PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Algoritmos de Particionamento para MPSoCs Heterogêneos Baseados em NoC

IGOR KRAMER PINOTTI

Dissertação apresentada como requisito parcial a obtenção do grau de Mestre em Ciência da computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. César Augusto Missio Marcon

Coorientadora: Dra. Thais Christina Webber dos Santos

Porto Alegre

2013

Dados Internacionais de Catalogação na Publicação (CIP)

P657a Pinotti, Igor Kramer
Algoritmos de particionamento para MPSoCs heterogêneos baseados em NoC / Igor Kramer Pinotti. – Porto Alegre, 2013.
80 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César Augusto Missio Marcon.
Co-orientador: Dra. Thais Christina Webber dos Santos.

1. Informática. 2. Algoritmos. 3. Multiprocessadores. I. Marcon, César Augusto Missio. II. Santos, Thais Christina Webber dos. III. Título.

CDD 004.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Algoritmos de Particionamento para MPSoCs Heterogêneos Baseados em NoC" apresentada por Igor Kramer Pinotti como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 22/03/2013 pela Comissão Examinadora:

Prof. Dr. César Augusto Missio Marcon – Orientador — PPGCC/PUCRS

Dra. Thais Christina Webber dos Santos – Coorientadora — PPGEE/PUCRS

Prof. Dr. Alexandre de Morais Amory – PPGCC/PUCRS

Prof. Dr. Edson Ifarraguirre Moreno – FACIN/PUCRS

Prof. Dr. Fabiano Passuelo Hessel – PPGCC/PUCRS

Homologada em 28./05./2013, conforme Ata No. 009 pela Comissão Coordenadora.

p/ Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Dedico este trabalho a minha família, que me apoia em todos os caminhos que tenho escolhido.

“Our greatest glory is not in never falling,

But in rising every time we fall.”

- Confucius.

AGRADECIMENTOS

Gostaria de agradecer ao GSE (Grupo de Sistemas Embarcados), ao GAPH (Grupo de Apoio ao Projeto de Hardware), e todos os professores do PPGCC, em especial ao Prof. Dr. César Augusto Missio Marcon, por me conceder esta grande oportunidade de realizar o mestrado, me ensinar os caminhos de um estudo dirigido, e tornar este trabalho possível. Gostaria de agradecer a Dra. Thais Webber, por esclarecer dilemas e propor ideias adotadas neste projeto, e ao bolsista e estudante de Engenharia de Computação, Natanael Ribeiro, pela grande ajuda e trabalho árduo na implementação das ideias que moveram este trabalho. Vocês foram essenciais para a concepção deste trabalho. Muito obrigado a todos.

ALGORITMOS DE PARTICIONAMENTO PARA MPSOCS HETEROGÊNEOS BASEADOS EM NOC

RESUMO

Várias aplicações novas são compostas por uma heterogeneidade de tarefas implicando alto grau de complexidade, e requerendo grande capacidade de processamento e comunicação eficiente. *Multiprocessor System-on-Chip* (MPSoC) baseado em *Network-on-Chip* (NoC) é uma arquitetura alvo promissora com capacidade de atender diversos requisitos de uma aplicação alvo, devido à alta capacidade de computação e grande paralelismo de comunicação que possibilitam a execução de diversas tarefas simultaneamente. Além disso, requisitos de diversas aplicações embarcadas são melhores atendidos por um MPSoC composto por vários tipos de processadores – MPSoC heterogêneo. Um desafio atual no projeto de MPSoC heterogêneo é particionar tarefas, almejando minimizar consumo de energia e ter balanceamento de carga apropriado. Este trabalho contribui duplamente em: (i) análise e comparação de algoritmos de particionamento; e (ii) avaliação do particionamento como uma atividade pré-mapeamento. Este trabalho analisa e compara algoritmos de particionamentos estocásticos e heurísticos, elaborados para obter baixo consumo de energia e balanceamento de carga eficiente quando aplicados a particionamento de tarefas em um MPSoC heterogêneo. Além disto, resultados de desempenho, obtidos através de simulações, indicam que a técnica de particionamento estático de tarefas pode ser previamente aplicada à atividade de mapeamento de grupos de tarefas em processadores da arquitetura alvo, aprimorando a qualidade do mapeamento estático ou dinâmico, e ainda, minimizando o tempo de processamento.

Palavras Chave: Algoritmos de Particionamento, Algoritmos de Mapeamento, MPSoC Heterogêneos, Avaliação de Performance, Simulação

PARTITIONING ALGORITHMS FOR HETEROGENEOUS NOC-BASED MPSOC

ABSTRACT

Several new applications are composed by heterogeneity of tasks implying high complexity degree, and requiring high processing and communicating rate. Multiprocessor System-on-Chip (MPSoC) based on Network-on-Chip (NoC) is a promising targeting architecture to fulfill these requirements, due to its high computation and communication parallelism that enables several tasks executed at the same time. Furthermore, these applications requirements are better fulfilled by MPSoC composed by different types of processors – heterogeneous MPSoC. One challenge in current heterogeneous MPSoC design is partitioning of application tasks, aiming energy consumption minimization and fair load balance. This work contribution is twofold: (i) analysis and comparison of partitioning algorithms; and (ii) the evaluation of partitioning as a pre-mapping task. This work analyzes and compares stochastic and new heuristic partitioning algorithms for obtaining low energy consumption and efficient load balance when applied to tasks partitioning onto heterogeneous MPSoC. In addition, performance results obtained from simulations indicate that the static partitioning technique can be used on application tasks before mapping activities to improve the quality on the static or dynamic mapping and also for minimizing processing time.

Keywords: Partitioning Algorithms, Mapping Algorithms, Heterogeneous MPSoC, Performance Evaluation, Simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Sistemas intrachip, do inglês, *Systems-on-Chip* (SoCs) são aqueles onde a completa funcionalidade de um *sistema* é implementada em um único Circuito Integrado (CI) [1]. Estes sistemas geralmente consomem menos energia, possuem melhor desempenho e alta capacidade de processamento, além de apresentarem alta confiabilidade se comparados a abordagens clássicas de arquiteturas implementadas com vários CIs. A necessidade crescente de desempenho de sistemas eletrônicos é um dos fatores que impulsiona a evolução de tecnologias intrachip e torna possível integrar bilhões de transistores em um único *chip*, gerando SoCs de maior desempenho.

MPSoCs (do inglês, *Multiprocessor Systems-on-Chip*) são sistemas compostos por múltiplos processadores implementados na forma de um SoC [2]. MPSoC emergiu na década passada com uma classe muito importante de VLSI (do inglês, *Very large-scale Integration* - o processo de integrar centenas de milhares de componentes em um único *chip*). Um MPSoC é um *SoC* que incorpora a maioria dos componentes necessários para uma aplicação, utilizando múltiplos processadores de várias naturezas, como componentes de sistema, ligados por uma arquitetura de comunicação, como uma NoC [10]. Estes componentes podem ser processadores de propósitos gerais (GPP – *General Purpose Processors*) (e.g. ARM, PowerPC), DSPs (*Digital Signal Processor)*, FPGAs (*Field Programmable Gate Arrays*), ASICs (*Application Specific Integrated Circuits*), módulos específicos de *hardware* reconfigurável (DSRH – Domain Specific Reconfigurable Hardware) entre outros.

MPSoCs formam dois importantes e distintos ramos na taxonomia de multiprocessadores: homogêneos e heterogêneos. Os MPSoCs homogêneos são compostos por processadores de uma mesma natureza, enquanto que os MPSoCs heterogêneos são compostos por processadores de diversas naturezas. Os ramos dos MPSoCs, homogêneo e heterogêneo, são voltados para aplicações distintas. A arquitetura homogênea é geralmente utilizada para sistemas de dados paralelos. Estações *wireless*, onde o mesmo algoritmo é aplicado a vários conjuntos de dados independentes, é um exemplo; estimativa de movimento, em que várias partes de uma imagem possam ser tratadas separadamente, é outro exemplo [8]. Arquiteturas heterogêneas são projetadas para aplicações heterogêneas com diagrama de blocos complexos que incorporam múltiplos algoritmos, geralmente utilizando o paradigma de transferência de dados produtor-consumidor. Um sistema compressão completa de vídeo é um exemplo específico para aplicações heterogêneas [8].

Enquanto MPSoCs homogêneos tendem a simplificar a aplicação de técnicas como migração de tarefas e ser mais fácil de programar, MPSoCs heterogêneos podem dar suporte a uma variedade maior de aplicações, possibilitando implementar uma arquitetura direcionada para, por exemplo, minimizar o consumo de energia e potência [3].

Para garantir qualidade e desempenho, um decodificador de TV digital, por exemplo, deve ser heterogêneo o suficiente para integrar processadores, núcleos de hardware dedicados e memórias. Além disso, cada um destes componentes possui diferentes funcionalidades, tamanhos e necessidades de comunicação, o que demonstra a complexidade de sistemas heterogêneos.

Grande parte dos sistemas embarcados modernos (e.g. *smartphones*, *tablets*, HDTVs, *set-top box*) já utilizam um sistema multiprocessado em único chip [4]. Devido à grande variedade de aplicações com diversos requisitos e restrições de naturezas diversas, muitos destes projetos de MPSoC podem ser melhor implementados com arquiteturas heterogêneas, podendo ter vários elementos de processamento, memórias e arquiteturas de comunicação com suas próprias características.

Muitas publicações recentes têm mostrado um aumento significativo no desenvolvimento de arquiteturas MPSoCs devido às suas inúmeras vantagens, como eficiência energética, melhoria no desempenho do sistema, e até mesmo a exploração do espaço de projeto proporcionada pela migração de tarefas entre elementos de processamento [3].

As redes intrachip (do inglês, *Networks-on-Chip* ou simplesmente NoCs) trazem uma abordagem de redes de comunicação para a comunicação interna do *chip*, baseados na adaptação de conceitos bem conhecidos de redes de computadores, sistemas distribuídos e telecomunicação.

A solução baseada em infraestruturas de comunicação do tipo NoC possui boa escalabilidade, possibilitando a conexão de diversos elementos, e capacidade de dar suporte a diversas comunicações simultâneas, de escritas e leituras, tornando-as adequadas para tratar aplicações que requerem comunicação intensiva e com grande quantidade de elementos de processamento [5]. Por outro lado, soluções baseadas em barramentos, em geral, são adequadas à comunicação em sistemas menores, com poucos elementos, além de ter um sistema de comunicação que possibilita apenas um único elemento escritor por vez.

Seja *tile* uma região limitada da arquitetura alvo com elemento de processamento e mais hardware dedicado à comunicação com os demais *tiles*, uma NoC é dita heterogênea quando for composta por *tiles* irregulares (i.e. com tamanhos e formatos distintos), ou roteadores irregulares (i.e. com características diferentes, tais como número de conexões e tamanho do *buffer*). Uma NoC homogênea difere da anterior, pois *tiles* e roteadores possuem áreas, formatos e características iguais. A heterogeneidade pode também ser especificada quanto ao tipo de elemento de processamento ou com relação ao tamanho do mesmo, ou seja, sua granularidade. Normalmente, elementos de processamento diferentes ocupam áreas diferentes. Porém, NoCs com *tiles* regulares

podem conter diferentes processadores desde que estes *tiles* sejam dimensionados para o tamanho do maior processador [6]. A Figura 1 (a) apresenta formas geométricas de *tile* heterogêneo, enquanto que a Figura 1 (b) compara estes com formas geométricas homogêneas [6].

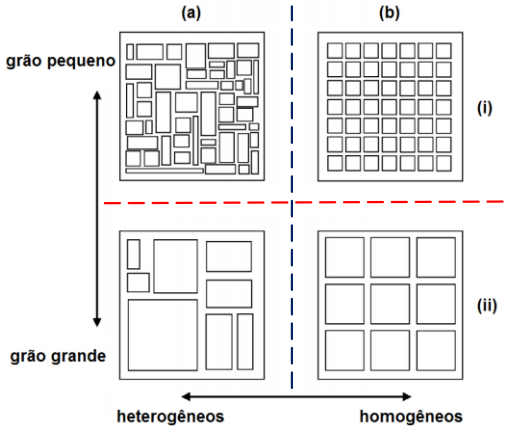


Figura 1. Representações de geometria e granularidade. (a) apresenta geometria irregular – heterogênea e (b) geometria regular – homogênea. Enquanto que (i) e (ii) mostram a relação da granularidade, sendo (i) grão pequeno e (ii) grão grande [6].

No caso de aplicações que demandem comunicação intensiva, tais como aplicações que envolvem transmissão e processamento de áudio e vídeo, o gerenciamento da comunicação na arquitetura alvo é de extrema importância, uma vez que em muitos casos a ocorrência de congestionamentos pode acarretar problemas na transmissão e no processamento dos dados, degradando o desempenho da aplicação.

A especificação de uma aplicação pode ser realizada em diversos níveis, considerando, por exemplo, a interação entre elementos de processamento ou tarefas de um sistema. Este trabalho parte de aplicações descritas como um conjunto de tarefas, considerando que estas podem se comunicar, sendo executadas em um mesmo processador ou em processadores distintos (de naturezas diversas), localizados em uma NoC com *tiles* de tamanhos regulares.

As atividades abordadas neste trabalho se resumem no agrupamento das tarefas de uma aplicação em processadores (**Particionamento**), por meio de algorítmicos heurísticos ou estocásticos, seguido da associação de grupo de tarefas em *tiles* de um MPSoC (**Mapeamento**), objetivando mostrar a eficiência da utilização da técnica de particionamento como pré-mapeamento. Para estas atividades são levados em consideração requisitos como minimizar o consumo de energia, balancear a carga dos processadores, reduzir o tempo de processamento e reduzir o uso de memória.

1.1 Arquitetura de Comunicação Intrachip

A topologia de uma rede intrachip é definida pelo arranjo de interconexões entre seus elementos de roteamento. Topologias físicas e lógicas costumam ser modeladas como grafos onde os elementos de roteamento são representados por vértices e as interconexões são representadas por arestas. As topologias podem ser regulares, quando é possível definir um padrão para este arranjo, com base na estrutura dos elementos de roteamento. Caso este não possa ser identificado, ou possua um modelo não ortogonal, a topologia é denominada irregular [7]. Esta diferença pode ser observada na Figura 2.

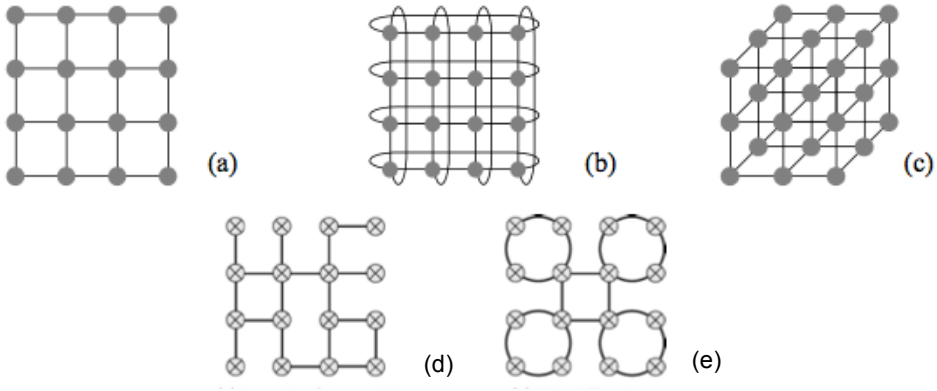


Figura 2. Exemplo de topologias de rede intrachip: Regulares - (a) malha 2D , (b) toro 2D e (c) hipercubo 3D; Irregulares – (d) e (e).

Este trabalho utiliza NoC malha 2D como arquitetura de comunicação, que é a arquitetura regular ilustrada na Figura 3. Os retângulos contendo as letras A, B, C e D são processadores de diferentes tipos (e.g. ARM, MIPS, PowerPC, PentiumV). R_1, R_2, R_3 e R_4 representam roteadores. As flechas entre os roteadores são conexões bidirecionais.

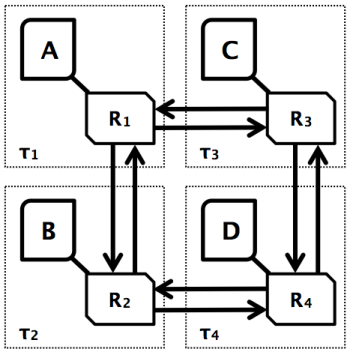


Figura 3. Exemplo de NoC malha 2D.

Uma NoC malha 2D pode ser representada por um grafo de recursos de comunicação CRG (*Communication Resource Graph*). Este é um grafo dirigido <Γ, L>, onde o conjunto de vértices é o conjunto de *tiles* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, e o conjunto de arestas L = $\{(\tau_i, \tau_j), \forall\ \tau_i, \tau_j \in \Gamma\}$, representa o conjunto de caminhos de τ_i para τ_j, sendo τ_i e τ_j *tiles* adjacentes. Este modelo é mais bem explorado no Capítulo 3.

1.2 Definição da Atividade de Particionamento e Mapeamento

Esta Seção define a atividade de particionamento e mapeamento dentro de um fluxo de projeto de MPSoCs.

Aplicações podem ser descritas como um conjunto de tarefas e suas inter-relações. A partir desta definição, no contexto deste trabalho, particionamento é definido como o ato de agrupar tarefas conforme regras (requisitos e restrições de projeto). Já o mapeamento é definido como a associação de cada grupo de tarefas a um núcleo específico (processador) da arquitetura alvo.

Neste trabalho, as regras de particionamento de tarefas em arquiteturas heterogêneas, diferente das regras aplicadas a arquiteturas homogêneas, levam em consideração o desempenho que cada tarefa terá em processadores de naturezas diferentes. Certas tarefas são concebidas para terem melhor desempenho em certo tipo de arquitetura, fazendo-se necessário atribuir regras específicas de natureza de arquitetura.

A etapa de particionamento é ilustrada na Figura 4, através das setas que partem do nível de descrição das tarefas de uma aplicação para o nível de descrição de grupos selecionados. O nível de tarefas é composto por um conjunto de tarefas $T = \{t_1, t_2, ..., t_n\}$, onde cada tarefa é representada por um hexágono. Sobre o nível de tarefas é realizado a atividade de particionamento, gerando um conjunto de grupos de tarefas $G = \{g_a, g_b, ..., g_m\}$.

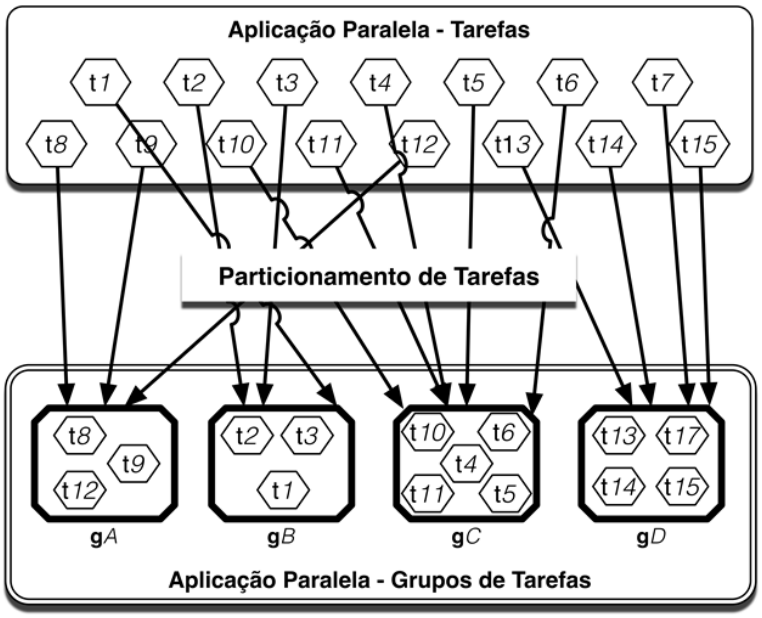


Figura 4. Particionamento de tarefas de uma aplicação paralela.

A atividade de particionamento é realizada levando em consideração uma função de custo de particionamento, formado por conjunto estipulado de regras de particionamento. Este custo é planejado de forma a atender requisitos e restrições do

projeto em questão. Esta função é utilizada para agrupar tarefas de forma a minimizar a quantidade de comunicação entre processadores e atender as necessidades e peculiaridades de cada tarefa, sob o ponto de vista heterogêneo, por exemplo, quando uma tarefa somente pode ser executada em certo tipo de arquitetura.

A infraestrutura de comunicação utilizada neste trabalho é uma NoC com topologia do tipo malha 2D. Em topologia 2D, o particionamento afeta o tempo da comunicação e consequentemente o tempo total de execução da aplicação, que está diretamente ligado ao consumo de energia utilizado pela arquitetura alvo. Sendo assim, uma vez que as tarefas foram agrupadas, e estes grupos, associados a processadores específicos, estes grupos devem ser mapeados em *tiles* da arquitetura alvo, considerando os custos de cada mapeamento, já que os processadores são heterogêneos.

A atividade de mapeamento é ilustrada na Figura 5 através das setas que partem do nível de descrição de grupos formados por tarefas para a associação dos grupos em *tiles* selecionados. Este mapeamento gera associações entre processadores $P = \{p_a, p_b, ..., p_x\}$ e *tiles* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_y\}$, ou seja, posiciona os processadores fisicamente na arquitetura alvo. Estas associações entre processadores e *tiles* levam em consideração a heterogeneidade da arquitetura, uma vez que processadores específicos realizam operações especificas, o mapeamento leva em consideração a comunicação entre estes processadores, assim os posicionando na arquitetura de forma a minimizar a comunicação entre os mesmos.

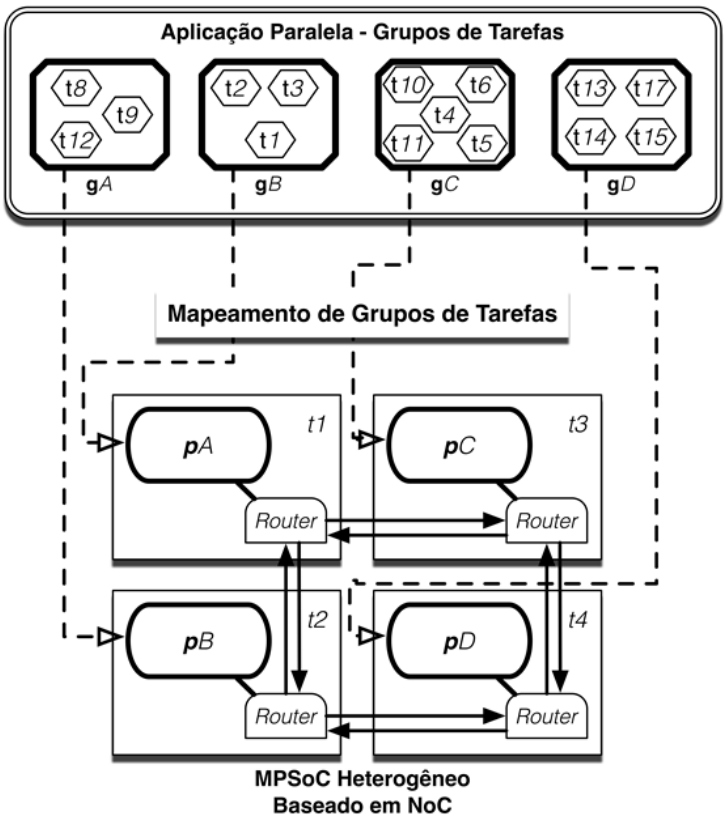


Figura 5. Mapeamento de grupos de tarefas previamente particionadas, em um MPSoC heterogêneo baseado em NoC.

As arquiteturas podem apresentar dois enfoques referentes ao particionamento e mapeamento, as quais (i) possui uma plataforma fixa, onde os processadores têm uma posição específica e não podem ser trocados de lugar, possibilitando apenas a alteração do mapeamento do grupo de tarefas entre os processadores; (ii) possui uma plataforma livre, assim os processadores podem ser posicionados em qualquer lugar da arquitetura, flexibilizando o processo de mapeamento dos mesmos.

Elementos relevantes para o particionamento e o mapeamento, como (i) modelo de descrição de uma aplicação, (ii) estruturação da NoC, (iii) modelo de energia na NoC, (iv) balanceamento de carga entre processadores e (v) requisitos de sistema, são descritos no Capítulo 3.

O particionamento, e também o mapeamento, podem ser aplicados em cenários estáticos (i.e. tempo de projeto) ou cenários dinâmicos (i.e. tempo de execução). A maior vantagem da abordagem estática é a independência do tempo para a exploração de soluções. Por outro lado, a abordagem dinâmica é capaz de lidar com comportamentos imprevisíveis, ou seja, ter controle e percepção da localização de uma tarefa de uma aplicação paralela alocada para executar em determinado processador. Desta forma, algoritmos estáticos empregados no particionamento (ou mapeamento), podem explorar soluções mais adequadas para um grupo de possibilidades conhecidas.

O particionamento de uma aplicação pode ser realizado como uma forma de simplificar o mapeamento posterior. Esta prática será denominada, no contexto deste trabalho, de **pré-mapeamento** (***Pre-mapping***) [9]. O pré-mapeamento consiste em aplicar o particionamento estático de forma a diminuir a complexidade (i.e. quantidade) de conexões que o mapeamento vai ter de tratar. Esta redução da complexidade acontece através do agrupamento de tarefas de uma aplicação, diminuindo assim o número de componentes que o mapeamento deve tratar.

A Figura 6 ilustra o uso do pré-mapeamento versus uma abordagem cujo mapeamento é direto. O particionamento e o mapeamento consideram os requisitos e restrições (e.g. consumo de energia e balanceamento de carga, conectados por flechas pontilhadas), e uma dada arquitetura MPSoC baseada em NoC (e.g. topologia NoC, número de processadores, tipo de processadores). O processo de pré-mapeamento e mapeamento é ilustrado pelo caminho contínuo, já o mapeamento direto é ilustrado por um caminho tracejado.

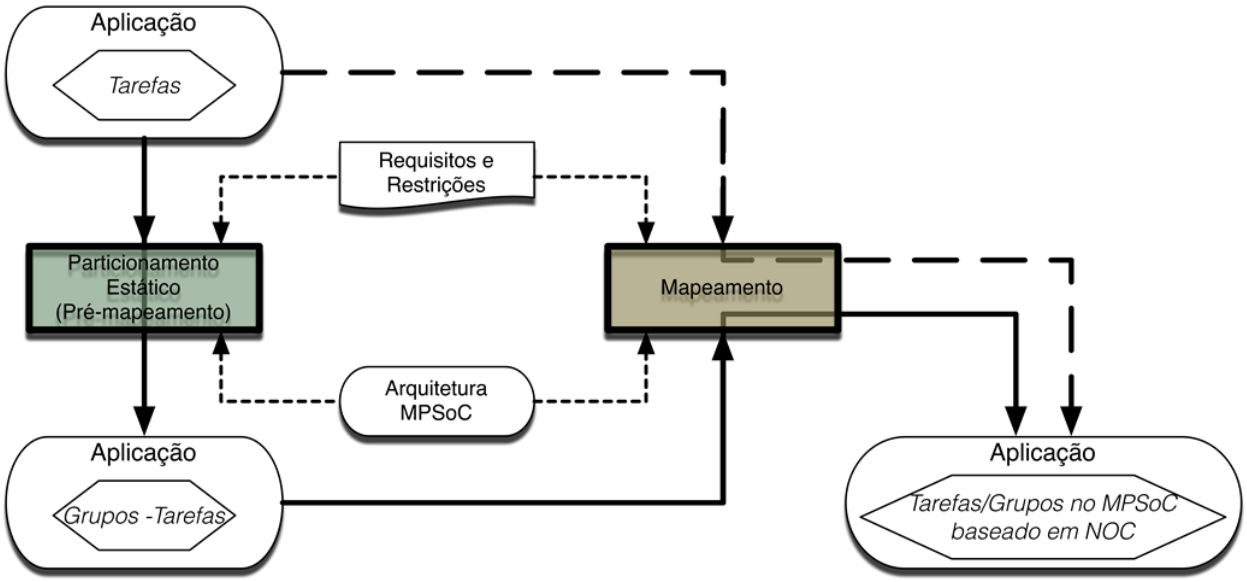


Figura 6. Pré-mapeamento e mapeamento de uma aplicação em um MPSoC baseado em NoC.

1.3 Motivação

A complexidade de aplicações embarcadas gera desafios como é o fato de não serem mais suportadas por um processador único de propósito geral, inevitavelmente requerendo plataformas computacionais de alto desempenho, permitindo realizar as tarefas da aplicação com desempenho satisfatório.

Para atender a esta demanda de complexidade de aplicações, MPSoCs heterogêneos de características e naturezas diversas são criados, adicionando ainda mais complexidade a solução de sistemas embarcados.

Dada esta premissa, um dos problemas gerado por esta soma de complexidades é o direcionamento das tarefas de aplicações em MPSoCs heterogêneos, ou seja, quais grupos de tarefas específicas de uma aplicação serão executados em quais processadores específicos, para que as restrições sejam atendidas e otimizadas.

Este trabalho objetiva abranger este problema, através da utilização da técnica de particionamento estático de tarefas em arquiteturas MPSoCs heterogêneas, como uma forma de agrupar certas tarefas de naturezas comuns relacionadas a um tipo de arquitetura alvo, abstraindo a complexidade para um eventual mapeamento destes grupos em processadores distintos de um MPSoC.

A técnica de particionamento de tarefas pode ajudar a minimizar a quantidade de comunicação entre núcleos, e também permitir atender às necessidades de computação de cada tarefa. A redução do volume de bits que trafegam na infraestrutura de comunicação reduz o consumo de energia dinâmica. Esta migração de tarefas pode gerar alto grau de trocas de contexto entre processos, mas oferece a vantagem da execução

otimizada de tarefas destinadas a processadores específicos, reduzindo o consumo de energia. O mapeamento de um grupo de tarefas, em um cenário heterogêneo, possibilita o direcionamento de certo grupo de tarefas com necessidades específicas a serem executadas em processadores especializados, melhorando o desempenho geral. Devido a estes aspectos, o particionamento e o mapeamento permitem reduzir o consumo de energia e o tempo de processamento, melhorando o desempenho do sistema.

1.4 Objetivos

Fazem parte do escopo deste trabalho a proposta e a avaliação de algoritmos de particionamento para a atividade de particionamento de tarefas em infraestruturas de comunicação com ênfase em arquiteturas do tipo heterogêneas, como forma de otimizar um mapeamento futuro. Todavia, para que seja possível obter dados comparativos de desempenhos de particionamento sobre a arquitetura alvo, foram definidos algoritmos clássicos como estudo de caso de algoritmos de particionamento. Este trabalho possui os seguintes objetivos estratégicos:

- Dominar tecnologias emergentes de infraestruturas de comunicação intrachip;
- Dominar arquiteturas de processamento paralelo intrachip;
- Dominar técnicas de particionamento e mapeamento de tarefas;
- Avaliar e propor métodos algorítmicos capazes de realizar o particionamento de tarefas;
- Explorar e avaliar a vantagem de aplicar a técnica de particionamento de tarefas como uma etapa pré-mapeamento, sendo que este pode beneficiar tanto o mapeamento estático quanto dinâmico.

Para alcançar os objetivos estratégicos, derivam-se os seguintes objetivos específicos:

- Explorar trabalhos relacionados, valorizando aqueles que modelam aplicações e infraestruturas de comunicações do tipo NoC, focadas principalmente nas atividades de particionamento e mapeamento;
- Analisar e compreender modelos de aplicações utilizadas em pesquisas relacionadas a MPSoCs;
- Explorar requisitos de projeto a serem atendidos com a utilização da técnica de particionamento de tarefas, como a minimização do consumo de energia, a redução do tempo de execução da aplicação e o balanceamento de carga;

- Pesquisar algoritmos clássicos de particionamento e mapeamento, implementando um conjunto alvo estipulado para realizar análise de desempenho sobre testes sintéticos aleatórios e testes extraídos de cenários reais;
- Elaborar novas abordagens algorítmicas inspiradas em algoritmos clássicos estudados, a fim de otimizar o particionamento de tarefas, e sugerir casos em que certas abordagens são melhores empregadas;
- Aplicar abordagens algorítmicas (elaboradas e estudadas) na etapa de particionamento, e com este resultado, avaliar a utilização deste como uma etapa pré-mapeamento, dinâmico e estático.
- Pesquisar aplicações reais onde a atividade de particionamento de tarefas atenda os requisitos utilizados no trabalho proposto - redução do consumo de energia e balanceamento de carga. Estas aplicações serão usadas para validar a ferramenta de particionamento, bem como o modelo de descrição propostos;

1.5 Estrutura do Trabalho

Este Capítulo introduziu o tema do trabalho e alguns aspectos conceituais relevantes à compreensão; o restante deste trabalho está estruturado da seguinte forma: O Capítulo 2 aborda trabalhos relacionados de particionamento e mapeamento. O Capítulo 3 ilustra os modelos teóricos utilizados em algoritmos auxiliares ao particionamento. O Capítulo 4 apresenta os algoritmos clássicos e desenvolvidos, aplicados ao particionamento de tarefas de aplicações paralelas. No Capítulo 5 é apresentado o *framework* de particionamento PALOMA, e a metodologia aplicada, desde o particionamento até o mapeamento, é ilustrada no Capítulo 6. O Capitulo 7 apresenta os resultados da utilização de diferentes abordagens algorítmicas aplicadas ao particionamento, e o desempenho de se aplicar o pré-mapeamento antes do mapeamento. O Capítulo 8 finaliza apresentando as principais conclusões deste trabalho.

2 TRABALHOS RELACIONADOS

Com o passar dos anos, vários *chips* multiprocessados foram desenvolvidos pela indústria e pela academia. Alguns deles projetados para um propósito específico, como por exemplo, processador digital de sinais (DSP); outros projetados para propósito geral. Alguns deles possuem uma arquitetura homogênea; outros exibem uma arquitetura heterogênea a fim de cobrir um largo espectro de necessidade sobre requisitos de computação. Independente da heterogeneidade do processador, para melhorar o desempenho geral, todos os sistemas multiprocessados podem se beneficiar do balanceamento de carga de tarefas entre processadores. Ainda, consumo de energia não pode ser negligenciado, principalmente devido ao uso portátil destes processadores, que a está aumentando a cada ano que passa. Muito trabalho nesta área tem sido feito para melhorar as técnicas de particionamento e mapeamento sobre estes requisitos. Este capítulo aborda um seleto grupo exemplar dos mesmos, que estão relacionados a este trabalho.

Particionamento e mapeamento são desafios de pesquisas importantes que foram abordados nos últimos anos em diversos trabalhos. Por exemplo, em 1995, Alpert et al. [11] apresentou uma análise descrevendo a pesquisa em particionamentos de *netlist*, discutindo abordagens de soluções em quatro grandes categorias, abordagens baseadas em movimento (*move-based)*, representações geométricas, fórmulas combinatórias, e abordagens de agrupamentos (*clustering)*. Na discussão, algoritmos clássicos dse particionamento como o Kerninghan & Lin (KL), Tabu Search (TS), Genetic Algorithms (GA), fórmulas de biparticionamento *Min-Cut*, Simulated Annealing (SA), e outros, são descritos e sugeridos como direções promissoras para pesquisa, mas salienta que pequenas diferenças na implementação podem levar a grandes diferenças na qualidade das soluções. Estas pesquisas ainda são foco em trabalhos como o de Sahu e Chattopadhyay [12] que apresentam uma análise contendo estratégias para mapeamento e particionamento da última década. Eles enfatizam que o mapeamento e o particionamento são dimensões importantes na pesquisa relacionada à NoC, visto que estas afetam o desempenho geral e os requisitos de energia do sistema. Os trabalhos classificam e comparam algumas estratégias de mapeamento visando prover entendimento dos esforços necessários e à qualidade da solução obtida em diferentes abordagens de mapeamento.

Como mostrado em Hu e Marculescu [15], o mapeamento de núcleos de propriedade intelectual, em inglês *IP* (*Intellectual Property*) *cores*, em tiles da arquitetura alvo é um problema de complexidade computacional na ordem *NP*-Completa, por ser uma instância do problema de atribuição quadrática restrita. Várias abordagens algorítmicas foram propostas, e podem ser geralmente classificadas em quatro categorias [14].

1. Algoritmos Branch-and-Bound

Hu e Marculescu [13][15] propuseram um algoritmo *branch-and-bound* que tenta buscar uma solução ótima para mapeamento de núcleos, através da alternância de passos de ramificação e salto, em uma arquitetura MPSoC baseada em NoC do tipo malha. Este algoritmo pode gerar soluções otimizadas devido ao seu grande espaço de busca, assumindo um valor considerável configurado para o tamanho da fila de trabalho. Com um grande tamanho de fila, entretanto, este algoritmo demanda alta quantia de memória e requer longo tempo de processamento. Para mapear N tarefas em M *tiles*, a complexidade de tempo seria O(*N!*). Assim, esta abordagem é apenas praticável em mapeamentos de núcleos IP em *tiles* de NoCs de tamanho relativamente pequeno, a menos que uma relação entre tempo de execução e a qualidade das soluções seja estipulada. É apresentado que com o uso do algoritmo de mapeamento *branch-and-bound* estático (i.e. algoritmo de mapeamento executado em tempo de projeto) é possível reduzir mais de 60% o consumo de energia se comparado soluções de mapeamento randômicas. Hu e Marculescu [15] melhoraram o seu trabalho anterior para mapeamento estático de uma aplicação em uma NoC do tipo malha considerando requisitos de desempenho como não exceder o limite da largura de banda. O trabalho anterior [13] considerava a comunicação da largura de banda em seu pior caso (i.e. todas as comunicações aconteciam simultaneamente). A abordagem mais recente [15] inclui a largura de banda de cada comunicação no mesmo modelo de aplicação a fim de superar as limitações do modelo anterior. Na literatura, técnicas e aprimoramentos do processo algorítmico têm sido sugeridos para reduzir tanto o requisito de memória, quanto o tempo de execução, limitando o tamanho da fila de trabalho, que adversamente impacta no espaço de busca e assim na qualidade da solução final.

Vivekanandarajah et al [16] apresentam um algoritmo baseado em *branch-and-bound* para mapear automaticamente uma aplicação para a arquitetura de tal forma que o tempo total de execução é minimizado. O algoritmo proposto pelos autores tem a descrição da aplicação e a descrição da arquitetura em forma de grafos ponderados, representando o fluxo de tarefas e a arquitetura alvo, respectivamente. A aplicação é modelada usando um modelo paralelo de computação e representada como um grafo acíclico dirigido (DAG – *Directed Acyclical Graph*). Como parte do algoritmo, uma heurística (*Greedy List Scheduling*) é proposta para encontrar a solução inicial. Além disso, são propostas heurísticas para calcular limites superiores e inferiores de soluções mapeadas parcialmente. Esta solução proposta é implementada e validada por modelos de arquiteturas alvo com aplicações geradas aleatoriamente. Através dos resultados obtidos, pode-se observar que o algoritmo proposto converge rapidamente em busca de uma solução ótima.

2. Algoritmos baseados em heurística - Greedy (guloso)

Vários algoritmos de heurística do tipo guloso [17][18] têm sido propostos baseados em diferentes observações das propriedades de topologias NoC e nos padrões de comunicação entre núcleos IP. Geralmente, algoritmos de mapeamento IP baseados no conceito *Greedy* possuem um baixo tempo de execução, mas geralmente com o sacrifício da qualidade das soluções geradas. Entretanto, se os algoritmos forem adequadamente implementados, problemas de degradação da qualidade das soluções geradas podem ser minimizados.

A fim de reduzir a dissipação de potência, um ponto importante a considerar é que núcleos IP que demandam comunicação, com latência rígida ou grande largura de banda, deveriam ser mapeados primeiro. Mapeamentos com algoritmo de roteamento pelo menor caminho [18], algoritmo de fluxo transversal [17], e algoritmo LCF [19] são todos baseados nesta observação. Estes algoritmos podem falhar na produção de soluções de alta qualidade onde um núcleo IP (fonte) necessita se comunicar com um grande número de núcleos (IP vizinhos). Neste caso, o núcleo IP fonte pode ser mapeado em um *tile* com o seu número de *tiles* vizinhos muito menor que o número de núcleos IP vizinhos. Como resultado, nos passos subsequentes do mapeamento, vários núcleos IP vizinhos do núcleo fonte devem ser mapeados em um *tile* que está mais distante. Assim, a comunicação entre o núcleo fonte e diversos de seus vizinhos poderá passar fisicamente por longas distâncias com dissipação de potência extra.

O problema de mapeamento mencionado acima pode ser minimizado levando em consideração o grau de cada IP (número de vizinhos) no processo de mapeamento. No algoritmo Spiral [20], um núcleo IP com um grau alto é priorizado para ser mapeado em um *tile* que conecte a um grande número de *tiles* vizinhos (e.g. o centro de uma NoC baseada no tipo malha), resultando em pouco atraso de comunicação. Entretanto, este algoritmo também pode falhar em gerar soluções de alta qualidade quando se tem núcleos IP com graus expressivamente grandes, mas a comunicação entre eles e seus vizinhos não impõem a necessidade de muita banda larga e baixa latência, ou quando o grau de cada IP é pequeno e todos os IPs tem basicamente o mesmo grau.

O algoritmo MOCA [21] atinge um equilíbrio entre tempo de execução e qualidade das soluções. O MOCA utiliza um algoritmo de particionamento de grafos para recursivamente particionar ambos, a NoC e o grafo de comunicação em duas metades, e a cada interação, uma parte do grafo de comunicação é mapeado em uma região da NoC. Um problema referente a este algoritmo é que pares de núcleos IP comunicantes que requerem muita banda larga, mas podem tolerar alta latência, podem ser mapeados entre *tiles* com um grande número de saltos, resultando em alto consumo de energia.

Murali et al. [22] consideraram múltiplos casos de uso durante o projeto. O trabalho apresenta técnicas de mapeamento estático para aplicações visando arquiteturas baseadas em NoCs do tipo malha ou torus, as quais satisfaz as restrições de projeto de cada caso de uso individual. O algoritmo aplicado no mapeamento é do tipo *greedy*, chamado UMARS (*Unified Mapping*, *Routing and Slot Allocation*) [17]. Este unifica em um único passo o roteamento de pacotes e o mapeamento de núcleos. A abordagem utilizada resulta em torno de 80% de redução de área da NoC e em torno de 54% de dissipação de potência, quando comparado com abordagens tradicionais de projeto.

Singh et al. [10] descrevem dois mapeamentos heurísticos em tempo de execução utilizados para mapear tarefas de uma aplicação altamente comunicantes a fim de minimizar a sobrecarga de comunicação. A ideia proposta objetiva aliviar gargalos de congestionamento na NoC para assim melhorar o desempenho geral, baseado em uma configuração experimental de mapear tarefas de uma aplicação em tempo de execução, em um MPSoC heterogêneo com uma NoC de tamanho 8x8. O trabalho utiliza dois algoritmos heurísticos de mapeamento baseados na abordagem de espaço de busca do vizinho mais próximo (*nearest neighbor*) e carga computacional do caminho. Os autores concluem que mapeamentos heurísticos reduzem consideravelmente a carga média do canal e o tempo total de execução para os cenários avaliados.

Em outro trabalho, Singh et al. [23] apresentam uma estratégia de mapeamento, onde a atribuição de uma tarefa é feita após olhar para tarefas previamente mapeadas sobre um elemento de processamento, em inglês *Processing Element* (PE), na plataforma MPSoC. São apresentadas heurísticas de mapeamento em tempo de execução, em que mais de uma tarefa é suportada por cada PE. As heurísticas de mapeamento analisam os recursos disponíveis antes de atribuir as tarefas no mesmo PE. Além disso, as heurísticas dão prioridade às tarefas de uma aplicação que estão mais próximas, de modo a minimizar a sobrecarga de comunicação. As heurísticas foram avaliadas através de uma NoC 8 x 8, onde se obteve uma redução no tempo de execução total, consumo de energia, carga média do canal e latência.

Lu et al. [24] implementaram um algoritmo de mapeamento de tarefas dinâmico chamado *Rotating Mapping Algorithm* (RMA) com o objetivo de reduzir o consumo de energia e a latência de aplicativos. O RMA mapeia aplicações descritas por *CTGs* (*Communication Task Graphs* – um modelo que captura o tempo de execução, período e *deadline* de cada tarefa, e o volume total de comunicação entre tarefas) em NoCs 2D do tipo malha. Resultados mostram mais de 42% de redução no consumo total de energia quando comparado com outros algoritmos de mapeamento de tarefas.

Habibi et al. [25] introduziram um esquema de mapeamento estático topológico que considera o requerimento de largura de banda e prioridade de fluxos de comunicação visando garantir QoS (*Quality of Service*) para operações *unicast* e *multicast*. O objetivo é

reduzir o atraso fim-a-fim e consumo de energia em NoCs do tipo malha. Os autores propõem modelar aplicações utilizando o *Application Characteristic Graph* (i.e. um modelo onde cada vértice consiste de um conjunto de tarefas com requerimentos computacionais, e cada arresta direcionada entre núcleos caracterizam o requerimento de comunicação). Resultados de simulações mostram que a heurística proposta, considerando volumes de comunicação entre núcleos para uma aplicação, tem atingido em média 28% de melhora no desempenho e 22% de economia de energia quando comparado com um algoritmo de mapeamento bem conhecido chamado *nMap*.

Kaushik et al. [26][27] propõem uma nova técnica que executa um pré-processamento no grafo da aplicação, antes do mapeamento propriamente dito, a fim de reduzir a sobrecarga de comunicação e melhorar o balanceamento de carga em vários processadores. Esta técnica começa por apontar as arestas de comunicação intensas na aplicação, e realiza tentativas para unir estas tarefas altamente comunicantes no mesmo PE. Depois disso, o grafo otimizado da aplicação, obtido pelo pré-processamento, pode ser mapeado usando um mapeamento heurístico em tempo de execução, como o *Nearest Neighbor*, o qual é empregado neste experimento. Deste modo, o pré-processamento é realizado antes do mapeamento, em tempo de projeto.

Singh et al [28] apresentaram uma melhoria no algoritmo *Nearest Neighbor*. Esta técnica foi avaliada contra a heurística *Smart Nearest Neighbor*, medindo o tempo de execução, utilização de recursos, o consumo de energia, e a variação de carga de computação, para mapear aplicações em uma plataforma. Os resultados mostram que a técnica proposta tem melhores resultados em um cenário composto de aplicações com 5, 10 e 15 tarefas.

Castrillon et al. [29] introduziram o *framework* MAPS que fornece suporte ao mapeamento múltiplo de aplicações *dataflow* em MPSoCs heterogêneos. Aplicações são modeladas como KPN (*Kahn Process Networks*) com um conjunto de restrições. MAPS oferece diferentes médias para a estimativa de desempenho e suporta uma variedade de mapeamentos heurísticos, sendo construído em uma plataforma virtual contendo elementos de processamento heterogêneos.

Ferrandi et al. [30] apresentam um algoritmo baseado na *otimização de colônia de formigas* (*Ant Colony Optimization – ACO*) para escalonamento e mapeamento de aplicações em MPSoCs heterogêneos. Este algoritmo explora diferentes alternativas de projeto para determinar um particionamento eficiente de hardware/software, de forma a decidir a atribuição de tarefas e estabelecer a ordem de execução destas. Os autores adotaram um grafo de tarefas hierárquico (HTG - *Hierarchical Task Graph*) como representação intermediaria da aplicação. O HTG fornece em cada nível da hierarquia um grafo de tarefa acíclico dirigido (DAG - *Direct Acyclic Graph*) sobre o qual os algoritmos de escalonamento e mapeamento podem operar.

3. Algoritmos baseados em Simulated Annealing (SA) / Algoritmos Genéticos (GA) / Tabu Search (TS)

Os algoritmos genéticos propostos em Ascia et al. [31] e Zhou et al [32] para mapear núcleos IP tinham como objetivo atender a múltiplos requisitos de projeto, como minimização da dissipação de potência, maximização do desempenho, ou a combinação dos dois. Por outro lado, Harmanani e Farah [33] propuseram um algoritmo baseado em *Simulated Annealing* (SA), e Lu et al. [34] propuseram um algoritmo baseado em *cluster* SA de tal forma que o mapeamento de núcleos IP pode ser feito utilizando o paradigma dividir e conquistar (*divide-and-conquer*). Nestes algoritmos, a concepção da condição de convergência (i.e. em que ponto da interação algorítmica, o resultado é suficiente) é a chave para procurar uma solução de alta qualidade. Apesar destes algoritmos serem capazes de evitar mínimos locais, não é garantido obter soluções de alta qualidade. É interessante ressaltar que este grupo de algoritmos necessita tempo consideravelmente maior que os algoritmos de mapeamento baseados na técnica de heurística *greedy*.

Marcon et al. [35] compararam algoritmos de mapeamento visando baixo consumo de energia em NoCs do tipo malha, utilizando um modelo de comunicação ponderado. Esta abordagem inclui o uso de buscas exaustivas, métodos de pesquisa estocásticos e técnicas heurísticas, passando por algoritmos clássicos da literatura (e.g. SA e TS). Além disso, eles propuseram dois novos algoritmos heurísticos, chamados LCF (*Largest Commucation First*) e Greedy Incremental. Abordagens mistas também são sugeridas, combinando o LCF e SA em duas maneiras diferentes, e também misturando o LCF e o TS. O resultado demonstra que soluções combinadas de método de pesquisa estocástica com métodos heurísticos podem levar a um melhor resultado em comparação com métodos puros.

Antunes et al. [36] apresentaram a influência do particionamento e mapeamento no consumo de energia em MPSoC homogêneos baseados em NoC. O trabalho compara duas estratégias para alcançar mapeamentos dinâmicos eficientes: (i) mapeamento de tarefas diretamente em processadores, e (ii) a aplicação prévia de um particionamento de tarefas estático, utilizando a informação obtida para realizar o mapeamento de tarefas dinâmico. Vários experimentos sintéticos mostram a eficiência de se aplicar previamente o particionamento estático de tarefas a fim de minimizar o espaço de busca do mapeamento, o que possibilita a construção de algoritmos eficientes de mapeamento dinâmico que possam realizar um mapeamento de tarefas ideal, em um curto período de tempo.

Salienta-se que ambos Kaushik et al. [26][27] e Antunes et al. [36] apresentam o uso de uma estratégia de pré-processamento (particionamento) antes do mapeamento, objetivando a melhoria da qualidade de mapeamento. Este trabalho também segue esta ideia; entretanto, este trabalho investiga o pré-processamento quando aplicado a cenários heterogêneos e utiliza diferentes abordagens algorítmicas para realizar o particionamento.

4. Algoritmos baseados em Programação Linear (Linear Programming)

Programação linear (LP) é utilizada em Murali e De Micheli [18] para resolver o problema de mapeamento. Neste caso, formular o problema LP correto é a chave para gerar soluções de alta qualidade. Embora existam algumas ferramentas computacionais que resolvem problemas LP, o tempo computacional é alto e não é garantido que sempre serão atingidas soluções de alta qualidade em mapeamento de núcleos IP.

He et al. [37] introduziram um modelo unificado combinando agendamento e mapeamento estático utilizando MILP (*Mixed Integer Linear Programming*) para reduzir o consumo de energia e a latência da NoC. Eles exploraram grafo de tarefas contendo volume de comunicação entre tarefas com restrições rígidas de *deadline* juntamente com o agendamento de tarefas em núcleos. Seus experimentos mostram que, para uma NoC do tipo malha de tamanho regular, MILP alcança mais de 11% de melhoria no tempo de execução com consumo de energia similar, em média. Em relação a NoCs irregulares e personalizadas, a abordagem melhora o tempo de execução com um menor consumo de energia.

Muitos dos trabalhos estudados se baseiam e utilizam somente a técnica de mapeamento de aplicações em MPSoCs, que de certa forma, tem em comum o princípio de funcionamento dos algoritmos utilizados no particionamento. Algoritmos de naturezas diversas são explorados na tentativa de obter soluções ótimas atendendo algum requisito de projeto, em uma arquitetura alvo. A grande maioria destes trabalhos, por aplicar diretamente o mapeamento, não possui uma etapa de redução de complexidade do problema de distribuir uma aplicação ao longo de um MPSoC. Desta forma, o trabalho proposto utiliza a técnica de particionamento, de forma estática, através de algoritmos clássicos utilizados também para o mapeamento, como destacado em trabalhos relacionados, como uma forma de pré-mapeamento em arquiteturas MPSoCs heterogêneas.

A Tabela 1 resume os trabalhos relacionados, classificando os mesmos de acordo com: (i) objetivos de projeto (particionamento e/ou mapeamento, estático/dinâmico); (ii) algoritmos de particionamento e/ou mapeamento; (iii) requisitos de projeto (e.g. economia de energia e minimização do tempo de execução total); e (iv) arquitetura alvo – tipo de MPSoC (homogêneo/heterogêneo) e arquitetura de comunicação (e.g. NoC do tipo malha).

Tabela 1 – Resumo de trabalhos relacionados.

Trabalho, Ano	Objetivos Projeto	Algoritmos	Requisitos Projeto	Arquitetura Comunicação
[10], (2009)	Mapeamento dinâmico	Espaço de busca heurístico	Minimização de alta comunicação	NoC malha, heterogênea
[12], (2003)	Mapeamento estático	*Branch-and-bound*	Economia de energia	NoC malha, homogênea
[15], (2005)	Particionamento estático Mapeamento estático	*Branch-and-bound*	Economia de energia e reserva de largura de banda	NoC malha, homogênea
[16], (2008)	Mapeamento estático	*Branch-and-bound*	Minimização do tempo total de execução	NoC malha, heterogênea
[17], (2005)	Mapeamento estático	*Greedy*	Minimização de energia e área	NoC malha, Homogênea e Heterogênea
[18], (2004)	Mapeamento estático	Heurístico	Minimização do atraso médio de comunicação	NoC malha, homogênea
[19], (2007)	Mapeamento estático	Heurístico, SA e TS	Economia de energia	NoC malha, homogênea
[20], (2007)	Mapeamento estático	Heurístico	Economia de energia	NoC malha, homogênea
[21], (2005)	Mapeamento estático	Heurístico	Economia de energia	NoC malha, homogênea
[22], (2006)	Mapeamento estático	*Greedy*	Minimização de energia e área	NoC malha/torus, homogênea
[23], (2010)	Mapeamento dinâmico	Espaço de busca heurístico	Minimização de alta comunicação	NoC malha, Homogênea e heterogênea
[24], (2010)	Mapeamento dinâmico	Heurístico	Economia de energia e minimização de latência	NoC malha, homogênea
[25], (2011)	Mapeamento estático	Heurístico	Economia de energia e minimização de atraso ponto-a-ponto	NoC malha, homogênea
[26], (2011)	Particionamento estático, mapeamento dinâmico	Heurístico	Balanceamento de carga e redução no volume de comunicação	NoC malha, homogênea
[27], (2011)	Particionamento estático, mapeamento dinâmico	Heurístico	Economia de energia, minimização de tempo total de execução e recursos	NoC malha, homogênea
[28], (2010)	Particionamento estático, mapeamento dinâmico	Heurístico	Economia de energia, minimização de tempo total de execução e recursos	NoC malha, homogênea
[29], (2013)	Mapeamento estático	Heurístico	Minimização de tempo total de execução	Arbitrária, heterogênea
[30], (2010)	Mapeamento dinâmico	ACO	Minimização de tempo total de execução	NoC malha, heterogênea
[31], (2004)	Mapeamento estático	GAs	Economia de energia e melhora de desempenho	NoC malha, homogênea
[32], (2006)	Mapeamento estático	GAs	Minimização do volume de comunicação e economia de energia	NoC malha, homogênea
[33], (2008)	Mapeamento estático	SA	Minimização de área e largura de banda	NoC malha, heterogênea
[34], (2008)	Mapeamento estático	SA combinado *clustering*	Minimização do volume de comunicação	NoC malha, homogênea
[35], (2008)	Mapeamento estático	Estocástico, heurístico e *greedy*	Economia de energia e tempo	NoC malha, homogênea
[36], (2011)	Particionamento estático, mapeamento estático	SA	Economia de energia	NoC malha, homogênea
[37], (2012)	Mapeamento estático	ILP	Economia de energia	NoC malha, regular/irregular, homogênea
Este Trabalho	**Particionamento estático, mapeamento estático/dinâmico**	**Algoritmos base: SA, TS, GA e KL.**	**Economia de energia e balanceamento de carga**	**NoC malha, heterogênea**

3 MODELOS UTILIZADOS PARA O PARTICIONAMENTO DE TAREFAS EM MPSOCS HETEROGÊNEOS

Este capítulo apresenta o embasamento teórico relacionado as representações baseadas em grafos de tarefas de uma aplicação e requisitos gerais do projeto como o modelo de consumo de energia e balanceamento de carga. Estes modelos apresentam um papel auxiliar no particionamento de tarefas de uma aplicação. A partir destes modelos, algoritmos responsáveis pela análise de custo do consumo de energia e comunicação, como também pelo balanceamento de carga, são utilizados como auxiliares pelos algoritmos de particionamento (e.g. SA, TS, KL, GA), a fim de terceirizar operações de cálculos que não estão relacionadas com a rotina dos algoritmos de particionamento propriamente dito. A Figura 7 exemplifica esta relação.

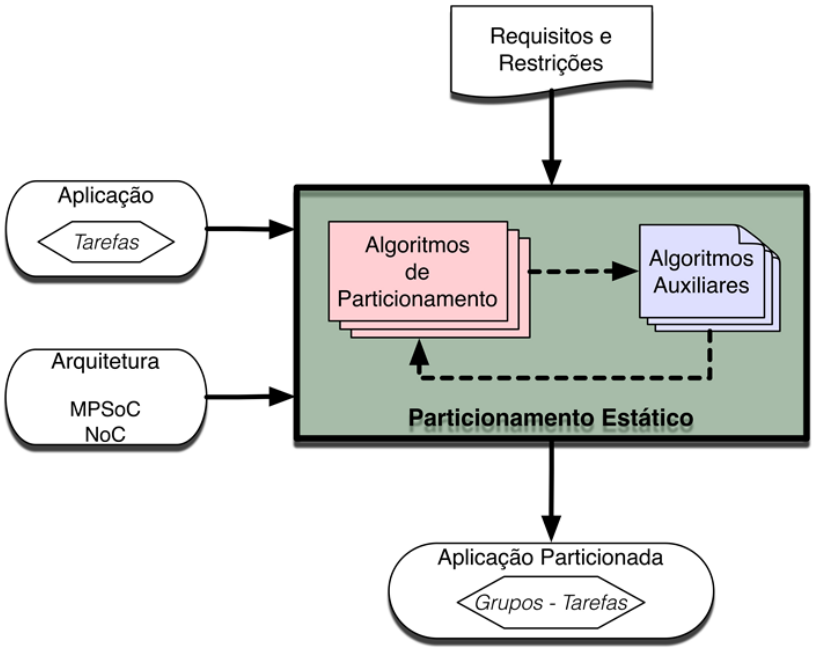


Figura 7. Relação entre algoritmos de particionamento e algoritmos auxiliares no particionamento estático.

3.1 Modelos de Descrição da Aplicação e MPSoC

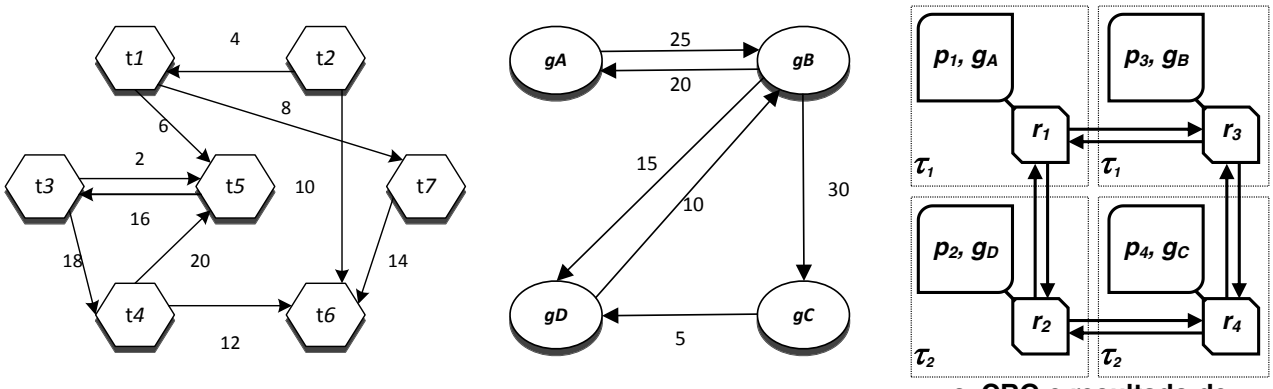
A descrição da aplicação e MPSoC é feita através de 3 definições baseadas em modelos de grafos:

Definição 1: Um grafo de comunicação de tarefas (TCG - *Task Communication Graph*) é um grafo direcionado $<T, S>$, onde $T = \{t_1, t_2, \ldots, t_n\}$ representa o conjunto de n tarefas em uma aplicação paralela, i.e. o conjunto de vértices TCG. Assumindo que s_{ab} é a quantidade de bits de pacotes enviados da tarefa t_a para a tarefa t_b, então o conjunto de aresta S é $\{(t_a, t_b) \mid t_a, t_b \in T \; s_{ab} \neq 0\}$, e cada aresta é rotulada com o valor s_{ab}, representando todas as comunicações entre as tarefas de uma aplicação.

Definição 2: Um grafo de peso de comunicação (CWG - *Communication Weighted Graph*) é um grafo direcionado *<G, W>*, similar ao TCG. No entanto o conjunto de vértices $G = \{g_1, g_2, \ldots, g_m\}$ representa o conjunto de *n* grupos de tarefas de uma aplicação, gerados pelo particionamento (i.e. pré-mapeamento), que possui como entrada o TCG. Além disso, w_{ab} é a quantidade total de comunicação (em *bits*) transmitidos de um grupo de tarefas g_a para um grupo de tarefas g_b. O conjunto de arestas *W* é $\{(g_a, g_b) \mid g_a, g_b \in G, w_{ab} \neq 0\}$, e cada aresta é rotulada com o valor w_{ab}, representando todas as comunicações entre os processadores do MPSoC, já que o mapeamento de grupos de tarefas atribui cada grupo de tarefas em um processador único. O CWG revela as informações do volume de comunicação relativa de uma aplicação.

Definição 3: Um grafo de recurso de comunicação (CRG - *Communication Resource Graph*) é um grafo direcionado $<\Gamma, L>$, onde $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_y\}$ representa o conjunto de *n tiles* (i.e. o conjunto de vértices CRG), onde cada *tile* contém um processador heterogêneo $P = \{p_1, p_2, \ldots, p_x\}$. Ainda, $L = \{(\tau_i, \tau_j), \forall\ \tau_i, \tau_j \in \Gamma\}$ corresponde ao conjunto de arestas CRG, i.e., o conjunto de caminhos roteáveis do *tile* τ_i para o *tile* τ_j. Os vértices e arrestas CRG representam, respectivamente, os roteadores $R = \{r_1, r_2, \ldots, r_z\}$ e suas conexões físicas.

A Figura 8 ilustra um exemplo de descrição utilizando modelos TCG, CWG e CRG.



a. TCG **b. CWG** **c. CRG e resultado de mapeamento**

Figura 8. Exemplo de descrição de uma aplicação e NoC.

A Figura 8(a) mostra um TCG, o qual representa o modelo de entrada para o particionamento (i.e. pré-mapeamento), onde T = {t1, …, t7}, e S = {(t1, t5) | 6, (t1, t7) | 8, (t2, t1) | 4, (t2, t6) | 10, ...}. O resultado do particionamento é ilustrado na Figura 8(b) contendo um CWG com G = {gA, …, gD} e W = {(gB, gC) | 30, (gC, gD) | 5, …}. O modelo CWG e o CRG são entradas para o mapeamento, cujo resultado é mostrado na Figura 8(c), onde T = {τ1, …, τ4} e L = {(τ1, τ2), (τ1, τ3), (τ2, τ4), (τ3, τ4), …}. Adicionalmente, cada tile τi contém um processador pi de um dado tipo, e o mapeamento produz a seguinte associação {(p1, gA), (p2, gD), (p3, gB), (p4, gC)}. Este exemplo pode ser mais

bem entendido, acompanhando o fluxo do processo e seus modelos correspondentes, exposto na Figura 9.

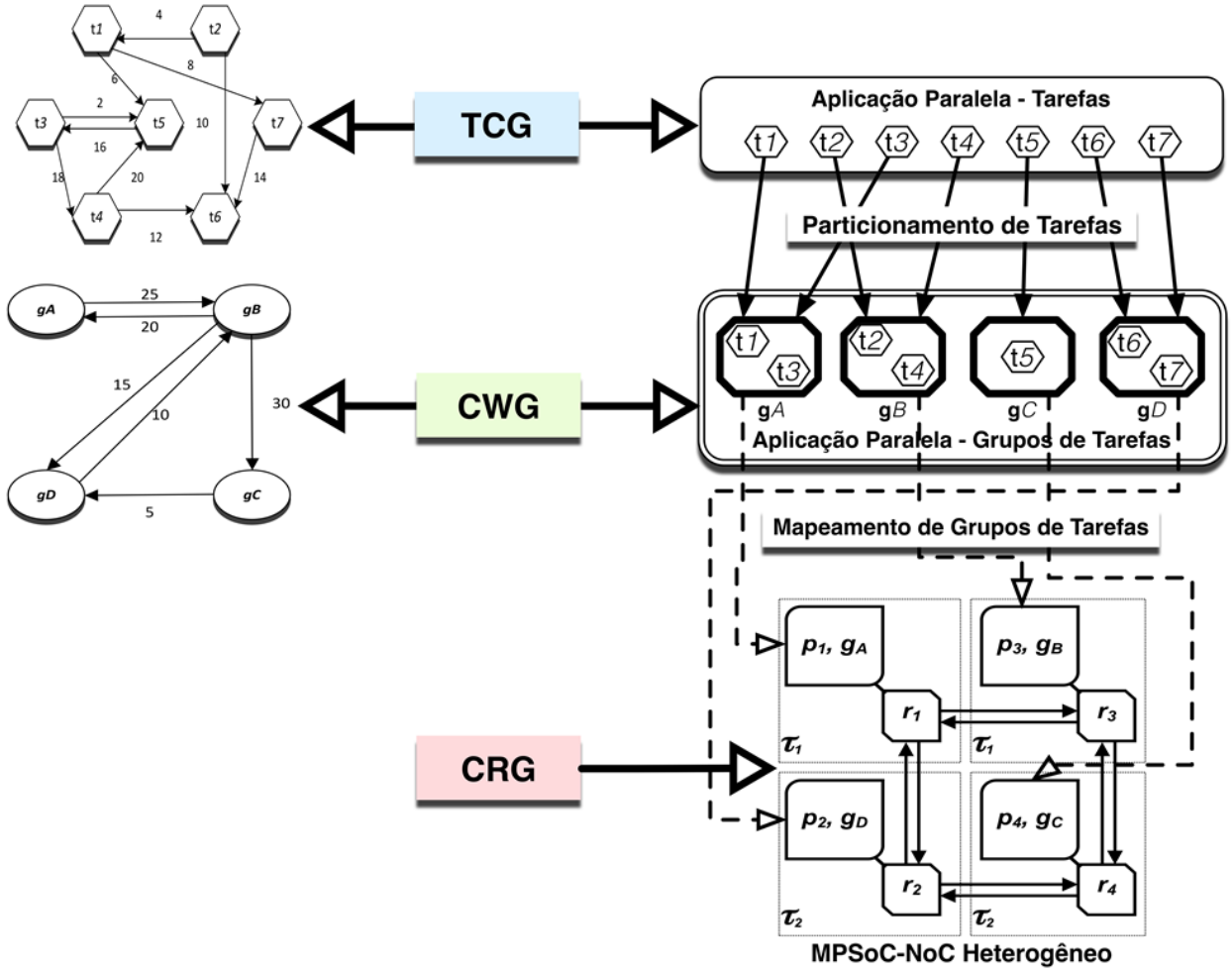


Figura 9. Modelos de descrição destacados no processo de particionamento e mapeamento de uma aplicação.

3.2 Modelo e Algoritmo do Consumo de Energia do MPSoC

Processadores e infraestrutura de comunicação têm alta influência no consumo de energia de aplicações mapeadas em MPSoCs. A soma da energia consumida, de todas as tarefas executadas em um dado processador, possibilita estimar o consumo de energia individual de um processador. Em um cenário heterogêneo, o consumo de energia em cada processador, por uma dada tarefa, pode variar, e.g. de acordo com a arquitetura como também procedimentos otimizados considerando o tipo do processador. A quantidade de *bits* trocados entre processadores proporciona o consumo total de energia relacionado à arquitetura de comunicação. A energia consumida por tarefas executadas em processadores, somada a energia consumida na arquitetura de comunicação, determinam as escolhas de particionamento e mapeamento.

O modelo de consumo de energia aplicado neste trabalho é similar a [35]. O consumo de energia dinâmico está relacionado com a troca de pacotes ao longo da NoC,

consumindo energia dentro de cada roteador e em cada conexão por onde passam os pacotes. E_{bit} é uma estimativa do consumo de energia dinâmica para cada *bit*, quando o *bit* troca seu valor (i.e. polaridade). E_{bit} é dividido em três componentes: (i) ER_{bit} – energia dinâmica consumida nos componentes do roteador (e.g. fios, *buffers* e portas lógicas); (ii) ELH_{bit} e ELV_{bit} – energia dinâmica consumida nas conexões horizontais e verticais entre *tiles*, respectivamente; e (iii) EC_{bit} – energia dinâmica consumida nas conexões entre cada roteador e seu processador local.

Para NoCs 2D do tipo malha com *tiles* de dimensões quadráticas, é razoável estimar que ELH_{bit} e ELV_{bit} possuem o mesmo valor. Acrescentando, é assumido que EL_{bit} é uma maneira simplificada de representar ELH_{bit} e ELV_{bit}. A Eq. (1) computa a energia dinâmica consumida por um *bit* passando por uma NoC de um *tile* τ_i para um *tile* τ_j, com η sendo o número de roteadores em que o *bit* percorre.

$$(1) \qquad Ebitij = \eta \times ERbit + (\eta - 1) \times ELbit + 2 \times ECbit$$

Sendo τ_i e τ_j os *tiles* que contém p_a e p_b, respectivamente, a energia dinâmica consumida por todos os tráfegos de comunicação $p_a \rightarrow p_b$ é dada pela Eq. (2). Enquanto que a quantidade de energia consumida pela NoC (*ENoC*) relacionada a todo o tráfego de comunicação entre processadores (|W|) é dado pela Eq. (3).

$$(2) \qquad E_{bitab} = W_{ab} \times E_{bitij}$$

$$(3) \qquad ENoC = \sum_{i=1}^{|W|} E_{Bitab}(i) \;,\; \forall\, p_a, p_b \in P$$

Ambas as funções de custo de particionamento e mapeamento usam os parâmetros do modelo de energia da NoC estabelecido pela Eq. (1); entretanto, o mapeamento fornece a associação de uma dada tarefa ou grupo de tarefas a um processador posicionado em um *tile* específico, enquanto que o particionamento apenas explora as necessidades da comunicação sem levar em consideração a informação da posição de cada processador (i.e. o número de saltos (*hops*) entre dois processadores comunicantes é desconhecido). Devido a este fato, a função de custo de particionamento utiliza o conceito de *média de saltos* (*Average of Hops*), o qual possibilita computar a média de consumo de energia de todos os caminhos possíveis.

Sendo ambos X e Y, o número de *tiles* nas dimensões horizontais e verticais de uma NoC, respectivamente. Portanto a Eq. (7) computa o número total de saltos dos caminhos que todos os processadores possuem, relacionados com o algoritmo de roteamento XY. A *média de saltos* é computada dividindo a soma de todos os saltos, de todos os caminhos entre processadores, pelo número total de comunicações, descrita pelas Eq. (4), (5), (6), e (7).

(4) $$Hops_{total} = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \sum_{i=0}^{X-1} \sum_{j=0}^{Y-1} (x - i + y - j)$$

(5) $$Num_{Processors} = X \times Y$$

(6) $$Max_{Comm} = Num_{Processor} \times (Num_{Processors} - 1)$$

(7) $$Hops_{Average} = \frac{Hops_{Total}}{Max_{Comm}}$$

O valor $Hops_{Average}$ é aplicado na Eq. (1) substituindo o valor η, resultando em um valor médio de E_{Bitij}. Assim, a estimativa de consumo de energia, de cada comunicação utilizada durante o particionamento (i.e. pré-mapeamento), é o resultado de uma multiplicação de E_{Bitij} pelo volume de comunicação. A Figura 10 exemplifica a aplicação destas equações relacionando ao modelo CWG anteriormente visitado.

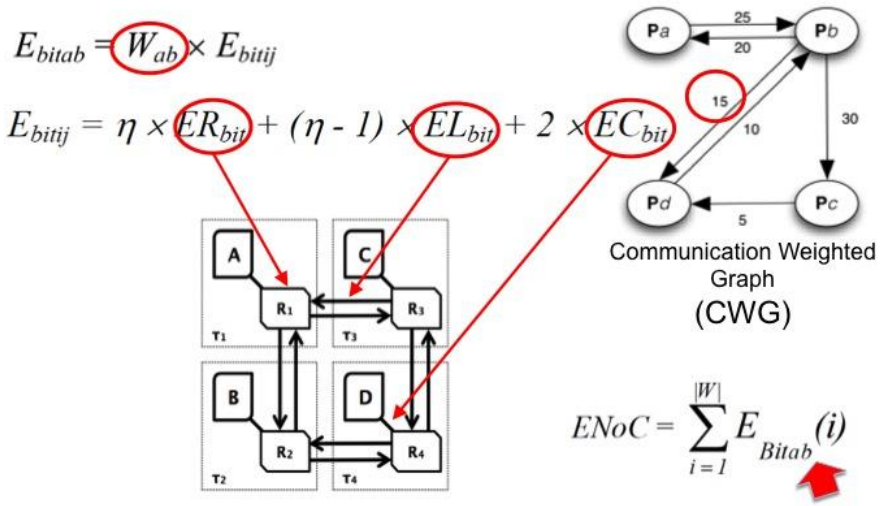


Figura 10. Relação entre o modelo de consumo de energia e o modelo CWG. O ER_{bit} esta relacionado com energia do roteador; EL_{bit} está relacionado com a energia na conexão entre roteadores; e EC_{bit} está relacionado com a energia entre o processador e o roteador. O volume de comunicação utilizado na estimativa de consumo de energia está destacado no CWG como exemplo.

O algoritmo *ENoC* é implementado com quatro laços aninhados, cujo pseudocódigo é ilustrado na Figura 11. O laço mais externo (linhas 2 a 11) pesquisa pelo processador origem de uma comunicação entre processadores em todas as associações processador–tarefas. O laço interno (linhas 3 a 10) pesquisa pelo processador alvo correspondente em todas as associações processador-tarefas. Observe que o algoritmo emprega o termo “associação”, ao invés de mapeamento, já que este algoritmo é aplicado para ambas às atividades - particionamento e mapeamento.

```
1.  cost ← 0;
2.  for(Association sa: associations) {
3.    for(Association ta: associations) {
4.      if(sa.equals(ta))
5.        continue;
6.      for(Task st: sa.getTask()) {
7.        for(Task tt: at.getTask())
8.          cost ← cost + st.computeEnergy(tt);
9.      }
10.   }
11. }
```

Figura 11. Pseudocódigo do algoritmo *ENoC*.

A energia consumida pela comunicação entre processador é computada pela função *computeEnergy*, dentro de dois laços aninhados (da linha 6 a 9). Enquanto que a função *computeEnergy* implementa a Eq. (2), a *ENoC* proveniente da Eq. (3) é o último valor a ser armazenado na variável *cost*, o qual é o valor de retorno da função *actualCost()* do algoritmo descrito na Figura 11.

3.3 Modelo e Algoritmo de Balanceamento de Carga

Particionamento ou mapeamento visam distribuir tarefas de uma aplicação entre processadores, atendendo alguns critérios. Ainda, distribuição de carga de trabalho de modo racional exige uma supervisão de balanceamento de carga a fim de evitar concentração de tarefas em um pequeno número de processadores, enquanto outros processadores estão em estado de repouso (i.e. sem tarefas a computar). Para atacar este problema, um modelo de balanceamento de carga foi aplicado baseado na minimização do *erro médio quadrático, em inglês Mean Square Error* (*MSE*), descrito na Eq. (8). Esta equação representa uma das muitas maneiras de se quantificar a diferença entre valores implícitos em um estimador e os verdadeiros valores de uma quantidade estimada. *MSE* mede a média dos quadrados dos erros (representado por *e*) pela população, representado por *n*. O erro é a quantia pelo qual o valor implícito no estimador difere da quantia a ser estimada. Esta diferença ocorre devido à aleatoriedade, ou devido ao estimador não levar em conta informações que poderiam produzir uma estimativa mais precisa.

$$MSE = \frac{\sum_{t=1}^{n} e_t^2}{n} \quad (8)$$

O algoritmo de balanceamento de carga (Figura 12), o qual implementa a Eq. (8), primeiro computa a carga de trabalho total de todos os processadores, em dois laços aninhados (linhas 2 a 10), armazenado na variável *totalCpusUse*. O laço mais externo (linhas 2 a 10) procura o processador de uma dada associação processador-tarefas, e seu tipo. O laço mais interno (linhas 6 a 7) computa a todas as tarefas associadas ao processador, a carga de trabalho correspondente, a qual leva em consideração o tipo do

processador. Então, o laço mais externo salva a carga de trabalho de todas as tarefas para um uso futuro.

```
1.  totalCpusUse ← 0;
2.  for(Association a: associations) {
3.     cpu ← a.getProcessor();
4.     cpuType ← a.getProcessorType();
5.     cpuUse ← 0;
6.     for(Task t: a.getTask())
7.        cpuUse ← cpuUse + t.getCpuUse(cpuType);
8.     cpu.setCpuUse(cpuUse);
9.     totalCpusUse ← totalCpusUse + cpuUse;
10. }
11. averageCpusUse ← totalCpusUse / size();
12. mse ← 0;
13. for(Association a: associations) {
14.    cpu ← a.getProcessor();
15.    abse ← averageCpusUse - cpu.getCpuUse();
16.    mse ← mse + abse x abse;
17. }
18. mse ← mse / size();
```

Figura 12. Pseudocódigo do algoritmo de balanceamento de carga.

A média de carga de trabalho por processador é armazenada na variável *averageCpusUse*. O laço das linhas 13 a 17 computam o *erro absoluto* (*Absolute Error – ABSE*), i.e. a diferença entre a *averageCpusUse* e a carga de trabalho de cada processador. Este valor é elevado ao quadrado e acumulado na variável *mse*, de tal modo que todos os *ABSE* computados sejam armazenados em *mse*. Por fim, *mse* é dividido pelo número de processadores, originando o *MSE*, o qual é o valor de retorno da função *actualCost()* do algoritmo descrito na Figura 12. Esta aproximação é uma maneira rápida e simples de detectar erros, geralmente levando a um bom balanceamento de carga nos resultados de particionamento.

4 ALGORITMOS DE PARTICIONAMENTO

Este capítulo aborda os algoritmos clássicos SA, TS, GA e KL, além do algoritmo BIA, empregados no processo de particionamento de tarefas, os quais realizam o particionamento propriamente dito, como ilustrado na Figura 7. O particionamento é dependente do modo como trabalham os diferentes algoritmos empregados neste procedimento, uma vez que possuem naturezas, ou filosofias, diferentes entre si. Estas diferenças fazem com que certos tipos de algoritmos sejam mais indicados para determinados cenários ou aplicações, levando em conta o requisito principal que se queira atender.

Os problemas de particionamento e mapeamento de tarefas, como mencionado anteriormente, possui natureza NP-completa, o que impede a busca por soluções exaustivas. Por outro lado, estes tipos de problemas apresentam auto-similaridade [6], que acontece quando os possíveis grupos de soluções são relacionados com os resultados da função de custo utilizado pelo algoritmo responsável, acarretando em vários grupos com o mesmo custo atribuído. Assim, abordagens algorítmicas que explorem um ambiente de possíveis soluções das formas mais distintas, mesmo que ao acaso, possuem a capacidade de gerar resultados de alta qualidade, desde que sua implementação, como o refinamento de resultados alcançados, sejam bem planejados.

A Figura 13, que apresenta o resultado da execução de um algoritmo de pesquisa exaustiva gerando 720 possibilidades de solução, ilustra o conceito de “vale” (região entre dois máximos locais) relacionando a exploração de mínimos locais.

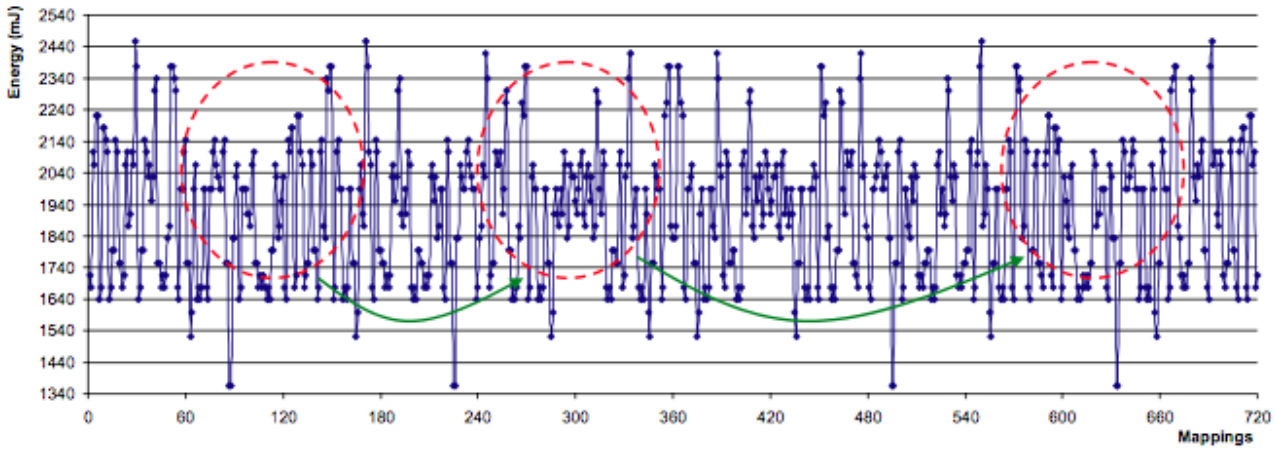


Figura 13. Ilustração do consumo de energia dos 720 possíveis mapeamentos de uma aplicação sintética com 6 núcleos, e uma correspondente pesquisa por soluções com SA. Este algoritmo é composto por dois laços aninhados. Os círculos pontilhados exemplificam o espaço de busca do laço interno (exploração de mínimos locais), enquanto que as setas seriam saltos do laço externo (pesquisas por novos mínimos) [7].

A partir destas premissas, classes alternativas de algoritmos são exploradas, envolvendo algoritmos heurísticos, estocásticos, genéticos, ou mesmo determinísticos.

Algoritmos clássicos como o *Simulated Annealing (SA)* e *Taboo Search (TS)* e *Genetic Algorithms (GA)* pertencem a estas classes, e são satisfatoriamente empregados nos procedimentos de particionamento e mapeamento.

Dentro da classe de algoritmos determinísticos, um grupo de algoritmos relacionados à bissecção de grafos é utilizado no particionamento de *netlists* e esquemas elétricos. Percebendo o sucesso da utilização de algoritmos determinísticos relacionados à bissecção de grafos em particionamento de circuitos, alguns algoritmos membros desta classe foram estudados a fim de serem aplicados ao particionamento de tarefas, resultando na utilização do algoritmo *Kernighan & Lin (KL)*.

Levando em consideração que mesmo algoritmos clássicos podem ser implementados de diferentes maneiras, mantendo ainda seu modelo básico, mas mudando alguns aspectos que influenciam sua complexidade computacional, este Capítulo descreve e discute os pseudocódigos dos algoritmos utilizados neste trabalho visando o problema de particionamento de tarefas. Dentre os algoritmos estudados, os seguintes foram utilizados: dois algoritmos estocásticos de busca (i.e. *Simulated Annealing (SA)* e *Tabu Search (TS)*), um algoritmo heurístico do tipo genético (i.e. *Genetic Algorithm (GA)*) e um algoritmos heurísticos de bissecção baseados no algoritmo *Kernighan & Lin (KL)*.

4.1 Simulated Annealing (SA)

O algoritmo Simulated Annealing (SA) [6][23][38][39] é do tipo estocástico com características probabilísticas, utilizado em problemas de otimização combinatória, mas aplicável a inúmeros problemas de naturezas diversas. A palavra *annealing* refere-se ao processo utilizado para fundir um metal, onde este é aquecido a uma temperatura elevada e em seguida é resfriado lentamente, de modo que o produto final seja uma massa homogênea. A ideia do algoritmo é explorar diversas soluções através do uso de dois laços aninhados. Cada solução local gerada é aceita ou rejeitada de acordo com certa probabilidade. Esta probabilidade de aceitação decresce de acordo com o nível do processo, ou equivalentemente, de acordo com a temperatura. De forma análoga, quando aplicado na atividade de particionamento, o laço externo tenta localizar soluções bem distintas objetivando alcançar mínimos globais. O laço interno, por sua vez, explora pequenas modificações da solução obtida pelo laço externo, objetivando encontrar mínimos locais - ou seja, realiza refinamentos na solução provida pelo laço externo. Os laços são controlados por um parâmetro chamado **temperatura**. À medida que o algoritmo progride, o valor da **temperatura** é decrementado, começando o algoritmo a converter para uma solução ótima, necessariamente local. Assim, relacionando o algoritmo com o particionamento, a energia no material corresponde à qualidade do particionamento gerado pelo algoritmo.

Um *annealing schedule* especifica (i) uma temperatura inicial, (ii) uma função de decremento da temperatura, (iii) uma condição de equilíbrio a cada temperatura, e (iv) uma condição de convergência (ou congelamento). O método *simulated annealing*, segundo a descrição clássica, começa com um particionamento inicial aleatório. Seja Δs um valor que define uma variação entre dois custos de particionamento, usualmente conhecido por *threshold*, um particionamento alterado é gerado, e as mudanças resultantes em Δs são calculadas. Se $\Delta s < 0$, a energia do sistema é reduzida, então o movimento é aceito. Se, $\Delta s \geq 0$, então o movimento é aceito com probabilidade. À medida que a temperatura *t* simulada diminui, a probabilidade de se aceitar um Δs maior diminui.

A Figura 14 apresenta o pseudocódigo do algoritmo SA, que explora várias soluções de particionamento, utilizando dois laços principais. O laço mais externo (linhas 3 a 26) gera partições com grandes modificações para explorar mínimos globais. O laço mais interno (linhas 10 a 25) explora pequenas modificações dentro da partição fornecida pelo laço externo, visando encontrar uma partição local ótima; i.e. o laço mais interno refina o particionamento fornecido pelo laço externo.

```
1.   globalMinimumCost ← Maximum value
2.   interaction ← Interaction input parameter
3.   while(interaction > 0) {
4.     interaction--
5.     if(randomBigMove() == false)
6.       continue
7.     localMinimumCost ← actualCost()
8.     saveMoveAsLocalMinimum()
9.     temperature = Temperature input parameter
10.    while(temperature > 0) {
11.      temperature--
12.      if(randomSmallMove() == false)
13.        continue
14.      if(localMinimumCost > actualCost()) {
15.        localMinimumCost ← actualCost()
16.        saveMoveAsLocalMinimum()
17.      }
18.      else
19.        if(!acceptableThreshold(temperature, actualCost(), localMinimumCost))
20.          restoreLocalMinimumMove()
21.    }
22.    if(globalMinimumCost > localMinimumCost) {
23.      globalMinimumCost = localMinimumCost
24.      saveLocalMinimumMoveAsGlobal()
25.    }
26.  }
```

Figura 14. Pseudocódigo do algoritmo SA.

Uma pesquisa aleatória em um amplo espectro de possibilidades é o efeito gerado pelos dois laços aninhados que possibilita ao SA achar partições que minimizem a função *actualCost()* – o objetivo do particionamento. O laço interno começa com um particionamento totalmente aleatório, fornecido pela função *randomBigMove()* proveniente do laço externo, a qual o custo mínimo é armazenado na variável *localMinimumCost*, aplicando pequenas variações aleatórias na partição com a função *randomSmallMove()*.

A cada fim do laço interno, a variável *localMinimumCost* é comparada com o valor anteriormente armazenado. Se o valor atual for menor que o valor armazenado, o valor

atual é então armazenado como o melhor e mais novo particionamento, e assim se torna o atual novo particionamento. Ao mesmo tempo, alguns valores de particionamento ruins podem ser aceitos como atuais novos particionamentos devido ao parâmetro *temperature*, utilizado para controlar o procedimento de aceitação estocástica (linhas 19 a 20). Este parâmetro, quando carregado com valores altos, implica em uma grande probabilidade de se aceitar particionamentos piores. Se uma partição ruim não for aceita, então as últimas sequências de movimentos são ignoradas e o sistema retorna para a partição mínima local anterior, através da execução da função *restoreLocalMinimumMove()*. O parâmetro *temperature* é decrementado a cada execução do laço interno, e reiniciado a cada laço externo.

As funções *randomBigMove()* e *randomSmallMove()* (lines 5 a 12, respectivamente) exploram particionamentos aleatórios e tem o papel de retornar um estado *Boolean* se um dado particionamento preencher as restrições atualmente configuradas (i.e. limites de consumo de energia e ocupação do processador). Se o particionamento atual não conseguir atender as restrições atuais, o parâmetro *iteration* é decrementado, e um novo conjunto de particionamento é explorado. O melhor particionamento obtido ao fim do laço interno é comparado com o melhor particionamento global; e aquele com o menor custo é armazenado como o melhor e mais novo particionamento.

4.2 Tabu Search (TS)

Tabu Search (TS) [39][43][40][41][42] é um método de otimização que utiliza uma memória de curta duração para evitar que o processo de procura permaneça em um mínimo local. A origem da palavra *tabu*, proveniente da língua Tongan da Polinésia, era utilizada por aborígenes locais para indicar coisas que não poderiam ser tocadas por serem sagradas. De forma análoga, uma ***lista tabu*** é formada para gravar a trajetória de uma solução recente. A cada iteração em um processo de otimização, as soluções são verificadas contra a *lista tabu*. A solução que estiver na *lista* não será escolhida para a próxima iteração (a menos que a solução se sobressaia a sua condição *tabu*, chamado de condição de aspiração - ***aspiration condition***). A *lista tabu* forma o núcleo do algoritmo *TS* e mantém o processo em ciclo contínuo em uma vizinhança do espaço de soluções. A cada iteração, a solução que se encontra em uma posição mais inferior na lista, e que não está violando a condição *tabu*, é escolhida. Se não houver uma solução melhor fora da *lista tabu,* a melhor solução das já conhecidas é escolhida. A combinação de memória e gradiente de descida possibilita a diversificação intensa da procura, de forma que mínimos locais são evitados no espaço de procura enquanto que as áreas restantes são bem exploradas.

Este algoritmo de pesquisa utilizando uma técnica combinatória é empregado em uma série de problemas de otimização. A pesquisa algorítmica é similar à melhoria iterativa, a qual movimentos são necessários, transformando a solução atual na melhor solução da vizinhança. O TS mantém a *lista tabu* de seus *rm* movimentos mais recentes (e.g. pares de tarefas que foram recém trocados), sendo *rm* uma constante prescrita que determina o tamanho da *lista tabu*; e movimentos utilizando elementos que fazem parte da *lista tabu* não podem ser realizados [11].

A *lista tabu* é o centro do TS, evitando que o processo fique circulando próximo a um mínimo local e também possibilitando movimentos de subida (que aumentem o custo de particionamento) [43]. A cada iteração, as soluções alcançadas no espaço de pesquisa são verificadas contra a *lista tabu*. Se uma solução estiver na *lista tabu*, esta solução não será escolhida para a próxima iteração, a menos que ela anule a *condição tabu*. Ainda, a cada iteração, uma solução de descida íngreme que não viole a *condição tabu* é selecionada. Se não existir uma solução aprimorada, a melhor solução não aprimorada é escolhida. A combinação de memória e descida gradual possibilita uma diversificação e uma intensificação a pesquisa algorítmica, e mínimos locais são evitados enquanto que soluções de qualidade são bem explorados [43]. Em contraste ao SA que explora movimentos aleatórios, o TS explora estruturas de dados do histórico da pesquisa como uma condição para próximos movimentos. Geralmente, o TS é um método de pesquisa projetado para cruzar limites de viabilidade, normalmente tratados como barreiras, e ele sistematicamente impõem e libera restrições a fim de explorar regiões proibidas [39][43].

Similarmente ao SA, o TS é implementado com dois laços aninhados, cujo pseudocódigo é apresentado Figura 15. O laço mais interno do TS (linhas 7 a 13) gera ao menos um par de candidatos à *troca* que é realizada no laço mais externo (linhas 2 a 23).

```
1.   interaction ← Interaction input parameter
2.   while(interaction > 0) {
3.      interaction--
4.      if(randomBigMove() == false)
5.        continue
6.      temperature ← Temperature input parameter
7.      while(temperature > 0) {
8.        temperature--
9.        if(randomSmallMove() == true) {
10.         if(candidateIsNotOnTabuList())
11.           addCandidateToCandidateList()
12.       }
13.     }
14.     if(locateBestCandidate()) {
15.       if(candidateBetterThanBest() {
16.         candidateIsNewBest()
17.         saveMoveAsMinimum()
18.       }
19.       addFeaturesDifferences()
20.       if(tabuListIsFull())
21.         expireFeaturesOfTabuList()
22.     }
23.  }
```

Figura 15. Pseudocódigo do algoritmo TS.

A computação do consumo de energia é incremental, como no algoritmo SA. O laço interno utiliza a função *randomSmallMove()* para aleatoriamente buscar pares de módulos para trocar, procurando pelo par que melhor atende uma dada exigência, e a exigência computacional é realizada pela função *locateBestCandidate()*, que também contém a função *actualCost()* (não descrita no pseudocódigo da Figura 15).

Uma lista de trocas com todos os pares distintos de módulos aptos para a troca otimiza a busca realizada pela função *addCandidateToCandidateList()*. Esta função produz aleatoriamente índices para realizar o acesso à lista de trocas. Se uma troca tem um custo menor que o anterior, este par é armazenado na lista de trocas. Quando o laço interno terminar, se um par que minimiza o custo é encontrado, este é adicionado à *lista tabu*, removido da lista de trocas (linhas 14 a 22), e a partição atual é modificada com a troca selecionada. Se nenhum par existir, nenhuma ação é tomada, e uma nova execução do laço interno é realizada.

Os principais parâmetros do algoritmo TS são *neighborhood* e *iteration*. O parâmetro *neighborhood* foi renomeado para *temperature* a fim de manter uma similaridade com o algoritmo SA. Deste modo, *temperature* significa o número de pares avaliados a cada passo de troca; e *iteration*, a qual controla o laço externo, significa o número de trocas realizadas.

4.3 Genetic Algorithm (GA)

O algoritmo *Genetic Algorithms* (GA) [43][39][44] é baseado na seleção natural Darwiniana. O algoritmo possui quatro operações principais: **avaliação**, **seleção**, ***crossover*** e **mutação**. Iniciando com uma dada população, o grau de ***fitness*** para cada membro da população é avaliado de acordo com uma função de *fitness* (i.e. um tipo particular de função objetiva que é utilizada para resumir, como uma única figura de mérito, o quão perto uma solução desenvolvida está para atingir o conjunto visado). Um conjunto de pais é então selecionado desta população, baseado na avaliação de *fitness* para gerar uma nova geração de soluções candidatas. O aspecto desejado nos pais está encapsulado em um código associado a cada um deles, chamado de **cromossomo**. Cada cromossomo é feito de um número de genes, carregando um aspecto desejado. Dois filhos são reproduzidos de dois pais pela divisão randômica de cada cromossomo relativo aos pais, em dois conjuntos de genes, e então estes são misturados de forma a se cruzarem. Este processo é chamado de *crossover*, que faz com que os bons aspectos dos pais sejam retransmitidos a próxima geração. Por fim, mutação é usada para evitar que o processo de procura fique preso a um mínimo local. Isto é feito modificando ocasionalmente algum gene.

A utilização do GA possui um amplo campo de aplicação, devido à essência de seu projeto, uma formalização e uma abstração de um mecanismo de adaptação natural

para computação de propósito geral, em oposição a um projeto de algoritmo para um problema específico.

A eficiência do GA em resolver um dado problema depende fortemente da representação de características desejadas no *cromossomo* [39]. As soluções encontradas pelo GA são descobertas baseadas em probabilidade; é traçado um máximo global (o algoritmo percorre todo o espaço de busca).

O GA é baseado na evolução, aplicando um método de busca massivamente paralelo, possibilitando a pesquisa por uma solução ideal em um grande número de soluções possíveis; enquanto o sistema evolui, utiliza a contínua mutação do critério de *fitness*, respondendo adequadamente a problemas com ambientes mutáveis; e ainda possui uma construção simples. Estas qualidades do GA fazem com que este algoritmo seja um candidato qualificado para ser empregado no uso de particionamento de tarefas.

Diferente dos algoritmos anteriores, SA e TS, o GA é implementado com um laço de controle sobre as interações, cujo pseudocódigo é apresentado na Figura 16.

O algoritmo GA começa inicializando uma população aleatória de partições (linhas 3 a 6), sendo esta população criada pela função *randomBigMove()*, mencionada anteriormente nos algoritmos SA e TS. Esta população aleatória é formada de acordo com o número de partições iniciais desejáveis, armazenado na variável *temperature* (nome mantido para manter relação com os demais algoritmos).

Dentro do laço interno (linhas 7 a 19) são aplicadas as principais operações do GA. Inicialmente, é executado a função *selection()* que elege e armazena a melhor partição da população de partições, ou seja, o que possui um menor custo - *actualCost()* (i.e. mesma função presente em SA e TS; não descrita no pseudocódigo da Figura 16).

```
1.   interaction ← Interaction input parameter
2.   temperature ← Temperature input parameter
3.   while(initialPopulationIsNotDone(temperature)) {
4.     if(randomBigMove() == false)
5.       continue
6.   }
7.   while(interaction > 0) {
8.     interaction—
9.     selection()
10.    crossover()
11.    mutation()
12.    checkPopulation()
13.    if(verifyPartition()) {
14.      if(costFunctionComparison()) {
15.        partitionIsNewBest()
16.        atLeastOnePartition = true
17.      }
18.    }
19.  }
```

Figura 16. Pseudocódigo do algoritmo GA.

Em seguida, a função *crossover()* seleciona a pior associação grupo-tarefas (i.e. em relação a função custo) dentro da partição eleita como a melhor, e realiza trocas de associações grupo-tarefas entre as partições da população. Estas trocas geram novas e únicas partições (i.e. filhos), representando o espaço de busca de soluções totais do algoritmo.

Logo após, é efetuado o *mutation()*. Esta função opera em cada uma das partições disponíveis no espaço de busca. Dentro de uma partição, é selecionado a tarefa que mais consome energia em uma determinada associação grupo-tarefas, e tenta realoca-la em outra associação grupo-tarefa da mesma partição, a fim de minimizar o consumo de energia.

Na função *checkPopulation()*, é selecionada a partição que possui o melhor valor referente a função *actualCost()* (i.e. melhor economia de energia e balanceamento de carga). Após, é feito a verificação desta partição quanto as restrições a serem atendidas (i.e. economia de energia e balanceamento de carga), através da função *verifyPartition()*. Caso esta partição selecionada atenda as restrições, a mesma é comparada com a melhor partição armazenada anteriormente. Se a nova melhor partição tiver um desempenho melhor (i.e. melhor valor para *actualCost()*, esta partição é então tomado como a mais nova melhor partição, através da função *partitionIsNewBest()*, e a variável *atLeastOnePartition* recebe um valor positivo.

Caso o algoritmo tenha atingido o número de interações pretendidas, a melhor partição é então retornada como resultado do particionamento, caso contrário, uma nova rodada de operações do GA são aplicadas.

4.4 Kernighan & Lin (KL)

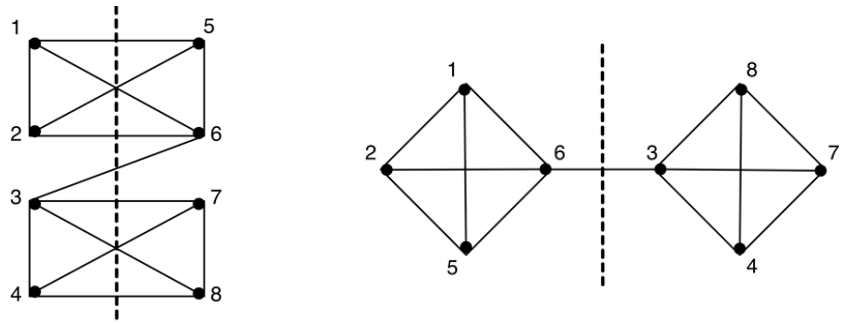
Kernighan & Lin (KL) [45] propôs um algoritmo de bissecção de grafos, o qual começa com uma partição inicial e iterativamente (i.e. realiza trocas iterativas entre pares, em todos os pares de nodos) a modifica para melhorar o ***cutsize*****,** até que nenhuma melhoria possa ser executada. O *cutsize* representa o número de redes conectadas a nodos em ambas as partições, e é o valor a ser minimizado. A essência da heurística aplicada é seu controle estratégico, o qual aplica um particionamento com aperfeiçoamento iterativo, superando vários mínimos locais sem movimentos excessivos.

O algoritmo KL foi melhorado em vários aspectos, por exemplo, Schweikert e Kernighan [46] propôs o uso de um modelo em malha a fim de manipular hipergrafos; e Fidducia e Mattheyses [47] reduziram a complexidade do tempo do algoritmo KL de $O(n^3)$ para $O(n \log n)$, através de modificações operacionais.

O algoritmo KL clássico começa particionando o grafo $G = (V, E)$ em dois subgrupos de tamanhos iguais. Pares de vértices são trocados através da bissecção, se a

troca melhorar o *cutsize*. O procedimento é realizado de forma iterativa até que nenhuma melhoria adicional possa ser obtida. Este algoritmo é do tipo migração de grupo, o qual utiliza métodos determinísticos que infelizmente, mas muitas vezes, leva a mínimos locais. O método é particularmente bem empregado para bissecção, pois divide o grafo em duas partes, mas pode ser generalizado para particionamento em partes desiguais, tornando-se a base de um esquema de particionamento hierárquico.

Para ajudar na compreensão da ideia básica do algoritmo KL, um exemplo prático e seu pseudocódigo são apresentados a seguir, extraídos de [48].



a. Bissecção Inicial b. Bissecção Final

Figura 17. Exemplo de grafo bi secionado pelo algoritmo KL [48].

Considerando o exemplo da Figura 17(a), as partições iniciais são:

$$A = \{ 1, 2, 3, 4 \}$$

$$B = \{ 5, 6, 7, 8 \}$$

Note que o *cutsize* inicial é 9 (baseado na bissecção inicial da Figura 17(a)). O próximo passo do algoritmo KL é escolher um par de vértices cuja troca resultaria em um maior decremento do *cutsize* ou resulte em um menor incremento, se nenhum decremento for possível. O decremento do *cutsize* é computado utilizando os valores de ganho de *D(i)* do vértice v_i. O ganho do vértice v_i é definido por

$$D(i) = INedge(i) - OUTedge(i)$$

onde *INedge(i)* é o número de arrestas do vértice *i* que não cruza as fronteiras da bissecção, e *OUTedge(i)* é o numero de arrestas que cruzam as fronteiras. O montante pelo qual o *cutsize* decrementa, se o vértice v_i mudar para a outra partição, é representado pelo *D(i)*. Se v_i e v_j são trocados, o decremento do *cutsize* é *D(i) + D(j)*. No exemplo ilustrado na Figura 17(a), um par de vértice adequado é o (3, 5), o qual decrementa o *cutsize* em 3. Uma tentativa de troca deste par é realizada. Estes dois vértices então são bloqueados. Este bloqueio nos vértices os proíbem de tomar parte em qualquer outra tentativa de troca. O procedimento acima é aplicado a uma nova partição, o qual resulta em um segundo par de vértices de (4, 6). Este procedimento é contínuo até que todos os vértices estejam bloqueados. Durante o processo, uma estrutura armazena

todas as tentativas de trocas e os resultados dos *cutsizes*. A Figura 18 ilustra as trocas realizadas e os *cutsizes* armazenados na estrutura.

i	***Par de vértices***	***g(i)***	$\sum_{j=1}^{i} g(i)$	***Cutsize***
0	-	-	-	9
1	(3, 5)	3	3	6
2	(4, 6)	5	8	1
3	(1, 7)	-6	2	7
4	(2, 8)	-2	0	9

Figura 18. Estrutura de registro das trocas de vértices [48].

Note que a soma parcial de decremento do *cutsize*, *g(i)*, sobre as trocas dos primeiros *i* pares de vértices é ilustrado na Figura 18, e.g. *g(1) = 3* e *g(2) = 8*. O valor de *k* para o qual *g(k)* fornece o valor máximo de todos os *g(i)* é determinado pela estrutura. Neste exemplo, *k = 2* e *g(2) = 8* é a soma parcial máxima. Os primeiros *k* pares de vértices são efetivamente trocados. No exemplo, os primeiros dois pares de vértices *(3, 5)* e *(4, 6)* são efetivamente trocados, resultando na bissecção ilustrada na Figura 17(b). Esta atividade completa uma interação e uma nova interação se inicia. Entretanto, se nenhum decremento no *cutsize* for possível durante uma interação, o algoritmo para. A Figura 19 apresenta o pseudocódigo do algoritmo KL.

```
1.   INITIALIZE()
2.   while( IMPROVE(table) = TRUE ) {
3.      (* se alguma melhoria for feita durante a última interação,
4.      o processo é levado a diante novamente. *)
5.      while( UNLOCK(A) = TRUE ) {
6.         (* se existir algum vértice desbloqueado em A,
7.         mais tenttivas de trocas são levadas a diante. *)
8.         for ( each a ∈ A ) {
9.            if(a = unlocked) {
10.              for ( each b ∈ B ) {
11.                 if(b = unlocked) {
12.                    if(D_max < D(a) + D(b)) {
13.                           D_max = D(a) + D(b)
14.                           a_max = a
15.                           b_max = b
16.                    }
17.                 }
18.              }
19.           }
20.        }
21.        TENT-EXCHANGE(a_max ,b_max)
22.        LOCK(a_max ,b_max)
23.        LOG(table)
24.        D_max = - ∞
25.     }
26.     ACTUAL-EXCHANGE(table)
27.  }
```

Figura 19. Pseudocódigo do algoritmo KL [48].

A função INITIALIZE gera as bissecções iniciais, e inicializa os parâmetros do algoritmo. A função IMPROVE verifica se nenhuma melhoria foi feita durante a última interação, enquanto que a função UNLOCK verifica se algum vértice está desbloqueado. Cada vértice tem o estado de bloqueado (*locked*) ou desbloqueado (*unlocked*). Somente os vértices cujo estado se encontra em desbloqueado são candidatos para a próxima

tentativa de troca. A função TENT-EXCHANGE tentativamente troca pares de vértices. A função LOCK bloqueia o par de vértices, enquanto que a função LOG armazena as trocas de vértices e resultados na estrutura de armazenamento. A função ACTUAL-EXCHANGE determina a soma parcial máxima de *g(i)*, seleciona os pares de vértices para serem trocados e executa a troca efetiva destes pares de vértices.

O algoritmo KL é bem robusto, entretanto, apresenta algumas desvantagens. Por exemplo, o algoritmo não é aplicável a hipergrafos, ou seja, não suporta grafos arbitrariamente ponderados, e o tamanho das partições devem ser especificados antes do particionamento. Desta forma, o algoritmo KL não é totalmente compatível com o particionamento de tarefas tratado neste trabalho, porque o particionamento de tarefas necessita associar uma tarefa ou um grupo de tarefas a um dado tipo de processador (i.e. MPSoC heterogêneo). Ainda, grupos de tarefas são limitados à quantidade de processadores; e o custo de cada grupo de tarefas associado a um processador depende do tipo do processador. Consequentemente, foi implementado um algoritmo modificado do KL [49], chamado *BIA – Bisection Algorithm*, baseado na ideia do KL clássico, mas com várias melhorias, como por exemplo, aquelas descritas em [46][47]. O algoritmo *BIA* está descrito na Seção 4.5.

4.5 Bisection Algorithm (BIA)

Como comentado anteriormente, o algoritmo KL clássico não é totalmente compatível com o particionamento de tarefas utilizado neste trabalho. Para que a ideia principal deste algoritmo fosse utilizada, melhorias e modificações permitindo que este algoritmo lide com particionamento de tarefas em um MPSoC heterogêneo [49][9], gerando o algoritmo BIA.

A Figura 20 apresenta o pseudocódigo do BIA, que foi implementado de acordo com duas abordagens, (i) uma forma de bissecção em profundidade e (ii) uma forma de bissecção em largura.

```
1.   interaction ← Interaction input parameter
2.   while(interaction > 0) {
3.     interaction--
4.     while(!reachConstraints() || !reachBipartitionLimit()) {
5.       if(isDepthBipartition)
6.         depthBipartition()
7.       else
8.         widthBipartition()
9.     }
10.    if(minimumPartitionCost > actualPartitionCost) {
11.      minimumPartitionCost = actualPartitionCost
12.      saveActualPartitionAsMinimum()
13.    }
14.  }
```

Figura 20. Pseudocódigo do algoritmo BIA.

O laço mais externo do BIA (linhas 2 a 14) é responsável por ampliar o espaço de busca, uma vez que cada laço começa uma nova e aleatória partição inicial, o qual é controlado pelo parâmetro *interaction*. O laço mais interno (linhas 4 a 9) realiza o particionamento em profundidade através do algoritmo *depthBipartition* (BIA-depth), ou o particionamento em largura através do algoritmo *widthBipartition* (BIA-width), de acordo com o parâmetro de entrada Booleano *isDepthBipartition*.

A Figura 21 apresenta um diagrama de fluxo simplificado sobre as operações fundamentais do algoritmo BIA (i.e. operações comuns a ambas as abordagens, *depthBipartition* e *widthBipartition*), a bissecção de partições e a troca de tarefas entre grupos de tarefas, a fim de atender as restrições estipuladas.

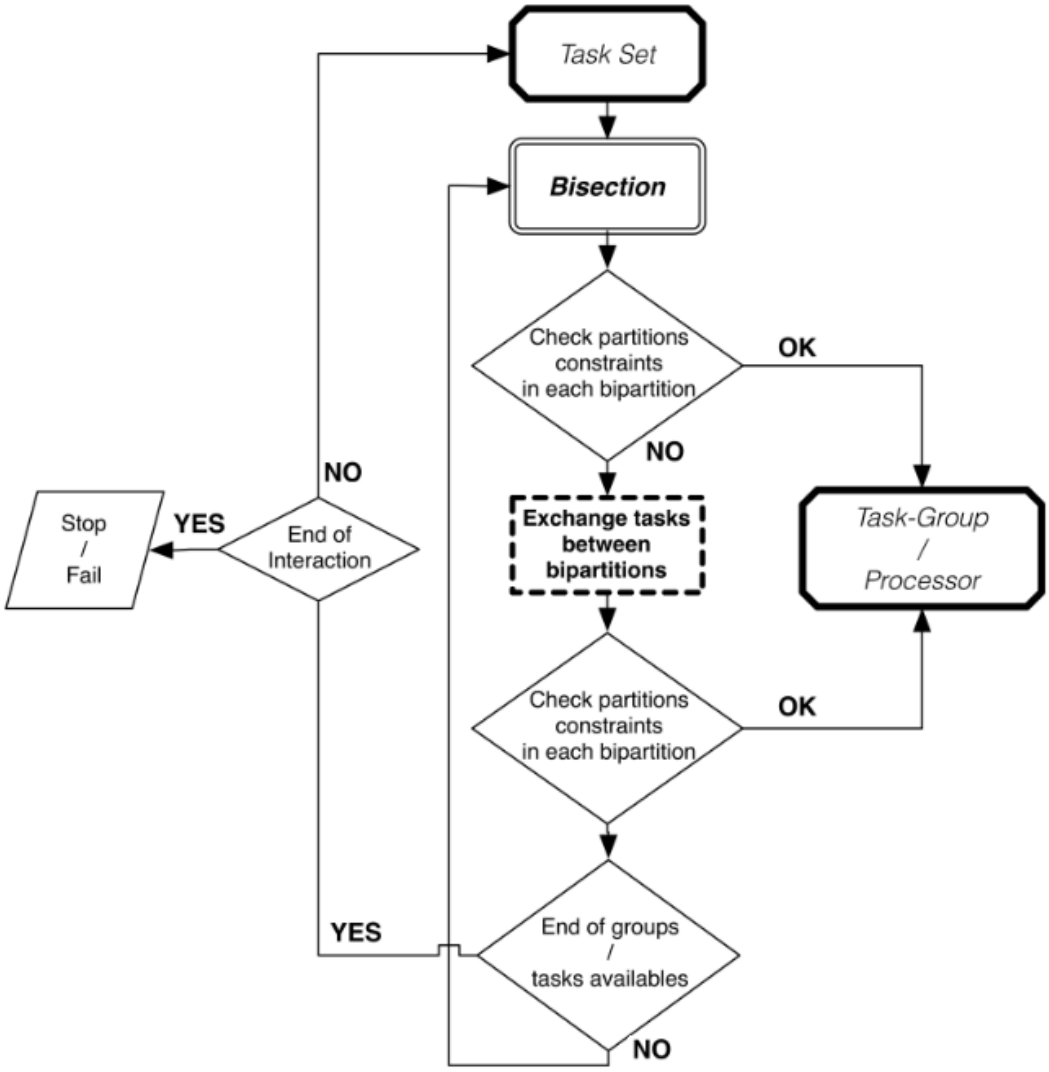
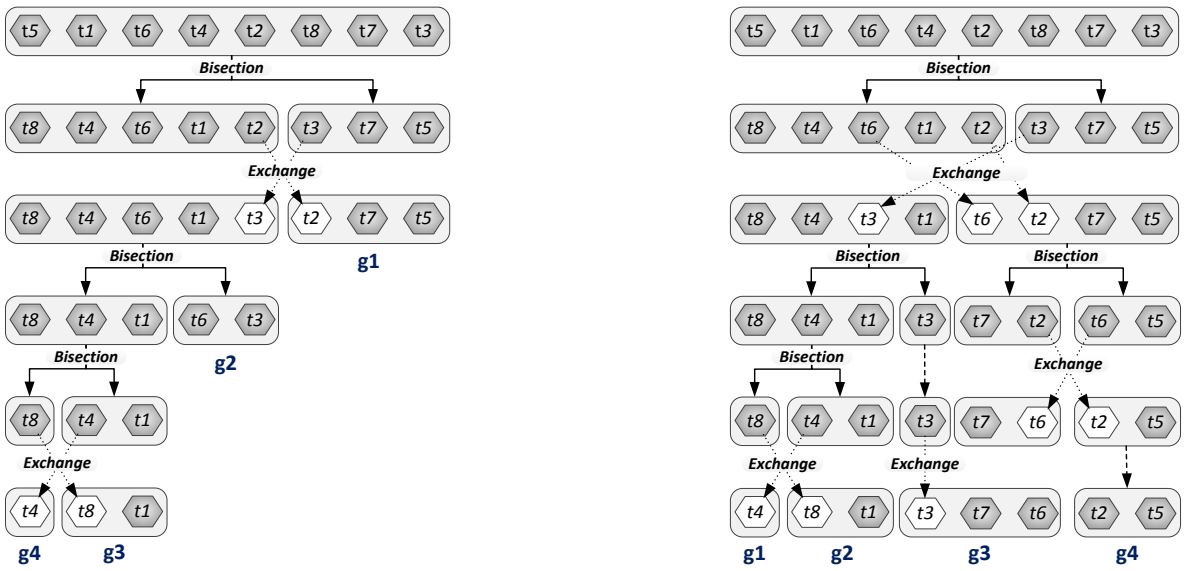


Figura 21. Diagrama de fluxo das principais operações do algoritmo BIA.

A Figura 22 exemplifica o processo de ambas as abordagens algorítmicas.



a. BIA-depth b. BIA-width

Figura 22. Exemplo de particionamento de grupo de tarefas com as abordagens BIA-depth e BIA-width. A aplicação sintética é composta por 8 tarefas, e tem como alvo um MPSoC heterogêneo com 5 processadores. Observar que o número de processadores limita o número de grupos criados pela bissecção. Entretanto, o número de grupos pode ser menor que o número de processadores, o qual é o caso deste exemplo.

O algoritmo *depthBipartition* começa realizando uma bissecção, tentando encontrar um algo grupo-tarefa que atenda as restrições. Como consequência, o sistema é dividido em dois grupos, o grupo alvo em que o algoritmo está tentando otimizar, e o grupo que contém as tarefas remanescentes. Por isso, o algoritmo realiza trocas de tarefas entre grupos-tarefa, com o objetivo de minimizar a função de custo do alvo grupo-tarefa; e a quantidade de trocas é previamente definida por um parâmetro de entrada. A troca de tarefas pode ser realizada de duas maneiras: (i) um par de tarefas é trocado entre um par de grupos de tarefas; e (ii) uma tarefa migra de um grupo de tarefas para outro. Quando o procedimento de troca tiver acabado, o grupo-tarefas permanece inalterado enquanto o algoritmo não chegar ao fim do laço interno. Se a condição do laço interno (linha 4) é satisfeita, o algoritmo reinicia uma nova sequência de bissecções e trocas de tarefas, mas agora tomando como entrada os grupo-tarefas remanescentes fornecidos pela última bipartição. Como o grupo de tarefas remanescente é sempre menor que o anterior, a cada interação realizada, o problema se torna cada vez menos complexo e menos demorado.

A outra abordagem algorítmica do BIA, o algoritmo *widthBipartition* começa realizando uma bissecção tentando encontrar grupos-tarefa que igualmente preencham as restrições. Assim, todos os grupos têm a mesma prioridade de otimização. Por isso, o algoritmo realiza trocas de tarefas entre grupos-tarefa tendo como alvo minimizar a função custo de todos os grupos-tarefa; e a quantidade de trocas é previamente definida por um parâmetro de entrada, o qual é tipicamente menor que o utilizado no algoritmo

depthBipartition. A troca de tarefas pode ser realizada do mesmo modo como o algoritmo *depthBipartition* realiza. Quando o procedimento de troca termina, a condição do laço mais interno (linha 4) é verificada; se satisfeita, o algoritmo *widthBipartition* recomeça aplicando bissecções em todos os grupos, até alcançar a quantidade de processadores da arquitetura alvo. Ainda, a troca é realizada entre todos os grupos-tarefa. Como consequência, todas as iterações são similarmente complexas e demoradas.

Independente da abordagem algorítmica de bipartição utilizada, o laço mais interno para quando todas as restrições (e.g. limite de carga de trabalho do processor e consumo máximo de energia) são satisfeitas por todas as associações de grupos-tarefa com seu tipo de processador correspondente, ou quando o algoritmo de bipartição alcançar o limite (i.e. não pode executar mais bissecções, já que não há mais processadores para associar um novo grupo-tarefa e uma rodada de trocas já foi realizada). Estas verificações são realizadas pelas funções *reachConstraints()* e *reachBipartitionLimit()*, respectivamente.

Por fim, cada vez que o laço mais interno termina, o custo da partição alcançada é comparado com os custos anteriores a fim de armazenar a melhor partição já alcançada pelo algoritmo em determinado processo.

O algoritmo BIA, como os outros algoritmos apresentados (SA, TS e GA), serão avaliados e comparados no Capítulo 7, aplicados ao particionamento de tarefas de aplicações paralelas em MPSoCs heterogêneos.

5 FRAMEWORK PALOMA

Este Capítulo apresenta o *framework* utilizado neste trabalho, chamado PALOMA, do inglês, *Partitioning Algorithm for MPSoC Automated Design*. Este *framework* automatiza a geração de aplicações sintéticas, implementa os algoritmos clássicos e propostos de particionamento, e através dos mesmos, realiza a atividade de particionamento (i.e. Pré-mapeamento) de tarefas de uma aplicação. O *framework* foi projetado para desenvolver três papéis: (i) poder descrever classes de aplicações, que embora sintéticas, pudessem ter comportamento similar ao de aplicações reais (i.e. para tanto, boa parte dos parâmetros são expressos em termos de distribuições Gaussianas); (ii) realizar testes de desempenho sobre diferentes abordagens algorítmicas empregadas no particionamento de tarefas; e (iii) possibilitar a análise de desempenho do uso da técnica de pré-mapeamento e mapeamento, versus o mapeamento direto. O processo de mapeamento é feito com o uso do *framework CAFES,* desenvolvido em [50], e não entra em análise neste trabalho.

Originalmente, este *framework* foi concebido no trabalho de [6], e possuía apenas suporte a arquiteturas homogêneas, com o uso do SA como algoritmo de particionamento. Neste trabalho, o *framework* recebeu a capacidade de trabalhar com arquiteturas MPSoCs heterogêneas, e sofreu o incremento de novas abordagens algorítmicas para realizar o particionamento de tarefas, e.g. (SA – heterogêneo, TS, GA, e BIA).

O PALOMA tem como entradas, (i) a descrição da NoC, (ii) a descrição da arquitetura, (iii) a caracterização da aplicação, (iv) a descrição da aplicação, e (v) a escolha do algoritmo para a realização do particionamento de tarefas. Feito o processamento dos dados, o PALOMA tem como saída, (a) aplicação particionada, (b) resultados do particionamento, e (c) desempenho do particionamento. A Figura 23 ilustra o funcionamento do *framework PALOMA.*

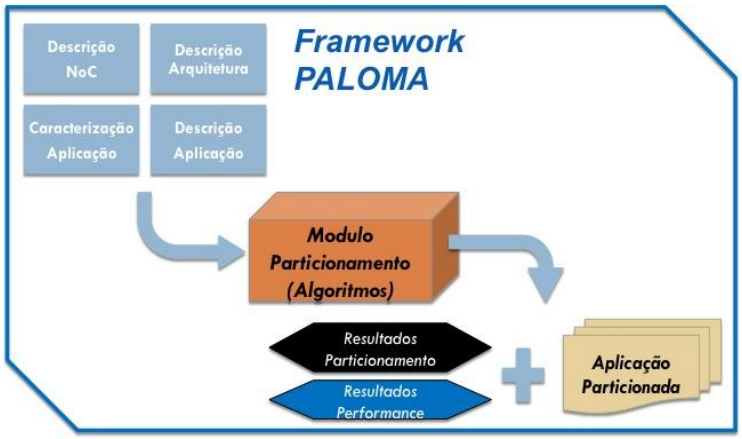


Figura 23. Representação do funcionamento do *Framework PALOMA*.

A Figura 24 apresenta a interface gráfica reformulada, tendo seu conjunto de ferramentas realocado e redefinido, com a adição de várias melhorias, como por exemplo,

a tela de acompanhamento dos procedimentos (campo *log*), e a organização distribuída das configurações da arquitetura e aplicação.

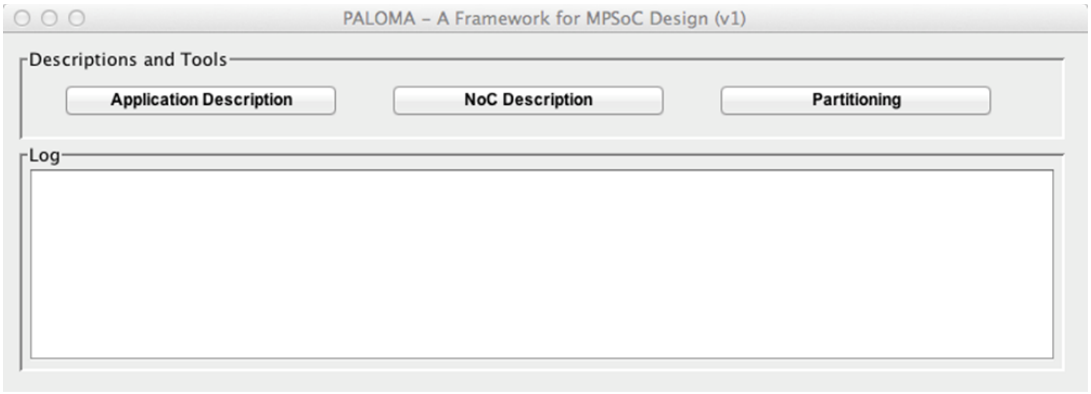


Figura 24. *Framework* PALOMA: Tela inicial da interface gráfica.

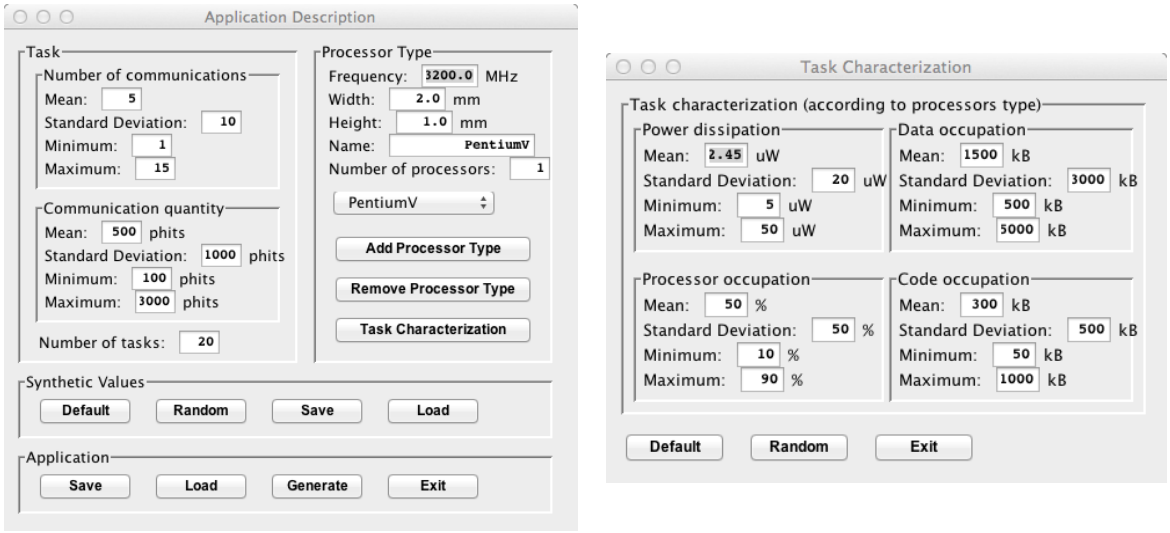
A descrição da aplicação é realizada na janela *Application Description*, exemplificada na Figura 25(a). A aplicação é definida por dois grupos:

- *Task*: Este campo configura as tarefas da aplicação, que são definidas pelos parâmetros: (i) *Task communication* e *Communication quantity*, que contém informações sobre a comunicação entre as tarefas da aplicação; (ii) *Number of tasks*, que contém o número total de tarefas da aplicação; e (ii) *Task characterization*, uma subjanela contendo quatro campos que detalham caracterizações das tarefas quando executadas na arquitetura alvo;
- *Processor Type*: Este campo possibilita configurar, adicionar e remover processadores de diversas naturezas. As características configuráveis são: *Frequency* - frequência de operação (em MHz), *Width* - largura física do processador (em mm), *Height* - altura física do processador (em mm) do processador, *Name* - nome do tipo de processador e *Number of processors* – número de processadores do tipo selecionado.

Os campos *Power dissipation* e *Processor occupation* da subjanela *Task characterization* (Figura 25(b)) descrevem o consumo de energia e o percentual de ocupação de uma tarefa quando executada no processador especificado. Os campos *Data occupation* e *Code occupation*, que contém informações sobre a quantidade de dados e código de uma dada tarefa respectivamente, são obtidos após a compilação de cada tarefa para o processador especificado.

Campos da janela *Application Description* e da subjanela *Task Characterization* contendo média, desvio padrão e intervalo (valor máximo e mínimo) produzem valores aleatórios para a aplicação com probabilidade que respeita uma distribuição Gaussiana. Uma distribuição Gaussiana permite que a aplicação sintética gerada possa ser direcionada para uma determinada classe, tal como *dataflow*, IO-*bounded* ou CPU-*bounded*. O objetivo aqui é que ao analisar uma aplicação, mesmo que sintética, se possa

ter ideia de como o particionamento de uma aplicação real se comportaria, uma vez que esta faz parte da mesma classe de aplicação.



(a) (b)

Figura 25. *Framework* PALOMA: (a) Janela *Application Description*; (b) Subjanela *Task Characterization.*

A descrição da infraestrutura de comunicação NoC é realizada na janela *NoC Description*, exemplificada na Figura 26. A NoC possui os seguintes parâmetros: (i) *topology* - define o tipo de topologia utilizada pela NoC, disponível no momento o tipo *mesh*; (ii) NoC – linhas e colunas que compõem a infraestrutura; (iii) *Tile* – tamanho do *tile* utilizado; (iv) *Router* – define tamanho de *buffer* utilizado nos roteadores da NoC; (v) *Energy Parameters* – define características relativas a energia da NoC; (vi) *Timing Parameters* – configurações referentes a frequência e tempos de operação.

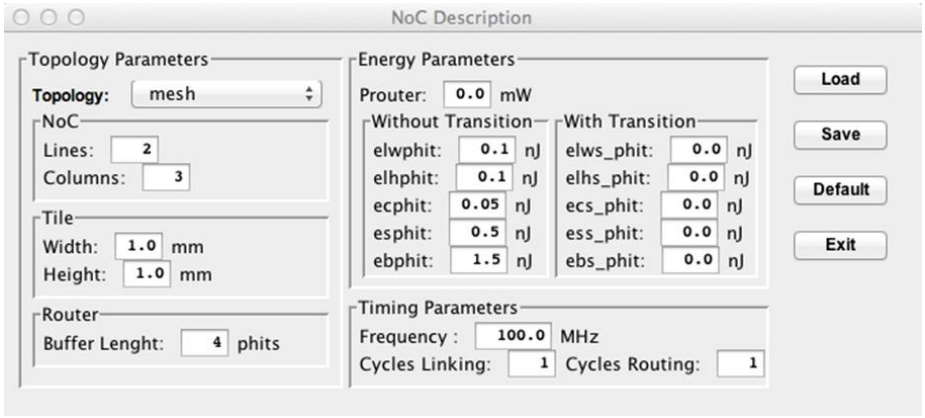


Figura 26. *Framework* PALOMA: Janela *NoC Description.*

A configuração do particionamento de tarefas é realizada na janela *Partitioning*, mostrado na Figura 27. O particionamento possui os seguintes parâmetros configuráveis: (i) *Requirements* – configura os requisitos que se quer atender no cenário alvo, e.g.

redução de energia, balanceamento de carga, ou ambos; (ii) *Constraits* - restrições de particionamento, como energia (em micro Joules), balanceamento de carga (em porcentagem), tamanho dos dados (em Kb) e tamanho de código (em Kb); (iii) *Algorithm* – setor de configuração do algoritmo desejado para realizar o particionamento (e.g. SA, TS, GA e BIA); é configurado os parâmetros *temperature*, *interaction*, e outras opções pertinentes a informações resultantes do particionamento.

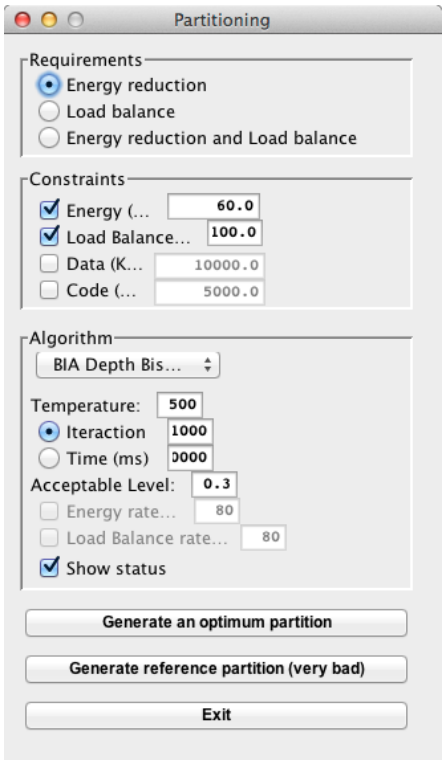


Figura 27. *Framework* PALOMA: Janela *Partitioning*.

Para gerar uma aplicação sintética e realizar o particionamento de tarefas, alguns passos devem ser respeitados. Primeiro, deve ser realizado o preenchimento dos campos dispostos na janela *Application Description*, para assim gerar uma aplicação sintética - através do botão *Generate*. Este procedimento gera um arquivo de texto no formato XML com a descrição da aplicação desejada (exemplificado no Capítulo 5.1). Esta descrição XML pode ser salva em um arquivo desejado – botão *Save*, ou carregado de um arquivo pré-montado – botão *Load*. Os campos desta janela podem ser preenchidos automaticamente com valores padrões - pressionando o botão *Default*, realizar o carregamento de uma aplicação sintética salva (opção de salvar a descrição da aplicação também esta disponível), ou então gerar esses valores aleatoriamente - através do botão *Random*. A janela *Task Characterization* possui as opções de preencher os campos com valores padrões – através do botão *Default*, ou o preenchimento com valores aleatórios – através do botão *Random*.

Uma vez tendo a aplicação sintética, o segundo passo seria configurar os parâmetros da janela *NoC Description*. Esta janela possibilita configurar a NoC utilizada

no particionamento. As configurações da NoC podem ser estipuladas manualmente, através do carregamento de um arquivo de descrição NoC salvo – botão *Load*, ou carregar os parâmetros com valores padrões – botão *Default.*

Com a aplicação sintética e as configurações da NoC, é possível realizar o particionamento através da janela *Partitioning*. Configuram-se os parâmetros de particionamento, preenchendo os requisitos e restrições do particionamento, e escolhendo o algoritmo de particionamento desejado, e.g. SA, TS, GA ou BIA (BIA-width ou BIA-depth), com suas determinadas configurações. O PALOMA pode particionar a aplicação sintética objetivando minimizar ou maximizar a função custo de particionamento. Ou seja, objetivando obter particionamentos ótimos (aqueles com menor custo de particionamento) – através do botão *Generate an optimum partition*, ou péssimos (aqueles com maior custo de particionamento) – através botão *Generate reference partition (very bad*). A operação de particionamento tem como resultado dois arquivos texto XML, um com extensão PAR e outro com extensão CWG, portando as informações, aplicação particionada, resultados do particionamento, e desempenho do particionamento.

5.1 Descrição de Arquivos de Entrada do PALOMA

O *framework* PALOMA recebe como entrada uma aplicação descrita em XML (em inglês, *Extensible Markup Language*). Os principais propósitos de usar XML são: (i) interoperabilidade de dados por diferentes ferramentas de projeto; (ii) a facilidade de documentar a aplicação; (iii) são menos restritivos do que formatos de documentos proprietários [51]; e (iv) a extensibilidade da linguagem, uma vez que esta permite que sejam adicionados modificações ao arquivo de texto, como *tags*, sem prejudicar a estrutura dos dados já especificados. Desta maneira, outras ferramentas podem compartilhar o mesmo arquivo XML, tornando um formato de descrição eficiente e robusto.

Na descrição da aplicação existem características que devem ser levadas em consideração para obtenção de resultados desejados, enquanto que outras devem ser relevadas. Exemplos destas características são o tamanho do *tile* que é irrelevante para o cálculo do tempo de execução, enquanto que o tempo de execução de uma tarefa em certo tipo de processador é imprescindível para realizar o correto mapeamento da tarefa.

Detalhando a descrição para a atividade de particionamento, a aplicação e a arquitetura alvo são descritas através de três *tags* XML primárias, ilustradas na Figura 28: TARGET_ARCHITECTURE, APPLICATION_CHARACTERIZATION, APPLICATION_DESCRIPTION. O campo TARGET_ARCHITECTURE contém o campo interno PROCESSOR_TYPE_LIST, que lista os tipos de processadores e suas características, como tipo do processador, características e lista de códigos relativos ao número de processadores daquele tipo, inseridos na arquitetura.

```
<SYSTEM_SPECIFICATION>

  <TARGET_ARCHITECTURE> ... </TARGET_ARCHITECTURE>

  <APPLICATION_CHARACTERIZATION> ... </APPLICATION_CHARACTERIZATION>

  <APPLICATION_DESCRIPTION> ... </APPLICATION_DESCRIPTION>

</SYSTEM_SPECIFICATION>
```

Figura 28. *Framework* PALOMA: Padrão de especificação da plataforma MPSoC e aplicação em formato XML.

A Figura 29 apresenta um exemplo de descrição do campo PROCESSOR_TYPE_LIST, onde está contida uma lista com três tipos de processadores.

```
<PROCESSOR_TYPE_LIST>

  <PROCESSOR_TYPE type="PentiumV">
    <FEATURES frequency="3200.0" width="2.0" height="1.0" />
    <LIST>P0</LIST>
  </PROCESSOR_TYPE>

  <PROCESSOR_TYPE type="PowerPC">
    <FEATURES frequency="2500.0" width="1.5" height="1.5" />
    <LIST>P1 P2</LIST>
  </PROCESSOR_TYPE>

  <PROCESSOR_TYPE type="MIPS">
    <FEATURES frequency="1500.0" width="0.5" height="0.7" />
    <LIST>P5 P4 P3</LIST>
  </PROCESSOR_TYPE>

</PROCESSOR_TYPE_LIST>
```

Figura 29. Descrição exemplo de campos da tag PROCESSOR_TYPE_LIST, contendo uma lista de tipos de processadores com suas características físicas e uma lista de identificadores (nomes) de processadores de cada tipo incluídos na arquitetura. A figura ilustra três tipos de processadores contendo a descrição de suas frequências de operação e suas dimensões. Por exemplo, no caso do processador PowerPC, este opera a 2.5GHz, tem como largura 1,5mm e altura 1,5mm, e apresenta dois processadores do tipo PowerPC incluídos na arquitetura, com os nomes P1 e P2.

O segundo campo, APPLICATION_CHARACTERIZATION (Figura 30), possui as caracterizações das tarefas da aplicação descrita para os tipos de processadores disponíveis na arquitetura alvo. As tarefas são caracterizadas em termos de: (i) tipo de processador que irá executar a tarefa; (ii) potência média dissipada pelo processador, quando este executa a tarefa (*power*); (iii) áreas de dados (*data*) e de código (*code*), obtidas pela compilação da tarefa no sistema operacional que estará executando no processador; e (iv) percentual de processamento requerido pela tarefa (*cpuOccupation*). A Figura 30 ilustra esta descrição de tarefas.

```
<APPLICATION_CHARACTERIZATION>

  <TASK_LIST>

    <TASK id="T1">
      <PROCESSOR_TYPE type="MIPS" power="26.74" data="3494" code="105" cpuOccupation="51.41" />
      <PROCESSOR_TYPE type="PowerPC" power="19.23" data="2172" code="537" cpuOccupation="11.87" />
      <PROCESSOR_TYPE type="PentiumV" power="31.81" data="2274" code="107" cpuOccupation="63.9" />
    </TASK>

    <TASK id="T2">
      <PROCESSOR_TYPE type="MIPS" power="8.61" data="521" code="712" cpuOccupation="31.95" />
      <PROCESSOR_TYPE type="PowerPC" power="20.29" data="1331" code="893" cpuOccupation="60.14" />
      <PROCESSOR_TYPE type="PentiumV" power="20.87" data="665" code="184" cpuOccupation="51.26" />
    </TASK>

    <TASK id="T3">
      <PROCESSOR_TYPE type="MIPS" power="27.59" data="578" code="526" cpuOccupation="72.13" />
      <PROCESSOR_TYPE type="PowerPC" power="20.93" data="1196" code="566" cpuOccupation="50.86" />
      <PROCESSOR_TYPE type="PentiumV" power="25.23" data="2666" code="481" cpuOccupation="11.39" />
    </TASK>

  </TASK_LIST>

</APPLICATION_CHARACTERIZATION>
```

Figura 30. Caracterização de tarefas da aplicação frente aos tipos de processador disponíveis. A figura ilustra um exemplo sintético de caracterização de três tarefas (T1, T2, T3) em três processadores distintos (MIPS, PowerPC e PentiumV). Exemplificando, a tarefa T1 quando executada no processador MIPS dissipa 26.74 uW (micro watts) de potência, ocupa 3494 KB (kilobytes) de área de dados e 105 KB de área de código, e requer 51.41% de processamento. Esta mesma tarefa executada em um processador PowerPC dissipa 19.23 uW de potência, ocupa 2172 KB de área de dados e 537 KB de área de código, requerendo 11.87% de processamento. A tarefa T1, executando em um processador PentiumV dissipa 31.81 uW de potência, ocupa 2274 KB de área de dados e 107 KB de área de código, requerendo 63.9% de processamento.

O terceiro campo, APPLICATION_DESCRIPTION (Figura 31), descreve o volume de comunicação entre as tarefas da aplicação (descrito em *kilobytes*), assim como o fluxo da comunicação (i.e. qual tarefa origina a comunicação, e qual recebe). Esta descrição é exemplificada na Figura 31.

```
<APPLICATION_DESCRIPTION>

  <COMMUNICATION_TASK_LIST>

    <SOURCE_TASK source="T1">
      <COMMUNICATION target="T2" volume="915" />
      <COMMUNICATION target="T3" volume="318" />
    </SOURCE_TASK>

    <SOURCE_TASK source="T2">
      <COMMUNICATION target="T3" volume="540" />
      <COMMUNICATION target="T1" volume="230" />
    </SOURCE_TASK>

    <SOURCE_TASK source="T3">
      <COMMUNICATION target="T1" volume="774" />
      <COMMUNICATION target="T2" volume="825" />
    </SOURCE_TASK>

  </COMMUNICATION_TASK_LIST>

</APPLICATION_DESCRIPTION>
```

Figura 31. Descrição da aplicação em relação à intercomunicação de suas tarefas. Esta figura descreve um exemplo sintético contendo três tarefas (T1, T2, T3) que originam e recebem a comunicação. Por exemplo, existe uma comunicação que parte da tarefa T1 em direção à tarefa T2 com o volume de comunicação igual a 915 kB.

6 METODOLOGIA APLICADA

Este capítulo apresenta (i) a metodologia utilizada para realizar o particionamento de tarefas de uma aplicação (i.e. pré-mapeamento), e o mapeamento desta aplicação em uma arquitetura MPSoC (mapeamento realizado através do *framework CAFES* [50]); e (ii) uma exemplificação do pré-mapeamento (aplicação da técnica de particionamento) ao mapeamento da aplicação em um MPSoC.

6.1 Descrição da metodologia

A Figura 32 ilustra a metodologia utilizada através de um fluxo de projeto.

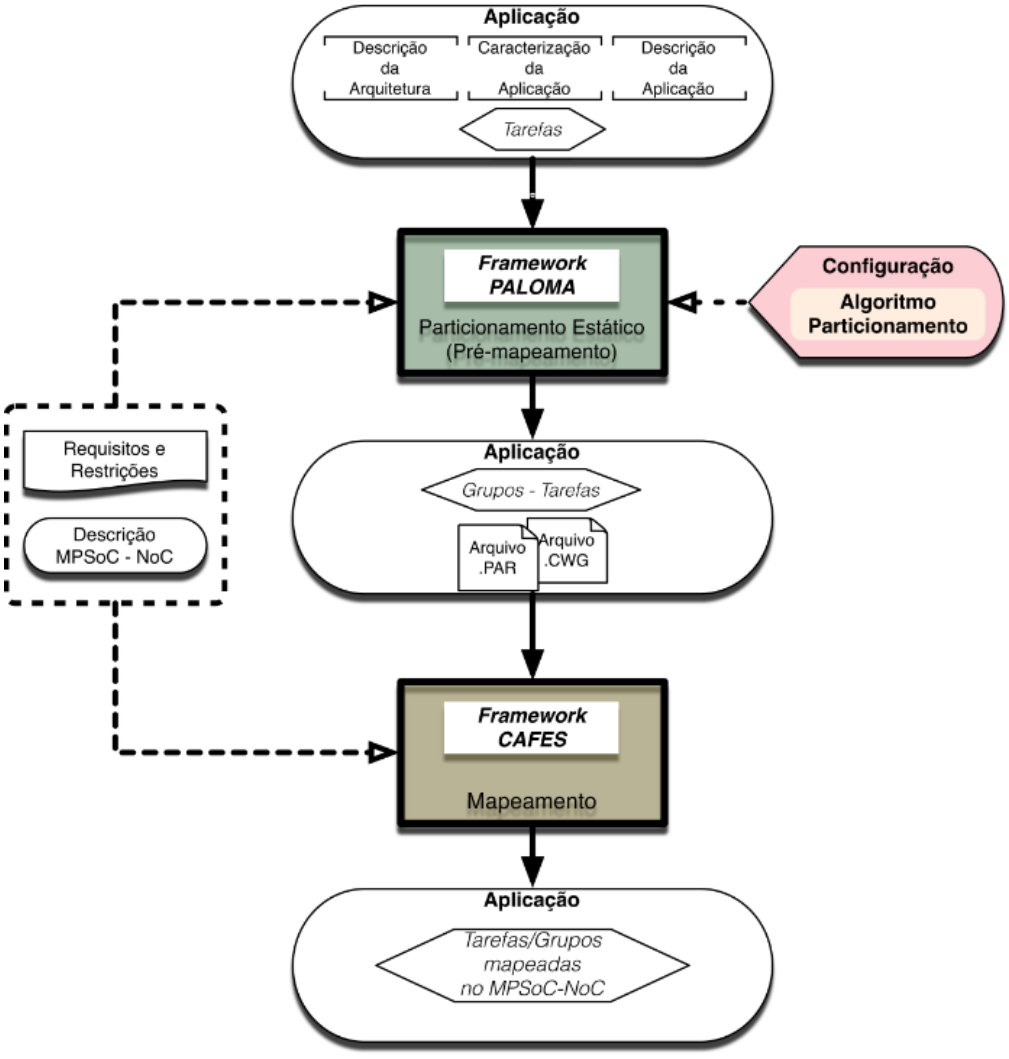


Figura 32. Fluxo de projeto ilustrando as atividades de particionamento e mapeamento.

Para este fluxo, o particionamento de tarefas em processadores heterogêneos tem como entrada: (i) descrição da arquitetura, onde consta uma lista de processadores diversos, descrevendo o tipo, características específicas e o número de todos os processadores daquele tipo, permitindo calcular quantos grupos terá uma partição; (ii) caracterização da aplicação, onde apresenta as tarefas com suas características distintas

relacionadas a cada tipo de processador, como energia consumida (utilizado para calcular o consumo de energia em cada partição na NoC), área de dados e código (em Kb), e ocupação da CPU, que é uma restrição para evitar sobrecarga de processamento, limitando o número de tarefas agrupadas em um mesmo processador (i.e. balanceamento de carga); (iii) descrição da aplicação, descrevendo o fluxo de comunicação entre as tarefas com suas características; (iv) descrição da NoC, contendo suas características; e (v) a escolha e configuração do algoritmo de particionamento a ser utilizado, e.g. SA, TS, GA e BIA.

A função custo do particionamento leva em conta a minimização do volume de comunicação global e o modo como as tarefas são agrupadas. Os algoritmos empregados tentam alcançar um custo mínimo de energia, o que implica agrupar tarefas altamente comunicantes em um mesmo processador, respeitando a heterogeneidade. Além disso, os algoritmos tentam equilibrar a ocupação da CPU através da distribuição de tarefas ao longo dos processadores disponíveis (o consumo de processamento do sistema operacional não é levado em consideração, mas pode ser simulado, estipulando uma margem de ocupação dos processadores). Deste modo, as tarefas mais comunicantes são agrupadas em um processador, respeitando o limite de processamento estipulado. A restrição de minimização de energia é ignorada apenas em casos de falta de processadores aptos, ou seja, processadores que estejam disponíveis no momento (i.e. com carga da CPU menor do que a máxima estipulada) para receber a tarefa, e neste caso a distribuição de tarefas é feita de forma balanceada.

A atividade de particionamento tem como saída: (i) um arquivo texto XML com extensão PAR descrevendo as tarefas agrupadas em cada processador, o tipo do processador, energia a ser consumida, área de dados e código, ocupação da CPU e o custo do particionamento final; (ii) resultados de desempenho relativos ao processamento do particionamento de tarefas (e.g. tempo consumido, memória RAM consumida), e (ii) um arquivo texto XML com extensão CWG representando um grafo, contendo o volume de comunicação entre os processadores nos quais as tarefas foram agrupadas e descrição da NoC utilizada.

O *framework CAFES* tem como entrada: (i) os mesmos parâmetros de energia da NoC descritos na atividade de particionamento; (ii) o arquivo CWG que descreve a aplicação em termos de volume de comunicação entre processadores, além da descrição da NoC e aspectos de energia, tanto da NoC como dos processadores distintos. Por ter um comportamento de auto-similaridade similar ao particionamento, o mapeamento realizado pelo *CAFES* também utiliza o algoritmo *simulated annealing*.

A saída do mapeamento (*Aplicação mapeada*) é um arquivo contendo todas as associações de processador – *tile* com os grupos de tarefas gerados pelo particionamento.

6.2 Exemplificação da metodologia

Esta Seção exemplifica a metodologia aplicada no *framework* PALOMA para o particionamento de tarefas, e através do *framework CAFES*, o mapeamento deste particionamento resultante na arquitetura MPSoC.

A Figura 33 apresenta um exemplo de entrada do tipo XML, com a descrição da arquitetura alvo composta pelos processadores: P0 (ARM11), P1 (PentiumV) e P2 (PowerPC), e a caracterização e descrição de uma aplicação paralela sintética composta por 6 tarefas (T0 a T5).

```
<SYSTEM_SPECIFICATION>
 <TARGET_ARCHITECTURE>
  <PROCESSOR_TYPE_LIST>
   <PROCESSOR_TYPE type="ARM11">
    <FEATURES freq="1800.0" width="0.5" height="0.7" />
    <LIST>P0</LIST>
   </PROCESSOR_TYPE>
   <PROCESSOR_TYPE type="PentiumV">
    <FEATURES freq="3200.0" width="2.0" height="1.0" />
    <LIST>P1</LIST>
   </PROCESSOR_TYPE>
   <PROCESSOR_TYPE type="PowerPC">
    <FEATURES freq="2500.0" width="1.5" height="1.5" />
    <LIST>P2</LIST>
   </PROCESSOR_TYPE>
  </PROCESSOR_TYPE_LIST>
 </TARGET_ARCHITECTURE>
<APPLICATION_DESCRIPTION>
  <COMMUNICATION_TASK_LIST>
   <SOURCE_TASK source="T4">
    <COMMUNICATION target="T1" volume="941" />
    <COMMUNICATION target="T3" volume="859" />
    <COMMUNICATION target="T2" volume="1371" />
    <COMMUNICATION target="T0" volume="631" />
    <COMMUNICATION target="T5" volume="724" />
   </SOURCE_TASK>
   <SOURCE_TASK source="T5">
    <COMMUNICATION target="T1" volume="914" />
    <COMMUNICATION target="T3" volume="1385" />
    <COMMUNICATION target="T4" volume="554" />
    <COMMUNICATION target="T2" volume="1382" />
    <COMMUNICATION target="T0" volume="1602" />
   </SOURCE_TASK>
   <SOURCE_TASK source="T1">
    <COMMUNICATION target="T3" volume="1207" />
    <COMMUNICATION target="T4" volume="1455" />
    <COMMUNICATION target="T2" volume="1994" />
    <COMMUNICATION target="T0" volume="459" />
   </SOURCE_TASK>
   <SOURCE_TASK source="T0">
    <COMMUNICATION target="T5" volume="510" />
   </SOURCE_TASK>
   <SOURCE_TASK source="T3">
    <COMMUNICATION target="T1" volume="1018" />
    <COMMUNICATION target="T4" volume="759" />
    <COMMUNICATION target="T0" volume="203" />
    <COMMUNICATION target="T5" volume="657" />
   </SOURCE_TASK>
   <SOURCE_TASK source="T2">
    <COMMUNICATION target="T1" volume="1037" />
    <COMMUNICATION target="T3" volume="369" />
    <COMMUNICATION target="T4" volume="1939" />
    <COMMUNICATION target="T0" volume="1330" />
    <COMMUNICATION target="T5" volume="2719" />
   </SOURCE_TASK>
  </COMMUNICATION_TASK_LIST>
 </APPLICATION_DESCRIPTION>
<APPLICATION_CHARACTERIZATION>
   <TASK_LIST>
    <TASK id="T4">
     <PROCESSOR_TYPE type="PentiumV" power="29.59" data="641"
code="260" cpuOccupation="39.67" />
     <PROCESSOR_TYPE type="ARM11" power="23.39" data="2860"
code="885" cpuOccupation="49.75" />
     <PROCESSOR_TYPE type="PowerPC" power="19.34" data="2527"
code="400" cpuOccupation="11.17" />
    </TASK>
    <TASK id="T5">
     <PROCESSOR_TYPE type="PentiumV" power="30.97" data="1156"
code="501" cpuOccupation="55.54" />
     <PROCESSOR_TYPE type="ARM11" power="21.89" data="3227"
code="800" cpuOccupation="55.49" />
     <PROCESSOR_TYPE type="PowerPC" power="32.14" data="2702"
code="509" cpuOccupation="22.1" />
    </TASK>
    <TASK id="T1">
     <PROCESSOR_TYPE type="PentiumV" power="46.15" data="1178"
code="366" cpuOccupation="85.16" />
     <PROCESSOR_TYPE type="ARM11" power="9.31" data="1370"
code="467" cpuOccupation="77.11" />
     <PROCESSOR_TYPE type="PowerPC" power="7.72" data="3450"
code="531" cpuOccupation="72.74" />
    </TASK>
    <TASK id="T0">
     <PROCESSOR_TYPE type="PentiumV" power="41.43" data="957"
code="459" cpuOccupation="29.69" />
     <PROCESSOR_TYPE type="MIPS" power="32.23" data="3043"
code="119" cpuOccupation="66.31" />
     <PROCESSOR_TYPE type="ARM11" power="13.82" data="679"
code="626" cpuOccupation="16.66" />
     <PROCESSOR_TYPE type="PowerPC" power="18.4" data="2122"
code="272" cpuOccupation="87.61" />
    </TASK>
    <TASK id="T3">
     <PROCESSOR_TYPE type="PentiumV" power="11.77" data="1036"
code="228" cpuOccupation="40.97" />
     <PROCESSOR_TYPE type="ARM11" power="39.05" data="4780"
code="490" cpuOccupation="70.25" />
     <PROCESSOR_TYPE type="PowerPC" power="8.23" data="711"
code="82" cpuOccupation="80.13" />
    </TASK>
    <TASK id="T2">
     <PROCESSOR_TYPE type="PentiumV" power="13.17" data="1774"
code="186" cpuOccupation="55.41" />
     <PROCESSOR_TYPE type="ARM11" power="15.8" data="4864"
code="467" cpuOccupation="60.15" />
     <PROCESSOR_TYPE type="PowerPC" power="17.35" data="555"
code="129" cpuOccupation="38.12" />
    </TASK>
   </TASK_LIST>
 </APPLICATION_CHARACTERIZATION>
</SYSTEM_SPECIFICATION>
```

Figura 33. Exemplo de uma descrição de aplicação sintética paralela com processadores heterogêneos.

O campo (*tag)* TARGET_ARCHITECTURE contém os 4 processadores de tipos diferentes com suas respectivas configurações e nomenclatura no sistema. O campo APPLICATION_DESCRIPTION descreve o volume de dados e a orientação de comunicação entre as tarefas da aplicação (e.g. a tarefa T0 é fonte de comunicação para a tarefa T5, com um volume de 510, representado em Kb). O campo APPLICATION_CHARACTERIZATION apresenta as 6 tarefas definidas com suas características sobre os 4 tipos de processadores estipulados.

Tendo como entrada a descrição XML da Figura 33 e aplicando o particionamento, o PALOMA gera um arquivo texto de extensão PAR, ilustrado pela Figura 34, como resultado do particionamento de tarefas da aplicação proposta. Este arquivo apresenta o algoritmo escolhido para realizar o particionamento de tarefas, o consumo de energia total após aplicar o particionamento (*Energy Consumption:* 349.202 mJ), o erro médio referente ao balanceamento de carga entre processadores (*Load Balance:* 1289.815), a lista de processadores relacionados, com suas propriedades referentes as tarefas associadas aos mesmos, e por fim, detalhes de desempenho gerados pelo processamento do particionamento (e.g. memória RAM consumida e tempo levado para realizar o particionamento). Por exemplo, o processador P1 (*Processor:* P1), do tipo PentiumV (*Type:* PentiumV), possui as tarefas T3 e T4 associadas (*Mapped Tasks:* T3 T4), consumindo uma energia representada em mJ (*Energy:* 41.36), com uma área de dados (*Data:* 1677.0) e código (*Code:* 488.0) representado em kB, e a taxa de ocupação da CPU (*CPUOccupation:* 80.64).

```
Algorithm:        Simulated Annealing
Partition:                Energy Consumption:
        349.202 mJ
                Load Balance (mean square error):
        1289.815
        Processor: P1, Type: PentiumV
        Mapped tasks: T3 T4
        Energy: 41.36
        Data: 1677.0 / Code: 488.0
        CPUOccupation: 80.64
        Processor: P0, Type: ARM11
        Mapped tasks: T1 T0
        Energy: 23.13
        Data: 2049.0 / Code: 1093.0
        CPUOccupation: 93.77
        Processor: P2, Type: PowerPC
        Mapped tasks: T2 T5
        Energy: 49.49
        Data: 3257.0 / Code: 638.0
        CPUOccupation: 60.22
Partition runtime: 10873ms
Used memory: 2102856bytes
Used memory: 2Mb
```

Figura 34. Exemplo de um arquivo PAR gerado pelo PALOMA, a partir da entrada do tipo XML apresentado na Figura 33.

Após o particionamento concluído é gerado um arquivo de extensão CWG, representando o formato padrão para o *framework* CAFES. Este arquivo, que representa a aplicação, pode então ser carregado pelo CAFES para gerar o mapeamento, considerando neste caso apenas critérios de redução do consumo de energia dinâmica.

O arquivo CWG gerado pelo particionamento da aplicação exemplo é ilustrado na Figura 35. Este arquivo apresenta o tamanho da NoC (*tag* #_NoC_Size) – neste exemplo, 2 linhas e 2 colunas; a lista de elementos de processamento (*tag* #_CWG_Vertices) – neste exemplo, 4 processadores; e uma lista contendo a quantidade de comunicação entre o processador origem e destino (*tag* #_CWG_Edges) – no caso 6 comunicações.

```
#_NoC_Size (lines columns)
 2 2

#_CWG_Vertices (list of: vertices)
 P1 P0 P3 P2

#_CWG_Edges (list of: sourceVertex - targetVertex numberOfPhitsTransmited)
 P1 - P0 2793
 P1 - P2 2752
 P0 - P1 2662
 P0 - P2 2504
 P2 - P0 4883
 P2 - P1 4247
```

Figura 35. Arquivo CWG gerado automaticamente pelo particionamento da aplicação descrita na Figura 33.

Utilizando o CAFES na opção *Communication Weight Model* como modelo de aplicação para carregar o arquivo de extensão CWG contendo a descrição da comunicação da aplicação particionada, uma representação gráfica da aplicação é apresentada, como ilustrado na Figura 36 (a).

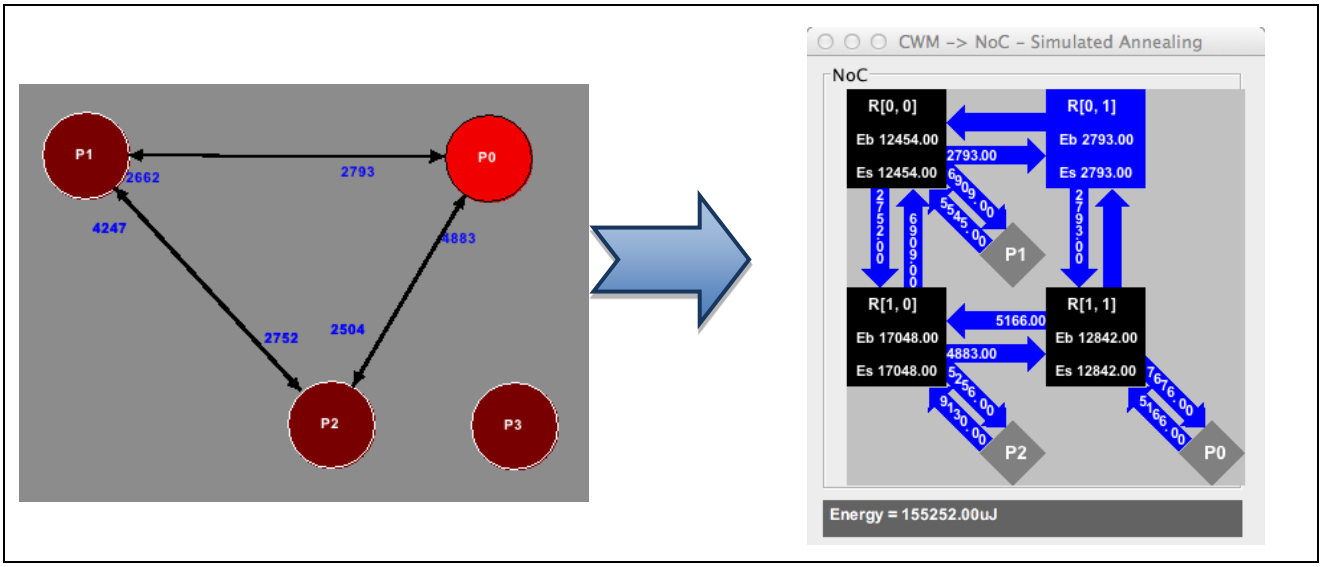


Figura 36. (a) Representação gráfica da descrição de comunicação da aplicação particionada (arquivo CWG); (b) Mapeamento da aplicação descrita na Figura 33 em uma NoC do tipo malha de tamanho 2x2.

A partir desta representação gráfica do arquivo CWG, o CAFES realiza o mapeamento da aplicação, resultando na Figura 36 (b). Este mapeamento apresenta uma relação associada a uma estimativa do consumo de energia dinâmica para cada recurso da arquitetura de comunicação (i.e. roteadores e conexões), além de fornecer uma estimativa global do consumo de energia (*Energy*).

7 RESULTADOS

Este Capítulo aborda dois tipos de experimentos. O primeiro experimento verifica a influência de uma abordagem que emprega uma etapa de pré-mapeamento (i.e. particionamento) com relação a uma abordagem que apenas mapeia a aplicação. O segundo experimento avalia o conjunto de algoritmos utilizados para implementar o pré-mapeamento (e.g. TS, SA, GA e BIA).

7.1 Análise das Abordagens com e sem Pré-Mapeamento

Esta Seção compara o mapeamento direto de uma aplicação, composta por um conjunto de tarefas comunicantes, em processadores de um MPSoC heterogêneo (rotulado como *Direct Mapping – DM*, para salientar a aplicação solo de mapeamento), com o particionamento desta aplicação em grupos de tarefas e o subsequente mapeamento destes grupos em processadores do MPSoC (*Pre-Mapping* - PM).

Ambas as abordagens, ilustradas na Figura 6, e discutidas no Capítulo 1.2, estão exemplificadas com uma menor granularidade na Figura 37, extraída de [9], a fim de detalhar e diferenciar estas abordagens comparadas neste Capítulo.

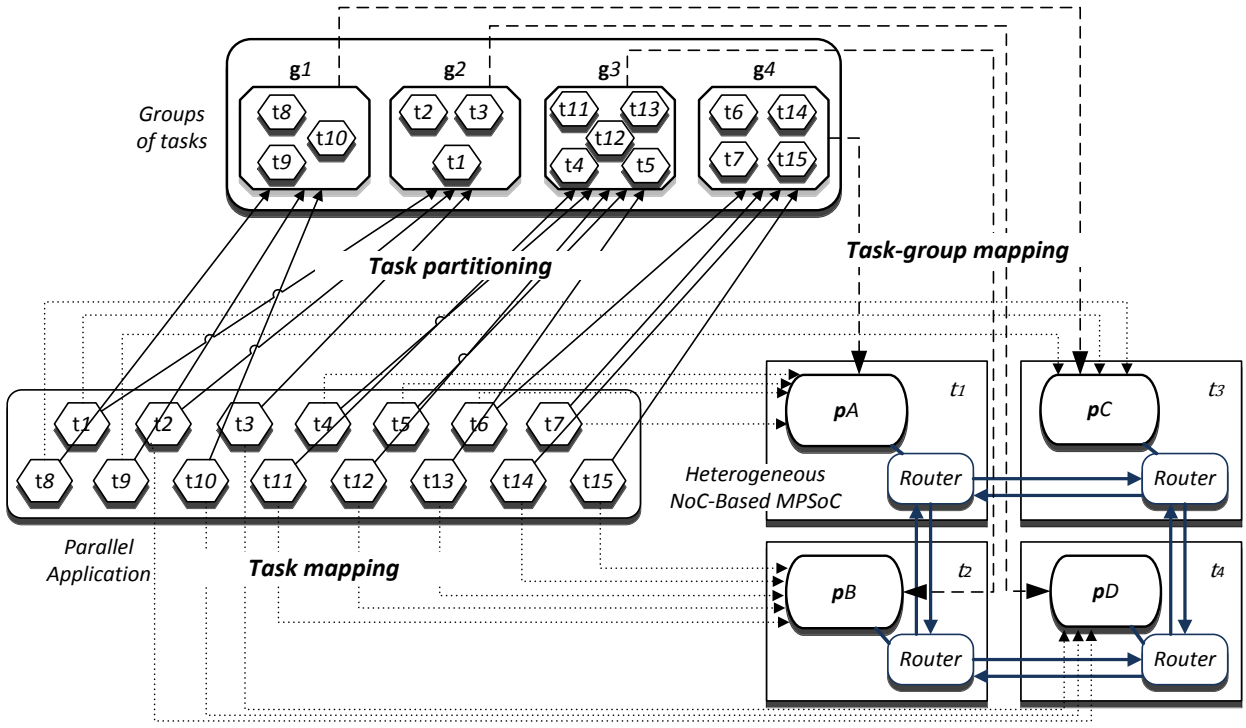


Figura 37. Exemplificação das abordagens Direct Mapping – DM (ilustrada com setas pontilhadas, representando o mapeamento de tarefas – *Task Mapping*) versus Pre-Mapping – PM (ilustrado com setas contínuas (*Task Partitioning*) e tracejadas (*Task-group Mapping*)), sobre uma aplicação sintética tendo como alvo um MPSoC heterogêneo baseado em NoC [9].

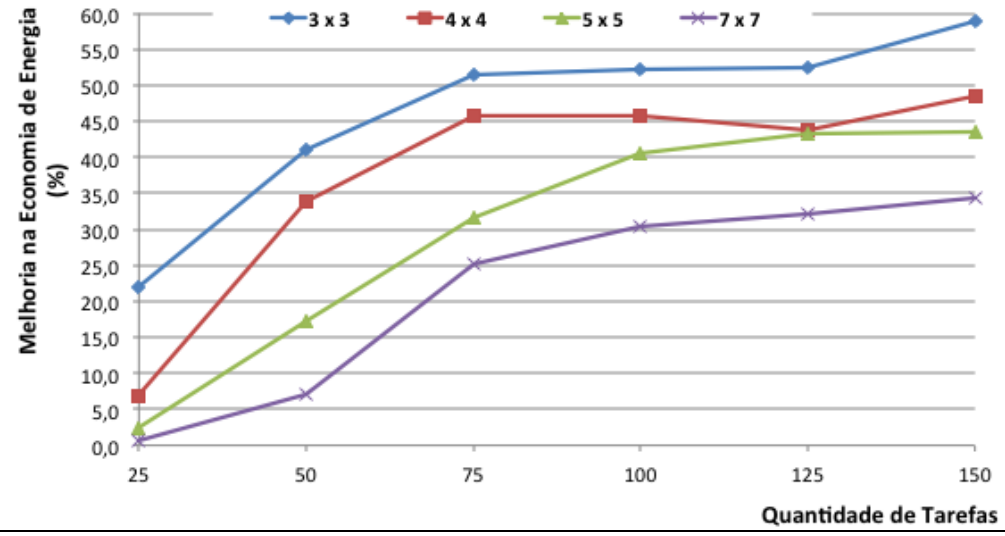
Um dos objetivos deste trabalho é explorar o uso do particionamento de tarefas (estático) como uma atividade de pré-mapeamento ao mapeamento (estático ou dinâmico) propriamente dito. O mapeamento é feito através de dois algoritmos similares aos usados em [52], mas customizados para arquiteturas heterogêneas: (i) o algoritmo de mapeamento tem como entrada a descrição TCG, a qual mapeia as tarefas mais comunicantes em um mesmo processador, enquanto as restrições do processador são atendidas (i.e. carga de trabalho máxima e limite de consumo de energia dependente do tipo do processador). Quando alguma restrição é alcançada, um novo processador vizinho é procurado. A abordagem heterogênea expande o algoritmo implementado em [52], levando em consideração o tipo do processador e a influência na função de custo por ter escolhido o tipo de processador no mapeamento de tarefas; e (ii) o segundo algoritmo de mapeamento tem como entrada a descrição CWG. Este algoritmo procura dentro do grupo de tarefas por um processador onde outras tarefas do mesmo grupo já estão mapeadas. Se nenhuma tarefa foi previamente mapeada, o algoritmo procura por um processador próximo que seja do tipo eleito pelos algoritmos de particionamento estático.

Um conjunto de experimentos foi realizado a fim de avaliar como a utilização do PM minimiza o consumo de energia e melhora o balanceamento de carga em comparação com a abordagem DM. Este conjunto é composto por aplicações sintéticas com média de 15% de canais comunicação entre tarefas (e.g. para uma aplicação contendo 20 tarefas, cada tarefa se comunica com outras três), cada comunicação possui 100 *phits*, e cada *phit* tem 16-bit.

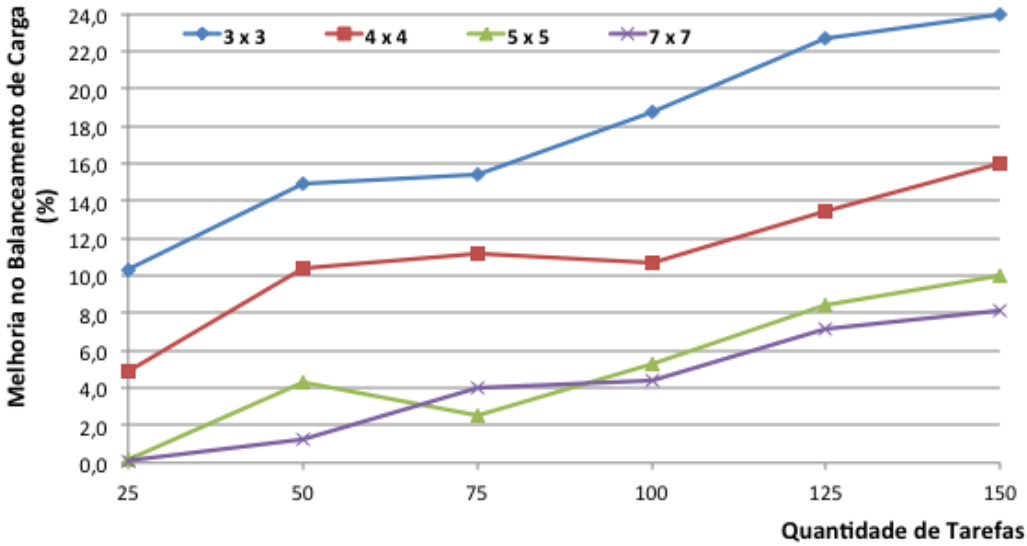
Todos os MPSoCs são baseados em NoC e compostos por três tipos de processadores, com diferentes desempenhos e consumos de energia. A quantidade de processadores é proporcional ao tamanho da NoC, sendo suas posições aleatoriamente distribuídas nos *tiles* da arquitetura alvo. Ainda, as aplicações sintéticas foram geradas considerando que qualquer tarefa da aplicação tivesse a capacidade de ser executada em qualquer tipo de processador. Assim, combinando seis quantidades de tarefas por aplicação (25, 50, 75, 100, 125, e 150) com quatro tamanhos diferentes de NoC (3x3, 4x4, 5x5 e 7x70), totaliza 24 aplicações sintéticas.

O conjunto de experimentos é utilizado como entrada para ambos os fluxos na Figura 38, relacionando duas funções de custo (i.e. função de consumo de energia e função de balanceamento de carga), gerando dois conjuntos de resultados para ambas as abordagens (i.e. PM e DM).

Para cada função custo, os resultados são comparados proporcionalmente, e as porcentagens das melhorias utilizando a abordagem PM ao invés de se utilizar a DM são ilustradas na Figura 38(a) e na Figura 38(b).



a. Melhoria na economia de energia.



b. Melhoria no balanceamento de carga.

Figura 38. Melhoria na economia de energia e no balanceamento de carga quando a abordagem PM é utilizada ao invés da DM.

Os resultados experimentais mostram que, em média, a abordagem PM sempre possibilita alcançar melhores resultados que a DM. Isso se deve porque a abordagem PM pode capturar algumas informações durante o tempo de projeto, onde existe uma grande quantidade de tempo para possibilitar a exploração de várias alternativas de implementação. Além disso, as informações estáticas também possibilitam a redução do espaço de busca por soluções em tempo de execução (i.e. a quantidade de grupos de tarefas utilizada no mapeamento PM é sempre menor que a quantidade de tarefas utilizadas no mapeamento DM), que é uma vantagem evidente para problemas NP Completos.

De fato, o aumento da quantidade de tarefas agrupadas melhora a qualidade dos resultados obtidos com a abordagem PM em relação aos resultados obtidos com a abordagem DM, devido ao incremento da razão entre o número de tarefas pela

quantidade de grupos de tarefas. Mas as melhorias quando utilizado a abordagem PM também são diretamente proporcionais à quantidade de tipos de processador, já que mesmo para pequenos grupos de tarefas, a decisão do tipo de processador alvo pode ser realizada estaticamente pela abordagem PM, enquanto que na abordagem DM esta decisão é feita dinamicamente. Assim, apesar de não mostrar explicitamente na Figura 38(a, b), a abordagem PM melhora o balanceamento de carga e a economia de energia em 9,5% e quase 34%, respectivamente, quando comparado com a abordagem DM.

7.2 Análise da Atividade de Particionamento de Tarefas

Uma vez demonstrada a importância dos algoritmos de particionamento como uma atividade pré-mapeamento, principalmente em relação ao mapeamento dinâmico, esta Seção tem como objetivo avaliar cinco algoritmos de particionamento estático – algoritmos descritos nos Capítulos 4 – em relação a minimização do consumo de energia, balanceamento de carga e tempo de computação.

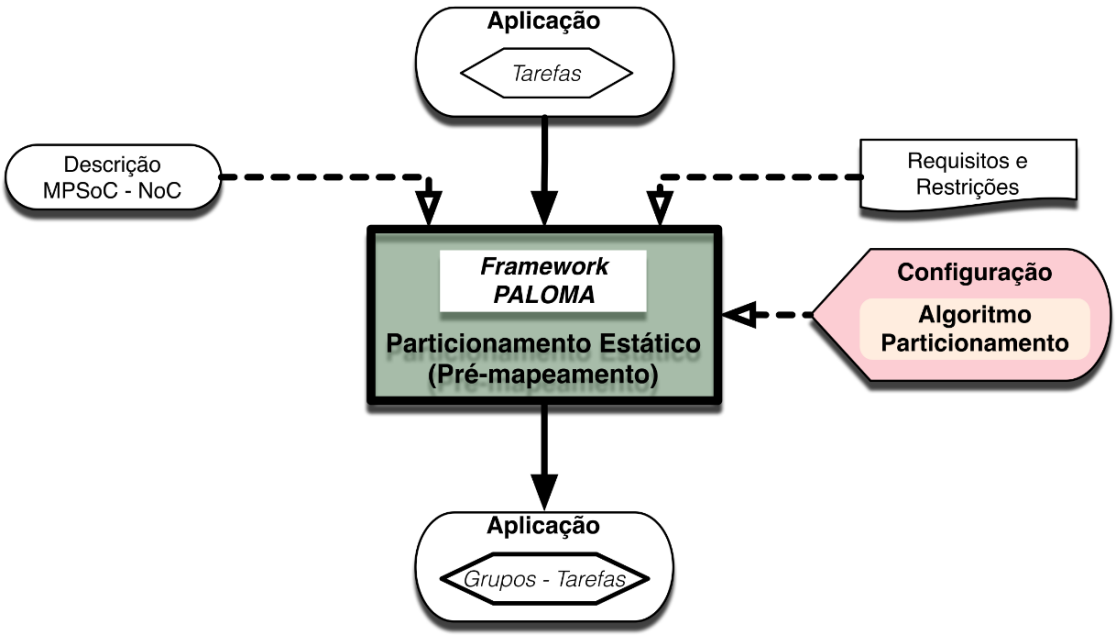


Figura 39. Fluxo de avaliação de algoritmos direcionados a atividade de particionamento de tarefas (pré-mapeamento).

Esta Seção utiliza aplicações sintéticas, similares às aplicadas na Seção 7.1, mas nestes experimentos explorando conectividade de tarefas. Para todo o conjunto de experimentos, o MPSoC alvo tem as seguintes características: (i) comunicações realizadas em uma NoC do tipo malha, de tamanho 3x3; (ii) cada conexão física de comunicação (*phit*) possui um tamanho de 16-bit; (iii) heterogeneidade implementada por três tipos de processadores, todos como diferentes características de desempenho e consumo de energia; (iv) o MPSoC utilizado contém exatamente três processadores de cada tipo selecionados, totalizando nove processadores; (vi) as posições dos processadores nos *tiles* da NoC são escolhidas aleatoriamente.

Além disso, as aplicações sintéticas têm as seguintes características: (i) seis quantidades de tarefas (25, 50, 75, 100, 125 e 150) para cada conjunto de experimentos; (ii) TCG gerado considerando que qualquer tarefa possa ser executada em qualquer tipo de processador; (iii) todas as comunicações das tarefas possuem 1000 *phits*; (iv) TCG gerado considerando cinco percentuais de conectividade entre tarefas (10%, 15%, 20%, 25% e 30%).

A carga de trabalho e a dissipação de energia de cada tarefa, que são dependentes do tipo de processador, são geradas aleatoriamente, cujo alcance vai de 5% a 30% e de 5uW a 15uW, respectivamente. O particionamento tem como restrições para qualquer tipo de processador: a) 100% de carga máxima de trabalho e b) 150uW de consumo máximo de energia.

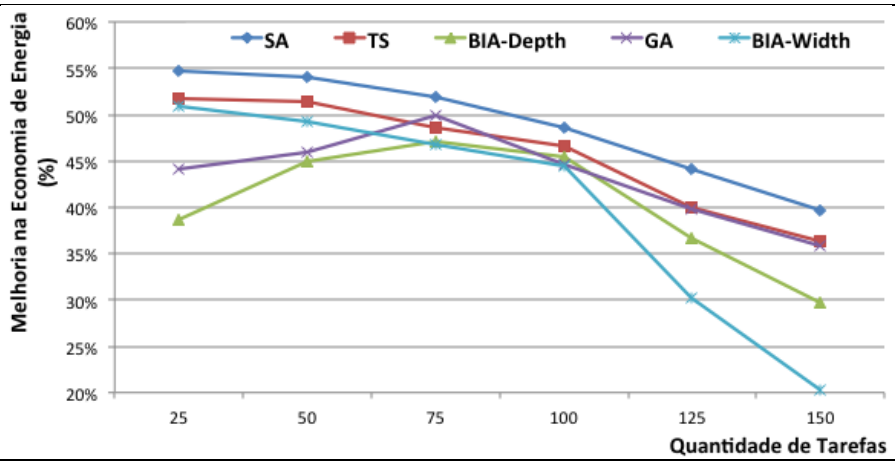
Multiplicando as diferentes quantias de tarefas aplicadas pela quantidade selecionada de conectividade de tarefas totaliza 30 experimentos realizados. Entretanto, para sintetizar os resultados, os experimentos foram agrupados de acordo com a quantidade de tarefas, realizando seis conjuntos de experimentos.

A Figura 40(a, b) agrupa o melhoramento de economia de energia e o melhoramento de balanceamento de carga de todos os conjuntos de experimentos sintéticos, onde cada ponto em cada curva representa uma média de valores alcançados com cinco conjuntos de conectividades de tarefas (10%, 15%, 20%, 25% e 30%).

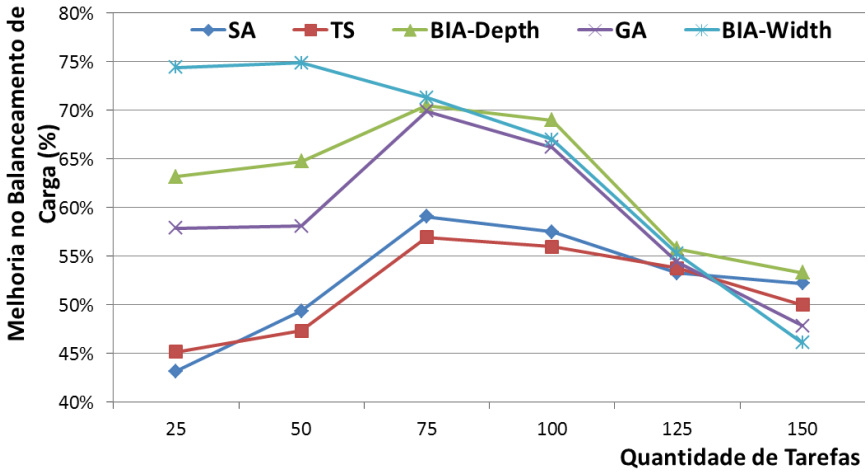
Para todos os experimentos, os ***valores de referência*** utilizados foram produzidos invertendo o objetivo da função custo, i.e. o algoritmo de referência tenta maximizar o consumo de energia e desequilibrar a carga de trabalho. Assim, os valores da Figura 40(a) e Figura 40(b) são porcentagens de quanto os algoritmos avaliados melhoraram o consumo de energia e o balanceamento de carga, quando comparados com os valores adquiridos pelo algoritmo de referência, respectivamente.

A Figura 40(a) mostra que métodos estocásticos (i.e. SA e TS) e genéticos (i.e. GA) minimizam mais o consumo de energia em comparação com as implementações em profundidade e largura do BIA. Entretanto, as melhorias não são suficientemente significativas, se comparado com a abordagem BIA-Width, para experimentos com um número de 100 ou menos tarefas, e se comparado com a abordagem BIA-Depth para experimentos com um número de 50 a 150 tarefas. Na verdade, este resultado mostra que a abordagem algorítmica BIA-Width tem um melhor desempenho em comparação a abordagem BIA-Depth para aplicações de baixa complexidade, enquanto que a abordagem BIA-Depth possui um melhor desempenho sobre aplicações de alta complexidade. Este comportamento acontece provavelmente porque a abordagem BIA-Depth tenta produzir uma associação ótima de grupo-tarefa, antes de considerar otimizações nas tarefas restantes (relativo à Figura 22(a)). Entretanto, quando a

associação grupo-tarefa focada é otimizada, ela é retirada do particionamento, simplificando assim os próximos passos. Por outro lado, o BIA-Width sempre pode revisitar associações grupos-tarefa anteriores, aplicando novos particionamentos, caso estes efetivamente minimizem a função custo. Consequentemente, para aplicações complexas, a abordagem BIA-Width possui um melhor desempenho porque considera todo o conjunto de tarefas durante o particionamento, enquanto que para problemas de maior complexidade, o BIA-Depth possui um desempenho melhor devido à minimização da complexidade da aplicação a cada passo do algoritmo.



a. Melhoria na economia de energia.



b. Melhoria no balanceamento de carga.

Figura 40. Melhorias alcançadas com os algoritmos de particionamentos (SA, TS, BIA-Depth, GA, BIA-Width) relativas à economia de energia e balanceamento de carga.

Figura 40(b) ilustra a comparação entre balanceamentos de carga, onde a natureza das abordagens BIA possibilita a produção de partições com uma alta qualidade de balanceamento de carga. Entretanto, a complexidade das aplicações pode minimizar os ganhos das abordagens BIA, similarmente ao GA, quando comparadas com os métodos estocásticos. Para alguns experimentos com 125 tarefas ou mais, métodos estocásticos têm demonstrado melhores resultados referentes ao balanceamento de carga.

Os experimentos apresentados na Figura 40(a) e Figura 40(b) foram adquiridos considerando o número de interações e temperatura em torno de um milhão para todos os algoritmos. Para os algoritmos estocásticos (e.g. SA e TS), esta configuração implicou em média dois minutos de execução para cada experimento; o algoritmo GA levou em média quatro a cinco vezes mais que os algoritmos estocásticos, em execução para cada experimento; enquanto que a abordagem BIA-Width levou menos de cinco segundos para executar (experimento com 25 tarefas), a abordagem BIA-Depth é ainda quatro vezes mais rápida, na média, o que mostra a eficiência das abordagens BIA principalmente quando leva em consideração o requisito de balanceamento de carga. Os tempos de computação alcançados pelos algoritmos de particionamento nos experimentos estão ilustrados na Figura 41.

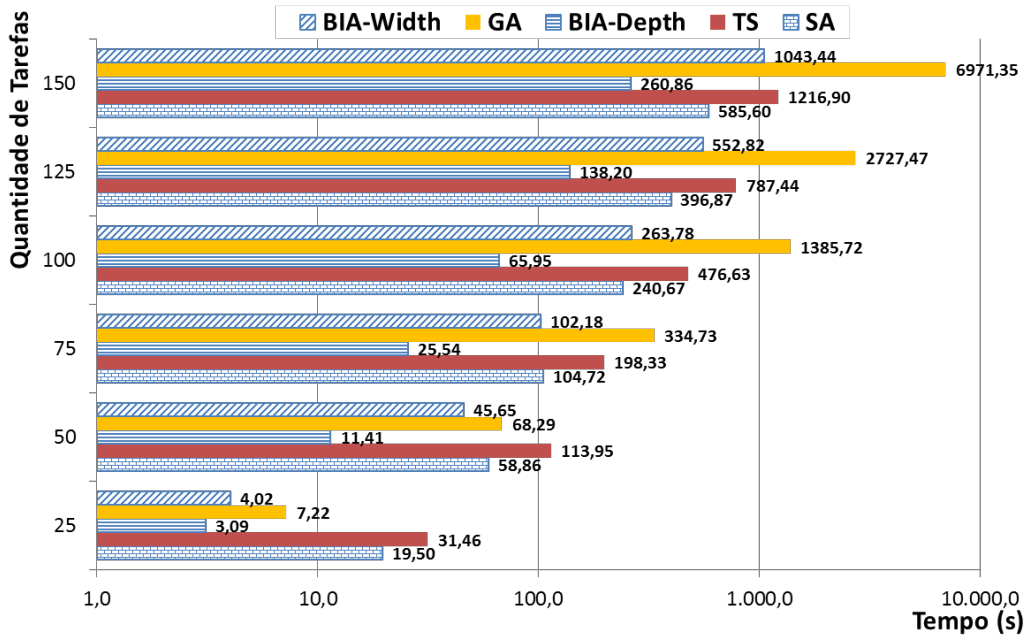


Figura 41. Tempo de computação alcançado pelos algoritmos de particionamento nos experimentos.

Finalizando, é importante salientar que todos os algoritmos são penalizados com a complexidade das aplicações, incluindo o algoritmo SA usado como referência. Esta desvantagem reduz a qualidade das partições, mas também minimiza a diferença entre os valores alcançados e a referência correspondente, o que é provavelmente uma das principais razões pelas quais as melhorias são notavelmente percebidas em aplicações com 100 tarefas ou mais.

8 CONCLUSÕES

Este trabalho abordou como tema principal o particionamento de tarefas em MPSoCs heterogêneos baseados em NoCs do tipo malha 2D. Foram exploradas diferentes abordagens algorítmicas bem como a avaliação de utilizar o particionamento como uma atividade de pré-mapeamento de forma a obter mapeamentos de melhor qualidade.

Os principais objetivos do trabalho são: (i) avaliar o impacto do particionamento de tarefas como uma técnica pré-mapeamento a fim de melhorar o mapeamento, seja este dinâmico ou estático; e, uma vez demostrado o impacto do particionamento de tarefas, (ii) analisar, comparar, e propor algoritmos estáticos de particionamento tendo como alvo arquiteturas heterogêneas. Ambos objetivos visam adquirir resultados que atendam os requisitos de minimização do consumo de energia e balanceamento de carga.

O particionamento estático de tarefas em grupos reduz a complexidade do mapeamento, reduzindo o espaço de busca que o mapeamento necessita realizar, o que permite uma execução mais eficiente de algoritmos de mapeamento dinâmicos ou estáticos, resultando em um mapeamento de melhor qualidade. Isto pode ser observado para aplicações compostas por centenas de tarefas comunicantes mapeadas dinamicamente em vários processadores de MPSoCs heterogêneos. Na maioria dos trabalhos encontrados na literatura propõem-se soluções para o mapeamento de tarefas sem um prévio processo de minimização da complexidade do problema (vide Capítulo 2). Ainda, este trabalho estende a utilização do particionamento de tarefas, utilizando várias abordagens algorítmicas, avaliando a utilização dos mesmos em diferentes complexidades. Experimentos mostraram que a utilização do particionamento de tarefas como uma forma de pré-mapeamento traz um ganho na economia de energia em torno de 34% se comparado com o mapeamento direto.

No presente trabalho, foram propostas e avaliadas cinco abordagens algorítmicas (e.g. SA, TS, GA, BIA-Width e BIA-Depth) para o particionamento estático de tarefas, sobre os quais foram feitos diversos experimentos. SA, TS e GA são algoritmos clássicos, largamente utilizados, enquanto que as abordagens BIA-Width e BIA-Depth foram propostas no contexto deste trabalho, sendo estas implementadas por algoritmos semelhantes, onde o grafo da aplicação descrevendo as comunicações paralelas é percorrido na forma de profundidade - *depth* ou na largura – *width*. Estas abordagens foram baseadas no algoritmo *Kernighan-Lin (KL)*, onde sua essência foi alterada para contemplar a atividade de particionamento de tarefas.

Os resultados mostraram que a complexidade das aplicações (i.e. número de conexões e distribuição das mesmas entre tarefas de uma aplicação) é decisiva para o desempenho dos algoritmos avaliados. À medida que a complexidade aumenta,

algoritmos do tipo estocásticos tendem a apresentar melhores resultados em relação à economia de energia. Por outro lado, aplicações menos complexas particionadas com os algoritmos BIA tendem a ter melhor balanceamento de carga com menor esforço computacional, se comparadas com SA, TS e GA.

8.1 Contribuições do Trabalho

Dentre as contribuições do trabalho constam:

Revisão de trabalhos relacionados – Investigação de trabalhos relacionados (i) ao processo de particionamento e mapeamento de tarefas em MPSoCs, e (ii) algoritmos utilizados na atividade de particionamento e mapeamento de tarefas. No que diz respeito ao alvo desta pesquisa, os algoritmos utilizados no processo de particionamento e mapeamento de tarefas, e o processo de particionamento de tarefas estático e o mapeamento de tarefas dinâmico e estático, com ênfase em MPSoCs heterogêneos, conforme apresentado no Capítulo 2.

Exploração Algorítmica – Estudo, implementação, e comparação de algoritmos clássicos da literatura, aplicados ao particionamento de tarefas, visando a melhoria da economia de energia e um melhor balanceamento de carga, em arquiteturas MPSoCs heterogêneas baseadas em NoC. Dentre os algoritmos clássicos apresentados no Capítulo 4, constam: SA e TS, do grupo de algoritmos estocásticos; GA do grupo de algoritmos genéticos; e KL, do grupo de algoritmos de bissecção (i.e. migração).

Proposta de Algoritmo – Proposta de um algoritmo, chamado BIA, baseado no estudo de um algoritmo clássico da literatura (i.e. KL), empregado na bissecção de grafos, e usualmente utilizado no particionamento de circuitos. O algoritmo implementado, em duas abordagens diferentes, baseia-se na bissecção de partições e trocas de tarefas entre partições bisseccionadas, até que grupos de tarefas sejam formados, atendendo determinados requisitos e restrições. Dessa forma, o algoritmo visa reduzir o consumo de energia e balancear as tarefas de forma uniforme nas partições. Apresentado conforme o Capítulo 4.5.

Framework *PALOMA* – Extensão do framework PALOMA para inclusão do particionamento de tarefas tendo como alvo MPSoCs heterogêneos, além da inclusão de cinco abordagens algorítmicas, conforme apresentado no Capítulo 5.

Atividade de Pré-mapeamento – Utilização do particionamento de tarefas estático como uma atividade de pré-mapeamento, a fim de reduzir a complexidade da aplicação para o mapeamento dinâmico ou estático.

Publicações – O desenvolvimento deste trabalho resultou em uma publicação e uma submissão. A primeira delas, *Partitioning Algorithms Analysis for Heterogeneous*

NoC based MPSoC [49], é relacionada à análise e comparação algorítmica empregada na atividade de particionamento de tarefas estático, com ênfase em MPSoCs heterogêneos baseados em NoC. O amadurecimento deste trabalho resultou na criação do artigo *Evaluation of Partitioning Algorithms: A Pre-Mapping Targeting Heterogeneous NoC-based MPSoCs* [9] que trata da aplicação da atividade de particionamento de tarefas estático como uma atividade pré-mapeamento dinâmico ou estático. Este artigo foi submetido à revista DAES.

9 REFERÊNCIAS

[1] Martin, G.; Chang, H. “System-on-Chip design.” *4th International Conference on ASIC*. 2001, pp. 12-17.

[2] Jerraya, A.; Tenhunen, H. and Wolf, W.; Guest Editors’ Introduction: Multiprocessor Systems-on-Chips. IEEE Computer, 2005, Vol.33, pp. 36-40.

[3] Wolf, W. “The Future of Multiprocessor Systems-on-Chips”. *Proceedings of the Design Automation Conference* (DAC), 2004, pp. 681-685.

[4] Cornero M.; Anyuru A. “Multiprocessing in Mobile Platforms: The Marketing and The Reality”. Capturado em: www.stericsson.com/technologies/FD-SOI-eQuad-white-paper.pdf, Março 2013.

[5] Dally, W. J.; Towles, B. “Route Packets, Not Wires: on Chip Interconnection Networks”. *Proceedings of the Design Automation Conference* (DAC), 2001, pp. 684-689.

[6] Antunes, E. B. “Particionamento de MPSoCs Homogêneos Baseados em NoCs”, *Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação*, PUCRS, 2011.

[7] Marcon, C. “Modelos para o Mapeamento de Aplicações em Infraestruturas de Comunicações Intrachip”. *Tese de Doutorado, PPGC/UFRGS*, Porto Alegre, Brasil, 2005.

[8] Wolf, W. et al. “Multiprocessor System-on-Chip (MPSoC) Technology”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008, pp. 1701-1713.

[9] Marcon, C.; Webber, T.; Pinotti, I. K.; Ribeiro, N.; Fagundes, R. D. R.; Poehls, L. B. “Evaluation of Partitioning Algorithms: A Pre-Mapping Targeting Heterogeneous NoC-based MPSoCs”. *Proceedings of Springer Journal of Design Automation for Embedded Systems – DAES, Special Issue on the III Brazilian Workshop on Embedded Systems – WSE*, 2013, pp. 1-21.

[10] Singh, A; Jigang, W.; Prakash, A.; Srikanthan, T. “Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms”. *Proceedings of the Euromicro Conference on Digital System Design, Architectures, Methods and Tools – DSD*, 2009, pp. 133-140.

[11] *Alpert, C.; Kahng, A.* “Recent directions in netlist partitioning: a survey”. *Integration, the VLSI Journal 19*, 1995, pp. 1-81.

[12] Sahu, P.; Chattopadhyay, S. “A survey on application mapping strategies for Network-on-Chip design”. *Journal of Systems Architecture (JSA): The Euromicro Journal*, 2003, pp. 60-76.

[13] Hu, J.; Marculescu, R. “Energy-Aware Mapping for Tile-Based NoC Architectures under Performance Constraints”. *Proceedings of the Asia and South Pacific Design Automation Conference* (ASPDAC), 2003, pp. 233-239.

[14] Wang, X.;Yang, M.; Jiang, Y.; Liu, P. “A Power-Aware Mapping Approach to Map IP Cores onto NoCs under Bandwidth and Latency Constraints”. *Transaction on Architecture and Code Optimization - ACM*, 2010.

[15] Hu, J.; Marculescu, R. “Energy and Performance-Aware Mapping for regular NoC Architectures”. *IEEE Trans. Comp. Aind. Des. Integr. Circuits Syst.*, 2005, pp. 551-562.

[16] Vivekanandarajah, K.; Pilakkat, S.K. “Task Mapping in Heterogeneous MPSoCs for System Level Design”. *IEEE International Conference on Engineering of Complex Computer Systems*, 2008, pp. 56-65.

[17] Hansson, A.; Goossens, K.; Radulescu, A. “A Unified Approach to Constrained Mapping and Routing on Network-on-Chip Architectures”. *Proceedings of the International Conference on Hardware/Software Co-Design and System Synthesis*, 2005, pp. 75-80.

[18] Murali, S.; De Micheli, G. “Bandwidth-Constrained Mapping of Cores onto NoC Architectures”. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2004, pp. 896-901.

[19] Marcon, C. A. M.; Moreno, E. I.; Calazans, N. L. V.; Moraes, F. G. “Evaluation of Algorithm for Low Energy Mapping into NoCs”. *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2007, pp. 389-392.

[20] Mehran, A.; Saeidi, S.; Khademzadeh, A.; Afzali-Kusha, A. “Spiral: A Heuristic Mapping Algorithm for Network on Chip”. *IEICE Electron*, 2007, pp. 478-484.

[21] Srinivasan, K.; Chatha, K. S. “A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures”. *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005, pp. 387-392.

[22] Murali, S.; Coenen, M.; Radulescu, A.; Goossens, K.; De Micheli, G. “Mapping and Configuration Methods for Multi-Use-Case Networks on Chips”. *Proceedings of the Asia and South Pacific Design Automation Conference* (ASPDAC), 2006, pp. 146-151.

[23] Singh, A. K.; Srikanthan, T.; Kumar, A.; Wu, J. “Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms”. *J. Syst. Archit.*, 2010, pp. 242-255.

[24] Lu, S-S.; Lu, C-H.; Hsiung, P-A.; “Congestion and Energy-aware Run-time Mapping for Tile-based Network-on-chip Architecture”. *Proceedings of IET International Conference on Theory, Technologies and Applications*, 2010, pp. 300-305.

[25] Habibi, A.; Arjomand, M.; Sarbazi-Azad, H. “Multicast-aware Mapping Algorithm for On-chip Networks”. *Proceedings of the Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2011, pp. 455-462.

[26] Kaushik, S.; Singh, A.; Srikanthan, T. “Preprocessing-based Run-time Mapping of Applications on NoC-based MPSoCs”. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2011, pp. 337-338.

[27] Kaushik, S.; Singh, A.; Srikanthan, T. “Computation and Communication Aware Run-Time mapping for NoC-based MPSoC Platforms”. *Proceedings of the IEEE International SOC Conference (SOCC)*, 2011, pp. 185-190.

[28] Singh, A. et al. “Run-time Mapping of Multiple Communicating Tasks on MPSoC Platforms”. *Proceedings of the International Conference on Computational Science (ICSS),* 2010, pp. 1019-1026.

[29] Castrillon, J.; Leupers, R.; Ascheid, G. “MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics*, 2013, pp. 527-545.

[30] Ferrandi, F.; Pilato, C.; Sciuto, D.; Tumeo, A. “Mapping and scheduling of parallel C applications with ant colony optimization onto heterogeneous reconfigurable MPSoCs”. *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2010, pp. 799-804.

[31] Ascia, G.; Catania, V.; Palesi, M. “Multi-objective Mapping for Mesh-based NoC Architectures”. *Proceedings of the International Conference on Hardware/Software Co-Design and System Synthesis*, 2004, pp. 182-187.

[32] Zhou, W.; Zhang, Y.; Mao, Z. “Pareto-based Multi-objective Mapping IP Cores onto NoC Architectures”. *Proceedings of the Asia Pacific Conference on Circuits and Systems*, 2006, pp. 331-334.

[33] Harmanani, H. M.; Farah, R. “A Method for Efficient Mapping and Reliable Routing for NoC Architectures with Minimum Bandwidth and Area”.

Proceedings of the Conference on Circuits and Systems and TAISA, 2008, pp. 29-32.

[34] Lu, Z.; Xia, L.; Jantsch, A.; “Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip”. *Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, 2008, pp. 1-6.

[35] Marcon, C. et al. “Comparison of Network-on-chip Mapping Algorithms Targeting Low Energy Consumption”. *IET Computers & Digital Techniques, 2008, pp. 471-482.*

[36] Antunes, E. et al. “Partitioning and Mapping on NoC-Based MPSoC: An Energy Consumption Saving Approach”. *Proceedings of the International Workshop on Network on Chip Architectures (NoCArc)*, 2011, pp. 51-56.

[37] He, O.; Dong, S.; Jang, W.; Bian, J.; Pan, D. “UNISM: Unified Scheduling and Mapping for General Networks on Chip”. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2012, pp. 1496-1509.

[38] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. Optimization by simulated annealing”. *Science*, 1983, pp. 671–680.

[39] Wiangtong, T.; Cheung, P.; Luk, W. “Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign”. *Design Automation for Embedded Systems*, 2002, pp. 425-449.

[40] Glover, F. “Tabu Search – Part 1”. *ORSA J. Comput. 1*, 1989, pp. 190-206.

[41] Glover, F. “Tabu Search – Part 2”. *ORSA J. Comput. 1*, 1990, pp. 4-32.

[42] Glover, F. “Tabu Search: A Tutorial”. *Interfaces*, 1990.

[43] Beaty, S. J. “Genetic Algorithms versus Tabu Search for Instruction Scheduling”. *Int. Conf. on Neural Network and Genetic Algorithms,* 1993, pp. 496–501.

[44] Grajcar, M. “Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system”. *Proceedings of the Design Automation Conference (DAC)*, 1999, pp. 280–285.

[45] Kernighan, B.; Lin, S. “An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Systems Technical Journal*, 1970, pp. 291-307.

[46] Schweikert, D.; Kernighan, B. “A Proper Model for the Partitioning of Electrical Circuits”. *Proceedings of the Design Automation Conference (DAC)*, 1972, pp 57–62.

[47] Fiduccia, C. M.; Mattheyses, R. M. "A Linear Time Heuristic for Improving Network Partitions," *Proceedings of the Design Automation Conference (DAC),* 1982, pp. 175-181.

[48] Sherwani, N. A. “Algorithms for VLSI Physical Design Automation”. Norwell, Massachusetts, USA: Kluwer Academic Publishers, 1993, 487p.

[49] Pinotti, I. K.; Webber, T.; Ribeiro, N.; Fraga, C. N.; Fagundes, R. D. R.; Marcon, C. “Partitioning Algorithms Analysis for Heterogeneous NoC based MPSoC”. *Proceedings of the Simpósio Brasileiro de Engenharia de Sistemas Computacionais – SBESC – III – Workshop de Sistemas Embarcados – WSE*, 2012, pp. 1-6.

[50] Marcon, C.; Calazans, N.; Moreno, E.; Moraes, F.; Hessel, F.; Susin, A. “CAFES: A framework for intrachip application modeling and communication architecture design”. *Journal of Parallel and Distributed Computing*, 2010.

[51] Salminen, E.; Grecu, C.; Hamalainen, T.D.; Ivanov, A. “Application modeling and hardware description for network-on-chip benchmarking”. *Computers & Digital Techniques - IET*, 2009, pp. 539-550.

[52] Antunes, E. et al. “Partitioning and Dynamic Mapping Evaluation for energy consumption minimization on NoC-based MPSoC”. *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*, 2012, pp. 451-457.