,0	00:04,0	00:01,4
2	9	17	08:33,0	00:55,0	00:57,0	00:03,2
3	621	1.435	01:17,0	04:38,0	00:00,1	00:00,2
4	11	13	01:48,0	01:36,0	00:09,8	00:07,4
5	157	1.050	02:09,0	04:01,0	00:00,8	00:00,2
6	67	150	01:55,0	00:53,0	00:01,7	00:00,4
7	35	95	01:41,0	01:01,0	00:02,9	00:00,6
8	39	92	01:51,0	01:30,0	00:02,8	00:01,0
9	37	103	01:47,0	01:14,0	00:02,9	00:00,7
10	159	1010	04:38,0	07:46,0	00:01,7	00:00,5
11	17	51	01:50,0	01:04,0	00:06,5	00:01,3
12	266	1.169	08:48,0	04:55,0	00:02,0	00:00,3
13	254	1.158	18:24,0	19:46,0	00:04,3	00:01,0
14	7	20	01:54,0	01:34,0	00:16,3	00:04,7
15	11	26	01:51,0	01:38,0	00:10,1	00:03,8
16	11	11	01:40,0	01:11,0	00:09,1	00:06,5

*mm:ss,ms



(a) Número de soluções evoluídas

(b) Tempo gasto na busca

Figura 44: Número de soluções não dominadas encontradas pelo NSGA-II e microGA, para as aplicações do E3S, e seus respectivos tempos de busca

Tabela 13: Desempenho do NSGA-II e do microGA em relação à quantidade de soluções encontradas e o tempo de busca aproximado para cada solução

ID	NSGA-II	microGA
1	0,35	2,85
2	0,06	17,62
3	1,56	0,64
4	0,75	1,33
5	0,28	3,58
6	0,21	4,86
7	0,22	4,49
8	0,34	2,91
9	0,25	4,03
10	0,26	3,79
11	0,19	5,16
12	0,13	7,87
13	0,24	4,24
14	0,29	3,47
15	0,37	2,68
16	0,71	1,41

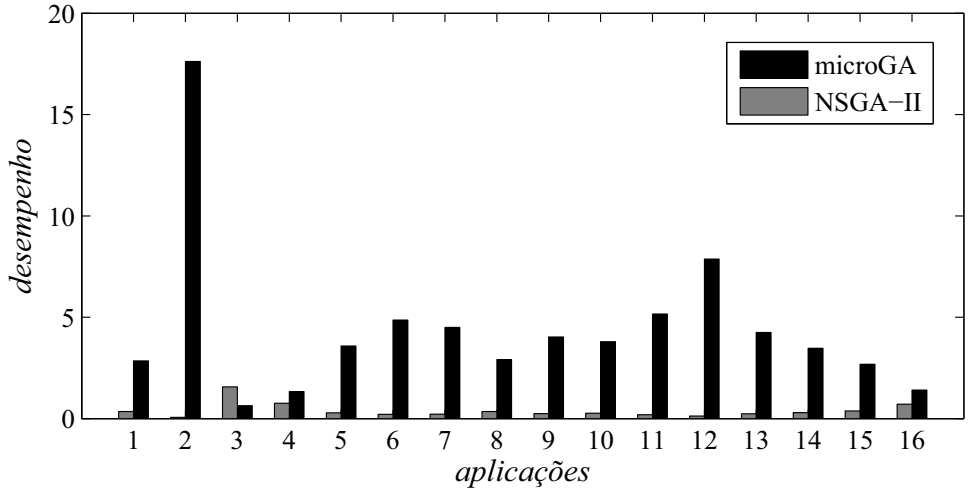
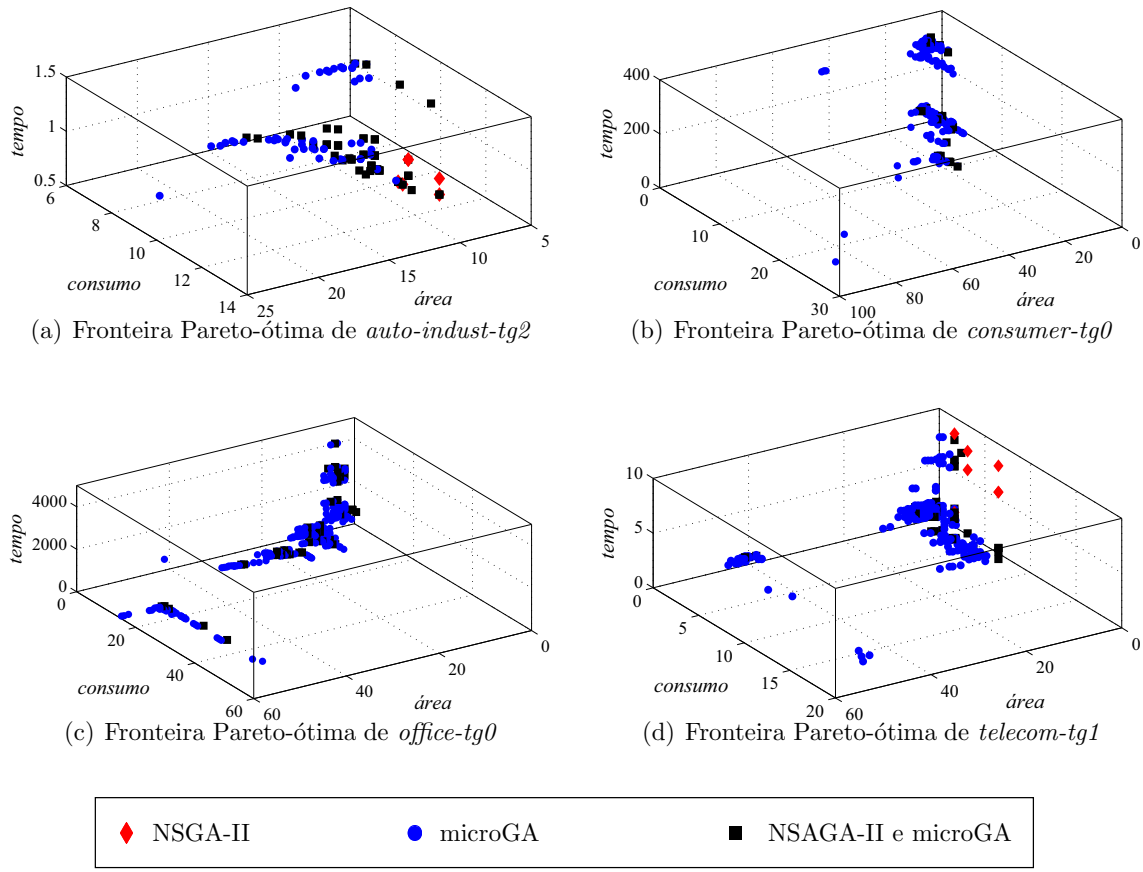


Figura 45: Comparação de desempenho entre o NSGA-II e o microGA de acordo com os dados da Tabela 13



(a) Fronteira Pareto-ótima de *auto-indust-tg2*

(b) Fronteira Pareto-ótima de *consumer-tg0*

(c) Fronteira Pareto-ótima de *office-tg0*

(d) Fronteira Pareto-ótima de *telecom-tg1*

Figura 46: Representação das fronteiras ótimas de Pareto para as aplicações 3, 5, 10 e 12

O resultado da alocação junto com o grafo de tarefas de uma aplicação representa o grafo de caracterização de aplicação correspondentes. A Figura 47 demonstra um dos GCAs obtidos para a aplicação *auto-indust-tg2* usando o NSGA-II enquanto que a Figura 47 monstra um dos GCAs obtidos para a mesma aplicação usando o microGA. Note que esta aplicação tem a maior complexidade entre as aplicações do E3S. Cada tarefa do GT é identificada pela dupla (*identificador do IP/ identificador do processador*). Estas soluções apresentam as seguintes características: consumo = 13,2 W, área = 10,1824×10^{-6} m^2 e tempo = 0.85266×10^{-3} s, e consumo = 13,2 W, área = 10,1824×10^{-6} m^2 e tempo = 0.99×10^{-3} s, para o NSGA-II e microGA, respectivamente.

4.4 Considerações Finais do Capítulo

A alocação de IPs em plataforma NoC consiste em determinar IPs responsáveis por executar uma aplicação em uma plataforma NoC. Para que os IPs possam ser alocados em tarefas é preciso, inicialmente, dividir a aplicação em tarefas. Esta divisão pode ser representada através

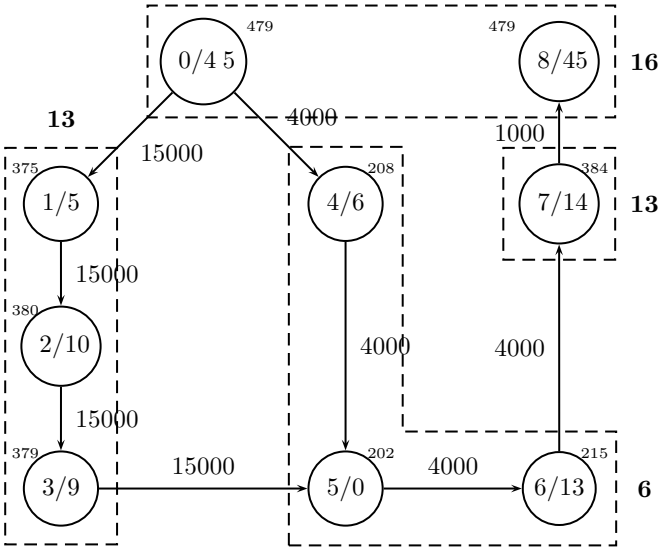


Figura 47: CGA da aplicação *auto-indust-tg2* obtido pela alocação evolutiva usando NSGA-II

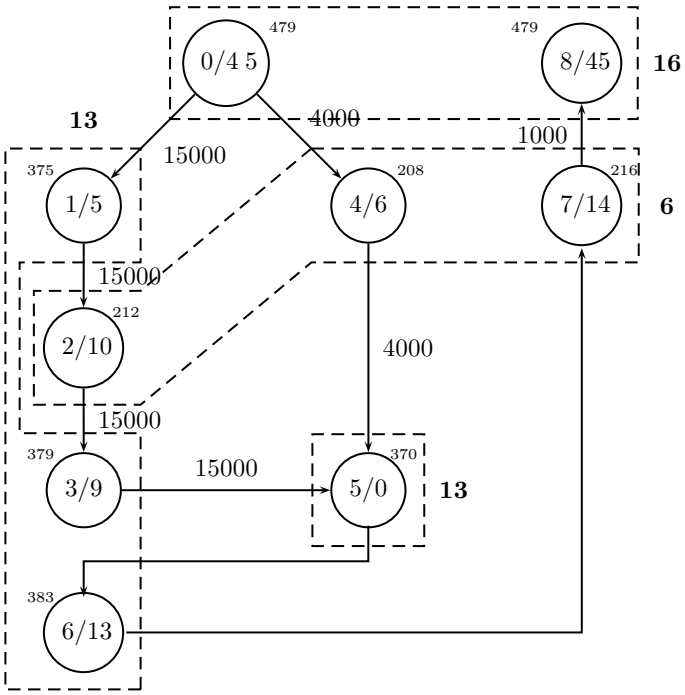


Figura 48: CGA da aplicação *auto-indust-tg2* obtido pela alocação evolutiva usando microGA

de um grafo de tarefas, que inclui as tarefas e suas dependências de dados e controle. Neste capítulo, foi apresentado um modelo para representar um grafo de tarefas e um repositório de IPs utilizando XML. O repositório consiste de uma biblioteca contendo a descrição dos diferentes IPs disponíveis para alocação. No processo de alocação, os IPs mais adequados à aplicação devem ser selecionados. A verificação de todas as combinações possíveis de IPs pode levar muito tempo, dependendo da complexidades da aplicação e da abrangência do repositório. Neste capítulo foi apresentado o método de alocação evolutiva de IPs utilizando os algoritmos evolucionários multiobjetivos NSGA-II e microGA. Os resultados obtidos mostraram que os dois algoritmos conseguem evoluir soluções de boa e compatível qualidade. No entanto, foi constatado que o microGA é capaz de obter um conjunto de soluções não dominadas mais extenso, com relação àquele obtido pelo NSGA-II.

Após a alocação de tarefas, o grafo de tarefas é transformado em um grafo de IPs, passando a se chamar de grafo de caracterização de aplicação. O grafo de caracterização de aplicação será utilizado para obter um mapeamento dos IPs da aplicação em uma plataforma NoC. O mapeamento consiste em determinar a alocação física dos IPs selecionados nesta plataforma. O próximo capítulo apresentará o problema de mapeamento e o método de mapeamento evolutivo de IPs na sua resolução.

Capítulo 5

MAPEAMENTO EVOLUTIVO EM PLATAFORMA NOC

NO Capítulo 1 foram apresentadas as etapas de um projeto de síntese de uma plataforma NoC. No capítulo anterior foi apresentada a etapa de alocação de tarefas. Foi apresentado um modelo para representar as tarefas de uma aplicação através de um grafo de tarefas (GT), um modelo de repositório de IPs e um método evolutivo de alocação de tarefas. Três objetivos de interesse foram destacados (consumo de energia, tempo de execução e área ocupada) e a alocação de tarefas foi guiada pela otimização destes objetivos. O resultado obtido foi um grafo de caracterização de aplicação (GCA) que é uma associação entre o repositório de IPs e o GT.

Após a etapa de alocação de tarefas, a etapa seguinte do projeto de síntese de uma plataforma NoC é a etapa de mapeamento. Após selecionar os melhores recursos do repositório de IPs, em função das características do GT e do próprio repositório, é preciso selecionar a melhor localização para estes recursos na implementação física da NoC, a esta seleção dá-se o nome de *mapeamento*. Enquanto que a alocação de tarefas não considera aspectos físicos, o mapeamento leva em consideração as distâncias entre os núcleos da rede, o roteamento e o impacto causado pela plataforma de comunicação.

Este capítulo apresenta o problema de mapeamento na Seção 5.1, juntamente com a sua complexidade. O modelo de energia utilizado para avaliar o consumo de energia referente à comunicação é discutido na Seção 5.2. O modelo de tempo utilizado para avaliar o tempo gasto em comunicação é detalhado na Seção 5.3, juntamente com o impacto do algoritmo de roteamento escolhido e a técnica de chaveamento aplicada. O método evolutivo de mapeamento proposto é apresentado na Seção 5.4, detalhando a codificação e as funções de avaliação de aptidão das soluções. Além das aplicação da E3S, uma aplicação de processamento de imagens é utilizada para avaliar o método proposto, e é apresentada na Seção 5.5.3.2. Os resultados

experimentais são comentados na Seção 5.5 juntamente com a análise e comparação com resultados obtidos de um outro sistema de mapeamento. A Seção 5.6 faz o fechamento deste capítulo com algumas considerações finais e introduz o assunto a ser abordado no próximo capítulo

5.1 O Problema de Mapeamento de IPs

Considerando a metodologias de projeto para uma plataforma NoC, o mapeamento de IPs é um dos principais problemas (OGRAS; HU; MARCULESCU, 2005). Informalmente, o problema de mapeamento consiste em mapear os IPs de um GCA em uma arquitetura de comunicação fixa, de modo que determinados objetivos de interesse sejam otimizados.

A arquitetura de comunicação é dada por $A(S, C)$, onde S é o conjunto de *switches* e C é o conjunto de canais de comunicação entre os *switches*. O GCA, como visto na Seção 4.1.4 do Capítulo 4, é dado por $G(I, D)$. O mapeamento de IPs é dado por $\mu : I \rightarrow S$. Um mapeamento μ pode ser considerado como uma função injetora, onde $\mu(i) = s$ indica que o IP $i \in I$ do GCA é mapeado em um recurso de comunicação $s \in S$ da Plataforma NoC. Os objetivos de interesse podem ser o custo da implementação, consumo de energia, tempo de execução, etc. As restrições de projeto são geralmente dados em termos da geometria da rede de interconexão e número máximo de IPs alocados no mesmo elemento processador da plataforma.

As restrições e os objetivos de interesse são de responsabilidade do projetista. Cada mapeamento deve ser avaliado com o propósito de identificar se as restrições foram atendidas e se os objetivos de interesse foram alcançados. A avaliação de um mapeamento pode ser feita de duas maneiras: *intrínseca* ou *extrínseca*. A avaliação intrínseca é realizada usando protótipos em *hardware* e tem como vantagem a grande precisão dos resultados obtidos, mas como grande desvantagem, esta abordagem demanda muito tempo para avaliar as diferentes soluções. A avaliação extrínseca é realizada usando simulações. Esta avaliação apresenta como vantagem a rapidez, mas como desvantagem, esta abordagem introduz uma margem de erro nos resultados obtidos.

Para reduzir o imprecisão dos resultados gerados por simulação, é preciso utilizar modelos adequados para o cálculo do consumo de energia e tempo de execução da aplicação para um mapeamento específico. Ambos os parâmetros são fortemente afetados pelo mapeamento adotado. Em seguida, serão descritos os modelos de estimativa do consumo de energia e tempo de execução da aplicação que serão utilizados nesta dissertação. Embora os modelos apresentados sejam capazes de gerar bons resultados em uma avaliação de alto nível, o desenvolvimento

de um modelo mais avançado, que contemple mais aspectos de *hardware* é ainda um problema parcialmente solucionado e tratado, hoje em dia, como pesquisa de ponta (OGRAS; HU; MARCULESCU, 2005).

5.1.1 Complexidade do problema de mapeamento de IPs

Neste contexto, o número de mapeamentos possíveis aumenta em uma escala fatorial de acordo com a quantidade de recursos da rede. O problema de mapeamento é um tipo de problema de atribuição quadrática, que é um conhecido caso de problema *NP*-Difícil (GAREY; JOHNSON, 1979). Este tipo de problema é no mínimo tão difícil de ser resolvido quanto um problema *NP*-completo.

Considere uma aplicação cujo grafo inclui m tarefas, t_0, t_1, ..., t_{m-1} e uma plataforma NoC de n recursos. Seja ℓ o número de IPs usados na alocação a ser mapeada na NoC. O número de diferentes mapeamentos possíveis, B_1, depende do número de recursos disponíveis na NoC e também varia com a dedicação dos IPs. Para os ℓ IPs usados na alocação, seja p o número de processadores empregados, sendo que alguns desses são de uso dedicado e outros de uso compartilhado. No caso em que os IPs incluídos na alocação são todos dedicados, tem-se $m = p = \ell$ enquanto no caso de uso de IPs compartilhados, tem-se $p < m = \ell$.

Assume-se que a plataforma NoC tenha recursos suficientes para a implementação da aplicação, usando a alocação selecionada. Portanto, tem-se $n \geq p$. Considerando uma estrutura de comunicação de tipo malha quadrada, o número mínimo de recursos n é, por consequência, definido conforme Equação 19. Neste caso, o número de mapeamentos possíveis referentes à uma alocação pode ser calculado conforme Equação 20. Note que tem-se $1 \leq M \leq n!$. O caso limite com $B_1 = 1$ acontece quando a plataforma NoC é formada por um único recurso e a alocação inclui um único IP e o outro caso limite $B_1 = n!$ acontece quando o número de tarefas, IPs, processadores e recursos coincidem, sendo que $m = \ell = p = n$. Para ilustração, o número de alocações factíveis para alguns valores de p e n é mostrado na Tabela 14.

$$n = \lceil \sqrt{p} \rceil^2 \tag{19}$$

$$B_1 = \frac{n!}{(n-p)!} \tag{20}$$

A etapa de mapeamento utiliza o resultado da alocação, que consiste de várias alocações não dominadas. Sendo assim, seja s o número das alocações distintas obtidas e p_i o número de processadores alocados na solução i e n_i o número mínimo de recursos na NoC para a implementação da aplicação utilizando a alocação i. Neste caso, o número total de mapeamentos a

Tabela 14: Ilustração da explosão exponencial de número de mapeamentos factíveis para uma alocação

Número de processadores (p)	Número de recursos (n)		
	4	9	16
2	12	72	240
3	24	504	3.360
4	24	3.024	43.680
5	–	15.120	524.160
6	–	60.480	5.765.760
7	–	181.440	57.657.600
8	–	362.880	518.918.400
9	–	362.880	4.151.347.200
10	–	–	2.905.9430.400

Tabela 15: Número total de mapeamentos possíveis para a aplicação e alocações mostradas na Figura 37 do Capítulo 4

Alocação (i)	Número de		
	processadores (p_i)	recursos (n_i)	Mapeamentos (B_1)
1	6	9	60.480
2	4	4	24
3	3	4	24
Número total de mapeamentos (B_3)			**60.528**

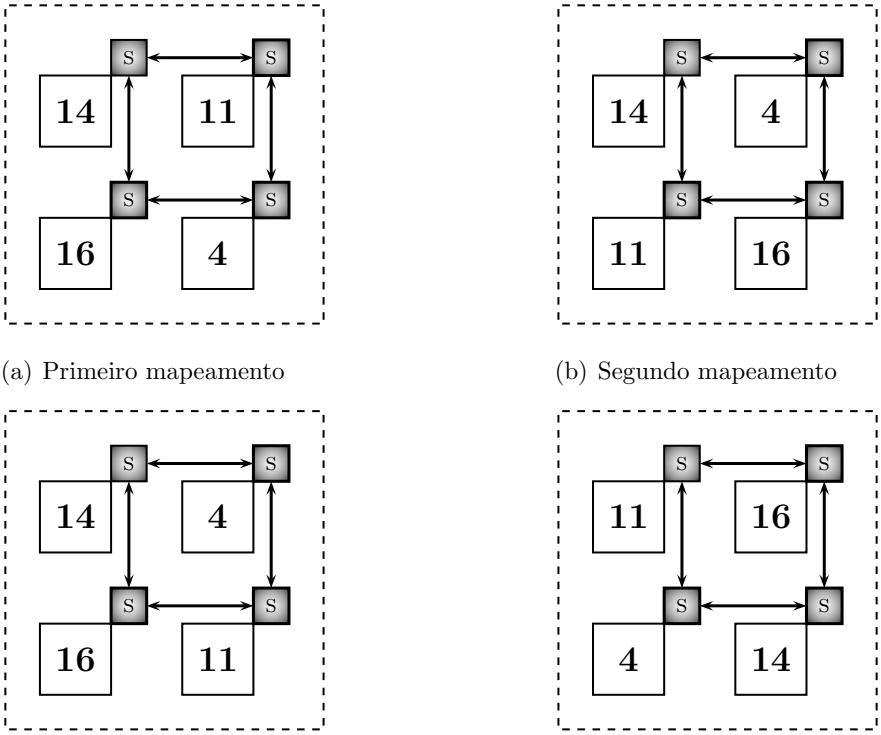
serem considerados é definido pela soma dos mapeamentos possíveis referentes à cada uma das alocações, conforme descrito na Equação 21. Por exemplo, considere as três alocações válidas da Figura 37 do Capítulo 4. O cálculo do número de mapeamentos possíveis é ilustrado na Tabela 15.

$$B_s = \sum_{i=1}^{s} \frac{n_i!}{(n_i - p_i)!} \tag{21}$$

5.1.2 Impacto do mapeamento de IPs

O mapeamento de IPs, alocados à tarefas de uma aplicação, na infra-estrutura de uma plataforma NoC, tem grande impacto no desempenho da aplicação. Para ilustrar isso, considere o grafo de tarefas da Figura 37(a) assim como a alocação da Figura 37(c) do Capítulo 4. O grafo tem 7 tarefas no total e o número de IPs alocados à cada uma dessas tarefas pode ser encontrado na Tabela 35 e na Tabela 36 do Apêndice A. As Figuras 49(a)–(d) apresentam quatro possíveis mapeamentos dos processadores usados.

Os n recursos de uma NoC são numerados sequencialmente linha a linha de 0 a $n-1$. Por consequência, um mapeamento pode ser denotado pela lista dos números de recursos onde



(a) Primeiro mapeamento

(b) Segundo mapeamento

(c) Terceiro mapeamento

(d) Quarto mapeamento

Figura 49: Impacto de diferentes mapeamentos de IPs

a tarefa é mapeada, sendo que a posição nesta lista corresponde ao identificador da tarefa. O primeiro mapeamento, na Figura 49(a), é um dos piores mapeamentos, já que os processadores 4 e 14 têm a comunicação mais intensa, de $15 + 10 + 12 = 37$, mas estão longe um do outro. O segunda mapeamento, na Figura 49(b), é melhor que o primeiro mapeamento já que os processadores 4 e 14 são mapeados lado a lado, mas em contra partida, os processadores 4 e 11, que têm a segunda comunicação mais intensa, de $10 + 15 = 25$, estão longe um do outro. O terceiro e o quarto mapeamentos, na Figura 37(c) e Figura 49(d), os custos de comunicação dos processadores 4 e 14 de um lado e 4 e 11 do outro são minimizados. A Tabela 16 apresenta o tempo de execução e o consumo de energia, em termos de computação e comunicação, para cada mapeamento. O procedimento que permite calcular o consumo de energia e o tempo de execução, devidos a um mapeamento específico, é detalhado na Seção 5.4.3.

5.2 Modelo de energia

O modelo de energia adotado foi desenvolvido originalmente para calcular o consumo de energia de *switches* em uma rede de comunicação (YE; MICHELI; BENINI, 2002). Para isto, foi definida a unidade *energia do bit*, denotada por E_{bit}, que representa a energia consumida por um *bit*

Tabela 16: Impacto dos mapeamentos ilustrados na Figura 49

No.	Mapeamento	Tempo[1]		Consumo[2]	
		Computação	Comunicação	Computação	Comunicação
1	[0, 3, 3, 3, 1, 2, 1]	31.93	181,07	53.64	7.3854E-10
2	[0, 1, 1, 1, 2, 3, 2]		144,07		6.386E-10
3	[0, 1, 1, 1, 3, 2, 3]		101,07		5,564E-10
4	[3, 2, 2, 2, 0, 1, 0]		101,07		5,564E-10

[1]($\times 10^{-3}$s), [2](W)

quando este trafega entre *switches*. O E_{bit} pode ser calculado através da Equação 22.

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \tag{22}$$

onde $E_{S_{bit}}$, $E_{B_{bit}}$ e $E_{W_{bit}}$ representam o consumo de energia do *switch*, *buffer* e dos fios de conexão, respectivamente.

Embora o modelo de energia representado pela Equação 22, tenha sido desenvolvido para *switches* que possuem um CI dedicado para realizar o roteamento de mensagens na rede, com algumas modificações é possível utiliza-lo para *switches* implementados em uma NoC. Para isso, as seguintes considerações devem ser feitas (HU; MARCULESCU, 2003):

- $E_{B_{bit}}$ foi calculado considerando-se que os *buffers* foram implementados em memórias SRAM ou DRAM. Quando ocorre congestionamento na rede, este parâmetro se torna dominante, já que acessos à memória são custosos em termos de consumo de energia. Em uma NoC, os *buffers* dos *switches* são implementados com registradores, que consomem menos energia do que memórias SRAM e DRAM.

- $E_{W_{bit}}$ representa a energia consumida pelos fios internos de conexão do *switch*. Em um *switch* de grande escala, estes fios representam um longo caminho para condução de corrente elétrica, gerando um alto consumo de energia. Em uma NoC, a conexão entre os *switches* é feita através dos canais e estes são de ordem milimétrica, no máximo.

A partir das considerações apresentadas anteriormente, algumas modificações devem ser feitas no modelo do E_{bit} para que este possa ser utilizado para modelar o consumo de energia de um *bit* em uma NoC.

O consumo de energia dos fios internos de conexão pode ser substituído pelo consumo de energia de um *bit* ao atravessar um canal ($E_{C_{bit}}$), resultando na Equação 23.

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{C_{bit}} \tag{23}$$

A energia consumida nos *buffers* é irrisória se comparada com a energia consumida nos canais de comunicação ($E_{B_{bit}} \ll E_{C_{bit}}$). Enquanto que a energia consumida nos *buffers*, em uma tecnologia de 0,18μm, é da ordem de fJ (10^{-15}J), a energia consumida nos canais de comunicação é da ordem de pJ ($10^{-12}J$), e portanto o fator $E_{B_{bit}}$ pode ser desprezado. Removendo $E_{B_{bit}}$, obtemos a Equação 24 para o cálculo da energia de um *bit*.

$$E_{bit} = E_{S_{bit}} + E_{C_{bit}} \tag{24}$$

Consequentemente, a energia consumida no envio de um *bit*, do recurso i para o recurso j, pode ser calculada através da Equação 25:

$$E_{bit}^{i,j} = \eta \times E_{S_{bit}} + (\eta - 1) \times E_{C_{bit}} \tag{25}$$

onde η é o número de *switches* e $\eta - 1$ é o número de canais, no caminho percorrido pelo *bit*.

O tempo consumido na execução de uma aplicação, devido à comunicação na rede, depende fortemente dos algoritmos de roteamento e chaveamento utilizados. A próxima seção descreve o modelo usado para calcular o tempo de comunicação, após de introduzir os algoritmos utilizados para rotear e chavear as mensagens na plataforma de NoC.

5.3 Modelo de Tempo de Comunicação

Além do tempo de computação gasto em cada núcleo da NoC, o tempo despendido em comunicação (ou tempo de troca de mensagens) também influencia no tempo total de execução da aplicação. O tempo de comunicação depende de aspectos físicos da rede e da lógica de roteamento e chaveamento adotada.

5.3.1 Roteamento e chaveamento

O algoritmo de roteamento define o caminho que será utilizado por cada pacote, desde o seu remetente até o seu destinatário. Este algoritmo é fortemente relacionado com a topologia da rede. A técnica de chaveamento estabelece uma conexão lógica entre o emissor e o receptor e determina como será realizada a troca de mensagens entre ambos (NI; MCKINLEY, 1993).

5.3.1.1 Algoritmos de roteamento

A responsabilidade pela definição do roteamento pode ser distribuída, quando cada *switch* determina o canal seguinte a ser utilizado; ou centralizada, quando o *switch* remetente define a rota inteira até o destinatário. Neste último caso, é necessário que a informação da rota seja embutida no pacote.

Os algoritmos de roteamento podem ser classificados como *determinísticos* ou *adaptativos*. No roteamento determinístico, para cada par (remetente, destinatário), existe apenas uma rota possível. Já no roteamento adaptativo, existem várias opções de caminhos. Nesta abordagem, o caminho de um pacote é estabelecido dependendo das condições da trafego e do estado de congestionamento da rede.

Um algoritmo de roteamento determinístico bastante utilizado em redes malha é o *algoritmo XY* (DUATO; YALAMANCHILI; NI, 2003). Neste algoritmo, o pacote é roteado inicialmente na direção horizontal (X) até atingir a coluna do recurso destinatário e em seguida é roteado na direção vertical (Y) até alcançar a linha do recurso de destino. A Figura 50 mostra a rota traçada pelo algoritmo XY em uma rede malha regular do nó (1,0) até o nó (2,2). Neste tipo de roteamento, a distância entre dois nós, i e j, pode ser calculada através da distância de *Manhattan*, que é dada a partir Equação 26.

$$d_{i,j} = \mid x_i - x_j \mid + \mid y_i - y_j \mid \tag{26}$$

onde (x_i, y_i) e (x_j, y_j) são as coordenadas (X,Y) dos nós i e j na malha, respectivamente.

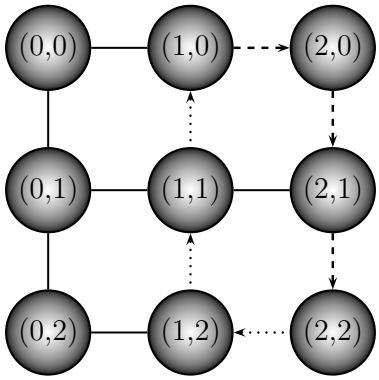


Figura 50: Rota do nó (1,0) até o nó (2,2), de acordo com o algoritmo XY

Um roteamento adaptativo estabelece rotas dinamicamente, de acordo com o estado da rede. Este tipo de roteamento pode ser *totalmente* ou *parcialmente* adaptativo. Usando um roteamento totalmente adaptativo, é possível rotear um pacote através de qualquer caminho físico existente, enquanto que no caso de um roteamento parcialmente adaptativo, regras restringem as possíveis escolhas de caminhos, como no caso do algoritmo *west-first* (BARATI *et al.*, 2008).

Um algoritmo de roteamento pode ser *mínimo* ou não. No primeiro caso, a cada roteamento o pacote deve sempre se aproximar do destino, enquanto no segundo caso, o pacote pode ser roteado para um caminho que o afaste momentaneamente de seu destino.

5.3.1.2 Técnicas de chaveamento

Existem basicamente duas técnicas de chaveamento utilizadas em redes de comunicação de dados: *chaveamento por circuito* e *chaveamento por pacote*.

- Chaveamento por circuito: Permite estabelecer um caminho fim-a-fim denominado de *conexão*, formando um circuito virtual. Somente após o formação do circuito que a mensagem começa a ser transferida.

- Chaveamento por pacote: Permite rotear cada pacote de acordo com os dados do destinatário. Como não é estabelecido um circuito virtual, cada pacote pode seguir por um caminho diferente.

A vantagem do chaveamento por circuito é que a mensagem é transmitida por um caminho sem congestionamento; entretanto, pode levar a um desperdício de canais ociosos assim que estes ficam inativos. O chaveamento por pacote não apresenta a desvantagem de subutilizar os canais da rede; entretanto, requer que a cada *switch* a conexão seja estabelecida com o *switch* seguinte para que o pacote seja enviado.

A exploração da técnica de chaveamento por pacote implica no uso de uma política de armazenamento de pacotes nos *switches*. As mais utilizadas são: *store-and-forward*, *virtual cut-through* e *wormhole* (NI; MCKINLEY, 1993).

- *Store-and-forward*: Cada pacote deve ser completamente armazenado antes de ser enviado para o *switch* seguinte. Além disso, o pacote só pode ser enviado caso o receptor autorize o envio. É necessário que os *switches* tenham grande capacidade de armazenamento para manter o fluxo da rede.

- *Virtual cut-through*: Não é necessário armazenar o pacote no switch antes do envio, mas este só pode ser enviado caso o receptor autorize o envio. Portanto, no pior caso, quando a autorização não chegue à tempo, é necessário armazenar todo o pacote, assim como na política do *sore-and-forward*.

- *Wormhole*: É uma variação do *virtual cut-through* que requer menor capacidade da estrutura de *buffers* nos *switches*. Para isso, os pacotes são divididos em unidades menores, chamadas de *flits* (*FLow control unITS*). Um *switch s* pode receber *flits* e enviar tão logo o receptor autorize. Em caso de indisponibilidade do receptor, não é necessário que todos os *flits* do pacote sejam recebidos pelo *switch s*. Os *flits* restantes são mantidos nos *buffers* dos *switches* anteriores ao *switch s*.

No restante desse capítulo, Analisa-se o tempo de comunicação com base em uma topologia regular em malha utilizando o algoritmo determinístico de roteamento mínimo XY e a técnica de chaveamento por pacote *wormhole*.

O modelo de tempo de comunicação é baseado no atraso total do chaveamento *wormhole*, que pode ser decomposto em *atraso de roteamento* e *atraso de carga útil* (MARCON *et al.*, 2005c). O atraso de roteamento é o tempo necessário para que o *flit* de cabeçalho do pacote trafegue do recurso de origem ao de destino. O atraso de carga útil é definido como o tempo transcorrido entre a chegada do *flit* de cabeçalho ao destino e a chegada do último *flit* do pacote ao destino. Portanto, este parâmetro varia de acordo com o número de *flits* de cada pacote.

As definições dos tipos de atraso são válidas para um *flit* de qualquer tamanho, geralmente chamado de um *phit* (*PHysical unIT*), onde *phit* representa a quantidade de *bits* que podem ser transmitidos simultaneamente nos canais de comunicação disponíveis na NoC. Sendo assim, o parâmetro *phit* é uma característica física da NoC que representa a largura dos canais de comunicação, enquanto que o *flit*, é uma unidade lógica, cujo tamanho é um valor múltiplo do *phit*. Assume-se, nesta dissertação, que um *flit* tem o tamanho de um *phit*.

Como o roteamento XY é mínimo e determinístico, definindo com precisão a rota utilizada pelos pacotes, é possível definir o número de *switches*, η, e o número de conexões, que coincide com $\eta + 1$ (entre *switches* e recursos e entre recursos) por onde o pacote irá passar. Sejam t_R o tempo necessário para a decisão de chaveamento em um *switch*, t_L o tempo necessário para transmitir um *flit* entre dois *switches* e t_l o tempo necessário para transmitir um *flit* entre um *switch* e o respectivo recurso local. O atraso de roteamento $\mathcal{T}_{R_{ij}}$ de um pacote, enviado pelo recurso r_i para o recurso r_j, considerando que a rota contem η *switches* e não há contenção de pacotes nos *switches*, é definido na Equação 27 (MARCON *et al.*, 2005c). Os atrasos básicos, assim definidos, são ilustrados na Figura 51.

$$\mathcal{T}_{R_{ij}} = \eta \times t_R + 2 \times t_l + (\eta - 1) \times t_L \tag{27}$$

Note que o termo $2 \times t_l$ é devido às comunicações entre recursos locais e seus respectivos *switches* na origem e no destino.

Nas implementações de NoCs conhecidas, os tempos de propagação entre *switches* e recursos locais e entre *switches* são compatíveis. Assim, assume-se que $t_l = t_L$, e portanto, a Equação 27 pode ser simplificada, conforme descrito na Equação 28.

$$\mathcal{T}_{R_{ij}} = \eta \times t_R + (\eta + 1) \times t_L \tag{28}$$

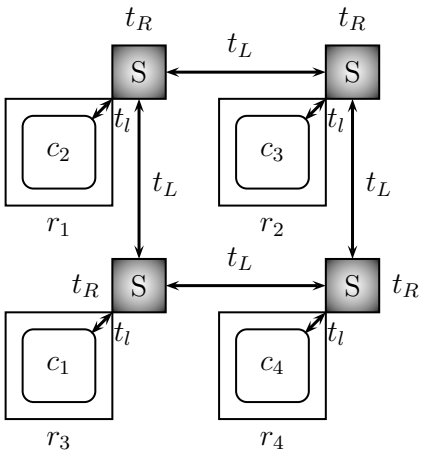


Figura 51: Ilustração dos atrasos básicos da rede: t_R, t_l e t_L

Seja nf_{ab} o número de *flits* de um pacote que parte do núcleo a, mapeado no recurso i, para o núcleo b, mapeado no recurso j. De forma análoga ao atraso de roteamento, o atraso de carga útil $\mathcal{T}_{P_{ij,ab}}$ é definido na Equação 29. Note que os *flits* são enviados em *pipeline*.

$$\mathcal{T}_{P_{ij,ab}} = (nf_{ab} - 1) \times t_L \tag{29}$$

Somando os dois atrasos de roteamento e de carga útil, o atraso total de um pacote, que parte do núcleo a, mapeado no recurso i, e vai até o núcleo b, mapeado no recurso j, é identificado como $\mathcal{T}_{ij,ab}$, e é dado pela soma de $\mathcal{T}_{R_{ij}}$ com $\mathcal{T}_{P_{ij,ab}}$, resultando na Equação 30:

$$\mathcal{T}_{ij,ab} = \mathcal{T}_{R_{ij}} + \mathcal{T}_{P_{ij,ab}} = (\eta \times (t_R + t_L) + nf_{ab} \times t_L) \tag{30}$$

A Figura 52 exemplifica o uso da Equação 30 para calcular o atraso total, em uma NoC 2×2, de um pacote que parte do núcleo c_1 mapeado no recurso r_3, e vai até o núcleo c_3 mapeado no recurso r_2, considerando $t_R = 2$, $t_L = 1$, $\lambda = 10$ ns e $n_{F_{137}} = 20$ *flits*. Para este caso, o atraso total é: $\mathcal{T}_{32,13} = (3 \times (2 + 1) + 20 \times 1) \times 10 = 290$ ns.

5.4 Mapeamento Evolutivo de IPs

O mapeamento evolutivo de IPs consiste em solucionar o problema de mapeamento de IPs, utilizando técnicas evolutivas. Como trata-se de um POM, as técnicas evolutivas utilizadas serão AGMs, assim como foi feito para resolver o problema de alocação de tarefas descrito no Capítulo 4. Os AGMs utilizados serão os mesmos utilizados anteriormente, i.e. o NSGA-II e o microGA.

Nesta etapa, a população inicial dos AGMs será composta pelas soluções não dominadas obtidas na etapa de alocação de IPs. O espaço de busca é composto pelos diversos mapeamentos

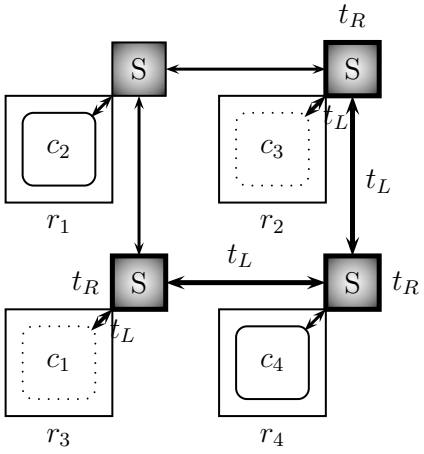


Figura 52: Rota para enviar o sétimo pacote que do recurso r_3 até o recurso r_2 indicando as contribuições dos *switches* e canais

possíveis considerando a arquitetura de comunicação da NoC e as alocações evoluídas. A Figura 53 ilustra a dinâmica do processo de mapeamento evolutivo.

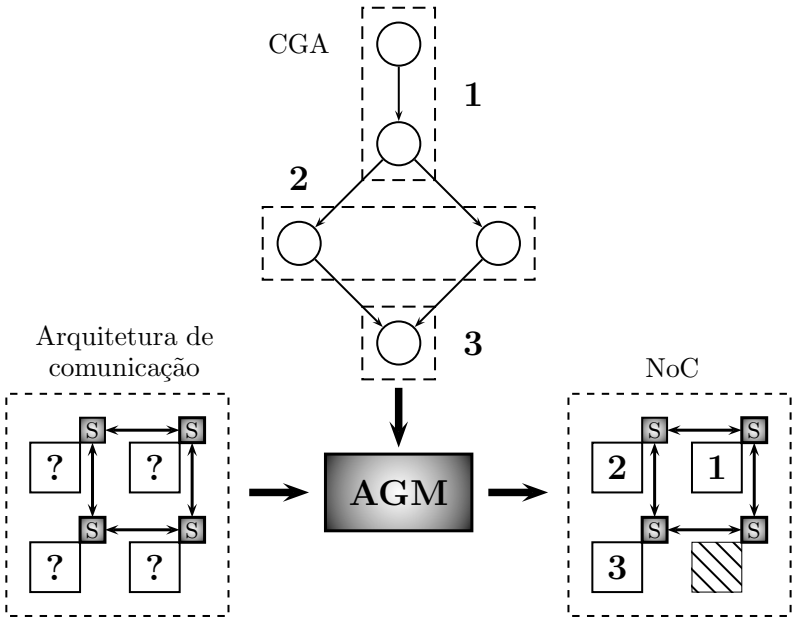


Figura 53: Dinâmica do processo de mapeamento evolutivo de IPs

A topologia de rede utilizada será a topologia malha quadrada, sendo que o número de recursos por linha e coluna são iguais. Como foi explicado anteriormente, na Seção 5.1.1, este número depende do número de processadores empregados na alocação que está sendo explorada. Note que, a população de um processo evolutivo pode conter mapeamentos com topologias diferentes, já que estes podem implementar alocações com diferentes números de processadores. Como foi explicado na Seção 5.1.1, o número total de recursos n é definido na

Equação 19. A arquitetura de comunicação resultante será formada por uma matriz $\sqrt{n} \times \sqrt{n}$ de n recursos e n *switches*.

No processo de mapeamento evolutivo de IPs, cada solução, representada por um indivíduo, deve ser codificada na forma de um cromossomo para que os operadores genéticos possam atuar nas soluções e as mesmas possam ser avaliadas. Nesta etapa, os indivíduos representam mapeamentos de uma alocação de IPs em uma plataforma NoC e são avaliados de acordo com aspectos físicos que influenciam o desempenho da aplicação mapeada.

5.4.1 Codificação

A codificação do indivíduo utilizada é uma extensão da codificação utilizada na etapa de alocação de IPs. A Figura 54 ilustra a codificação do cromossomo que forma cada indivíduo, representando o mapeamento de uma aplicação de m tarefas. Nesta codificação, além dos atributos utilizados na etapa de alocação de tarefas, é introduzido um novo atributo identificador do recurso onde o IP deveria ser mapeado. Lembre que a posição do gene identifica a tarefa mapeada.

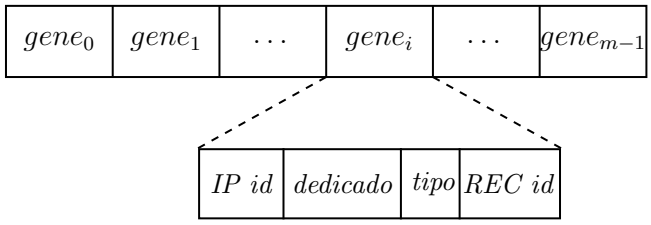


Figura 54: Codificação do cromossomo para o mapeamento de IPs

Na topologia de rede adotada, cada recurso é identificado por um índice inteiro e sequencial, iniciando a sequência do canto superior esquerdo e indo até o canto inferior direito, linha a linha. O primeiro recurso é identificado com o índice 0, assim como a linha no topo é a linha 0 e a coluna mais à esquerda é a coluna 0. A partir do índice do recurso, em uma rede malha $N \times N$, a linha do i-ésimo recurso é igual a $\lfloor i/N \rfloor$, que representa o maior inteiro menor ou igual a i/N e sua coluna é igual a $i \backslash N$, que representa o resto da divisão de i por N. Por exemplo, o recurso de índice 7, em uma rede em malha 3×3, está localizado na linha $\lfloor 7/3 \rfloor = 2$, coluna $7 \backslash 3 = 1$.

Na população inicial, os recursos são atribuídos aleatoriamente para cada solução. Sabendo a localização exata de cada tarefa na arquitetura de comunicação é possível determinar toda a rota de comunicação utilizada pela aplicação mapeada.

5.4.2 Operadores genéticos

O operador de seleção utilizado é o operador de seleção por torneio, o mesmo utilizado na etapa de alocação de tarefas. O funcionamento deste operador é apresentado com detalhes na Seção 3.1.2 do Capítulo 3. Mais uma vez, este operador de seleção foi escolhido porque a perda de diversidade da população pode ser controlada através do número de indivíduos do torneio (BLICKLE; THIELE, 1995).

Os operadores de recombinação e mutação, podem gerar soluções inválidas quando modificam os genes dos indivíduos. Em muitos problemas de busca e otimização, esta modificação genética não precisa ser controlada mas no caso do mapeamento de IPs, é preciso controlar a formação de novos indivíduos.

Como a população inicial é formada de indivíduos oriundos do processo evolutivo d alocação de IPs, é preciso se prevenir para que o trabalho de otimização já realizado não seja perdido. Não é possível trocar material genético entre indivíduos pois cada indivíduo representa uma solução ótima de alocação de tarefas. Diante da impossibilidade de recombinar os genes de dois indivíduos distintos para formar novos indivíduos, é preciso gerar novos indivíduos a partir da recombinação dos genes de um único indivíduo.

O operador de recombinação foi adaptado para simular um processo biológico conhecido como *partenogênese* (WATTS *et al.*, 2006). A partenogênese consiste na capacidade de geração de indivíduos filhos a partir de um único indivíduo fêmea. Algumas espécies, como os dragões de Comodo, utilizam este mecanismo como uma opção para gerar filhotes quando a população está reduzida. A vantagem da partenogênese em relação a outros mecanismos de reprodução que envolvem apenas um indivíduo, é que o novo indivíduo gerado não é um clone do indivíduo gerador, mantendo assim a diversidade genética da população.

A Figura 55 ilustra o método de recombinação que foi chamado de *recombinação por deslocamento*. Neste método, cada recurso é representado por uma posição de um *vetor de recursos*. Cada posição é preenchida com o gene que foi inicialmente mapeado na respectiva posição. Um ponto de deslocamento é escolhido aleatoriamente e o vetor é deslocado circularmente. O mapeamento dos genes é alterado, contudo, a alocação de tarefas é preservada.

Com a mesma preocupação de não perder as soluções obtidas na etapa de alocação de tarefas, precisou-se alterar o operador de mutação para a etapa de mapeamento de IPs. Como o objetivo agora é encontrar o mapeamento ótimo para uma determinada alocação de IPs, o papel do operador de mutação é tentar encontrar este mapeamento através de uma abordagem

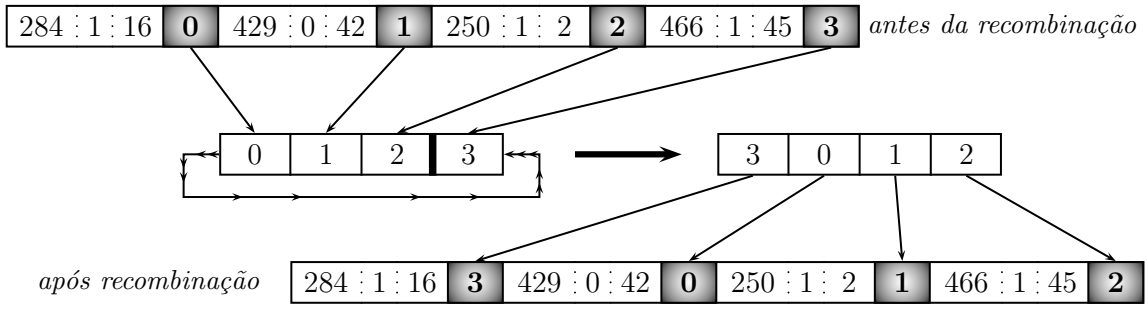


Figura 55: Método de recombinação por deslocamento

mais agressiva e menos controlada do que a realizada pelo operador de recombinação por deslocamento. O operador de mutação proposto foi chamado de *mutação interna*.

A Figura 56 ilustra o método de mutação interna. Neste método o operador de mutação gera um mapeamento diferente permutando os identificadores de recurso de dois genes selecionados aleatoriamente. Dessa maneira, uma solução que jamais seria descoberta através do operador de recombinação por deslocamento, tem chances de ser encontrada pelo operador de mutação interna.

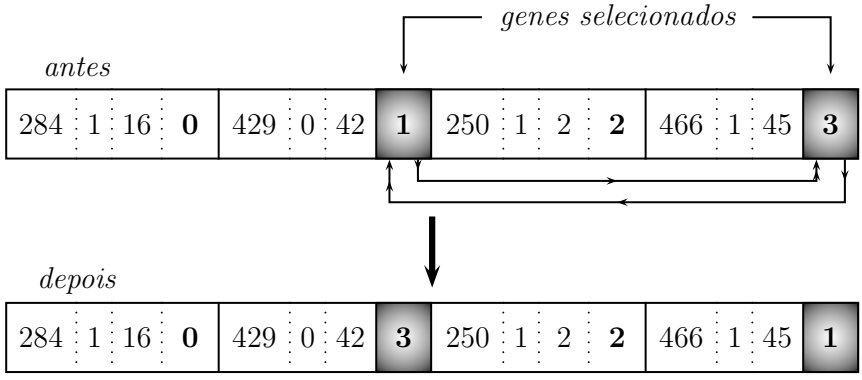


Figura 56: Método de mutação interna

Os operadores de recombinação e mutação propostos garantem a geração de soluções válidas ao longo do processo evolutivo e preservam as soluções encontradas durante o processo de alocações de IPs. As novas soluções geradas podem então ser avaliadas com base na alocação e no mapeamento.

5.4.3 Avaliação de aptidão

Após submeter os indivíduos aos operadores genéticos, os mesmos precisam ser avaliados segundo os objetivos de interesse. Na etapa de alocação de tarefas foram definidos três objetivos, que são: o consumo de energia, tempo de execução da aplicação e a área ocupada. Estes três

objetivos devem ser minimizados. Eles foram escolhidos por serem concorrentes e por representarem os principais objetivos a serem balanceados no projeto de uma NoC, como visto na Seção 4.2.3 do Capítulo 4. O cálculo destes objetivos foi feito com base nos dados reais de processadores, dados estes obtidos no E3S (DICK, 2008), que é o repositório de IPs utilizado.

Na etapa de mapeamento, estes três objetivos serão considerados novamente, sendo que o cálculo de cada um será baseado nos dados do repositório e no perfil de comunicação da aplicação. O mapeamento fornece uma visão da implementação em *hardware* e possibilita avaliar aspectos que não podem ser avaliados durante a etapa de alocação de IPs, tais como: a parcela de tempo gasto em comunicação no total de tempo de execução de uma aplicação, o consumo de energia adicional decorrente do trafego de *bits* nos canais de comunicação e nos *switches* e a área adicional dos canais de comunicação e dos *switches* necessários.

5.4.3.1 Área

Para calcular a área requerida por um determinado mapeamento, é necessário conhecer as áreas dos IPs selecionados, dos canais de comunicação e dos *switches* utilizados. Como no E3S os IPs estão relacionados a processadores, e como um processador pode ser responsável pela execução de mais de uma tarefa, é necessário percorrer o GCA de uma solução S para verificar cada processador associado a esta solução. O número total de canais de comunicação e *switches* pode ser obtido através da verificação de todos os caminhos de comunicação utilizados. Como a arquitetura de comunicação adotada equivale a uma matriz de tamanho $N \times N$, para determinadas alocações, alguns recursos, canais de comunicação e *switches* podem não ser utilizados. Por outro lado, alguns canais de comunicação podem ser reutilizados várias vezes.

O algoritmo de roteamento utilizado influencia bastante na utilização dos canais de comunicação e dos *switches* da rede. Para avaliar os componentes da arquitetura de comunicação utilizados é preciso saber o algoritmo de roteamento utilizado. Adotando o algoritmo de roteamento XY, é possível calcular o número de canais de comunicação utilizados entre dois recursos. A Equação 31 permite calcular o número de canais utilizados durante uma comunicação entre os recursos de índice i e j, em uma rede em malha $N \times N$. O valor $\mathcal{CH}(i,j)$ representa também a distância de *Manhattan* entre os dois recursos.

$$\mathcal{CH}(i,j) = |\lfloor i/N \rfloor - \lfloor j/N \rfloor| + |\, i \backslash N - j \backslash N| \tag{31}$$

O número de canais de comunicação entre todos dois recursos permite calcular o número de *switches* usados, conforme mostra Equação 32.

$$\mathcal{SW}(i,j) = \mathcal{CH}(i,j) + 1 \tag{32}$$

A área total pode ser obtida a partir da soma da área necessária para implementar os IPs, a área dos canais de comunicação e dos *switches* usados. A área dos *switches* e dos canais de comunicação depende da plataforma NoC usada. Uma ferramenta de auxílio a projeto deve possibilitar a configuração destes parâmetros pelo projetista.

A Equação 33 apresenta o cálculo da área total necessária para implementar o mapeamento indicado na solução S. A função $\acute{A}rea_A(a)$ fornece, para uma dada alocação a, a área ocupada pelos recursos locais da rede. Esta função é definida na Equação 16 do Capítulo 4. A alocação que deu origem ao mapeamento S é obtida através de $\mathcal{A}_S$. A função $\mathcal{E}(g)$ retorna todos as arestas do grafo de tarefas g enquanto os atributos src e tgt fornecem, para uma dada aresta, as tarefas de origem e destino, respectivamente. A notação $S[t]_{rec}$ indica o índice do recurso onde a tarefa t é mapeada na rede, considerando a solução S. As constantes $\acute{A}rea_c$ e $\acute{A}rea_s$ representam as áreas de um canal de comunicação e *switch*, respectivamente.

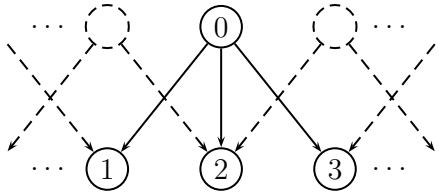
$$\begin{aligned} \acute{A}rea_M(S) = \; & \acute{A}rea_A(\mathcal{A}_S) + \\ & \acute{a}rea_c \times \sum_{d \in \mathcal{E}(GT)} \mathcal{CH}\left(S[d_{src}]_{rec}, S[d_{tgt}]_{rec}\right) + \\ & \acute{a}rea_s \times \sum_{d \in \mathcal{E}(GT)} \mathcal{SW}\left(S[d_{src}]_{rec}, S[d_{tgt}]_{rec}\right) \end{aligned} \tag{33}$$

5.4.3.2 Tempo de execução

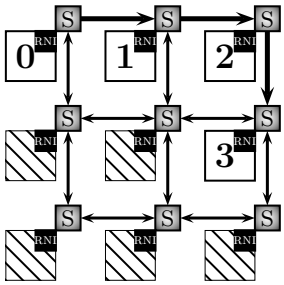
Para calcular o tempo de execução imposto por um dado mapeamento, é necessário considerar o tempo de execução de cada tarefa do caminho crítico, sua ordem de escalonamento e o tempo adicional decorrente do tráfego de dados via os canais de comunicação e *switches*. O caminho crítico é obtido visitando-se todos os nós do GT, verificando o tempo de execução de cada tarefa e registrando o caminho com maior tempo de execução. Adotando o algoritmo de roteamento XY, os canais de comunicação e os *switches* utilizados são identificados através das Equações 30 e 31, respectivamente.

Foram identificadas duas situações críticas, devidas ao mapeamento, que aumentam o tempo de execução da aplicação, degradando o desempenho final da mesma. São elas:

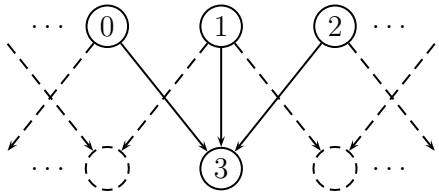
1. **Tarefas paralelas com fontes comuns e compartilhando canais de comunicação**: É o caso em que uma tarefa está mapeada em um recurso e envia dados para tarefas paralelas que estão mapeadas em recursos situados na mesma linha ou na mesma coluna. Os dados a serem usados pelas tarefas paralelas não poderão ser enviados simultaneamente, e portanto, estas não poderão iniciar no mesmo tempo. Note que quanto mais



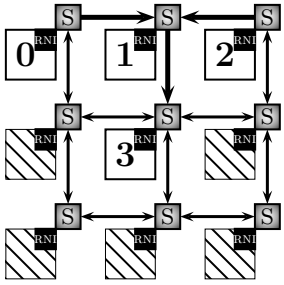
(a) Fonte comum – múltiplos destinos



(b) Mapeamento ilustrando a situação (1)



(c) Múltiplas fontes – destino comum



(d) Mapeamento ilustrando a situação (2)

Figura 57: Impacto do mapeamento de tarefas paralelas no tempo de execução da aplicação

tarefas paralelas estiverem mapeadas em recursos localizados na mesma linha ou mesma coluna, maior será o atraso. A Figuras 57(a) mostra a configuração desta situação no grafo de tarefas enquanto que a Figura 57(b) ilustra o impacto do mapeamento das tarefas paralelas 1 e 2 em recursos localizados na mesma linha 0, e 2 e 3 na mesma coluna 2.

2. **Tarefas paralelas com destinos comuns e compartilhando canais de comunicação**: É o caso em que tarefas paralelas, mapeadas em recursos diferentes, enviam dados para uma tarefa mapeada em um recurso e em algum instante, durante o envio, os canais de comunicação são compartilhados. A Figuras 57(c) mostra a configuração desta situação no grafo de tarefas enquanto a Figura 57(d) ilustra o impacto do mapeamento das tarefas paralelas 0, 1 e 2 em recursos localizados na linha 0 e compartilhando o canal entre os *switches* 1 e 4 para se comunicar com a tarefa 3 mapeada no recurso 4.

Levando em conta as duas situações destacadas, a Equação 34 fornece o tempo de execução imposto pela alocação e mapeamento ditados pela solução S, onde como antes, a função $\mathcal{C}(g)$ retorna todos os possíveis caminhos P de um grafo de tarefa g, enquanto que $Tempo_p(Q)$ representa o tempo gasto no processamento das tarefas do caminho Q e $Tempo_c(Q)$

é o tempo gasto na comunicação entre as tarefas do caminho Q, assumindo que não há nenhuma contenção nos canais da rede. No entanto, como essa suposição não é sempre válida, é necessário adicionar os atrasos devidos às duas situações identificadas no começo dessa seção. O atraso devido ao enfileiramento de dados causado pela primeira situação é quantificado pela função f_1 na Equação 34 e o atraso causado por diferentes tarefas enviando dados para um mesmo recurso através do mesmo canal de comunicação é calculado pela função f_3 na Equação 34.

$$Tempo(S) = \max_{Q \in \mathcal{C}(GT)} (Tempo_p(Q) + Tempo_c(Q) + t_L \times (f_1(Q) + f_2(Q))) \tag{34}$$

O tempo gasto em processamento para um caminho Q do grafo de tarefa, é inspirado daquele calculado durante a etapa de alocação na Equação 17 do Capítulo 4, conforme mostrado na Equação 35.

$$Tempo_p(Q) = \sum_{t \in Q} tempo_{\mathcal{A}_S[t]_{ip}} + \begin{cases} 0 & \textbf{se} \quad \mathcal{A}_S[t]_{dedicado} = 1 \\ & \textbf{ou} \quad \mathcal{P}_{(t)} = \mathcal{D}_{(t)} = \emptyset \\ \sum\limits_{\substack{t' \in \mathcal{P}_{(t)} \cup \mathcal{D}_{(t)} \\ t' < t}} tempo_{\mathcal{A}_S[t']_{ip}} & \textbf{senão} \end{cases} \tag{35}$$

O tempo gasto em comunicação é espelhado no modelo descrito na Seção 5.3, resumido na Equação 30. O termo $Tempo_cQ$ para um caminho Q do grafo de tarefas é calculado de acordo com a Equação 36, onde $\mathcal{E}(GT)$ representa o conjunto de arestas $d(src, tgt)$, do grafo de tarefas, $volume_{d(t,t')}$ é o volume de bits transmitidos da tarefa t para a tarefa t', e t_R e t_L são os atrasos impostos pela lógica de roteamento do *switch* e pelo canal de comunicação, respectivamente.

$$Tempo_c(Q) = \sum_{\substack{d(t,t') \in \mathcal{E}(GT) | \\ t \in Q, t' \in Q}} \mathcal{SW}(S[d_t]_{rec}, S[d_{t'}]_{rec}) \times (t_R + t_L) + \left\lceil \frac{volume_{d(t,t')}}{phit} \right\rceil \times t_L \tag{36}$$

A função $f_1(Q)$ calcula o atraso causado por todas as tarefas do caminho Q que satisfazem as condições da primeira situação crítica, descrita no início desta seção. O Algoritmo 10 descreve o funcionamento desta função. Para cada tarefa paralela que deve ser alcançada através do mesmo canal de comunicação inicial, ocorre uma penalidade no tempo de execução total devido ao enfileiramento de dados nos canais de comunicação. A função $Alvos(t)$ retorna um conjunto de todas as tarefas no grafo que dependem da tarefa t, $i\mathcal{CH}(t, t')$ retorna o índice do canal inicial de comunicação da tarefa t para a tarefa t' e *penalidade* é a quantidade de *flits* que seriam transmitidos no momento que ocorre a situação crítica.

A função f_2 calcula o atraso causado por todas as tarefas do caminho Q que satisfazem as condições da segunda situação crítica, descrita no início desta seção. Nesta situação, o

Algoritmo 10 $f_1(Q)$ – MesmaFonteDifAlvos

```
1: penalidade := 0
2: Para todo t ∈ Q Faça
3:   Se alvos(t) > 1 Então
4:     Seja t_1 ∈ alvos(t) | t_1 ∈ Q
5:     Para todo t_2 ∈ alvos(t)\t_1 Faça
6:       Se iCH(t, t_1) = iCH(t, t_2) e t_1 > t_2 Então
7:         penalidade := penalidade + ⌈volume(t,t_2) / phit⌉
8:       Fim Se
9:     Fim Para
10:  Fim Se
11: Fim Para
12: Retorna penalidade
```

atraso ocorre toda vez que algum canal de comunicação é solicitado no mesmo tempo. Se um *switch* recebe pacotes de dois canais simultaneamente e precisa rotear os pacotes para o mesmo canal de saída, haverá um atraso no envio do segundo pacote. O Algoritmo 11 calcula quantas vezes essa situação ocorre. A função $\mathcal{CH}s(t, t')$ retorna uma lista ordenada dos canais utilizados durante a comunicação da tarefa t com a tarefa t' e *penalidade* é a quantidade de *flits* que seriam transmitidos no momento que ocorre a situação crítica.

Algoritmo 11 $f_2(Q)$ – DifFonteMesmoAlvo

```
1: penalidade := 0
2: Para todo t ∈ Q Faça
3:   Para todo t_1 ∈ GT | t_1 ≠ t e nivel(t) = nivel(t_1) Faça
4:     Para todo s ∈ alvos(t) e s_1 ∈ alvos(t_1) | s = s_1 Faça
5:       w = CHs(t, s)
6:       w_1 = CHs(t_1, s_1)
7:       Se existir i ∈ [0, min(w.tamanho, w_1.tamanho)] | w(i) = w_1(i) e t > t_1 Então
8:         penalidade := penalidade + ⌈volume(t_1,s) / phit⌉
9:       Fim Se
10:    Fim Para
11:  Fim Para
12: Fim Para
13: Retorna penalidade
```

5.4.3.3 Consumo de energia

Para calcular o consumo de energia de um dado mapeamento é necessário considerar a energia consumida pelos IPs para processamento de dados, e a energia gasta na comunicação de dados entre os IPs. O total de energia consumida por uma aplicação é então calculado conforme mostra a Equação 37, onde $Consumo_p$ e $Consumo_c$ representam os termos devidos ao

processamento e à comunicação, respectivamente.

$$Consumo(S) = Consumo_p(S) + Consumo_c(S) \tag{37}$$

O consumo de energia devido ao processamento é o mesmo que foi calculado na etapa de alocação de tarefas pois dependia apenas de dados fornecidos no repositório de IPs. A Equação 18 do Capítulo 4 mostra como o devido valor é calculado.

O consumo de energia referente à comunicação representa um fator importante no projeto de uma NoC e precisa ser considerado, mesmo em uma descrição de alto nível, para obter uma implementação de baixo consumo de energia. Este fator depende do perfil de comunicação da aplicação e das características da plataforma de comunicação. O perfil de comunicação da aplicação é obtido da alocação de tarefas e do mapeamento, enquanto que as características da plataforma de comunicação variam de acordo com a topologia de rede, a técnica de chaveamento e o algoritmo de roteamento utilizado. O modelo de energia consumida por um *bit* ao ser enviado de um recurso para outro foi apresentado na Seção 5.2. Este modelo será utilizado para calcular a energia total gasta em comunicação por uma aplicação executada em uma plataforma NoC.

Lembrando que, o grafo de tarefas de uma aplicação informa o volume de comunicação entre duas tarefas t e t' em termos de *bits*, enviados de t para t', associando-o à aresta orientada $d_{t,t'}$. Assumindo que as tarefas t e t' foram mapeadas nos recursos i e j respectivamente, o volume de comunicação entre os recursos i e j é denotado por $Volume_{d(t,t')}$. A comunicação entre os recursos i e j pode utilizar um único canal de comunicação $C_{i,j}$ ou uma sequencia de $m > 1$ canais de comunicação $[c_{i,x_0}, c_{x_0,x_1}, c_{x_1,x_2}, \ldots, c_{x_{m-1},j}]$. Por exemplo, em uma NoC malha 3×3 e usando o roteamento XY, para uma tarefa mapeada no recurso 0 (canto superior esquerdo) da malha, se comunicar com uma tarefa mapeada no recurso 8 (canto inferior direito), os canais de comunicação utilizados serão $[c_{0,1}, c_{1,2}, c_{2,5}, c_{5,8}]$.

O consumo de energia referente a toda comunicação da aplicação é dado pela Equação 38, onde como antes, $Alvos(t)$ retorna o conjunto de tarefas que dependem da tarefa t e $S[t]_{rec}$ retorna o índice de mapeamento da tarefa t, considerando a solução S.

$$Consumo_c(S) = \sum_{\substack{t \in GT, \\ \forall t' \in alvos(t)}} volume_{d(t,t')} \times E_{bit}^{S[t]_{rec}, S[t']_{rec}} \tag{38}$$

5.4.3.4 Restrição quanto à formação de *hot spots*

Como um dos objetivos de interesse é minimizar a área ocupada pela implementação final, isso implica em reduzir a quantidade de IPs utilizados. Em uma implementação MPSoC, essa redução de área implica em alocar mais tarefas por processador, neste caso, mais tarefas por IPs associados ao mesmo processador. A atribuição de muitas tarefas para o mesmo processador e/ou para um mesmo recurso, pode gerar o que é identificado na área de projetos VLSI como *hot spots* (ZHOU; ZHANG; MAO, 2006).

Hot spots são regiões dentro do CI que apresentam um alto perfil de processamento. A comunicação de dados nestas regiões é intensa, ocasionando um aquecimento localizado que pode danificar o CI. Para evitar a formação de *hot spots* é necessário criar uma restrição na avaliação, restrição esta que limita o número máximo de tarefas alocadas por processador.

Esta é uma restrição que foi implementada para controlar de forma explicita o problema de formação de *hot spots*. De forma implícita, a formação de *hot spots* é evitada em aplicações com alto grau de paralelismo, pois a alocação de muitas tarefas paralelas em um mesmo processador aumenta o tempo de execução da aplicação, gerando soluções muito lentas que provavelmente serão descartadas em sistemas de apoio a projeto baseados na agregação de objetivos. No caso de sistemas que utilizam o conceito de dominância para a seleção, talvez estas soluções não sejam dominadas, mas apresentarão tempo de execução bem maior quando comparadas com soluções que exploram o paralelismo.

5.5 Resultados Experimentais

Nesta seção são apresentados os resultados obtidos pelas implementações do método de mapeamento evolutivo de IPs em plataforma NoC utilizando os algoritmos NSGA-II e microGA. Foram utilizadas as 16 aplicações obtidas do E3S (DICK, 2008), as mesmas usadas no Capítulo 4, além da aplicação SegImag (CARDOZO, 2005), descrita nesta seção. Os resultados de cada implementação são apresentados separadamente em relação à quantidade de soluções não dominadas encontradas e aos patamares obtidos para os objetivos considerados. Em seguida, é realizada uma comparação entre os mapeamentos obtidos pelo NSGA-II e microGA e o sistema CAFES (MARCON *et al.*, 2005b). Esta comparação tem como objetivo comparar o desempenho dos algoritmos utilizados e comprovar a eficiência do método através dos mapeamentos obtidos.

Para ambas as implementações, foi estabelecida a restrição de mapear processadores com no máximo 3 tarefas alocadas. Esta quantidade de tarefas foi obtida experimentalmente, variando a quantidade de tarefas permitidas por processador e avaliando as soluções obtidas.

A complexidade de cada problema varia com o número de processadores utilizados em cada alocação, o que definirá a quantidade de recursos da plataforma NoC, de acordo com a Equação 19. A Tabela 17 apresenta para cada aplicação do repositório E3S, o número de alocações não dominadas usadas como base e o número de mapeamentos factíveis em cada caso, calculado de acordo com a Equação 21.

Tabela 17: Complexidade dos problemas de mapeamento para as aplicações do repositório do E3S

ID	Aplicação	#Alocações não dominadas		#Mapeamentos possíveis	
		NSGA-II	microGA	NSGA-II	microGA
1	*auto-indust-tg0*	33	47	792	1.128
2	*auto-indust-tg1*	9	17	192	300
3	*auto-indust-tg2*	621	1.435	23.513.784	143.566.872
4	*auto-indust-tg3*	11	13	240	264
5	*consumer-tg0*	157	1.050	939.720	12.167.664
6	*consumer-tg1*	67	150	1.368	33.480
7	*networking-tg1*	35	95	564	1.500
8	*networking-tg2*	39	92	612	1.476
9	*networking-tg3*	37	103	588	1.632
10	*office-tg0*	159	1.010	18.672	2.152.848
11	*telecom-tg0*	17	51	204	612
12	*telecom-tg1*	266	1.169	6.024	2.977.608
13	*telecom-tg2*	254	1.158	5.628	2.660.760
14	*telecom-tg3*	7	20	51	207
15	*telecom-tg4*	11	26	99	279
16	*telecom-tg5*	11	11	99	99

5.5.1 Resultados do NSGA-II

Para os resultados expostos nesta seção, a implementação do NSGA-II foi configurada com os seguintes parâmetros: uma população de 850 indivíduos, taxa de mutação de 0,01, taxa de recombinação de 0,8, um número de gerações de 100. O torneio de seleção foi realizado entre 3 indivíduos. Os valores adequados desses parâmetros foram identificados após várias simulações.

A Tabela 18 e a Tabela 19 apresentam os resultados obtidos pelo NSGA-II para as aplicações do E3S. A Tabela 18 mostra, para cada aplicação, o número total de soluções não dominadas, as médias de consumo de energia, área de *hardware* e tempo de execução, obtidos para estas soluções. A Tabela 18 inclui também a quantidade média, máxima e mínima de recursos utilizados, que neste caso, corresponde à quantidade de processadores. A Tabela 19 apresenta os valores mínimos obtidos para cada objetivo de interesse. Note que esses valores

Tabela 18: Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de recursos mapeados com o NSGA-II para as aplicações do E3S

ID	#	Médias de			# Recursos		
	Soluções	Consumo[1]	Área[2]	Tempo[3]	Mínimo	Máximo	Médio
1	34	4,3507	5,2689	0,0222	3	4	3,63
2	24	2,2450	3,4823	0,0475	2	3	2,77
3	965	8,2950	17,5436	6,0473	4	6	5,39
4	24	3,2450	3,4823	0,0396	2	4	2,9
5	26	10,4862	22,6524	59,3733	3	5	4,16
6	40	6,2531	8,9001	14,5200	2	3	2,7
7	16	1,7851	40,3042	0,2820	2	3	2,34
8	16	1,7851	40,3042	0,3110	2	3	2,3
9	21	2,2935	40,2228	0,3900	2	3	2,32
10	49	15,3425	38,3728	5,0175	2	5	3,04
11	16	3,4450	2,8642	0,0346	2	2	2
12	8	5,0250	3,1084	0,2198	2	4	3,6
13	8	5,0250	3,1084	0,2198	2	4	3,55
14	36	1,9272	3,2096	0,0587	1	2	1,57
15	59	1,4980	20,9482	0,3457	1	2	1,72
16	36	1,6664	2,8056	0,2534	1	2	1,72

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

não provêm necessariamente da mesma solução, mas indicam os valores mínimos encontrados em soluções não dominadas.

A Tabela 20 expõe os mapeamentos não dominados, encontrados pelo NSGA-II, para uma amostra das aplicações do E3S. Entre as 16 aplicações usadas, foram selecionadas 6 dos diferentes domínios com diferentes complexidades. A segunda coluna indica a alocação, onde cada posição corresponde ao identificador da tarefa no GT e o valor na entrada corresponde ao identificador do IP alocado para a tarefa correspondente. A terceira coluna apresenta o mapeamento, onde nas listas de cima, a posição indica o identificador da tarefas no GT e o valor da entrada indica o identificador do recurso mapeado, já nas listas de baixo, a posição indica o identificador do recurso e o valor da entrada indica o identificador do processador mapeado no recurso, de acordo com a Tabela 31 do Apêndice A. A presença do símbolo $\times$ em uma das últimas colunas indica que a alocação juntamente com o mapeamento indicado permitiram alcançar o valor mínimo em área (A), consumo de energia (C) e/ou tempo (T).

5.5.2 Resultados do microGA

Para os resultados expostos nesta seção, a implementação do microGA foi configurada com os seguintes parâmetros: uma memória populacional de 1.000 indivíduos, sendo 70% renovável e

Tabela 19: Mínimos dos objetivos para as soluções não dominadas, encontradas pelo NSGA-II

ID	Mínimos de		
	Consumo[(1)]	Área[(2)]	Tempo[(3)]
1	4,150000124	4,2300	0,0222
2	2,15000005	2,6200	0,0475
3	7,750000302	13,6324	0,8451
4	3,150000013	2,6200	0,0396
5	6,300068495	11,3892	57,3400
6	4,225081965	8,8024	14,5200
7	1,69005289	40,0600	0,2820
8	1,69010578	40,0600	0,3110
9	1,690211561	40,0600	0,3900
10	5,940009937	16,5948	4,2300
11	3,350000107	2,6200	0,0346
12	5,025000101	3,1084	0,2198
13	5,025000101	3,1084	0,2198
14	0,225	1,0100	0,0290
15	0,225	1,0100	0,1300
16	0,15	1,0100	0,0730

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

30% não renovável, memória externa de 200 indivíduos, população de 4 indivíduos, um número de gerações de 3.000, ciclos de recolocação realizados a cada 200 gerações, uma convergência nominal a cada 4 gerações e grade adaptativa com bisseção de 5. O torneio binário de seleção foi usado. Os valores adequados desses parâmetros foram identificados após várias simulações.

A Tabela 21 apresenta resultados obtidos pelo microGA para as aplicações do E3S, sendo estes, o número total de soluções não dominadas, as médias de consumo de energia, área de *hardware* e tempo de execução, obtidos para estas soluções. A Tabela 21 inclui também a quantidade média, máxima e mínima de elementos processadores que foram alocados. A Tabela 22 apresenta os valores mínimos obtidos para cada objetivo de interesse. Note que esses valores não provêm necessariamente da mesma solução, mas indicam os valores mínimos encontrados em soluções não dominadas.

Como foi feito para o NSGA-II, a Tabela 23 expõe os mapeamentos não dominados, encontrados pelo microGA, para uma amostra das aplicações do E3S. Entre as 16 aplicações usadas, foram selecionadas 6 dos diferentes domínios e com diferentes complexidades. Como antes, a presença do símbolo $\times$ em uma das últimas colunas indica que a alocação junto com o mapeamento indicado permitiram alcançar o valor mínimo em área (A), consumo de energia (C) e/ou tempo (T).

Tabela 20: Amostra de mapeamentos não dominados, encontrados pelo NSGA-II, apresentando valores mínimos nos objetivos, conforme listado na Tabela 19

ID	Alocação não dominada	Mapeamento não dominado	C	A	T
1	[762, 370, 371, 370, 382, 462]	[1, 0, 3, 3, 3, 1] [13, 15, –, 13]	×		×
	[369, 370, 371, 370, 382, 369]	[1, 0, 0, 0, 3, 1] [13, 12, –, 13]		×	
3	[162, 375, 380, 379, 6, 370, 383, 384, 462]	[3, 6, 6, 6, 4, 7, 8, 5, 3] [–, –, –, 15, 0, 13, 13, 13, 13]	×		×
	[247, 375, 380, 379, 208, 370, 383, 384, 247]	[5, 4, 4, 4, 5, 3, 8, 7, 5] [–, –, –, 13, 13, 6, –, 13, 13]		×	×
5	[462, 456, 456, 241, 243, 239, 462]	[2, 3, 2, 0, 0, 0, 2] [6, –, 15, 15]	×	×	
	[369, 241, 241, 241, 243, 239, 369]	[3, 1, 0, 2, 0, 0, 3] [6, 6, 6, 12]			×
7	[462, 345, 346, 462]	[0, 2, 2, 0] [15, –, 11, –]	×		×
	[369, 345, 346, 369]	[2, 0, 0, 2] [11, –, 12, –]		×	×
10	[369, 44, 369, 245, 244]	[2, 3, 0, 0, 0] [–, 0, 6, 12]	×		
	[369, 246, 369, 245, 244]	[3, 1, 3, 1, 1] [6, 12, – 6]		×	
	[462, 96, 462, 245, 244]	[2, 3, 2, 0, 0] [6, –, 15, 2]			×
12	[462, 463, 441, 472, 469, 462]	[2, 0, 2, 0, 0, 2] [16, –, 15, –]	×		
	[462, 463, 441, 472, 469, 462]	[2, 0, 2, 0, 0, 2] [16, –, 15, –]		×	
	[462, 463, 441, 472, 469, 462]	[2, 0, 2, 0, 0, 2] [16, –, 15, –]			×

5.5.3 Comparação dos resultados

A comparação dos resultados obtidos na resolução do problema de mapeamento será realizada em duas etapas. Primeiro, os mapeamentos obtidos pelo NSGA-II são comparados com aqueles obtidos pelo microGA. Em seguida, os resultados obtidos pelo NSGA-II e microGA são comparados com o mapeamento gerado pelo sistema CAFES. Para isso, é usada a aplicação de segmentação de imagem *SegImag* que é descrita na Seção 5.5.3.2.

5.5.3.1 NSGA-II *vs.* microGA

As execuções do NSGA-II e microGA foram repetidas igualmente e as soluções não dominadas de cada execução foram aproveitadas. Cada evolução do microGA foi aproximadamente sete

Tabela 21: Quantidade de soluções ótimas encontradas, médias dos objetivos e quantidade de EPs alocados com o microGA para as aplicações do E3S

ID	#	Médias de			# Processadores		
	Soluções	Consumo[1]	Área[2]	Tempo[3]	Mínimo	Máximo	Média
1	62	5,0228	5,4026	0,0222	3	4	3,66
2	48	2,5950	4,0947	0,0475	2	3	2,47
3	2.991	7,9811	17,7408	37,3531	3	9	6,01
4	64	4,0686	3,9902	0,0396	2	3	2,7
5	1.539	2,8241	16,2651	222,2263	3	7	4,51
6	156	3,0056	8,3826	57,7709	2	5	2,96
7	241	2,5131	14,5534	1,9358	2	4	2,32
8	236	2,7321	15,3479	2,2920	2	4	2,35
9	249	2,5519	16,7641	3,2792	2	4	2,35
10	958	6,9900	24,8364	267,1352	2	5	3,58
11	94	2,9505	3,3552	1,1538	2	2	2,00
12	506	2,3998	16,8714	8,7820	2	6	3,85
13	500	2,4893	15,9994	7,8653	2	6	3,84
14	60	2,6329	2,8674	0,0853	1	2	1,85
15	77	2,0809	14,8663	0,4437	1	2	1,88
16	36	1,7228	2,4337	0,5501	1	2	1,72

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

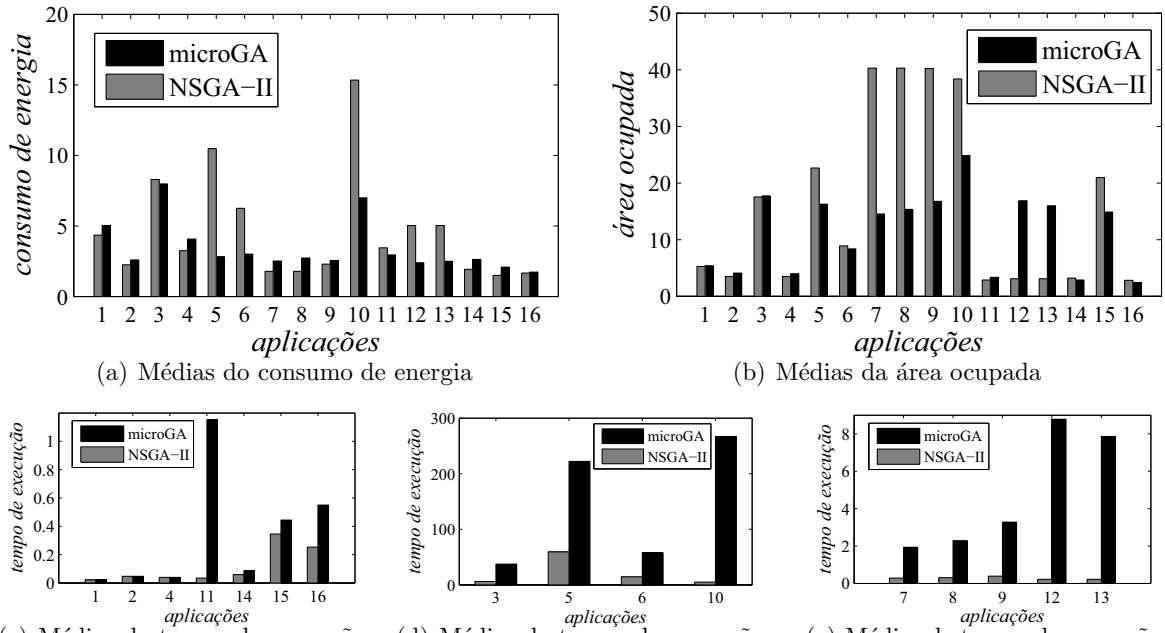
Tabela 22: Mínimos dos objetivos para as soluções não dominadas, encontradas pelo microGA

ID	Mínimos de		
	Consumo[1]	Área[2]	Tempo[3]
1	4,150000101	4,2300	0,0222
2	2,15000005	2,6200002	0,0475
3	7,150000302	9,0600006	0,84514
4	3,150000013	2,6200002	0,03955
5	0,52509873	7,3051998	57,34
6	0,37505001	3,1084	14,52
7	0,300026445	3,1084	0,282
8	0,30005289	3,1084	0,311
9	0,30010578	3,1084	0,39
10	0,375004981	2,6200002	4,23
11	0,300000044	2,6200002	0,03456
12	0,450000195	2,6200002	0,21976
13	0,450000182	2,6200002	0,21976
14	0,225000038	1,0100001	0,029
15	0,225000038	1,0100001	0,13
16	0,15	1,0100001	0,073

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s)

Tabela 23: Amostra de mapeamentos não dominados, encontrados pelo microGA, apresentando valores mínimos nos objetivos, conforme listado na Tabela 19

ID	Alocação não dominada	Mapeamento não dominado	C	A	T
1	[462, 370, 371, 370, 382, 462]	[2, 0, 1, 1, 1, 3] [13, 13, 15, 15]	×		×
	[369, 370, 371, 370, 382, 369]	[0, 3, 3, 3, 1, 0] [12, 13, –, 13]		×	
3	[462, 375, 380, 379, 376, 370, 383, 384, 462]	[3, 0, 1, 1, 4, 1, 2, 5, 3] [13, 13, 13, 15, 13, 13, –, –, –]	×		
	[369, 375, 380, 379, 376, 370, 383, 384, 369]	[1, 0, 2, 5, 2, 8, 7, 2, 1] [13, 12, 13, –, –, 13, –, 13, 13]		×	
	[462, 375, 380, 379, 6, 370, 383, 384, 462]	[7, 4, 1, 1, 8, 1, 0, 3, 7] [13, 13, –, 13, 13, –, –, 15, 0]			×
5	[462, 456, 456, 456, 458, 454, 462]	[7, 5, 1, 4, 4, 4, 2] [–, 15, 15, –, 15, 15, –, 15, –]	×		
	[369, 456, 456, 456, 458, 454, 369]	[3, 1, 1, 2, 1, 0, 3] [15, 15, 15, 12]		×	
	[462, 241, 241, 241, 243, 239, 462]	[3, 1, 2, 0, 0, 0, 3] [6, 6, 6, 15]			×
7	[462, 434, 435, 462]	[2, 2, 2, 3] [–, –, 15, 15]	×		
	[369, 434, 435, 462]	[1, 3, 3, 3] [–, 12, –, 15]		×	
	[462, 345, 346, 462]	[1, 3, 3, 1] [–, 15, –, 11]			×
10	[462, 461, 462, 460, 459]	[2, 3, 0, 0, 0] [15, –, 15, 15]	×		
	[201, 368, 201, 367, 366]	[3, 1, 3, 1, 1] [–, 12, –, 5]		×	
	[462, 96, 462, 245, 244]	[2, 3, 2, 0, 0] [6, –, 15, 2]			×
12	[462, 438, 441, 447, 444, 462]	[8, 8, 7, 3, 5, 4] [–, –, –, 15, 15, 15, –, 15, 15]	×		
	[462, 438, 441, 194, 191, 201]	[2, 2, 2, 3, 3, 3] [–, –, 16, 5]		×	
	[462, 438, 441, 472, 469, 462]	[2, 3, 2, 3, 3, 2] [–, –, 15, 16]			×



(a) Médias do consumo de energia

(b) Médias da área ocupada

(c) Médias do tempo de execução de 1, 2, 4, 11, 14, 15 e 16

(d) Médias do tempo de execução de 3, 5, 6 e 10

(e) Médias do tempo de execução de 7, 8, 9, 12 e 13

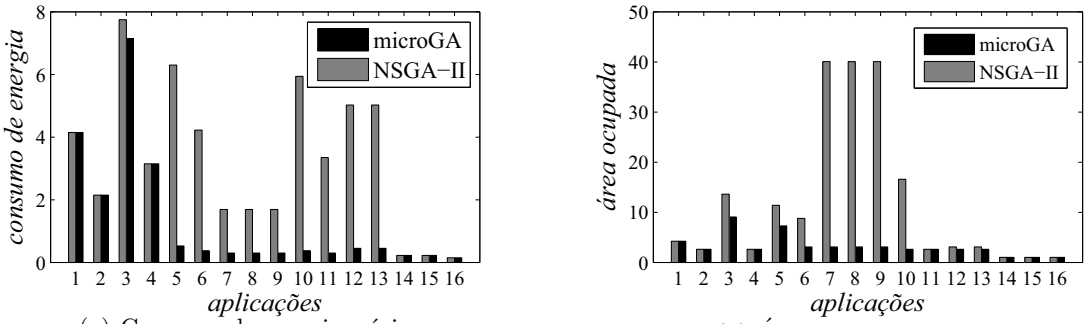
Figura 58: Representação dos valores médios dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S

vezes mais rápida do que cada evolução do NSGA-II.

Os resultados médios para os objetivos avaliados por ambos os algoritmos pode ser observado nos histogramas da Figura 58. É possível observar que, na maior parte das aplicações, o algoritmo que obteve a melhor média em relação ao consumo de energia, não obteve a melhor média em relação à área ocupada e vice-versa. Em relação ao tempo de execução médio, o NSGA-II obteve, na média, soluções melhores.

A Figura 59 mostra os resultados mínimos encontrados por ambos os algoritmos em relação ao consumo de energia e a área ocupada. Pode-se observar que o microGA encontrou valores menores ou iguais aos encontrados pelo NSGA-II, sendo que na maior parte dos casos, os valores encontrados foram menores. Em relação ao tempo de execução, ambos os algoritmos encontraram os mesmos valores.

As Figuras 60(a) e (b) permitem comparar o número de soluções não dominadas encontradas por ambos os algoritmos. Das 16 aplicações do E3S, o microGA obteve mais soluções não dominadas do que o NSGA-II em 15 aplicações e na aplicação 16 a quantidade de soluções não dominadas encontradas foi igual. As Figuras 44(c) e (d) mostram as diferenças entre o tempo gasto pelo NSGA-II e o microGA na busca das soluções com os menores valores em relação aos objetivos avaliados. Desses histogramas, percebe-se que o tempo de busca referente ao mi-



(a) Consumo de energia mínimo

(b) Área ocupada mínima

Figura 59: Representação dos valores mínimos dos objetivos nas soluções não dominadas, obtidas pelo NSGA-II e microGA, para as aplicações do E3S

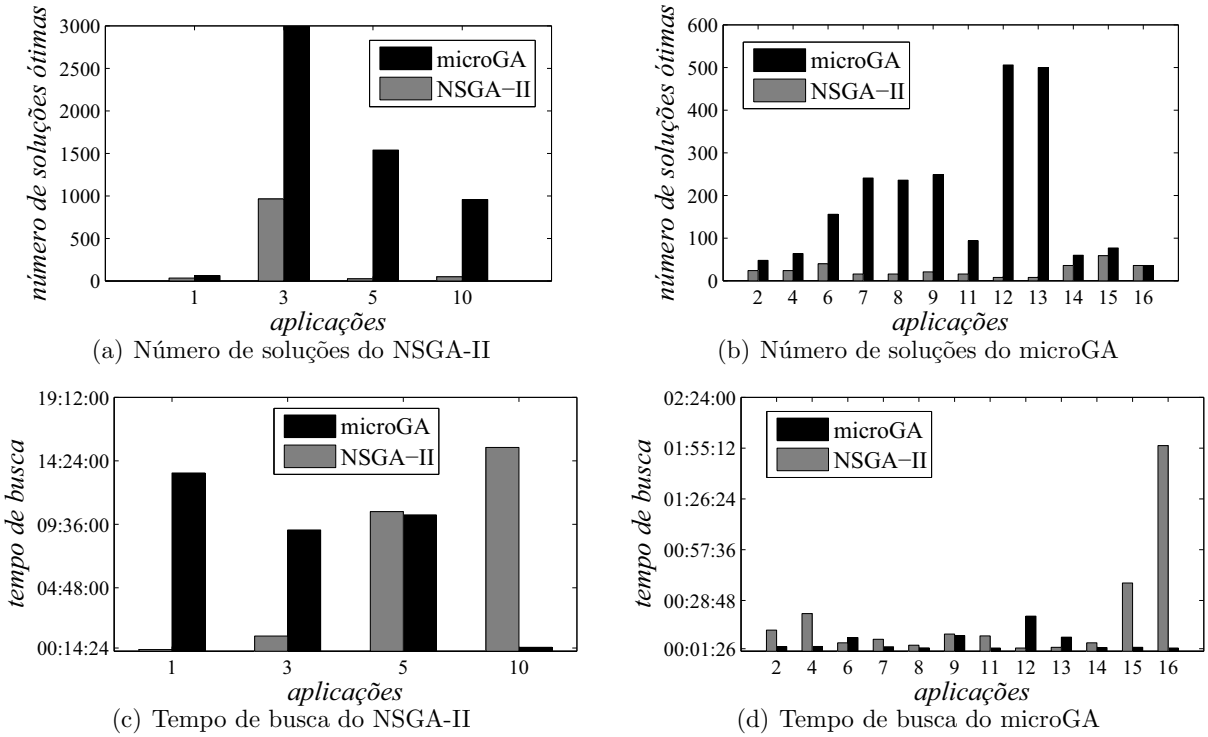
croGA foi menor do que aquele do NSGA-II para 11 das 16 aplicações e o microGA encontrou mais soluções do que NSGA-II em 15 aplicações, sendo que em uma aplicação a quantidade de soluções encontradas foi igual para ambos os algoritmos. Isso demonstra o desempenho superior do microGA sobre o NSGA-II para o problema de mapeamento de IPs. Para consolidar esta observação, foram calculados os tempos aproximados, gastos por cada algoritmo, na geração de uma única solução, conforme mostra a Tabela 24. O fator de desempenho de cada algoritmo foi avaliado em relação ao tempo gasto na busca e à quantidade de soluções encontradas. Este fator de desempenho expressa a quantidade de soluções que um algoritmo é capaz de encontrar no tempo em que o outro algoritmo gasta para encontrar uma solução. A Tabela 25 apresenta os fatores de desempenho do NSGA-II e do microGA para cada uma das aplicações. A diferença entre os dois algoritmos foi grande, sendo que apenas nas aplicações 1 e 3 o microGA não superou o NSGA-II.

5.5.3.2 Aplicação de segmentação de imagem

Para acelerar o processo de segmentação de imagem, a aplicação *SegImag* trata segmentos da imagem paralelamente. Cada segmento deve ser tratado por um processador auxiliar (PA). Na implementação original (CARDOZO, 2005), cada PA recebe e envia segmentos da imagem para uma memória externa (ME) e para um processador central (PC), como mostra a Figura 61.

A aplicação *SegImag* permite avaliar o melhor mapeamento que reduz o consumo de energia e o tempo de execução da aplicação de acordo com o nível de segmentação da imagem. A segmentação está diretamente relacionada com a quantidade de EPs utilizados, já que cada segmento deve ser processado por um PA.

Assume-se que os segmentos da imagem encontram-se inicialmente na ME. Cada seg-



(a) Número de soluções do NSGA-II

(b) Número de soluções do microGA

(c) Tempo de busca do NSGA-II

(d) Tempo de busca do microGA

Figura 60: Número de mapeamentos não dominados encontrados pelo NSGA-II e microGA, para as aplicações do E3S, e seus respectivos tempos de busca

Tabela 24: Tempo gasto aproximadamente pelos dois algoritmos na geração de um único mapeamento não dominado

ID	#Mapeamentos		Tempo de busca*		Tempo por solução*	
	NSGA-II	microGA	NSGA-II	microGA	NSGA-II	microGA
1	34	62	00:07:19,0	13:28:47,0	00:00:12,9	00:13:02,7
2	24	48	00:11:56,0	00:02:37,0	00:00:29,8	00:00:03,3
3	965	2.991	01:08:36,0	09:10:11,0	00:00:04,3	00:00:11,0
4	24	64	00:21:21,0	00:02:43,0	00:00:53,4	00:00:02,5
5	26	1.539	10:33:28,0	10:18:31,0	00:24:21,8	00:00:24,1
6	40	156	00:04:48,0	00:07:42,0	00:00:07,2	00:00:03,0
7	16	241	00:06:44,0	00:02:32,0	00:00:25,3	00:00:00,6
8	16	236	00:03:26,0	00:01:55,0	00:00:12,9	00:00:00,5
9	21	249	00:09:42,0	00:08:52,0	00:00:27,7	00:00:02,1
10	49	958	15:24:17,0	00:17:20,0	00:18:51,8	00:00:01,1
11	16	94	00:08:41,0	00:01:50,0	00:00:32,6	00:00:01,2
12	8	506	00:01:50,0	00:19:52,0	00:00:13,7	00:00:02,4
13	8	500	00:02:11,0	00:07:56,0	00:00:16,4	00:00:01,0
14	36	60	00:04:44,0	00:02:03,0	00:00:07,9	00:00:02,1
15	59	77	00:38:43,0	00:02:10,0	00:00:39,4	00:00:01,7
16	36	36	01:56:35,0	00:01:49,0	00:03:14,3	00:00:03,0

*mm:ss,ms

Tabela 25: Desempenho do NSGA-II e do microGA em relação à quantidade de soluções encontradas e o tempo de busca aproximado para cada solução

ID	NSGA-II	microGA
1	60,619	0,016
2	0,110	9,121
3	2,588	0,386
4	0,048	20,957
5	0,016	60,623
6	0,411	2,431
7	0,025	40,035
8	0,038	26,422
9	0,077	12,972
10	0,001	1.042,539
11	0,036	27,826
12	0,171	5,837
13	0,058	17,201
14	0,260	3,848
15	0,043	23,321
16	0,016	64,174

mento é enviado para o seu respectivo PA. Então, cada PA calcula os objetos contidos em seu segmento de imagem e atribui uma numeração para cada *pixel*, gerando uma associação de *pixel* com objeto. Em seguida, cada PA se comunica com o seu vizinho da esquerda e envia a coluna de *pixels*, já numerados, da fronteira esquerda do segmento. O vizinho da esquerda verifica se existem objetos em comum no segmento recebido do vizinho da direita. O mesmo é feito em relação aos PAs de baixo, que enviam os *pixels* da fronteira superior do segmento associado, para os vizinhos de cima. Após enviar, receber e processar as fronteiras, todos os PAs enviam para o PC o número de objetos encontrados e o segmentos adjacentes. O PC recebe todas as numerações individuais, gera uma numeração global, compacta esta numeração removendo as redundâncias geradas pelos segmentos adjacentes e a devolve para os PAs. Os PAs substituem a numeração antiga pela nova e transmitem seus segmentos, com o número de objetos, de volta para ME.

De acordo com a aplicação *SegImag*, processando uma imagem de 640×480 *pixels*, segmentada em 4 partes iguais, pode-se quantificar os dados enviados e recebidos pelos EPs utilizados para implementar a aplicação. O número de *bytes* na horizontal (n_{BX}) é 640 e o número de *bytes* na vertical (n_{BY}) é 480. Como existem 2 PAs na horizontal e 2 PAs na vertical, o número de *bytes* na fronteira de cada PA na horizontal (n_{BPX}) é 320 e o número de *bytes* na fronteira de cada PA na vertical (n_{BPY}) é 240. As características desta imagem juntamente com aquelas dos segmentos são apresentadas na Figura 62. Os tamanhos dos pacotes enviados

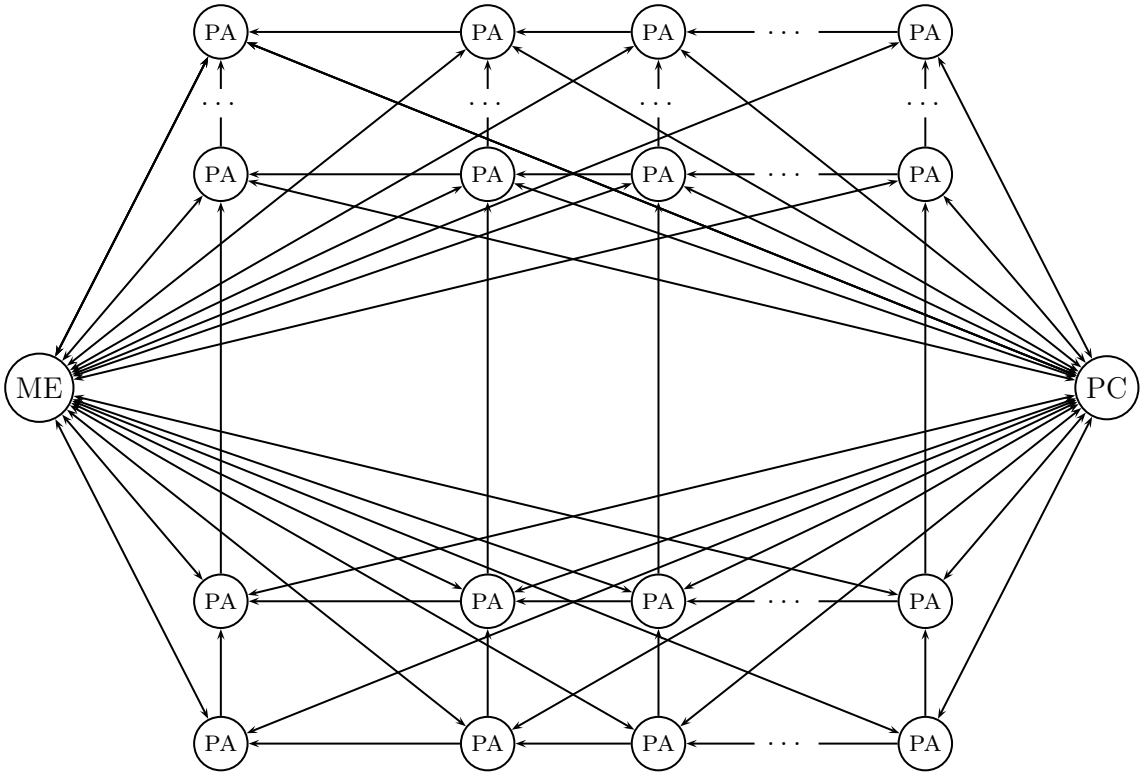


Figura 61: Implementação generalizada da aplicação *SegImag*

durante a execução da aplicação são apresentados na Tabela 26

Em termos de EPs utilizados, como mostra a Figura 61, o grafo de tarefas da aplicação *SegImag* pode ser dividido em três processos: um para os PAs, um para o PC e um para a ME. A Tabela 27 apresenta o fluxo de tarefas para cada um destes EPs.

Para submeter a aplicação *SegImag* ao processo de mapeamento evolutivo é necessário obter um grafo de tarefas da aplicação que seja único, acíclico, expresse o paralelismo entre os processos realizados pelos PAs e descreva fielmente as trocas de mensagens descritas na Tabela 26. Algumas modificações foram feitas para obter o GT desejado sem descaracterizar a aplicação *SegImag* original. As tarefas de envio e recebimento descritas na Tabela 27 serão representadas por arestas indicando as trocas de mensagens, conforme os valores descritos na Tabela 26. A tarefa t_1 será substituída pela tarefa *SI (Segmenta Imagem)* que será executada por um EP capaz de segmentar uma imagem. Este elemento substituíra a ME e enviará os segmentos de imagem para cada PA. A tarefa t_{11} será substituída pela tarefa *envia imagem para MI* (*Mescla Imagem*). Após t_{11} será adicionada a tarefa *MI* que será executada por um EP capaz de mesclar os segmentos da imagem. Note que nada impede que as tarefas *SI* e *MI* sejam executadas pelo mesmo EP. O GT único e acíclico para a aplicação *SegImag*, considerando o processamento de uma imagem dividida em 4 segmentos, está representado na Figura 63.

Tabela 26: Tamanhos de mensagens para uma imagem 640×480 dividida em quatro segmentos

PA $\leftrightarrow$ ME		Número de *bytes* total da imagem (n_{BI}) $= 640 \times 480 = 307200$ *bytes*
		Número de *bytes* de cada segmento (n_{BS}) $= 307200/4 = 76800$ *bytes*
PA $\leftrightarrow$ PA	em X	Número de *bytes* em Y (n_{BY}) - altura $= 480$ *bytes*
		Número de segmentos em Y (n_{SY}) $= 2$
		Número de *bytes* na fronteira de cada PA para Y (n_{BPY}) = $n_{BY}/n_{SY} = 480/2 = 240$ *bytes*
	em Y	Número de *bytes* em X (n_{BX}) - largura $= 640$ *bytes*
		Número de segmentos em X (n_{SX}) $= 2$
		Número de *bytes* na fronteira de cada PA para X (n_{BPX}) = $n_{BX}/n_{SX} = 640/2 = 320$ *bytes*
PA $\leftrightarrow$ PC		Número de *bytes* de controle de vizinhança (N_{CV}) $= 128$ *bytes* (estimado com base no tamanho dos objetos em imagens típicas)

Tabela 27: Fluxo de tarefas nos PAs, no PC e na ME

PA	PC	ME
t_1: Recebe imagem da ME		
t_2: Processa imagem		
Se (tem PA vizinho esquerdo)		
t_3: Envia fronteira p/ PA esquerdo		
Se (tem PA vizinho acima)	Enquanto houverem PAs	
t_4: Envia fronteira p/ PA acima	t_{12}: Recebe *cv* de um PA	Enquanto houverem PAs
Se (tem PA vizinho direito)		t_{15}: Envia imagem
t_5: Recebe fronteira do PA direito	t_{13}: Processa todos os *cvs*	para um PA
Se (tem PA vizinho abaixo)		Enquanto houverem PAs
t_6: Recebe fronteira do PA abaixo	Enquanto houverem PAs	t_{16}: Recebe imagem
Se (recebeu fronteira)	t_{14}: Envia *cv* para PA	de cada PA
t_7: Processa fronteira		
t_8: Envia *cv* para PC		
t_9: Recebe *cv* de PC		
t_{10}: Processa imagem		
t_{11}: Envia imagem para ME		

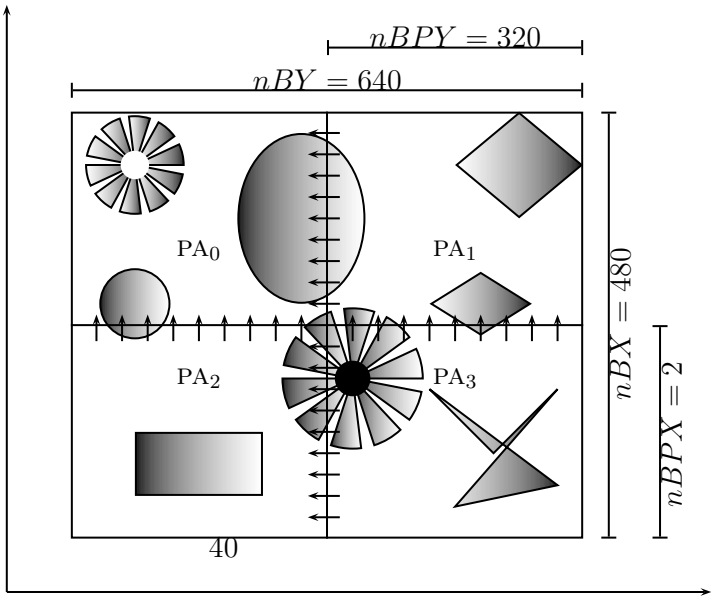


Figura 62: Exemplo de uma imagem com 640 × 480 *pixels* segmentada em quatro partes

O grafo representado na Figura 63 será submetido ao processo de alocação e mapeamento evolutivo para implementação em uma plataforma NoC com topologia em malha e usando o algoritmo de roteamento XY e a técnica de chaveamento *Wormhole*. Note que, não necessariamente os resultados obtidos devem ser iguais à implementação sugerida na Figura 61.

5.5.3.3 NSGA-II e microGA *vs.* CAFES

Para comparação com outros métodos de mapeamento em plataforma NoC, os mapeamentos obtidos pelo NSGA-II e microGA foram comparados com os mapeamentos gerados pelo sistema CAFES (*Communication Analysis for Embeeded Systems*) (MARCON *et al.*, 2005b), utilizando como base a aplicação de segmentação de imagem *SegImag*.

O CAFES utiliza o método de otimização estocástico denominado *simulated annealing* ou recozimento simulado (RUSSEL; NORVIG, 1995). O objetivo de interesse a ser otimizado pelo CAFES é o consumo de energia da plataforma NoC. Este objetivo é calculado com base na comunicação entre os recursos, utilizando um *grafo de dependência de comunicação*, também chamado de CDG (*Communication Dependence Graph*) (MARCON *et al.*, 2005a). Neste modelo de grafo, cada vértice representa uma comunicação e as arestas representam as dependências das comunicações. Para gerar o mapeamentos de uma aplicação no CAFES, é necessário transformar um GCA em um CDG.

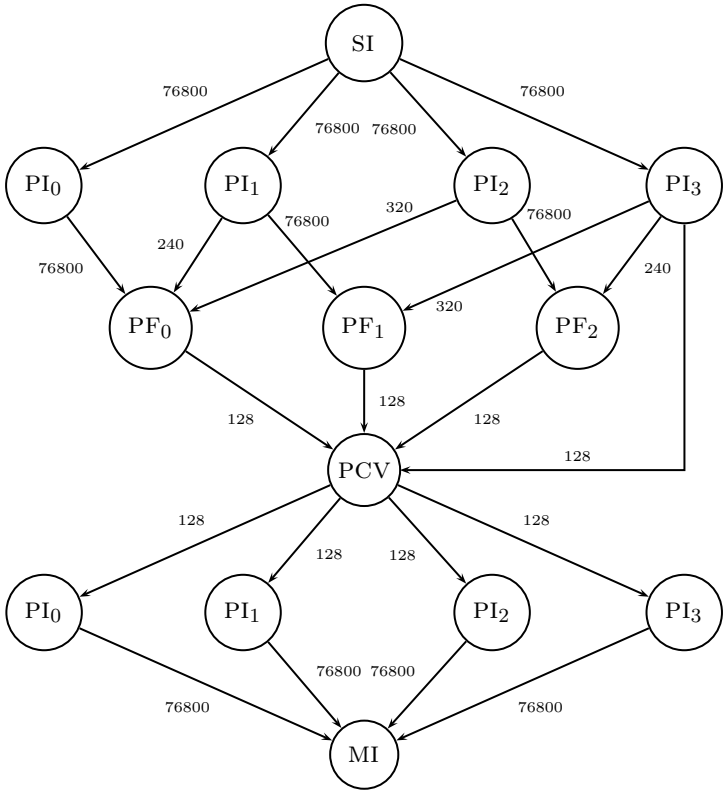


Figura 63: GT da aplicação *SegImag* para 4 segmentos de imagem

A aplicação *SegImag* foi submetida aos processos de alocação e mapeamento evolutivo realizados pelas implementações do NSGA-II e microGA. A complexidade desta aplicação é maior do que a complexidade das aplicações do E3S tendo em vista a maior quantidade de tarefas e dependências envolvidas. Os resultados obtidos por ambos os algoritmos está exposto na Tabela 28.

As alocações que deram origem aos mapeamentos de menor consumo de energia tanto no caso do NSGA-II quanto no do microGA são mostradas na Figura 64, onde os nós de mesma cor (cinza ou preta) mostram as tarefas que foram alocadas aos mesmos IPs e as restantes foram alocadas a IPs dedicados. Os mapeamentos de menor consumo de energia, obtidos pelo NSGA-II e pelo microGA, foram avaliados pelo CAFES. A partir das alocações destes mapeamentos, foram gerados CDGs para serem mapeados pelo CAFES. A Tabela 29 apresenta detalhadamente as alocações obtidas pelo NSGA-II e pelo microGA, e os mapeamentos obtidos pelo CAFES a partir destas alocações. O formato desta tabela se assemelha ao formato da Tabela 20, sendo que uma terceira representação de mapeamento foi introduzida, utilizando letras referentes aos mapeamentos da Figura 64.

A Tabela 30 apresenta dados referentes ao consumo de energia, tempo de execução, área

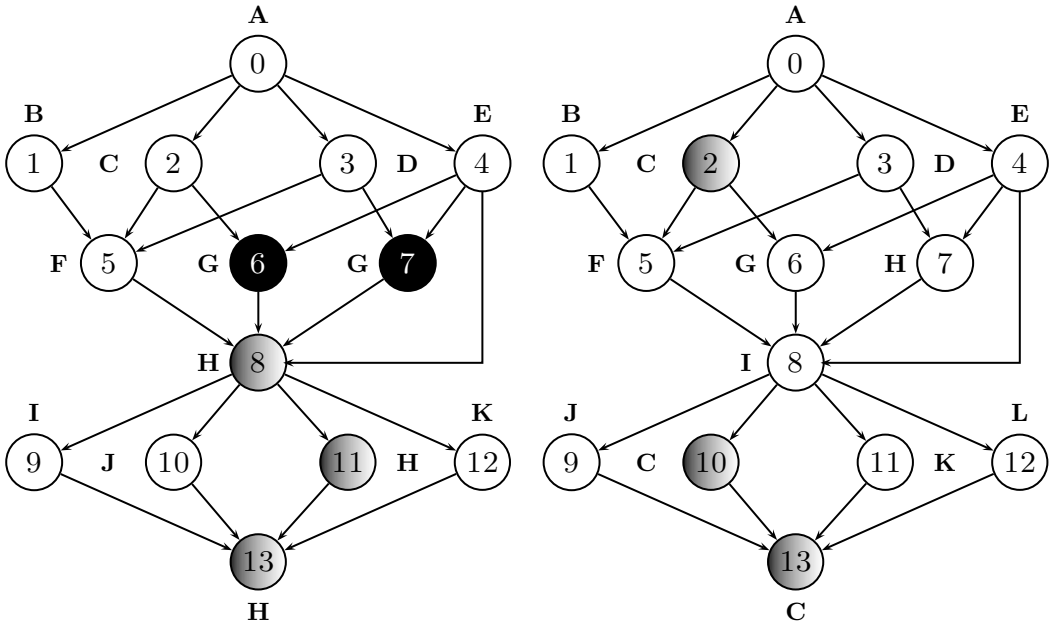
Tabela 28: Resultados obtidos pelo NSGA-II e pelo microGA para a aplicação SegImag

		NSGA-II	microGA
# Soluções		1.992	3.538
# Recursos	Mínimo	6	5
	Máximo	14	14
	Média	11,3	9,5
Médias	Consumo[(1)]	7,4509	10,6461
	Área[(2)]	30,3599	32,6802
	Tempo[(3)]	3.614,3	122.170,04
Mínimos	Consumo[(1)]	3,8250	3,8250
	Área[(2)]	14,5684	11,3484
	Tempo[(3)]	589,6462	594,8192
Tempo de busca[(4)]		03:03:34,0	01:00:40,0
Tempo por solução[(4)]		00:01:32,3	00:00:49,9
Desempenho		0,54	1,85

[1](W), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s), [4](hh:mm:ss,ms)

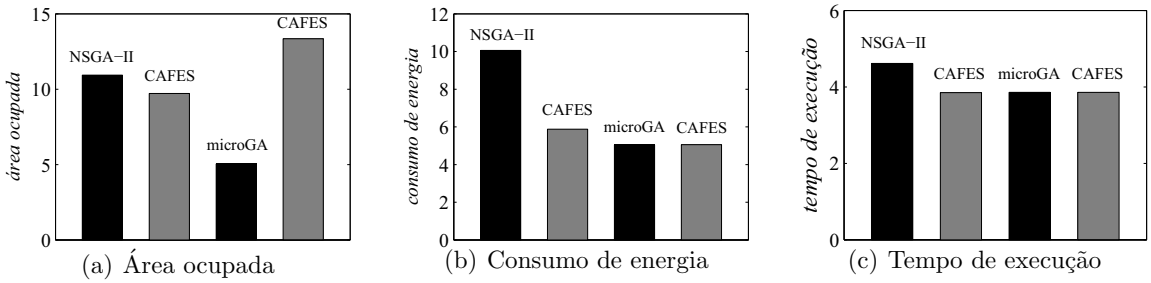
Tabela 29: Mapeamentos obtidos pelo NSGA-II, microGA e CAFES a partir de alocações não dominadas

AEM		Alocações e mapeamentos não dominados
NSGA–II	Alocação	[455, 449, 449, 449, 449, 371, 371, 371, 439, 449, 449, 449, 449, 454]
		Mapeamentos NSGA-II *vs.* CAFES
	NSGA–II	[1, 0, 3, 6, 14, 2, 10, 10, 4, 9, 5, 4, 13, 4] [15, 15, 13, 15, 15, 15, 15, –, –, 15, 13, –, –, 15, 15, –] [B, A, F, C, H, J, D, –, –, I, G, –, –, K, E, –]
	CAFES	[5, 7, 9, 1, 4, 13, 6, 6, 2, 3, 10, 2, 0, 2] [15, 15, 15, 15, 15, 15, 13, 15, –, 15, 15, –, –, 13, –, –] [K, D, H, I, E, A, G, B, –, C, J, –, –, F, –, –]
microGA	Alocação	[455, 449, 449, 449, 449, 371, 371, 371, 439, 449, 449, 449, 449, 454]
		Mapeamentos – microGA *vs.* CAFES
	microGA	[9, 5, 14, 13, 7, 12, 3, 2, 11, 15, 14, 10, 6, 14] [–, –, 13, 13, –, 15, 15, 15, –, 15, 15, 15, 13, 15, 15, 15] [–, –, G, H, –, L, E, I, –, K, C, J, F, D, A, B]
	CAFES	[11, 7, 2, 10, 15, 0, 13, 14, 5, 1, 2, 6, 3, 2] [13, 15, 15, 15, –, 15, 15, 15, –, –, 15, 15, –, 13, 13, 15] [F, B, A, E, –, J, C, L, –, –, D, K, –, H, G, I]



(a) Alocação usada pelo NSGA-II (b) Alocação usada pelo microGA

Figura 64: Alocações da aplicação *SegImag* que deram origem aos mapeamentos de consumo mínimo de energia usando o NSGA-II e microGA



(a) Área ocupada (b) Consumo de energia (c) Tempo de execução

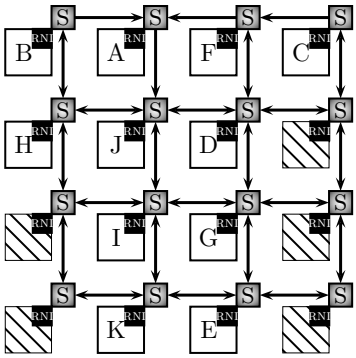
Figura 65: Comparação de mapeamentos entre NSGA-II e microGA com o CAFES

ocupada e quantidade de *switches* e canais utilizados nos mapeamentos obtidos pelo NSGA-II, microGA e CAFES. Os histogramas da Figura 65 ilustram as diferenças em relação aos objetivos de interesse; onde cada barra cinza, representando o CAFES, está relacionada com o algoritmo a sua esquerda, representado por uma barra preta. Observando esta figura é possível verificar que o microGA foi o algoritmo que encontrou o mapeamento de menor área e observando a Tabela 30, verifica-se também que o microGA encontrou o mapeamento de menor tempo de execução e menor consumo, tendo uma diferença pequena em relação ao CAFES, que é imperceptível graficamente. A Figura 66 ilustra como seria a distribuição espacial dos mapeamentos obtidos em uma plataforma NoC, topologia malha, com 16 recursos.

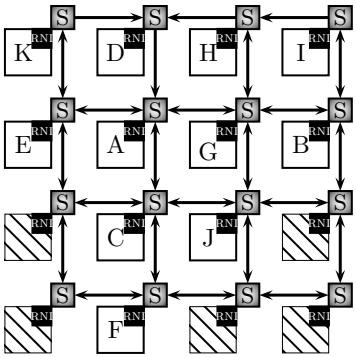
Tabela 30: Características dos mapeamentos obtidos para a aplicação SegImag

		NSGA-II	CAFES	microGA	CAFES
# *Switches*		13	12	13	15
# Canais		18	16	18	22
Comunicação	Consumo[1]	10,05562	5,87691	5,040	5,058
	Área[2]	10,93	9,72	10,93	13,35
	Tempo[3]	4,61345	3,85222	3,85602	3,85612
Total	Consumo[4]	3,8250100556	3,8250058769	3,825005040	3,825005058
	Área[2]	30,3024	29,0924	30,3024	32,7224
	Tempo[5]	5,1626451200	5,1618838900	3,21121425	3,21121435

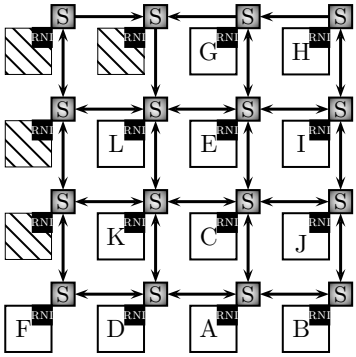
[1]($\times 10^{-6}$W^2), [2]($\times 10^{-6}$m^2), [3]($\times 10^{-3}$s), [4](W), [5](s)



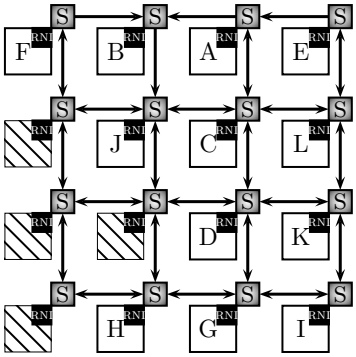
(a) Mapeamento obtido pelo NSGA-II



(b) Mapeamento obtido pelo CAFES



(c) Mapeamento obtido pelo microGA



(d) Mapeamento obtido pelo CAFES

Figura 66: Ilustração dos mapeamentos encontrados pelo NSGA-II, microGA e CAFES para a aplicação *SegImag*

5.6 Considerações Finais do Capítulo

O mapeamento de IPs em plataforma NoC consiste em determinar o espaço onde os IPs serão implementados fisicamente em uma plataforma NoC. Para que os IPs possam ser mapeados é preciso conhecer o grafo de caracterização da aplicação para saber a quantidade de recursos da plataforma NoC que são necessários para a implementação. De posse do grafo de caracterização de aplicação é preciso um método para gerar e avaliar diferentes mapeamentos, com o objetivo de encontrar o mapeamento que atenda às necessidades de projeto.

Neste capítulo foi apresentado o problema de mapeamento, sua complexidade e modelos utilizados para calcular o consumo de energia gasto pela comunicação dos IPs na plataforma NoC e o tempo de comunicação entre eles. Foram apresentadas diferentes classificações de algoritmos de roteamento e diferentes técnicas de chaveamento que podem ser utilizadas em plataforma NoC. Dois operadores genéticos, um de recombinação e outro de mutação, foram propostos com o objetivo de modificarem os indivíduos sem que os mesmos perdessem as características obtidas no processo evolutivo de alocação de IPs. Funções de avaliação foram definidas e utilizadas nas implementações dos algoritmos evolutivos multiobjetivos NSGA-II e microGA.

Uma aplicação de segmentação de imagem, de maior complexidade em relação às aplicações do E3S, foi apresentada e submetida ao processo de mapeamento evolutivo. Os resultados obtidos pelo NSGA-II e pelo microGA foram comparados com os resultados do sistema CAFES que é baseado no método estocástico de recozimento simulado. Os resultados obtidos mostraram que os mapeamentos obtidos pelos algoritmos evolucionários multiobjetivos são compatíveis com os mapeamentos obtidos pelo CAFES. Foi constatado que o microGA é capaz de obter um conjunto de soluções não dominadas mais extenso, com relação àquele obtido pelo NSGA-II. No próximo capítulo são apresentados os pontos mais relevantes abordados nesta dissertação e a possibilidade de trabalhos futuros.

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

ESTA dissertação apresentou um estudo sobre o emprego de algoritmos evolucionários multiobjetivos no auxílio a projetos baseados em plataforma NoC. Especificamente, estes algoritmos foram aplicados nas etapas de alocação e mapeamento de IPs. Dois algoritmos foram analisados para que a eficácia do método pudesse ser comprovada e o desempenho de ambos pudesse ser comparado.

6.1 Considerações Finais

Esta dissertação abordou a otimização das etapas de alocação de IPs e mapeamento de IPs em plataforma NoC utilizando algoritmos evolucionários multiobjetivos (AEMs). Este estudo foi motivado pela carência de ferramentas de EDA voltadas para este tipo de arquitetura (OGRAS; HU; MARCULESCU, 2005) e pela perspectiva de que em breve esta será a arquitetura predominante em sistemas embutidos. As duas etapas de projeto abordadas, influenciam bastante no desempenho da implementação final e representam problemas de alta complexidade a serem resolvidos. Dependendo da quantidade de tarefas que uma aplicação possui, da quantidade de IPs possíveis a serem escolhidos para alocar às tarefas e da quantidade de recursos a serem mapeados, a resolução destes problemas pode ser difícil até mesmo com o auxílio de uma ferramenta computacional.

A escolha de utilizar AEMs para a alocação e mapeamento de IPs, foi baseada na natureza destes problemas. Após analisá-los, foram identificados três objetivos de interesse que deveriam ser minimizados, sendo eles: o consumo de energia, o tempo de execução da aplicação implementada e a área ocupada pela NoC. Como estes problemas possuem múltiplos objetivos que devem ser alcançados e restrições que devem ser respeitadas durante o projeto, para não comprometer o resultado final, eles são caracterizados como problemas de otimização

multiobjetivos (POMs). Problemas como estes, normalmente possuem um espaço de busca muito grande, tornando inviável realizar uma buscar exaustiva pela melhor solução, até mesmo porque pode não existir uma única melhor solução, mas sim um conjunto de soluções ditas não dominadas que apresentam um balanço em termos dos objetivos considerados.

Foram utilizados dois algoritmos evolucionários multiobjetivos para que os resultados obtidos, assim como os próprios algoritmos, pudessem ser comparados. Os algoritmos evolucionários multiobjetivos escolhidos foram o NSGA-II e o microGA. Estes algoritmos foram selecionados devido aos bons resultados obtidos em *benchmarks* para POMs (DEB *et al.*, 2002) (COELLO; PULIDO, 2001) e por apresentarem diferentes técnicas de evolução, o que justifica a comparação dos resultados. Outro fator importante para a escolha destes algoritmos foi o método de seleção e classificação utilizado, que é o método baseado em dominância. Este método é interessante pois ao final do processo evolutivo é obtido um conjunto de soluções ótimas considerando os três objetivos já citados.

Escolhidos os algoritmos de busca e otimização, foi necessário definir uma fonte de dados que seria utilizada como repositório de IPs, codificar os indivíduos que representariam as soluções dos problemas, adaptar os operadores genéticos de acordo com a codificação dos indivíduos, determinar as funções objetivo de avaliação dos indivíduos e analisar os resultados obtidos.

O repositório utilizado foi o E3S (DICK, 2008), que contêm dados de processadores, tarefas e aplicações de *benchmark*. Este repositório foi todo re-escrito em XML (W3C, 2008) para facilitar a integração com os algoritmos evolucionários implementados. A codificação dos indivíduos foi feita em duas etapas, uma mais adequada para o problema de alocação de IPs e outra mais adequada para o problema de mapeamento de IPs.

Para lidar com o problema de alocação de IPs, foi implementada a alocação evolutiva de IPs (SILVA; NEDJAH; MOURELLE, 2009a) (SILVA; NEDJAH; MOURELLE, 2009c). Os algoritmos evolucionários foram modificados de modo que os genes dos indivíduos pudessem representar tarefas do grafo de tarefas, de uma dada aplicação, e que IPs pudessem ser alocados às tarefas, representando então uma solução. Para garantir sempre a geração de soluções factíveis, foi utilizado o operador genético de recombinação em um único ponto e ao operador genético de mutação, foi imposta a restrição de não alocar IPs incapazes de executar a tarefa representada pelo gene. Funções de avaliação para cada objetivo de interesse foram desenvolvidas a partir do estudo de impacto da alocação no desempenho da implementação da NoC. Nesta etapa, a abstração da descrição do projeto omite detalhes referentes à aspectos físicos da plataforma

NoC. A otimização foi feita com base nos dados dos IPs e do grafo de tarefas, que por sinal, representam as entradas iniciais dos AEMs implementados. A análise dos resultados obtidos revelou que ambos os algoritmos encontraram uma grande quantidade de soluções em comum. Porém, na maioria dos problemas, o microGA encontrou mais soluções, em menor tempo e com valores mínimos menores.

Para o problema de mapeamento de IPs, foi implementado o mapeamento evolutivo de IPs (SILVA; NEDJAH; MOURELLE, 2009b). Os algoritmos evolucionários foram implementados de modo que os genes dos indivíduos pudessem representar a plataforma NoC, que seria o alvo do mapeamento, e que os IPs alocados, através da alocação evolutiva de IPs, pudessem ser mapeados, representando então uma solução. Para garantir sempre a geração de soluções factíveis e para preservar a solução obtida no processo anterior, foi utilizado o operador genético de recombinação por deslocamento e o operador genético de mutação interna. A recombinação de duas soluções obtidas no processo de alocação, resultaria na destruição destas soluções. A recombinação por deslocamento foi inspirada no processo biológico conhecido como partenogênese (WATTS *et al.*, 2006), onde um novo indivíduo é gerado a partir do material genético de um único indivíduo, sendo que o novo indivíduo gerado não é um clone do indivíduo gerador. Este mecanismo de recombinação mantem o caráter evolucionário, que é a inspiração dos AEMs, e ao mesmo tempo mantem a diversidade genética da população. O operador de mutação interna realiza uma troca de posição entre os genes do indivíduo, desconsiderando o ambiente externo. Funções de avaliação para cada objetivo de interesse foram desenvolvidas a partir do estudo de impacto do mapeamento no desempenho da implementação da NoC. Nesta etapa, fazem parte da descrição do projeto, detalhes referentes à aspectos físicos da plataforma NoC. A otimização foi feita com base nas alocações obtidas e nos aspectos físicos da plataforma, que nesta etapa, representam as entradas dos AEMs implementados. O algoritmo de roteamento XY e a técnica de chaveamento *wormhole*, foram usados na avaliação da aptidão dos mapeamentos evoluídos. Além disso, a topologia de malha foi usada como estrutura de NoC. A análise dos resultados obtidos, revelou que na maioria dos problemas, o microGA encontrou mais soluções em menos tempo, comparado com o NSGA-II.

Os mapeamentos obtidos por ambos os algoritmos foram comparados com os mapeamentos obtidos pelo sistema CAFES (MARCON *et al.*, 2005b). O CAFES utiliza a técnica de busca e otimização de recozimento simulado (RUSSEL; NORVIG, 1995). Os mapeamentos obtidos pelos AEMs implementados e pelo CAFES foram compatíveis. Dada uma alocação de IPs, o NSGA-II não foi capaz de encontrar mapeamentos tão bons do que os mapeamentos

obtidos pelo CAFES. No entanto, o microGA foi capaz de encontrar mapeamentos tão bons e até melhores do que os mapeamentos obtidos pelo CAFES.

Os métodos propostos de alocação evolutiva de IPs e mapeamento evolutivo de IPs, utilizando o NSGA-II e o microGA, mostraram ser eficientes no auxílio a projetos baseados em plataforma NoC. Os resultados experimentais demonstraram a maior eficiência do microGA em relação ao NSGA-II, já que o primeiro foi capaz de encontrar melhores soluções em menor tempo.

6.2 Trabalhos Futuros

Diferentes alterações podem ser feitas para explorar ainda mais a capacidade de busca dos AEMs. Como estes algoritmos priorizam as soluções mais aptas ao meio em que vivem, diferenças neste meio poderiam revelar uma maior eficácia e talvez soluções inovadoras. Considerando como o meio, a plataforma NoC, uma mudança que poderia causar grande impacto é a mudança do algoritmo de roteamento adotado. Um algoritmo de roteamento evolutivo poderia ser adotado para que as soluções se adaptassem de acordo com o perfil de comunicação entre os IPs e a localização dos mesmos na plataforma.

Uma outra alteração no meio seria a mudança de topologia utilizada que poderia ser evoluída em conjunto com o mapeamento e o roteamento. Estas novas possibilidades de evoluções aumentam muito a complexidade do problema, fazendo necessária a investigação de uma técnica mais eficiente para lidar com um problema desta grandeza. Uma alternativa seria a utilização de AEMs distribuídos usando a técnica de co-evoluções.

Por fim, estas etapas poderiam ser integradas a uma ferramenta de EDA para complementar as demais etapas de um projeto baseado em plataforma.

REFERÊNCIAS

ANTON, H. *Calculus with Analytic Geometry*. 3rd. ed. New York: John Wiley & Sons, 1988.

BÄCK, T. *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1996.

BARATI, H. *et al.* Routing algorithms study and comparing in interconnection networks. In: *3rd International Conference on Information and Communication Technologies: From Theory to Applications*. Los Alamitos, CA: IEEE Press, 2008. p. 1–5.

BENINI, L.; MICHELI, G. D. Networks on chips: A new SoC paradigm. *Computer*, v. 35, n. 1, p. 70–78, Jan. 2002.

BLICKLE, T.; THIELE, L. A mathematical analysis of tournament selection. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann, 1995. p. 9–16.

BRASSARD, G.; BRATLEY, P. *Algorithmics: theory & practice*. Upper Saddle River, NJ: Prentice-Hall, 1988.

CARDOZO, R. da S. *Redes-em-Chip de Baixo Custo*. 75 p. Tese (Mestrado) — Instituto de Informática – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2005.

CHARNES, A.; COOPER, W. W. *Management models and industrial applications of linear programming*. 2nd. ed. New York, NY: John Wiley & Sons, 1967.

COELLO, C. A.; PULIDO, G. T. Multiobjective structural optimization using a micro-genetic algorithm. *Structural and Multidisciplinary Optimization*, v. 30, n. 5, p. 388–403, Nov. 2005.

COELLO, C. A. C. *An empirical study of evolutionary techniques for multiobjective optimization in engineering design*. Tese (Doctor of Philosophy) — Tulane University Department of Computer Science, New Orleans, LA, 1996.

COELLO, C. A. C. A short tutorial on evolutionary multiobjective optimization. In: *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*. London, UK: Springer, 2001. p. 21–40.

COELLO, C. A. C.; PULIDO, G. T. A micro-genetic algorithm for multiobjective optimization. *Lecture Notes in Computer Science*, v. 1993, p. 126–138, 2001.

DARWIN, C. R. *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life*. London, UK: Murray, 1859.

DAY, J.; ZIMMERMANN, H. The OSI reference model. *Proceedings of the IEEE*, v. 71, n. 12, p. 1334–1340, Dez. 1983.

DEB, K.; GOLDBERG, D. E. An investigation of niche and species formation in genetic function optimization. In: *Proceedings of the Third International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufman, 1989. p. 42–50.

DEB, K. *et al.* A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, v. 6, p. 182–197, Abril 2002.

DEMMELER, T.; GIUSTO, P. A universal communication model for an automotive system integration platform. In: *Proceedings of the conference on Design, automation and test in Europe*. Los Alamitos, CA: IEEE Press, 2001. p. 47–54.

DICK, R. P. *Embedded System Synthesis Benchmarks Suite (E3S)*. 2008. Disponível em: <http://ziyang.eecs.northwestern.edu/ dickrp/e3s/>. Acesso em: 10 Mar. 2008.

DORIGO, M.; MANIEZZO, V. Parallel genetic algorithms: Introduction and overview of current research. In: STENDERS, J. (Ed.). *Parallel Genetic Algorithms: Theory and Applications*. Amsterdam, Holanda: IOS Press, 1993. p. 5–42.

DUATO, J.; YALAMANCHILI, S.; NI, L. *Interconnection Networks: An Engineering Approach*. San Francisco, CA: Morgan Kaufmann, 2003. xxiii + 600 p.

EDGEWORTH, F. Y. *Mathematical Psychics: An Essay on the Application of Mathematics to the Moral Sciences*. London, UK: C. Kegan Paul and Co., 1881.

EDWARDS, S. *et al.* Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, v. 85, n. 3, p. 366–390, Mar. 1997.

ENGELBRECHT, A. P. *Fundamentals of Computational Swarm Intelligence*. New York, NY: John Wiley & Sons, 2006.

FOGEL, D. B. Chapter introduction. In: BÄCK, T.; FOGEL, D.; MICHALEWICZ, Z. (Ed.). *Handbook of Evolutionary Computation*. Bristol, UK: IOP Press and Oxford University Press, 1997. v. 1, p. A.1.1:1–A.1.1:2.

FOGEL, D. B.; GHOZEIL, A. A note on representations and variation operators. *IEEE Trans. on Evolutionary Computation*, v. 1, n. 2, p. 159–161, 1997.

FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. *Artificial Intelligence through Simulated Evolution*. New York, NY: John Wiley & Sons, 1966.

FONSECA, C.; FLEMING, P. Multiobjective optimization and multiple constraint handling with evolutionary algorithms-Part ii. application example. *IEEE Transactions on Man and Cybernetics Systems–Part A*, v. 28, n. 1, p. 38–47, Jan. 1998.

FONSECA, C. M.; FLEMING, P. J. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: *Genetic Algorithms: Proceedings of the Fifth International Conference*. San Francisco, CA: Morgan Kaufmann, 1993. p. 416–423.

GAJSKI, D. D. *et al. Specification and design of embedded systems*. Upper Saddle River, NJ: Prentice-Hall, 1994.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability; a guide to the theory of NP-completeness*. USA: W. H. Freeman, 1979.

GIZOPOULOS, D.; PASCHALIS, A.; ZORIAN, Y. *Embedded Processor-based Self-test*. New York, NY: Springer, 2004.

GLOVER, F.; LAGUNA, F. *Tabu Search*. Norwell, MA: Kluwer Academic Publishers, 1997.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA: Addison-Wesley Press, 1989.

GOLDBERG, D. E. Sizing populations for serial and parallel genetic algorithms. In: *Proceedings of the third international conference on Genetic algorithms*. San Francisco, CA: Morgan Kaufmann, 1989. p. 70–79.

GREFENSTETTE, J. J. GENESIS: A system for using genetic search procedures. In: *Proceedings of the 1984 Conference on Intelligent Systems and Machines*. Rochester, MI: Oakland University, 1984. p. 161–165.

GUPTA, R.; CLAUDIONOR N.C., J.; MICHELI, G. D. Program implementation schemes for hardware-software systems. *Computer*, v. 27, n. 1, p. 48–55, Jan 1994.

GUPTA, R.; ZORIAN, Y. Introducing core-based system design. *IEEE Design & Test of Computers*, v. 14, n. 4, p. 15–25, Out-Dez 1997.

HILLIER, F. S.; LIEBERMAN, G. J. *Introduction to Operations Research*. 8th. ed. New York, NY: McGraw-Hill, 2005.

HORN, J. Multicriterion decision making. In: BÄCK, T.; FOGEL, D.; MICHALEWICZ, Z. (Ed.). *Handbook of Evolutionary Computation*. Bristol, UK: IOP Press and Oxford University Press, 1997. v. 1, p. F1.9:1–F1.9:15.

HORN, J.; NAFPLIOTIS, N.; GOLDBERG, D. E. A niched pareto genetic algorithm for multiobjective optimization. In: *International Conference on Evolutionary Computation*. Los Alamitos, CA: IEEE Press, 1994. p. 82–87.

HU, J.; MARCULESCU, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. In: *Proceedings of the 2003 conference on Asia South Pacific design automation*. New York, NY: ACM, 2003. p. 233–239.

HUSBANDS, P. Genetic algorithms in optimisation and adaptation. John Wiley & Sons, New York, NY, p. 227–276, 1992.

IEEE. *VASG: VHDL Analysis and Standardization Group*. 2009. Disponível em: <http://www.eda.org/vasg/>. Acesso em: 22 Mar. 2008.

IJIRI, J. *Management Goals and Accounting for Control*. New York, NY: Elsevier, 1965.

JASZKIEWICZ, A. Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, v. 137, n. 1, p. 50–71, 2002.

JERRAYA, A. A.; TENHUNEN, H.; WOLF, W. Guest editors' introduction: Multiprocessor Systems-on-Chips. *IEEE Computer*, v. 38, n. 7, p. 36–40, 2005.

JOHNSON, E. G.; ABUSHAGUR, M. A. G. Microgenetic-algorithm optimization methods applied to dielectric gratings. *Journal of the Optical Society of America*, Optical Society of America, v. 12, p. 1152–1160, 1995.

KERNIGHAN, B. W.; RITCHIE, D. M. *The C Programming Language*. 2nd. ed. New York, NY: Prentice-Hall, 1988.

KEUTZER, K. *et al.* System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 19, p. 1523–1543, 2000.

KNOWLES, J. D.; CORNE, D. W. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, MIT Press, v. 8, n. 2, p. 149–172, 2000.

KNUTH, D. E. *The Art of Computer Programming.* Vol. 3: *Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

KRISHNAKUMAR, K. Micro-genetic algorithms for stationary and non-stationary function optimization. In: *Intelligent Control and Adaptive Systems*. Philadelphia, PA: SPIE, 1990. v. 1196, p. 289–296.

KUMAR, S. *et al.* A network on chip architecture and design methodology. In: *ISVLSI '02: Proceedings of the IEEE Computer Society Annual Symposium on VLSI*. Los Alamitos, CA: IEEE Press, 2002. p. 117–124.

KURSAWE, F. A variant of evolution strategies for vector optimization. In: *1st Workshop in Parallel Problem Solving from Nature*. Berlin: Springer, 1991. (Lecture Notes in Computer Science, v. 496), p. 193–197.

LAUMANNS, M.; RUDOLPH, G.; SCHWEFEL, H. paul. *Approximating the Pareto Set: Concepts, Diversity Issues, and Performance Assessment*. Dortmund, 1999.

LAUMANNS, M. *et al.* Combining convergence and diversity in evolutionary multiobjective optimization. *Evol. Comput.*, MIT Press, Cambridge, MA, v. 10, n. 3, p. 263–282, 2002.

MADISETTI, V. K.; SHEN, L. Interface design for core-based systems. *IEEE Design and Test of Computers*, IEEE Computer Society Press, Los Alamitos, CA, v. 14, n. 4, p. 42–51, Oct–Dec 1997.

MARCON, C. *et al.* Exploring NoC mapping strategies: An energy and timing aware technique. In: *Proceedings of the conference on Design, Automation and Test*. Washington, DC: IEEE Computer Society, 2005c. p. 502–507.

MARCON, C. A. M. *Modelos para o Mapeamento de Aplicações em Infra-Estruturas de Comunicação Intrachip*. 188 p. Tese (Doutorado) — Instituto de Informática – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2005.

MARCON, C. A. M. *et al.* Modeling the traffic effect for the application cores mapping problem onto nocs. In: REIS, R. A. da L.; OSSEIRAN, A.; PFLEIDERER, H.-J. (Ed.). *VLSI-SoC*. Berlin: Springer, 2005b. (IFIP, v. 240), p. 179–194.

MARCON, C. A. M. *et al.* Time and energy efficient mapping of embedded applications onto nocs. In: TANG, T.-A. (Ed.). *Proceedings of the 2005 Conference on Asia South Pacific Design Automation*. New York, NY: ACM Press, 2005a. p. 33–38.

MARSLAND, T. A.; YANG, Z. *Global States and Time in Distributed Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1994.

MERKLE, L. D.; LAMONT, G. B. A random function based framework for evolutionary algorithms. In: BÄCK, T. (Ed.). *Proc. of the Seventh Int. Conf. on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann, 1997. p. 105–112.

METCALFE, R. M.; BOGGS, D. R. Ethernet: distributed packet switching for local computer networks. *Commun. ACM*, ACM Press, New York, NY, v. 19, n. 7, p. 395–404, July 1976.

MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs*. 2nd. ed. New York, NY: Springer, 1994.

MOORE, G. E. Cramming more components onto integrated circuits. *Electronics*, v. 38, n. 8, p. 114–117, 1965.

NASH, J. Non-cooperative games. *The Annals of Mathematics*, Annals of Mathematics, v. 54, n. 2, p. 286–295, 1951.

NASH, J. F. Equilibrium points in n-person games. In: *Proceedings of the National Academy of Sciences of the United States of America*. Washington, DC: National Academy of Sciences of the United States of America, 1950.

NEAPOLITAN, R. E.; NAIMIPOUR, K. *Foundations of algorithms*. Lexington, MA: D. C. Heath and Company, 1996.

NEUMANN, J. V.; MORGENSTERN, O. *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press, 1944.

NI, L. M.; MCKINLEY, P. K. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, v. 26, n. 2, p. 62, feb 1993.

OGRAS, Ü. Y.; HU, J.; MARCULESCU, R. Key research problems in NoC design: a holistic perspective. In: ELES, P.; JANTSCH, A.; BERGAMASCHI, R. A. (Ed.). *Proceedings of the 3rd International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY: ACM, 2005. p. 69–74.

PALMA, J. *et al.* Core communication interface for FPGAs. In: *15th Symposium on Integrated Circuits and Systems Design*. Los Alamitos, CA: IEEE Press, 2002. p. 183–188.

PANDE, P. *et al.* Design, synthesis, and test of networks on chips. *IEEE Design & Test of Computers*, v. 22, n. 5, p. 404–413, Sept.-Oct. 2005.

PARETO, V. *Cours D'Economie Politique*. Lausanne: F. Rouge, 1896.

PEARL, J. *Heuristics: Intelligent search strategies for computer problem solving*. Boston, MA: Addison-Wesley, 1989.

RAVINDRAN, A.; PHILIPS, D. T.; SOLBERG, J. J. *Operations Research - Principles and Practice*. 2nd. ed. New York: John Wiley & Sons, 1987.

RITZEL, B. J.; EHEART, J. W.; RANJITHAN, S. Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, v. 30, n. 5, p. 1589–1603, 1994.

ROBERT, C. P.; CASELLA, G. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Secaucus, NJ: Springer, 2005.

RUSSEL, S.; NORVIG, P. *Artificial Intelligence: a Modern Approach*. New York, NY: Prentice-Hall, 1995.

SCHAFFER, J. D. Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Mahwah, NJ: Lawrence Erlbaum Associates, 1985. p. 93–100.

SCHWEFEL, H.-P. P. *Evolution and Optimum Seeking: The Sixth Generation*. New York, NY: John Wiley & Sons, 1995.

SGROI, M. *et al.* Addressing the system-on-a-chip interconnect woes through communication-based design. In: *Proceedings of the 38th conference on Design automation*. New York, NY: ACM, 2001. p. 667–672.

SIEWERT, S. *SoC drawer: The resource view*. Oct 2005. Disponível em: <`http://www.ibm.com/developerworks/power/library/pa-soc1/index.html?S_TACT=105AGX16&S_CMP=EDU#h1`>. Acesso em: 11 Dez. de 2008.

SILVA, M. V. C. da; NEDJAH, N.; MOURELLE, L. de M. Evolutionary ip assignment for efficient NoC-based system design using multi-objective optimization. In: *IEEE Congress of Evolutionary Computation*. Los Alamitos, CA: IEEE Computer Science Press, 2009. *Aceito*.

SILVA, M. V. C. da; NEDJAH, N.; MOURELLE, L. de M. Optimal application mapping on NoC infrastructure using NSGA-II and MicroGA. In: *IEEE International Conference on Intelligent Engineering Systems*. Los Alamitos, CA: IEEE Computer Science Press, 2009. *Aceito*.

SILVA, M. V. C. da; NEDJAH, N.; MOURELLE, L. de M. Optimal ip assignment for efficient NoC-based system implementation using NSGA-II and MicroGA. *International Journal of Computational Intelligence Systems*, Atlantis Press, Amesterdam, Holanda, 2009. *Aceito*.

SOININEN, J.-P.; HEUSALA, H. A design methodology for NoC-based systems. Kluwer Academic Publishers, Hingham, MA, p. 19–38, 2003.

SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, v. 2, p. 221–248, 1994.

SYSTEMC, C. *SystemC 2.1 Language Reference Manual*. 2009. Disponível em: <http://www.systemc.org>. Acesso em: 19 Ago. 2008.

TANENBAUM, A. S. *Structured Computer Organization*. 5th. ed. Upper Saddle River, NJ: Prentice-Hall, 2005.

VAHID, F.; GIVARGIS, T. Platform tuning for embedded systems design. *Computer*, v. 34, n. 2, p. 112–114, Feb 2001.

VELDHUIZE, D. A. V.; LAMONT, G. B. Evolutionary computation and convergence to a pareto front. In: KOZA, J. R. (Ed.). *Late Breaking Papers at the Genetic Programming 1998 Conference*. University of Wisconsin, Madison, Wisconsin, USA: Stanford University Bookstore, 1998.

VELDHUIZEN, D. A. V. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and Newc Innovations*. Tese (Doctor of Philosophy) — Department of Electrical and Computer Engineering – Graduate School of Engineering – Air Force Institute of Technology, Wright-Patterson AFB, Ohio, Maio 1999.

W3C. *World Wide Web Consortium*. 2008. Disponível em: <http://www.w3.org>. Acesso em: 15 Jul. 2008.

WATTS, P. C. *et al.* Parthenogenesis in komodo dragons. *Nature*, Nature Publishing Group, v. 444, n. 7122, p. 1021–1022, Dez. 2006.

WHITE, W. W. A status report on computing algorithms for mathematical programming. *ACM Comput. Surv.*, ACM, New York, NY, v. 5, n. 3, p. 135–166, 1973.

WOLF, W. *Computers as components: principles of embedded computing system design*. San Francisco, CA: Morgan Kaufmann, 2001.

XIAO, F.; YABE, H. Microgenetic-algorithm optimization methods applied to dielectric gratings. *EICE transactions on Electronics*, IEICE, E81-C, n. 12, p. 1784–1792, 1998.

YANG, J. Minimax reference point approach and its application for multiobjective optimisation. *European Journal of Operational Research*, v. 126, n. 3, p. 541–556, 2000.

YE, T. T.; MICHELI, G. D.; BENINI, L. Analysis of power consumption on switch fabrics in network routers. In: *Proceedings of the 39th conference on Design automation*. New York, NY: ACM, 2002. p. 524–529.

ZEFERINO, C. A. *Redes-em-Chip: Arquiteturas e Modelos para Avaliação de Área e Desempenho*. 242 p. Tese (Doutorado) — Instituto de Informática – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2003.

ZHOU, W.; ZHANG, Y.; MAO, Z. Pareto based multi-objective mapping IP cores onto NoC architectures. In: *APCCAS*. Los Alamitos, CA: IEEE Press, 2006. p. 331–334.

ZITZLER, E.; DEB, K.; THIELE, L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, MIT Press, v. 8, n. 2, p. 173–195, Summer 2000.

ZITZLER, E.; THIELE, L. Multiobjective optimization using evolutionary algorithms: A comparative case study. In: EIBEN, A. E. *et al.* (Ed.). *Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V)*. Berlin: Springer, 1998. p. 292–301.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *Evolutionary Computation, IEEE Transactions on*, v. 3, n. 4, p. 257–271, Nov 1999.

APÊNDICE A – Processadores e Tarefas do Repositório

Este apêndice apresenta detalhes sobre o conteúdo do repositório E3S, disponível no endereço eletrônico http://ziyang.eecs.northwestern.edu/~dickrp/e3s/. Os dados apresentados são estruturados em cinco seções, onde são listados os tipos de processadores e tarefas disponíveis, as relações processador/tarefa e tarefa/processador/IP, e as características de interesse de 6 processadores usados como exemplo nas explicações prestadas no Capítulo 4 e Capítulo 5.

A.1 Lista de Processadores

A Tabela 31 mostra todos os 17 tipos de processadores disponibilizados no repositório, juntamente com as respectivas frequências de operação e o número total de tarefas diferentes que cada processador permite implementar.

Tabela 31: Lista dos processadores do repositório

ID Proc	Nome	Frequência (MHz)	Número de Tarefas
0	AMD ElanSC520	133	46
1	AMD K6-2E	450	6
2	AMD K6-2E	400	46
3	AMD K6-2E+	500	46
4	AMD K6-IIIE+	550	41
5	Analog Devices 21065L	60	17
6	IBM PowerPC 405GP	266	46
7	IBM PowerPC 750CX	500	36
8	IDT32334	100	19
9	IDT79RC32364	100	22
10	IDT79RC32V334	150	19
11	IDT79RC64575	250	22
12	Imsys Cjip	40	4
13	Motorola MPC555	40	17
14	NEC VR5432	167	46
15	ST20C2	50	30
16	TI TMS320C6203	300	17

A.2 Lista de Tarefas

A Tabela 32 e a Tabela 33 listam os 46 diferentes tipo de tarefas usadas no repositório. As tarefas são dadas pela descrição da funcionalidade principal que esta implementa, juntamente com o tamanho dos dados de entrada, em casos específicos.

Tabela 32: Lista das tarefas usadas no repositório

Tipo de tarefa	Descrição da tarefa
0	Angle to Time Conversion
1	Basic floating point
2	Bit Manipulation
3	Cache Buster
4	CAN Remote Data Request
5	Fast Fourier Transform (Auto/Indust. Version)
6	Finite Impulse Response Filter (Auto/Indust. Vers)
7	Infinite Impulse Response Filter
8	Inverse discrete cosine transfom
9	Inverse Fast Fourier Transform (Auto/Indust. Vers)
10	Matrix arithmetic
11	Pointer Chasing
12	Pulse Width Modulation
13	Road Speed Calculation
14	Table Lookup and Interpolation
15	Tooth To Spark
16	OSPF/Dijkstra
17	Route Lookup/Patricia
18	Packet Flow – 512 kbytes
19	Packet Flow – 1 Mbyte
20	Packet Flow – 2 Mbytes
21	Autocorrelation – Data1 (pulse)
22	Autocorrelation – Data2 (sine)
23	Autocorrelation – Data3 (speech)
24	Convolutional Encoder – Data1 (xk5r2dt)
25	Convolutional Encoder – Data2 (xk4r2dt)
26	Convolutional Encoder – Data3 (xk3r2dt)
27	Fixed-point Bit Allocation – Data2 (typ)
28	Fixed-point Bit Allocation – Data3 (step)
29	Fixed Point Bit Allocation – Data6 (pent)
30	Fixed Point Complex FFT – Data1 (pulse)
31	Fixed point Complex FFT – Data2 (spn)
32	Fixed Point Complex FFT – Data3 (sine)
33	Viterbi GSM Decoder – Data1 (get)

Tabela 33: Lista das tarefas usadas no repositório – (Continuação)

Tipo de tarefa	Descrição da tarefa
34	Viterbi GSM Decoder – Data2 (toggle)
35	Viterbi GSM Decoder – Data3 (ones)
36	Viterbi GSM Decoder – Data4 (zeros)
37	Compress JPEG
38	Decompress JPEG
39	High Pass Grey-scale filter
40	RGB to CYMK Conversion
41	RGB to YIQ Conversion
42	Dithering
43	Image Rotation
44	Text Processing
45	src-sink

A.3 Lista de Tarefas por Processador

A Tabela 34 mostra, para cada um dos 17 processadores, o conjunto das diferentes tarefas que cada um é capaz de executar. Os processadores e as tarefas são identificados conforme consta na Tabele 31 e na Tabela 32, respectivamente.

A.4 Lista de Processadores por Tarefa

A Tabela 35 mostra, para cada uma dos 46 diferentes tipos de tarefas usadas, o número de processadores distintos que podem ser utilizados para executá-la e o conjunto dos identificadores desses processadores. O par (processador, tarefa) identifica um IP, cujo número é também mostrado.

A.5 Características de IPs

As características de alguns IPs (i.e. tarefa/processador) são listados para 6 diferentes processadores juntamente com 6 tarefas. Esses detalhes são usados nos exemplos ilustrativos apresentados no Capítulo 4 e Capítulo 5. Para cada IP são mostrados os respectivos consumo de energia, tempo de exceução e área de *hardware*.

Tabela 34: Lista de tarefas por processador

ID Proc.	Tipo de tarefas
0	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45}
1	{16, 17, 18, 19, 20, 45}
2	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45}
3	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 6, 37, 38, 39, 40, 41, 42, 43, 44, 45}
4	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 42, 43, 44, 45}
5	{21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 45}
6	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45}
7	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 42, 43, 44, 45}
8	{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 32, 33, 34, 35, 36, 45}
9	{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 45}
10	{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 30, 31, 32, 33, 34, 35, 36, 45}
11	{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 45}
12	{42, 43, 44, 45}
13	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 45}
14	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45}
15	{16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45}
16	{21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 45}

Tabela 35: Lista de processador por tarefa

Tipo de tarefa	#Processadores	*ProcID/IP ID*
0	8	{0/0, 2/52, 3/98, 4/144, 6/202, 7/248, 13/370, 14/387}
1	8	{0/1, 2/53, 3/99, 4/145, 6/203, 7/249, 13/371, 14/388}
2	8	{0/2, 2/54, 3/100, 4/146, 6/204, 7/250, 13/372, 14/389}
3	8	{0/3, 2/55, 3/101, 4/147, 6/205, 7/251, 13/373, 14/390}
4	8	{0/4, 2/56, 3/102, 4/148, 6/206, 7/252, 13/374, 14/391}
5	8	{0/5, 2/57, 3/103, 4/149, 6/207, 7/253, 13/375, 14/392}
6	8	{0/6, 2/58, 3/104, 4/150, 6/208, 7/254, 13/376, 14/393}
7	8	{0/7, 2/59, 3/105, 4/151, 6/209, 7/255, 13/377, 14/394}
8	8	{0/8, 2/60, 3/106, 4/152, 6/210, 7/256, 13/378, 14/395}
9	8	{0/9, 2/61, 3/107, 4/153, 6/211, 7/257, 13/379, 14/396}
10	8	{0/10, 2/62, 3/108, 4/154, 6/212, 7/258, 13/380, 14/397}
11	8	{0/11, 2/63, 3/109, 4/155, 6/213, 7/259, 13/381, 14/398}
12	8	{0/12, 2/64, 3/110, 4/156, 6/214, 7/260, 13/382, 14/399}
13	8	{0/13, 2/65, 3/111, 4/157, 6/215, 7/261, 13/383, 14/400}
14	8	{0/14, 2/66, 3/112, 4/158, 6/216, 7/262, 13/384, 14/401}
15	8	{0/15, 2/67, 3/113, 4/159, 6/217, 7/263, 13/385, 14/402}
16	12	{0/16, 1/46, 2/68, 3/114, 4/160, 6/218, 8/284, 9/303, 10/325, 11/344, 14/403, 15/433}
17	12	{0/17, 1/47, 2/69, 3/115, 4/161, 6/219, 8/285, 9/304, 10/326, 11/345, 14/404, 15/434}
18	12	{0/18, 1/48, 2/70, 3/116, 4/162, 6/220, 8/286, 9/305, 10/327, 11/346, 14/405, 15/435}
19	12	{0/19, 1/49, 2/71, 3/117, 4/163, 6/221, 8/287, 9/306, 10/328, 11/347, 14/406, 15/436}
20	12	{0/20, 1/50, 2/72, 3/118, 4/164, 6/222, 8/288, 9/307, 10/329, 11/348, 14/407, 15/437}
21	14	{0/21, 2/73, 3/119, 4/165, 5/185, 6/223, 7/264, 8/289, 9/308, 10/330, 11/349, 14/408, 15/438, 16/463}
22	14	{0/22, 2/74, 3/120, 4/166, 5/186, 6/224, 7/265, 8/290, 9/309, 10/331, 11/350, 14/409, 15/439, 16/464}
23	14	{0/23, 2/75, 3/121, 4/167, 5/187, 6/225, 7/266, 8/291, 9/310, 10/332, 11/351, 14/410, 15/440, 16/465}
24	14	{0/24, 2/76, 3/122, 4/168, 5/188, 6/226, 7/267, 8/292, 9/311, 10/333, 11/352, 14/411, 15/441, 16/466}

Tabela 36: Lista de processador por tarefa – Continuação

Tipo de tarefa	#Processadores	*ProcID/IP ID*
25	14	{0/25, 2/77, 3/123, 4/169, 5/189, 6/227, 7/268, 8/293, 9/312, 10/334, 11/353, 14/412, 15/442, 16/467}
26	14	{0/26, 2/78, 3/124, 4/170, 5/190, 6/228, 7/269, 8/294, 9/313, 10/335, 11/354, 14/413, 15/443, 16/468}
27	12	{0/27, 2/79, 3/125, 4/171, 5/191, 6/229, 7/270, 9/314, 11/355, 14/414, 15/444, 16/469}
28	12	{0/28, 2/80, 3/126, 4/172, 5/192, 6/230, 7/271, 9/315, 11/356, 14/415, 15/445, 16/470}
29	12	{0/29, 2/81, 3/127, 4/173, 5/193, 6/231, 7/272, 9/316, 11/357, 14/416, 15/446, 16/471}
30	14	{0/30, 2/82, 3/128, 4/174, 5/194, 6/232, 7/273, 8/295, 9/317, 10/336, 11/358, 14/417, 15/447, 16/472}
31	14	{0/31, 2/83, 3/129, 4/175, 5/195, 6/233, 7/274, 8/296, 9/318, 10/337, 11/359, 14/418, 15/448, 16/473}
32	14	{0/32, 2/84, 3/130, 4/176, 5/196, 6/234, 7/275, 8/297, 9/319, 10/338, 11/360, 14/419, 15/449, 16/474}
33	14	{0/33, 2/85, 3/131, 4/177, 5/197, 6/235, 7/276, 8/298, 9/320, 10/339, 11/361, 14/420, 15/450, 16/475}
34	14	{0/34, 2/86, 3/132, 4/178, 5/198, 6/236, 7/277, 8/299, 9/321, 10/340, 11/362, 14/421, 15/451, 16/476}
35	14	{0/35, 2/87, 3/133, 4/179, 5/199, 6/237, 7/278, 8/300, 9/322, 10/341, 11/363, 14/422, 15/452, 16/477}
36	14	{0/36, 2/88, 3/134, 4/180, 5/200, 6/238, 7/279, 8/301, 9/323, 10/342, 11/364, 14/423, 15/453, 16/478}
37	6	{0/37, 2/89, 3/135, 6/239, 14/424, 15/454}
38	6	{0/38, 2/90, 3/136, 6/240, 14/425, 15/455}
39	6	{0/39, 2/91, 3/137, 6/241, 14/426, 15/456}
40	6	{0/40, 2/92, 3/138, 6/242, 14/427, 15/457}
41	6	{0/41, 2/93, 3/139, 6/243, 14/428, 15/458}
42	9	{0/42, 2/94, 3/140, 4/181, 6/244, 7/280, 12/366, 14/429, 15/459}
43	9	{0/43, 2/95, 3/141, 4/182, 6/245, 7/281, 12/367, 14/430, 15/460}
44	9	{0/44, 2/96, 3/142, 4/183, 6/246, 7/282, 12/368, 14/431, 15/461}
45	17	{0/45, 1/51, 2/97, 3/143, 4/184, 5/201, 6/247, 7/283, 8/302, 9/324, 10/343, 11/365, 12/369, 13/386, 14/432, 15/462, 16/479}

Tabela 37: Características de IPs para uso no exemplo ilustrativo da Figura 37

Tipo das tarefas	Identificador dos processadores							
	4				10			
	Tempo[1]	Área[2]	Consumo[3]	Custo[4]	Tempo	Área	Consumo	Custo
17	0,46	70,06	16,00	125,60	1,00	2,99	1,70	24,00
20	0,44	70,06	16,00	125,60	1,20	2,99	1,70	24,00
22	0,27	70,06	16,00	125,60	0,98	2,99	1,70	24,00
30	0,10	70,06	16,00	125,60	0,29	2,99	1,70	24,00
42	3,50	70,06	16,00	125,60	-	-	-	-
45	0,01	70,06	16,00	125,60	0,01	2,99	1,70	24,00

[1]($\times 10^{-3}$s), [2]($\times 10^{-6}$m^2), [3](W), [4](\$US)

Obs.: As unidades para o processador 10 são as mesmas.

Tabela 38: Características de IPs para uso no exemplo ilustrativo da Figura 37 (Continuação)

Tipo das tarefas	Identificador dos processadores							
	11				13			
	Tempo[1]	Área[2]	Consumo[3]	Custo[4]	Tempo	Área	Consumo	Custo
17	38,44	0,77	52,10	0,23	-	-	-	-
20	38,44	0,77	52,10	0,14	-	-	-	-
22	38,44	0,77	52,10	0,29	-	-	-	-
30	38,44	0,77	52,10	0,17	-	-	-	-
42	-	-	-	-	-	-	-	-
45	38,44	0,77	52,10	0,01	0,01	1,00	1,00	45,00

[1]($\times 10^{-3}$s), [2]($\times 10^{-6}$m^2), [3](W), [4](\$US)

Obs.: As unidades para o processador 13 são as mesmas.

Tabela 39: Características de IPs para uso no exemplo ilustrativo da Figura 37 (Continuação)

Tipo das tarefas	Identificador dos processadores							
	14				16			
	Tempo[1]	Área[2]	Consumo[3]	Custo[4]	Tempo	Área	Consumo	Custo
17	4,00	19,18	2,50	30,00	-	-	-	-
20	4,20	19,18	2,50	30,00	-	-	-	-
22	1,30	19,18	2,50	30,00	0,04	1,00	1,60	111,20
30	0,58	19,18	2,50	30,00	0,01	1,00	1,60	111,20
42	31,00	19,18	2,50	30,00	-	-	-	-
45	0,01	19,18	2,50	30,00	0,01	1,00	1,60	111,20

[1]($\times 10^{-3}$s), [2]($\times 10^{-6}$m^2), [3](W), [4](\$US)

Obs.: As unidades para o processador 16 são as mesmas.

APÊNDICE B – Aplicações de Referência

Este apêndice lista as aplicações usadas como *benchmarks* para avaliar o mérito do trabalho descrito nesta dissertação. São 5 conjuntos de aplicações de domínios diferentes. Cada aplicação é apresentada através uma descrição geral seguida pelo grafo de tarefas correspondente. Os dados forma aproveitados da página localizada em http://www.eembc.org/.

B.1 Aplicações da Indústria Automotiva

São cinco aplicações utilizadas na indústria automobilística, que geralmente são implementadas em sistemas embarcados. Estas aplicações utilizam microprocessadores e microcontroladores para realizar as funcionalidades requeridas.

B.1.1 Aplicação: *auto-indust-tg0*

Essa aplicação permite a realizar testes de cargas. Esses testes incluem uma manipulação de *bits*, manipulação de matrizes, operações de ponto flutuante, estouro de cache, PWM (*Pulse width Modulation*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="0" period="0.0009" name="auto-indust-mocsyn0" tasks="6">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="0" taskName="can1" root="false" level="1" />
  <node id="2" type="1" taskName="fp" root="false" level="2" />
  <node id="3" type="0" taskName="can2" root="false" level="3" />
  <node id="4" type="12" taskName="pulse" root="false" level="4" />
  <node id="5" type="45" taskName="sync" root="false" level="5" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="4E3" />
  <edge id="1" type="0" source="1" target="2" averageBits="4E3" />
  <edge id="2" type="0" source="2" target="3" averageBits="4E3" />
  <edge id="3" type="0" source="3" target="4" averageBits="4E3" />
  <edge id="4" type="1" source="4" target="5" averageBits="8E3" />
 </edges>
</taskgraph>
```

B.1.2 Aplicação: *auto-indust-tg1*

Essa aplicação especifica algoritmos automotivos básicos, tais como algoritmo de sincronização do motor no momento da partida (*tooth-to-spark*), conversão de ângulo para hora, cálculo de velocidade e interpolação.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="1" period="0.00045" name="auto-indust-mocsyn1" tasks="4">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="7" taskName="iir" root="false" level="1" />
  <node id="2" type="8" taskName="idct" root="false" level="2" />
  <node id="3" type="45" taskName="sink" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="4E3" />
  <edge id="1" type="0" source="1" target="2" averageBits="4E3" />
  <edge id="2" type="0" source="2" target="3" averageBits="4E3" />
 </edges>
</taskgraph>
```

B.1.3 Aplicações: *auto-indust-tg2* e *auto-indust-tg3*

Essas aplicações especificam algoritmos de processamento de sinais e algoritmos usados para o cálculo de estabilidade do veículo a partir dos sinais provenientes de sensores. Incluem FFT (*Fast Fourier Transform*), IFFT (*Inverse fast Fourier transform*), FIR (*Finite Impulse Response Filter*) e IDCT (*Inverse Discrete Cosine Transform*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="2" period="0.0009" name="auto-indust-mocsyn2" tasks="9">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="5" taskName="fft" root="false" level="1" />
  <node id="2" type="10" taskName="matrix" root="false" level="2" />
  <node id="3" type="9" taskName="ifft" root="false" level="3" />
  <node id="4" type="6" taskName="fir" root="false" level="1" />
  <node id="5" type="0" taskName="angle" root="false" level="4" />
  <node id="6" type="13" taskName="road" root="false" level="5" />
  <node id="7" type="14" taskName="table" root="false" level="6" />
  <node id="8" type="45" taskName="sink" root="false" level="7" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="4" averageBits="4E3" />
  <edge id="1" type="0" source="4" target="5" averageBits="4E3" />
  <edge id="2" type="2" source="0" target="1" averageBits="15E3" />
  <edge id="3" type="2" source="1" target="2" averageBits="15E3" />
  <edge id="4" type="2" source="2" target="3" averageBits="15E3" />
  <edge id="5" type="2" source="3" target="5" averageBits="15E3" />
  <edge id="6" type="0" source="5" target="6" averageBits="4E3" />
  <edge id="7" type="0" source="6" target="7" averageBits="4E3" />
  <edge id="8" type="3" source="7" target="8" averageBits="1E3" />
 </edges>
</taskgraph>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="3" period="0.0009" name="auto-indust-mocsyn3" tasks="5">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="11" taskName="ptr" root="false" level="1" />
  <node id="2" type="3" taskName="cache" root="false" level="2" />
  <node id="3" type="15" taskName="tooth" root="false" level="3" />
  <node id="4" type="45" taskName="sync" root="false" level="4" />
 </nodes>
 <edges>
  <edge id="0" type="3" source="0" target="1" averageBits="1E3" />
  <edge id="1" type="1" source="1" target="2" averageBits="8E3" />
  <edge id="2" type="1" source="2" target="3" averageBits="8E3" />
  <edge id="3" type="3" source="3" target="4" averageBits="1E3" />
 </edges>
</taskgraph>
```

B.2 Aplicações de consumidor

São duas aplicações utilizadas em câmeras fotográficas digitais, impressoras e outros sistemas embutidos que manipulam imagens digitais.

B.2.1 Aplicação: *consumer-tg0*

Essa aplicação é dedicada à algoritmos usados para compressão e descompressão de imagem, tais como algoritmo JPEG de compressão e descompressão de imagens.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <taskgraph id="0" period="0.06" name="consumer-mocsyn" tasks="7">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="39" taskName="filt-r" root="false" level="1" />
  <node id="2" type="39" taskName="filt-g" root="false" level="1" />
  <node id="3" type="39" taskName="filt-b" root="false" level="1" />
  <node id="4" type="41" taskName="rgb-yiq" root="false" level="2" />
  <node id="5" type="37" taskName="cjpeg" root="false" level="3" />
  <node id="6" type="45" taskName="sync" root="false" level="4" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="2E6" />
  <edge id="1" type="0" source="0" target="2" averageBits="2E6" />
  <edge id="2" type="0" source="0" target="3" averageBits="2E6" />
  <edge id="3" type="0" source="1" target="4" averageBits="2E6" />
  <edge id="4" type="0" source="2" target="4" averageBits="2E6" />
  <edge id="5" type="0" source="3" target="4" averageBits="2E6" />
  <edge id="6" type="1" source="4" target="5" averageBits="6E6" />
  <edge id="7" type="2" source="5" target="6" averageBits="1E6" />
 </edges>
</taskgraph>
```

B.2.2 Aplicação: *consumer-tg1*

Essa aplicação é dedicada à algoritmos filtragem de cores e conversões, tais como filtro passa-alta em tons de cinza e conversão RGB-to-CMYK e conversão RGB-to-YIQ.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <taskgraph id="1" period="0.015" name="consumer-mocsyn" tasks="5">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="38" taskName="djpeg" root="false" level="1" />
  <node id="2" type="45" taskName="display" root="false" level="2" />
  <node id="3" type="40" taskName="rgb-cymk" root="false" level="2" />
  <node id="4" type="45" taskName="print" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="2" source="0" target="1" averageBits="1E6" />
  <edge id="1" type="1" source="1" target="2" averageBits="6E6" />
  <edge id="2" type="1" source="1" target="3" averageBits="6E6" />
  <edge id="3" type="1" source="3" target="4" averageBits="6E6" />
 </edges>
</taskgraph>
```

B.3 Aplicações de rede

São três aplicações que permitem simular o transporte de pacotes em uma rede de comunicações.

B.3.1 Aplicação: *networking-tg1*

Essa aplicação é usada para a verificação de pacote IP através do processamento do cabeçalho de pacotes IP durante o envio e o recebimento por roteadores.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="1" period="0.00135" name="networking-mocsyn" tasks="4">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="17" taskName="patricia" root="false" level="1" />
  <node id="2" type="18" taskName="pf512" root="false" level="2" />
  <node id="3" type="45" taskName="sink" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="4194304" />
  <edge id="1" type="0" source="1" target="2" averageBits="4194304" />
  <edge id="2" type="0" source="2" target="3" averageBits="4194304" />
 </edges>
</taskgraph>
```

B.3.2 Aplicação: *networking-tg2*

Essa aplicação modela a Tradução de Endereço de Rede IP (*NAT-Network Address Translator*), que consiste na re-escrita do endereço IP e da porta configurada nos pacotes de acordo com a configuração da tabela NAT do roteador.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<taskgraph id="2" period="0.00135" name="networking-mocsyn" tasks="4">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="17" taskName="patricia" root="false" level="1" />
  <node id="2" type="19" taskName="pf1m" root="false" level="2" />
  <node id="3" type="45" taskName="sink" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="1" source="0" target="1" averageBits="8388608" />
  <edge id="1" type="1" source="1" target="2" averageBits="8388608" />
  <edge id="2" type="1" source="2" target="3" averageBits="8388608" />
 </edges>
</taskgraph>
```

B.3.3 Aplicação: *networking-tg3*

Essa aplicação é usada para busca de rotas. Realiza o processamento de uma estrutura de dados conhecida como árvore de Patrícia, uma árvore binária compacta que permite uma busca rápida e eficiente em longas cadeias de *bits*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="3" period="0.00135" name="networking-mocsyn" tasks="4">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="17" taskName="patricia" root="false" level="1" />
  <node id="2" type="20" taskName="pf2m" root="false" level="2" />
  <node id="3" type="45" taskName="sink" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="2" source="0" target="1" averageBits="16777216" />
  <edge id="1" type="2" source="1" target="2" averageBits="16777216" />
  <edge id="2" type="2" source="2" target="3" averageBits="16777216" />
 </edges>
</taskgraph>
```

B.4 Aplicação de automação

Aplicações executadas em impressoras, plotters e outros sistemas de automação de escritório que incluem tarefas de processamento de imagem e texto.

B.4.1 Aplicação: *office-automation-tg0*

Essa aplicação implementa o processamento da curva de Bezier através da interpolação de um conjunto de pontos.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="0" period="0.03" name="office-automation-mocsyn" tasks="5">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="44" taskName="text" root="false" level="1" />
  <node id="2" type="45" taskName="sink" root="false" level="3" />
  <node id="3" type="43" taskName="rotate" root="false" level="1" />
  <node id="4" type="42" taskName="dith" root="false" level="2" />
```

```
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="1E3" />
  <edge id="1" type="1" source="0" target="3" averageBits="787E3" />
  <edge id="2" type="1" source="3" target="4" averageBits="787E3" />
  <edge id="3" type="1" source="4" target="2" averageBits="787E3" />
  <edge id="4" type="0" source="1" target="2" averageBits="1E3" />
 </edges>
</taskgraph>
```

B.5 Aplicações de telecomunicação

São seis aplicações executadas por modems e outros dispositivos embutidos utilizados na área de telecomunicação.

B.5.1 Aplicação: *telecom-tg0*

Essa aplicação especifica a autocorrelação e executa a relação cruzada de um sinal com ele próprio. Este mecanismo é utilizado em telecomunicação para análise de sinais.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="0" period="0.001" name="telecom-mocsyn" tasks="4">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="21" taskName="ac1" root="false" level="1" />
  <node id="2" type="24" taskName="ce1" root="false" level="2" />
  <node id="3" type="45" taskName="sink" root="false" level="3" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="10E3" />
  <edge id="1" type="1" source="1" target="3" averageBits="3E3" />
  <edge id="2" type="1" source="1" target="2" averageBits="3E3" />
  <edge id="3" type="2" source="2" target="3" averageBits="4E3" />
 </edges>
</taskgraph>
```

B.5.2 Aplicação: *telecom-tg1*

Essa aplicação implementa um codificador de convolução e permite a correção de erro. É utilizada para melhorar a qualidade de rádios digitais, telefones celulares, comunicações de satélite e conexões *bluetooth*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="1" period="0.001" name="telecom-mocsyn" tasks="6">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="21" taskName="ac2" root="false" level="1" />
  <node id="2" type="24" taskName="ce2" root="false" level="2" />
  <node id="3" type="30" taskName="fft2" root="false" level="3" />
  <node id="4" type="27" taskName="fpba2" root="false" level="2" />
  <node id="5" type="45" taskName="sink" root="false" level="4" />
 </nodes>
 <edges>
```

```
  <edge id="0" type="0" source="0" target="1" averageBits="10E3" />
  <edge id="1" type="1" source="1" target="4" averageBits="3E3" />
  <edge id="2" type="1" source="1" target="2" averageBits="3E3" />
  <edge id="3" type="1" source="4" target="3" averageBits="3E3" />
  <edge id="4" type="1" source="3" target="5" averageBits="3E3" />
  <edge id="5" type="2" source="2" target="5" averageBits="4E3" />
 </edges>
</taskgraph>
```

B.5.3 Aplicação: *telecom-tg2*

Essa aplicação permite a verificação de capacidade de transmissão de dados em linhas ADSL.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="2" period="0.001" name="telecom-mocsyn" tasks="6">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="21" taskName="ac3" root="false" level="1" />
  <node id="2" type="24" taskName="ce3" root="false" level="2" />
  <node id="3" type="30" taskName="fft3" root="false" level="3" />
  <node id="4" type="27" taskName="fpba3" root="false" level="2" />
  <node id="5" type="45" taskName="sink" root="false" level="4" />
 </nodes>
 <edges>
  <edge id="0" type="0" source="0" target="1" averageBits="10E3" />
  <edge id="1" type="1" source="1" target="4" averageBits="3E3" />
  <edge id="2" type="1" source="1" target="2" averageBits="3E3" />
  <edge id="3" type="1" source="4" target="3" averageBits="3E3" />
  <edge id="4" type="1" source="3" target="5" averageBits="3E3" />
  <edge id="5" type="2" source="2" target="5" averageBits="4E3" />
  </edges>
</taskgraph>
```

B.5.4 Aplicação: *telecom-tg3*

Essa aplicação permite a conversão de dados no domínio do tempo para o domínio da frequência, usando FFT (*Fast Fourier Transform*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="3" period="0.001" name="telecom-mocsyn" tasks="3">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="30" taskName="fft1" root="false" level="1" />
  <node id="2" type="45" taskName="sink" root="false" level="2" />
 </nodes>
 <edges>
  <edge id="0" type="1" source="0" target="1" averageBits="3E3" />
  <edge id="1" type="1" source="1" target="2" averageBits="3E3" />
 </edges>
</taskgraph>
```

B.5.5 Aplicação: *telecom-tg4*

Essa aplicação permite a conversão de dados no domínio da frequência para o domínio do tempo, usando IFFT (*Inverse Fast Fourier Transform*).

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="4" period="0.001" name="telecom-mocsyn" tasks="3">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="29" taskName="fpba" root="false" level="1" />
  <node id="2" type="45" taskName="sink" root="false" level="2" />
 </nodes>
 <edges>
  <edge id="0" type="1" source="0" target="1" averageBits="3E3" />
  <edge id="1" type="1" source="1" target="2" averageBits="3E3" />
 </edges>
</taskgraph>
```

B.5.6 Aplicação: *telecom-tg5*

Essa aplicação implementa um codificador *Viterbi*, usado em decodificação de dados codificados em TDMA.

```
<?xml version="1.0" encoding="UTF-8" ?>
<taskgraph id="5" period="0.000333333" name="telecom-mocsyn" tasks="2">
 <nodes>
  <node id="0" type="45" taskName="src" root="true" level="0" />
  <node id="1" type="35" taskName="gsm1" root="false" level="1" />
 </nodes>
 <edges>
  <edge id="0" type="3" source="0" target="1" averageBits="1E3" />
 </edges>
</taskgraph>
```