PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PARTICIONAMENTO E MAPEAMENTO DE MPSOCS HOMOGÊNEOS BASEADOS EM NOCS

EDUARDO DE BRUM ANTUNES

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

Porto Alegre

2012

Dados Internacionais de Catalogação na Publicação (CIP)

A636p Antunes, Eduardo de Brum
Particionamento e mapeamento de MPSOCS homogêneos baseados em NOCS / Eduardo de Brum Antunes. – Porto Alegre, 2012.
83 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César Augusto Missio Marcon.

1. Informática. 2. Multiprocessadores. 3. Energia Elétrica – Consumo. I. Marcon, César Augusto Missio. II. 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 ***"Particionamento e Mapeamento de MPSoCs Homogêneos Baseados em NoCs"***, apresentada por Eduardo de Brum Antunes como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 29/02/2012 pela Comissão Examinadora:

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

Prof. Dr. Fabiano Passuelo Hessel - — PPGCC/PUCRS

Prof. Dr. Fernando Gehm Moraes - — PPGCC/PUCRS

Prof. Dr. Altamiro Amadeu Susin – — UFRGS

Homologada em 20/03/2012, conforme Ata No. 006/12 pela Comissão Coordenadora.

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

AGRADECIMENTOS

Gostaria de agradecer a todas as pessoas que tiveram uma relação direta ou indireta para a realização deste trabalho de pesquisa de dois anos e na consequente escrita desta dissertação.

Agradeço a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo financiamento deste trabalho, provendo o apoio financeiro necessário para a realização deste trabalho e possibilitando a minha dedicação exclusiva ao mesmo.

Agradeço a todos os meus colegas de mestrado que me deram uma grande força na hora de cumprir todas as disciplinas do mestrado. Em especial ao Tales Marchesan e Thiago Raupp, que juntos sempre nos divertimos e motivamos um ao outro para a realização das atividades. Valeu Tales! Valeu Raupp!

Manifesto meu agradecimento aos bolsistas de iniciação científica Marcos Sartori (que auxiliou na estruturação inicial do *framework* PALOMA) e Matheus Soares (que ajudou a encontrar e corrigir vários erros no *framework* PALOMA, e também auxiliou na atividade de mapeamento das tarefas diretamente nos processadores, sem particionamento prévio).

Tenho muito a agradecer ao meu orientador, Cesar Augusto Missio Marcon, que é um exemplo de pesquisador e professor. Agradeço pelo constante incentivo à pesquisa, paciência, dedicação à orientação, e pelo apoio nos momentos de dúvidas e frustrações. Valeu Marcon!

Quero agradecer à minha mãe Maria Valdete, minha avó Odila Vargas, que mesmo estando longe, sempre estiveram muito presentes na minha vida, me apoiando com todo amor e carinho para que eu pudesse chegar até aqui. Agradeço aos meus irmãos Paulo, Cláudio, Márcio, Alexandre e Leandro, pelo estímulo, confiança e amor incondicional que sempre demonstraram durante minha vida. Essa minha vitória também é de vocês!

Agradeço aos meus tios (Valter e Loremi, Valdir e Graça), e aos meus primos (Christiano, Valdriano, Flávio e Anderson), pelo acolhimento em diversos finais de semana, por desejarem o sucesso deste trabalho e por momentos de alegria e diversão.

Sobre tudo, agradeço a Deus por sempre estar ao meu lado nos melhores e piores momentos da minha vida.

PARTICIONAMENTO E MAPEAMENTO DE MPSOCS HOMOGÊNEOS BASEADOS EM NOCS

RESUMO

O aumento da complexidade das aplicações demanda maior capacidade de processamento, impulsionando o desenvolvimento de um sistema computacional compostos por módulos como processadores, memórias e núcleos de hardware específicos, chamado de *Multi-Processor System-on-Chip* (MPSoC). Se os módulos deste sistema forem conectados por uma infraestrutura de comunicação do tipo *Network-on-Chip* (NoC) e todos os processadores forem de um único tipo, este é chamado de MPSoC homogêneo baseado em NoC.

Um dos principais problemas relativo ao projeto de MPSoCs é a definição de qual dos processadores do sistema será responsável pela execução de cada tarefa de uma aplicação, visando atender os requisitos de projeto, tais como a redução do consumo de energia e a redução do tempo de execução da aplicação.

Este trabalho tem como objetivo a realização de forma rápida e eficiente das atividades de particionamento e mapeamento para o projeto de MPSoCs homogêneos. Mais especificamente o particionamento de tarefas de uma aplicação em grupos, e o mapeamento de tarefas ou grupos de tarefas em processadores homogêneos de uma arquitetura alvo do tipo MPSoC baseado em NoC. Sendo estas atividades guiadas por requisitos de redução do consumo de energia e balanceamento de carga, e delimitadas por restrições de máximo consumo de energia, máxima carga de processamento e máximas áreas de dados e código associadas a cada processador.

O trabalho mostra a complexidade das atividades de particionamento e mapeamento, separadas e conjuntamente. Mostra também que o mapeamento é mais eficiente na redução de consumo de energia, quando comparado com o particionamento, mas mesmo assim o efeito do particionamento não pode ser negligenciado. Além disto, o efeito conjunto de ambas as atividades reduz em média 37% o consumo de energia.

O mapeamento, quando realizado em tempo de execução, pode ser pouco eficiente, devido ao tempo exíguo e ao grande número de soluções a serem exploradas. Utilizando uma abordagem que aplica um particionamento estático anterior ao

mapeamento dinâmico, permite obter mapeamentos mais eficientes. Isto porque o particionamento estático de tarefas em grupos reduz o espaço de busca que o mapeamento necessita realizar. Experimentos com várias aplicações sintéticas e quatro aplicações embarcadas mostram que a redução média do consumo de energia é de 23,5%.

Este trabalho apresenta o *framework* PALOMA que realiza o particionamento de tarefas em grupos e o *framework* CAFES para fazer o mapeamento destes em posições da arquitetura alvo, onde cada posição contém um processador. Estas atividades permitem planejar sistemas com menor consumo de energia, mais velozes e em tempo de projeto aceitável.

Palavras chave: MPSoC homogêneo, rede intrachip, particionamento, mapeamento, consumo de energia, balanceamento de carga.

PARTITIONING AND MAPPING OF HOMOGENEOUS NOCS-BASED MPSOCS

ABSTRACT

The increasing complexity of the applications demands more processing capacity, which boosts the development of a computational system composed of modules, such as processors, memories and specific hardware cores, called Multi-Processor System-on-Chip (MPSoC). If the modules of this system are connected through a Network-on-Chip (NoC) communication infrastructure and all processors are of the same type, they are known by *homogeneous NoC based MPSoC*.

One of the main problems relating to MPSoCs design is the definition of which processors of the system will be responsible for each application task execution, objecting to meet the design requirements, such as the energy consumption minimization and the application execution time reduction.

This work aims to carry out quickly and efficiently partitioning and mapping activities for the design of homogeneous MPSoCs. More specifically, the partitioning application's task into groups, and mapping of tasks or task groups into a target architecture type homogeneous NoC-based MPSoC. These activities are guided by requirements of energy consumption minimization and load balancing, and delimited by constraints of maximum energy consumption, maximum processing load and maxima areas of data and code of each processor.

The work shows the complexity of partitioning and mapping activities separately and jointly. It also shows that the mapping is more efficient on energy consumption minimization, when compared to partitioning, yet the effect of partitioning cannot be neglected. Moreover, the joint effect of both activities saves in average 37% of energy.

The mapping when performed at runtime may be inefficient, due to the short time and the large number of solutions to be explored. Having an approach that applies a static partition before the dynamic mapping, it is possible to achieve more efficient mappings. It happens due to the fact that static task partition onto groups minimizes the search space of the mapping. Experiments with several synthetic applications and four embedded applications show that the energy consumption is reduced 23.5%, in average.

This paper presents the PALOMA framework that performs the partitioning of tasks

onto groups and the CAFES framework to map these ones into tiles of the target architecture, where each position contains a processor. These activities enable planning systems with less energy consumption, faster and in an acceptable design time.

Keywords: Homogeneous MPSoCs, network-on-chip, partitioning, mapping, energy consumption, load balancing.

LISTA DE FIGURAS

Figura 1 – Particionamento e mapeamento de uma descrição da aplicação (conjunto de tarefas) para uma arquitetura SoC baseada em NoC. 26

Figura 2 - Fluxo de projeto mostrando o particionamento e mapeamento. 27

Figura 3 – Crescimento, em escala logarítmica, do número de combinações frente ao número de elementos em relação ao (i) Particionamento de tarefas em grupos, (ii) Mapeamento de grupos de tarefas em processadores e (iii) Mapeamento de tarefas em processadores. Número de elementos refere-se a (i) número de tarefas, (ii) número de processadores e (iii) número de tarefas x processadores. 28

Figura 4 – Estrutura genérica de um roteador [MAR05]. 29

Figura 5 – Formato do pacote. O *flit* corresponde a menor unidade sobre a qual é feita a regulação do tráfego e o *phit* a largura física do canal. Adaptado de [ZEF03] 30

Figura 6 – Exemplo de topologia regulares de rede intrachip: (a) malha 2D (em inglês, *mesh*), (b) toro 2D (em inglês, *torus*) e (c) hipercubo 3D. 31

Figura 7 – Exemplo de detalhamento de um *tile* em um SoC que usa como infraestrutura de comunicação uma rede com topologia malha. 31

Figura 8 – Fluxo de projeto ilustrando as atividades de particionamento e mapeamento. 44

Figura 9 – Esqueleto de uma especificação de plataforma e aplicação MPSoC em formato XML. 46

Figura 10 – Descrição exemplo de campos da tag PROCESSOR_TYPE_LIST, contendo uma lista de tipos de processadores com suas características físicas. A figura ilustra dois tipos de processadores contendo suas freqüências de operação e suas dimensões. Por exemplo, no caso do processador MIPS, este opera a 1,2GHz, tem como largura 1,2mm e altura 2,5mm. 46

Figura 11 – Campo que contém, para cada tipo de processador, uma lista de identificadores (nomes) de processadores deste tipo. A figura ilustra dois tipos de processadores. Por exemplo, os nomes p1 e p2 identificam processadores do tipo MIPS. 47

Figura 12 – Caracterização de tarefas da aplicação frente aos tipos de processador disponíveis. A figura ilustra um exemplo sintético de caracterização de duas tarefas (T1 e T2) em dois processadores distintos (MIPS e PowerPC). Por exemplo, a tarefa T1 quando executada no processador PowerPC dissipa 20.12 uW (micro watts) de potência, ocupa 3865 KB (kilobytes) de área de dados e 2793 KB de área de código, e requer 34.5% de processamento. Esta mesma tarefa executada em um processador MIPS dissipa 13.5 uW de potência, ocupa 6422 KB de área de dados e 334 KB de área de código, requerendo 24.6 % de processamento. 47

Figura 13 – Descrição da aplicação em relação à inter-comunicação de suas tarefas. Aqui é descrito um trecho sintético contendo duas tarefas que originam a comunicação (T1 e T2) e 3 tarefas que recebem as comunicações (T1, T2 e T3). Por exemplo, existe uma comunicação que parte da tarefa T1 em direção à tarefa T2 com o volume de comunicação igual a 250 kB. 48

Figura 14 – Interface de abertura do PALOMA. 49

Figura 15 – Interface gráfica da ferramenta de geração de aplicações sintéticas. Os valores aqui apresentados são meramente ilustrativos. 51

Figura 16 – Plotagem resultante a partir da execução do algoritmo exaustivo em aplicação com 6 tarefas. As 203 partições – *O(Bell(6))* - estão ordenados por proximidade. 53

Figura 17 – Pseudo-código para o algoritmo SA. 54

Figura 18 – Pseudo-código para o algoritmo que computa o custo com o objetivo de reduzir o consumo de energia. 56

Figura 19 – Pseudo-código para o algoritmo que computa o custo para balanceamento de carga. 57

Figura 20 - Exemplo do balanceamento de carga de uma aplicação composta por 4 processadores e 8 tarefas. 58

Figura 21 – (a) apresenta a interface de abertura do *framework* CAFES, indicando a escolha do modelo de aplicação e arquitetura de comunicação alvo. (b) apresenta a interface de configuração dos parâmetros da arquitetura alvo, para topologia malha 2D e aplicações modeladas com o modelo ECWM. Os parâmetros de energia são adaptados de acordo com a tecnologia CMOS TSMC 0,35μm. 60

Figura 22 – Descrição de uma aplicação sintética. 63

Figura 23 – Exemplo de um arquivo PAR gerado pela ferramenta de particionamento (PALOMA) e tendo como entrada o XML apresentado na Figura 22. 64

Figura 24 – Arquivo CWG gerado automaticamente pelo particionamento da aplicação descrita na Figura 22. 65

Figura 25 – Visualização da aplicação quando carregada no framework CAFES. 66

Figura 26 – Mapeamento da aplicação descrita na Figura 22 em uma NoC malha 2x2. 66

Figura 27 – Influência do particionamento e do mapeamento na redução do consumo de energia para uma série de volumes de comunicação. 70

Figura 28 – Influência do particionamento e mapeamento na redução do consumo de energia para uma série de número de comunicações. 71

Figura 29 – Influência do particionamento e mapeamento na redução do consumo de energia para uma variedade de tamanhos de NoC. 72

Figura 30 – Influência do particionamento e mapeamento na redução do consumo de energia para uma série de número de tarefas. 72

Figura 31 – Influência do particionamento e do mapeamento na redução do consumo de energia para quatro aplicações embarcadas. 73

Figura 32 - Mapeamento das tarefas diretamente nos processadores, sem particionamento prévio. 74

LISTA DE TABELAS

Tabela 1 – Resumo de trabalhos relacionados.....41

Tabela 2 - Ilustração do cálculo para realização do balanceamento de carga de uma aplicação sintética composta por 4 processadores.....58

Tabela 3 – Características de quatro aplicações embarcadas. 73

Tabela 4 – Resultado da minimização do consumo de energia para 30 aplicações sintéticas com o número de comunicações variando entre tarefas e quantidade de *bits* de cada comunicação.....75

Tabela 5 – Resultado da minimização do consumo de energia para 36 aplicações sintéticas obtidas através da combinação de 6 quantidades de tarefas e 6 tamanhos de NoCs. 76

Tabela 6 – Resultado da minimização do consumo de energia para quatro aplicações embarcadas. 77

LISTA DE SIGLAS

ACG	Application Concurrency Graph
ACPM	Application Communication Pattern Model
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BN	Best Neighbor
CAD	Computer-Aided Design
CAFES	Communication Analysis for Embedded Systems
CDCG	Communication Dependence and Computation Graph
CDCM	Communication Dependence and Computation Model
CDG	Communication Dependence Graph
CDM	Communication Dependence Model
CI	Circuito Integrado
CMOS	Complementary Metal-Oxide-Semiconductor
CRG	Communication Resource Graph
CTM	Communication Task Model
CWG	Communication Weighted Graph
CWM	Communication Weighted Model
DAG	Directed Acyclic Graph
DSP	Digital Signal Processing
ECWG	Extended Communication Weighted Graph
ECWM	Extended Communication Weighted Model
FF	First Free
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GSM	Global System for Mobile Communications
HDTV	High-Definition TeleVision
HW	Hardware
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
KB	Kilobytes
MQT	Média de quantidade de tarefas
MNC	Média do número de comunicações
MOEA	Multi-objective Evolutionary Algorithm
MP3	MPEG-1/2 Audio Layer 3
MPEG	Moving Picture Experts Group
MPSoC	Multi-Processor System-on-Chip
MTN	Média de tamanhos da NoC
MVC	Média do Volume de Comunicação
NN	Nearest Neighbor
NoC	Network-on-Chip
NSGA	Non-dominated Sorting Genetic Algorithm
PALOMA	Partitioning Algorithm for MPSoC Automated Design
PDA	Personal Digital Assistant
PE	Processing Element
PL	Path Load
PWM	Pulse Width Modulation
QoS	Quality of Service
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level

SA	Simulated Annealing
SCG	Subsystem Communication Graph
SoC	System-on-Chip
SW	Software
TCG	Task Communication Graph
TG	Grafo de Tarefa
TGFF	Task Graph For Free
TPG	Task Precedence Graph
TS	Tabu Search
TSMC	Taiwan Semiconductor Manufacturing Company
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VOPD	Video Object Plane Decoder
WCET	Worst-Case Execution Time
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 21
1.1 Objetivos 23
1.2 Estrutura do Documento 24
2 REFERENCIAL TEÓRICO 25
2.1 Particionamento e Mapeamento 25
2.2 Definição de Particionamento 26
2.3 Definição de Mapeamento 27
2.4 A Complexidade das Atividades de Mapeamento e Particionamento 27
2.5 Infraestrutura de Comunicação 29
2.5.1 Modelo de Energia para NoC do Tipo Malha 2D e Roteamento XY 32
3 TRABALHOS RELACIONADOS 35
4 METODOLOGIA 43
4.1 Definições de Estruturas de dados 43
4.2 Descrição da Metodologia 44
4.3 Linguagem de Descrição de Entrada 45
5 O *FRAMEWORK* PALOMA 49
5.1 Algoritmo de Particionamento 53
5.2 Algoritmo de Cálculo da Função Custo de Energia 55
5.3 Algoritmo de Cálculo da Função Custo de Balanceamento de Carga 56
5.3.1 Exemplificação da função custo que computa o balanceamento de carga 57
5.4 FERRAMENTA DE MAPEAMENTO (*FRAMEWORK* CAFES) 59
6 EXEMPLIFICAÇÃO DA METODOLOGIA PROPOSTA NESTE TRABALHO USANDO OS *FRAMEWORKS* PALOMA E CAFES 63
7 RESULTADOS 69
7.1 Influência do Particionamento Comparado com o Mapeamento no Consumo de Energia 69

7.2 Influência do particionamento estático como uma etapa prévia do mapeamento dinâmico no consumo de energia 74
8 CONCLUSÕES E TRABALHOS FUTUROS 79
8.1 Contribuições do trabalho 79
8.2 Trabalhos futuros 80
REFERÊNCIAS 81

1 INTRODUÇÃO

É grande a quantidade de aplicações atuais que exige enorme poder computacional, consumo de energia reduzido e comunicação eficiente, o que requer a pesquisa e desenvolvimento de arquiteturas alvos especiais.

Sistemas intrachip, do inglês, Systems-on-Chip (SoCs) são aqueles onde a completa funcionalidade de um sistema é implementada em um único circuito integrado [MAR01]. Estes sistemas têm características que melhoram o desempenho para execução de diversas aplicações. Entre estas características está a comunicação intrachip, que é naturalmente mais rápida e de menor consumo de energia se comparada com a comunicação entre chips.

A necessidade de alto poder de processamento de diversas aplicações impulsiona a pesquisa e o desenvolvimento de sistemas computacionais compostos por vários processadores, atraindo cada vez mais atenção à comunicação em SoCs [BJE06]. O emprego de SoCs também habilita reduzir o tempo de projeto, reduzindo consequentemente o *time-to-market*, que indica a necessidade das indústrias em lançar um produto no mercado no prazo mais curto possível para obter maior número de vendas e, consequentemente, maior lucratividade [RAU94].

MPSoCs (do inglês, *Multi-Processor Systems-on-Chip*) são sistemas compostos por múltiplos processadores implementados na forma de um SoC [JER05]. Sob o ponto de vista do multiprocessamento, um MPSoC é dito homogêneo quando os elementos de processamento que o compõem são de mesma natureza. De outra forma, quando o MPSoC possui elementos de processamento diferentes ele é dito heterogêneo [CAR09].

Enquanto MPSoCs homogêneos tendem a simplificar o uso de técnicas como migração de tarefas, MPSoCs heterogêneos podem dar suporte a uma variedade maior de aplicações. 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 desses 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. celulares, PDAs, HDTVs, *set-top boxes*) já utilizam um sistema multiprocessado em único chip. Devido à grande variedade de aplicações com diversos requisitos e restrições, 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.

As redes intrachip (do inglês, *Networks-on-Chip* ou simplesmente NoCs) consistem em uma arquitetura de comunicação intrachip baseada na adaptação de conceitos bem conhecidos de redes de computadores, sistemas distribuídos e telecomunicação no domínio intrachip.

NoCs são, em geral, infraestruturas de comunicação que têm boa escalabilidade e capacidade de dar suporte a diversas comunicações simultâneas, tornando-as adequadas para tratar aplicações que requerem comunicação intensiva e com grande quantidade de elementos de processamento [DAL01]. Barramentos, em geral, são adequados à comunicação em sistemas com até poucas dezenas de módulos, enquanto NoCs escaláveis podem dar suporte a sistemas com centenas de módulos. Além disso, o paralelismo na comunicação é outro fator que favorece o uso de NoCs, pois enquanto um barramento é uma arquitetura de comunicação que aceita um único escritor por vez (podendo ter um ou mais leitores), a maior parte das topologias de NoC suporta múltiplos escritores e leitores operando simultaneamente.

Seja *tile* uma região limitada da arquitetura alvo com elemento de processamento e mais hardware dedicado à comunicação com elementos posicionados nos 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 (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.

No caso de aplicações de comunicação intensiva, tais como aplicações de fluxo de dados, o gerenciamento da comunicação na arquitetura alvo é fundamental, uma vez que a ocorrência de congestionamentos pode acarretar problemas na transmissão, principalmente em aplicações de tempo real, tais como vídeo e áudio.

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.

1.1 Objetivos

Os objetivos deste trabalho estão na realização de forma rápida e eficiente das atividades de particionamento e mapeamento para o projeto de circuitos integrados. Mais especificamente no particionamento de tarefas de uma aplicação em grupos, sendo estes grupos associados a processadores homogêneos e o mapeamento destes processadores em posições de uma arquitetura alvo do tipo MPSoC baseado em NoC. Sendo estas atividades guiadas por requisitos de redução do consumo de energia e balanceamento de carga, e delimitadas por restrições de máximo consumo de energia, máxima carga de processamento e máximas áreas de dados e código associadas a cada processador. Para alcançar estes objetivos, as seguintes atividades são exploradas:

- Estudo de trabalhos relacionados, valorizando aqueles que exploram infra-estruturas de comunicações do tipo NoC, e focam as atividades de particionamento e mapeamento;
- Análise e compreensão de modelos de aplicações utilizadas em outros trabalhos relacionados. Com estes modelos poderão ser avaliados outros requisitos de projeto além da minimização do consumo de energia, tal como a redução do tempo de execução da aplicação.
- Desenvolvimento de um *framework* que suporte várias ferramentas. Dentre estas ferramentas estão (i) uma ferramenta que realiza o particionamento de tarefas em grupos, e (ii) um gerador automático de aplicação sintética;
- Implementação de algoritmos para reduzir o consumo de energia e balancear carga;
- Pesquisa por aplicações embarcadas onde a atividade de particionamento de tarefas resulte em requisitos utilizados no trabalho proposto. Estas aplicações serão usadas para validar o particionamento e o mapeamento, bem como o modelo de descrição propostos;
- Definição de cenários de teste para avaliar a inclusão dos novos requisitos de projeto na ferramenta de particionamento (*framework* Paloma).

1.2 Estrutura do Documento

Este Capítulo introduziu o trabalho e alguns aspectos conceituais relevantes para a sua compreensão, e o restante deste trabalho está estruturado da seguinte forma: O Capítulo 2 apresenta um referencial teórico em relação ao particionamento, mapeamento, complexidade dessas atividades, arquitetura alvo e infraestrutura de comunicação. Já o Capítulo 3 aborda trabalhos de particionamento de tarefas em grupos e mapeamento destes grupos em processadores. No Capitulo 4 é apresenta a metodologia aplicada para obter resultados de particionamento e mapeamento de aplicações. O Capítulo 5 apresenta o *Framework* PALOMA, desenvolvido no contexto desta dissertação, e também a ferramenta de mapeamento (*Framework* CAFES), que é utilizada para realizar o mapeamento de tarefas ou grupo de tarefas em processadores da arquitetura alvo. O Capítulo 6 exemplifica a metodologia utilizada aqui para melhor compreensão da mesma. O Capítulo 7 traz resultados experimentais de redução do consumo de energia obtidos com particionamento e mapeamento de várias aplicações sintéticas e embarcadas. Por fim, no Capítulo 8, são apresentados as conclusões, contribuições e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Particionamento e Mapeamento

Este trabalho tem como arquitetura alvo MPSoCs homogêneos baseados em NoC, onde a homogeneidade é definida em relação aos processadores e não a NoCs, ou seja, a homogeneidade está nos elementos de processamento e não na arquitetura de comunicação.

O projeto de um MPSoC homogêneo implica em diversas atividades com algumas especificidades de acordo com a natureza de descrição da aplicação e arquitetura alvo. Aqui, são descrito duas atividades de projeto, que são o particionamento de tarefas em grupos e mapeamento tarefas ou grupos de tarefas em processadores.

Aplicações paralelas podem ser descritas com um conjunto de tarefas comunicantes. De acordo com alguns requisitos (e.g. redução do consumo de energia, balanceamento de carga) e algumas restrições (e.g. limite do tamanho da memória, número de processadores alvo), essas tarefas podem ser agrupadas. O agrupamento de todas as tarefas de uma aplicação, que é a atividade de particionamento de tarefa, gera uma partição.

Em uma arquitetura de MPSoC homogêneo baseada em NoC direta, todos os *tiles* contém apenas um processador e todos estes são do mesmo tipo. Assim, sob o ponto de vista de processamento, não há porque escolher processadores para mapear tarefas. Porém, sob o ponto de vista de comunicação, o mapeamento de tarefas comunicante em processadores posicionados proximamente permite reduzir latência, consumo de energia e aumenta a vazão. Assim o mapeamento aqui, é a atividade de associar tarefas ou grupos de tarefas a processadores de forma a tratar o efeito da comunicação nos requisitos e restrições do projeto em vista.

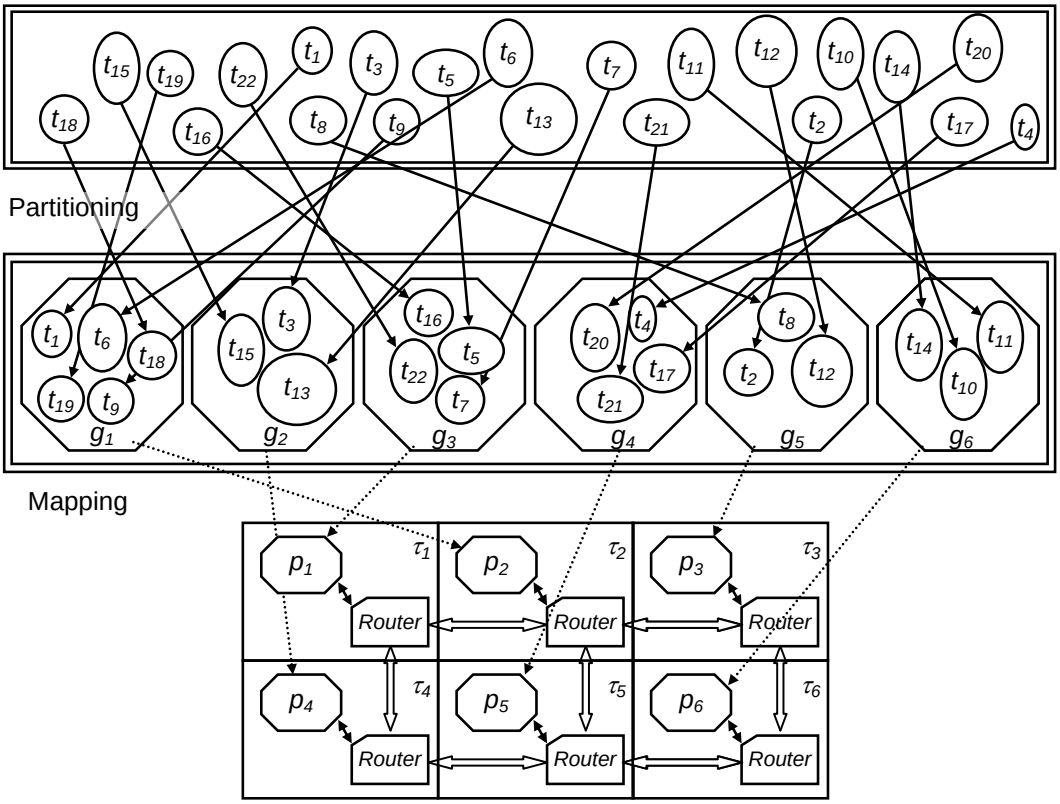


Figura 1 – Particionamento e mapeamento de uma descrição da aplicação (conjunto de tarefas) para uma arquitetura SoC baseada em NoC.

Para entender melhor o conceito de particionamento e mapeamento, a Figura 1 exemplifica o particionamento de uma aplicação hipotética composta por 22 tarefas em 6 grupos que são associados a 6 processadores e um correspondente mapeamento desses processadores em uma arquitetura NoC malha 2D. A aplicação é composta por um conjunto de tarefas paralelas comunicantes *T = {t1, t2, …, t22}*. O particionamento das tarefas, que é representado pelas setas continuas, gera *G = {g1, g2, …, g6}* que é um conjunto de grupos de tarefas.

Finalmente o mapeamento de grupos de tarefas em *tiles* é representada pelas setas tracejadas. Este associa cada elemento G para um conjunto de *tiles* da NoC $\Gamma = \{\tau 1, \tau 2, \ldots, \tau 6\}$. Além disso, cada *tile* contém um único elemento do conjunto de processadores *P = {p1, p2, …, p6}*.

2.2 Definição de Particionamento

Seja $O = \{o_1, o_2, ..., o_n\}$ o conjunto de todos os objetos de um sistema, $b \subseteq O$ um subconjunto de *O* denominado *bloco e* $P = \{b_1, b_2, ..., b_m\}$ uma partição que é o conjunto de todos os blocos de *O, particionamento* é o processo que gera *P*, tal que $b_1 \cup b_2 \cup ... \cup b_n = O$ *e* $b_k \cap b_i = \emptyset \ \forall \ b_k, b_i$ *com* $k \neq i$.

O maior bloco de uma partição ocorre quando todos os objetos que compõem o sistema fazem parte deste bloco. Neste caso, existe apenas um bloco (m = 1) que é o próprio sistema. O menor bloco de uma partição ocorre quando existe apenas um objeto por bloco. Neste caso, o número de blocos é igual ao número de objetos $m=n$. Logo, o número de blocos pode variar de um até o número de objetos que o compõem o sistema.

2.3 Definição de Mapeamento

Seja $X = \{x_1, x_2, ..., x_c\}$ o conjunto de objetos origem e $Y = \{y_1, y_2, ..., y_p\}$ o conjunto de objetos destinos. *Mapeamento* é a função injetora completa $\varphi: X \rightarrow Y$ que associa os objetos do conjunto origem aos objetos do conjunto destinos. Esta associação é chamada de *mapa*.

Para este trabalho os objetos do conjunto origem são as tarefas ou grupos de tarefas da aplicação e os objetos do conjunto destino são os processadores. Assim, o mapeamento é atividade que associa tarefas ou grupos de tarefas a processadores, sendo que cada processador esta posicionado em um *tile* e é conectado à infraestrutura de comunicação por um canal de comunicação. Os grupos de tarefas da aplicação são obtidos de atividades anteriores, tais como o particionamento que pode ser efetuado manualmente.

2.4 A Complexidade das Atividades de Mapeamento e Particionamento

Apesar do fluxo de projeto apresentado aqui utilizar uma abordagem que separa as etapas de particionamento de tarefas em grupos e o mapeamento de tarefas em processadores, é importante deixar claro que várias abordagens usam apenas o mapeamento de tarefas de uma aplicação em processadores, excluindo a atividades de particionamento. No entanto, devido à natureza NP-completo [SHE99] do mapeamento de tarefas em processadores, o resultado obtido com apenas abordagem de mapeamento tende ser pior quando comparado com a abordagem aplicada aqui; especialmente quando a abordagem é realizada em tempo de execução, uma vez que o mapeamento tem pouco tempo para ser concluído.

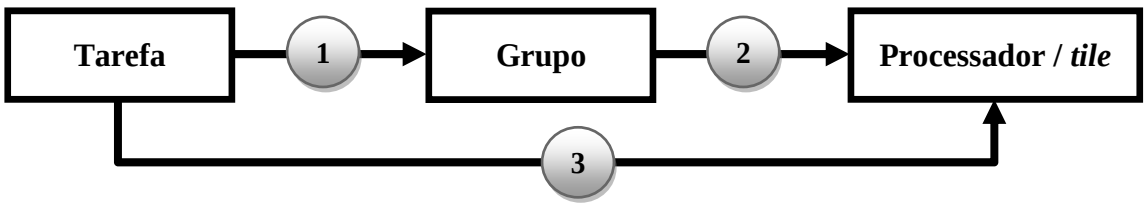


Figura 2 - Fluxo de projeto mostrando o particionamento e mapeamento.

O particionamento de tarefas em grupos (seta 1 da Figura 2) é uma atividade com complexidade proporcional ao número Bell [ZWI96], portanto *O(Bell(n))*, onde *n* é o número de tarefas, visto que não existe relação de ordem entre grupos e nem dentro de um grupo.

O mapeamento do grupo de tarefas em processadores da arquitetura alvo (seta 2 da Figura 2) é uma função injetora (respeita a relação 1→1) e tem complexidade *O(t!)*, onde *t* é o número de *tiles*, pois reflete todas as combinações de posições de processadores em todos os *tiles*.

O mapeamento de tarefas em processadores da arquitetura alvo (seta 3 da Figura 2) agrega as complexidades do particionamento de tarefas em grupos e o mapeamento do grupo de tarefas em processadores da arquitetura alvo. Neste caso, a complexidade é muito maior, pois agrega todas as atividades *O(t! x Bell(n))*.

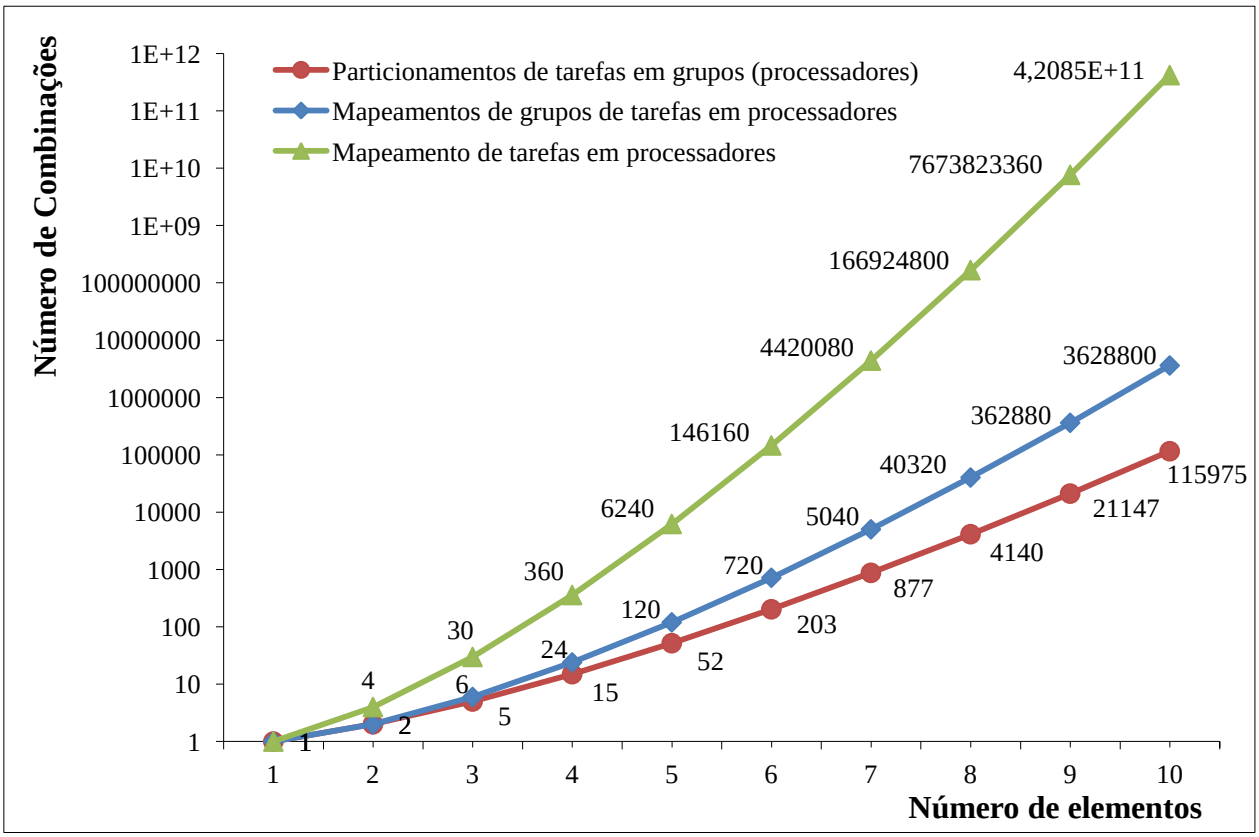


Figura 3 – Crescimento, em escala logarítmica, do número de combinações frente ao número de elementos em relação ao (i) Particionamento de tarefas em grupos, (ii) Mapeamento de grupos de tarefas em processadores e (iii) Mapeamento de tarefas em processadores. Número de elementos refere-se a (i) número de tarefas, (ii) número de processadores e (iii) número de tarefas x processadores.

Para comparar a complexidade das atividades, descritas acima, a Figura 3, mostra o crescimento exponencial do número de combinações de cada atividade. Nesta figura pode ser observado que a quantidade de soluções a serem exploradas com a atividade

de mapeamento de tarefas em processadores é muito superior que as demais. Assim, mesmo aplicando algoritmos heurísticos ou estocásticos os resultados obtidos com esta atividade tendem a ser piores que os obtidos com o fluxo aqui proposto.

2.5 Infraestrutura de Comunicação

Uma infraestrutura de comunicação é um conjunto de elementos que provê meios para o tráfego de informações entre núcleos a ela conectados. Duas infraestruturas de comunicação básicas de MPSoCs: *(i)* barramentos e *(ii)* redes intrachip.

As duas principais vantagens de implementar infraestrutura de comunicação com barramento são: o baixo custo e a extensibilidade. Baixo custo, pois apenas um conjunto de fios é compartilhado por vários dispositivos, enquanto a extensibilidade é dada pela facilidade em acrescentar novos dispositivos ao barramento. Um grande problema desta arquitetura está diretamente relacionado ao número de módulos conectados nela. Quanto mais módulos são conectados, maior a perda na taxa de comunicação entre eles.

Uma rede intrachip é uma infraestrutura de comunicação composta por roteadores (em inglês, *routers*) interconectados por canais de comunicação (em inglês, *links*). A forma como os roteadores estão conectados entre si, e como os núcleos estão conectados aos roteadores, define a *topologia* da rede. Um *roteador* é um dispositivo que transfere informações disponíveis nos seus canais de entrada para os seus canais de saída, através de portas de comunicação. O intervalo de tempo entre a entrada e a saída de uma informação do roteador é denominado de atraso ou *latência* de roteamento. A estrutura de um roteador consiste de um sistema de chaveamento entre os canais de entrada e saída, um módulo de controle de chaveamento, e elementos de armazenamento temporário para os dados dos canais de entrada e/ou de saída. A estrutura genérica de um roteador é ilustrada na Figura 4.

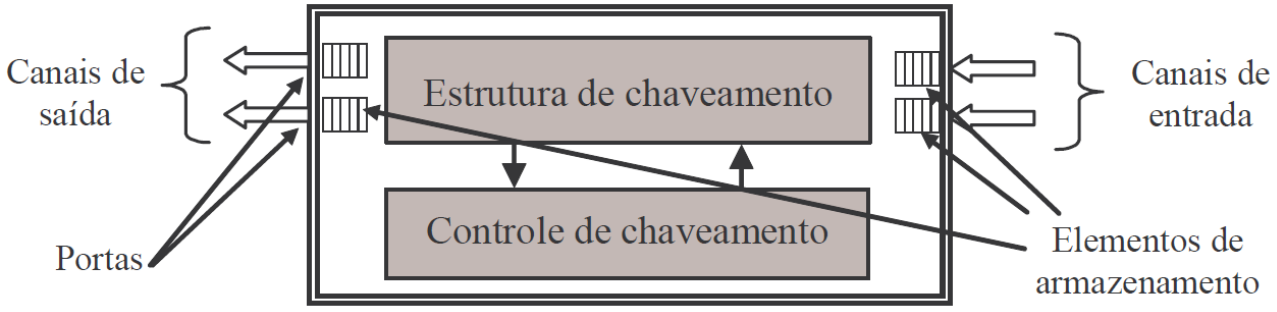


Figura 4 – Estrutura genérica de um roteador [MAR05].

O roteador é o elemento principal da rede intrachip, de forma que este deve ser projetado para não impactar a área final do SoC e ter consumo de energia e tempo de chaveamento que atendam as restrições de projeto. Um dos elementos que mais influencia no consumo de área e de energia é o armazenamento temporário (*buffer*). Sendo então, um elemento primordial a ser considerado no projeto.

Uma infraestrutura de comunicação deve permitir a transferência de informações de um núcleo origem para um núcleo destino. A transferência de informações entre núcleos se da através da troca de mensagens efetivadas através do envio de pacotes. Uma *mensagem* consiste de um conjunto de informações a ser transmitido de um núcleo origem para um núcleo destino. Um *pacote* é a unidade de transmissão de informação empregada pelo meio de comunicação para transmitir mensagens. Pacotes podem conter frações de uma mensagem, uma mensagem inteira ou mesmo múltiplas mensagens.

A Figura 5 ilustra como os pacotes são normalmente constituídos. Em geral o cabeçalho (em inglês, *header*) contém dados úteis para o roteamento, o corpo da mensagem (em inglês, *payload*) contém a(s) mensagem(ens) para núcleos destinos e o finalizador (em inglês, *trailer*) contém dados que garantem a coerência do pacote enviado.

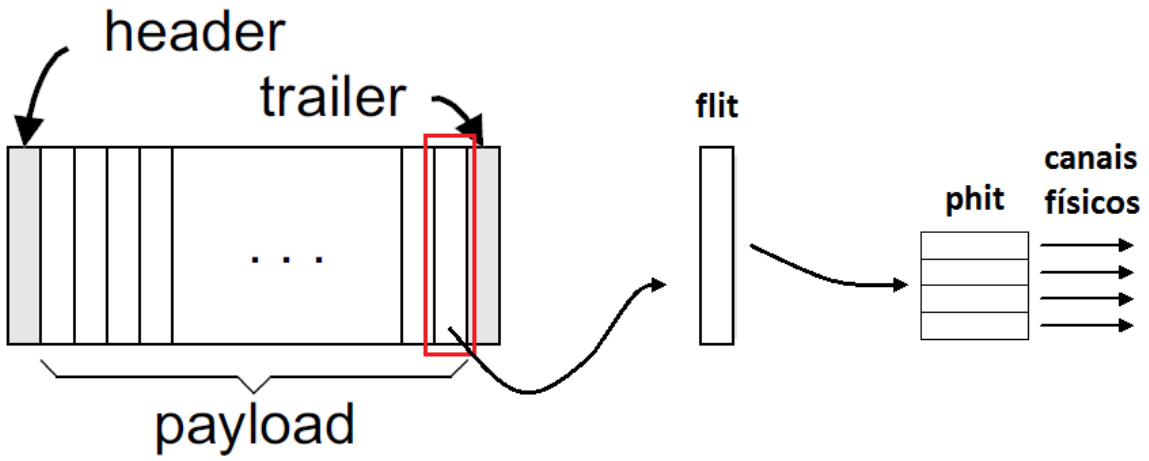


Figura 5 – Formato do pacote. O *flit* corresponde a menor unidade sobre a qual é feita a regulação do tráfego e o *phit* a largura física do canal. Adaptado de [ZEF03]

Para garantir a transferência de mensagens entre núcleos, torna-se necessário impedir que venham a ocorrer fenômenos como *deadlock*, *livelock* e *starvation*. *Deadlock* ocorre quando existe uma dependência cíclica de recursos na rede e as mensagens são paralisadas. Já *livelock* ocorre quando uma mensagem trafega permanentemente pela rede sem chegar ao seu destino. Por fim, *starvation* é a postergação indefinida de acesso a recursos de comunicação.

Topologia de uma rede intrachip consiste na descrição completa do arranjo de interconexões entre elementos de roteamento desta. As topologias podem ser *regulares*, quando é possível definir um padrão deste arranjo com base na estrutura dos elementos de roteamento. Caso este padrão não possa ser identificado, a topologia é denominada *irregular*.

As topologias costumam ser descritas como grafos onde os elementos de roteamento são representados por vértices e as interconexões são representadas por arestas. A Figura 6 ilustra o conceito, mostrando alguns tipos de topologias regulares.

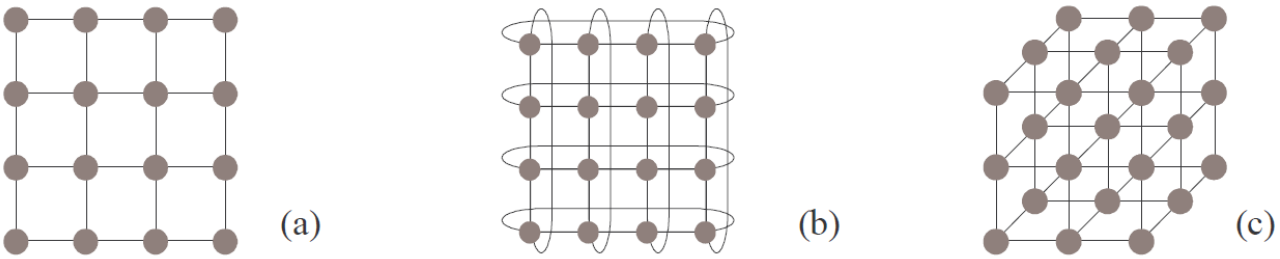


Figura 6 – Exemplo de topologia regulares de rede intrachip: (a) malha 2D (em inglês, *mesh*), (b) toro 2D (em inglês, *torus*) e (c) hipercubo 3D.

Um detalhamento do *tile* de uma rede com topologia malha está ilustrado na Figura 7. Tanto o roteador quanto seu núcleo local (e. g. um elemento de processamento) estão fisicamente posicionados em *tiles*, que são os nodos da rede. O roteador é o ponto de acesso do núcleo à rede. Este, tem a capacidade de direcionar as informações não apenas para o núcleo local ao *tile*, mas também para outros pontos de acesso, sem que os núcleos locais tomem conhecimento da informação transmitida.

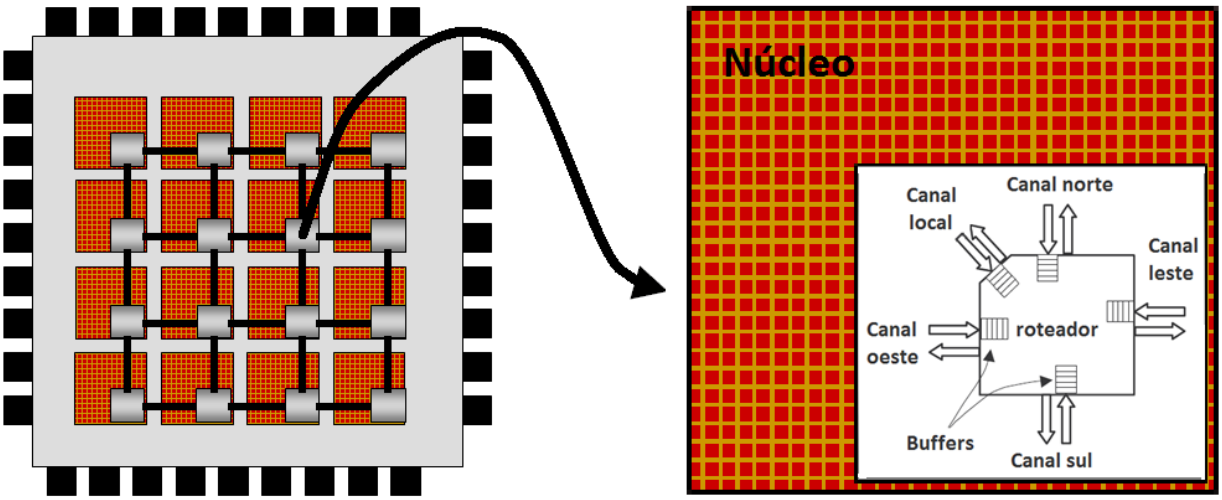


Figura 7 – Exemplo de detalhamento de um *tile* em um SoC que usa como infraestrutura de comunicação uma rede com topologia malha.

O método de chaveamento de pacotes (em inglês, *packet switching*) permite que os pacotes sejam transmitidos sem qualquer necessidade de procedimentos que estabeleçam caminhos prévios. O emprego de chaveamento de pacotes implica o uso de um dos modos de chaveamento definidos a seguir: (i) *store-and-forward* – um pacote tem que ser completamente armazenamento em um roteador antes de ser enviado para o próximo roteador; (ii) *virtual cut-through* – um roteador pode enviar um pacote a partir do momento que o próximo roteador garanta que pode receber todo o pacote; e (iii) *wormhole* – os pacotes são transmitidos entre os roteadores em pequenas unidades, denominadas *flits*. Uma desvantagem associada a este modo é que apenas o *flit* de cabeçalho contém informações sobre o endereçamento destino. Logo, os demais *flits* que compõem o pacote devem seguir o mesmo caminho reservado para o cabeçalho. Se o cabeçalho não puder avançar na rede em função de um congestionamento, todos os *flits* restantes são bloqueados ao longo do caminho, até que este seja liderado.

Quanto ao processo de seleção do caminho de um pacote, ele pode ser: (i) *determinístico*, quando dados os núcleos origem e destino, o caminho entre estes é sempre o mesmo; (ii) *adaptativo*, quando o caminho entre a origem e o destino é determinado por fatores da rede, tais como condições de tráfego. Um exemplo de roteamento determinístico é o XY para topologias malha. Neste roteamento, o pacote percorre um caminho que passa por todas as conexões e roteadores em X até chegar ao endereço em X do roteador destino, então percorre todos os roteadores em Y até chegar ao roteador destino.

Este trabalho utiliza como arquitetura de comunicação NoCs 2D com topologia malha usado algoritmo de roteamento XY e chaveamento *wormhole* determinístico.

2.5.1 Modelo de Energia para NoC do Tipo Malha 2D e Roteamento XY

Processadores, memórias e recursos de infraestrutura de comunicação geram o consumo de energia. A soma da energia consumida pela execução de todas as tarefas agrupadas em um processador permite estimar o consumo de energia neste elemento de processamento. Este valor é usado neste trabalho, juntamente com o volume de comunicação entre as tarefas, para escolher boas partições. Por outro lado, a quantidade de *bits* transmitidos entre as tarefas agrupadas em diferentes processadores contribui para estimar o consumo de energia usada para escolher bons mapeamentos.

A abordagem utilizada aqui para o modelo de consumo de energia da NoC é semelhante ao mostrado em [HU03] e [MUR04]. O consumo de energia dinâmica é

proporcional à atividade de chaveamento, decorrentes de pacotes que atravessam a NoC, dissipando a energia nos links e dentro de cada roteador.

O conceito de *bit* de energia EBit [YE02] é usado para estimar o consumo de energia dinâmica de cada *bit*. O EBit é dividido em três componentes: (i) energia dinâmica consumida pela passagem de um *bit* no roteador (fios, *buffers* e portas lógicas) (ERbit), (ii) energia dinâmica consumidas em conexões horizontais (ELHbit) e verticais (ELVbit) entre os *tiles*, e (iii) energia dinâmica consumida pela passagem de um *bit* em uma conexão entre roteador e processador local (ECbit). A Equação (1) expressa a relação entre estas quantidades, que calcula o consumo de energia dinâmica de um *bit* que passa por um roteador, uma conexão vertical ou horizontal e uma conexão local.

(1) $EBit = ERbit + (ELHbit \text{ or } ELVbit) + ECbit$

ERbit depende da estrutura do roteador, características do *buffer* e tecnologia para estimar quantos *bit-flips* ocorrem para escrever, para ler e para preservar as informações. Elbit é diretamente proporcional à dimensão do *tile*. Para uma NoCs malha 2D regulares com *tiles* quadrados, é razoável considerar que ELHbit e ELVbit têm o mesmo valor, sendo ambos representados apenas por ELBit.

A Equação (2) calcula a energia dinâmica consumida por um único *bit* atravessando uma NoC, a partir do *tile* τ_i para o *tile* τ_j, onde η corresponde ao número de roteadores por onde passa este *bit*.

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

Sendo τ_i e τ_j os *tiles* em que os processadores p_a e p_b, que fazem parte de P (todo conjunto de processadores) são respectivamente mapeados e w_{ab} a quantidade de *bits* da comunicação. Então, a energia dinâmica consumida pela comunicação $p_a \rightarrow p_b$, que faz parte de W (o conjunto de todas as comunicações da aplicação), é dada por $EBit_{ab} = w_{ab} \times EBit_{ij}$.

A Equação (3) dá o valor total do consumo de energia dinâmica da NoC (ENoC), que é calculado para todos os *bits* de todas as comunicações entre os processadores (|*W*|).

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

3 TRABALHOS RELACIONADOS

Este capítulo apresenta inicialmente a análise de propostas no âmbito de modelagem e síntese de sistemas computacionais tendo como foco as atividades de particionamento e mapeamento. Ao final deste é apresentada uma tabela comparando os trabalhos pesquisados com o desenvolvido aqui.

Beux et al. [BEU10] analisam topologias de NoCs que atendem melhor a requisitos de projeto de aplicações embarcadas do tipo fluxo de dados, tendo como atividades de projeto o particionamento da aplicação em módulos de HW e SW e o mapeamento de tarefas nestes módulos. Ambas as atividades são realizadas de forma combinada. Eles usam um modelo de grafo dirigido, onde os vértices, que são tarefas, contêm o tempo de execução em HW e em SW, e também a área em HW. As arestas, por sua vez, contêm o volume de dados transferido entre tarefas e a dependência de dados. Eles utilizam uma arquitetura com diversos módulos em HW e apenas um tipo de processador (SW). A arquitetura tem uma visão hierárquica. No nível superior, esta é vista como um conjunto de nodos que se comunicam via uma NoC. Dentro de cada nodo existe uma nova arquitetura de comunicação, no caso, barramento. Os autores fizeram uma otimização de desempenho em uma aplicação MP3 e obtiveram como resultado uma exploração de desempenho considerando uma solução toda em hardware, uma solução toda em software e uma exploração combinada de mapeamento e particionamento. Os resultados foram dispostos em soluções *Pareto*, de acordo com desempenho de execução, custo de área e flexibilidade. Foi avaliado o impacto do tamanho da infraestrutura de comunicação sob uma rede anel (2, 4 e 8 nodos) e sobre um barramento simples. Para avaliar o impacto de diferentes topologias de rede foi utilizada uma aplicação de codificação de voz GSM. A aplicação inclui 53 tarefas que foram mapeadas sobre uma NoC malha 2D, anel, *crossbar* interconectadas em 16 nodos. Ao final foi avaliado a escalabilidade da metodologia proposta, com uma aplicação MP3 (16 tarefas), um decodificador de voz GSM (34 tarefas) e um codificador de voz GSM (53 tarefas). Elas foram mapeadas em uma NoC malha 2D configurada como 2x2, 3x3, 4x4, 5x5 e 6x6.

Bononi et al. [BON07] comparam quatro topologias NoC, mais especificamente, Anel, Malha 2D, *Spidergon* e *Crossbar* usando o teorema de tráfego uniforme baseado em paradigma requisição/reposta, e utilizaram uma aplicação real Mpeg4 para obter os resultados. As tarefas da aplicação são grafos cujos vértices representam os IPs e as arestas representam a largura de banda de comunicação entre eles.O mapeamento de

cada IP é calculado pela ferramenta de particionamento SCOTCH. Esta ferramenta de particionamento SCOTCH fornece um algoritmo recursivo de bi-particionamento para mapeamento estático de qualquer grafo fonte da aplicação para qualquer grafo alvo da topologia. A principal tarefa do algoritmo de mapeamento consiste em selecionar uma implementação adequada de blocos IP que equilibre a carga de comunicação da NoC. A ferramenta de particionamento SCOTCH aloca de forma recursiva subconjuntos de processos para subconjuntos de processadores (chamados domínios). Os resultados demonstram que as topologias Anel e *Spidergon* surpreendentemente são 3,3% mais rápidas do que a topologia Malha 2D e 6,2% mais rápidas do que a crossbar. Analisando os grafos de vazão por canal observou-se que as NoCs *Spidergon* e Malha 2D exploram o mesmo números de conexões. O algoritmo de roteamento usado na Malha 2D, não explora todos os caminhos mínimos previstos pela NoC. De acordo com mapeamento perto do ideal, *Spidergon* oferece um bom balanceamento de canal enquanto a topologia Anel sofre de congestionamento de canais em relação a outras arquiteturas.

Leupers et al. [LEU10] propõem a ferramenta MAPS (*MPSoC Application Programming Studio*), que tem como objetivo facilitar a programação destinada à MPSoCs heterogêneos e reduzir o intervalo que existe entre a produção e a necessidade de SW para sistemas embarcados. O ambiente fornece um conjunto de algoritmos semi-automáticos para particionamento de códigos seqüenciais e programação em paralelo. Outras funcionalidades são: a abstração de plataforma, tornando a alteração da plataforma alvo mais fácil (*retargetability*); possibilidade de realizar mapeamentos e escalonamentos manualmente; e validação funcional, o que torna mais simples verificações que não exigem implementações. O ambiente obteve resultados positivos em relação às versões anteriores, com aumento de velocidade em 4,1 vezes, sendo que a versão anterior havia obtido uma melhora de 3,61 com o particionamento. Além disso, aplicações paralelas obtiveram maior velocidade e eficiência (9,5 vezes e 47% respectivamente).

Nedjah et al. [NED11] propõem uma ferramenta de projeto multi-objetivo que utiliza uma aplicação descrita como um grafo de tarefas (TG) e um repositório que alimenta os dados no sistema. Eles utilizam dados dispostos em um conjunto de *benchmarks* para síntese de sistemas embarcados (E3S). O E3S é uma coleção de recursos de processadores embarcados e TGs. Os autores utilizam dois algoritmos evolucionários multi-objetivos para as tarefas de mapeamento e particionamento: NSGA-II e microGA. Utilizam também um modelo de grafo dirigido, onde cada vértice representa uma tarefa e

cada aresta representa a taxa de comunicação, largura de banda da comunicação ou volume de *bits* trocados entre as tarefas. A direção das arestas representa a dependência de dados e controle. Os autores usam XML para representar o TG e o repositório de processadores. Para obter os resultados, foi utilizado o *Benchmark* E3S que contém aplicações comuns executados por sistemas embarcados em ambientes como a indústria automobilística, telecomunicações e redes. Estas aplicações são descritas pelo correspondente TGs. Cada um dos nodos nestes TGs está associado a um tipo tarefa. Um tipo de tarefa é uma instrução do processador ou um conjunto de instruções. Alguns exemplos de tarefas usadas nas aplicações do *Benchmark* E3S são FFT, IFFT, PWM, conversão de ângulo para tempo e rotação de imagem. Como aplicação real foi utilizada uma aplicação de segmentação de imagem e comparada com a ferramenta de mapeamento CAFES.

Tsai et al. [TSA10] propõem uma arquitetura híbrida (malha-barramento) para fornecer um ambiente de comunicação de baixa latência para projetos SoC. A idéia básica é utilizar os recursos de comunicação de cada IP para decidir a localização do IP. IPs com tráfego intenso e afinidade de comunicação são colocados no mesmo barramento do subsistema base para evitar hotspots e reduzir a latência de transmissão. A fase de particionamento é baseada em um algoritmo guloso. A entrada é o grafo de comunicação *G (V, E)* e a saída é o SCG (do inglês, Subsystem Communication Graph); No modelo de grafo dirigido *G (V, E),* cada vértice $v_i \in V$ representa um IP e cada aresta $e_{i,j} \in E$ representa a taxa de comunicação de v_i para v_j. A direção das arestas representa a dependência de dados e controle. O peso da aresta $e_{i,j}$, é o requisito de largura de banda de v_i para v_j. Os algoritmos de particionamento e mapeamento são implementados com a linguagem C++. O resultado do mapeamento é implementado com *Verilog* HDL e sintetizado por *Synopsys Design Compiler* usando tecnologia TSMC 0.18*μm* CMOS. O subsistema também é implementado com arquitetura de barramento ARM AMBA. Toda a função é verificada por *Debussy* e *Modelsim*. Para obter resultados foram utilizadas duas aplicações de processamento de vídeo (decodificador MPEG4 e decodificador VOPD) e 4 grafo randômicos criados por TGFF (*Task Graph For Free*). Os resultados mostram que o método proposto pode ajudar a reduzir a latência da comunicação, e pode obter boa porcentagem de melhora da latência à medida que aumenta o tamanho da malha.

Liu et al [LIU10] enfocam o problema de atribuição de tarefas e particionamento de cache L2 para sistemas embarcados. O objetivo é minimizar o WCET (em inglês, *Worst-Case Execution Time*) global do sistema. WCET é uma das mais importantes métricas de

desempenho para sistemas embarcados de tempo real. Os autores utilizam uma técnica de "bloqueio de cache" para garantir um WCET previsível e minimizado para cada tarefa. Também provaram que é um problema NP-Completo e propuseram um algoritmo para realizar a atribuição de tarefas em núcleos e particionamento da cache L2. O principal algoritmo proposto pelos autores é dividido em 3 etapas. A primeira etapa atribui *m* tarefas para *n* núcleos. A segunda etapa particiona a cache L2 em núcleos de tal forma que o WCET total é minimizado para a atribuição da primeira etapa. A última etapa ajusta a atribuição de tarefas geradas pela primeira etapa e partição da cache gerada pela segunda etapa para reduzir ainda mais o WCET total.

Göhringer et al [GOH10] apresentam uma metodologia de projeto para o particionamento de aplicações MPSoCs reconfiguráveis, consistindo de uma arquitetura heterogênea contendo processadores e aceleradores. O particionamento é separado em 3 fases: (i) particionamento SW/SW, particionamento HW/SW e, implementação da arquitetura e integração da aplicação. Em cada fase um conjunto específico de algoritmos e novas ferramentas são usadas para otimizar a etapa de integração do sistema. A arquitetura de comunicação utilizada é genérica, ou seja, não é levada em consideração no particionamento. Três ferramentas foram desenvolvidas e implementadas para automatizar as etapas dentro da metodologia de projeto, para as quais não existem ferramentas comerciais. São elas: (i) *Profiling* para analisar o tempo das diferentes funções; (ii) *Tracing* para gerar o gráfico de chamadas da aplicação, que representa a ordem e a relação entre as funções; (iii) Analisador de comunicação para extrair as dependências de dados entre as diferentes tarefas. A funcionalidade desta metodologia de projeto foi avaliada utilizando um algoritmo complexo de processamento de imagem.

Youness et al [YOU09] apresentam 2 algoritmos, um para escalonamento e otimização de processadores em caso de multiprocessadores homogêneos, e outro para particionamento HW-SW usando diferentes grafos de tarefas no MPSoC. O primeiro reduz o tempo total da aplicação, alocando e organizando corretamente a ordem de execução das tarefas sem violar as restrições de precedência entre elas, reduzindo o número de processadores no sistema alvo. O segundo é dependente da obtenção do melhor particionamento para obter os melhores resultados de desempenho para todo o sistema. A entrada do projeto é dada por um grafo de precedência de tarefa *G (V, E, W, C)*, onde *V* é o conjunto de nodos (tarefas), *E* é o conjunto de arestas (links), *W* é o custo de computação e *C* o custo de comunicação. Para validar os algoritmos, foram obtidos resultados através de escalonamento e particionamento de diferentes grupos de grafos de

tarefas em sistemas alvos diferentes (usando diferente número de processadores e barramentos).

Carvalho et al [CAR10] apresentam heurísticas de mapeamento que visam a redução do congestionamento em MPSoCs baseadas em NoC. As heurísticas tentam reduzir o congestionamento da rede, aproximando tarefas comunicantes e reduzindo a carga do caminho de comunicação entre tarefas. Foram feitas comparações em relação à carga dos canais, ocupação da rede, latência de pacotes, tempo de execução, complexidade dos algoritmos e número de *hops*. Para avaliação das técnicas de mapeamento foi utilizado um ambiente de simulação que utiliza uma NoC descrita em VHDL RTL, enquanto os PEs são descritos em *SystemC* TLM. A NoC utilizada possui topologia malha 2D com o algoritmo de roteamento XY. Primeiramente é feita uma comparação entre as heurísticas de mapeamento dinâmico. Para isto, é utilizado um MPSoC heterogêneo de tamanho 8x8, executando três cenários de teste que variam o tipo de aplicação (i.e. *pipeline*, árvore e genéricas), o número de tarefas de uma aplicação (i.e. de 5 a 10 tarefas), a taxa de injeção de dados, e o tipo de recursos do sistema. A heurística *First Free* (FF) foi utilizada como referência de pior caso. Observou-se que as heurísticas dinâmicas *Nearest Neighbor* (NN), *Path Load* (PL) e *Best Neighbor* (BN) geram resultados médios similares. Os autores também avaliaram o custo do mapeamento dinâmico frente ao mapeamento estático. Para isto são comparadas as heurísticas dinâmicas PL e BN com os algoritmos estáticos *Simulated Annealing* (SA) e *Tabu Search* (TS). A comparação foi realizada através de uma aplicação executando em um MPSoC homogêneo de dimensões 5x4. PL e BN apresentam respectivamente 4 e 3% de aumento do tempo de execução em relação ao algoritmo SA, mostrando o baixo impacto das heurísticas de mapeamento dinâmico no tempo total de execução. A energia consumida na comunicação, PL e BN apresentam 38% mais consumo comparado ao SA. Segundo os autores, o custo das heurísticas de mapeamento dinâmico, em relação ao mapeamento estático, é de 10% na ocupação de canal, de 8,5% em latência, 3,5% em tempo de execução total e 15,5% no consumo de energia de comunicação. Eles argumentam que é um *overhead* aceitável, considerando as vantagens oferecidas pelo mapeamento dinâmico.

Mandelli et al [MAN11] apresentam uma heurística de mapeamento de tarefas dinâmica, nomeada *Lower Energy Consumption Basead on Dependencies-Neighborhood* (LEC-DN). As principais características do trabalho incluem: (i) execução da heurística de mapeamento em um MPSoC modelado em RTL baseado em NoC; (ii) aplicações reais

são usadas como *benchmarks*; (iii) heurística dinâmica considerando apenas uma dependência mestre-escravo; e (iv) a principal função custo da heurística foi a distância em *hops* entre as tarefas comunicantes. Os cenários de teste utilizados foram os seguintes: (A) MPEG-4, VOPD, Aplicação Veicular e Circuito; (B) MPEG-4, VOPD, Segmentação de Imagem e sintética; (C) MPEG-4 e VOPD; e (D) MPEG-4, Aplicação Veicular e Circuito. Os cenários A e B contêm 38 e 36 tarefas, respectivamente. Estes cenários correspondem a uma ocupação do MPSoC igual a 93% (cenário A) e 86% (cenário B). Os cenários C e D contêm 24 e 26 tarefas, respectivamente, permitindo avaliar as heurísticas de mapeamento quando o espaço de busca não é restrito. Para a realização dos cenários de teste foi utilizado como plataforma o MPSoC HeMPS configurado como segue: NoC 7x6 com topologia malha 2D, algoritmo de roteamento XY, tamanho do *flit* de 16 bits, pacotes com tamanho 128 *flits* e controle de fluxo baseado em créditos. O modelo de energia foi calibrado usando tecnologia ST/IBM CMOS 65 *nn* em 1,0 V, adotando *clock-gating*, e uma frequência de relógio de 100 MHz. Os algoritmos de mapeamento avaliados são: *Simulated Annealing* (AS), *Nearest Neighbor* (NN), *Best Neighbor* (BN), e LEC-DN. Nos cenários A e D onde o tempo de execução da aplicação veicular domina, a proposta LEC-DN reduz o tempo de execução para 7,2% e 9,1%, respectivamente, comparado com SA. Comparado com NN e BN o tempo de execução aumenta em média para 3,2% e 1,9%, respectivamente. Os resultados demonstram que a heurística de mapeamento dinâmica não tem impacto no tempo de execução total. O número total de *hops* entre as tarefas comunicantes é a métrica usada para avaliar a qualidade do mapeamento. É possível verificar nos cenários B, C e D, que a proposta LEC-DN reduz o número de *hop* comparado com SA (108/102, 56/49 e 115/93 *hops*, respectivamente). Nos cenários C e D, com mais recursos livres, todas as heurísticas reduzem o número total de *hop* comparado com SA. Esses resultados mostram que LEC-DN permite explorar os recursos disponíveis de uma maneira eficiente, gerando soluções otimizadas. Foi comparado também o consumo total de energia, onde a proposta LEC-DN consome em média 7,1% a mais de energia comparado com SA. Comparando LEC-DN com NN e BN, a redução média de energia é de 11,4% e 10,4% respectivamente. No cenário D a redução de energia da proposta LEC-DN comparado com NN foi de 22,8%.

A Tabela 1 relaciona os trabalhos acima descritos com o trabalho aqui desenvolvido. Os aspectos aqui considerados relevantes são: (i) arquitetura de comunicação; (ii) momento da realização do particionamento e ou mapeamento; (iii) objetivo do trabalho; e (iv) modelo ou linguagem usado para descrição da aplicação.

Tabela 1 – Resumo de trabalhos relacionados

Autores	Ano	Tipo MPSoC	Arquitetura de comunicação	Atividade / Dinamicidade	Objetivo	Modelo / Linguagem
Beux et al.	2010	Heterogêneo	NoC	Particionamento e mapeamento / estático	Reduzir tempo de execução e diminuição de área	DAG (Grafo de tarefas acíclico)
Bononi et al.	2007	Heterogêneo	NoC	Particionamento e mapeamento / estático	Minimizar latência e maximizar vazão	TG (Grafo de tarefas)
Leupers et al.	2010	Heterogêneo	Genérica	Particionamento e mapeamento / estático	Balanceamento de carga	ACG (Grafo de aplicação concorrente)
Nedjah et al.	2011	Homogêneo	NoC	Particionamento e mapeamento / estático	Reduzir tempo de execução e consumo de energia	DAG (Grafo de tarefas acíclico)
Tsai et al	2010	Homogêneo	NoC	Particionamento e mapeamento / estático	Redução da latência de comunicação	SCG (Grafo de Comunicação de subsistema)
Youness et al.	2009	Homogêneo	Barramento	Escalonamento e particionamento / estático	Reduzir tempo de execução	TPG (Grafo de precedência de tarefas acíclico)
Göhringer et al.	2010	Homogêneo	Genérica	Particionamento e mapeamento / estático e dinâmico	Reduzir tempo de execução	C/C++
Liu et al.	2010	Heterogêneo	Genérica	Particionamento / estático	Reduzir tempo de execução	Não informado
Carvalho et al.	2010	Heterogêneo	NoC	Mapeamento / estático e dinâmico	Reduzir o congestionamento da rede	TG (Grafo de tarefas)
Mandelli et al.	2011	Homogêneo	NoC	Mapeamento / dinâmico	Reduzir o consumo de energia	Diagrama UML hierárquico
Este trabalho	**2011**	**Homogêneo**	**NoC**	**Particionamento estático**	**Redução do consumo de energia, balanceamento de carga.**	**XML, CWG (Grafo de tarefas com pesos de comunicação)**

4 METODOLOGIA

Este Capítulo descreve a metodologia aplicada para obter resultados de particionamento e mapeamento de aplicações, bem como um caso exemplo para compreensão da metodologia usada.

4.1 Definições de Estruturas de dados

O particionamento e o mapeamento utilizam três estruturas de dados principais que são definidas a seguir:

Definição 1: Um grafo de comunicação de tarefas (TCG - *Task Communication Graph*) é um grafo dirigido $<T, V>$. O conjunto de vértices $T = \{t_1, t_2, \ldots, t_m\}$ representa o conjunto de m tarefas em uma aplicação paralela. Assumindo que v_{ab} é a quantidade de *bits* de todos os pacotes enviados da tarefa t_a para a tarefa t_b, então o conjunto de arestas V é $\{(t_a, t_b) \mid t_a, t_b \in T \text{ e } v_{ab} \neq 0\}$, e cada aresta é rotulada com o valor v_{ab}. V representa todas as comunicações entre as tarefas da aplicação.

Definição 2: Um grafo de peso com comunicação (CWG - *Communication Weighted Graph*) é um grafo dirigido $<P, W>$, onde o conjunto de vértices $P = \{p_1, p_2, \ldots, p_n\}$ representa o conjunto de processadores da arquitetura alvo, e n é igual ao número total de *tiles*. Além disso, w_{ab} é o número total de *bits* transmitidos do processador p_a para o processador p_b. O conjunto de arestas W é $\{(p_a, p_b) \mid p_a, p_b \in P \text{ e } w_{ab} \neq 0\}$, e cada aresta é rotulada com o valor w_{ab}. W representa todas as comunicações entre os processadores do MPSoC.

A modelagem da arquitetura alvo (NoC malha 2D usado o algoritmo de roteamento XY e chaveamento *wormhole* determinístico) é feita pelo grafo de recursos de comunicação indicado.

Definição 3: Um grafo de recurso de comunicação (CRG - *Communication Resource Graph*) é um grafo dirigido $<\Gamma, L>$, onde o conjunto de vértices é o conjunto de *tiles* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ e o conjunto de aresta $L = \{(\tau_i, \tau_j), \forall \tau_i, \tau_j \in \Gamma\}$ contém o conjunto de caminhos de τ_i para τ_j. O valor de n é novamente o número total de *tiles* e é igual ao produto de linhas e colunas da NoC. As arestas e vértices do CRG representam as conexões físicas e roteadores da arquitetura alvo, respectivamente. As definições do CRG são equivalentes ao grafo de caracterização da arquitetura em [HU03] e ao grafo da topologia da NoC em [MUR04].

4.2 Descrição da Metodologia

A avaliação do efeito do particionamento de forma isolada ou em conjunto com o mapeamento foi feito através de duas abordagens: (i) uma que aplica o particionamento de tarefas em grupos e depois mapeia estes grupos em processadores; e outra que (ii) mapeia diretamente tarefas em processadores da arquitetura alvo.

A Figura 8 ilustra a metodologia utilizada aqui para avaliar a abordagem que explora as atividades de particionamento e mapeamento, conjuntas. O particionamento de tarefas em grupos tem como entrada: (i) uma lista de processadores, que tem o nome e o número de todos os processadores, permitindo calcular quantos grupos terá uma partição; (ii) descrição da aplicação, que tem todas as tarefas e as suas comunicações (TCG); (iii) requisitos da aplicação tal como a minimização da área ocupada ou a redução do consumo de energia; (iv) restrições do sistema, tal como o número limite de tarefas agrupadas em um mesmo processador, evitando que o mesmo seja sobrecarregado em processamento, energia, áreas de dados e de código; e (v) parâmetros de energia da NoC, que são usado para calcular o consumo de energia para cada partição.

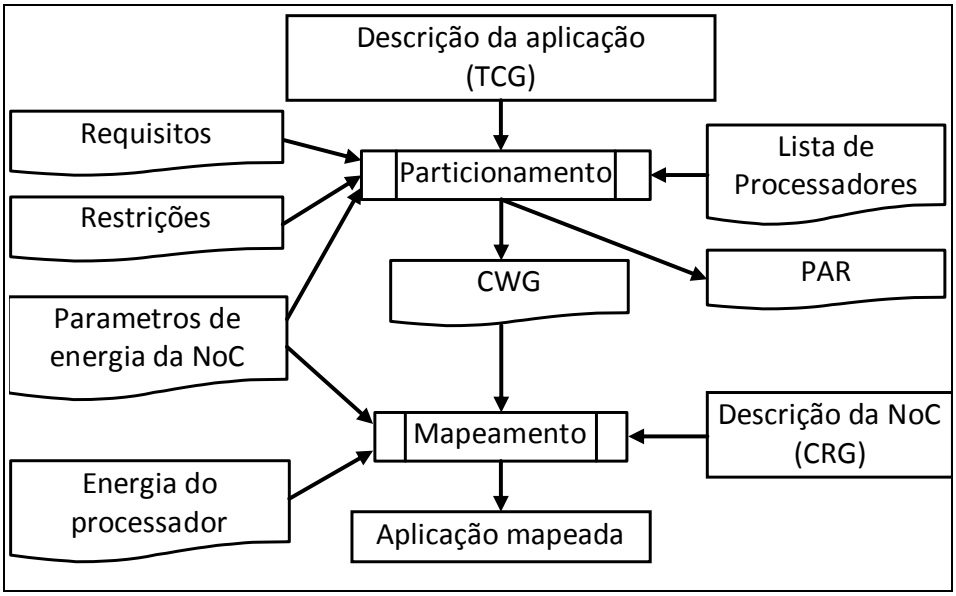


Figura 8 – Fluxo de projeto ilustrando as atividades de particionamento e mapeamento.

A atividade de particionamento tem como saídas: (i) um relatório (PAR) descrevendo os grupos de tarefas gerados e o custo de particionamento final; e (ii) um grafo CWG, contendo o volume de comunicação entre os grupos de tarefas.

Para a abordagem que tem o mapeamento de grupos de tarefas, esta atividade tem as seguintes entradas: (i) os mesmos parâmetros de energia da NoC descritos na atividade de particionamento; (ii) o CWG que descreve a aplicação em termos de volume

de comunicação entre tarefas; (iii) a descrição da NoC através do grafo CRG; e (iv) a energia consumida por cada processador. Esta última entrada não é usada para decidir o mapeamento, mas apenas para obter um resultado de consumo de energia final que considera não apenas o consumo na arquitetura de comunicação, mas também nos processadores.

A saída do mapeamento (Aplicação mapeada) é um arquivo contendo todas as associações de tarefas/grupos de tarefas em processadores - *tile*, que implica um consumo de energia mínimo de todos os mapeamentos avaliados.

No modelo atual, o custo de energia é proporcional ao custo de comunicação, visto que para minimizar o consumo de energia deve ser minimizada a quantidade de *bits* que trafega na rede.

4.3 Linguagem de Descrição de Entrada

Para realizar o particionamento, foi implementada uma ferramenta que tem como entrada uma aplicação descrita em XML (em inglês, *extensible markup language*). Os principais propósitos de usar XML são: (i) a facilidade de compartilhamento de dados por diferentes ferramentas de projeto; (ii) a facilidade de documentar a aplicação; e (iii) a extensibilidade da linguagem, uma vez que esta permite que sejam adicionados mais *tags* ao arquivo de texto sem prejudicar a estrutura dos dados já especificados. Desta maneira, outras ferramentas podem compartilhar o mesmo XML, tornando um formato intermediário mais eficiente. Além do mais, arquivos XML são normalmente menos restritivos do que formatos de documentos proprietários [SAL09].

Na descrição da aplicação (TCG) 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 um dado processador é imprescindível para realizar balanceamento de carga. Sendo assim, a aplicação e a arquitetura alvo são descritas através de três *tags* XML ilustradas na Figura 9:

- TARGET_ARCHITECTURE – contém uma lista de tipo de processadores e uma lista de processadores que pertencem a cada tipo de processador;
- APPLICATION_CHARACTERIZATION – caracteriza as tarefas da aplicação para os tipos de processadores disponíveis na arquitetura alvo;
- APPLICATION_DESCRIPTION – caracteriza a aplicação quanto ao volume

de comunicação entre as tarefas.

```
<SYSTEM_SPECIFICATION>
   <TARGET_ARCHITECTURE>
      <PROCESSOR_TYPE_LIST>…</PROCESSOR_TYPE_LIST>
      <PROCESSOR_LISTS>…</PROCESSOR_LISTS>
   </TARGET_ARCHITECTURE>
   <APPLICATION_CHARACTERIZATION>
      <TASK_LIST>…</TASK_LIST>
   </APPLICATION_CHARACTERIZATION>
   <APPLICATION_DESCRIPTION>
      <COMMUNICATION_TASK_LIST>…</COMMUNICATION_TASK_LIST>
   </APPLICATION_DESCRIPTION>
</SYSTEM_SPECIFICATION>
```

Figura 9 – Esqueleto de uma especificação de plataforma e aplicação MPSoC em formato XML.

Dentro da *tag* TARGET_ARCHITECTURE, o primeiro campo (PROCESSOR_TYPE_LIST) contém uma lista de tipos de processadores, onde cada processador da lista é descrito por características físicas, tais como dimensões e freqüência. No caso de um MPSoC homogêneo - foco deste trabalho, apenas uma entrada é necessária. Porém, para facilitar extensões futuras da linguagem de entrada, este campo está sendo planejado para dar suporte a arquiteturas heterogêneas A Figura 10 apresenta um exemplo de um formato aceito pelo campo PROCESSOR_TYPE_LIST, onde está contida uma lista com dois tipos de processadores.

```
<PROCESSOR_TYPE_LIST>
   <PROCESSOR_TYPE type="MIPS">
      <FEATURES frequency="1200" width="1.2" height="2.5"/>
   </PROCESSOR_TYPE>
   <PROCESSOR_TYPE type="PowerPC">
      <FEATURES frequency="3200" width="3.5" height="4.2"/>
   </PROCESSOR_TYPE>
</PROCESSOR_TYPE_LIST>
```

Figura 10 – Descrição exemplo de campos da tag PROCESSOR_TYPE_LIST, contendo uma lista de tipos de processadores com suas características físicas. A figura ilustra dois tipos de processadores contendo suas freqüências de operação e suas dimensões. Por exemplo, no caso do processador MIPS, este opera a 1,2GHz, tem como largura 1,2mm e altura 2,5mm.

Na Figura 11, é identificada a lista de processadores que pertencem a cada tipo de processador (PROCESSOR_LISTS). Esta lista é utilizada para informar quantos conjuntos terá a partição. Além disto, os nomes aqui descritos podem ser utilizados para a atividade de mapeamento, onde os processadores ficam posicionados fisicamente em

tiles específicos.

```
<PROCESSOR_LISTS>
        <PROCESSOR_TYPE type = "MIPS">
                <LIST> p1 p2 </LIST>
        </PROCESSOR_TYPE>
        <PROCESSOR_TYPE type = "PowerPC">
                <LIST> p3 p4 p5 p6 </LIST>
        </PROCESSOR_TYPE>
</PROCESSOR_LISTS>
```

Figura 11 – Campo que contém, para cada tipo de processador, uma lista de identificadores (nomes) de processadores deste tipo. A figura ilustra dois tipos de processadores. Por exemplo, os nomes p1 e p2 identificam processadores do tipo MIPS.

A Figura 12 exemplifica caracterizações de tarefas da aplicação para os tipos de processadores disponíveis na arquitetura alvo. Esta caracterização é definida pela *tag* TASK_LIST. As tarefas são caracterizadas em termos de: (i) potência média dissipada pelo processador (*power*), quando este executa a tarefa; (ii) áreas de dados (*data*) e de código (*code*), obtidas pela compilação da tarefa no sistema operacional que estará executando no processador; e (iii) percentual de processamento requerido pela tarefa (*cpuUse*).

```
<TASK_LIST>
   <TASK id="T1">
      <PROCESSOR_TYPE type="PowerPC" power="20.12" data="3865" code="2793" cpuUse="34.5"/>
      <PROCESSOR_TYPE type="MIPS" power="13.5" data="22" code="334" cpuUse="24.6"/>
   </TASK>
   <TASK id="T2">
      <PROCESSOR_TYPE type="PowerPC" power="26" data="86" code="49" cpuUse="23.7"/>
      <PROCESSOR_TYPE type="MIPS" power="30" data="2458" code="8368" cpuUse="29.3"/>
   </TASK>
</TASK_LIST>
```

Figura 12 – Caracterização de tarefas da aplicação frente aos tipos de processador disponíveis. A figura ilustra um exemplo sintético de caracterização de duas tarefas (T1 e T2) em dois processadores distintos (MIPS e PowerPC). Por exemplo, a tarefa T1 quando executada no processador PowerPC dissipa 20.12 uW (micro watts) de potência, ocupa 3865 KB (kilobytes) de área de dados e 2793 KB de área de código, e requer 34.5% de processamento. Esta mesma tarefa executada em um processador MIPS dissipa 13.5 uW de potência, ocupa 6422 KB de área de dados e 334 KB de área de código, requerendo 24.6 % de processamento.

Por fim, a aplicação também é caracterizada pelo volume de comunicação entre as tarefas. Esta caracterização é obtida com a *tag* COMMUNICATION_TASK_LIST, tal como ilustrado no trecho sintético de descrição XML da Figura 13. Nela estão definidas a identificação da tarefa que origina e a que recebe a comunicação, além da quantidade de

bytes da comunicação em *kB* (*kilobytes*).

```
<COMMUNICATION_TASK_LIST>
   <SOURCE_TASK source = "T1">
      <COMMUNICATION target = "T2" volume = "250"/>
      <COMMUNICATION target = "T3" volume = "320"/>
   </SOURCE_TASK>
   <SOURCE_TASK source = "T2">
      <COMMUNICATION target = "T1" volume = "50"/>
      <COMMUNICATION target = "T3" volume = "12.5"/>
   </SOURCE_TASK>
</COMMUNICATION_TASK_LIST>
```

Figura 13 – Descrição da aplicação em relação à inter-comunicação de suas tarefas. Aqui é descrito um trecho sintético contendo duas tarefas que originam a comunicação (T1 e T2) e 3 tarefas que recebem as comunicações (T1, T2 e T3). Por exemplo, existe uma comunicação que parte da tarefa T1 em direção à tarefa T2 com o volume de comunicação igual a 250 kB.

5 O *FRAMEWORK* PALOMA

Este capítulo descreve o *framework* PALOMA, do inglês, *Partitioning Algorithm for MPSoC Automated Design*, que é utilizado para automatizar a geração de aplicações paralelas sintéticas e também particionar tarefas em grupos de tarefas mapeáveis em processadores do MPSoC alvo.

A Figura 14 ilustra a interface principal do *framework*. O primeiro conjunto de parâmetros servem para caracterizar a NoC (NoC *Characterization*). Estes são divididos em três grupos: (i) parâmetros de topologia da NoC (*Topology Parameters*), (ii) parâmetros de consumo *de* energia da NoC (*Energy Parameters*), e (iii) parâmetros para definir a freqüência de operação e o número de ciclos necessários para realizar transferência de informação entre roteadores (*Timing Parameters*).

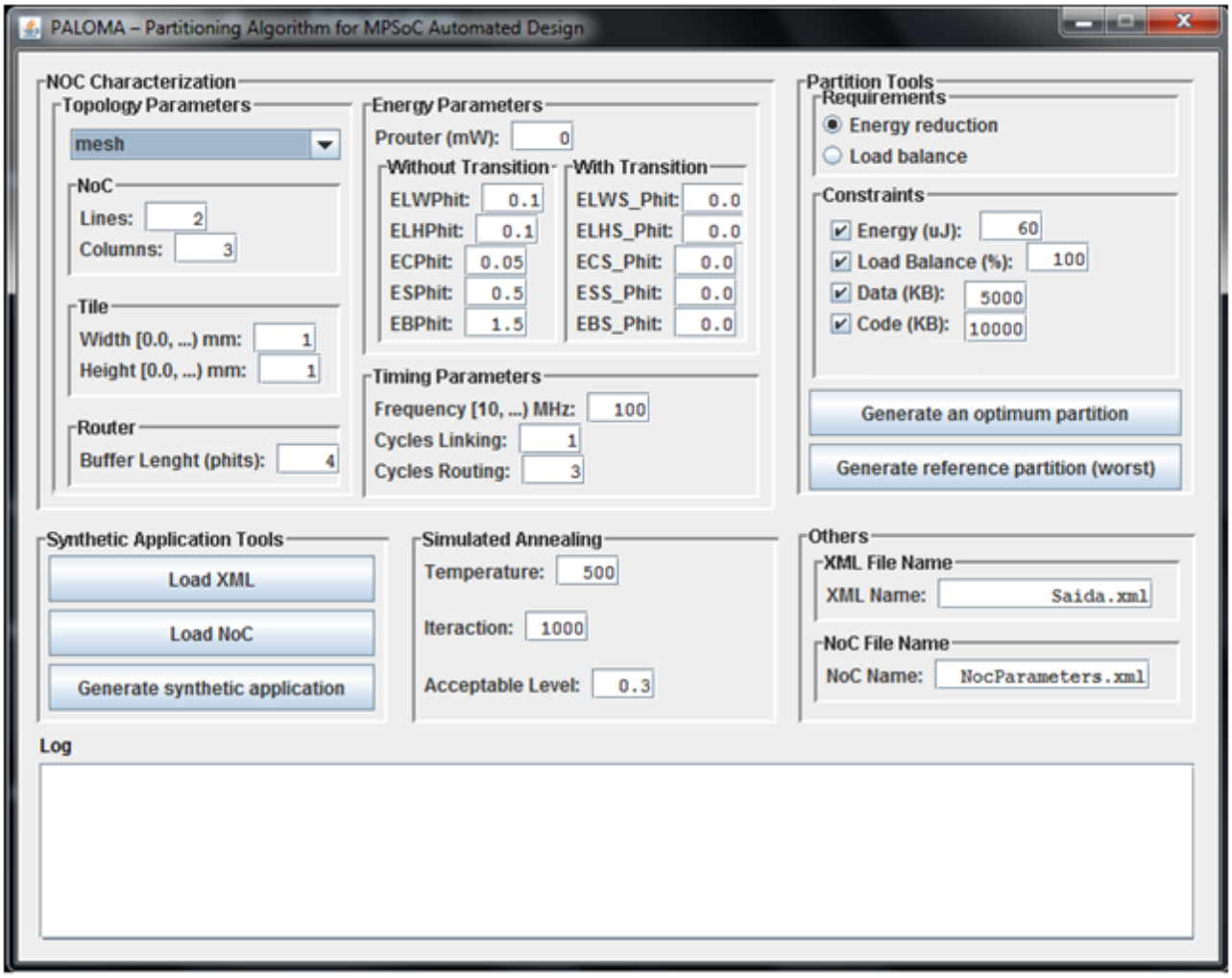


Figura 14 – Interface de abertura do PALOMA.

Quatro parâmetros definem a infraestrutura de comunicação:

1. *Topology Parameters* – Topologias de NoCs cujos modelos estão disponíveis no

framework: (i) *Mesh* - topologia malha 2D, roteamento XY e chaveamento *wormhole*; (ii) *Torus* - topologia toro 2D, roteamento XY e chaveamento *wormhole*;

2. NoC (*lines*, *columns*) – Número de *tiles* da Noc em linhas e colunas;
3. *Tile* (*width*, *height*) – Largura e altura dos *tiles* em *mm*. Permite estimar o comprimento das conexões entre roteadores;
4. *Router* (*Buffer length*) – Número de elementos que compõem as áreas de armazenamento temporário das entradas de cada roteador. A largura de cada área de armazenamento temporário é de um *phit*;

O segundo conjunto de itens contém parâmetros que permitem estimar o consumo de energia das NoCs disponibilizadas:

1. *Without transitions* – Para modelos computacionais que consideram transição de *bits*, estes campos representam os consumos de energia na ausência de transições entre *phits* consecutivos. Para os demais modelos, estes campos representam consumos médios de energia para *phits*. Os campos são: (i) ELWPhit (conexão horizontal) e ELHPhit (conexão vertical) - energia dinâmica de um *phit* consumida em uma conexão entre roteadores. Para NoCs retangulares, ELWPhit e ELHPhit são iguais e equivalem a ELbit multiplicado pelo número de *bits* de um *phit*; (ii) ECPhit - energia dinâmica de um *phit* consumida em uma conexão entre um roteador e uma núcleo local ao *tile*. Equivale ao ECbit multiplicado pelo número de *bits* de um *phit*; (iii) ESPhit - energia dinâmica de um *phit* consumida no circuito de controle de um roteador. Equivale ao ESbit multiplicado pelo número de *bits* de um *phit*; (iv) EBPhit - energia dinâmica de um *phit* consumida por um elemento do circuito de armazenamento. Equivale ao EBbit multiplicado pelo número de *bits* de um *phit*.
2. *With Transitions* – Parâmetros usados apenas nos modelos que consideram transição de *bits*. Representa os consumos de energia quando ocorrer transição entre *bits* de *phits* consecutivos considerando agora transições existem as seguintes equivalências: (i) ELWS_Phit e a ELWPhit; (ii) ELHS_Phit e ELHPhit; (iii) ECS_Phit e ECPhit; (iv) ESS_Phit e ESPhit; (v) EBS_Phit e EBPhit.
3. *PRouter* – Potência estática dissipada por todos os elementos ativos de um roteador, somado com a potência dinâmica dissipada pelos circuitos do roteador que operam mesmo frente a ausência de tráfego.

Por fim são descritos os parâmetros que permitem modelar o comportamento temporal da NoC:

1. *Frequency* – Frequência de operação da NoC (considerando uma NoC síncrona),

utilizado para determinar o tempo de execução da aplicação e o consumo de energia estática de toda rede, e dinâmica dos circuitos que operam mesmo frente a ausência de tráfego;

2. *Cycles Linking* – Número de ciclos de relógio necessário para realizar a transmissão de um *phit* de uma conexão entre roteadores ou entre roteador e núcleo local;
3. *Cycles Routing* – Número de ciclos de relógio para rotear um pacote de um canal de entrada para um canal de saída de um roteador.

Dentro de System Application Tools da Figura 15 encontram-se os botões: (i) *Load* XML que carrega aplicações descritas em XML; (ii) *Load* NoC que carrega parâmetros da NoC previamente salvos em arquivo e (iii) *Generate synthetic application* que abre uma nova interface gráfica para poder descrever classes de aplicações, que embora sintéticas, pudessem ter comportamento similar ao de aplicações embarcadas. Para tanto, boa parte dos parâmetros são expressos em termos de distribuições Gaussianas. A Figura 15 ilustra a interface gráfica, contendo parâmetros que permitem definir características da aplicação e da arquitetura alvo (MPSoC homogêneos).

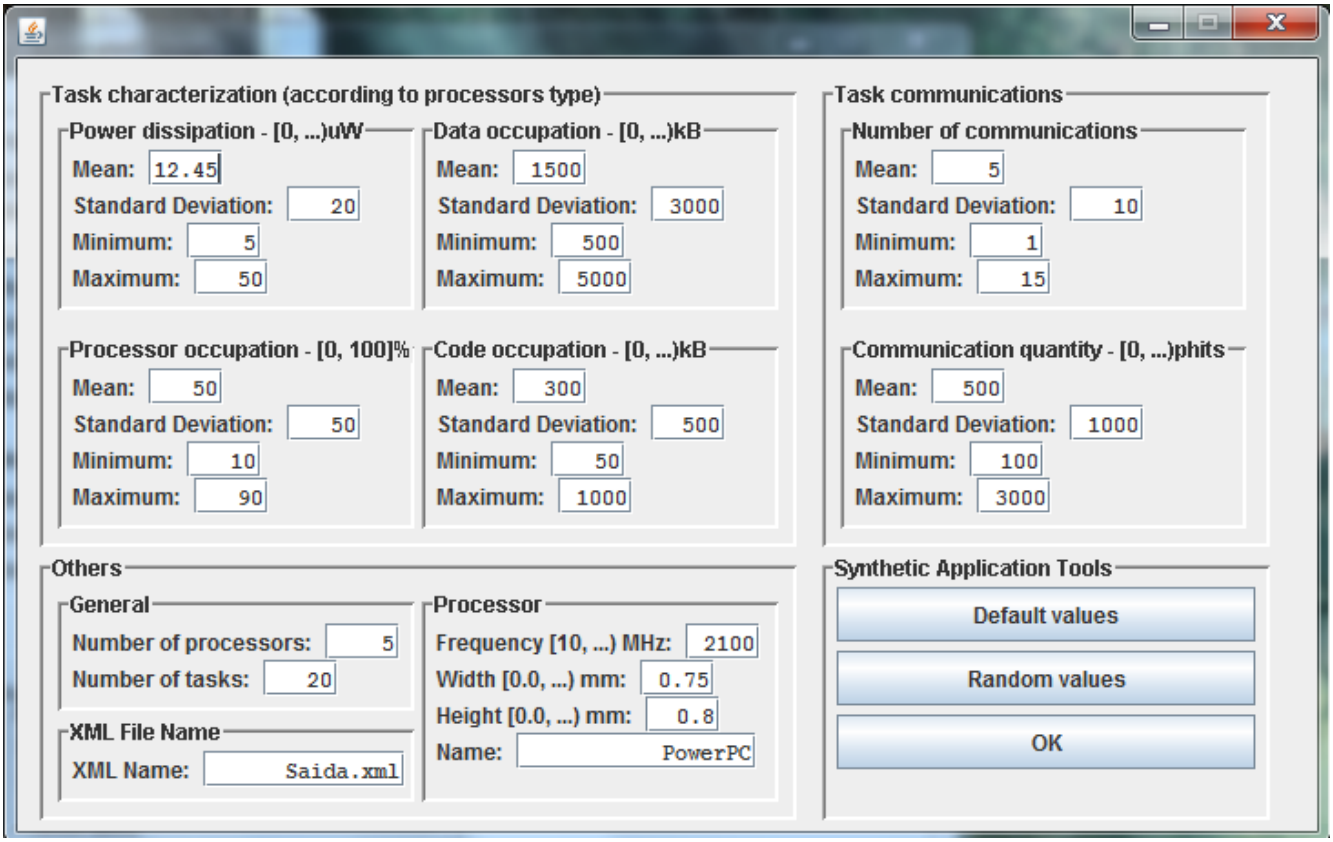


Figura 15 – Interface gráfica da ferramenta de geração de aplicações sintéticas. Os valores aqui apresentados são meramente ilustrativos.

Os parâmetros usados para especificar o MPSoC são: (i) *Number of processors*, que contém o número absoluto de processadores presentes no MPSoC alvo, considerando um processador por *tile*; (ii) *Processor*, contendo características do tipo de processador. As características aqui descritas são: *Frequency* - frequência de operação (em *MHz*), *Width* - largura física do processador (em *mm*), *Height* - altura física do processador (em *mm*) e *Name* - nome do tipo de processador (um identificador).

A aplicação é definida pelos parâmetros: (i) *Number of tasks*, que contém o número total de tarefas da aplicação a ser sintetizada; (ii) *Task characterization*, contendo quatro sub campos que caracterizam as tarefas quando executadas na arquitetura alvo; e (iii) *Task communication*, que contém dois sub campos com o informações sobre a comunicação entre as tarefas da aplicação.

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

Todos os campos contendo média, desvio padrão e intervalo (valor máximo e mínimo) produzem valores aleatórios com probabilidade que respeita uma distribuição Gaussiana. Uma distribuição Gaussiana permite que a aplicação sintética gerada possa ser direcionada para uma determinada classe de aplicações, tal como *IO-bounded* ou *CPU-bounded*. O objetivo aqui é que ao analisar uma aplicação, mesmo que sintética, o projetista possa ter idéia de como o particionamento iria se comportar como uma aplicação real, uma vez que esta fizesse parte da mesma classe de aplicações.

Após o preenchimento de todos os campos dispostos na interface, é possível gerar uma aplicação sintética - através do botão *OK*. Esta aplicação estará em um arquivo de texto no formato de XML. É possível também preencher os campos automaticamente com valores padrões - pressionando o botão *Default values*, ou então gerar aleatoriamente esses valores - através do botão *Random values*.

Uma vez tendo gerada a aplicação sintética, o projetista pode particionar a mesma através da configuração do conjunto de itens localizado dentro de *Partition Tools*, que contem: (i) requisitos que pretende explorar, como por exemplo, redução do consumo de energia ou balanceamento de carga e, (ii) restrições, como por exemplo, limitar o número de tarefas agrupadas em um mesmo processador, objetivando minimizar ou maximizar a

função custo de particionamento. Ou seja, objetivando obter particionamentos ótimos (aqueles com menor custo de particionamento) ou péssimos (aqueles com maior custo de particionamento). Estas partições podem ser obtidas através dos botões *Generate an optimum partition* e *Generate reference partition (worst)*, respectivamente.

5.1 Algoritmo de Particionamento

O problema de particionar tarefas tem natureza NP-completa (como visto na Seção 2.4), o que impede a busca por soluções exaustivas. Neste caso, algoritmos alternativos devem ser explorados, tais como algoritmos heurísticos (e.g. *Largest Communication First*, *Incremental*) ou estocásticos (e.g. *Simulated Annealing*, *Tabu Search*). Por outro lado, problemas desta natureza apresentam auto-similaridade, quando as possíveis partições são relacionadas com os resultados da função custo de particionamento. Assim, são diversas as partições que têm o mesmo custo. Desta forma, algoritmos que conseguem explorar soluções (partições) bastante distintas, mesmo que randomicamente geradas, podem alcançar soluções próximas das ótimas, se estes aplicarem uma etapa de refinamento para cada partição gerada.

A Figura 16 apresenta a plotagem do particionamento de uma aplicação sintética com 6 tarefas, tendo como avaliação o consumo de energia de cada partição. A partição 1 tem consumo de energia igual a zero, pois todos os tarefas estão em um único processador. É evidente que estas partições podem não ser válidas, pois não respeitam as restrições da aplicação como, por exemplo, carga máxima no processador. Esta, por exemplo, podem impossibilitar que a partição 1 fizesse parte do espaço de soluções. Na figura, a auto similaridade pode ser vista nos pontos circulados, que mesmo bem espaçados implicam em soluções com custos iguais.

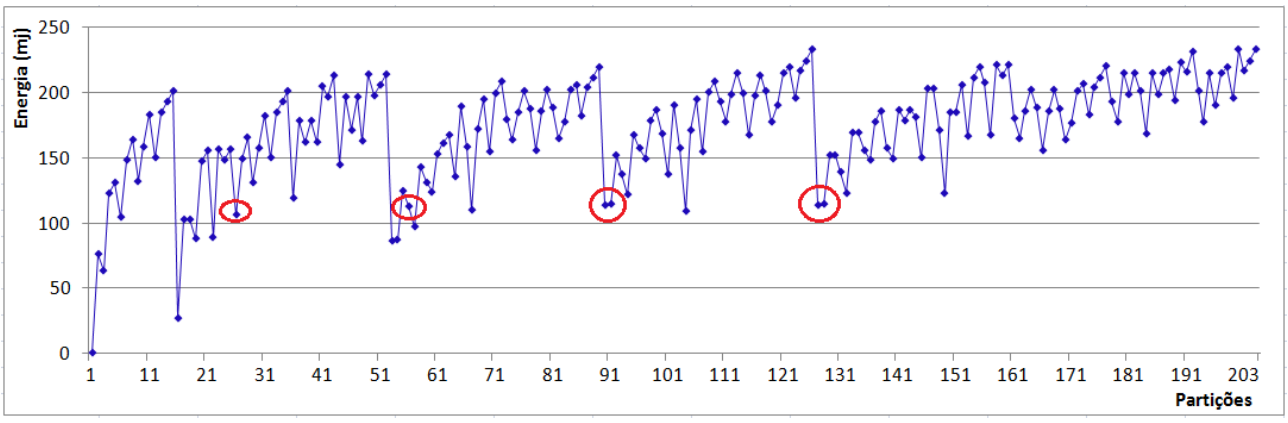


Figura 16 – Plotagem resultante a partir da execução do algoritmo exaustivo em aplicação com 6 tarefas. As 203 partições – *O(Bell(6))* - estão ordenados por proximidade.

O algoritmo *simulated annealing* (SA) [KIR83] é do tipo estocástico com características probabilísticas, que pode ser utilizado em inúmeros problemas. A idéia do algoritmo é explorar diversas soluções através do uso de dois laços aninhados. Quando aplicado na atividade de particionamento, o laço externo (linhas 3 a 27) tenta localizar partições bem distintas objetivando alcançar mínimos globais. O laço interno (linhas 10 a 26), por sua vez, explora pequenas modificações da partição obtida pelo laço externo, objetivando encontrar um mínimo - ou seja, realiza refinamentos na solução provida pelo laço externo.

```
1.   globalMinimumCost ← Maximum double value
2.   interaction ← Interaction parameter of Fig. 14
3.   while(interaction > 0) {
4.     interaction--
5.     if(randomBigMove() == false)
6.       continue
7.     localMinimumCost ← computedCost()
8.     saveComputedMoveInLocalMinimumOne()
9.     temperature = Temperature parameter of Fig. 14
10.    while(temperature > 0) {
11.      temperature--
12.      if(randomSmallMove() == false)
13.        continue
14.      if(costFunctionComparison(localMinimumCost, computedCost())){
15.        localMinimumCost ← computedCost()
16.        saveComputedMoveInLocalMinimumOne()
17.      }
18.      else {
19.        if(!acceptableThreshold(temperature, computedCost(), localMinimumCost))
20.          restoreLocalMinimumMoveToComputedOne()
21.      }
22.    }
23.    if(costFunctionComparison(globalMinimumCost, localMinimumCost)){
24.      globalMinimumCost = localMinimumCost
25.      saveLocalMinimumMoveInGlobalOne()
26.    }
27.  }
```

Figura 17 – Pseudo-código para o algoritmo SA.

A Figura 17 ilustra esses dois laços aninhados que constituem o algoritmo. O laço externo representa uma técnica de pesquisa esparsa, enquanto o laço interno implementa um SA elementar. O efeito de ambos os laços é a pesquisa aleatória por um largo espectro de possibilidades, permitindo encontrar particionamentos que minimizem a função custo *computedCost( )* (função de objetivo de particionamento), chamada pelo algoritmo interno.

O laço interno inicia com um particionamento aleatório, provido pelo laço externo, cujo custo é salvo na variável *localMinimumCost*. A variável *temperature* é usada para controlar o número de iterações do laço interno. Quando *temperature* aumenta, significa que aumenta a probabilidade do algoritmo aceitar um particionamento com custo pior que

os previamente computados. O oposto também é verdade – menor *temperature* significa que a probabilidade de aceitação é reduzida. A cada nova execução do laço interno a variável *temperature* é reinicializada. O objetivo desta abordagem é poder fugir de mínimos locais. A possibilidade de aceitar particionamento piores é avaliada pela função *acceptableThreshold.*

A cada final de laço interno o valor mínimo do laço é comparado com os previamente calculados. Caso este seja menor que os anteriores. O particionamento e seu custo são armazenados como sendo os melhores.

As funções *randomBigMove* e *randomSmallMove*, das linhas 5 e 12, respectivamente, exploram particionamentos aleatórios e resultam um booleano que informa se o particionamento resultante atende as restrições selecionadas na Figura 14. Caso o particionamento explorado não possa ser efetuado, uma iteração é decrementada, seja para o laço interno ou externo, e um novo particionamento é explorado.

5.2 Algoritmo de Cálculo da Função Custo de Energia

As funções custo do particionamento e mapeamento utilizam os mesmos parâmetros de energia da NoC apresentados na Equação (2). No entanto, enquanto o mapeamento especifica o local exato do processador para a NoC, o particionamento apenas explora as necessidades de comunicação, mas o número de saltos (roteadores intermediários) que existe entre dois processadores comunicantes é desconhecido. Assim, a de função custo do particionamento utiliza o conceito de média de saltos, que permite calcular o consumo médio de energia de todos os caminhos possíveis. Sejam X e Y o número de *tiles* em dimensão horizontal e vertical de uma NoC, respectivamente. A Equação (4) calcula o número total de saltos de todos os caminhos que todos os processadores têm em relação ao algoritmo de roteamento XY.

A média de saltos (*hopsAverage*) é calculada dividindo-se o somatório de todos os saltos de todos os caminhos de todos os processadores pelo número total de comunicações, que é indicado pelas Equações (5), (6) e (7).

(4)

$$\boldsymbol{totalHops} = \sum_{\boldsymbol{x}=\mathbf{0}}^{\boldsymbol{X}-\mathbf{1}} \sum_{\boldsymbol{y}=\mathbf{0}}^{\boldsymbol{Y}-\mathbf{1}} \sum_{\boldsymbol{i}=\mathbf{0}}^{\boldsymbol{X}-\mathbf{1}} \sum_{\boldsymbol{j}=\mathbf{0}}^{\boldsymbol{Y}-\mathbf{1}} (|\boldsymbol{x}-\boldsymbol{i}| + |\boldsymbol{y}-\boldsymbol{j}|)$$

(5) $\#\,\boldsymbol{Processors} = \boldsymbol{X} \times \boldsymbol{Y}$

(6) $\boldsymbol{maxComm} = \#\boldsymbol{Processors} \times (\#\boldsymbol{Processors} - \mathbf{1})$

(7) $\boldsymbol{hopsAverage} = \frac{\boldsymbol{totalHops}}{\boldsymbol{maxComm}}$

A função custo apresentada na Figura 18 descreve o algoritmo para atender o requisito de redução de consumo de energia (*Energy reduction* da Figura 14). O laço mais externo (linhas 2 a 17) pesquisa todas as associações processador / tarefas do MPSoC - associações fonte. O laço interno (linhas 6 a 16) pesquisa todas as associações processador / tarefas menos a previamente pesquisada - associações destino. Todos os custos de comunicação provenientes de comunicação das tarefas das associações fontes para as tarefas das associações destino são computados pelos laços aninhados das linhas 12 a 15. Estes custos são armazenados na variável *communicationCost*.

```
1.  double communicationCost ← 0;
2.  for(int source ← 0; source < numberOfAssociations; source ++){
3.    Association sourceAssociation ← getAssociation(source);
4.    if(sourceAssociation.getTasksList() == null)
5.      continue;
6.    for(int target ← 0; target < numberOfAssociations; target ++){
7.      if(target == source)
8.        continue;
9.      Association targetAssociation ← getAssociation(target);
10.     if(targetAssociation.getTasksList() == null)
11.       continue;
12.     for(Task sourceTask: sourceAssociation.getTasksList()){
13.       for(Task targetTask: targetAssociation.getTasksList())
14.         communicationCost ← communicationCost +
            sourceTask.communicationVolumeSentToTask(targetTask);
15.     }
16.   }
17. }
```

Figura 18 – Pseudo-código para o algoritmo que computa o custo com o objetivo de reduzir o consumo de energia.

5.3 Algoritmo de Cálculo da Função Custo de Balanceamento de Carga

A Figura 19 apresenta um pseudo-código da função custo que atender o requisito de balanceamento de carga (*Load balance* da Figura 14). Para atender esse requisito foram explorados particionamentos que levassem ao melhor balanceamento de carga utilizado como referência a minimização do erro quadrático médio descrito na Equação (8).

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

Primeiramente o algoritmo calcula o custo total de uso dos processadores e armazena na variável *totalCpuUse* (linha 7). Esse valor é dividido pelo número de processadores (*numberOfProcessors*) e armazenado na variável *mediumValue* - esta tem a média de uso por processador (linha 9). O laço entre as linhas 11 e 16, calcula o erro

absoluto (EABS) que é a média de uso do processador menos o uso de cada processador. EABS é elevado ao quadrado e acumulado em EQM, de forma que este último tenha a soma total de todos os erros quadráticos. Ao término, EQM é dividida pelo número de processadores, para refletir o erro médio quadrático. Esta abordagem tem cálculo simples e rápido e permite penalizar erros grandes, o que conduz normalmente a um particionamento com bom balanceamento de carga.

```
1.  double totalCpuUse ← 0;
2.  for(Association as: associations){
3.    if(as.getTasksList() == null)
4.      continue;
5.    procType ← as.getProcessor();
6.    for(Task task: as.getTasksList())
7.      totalCpuUse←totalCpuUse+task.getProcessorUse();
8.  }
9.  double mediumValue ← totalCpuUse / numberOfProcessors;
10. double EQM ← 0;
11. for(Association as: associations) {
12.   if(as.getTasksList() == null)
13.     continue;
14.   double EABS ← mediumValue - as.getProcessorUse();
15.   EQM ← EQM + EABS x EABS;
16. }
17. EQM ← EQM / numberOfProcessors;
```

Figura 19 – Pseudo-código para o algoritmo que computa o custo para balanceamento de carga.

5.3.1 Exemplificação da função custo que computa o balanceamento de carga

Para melhor entender o funcionamento do algoritmo que computa o custo para balanceamento de carga, a Figura 20 ilustra uma aplicação sintética composta de 4 processadores e 8 tarefas. Para o cálculo do balanceamento de carga é apenas preciso saber a quantidade de processadores e quanto cada tarefa usa de CPU.

```
<?xml version="1.0" encoding="UTF-8"?>
<SYSTEM_SPECIFICATION>
   <TARGET_ARCHITECTURE>
      <PROCESSOR_TYPE_LIST>
         <PROCESSOR_TYPE type="PowerPC">
            <FEATURES frequency="2100.0" width="0.8" height="0.75"/> </PROCESSOR_TYPE>
      </PROCESSOR_TYPE_LIST>
      <PROCESSOR_LISTS>
         <PROCESSOR_TYPE type="PowerPC">
            <LIST>P0 P1 P2 P3</LIST> </PROCESSOR_TYPE>
      </PROCESSOR_LISTS>
   </TARGET_ARCHITECTURE>
   <APPLICATION_CHARACTERIZATION>
      <TASK_LIST>
         <TASK id="T0">
            <PROCESSOR_TYPE power="13.8" data="2048" code="487" cpuUse="27.83"/> </TASK>
         <TASK id="T1">
            <PROCESSOR_TYPE power="20.21" data="557" code="116" cpuUse="42.76"/> </TASK>
         <TASK id="T2">
            <PROCESSOR_TYPE power="21.47" data="1930" code="817" cpuUse="19.86"/> </TASK>
         <TASK id="T3">
            <PROCESSOR_TYPE power="21.03" data="2894" code="496" cpuUse="65.5"/> </TASK>
         <TASK id="T4">
            <PROCESSOR_TYPE power="23.08" data="2508" code="819" cpuUse="59.45"/> </TASK>
         <TASK id="T5">
            <PROCESSOR_TYPE power="29.33" data="1020" code="345" cpuUse="36.58"/> </TASK>
         <TASK id="T6">
            <PROCESSOR_TYPE power="10.2" data="1066" code="196" cpuUse="51.41"/> </TASK>
         <TASK id="T7">
            <PROCESSOR_TYPE power="16.4" data="712" code="351" cpuUse="46.25"/> </TASK>
      </TASK_LIST>
   </APPLICATION_CHARACTERIZATION>
</SYSTEM_SPECIFICATION>
```

Figura 20 - Exemplo do balanceamento de carga de uma aplicação composta por 4 processadores e 8 tarefas.

A primeira etapa é encontrar a média de uso da CPU, que é o somatório do uso dos processadores (resultando em 349,64%), dividido pelo número total de processadores (nesse exemplo são 4 processadores - P0 a P3), resultando em 87,41%.

A segunda etapa é calcular o erro absoluto, que é o módulo (valor absoluto) da média de uso da CPU (87,41%) subtraindo o uso de cada processador. A coluna 5 da Tabela 2 ilustra o erro calculado em cada processador.

Tabela 2 - Ilustração do cálculo para realização do balanceamento de carga de uma aplicação sintética composta por 4 processadores.

Processadores	Tarefas	Uso da CPU (%)	Erro (Uso da CPU - Média)	EABS (|Erro|)	Erro quadrático
P0	T7, T5	82,83	-4,58	4,58	20,976
P1	T1, T6	94,17	6,76	6,76	45,697
P2	T4, T0	87,28	-0,13	0,13	0,016
P3	T3, T2	85,36	-2,05	2,05	4,202
Total		349,64			70,891
Média		87,41			**17,723**

A terceira etapa é encontrar o erro quadrático, ou seja, é o erro absoluto elevado ao quadrado, como pode ser visto na última coluna da Tabela 2.

Por fim é calculado o erro quadrático médio, que é o somatório de todos os erros quadráticos (resultando em 70,891) dividido pelo número total de processadores, como ilustrado no cálculo a seguir:

$$\mathrm{EQM} = \frac{\sum_{t=1}^{n} e_t^2}{n} = \frac{70{,}891}{4} = 17{,}723$$

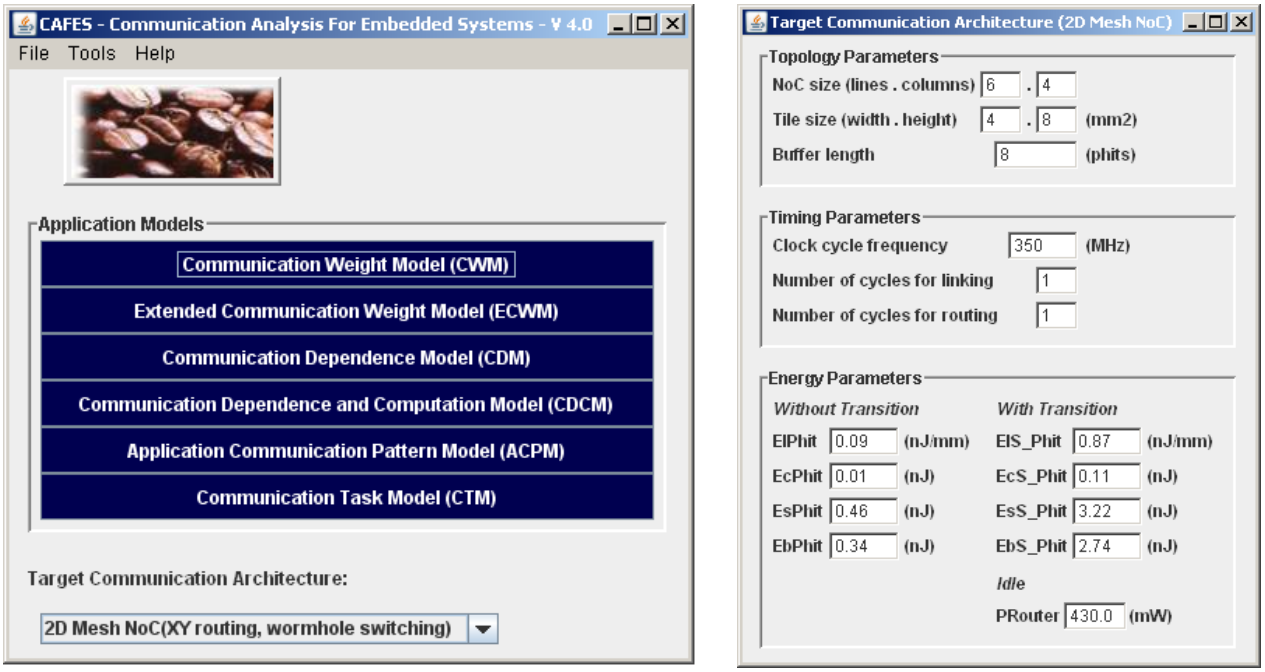
Para esse exemplo o erro quadrático médio foi de 17,723, mas cabe salientar que quanto menor o erro quadrático médio, mais balanceada está a carga.

5.4 FERRAMENTA DE MAPEAMENTO (*FRAMEWORK* CAFES)

O *framework* CAFES (do inglês, *Communication Analysis for Embedded Systems*) [MAR11] foi projetado para automatizar a execução de tarefas de projeto em nível arquitetural. O desenvolvimento inicial do CAFES era direcionado à avaliação do consumo de energia em arquiteturas de comunicação, sendo hoje direcionado para diversas tarefas, tais como o mapeamento de núcleos de processamento em *tiles* da arquitetura alvo.

CAFES é um *framework* extensível open-source que integra modelos, ferramentas e um fluxo de síntese para o projeto de sistemas integrado, tendo NoCs como arquitetura alvo. Os modelos utilizados para descrever as aplicações e arquiteturas alvo são construídos no interior do *framework*. Usando estes modelos, diversas ferramentas como geradores de tráfego, mapeadores e estimadores são implementados. A extensibilidade do *framework* permite a inserção de novas aplicações e modelos de NoC e ferramentas, permitindo projetar uma diversidade de sistemas baseados em NoC, existentes hoje.

Utilizando o *framework*, o projetista pode reduzir significativamente o consumo de energia e latência da aplicação alvo com um tempo de projeto aceitável. O CAFES permite estimar o consumo de energia e latência em diversos níveis de abstração, dependendo da precisão desejada e do tempo de projeto disponíveis.



(a) **(b)**

Figura 21 – (a) apresenta a interface de abertura do *framework* CAFES, indicando a escolha do modelo de aplicação e arquitetura de comunicação alvo. (b) apresenta a interface de configuração dos parâmetros da arquitetura alvo, para topologia malha 2D e aplicações modeladas com o modelo ECWM. Os parâmetros de energia são adaptados de acordo com a tecnologia CMOS TSMC 0,35μm.

Os modelos de descrição disponíveis permitem capturar descrições em diversos níveis e com diferentes características, e uma ferramenta de conversão entre modelos auxilia na verificação da capacidade de cada modelo e quanto são precisos os resultados obtidos com cada modelo.

A partir da interface de abertura do CAFES, parcialmente apresentado na Figura 21(a), o projetista pode escolher um modelo de aplicação e arquitetura de comunicação alvo, que são totalmente independentes da arquitetura de comunicação adotada. Por exemplo, quando selecionado o modelo de aplicação *Extended Communication Weight Model* (ECWM) é apresentada uma arquitetura de comunicação alvo, uma topologia de comunicação NoC do tipo malha 2D, roteamento XY e chaveamento *wormhole*. Esta arquitetura de comunicação e seus parâmetros estão ilustrados na Figura 21(b).

Como podemos observar na Figura 21(a), a versão atual do *framework* CAFES inclui seis modelos de aplicação. Levando em consideração apenas o projeto da arquitetura de comunicação, aspectos relevantes de um modelo de aplicação são: (i) o instante exato em que a comunicação ocorre, ou do que esta comunicação depende para ocorrer, e por quanto tempo esta comunicação ocupa recursos da arquitetura, (ii) a

quantidade de dados transmitidos, e (iii) o padrão de comunicação de dados.

Conforme a escolha do modelo, o projetista pode analisar diferentes aspectos de comunicação, bem como outros requisitos e restrições de projeto, criando diferentes compromissos de precisão, tempo de computação e consumo de memória.

Maiores detalhes do *framework* CAFES podem ser encontrados em [MAR05].

6 EXEMPLIFICAÇÃO DA METODOLOGIA PROPOSTA NESTE TRABALHO USANDO OS *FRAMEWORKS* PALOMA E CAFES

Este capítulo exemplifica a metodologia utilizada aqui. A Figura 22 contém uma estrutura XML com a descrição de uma aplicação paralela sintética composta por 8 tarefas (T0, T1, T2, T3, T4, T5, T6 e T7), suas caracterizações em um processador do tipo *PowerPC* e todas as comunicações entre tarefas da aplicação.

```
<?xml version="1.0" encoding="UTF-8"?>
<SYSTEM_SPECIFICATION>
   <TARGET_ARCHITECTURE>
      <PROCESSOR_TYPE_LIST>
         <PROCESSOR_TYPE type="PowerPC">
            <FEATURES frequency="2100.0" width="0.8" height="0.75"/> </PROCESSOR_TYPE>
      </PROCESSOR_TYPE_LIST>
      <PROCESSOR_LISTS>
         <PROCESSOR_TYPE type="PowerPC">
            <LIST>P0 P1 P2 P3</LIST> </PROCESSOR_TYPE>
      </PROCESSOR_LISTS>
   </TARGET_ARCHITECTURE>
   <APPLICATION_CHARACTERIZATION>
      <TASK_LIST>
         <TASK id="T0">
            <PROCESSOR_TYPE power="13.8" data="2048" code="487" cpuUse="27.83"/> </TASK>
         <TASK id="T1">
            <PROCESSOR_TYPE power="20.21" data="557" code="116" cpuUse="42.76"/> </TASK>
         <TASK id="T2">
            <PROCESSOR_TYPE power="21.47" data="1930" code="817" cpuUse="19.86"/> </TASK>
         <TASK id="T3">
            <PROCESSOR_TYPE power="21.03" data="2894" code="496" cpuUse="65.5"/> </TASK>
         <TASK id="T4">
            <PROCESSOR_TYPE power="23.08" data="2508" code="819" cpuUse="59.45"/> </TASK>
         <TASK id="T5">
            <PROCESSOR_TYPE power="29.33" data="1020" code="345" cpuUse="36.58"/> </TASK>
         <TASK id="T6">
            <PROCESSOR_TYPE power="10.2" data="1066" code="196" cpuUse="51.41"/> </TASK>
         <TASK id="T7">
            <PROCESSOR_TYPE power="16.4" data="712" code="351" cpuUse="46.25"/> </TASK>
      </TASK_LIST>
   </APPLICATION_CHARACTERIZATION>
   <APPLICATION_DESCRIPTION>
      <COMMUNICATION_TASK_LIST>
         <SOURCE_TASK source="T0">
            <COMMUNICATION target="T7" volume="688"/>
            <COMMUNICATION target="T3" volume="2839"/>
            <COMMUNICATION target="T4" volume="1020"/> </SOURCE_TASK>
         <SOURCE_TASK source="T1">
            <COMMUNICATION target="T6" volume="140"/>
            <COMMUNICATION target="T4" volume="211"/> </SOURCE_TASK>
         <SOURCE_TASK source="T2">
            <COMMUNICATION target="T3" volume="1597"/>
            <COMMUNICATION target="T6" volume="2224"/> </SOURCE_TASK>
         <SOURCE_TASK source="T3">
            <COMMUNICATION target="T7" volume="342"/>
            <COMMUNICATION target="T5" volume="1410"/>
            <COMMUNICATION target="T2" volume="1504"/> </SOURCE_TASK>
         <SOURCE_TASK source="T4">
            <COMMUNICATION target="T7" volume="144"/>
            <COMMUNICATION target="T0" volume="2376"/>
            <COMMUNICATION target="T5" volume="1311"/> </SOURCE_TASK>
         <SOURCE_TASK source="T5">
            <COMMUNICATION target="T7" volume="1692"/>
            <COMMUNICATION target="T3" volume="1590"/> </SOURCE_TASK>
         <SOURCE_TASK source="T6">
            <COMMUNICATION target="T7" volume="1787"/>
            <COMMUNICATION target="T2" volume="1078"/> </SOURCE_TASK>
         <SOURCE_TASK source="T7">
            <COMMUNICATION target="T5" volume="1323"/>
            <COMMUNICATION target="T1" volume="1841"/> </SOURCE_TASK>
      </COMMUNICATION_TASK_LIST>
   </APPLICATION_DESCRIPTION>
</SYSTEM_SPECIFICATION>
```

Figura 22 – Descrição de uma aplicação sintética.

A aplicação descrita na Figura 22 tem como arquitetura alvo um MPSoC baseado em NoC com 4 processadores (P0, P1, P2 e P3), todos do mesmo tipo (*PowerPC*).

A tag TASK_LIST descreve as características das tarefas da aplicação, quando executadas em um dado tipo de processador. Por exemplo, a tarefa T0 dissipa em média 13,8 uW de potência pelo processador (*power*), tem 2048 KB de área de dados estática (*data*), 487 KB de área de código (*code*), e ocupa 27,83% da CPU (*cpuUse*).

A tag COMMUNICATION_TASK_LIST descreve a quantidade, direção e o volume de cada comunicação em KB. Aqui, o volume é descrito por uma tripla, onde o primeiro elemento é a tarefa de origem da comunicação, o segundo é a tarefa destino e o terceiro elemento é o volume da comunicação. Por exemplo, (T0, T7, 688) significa que a tarefa T0 envia 688 KB para tarefa T7.

```
Partition:
      Energy Consumption: 170.115uJ
      Load Balance (mean square error): 17.723

      Processor: P0
      Mapped tasks: T7 T5
      Energy consumption: 45.73uJ
      Data: 1732.0KB
      Code: 696.0KB
      CPU occupation: 82.83%

      Processor: P1
      Mapped tasks: T1 T6
      Energy consumption: 30.41uJ
      Data: 1623.0KB
      Code: 312.0KB
      CPU occupation: 94.17%

      Processor: P2
      Mapped tasks: T4 T0
      Energy consumption: 36.88uJ
      Data: 4556.0KB
      Code: 1306.0KB
      CPU occupation: 87.28%

      Processor: P3
      Mapped tasks: T3 T2
      Energy consumption: 42.5uJ
      Data: 4824.0KB
      Code: 1313.0KB
      CPU occupation: 85.36%
```

Figura 23 – Exemplo de um arquivo PAR gerado pela ferramenta de particionamento (PALOMA) e tendo como entrada o XML apresentado na Figura 22.

Tendo como entrada a descrição XML da Figura 22. A ferramenta de particionamento gera um arquivo de texto (PAR) ilustrado pela Figura 23, contendo o (i) consumo de energia total após aplicar o particionamento (*Energy Consumption:*

170.115 *uJ*), o erro quadrático médio do balanceamento de carga entre os processadores (*Load Balance:* 17.723), e também a associação das tarefas aos processadores {(P0, (T7, T5)), (P1, (T1, T6)), (P2, (T4, T0)), (P3, (T3, T2))}. Esta associação é representada por uma tupla contendo processador e lista de tarefas associadas a este. Por exemplo, (P0, (T7, T5)) indica que as tarefas T7 e T5 ficaram agrupadas no processador P0.

Após o particionamento concluído é gerado um arquivo CWG que é formato padrão para o *framework* CAFES poder carregar a aplicação em seu sistema e gerar o mapeamento, que considera aqui apenas critérios de redução de consumo de energia dinâmica.

O arquivo CWG gerado pelo particionamento do XML exemplo é ilustrado na Figura 24. Nesse arquivo é especificado o tamanho da NoC (*tag* #_NoC_Size) – no caso 2 linhas e 2 colunas, a lista de elementos de processamento (*tag* #_CWG_Vertices) – no caso 4 processadores, e uma lista contendo a quantidade de comunicação entre o processador origem e destino (*tag* #_CWG_Edges) – no caso 9 comunicações.

```
#_NoC_Size (lines columns)
 2 2

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

#_CWG_Edges
# (sourceVertex - targetVertex numberOfPhitsTransmited)
 P1 - P2 211
 P1 - P0 1787
 P1 - P3 1078
 P0 - P1 1841
 P0 - P3 1590
 P3 - P0 1752
 P3 - P1 2224
 P2 - P0 2143
 P2 - P3 2839
```

Figura 24 – Arquivo CWG gerado automaticamente pelo particionamento da aplicação descrita na Figura 22.

Após o CAFES carregar o arquivo CWG, a aplicação é apresentada graficamente, tal como ilustrado na Figura 25.

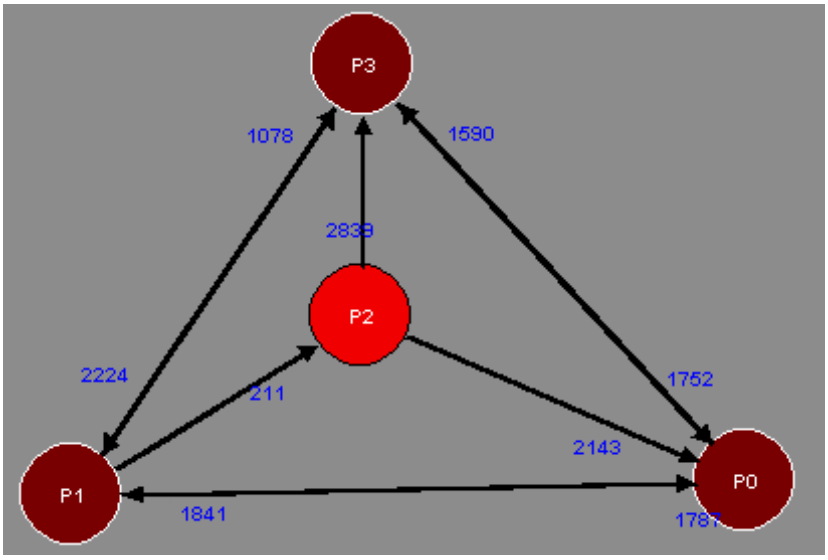


Figura 25 – Visualização da aplicação quando carregada no framework CAFES.

A Figura 26 apresenta o resultado do mapeamento realizado pelo CAFES, onde para cada recurso da arquitetura de comunicação (roteadores e conexões), está associada uma estimativa do consumo de energia dinâmica. Por exemplo, o canal de comunicação que parte de R[1,0] e vai para R[1,1] está anotado com 0,16 *uJ* que é a energia resultante do volume de comunicação multiplicado por ELBit. Eb e Es são respectivamente a energia dinâmica consumida pelos *buffers* (54,68 *uJ*) e portas lógicas (4,56 *uJ*). A interface também fornece uma estimativa global do consumo de energia (*Energy*), que para este exemplo é 227,59 *uJ*.

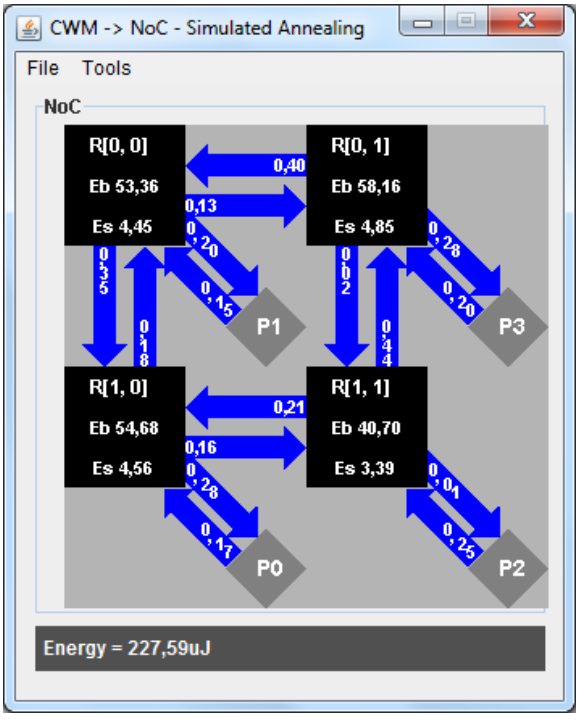


Figura 26 – Mapeamento da aplicação descrita na Figura 22 em uma NoC malha 2x2.

Como é possível observar os custos de mapeamento e particionamento são próximos, porém, o particionamento, por não considerar o mapa final avalia o custo de energia como uma média dos caminhos de comunicação, enquanto que o mapeamento tem uma estimativa mais precisa, por já considerar o mapa final dos processadores na arquitetura alvo.

7 RESULTADOS

Este capítulo apresenta alguns resultados de consumo de energia obtidos com dois conjuntos de experimentos:

- Seção 7.1 – Avalia a importância das atividades de particionamento e mapeamento separadas e conjuntamente para minimizar o consumo de energia da infraestrutura de comunicação. Estes experimentos fazem parte do trabalho apresentado em [ANT11];
- Seção 7.2 – Compara duas estratégias para alcançar mapeamentos dinâmicos eficientes: um que mapeia as tarefas diretamente sobre processadores e um outro que aplica um particionamento de tarefas estático e utiliza esta informação para escolher o mapeamento de tarefa dinâmico. Estes experimentos fazem parte do trabalho apresentado em [ANT12].

7.1 Influência do Particionamento Comparado com o Mapeamento no Consumo de Energia

Todos os resultados avaliam a influência de particionamento e/ou mapeamento sobre a redução do consumo de energia. Os resultados são avaliados em porcentagem, utilizando como referência os piores particionamentos e mapeamentos, e comparando com os melhores particionamentos e mapeamentos. Como não é garantido que o SA obtenha resultados ótimos (seja para um péssimo, quanto para um ótimo particionamento/mapeamento), todos os resultados apontados podem ser maiores, pois o ótimo pode ser melhor e o péssimo pode ser pior. Tanto o particionamento quanto o mapeamento, para esse conjunto de experimentos, são avaliados de forma estática.

A função custo utilizada nestes experimentos tenta alcançar um custo mínimo, o que implica agrupar tarefas altamente comunicantes em um mesmo processador. Além disso, o algoritmo tenta equilibrar a ocupação da CPU através da distribuição justa de tarefas ao longo dos processadores disponíveis. Em outras palavras, as tarefas mais comunicantes são agrupadas em um processador, até que não comprometa mais de 100% de processamento. Esta restrição é negligenciada apenas em casos que não tenha outros processadores disponíveis para fazer o particionamento, e neste caso a distribuição de tarefas é feita de forma balanceada.

O primeiro conjunto de avaliações utilizou o gerador de aplicação sintética (Figura 15) para caracterizar diversas classes de aplicações de acordo com os experimentos.

As Figura 27 e Figura 28 resumem o primeiro conjunto de avaliações que explora

aplicações sintéticas com 50 tarefas, consumo de energia quando executado no processador igual para todas as tarefas, uso da CPU variando aleatoriamente de 10% a 80%, 16 processadores, 6 percentuais do número de comunicações (10%, 20 %, 40%, 60%, 80% e 100%) e 5 volumes de comunicação (1, 10, 100, 1000 e 10000). O número de comunicações é expresso em porcentagem - 100% significa que todas as tarefas se comunicam com todas as outras tarefas, 0% é a ausência de comunicação, e os valores intermediários são calculados linearmente. As combinações de número de comunicações e volume de comunicação totalizam 30 aplicações sintéticas.

A Figura 27 ilustra o efeito da variação do volume de comunicação na redução do consumo de energia para 5 volumes de comunicação (1, 10, 100, 1000 e 10000). Cada ponto da curva contém a média de todos os valores de consumo de energia calculados com experimentos variando todos os 6 percentuais de comunicação (10%, 20%, 40%, 60%, 80% e 100%) - campo *number of communications* da Figura 15 - para um determinado volume de comunicação.

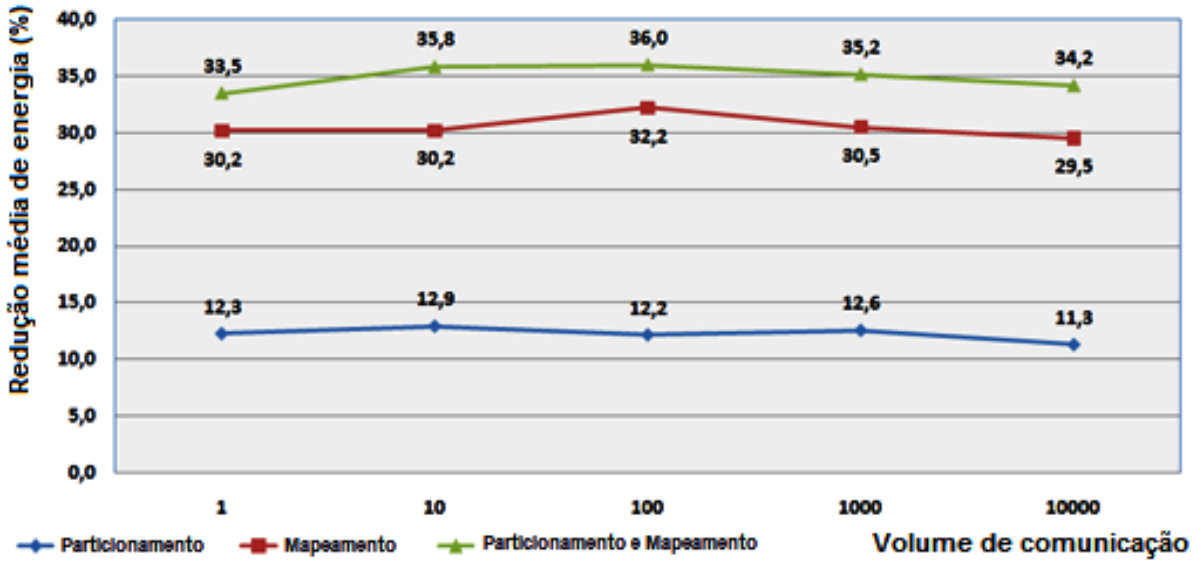


Figura 27 – Influência do particionamento e do mapeamento na redução do consumo de energia para uma série de volumes de comunicação.

É fato que o aumento do volume de comunicação aumenta o consumo de energia da arquitetura alvo. No entanto, a Figura 27 mostra que a redução de energia média é semelhante para todos os volumes de comunicação, independentemente da atividade de projeto. Além disso, o mapeamento é cerca de 3 vezes mais eficiente, quando comparado com o particionamento, e o efeito conjunto das duas atividades não é significativo, se comparado apenas com os resultados do mapeamento.

A Figura 28 ilustra o efeito da variação do número de comunicação sobre a

redução do consumo de energia. Em oposição a Figura 27, cada ponto concentra os valores de todos os 5 volumes de comunicação (1, 10, 100, 1000 e 10000) para um determinado número de comunicações.

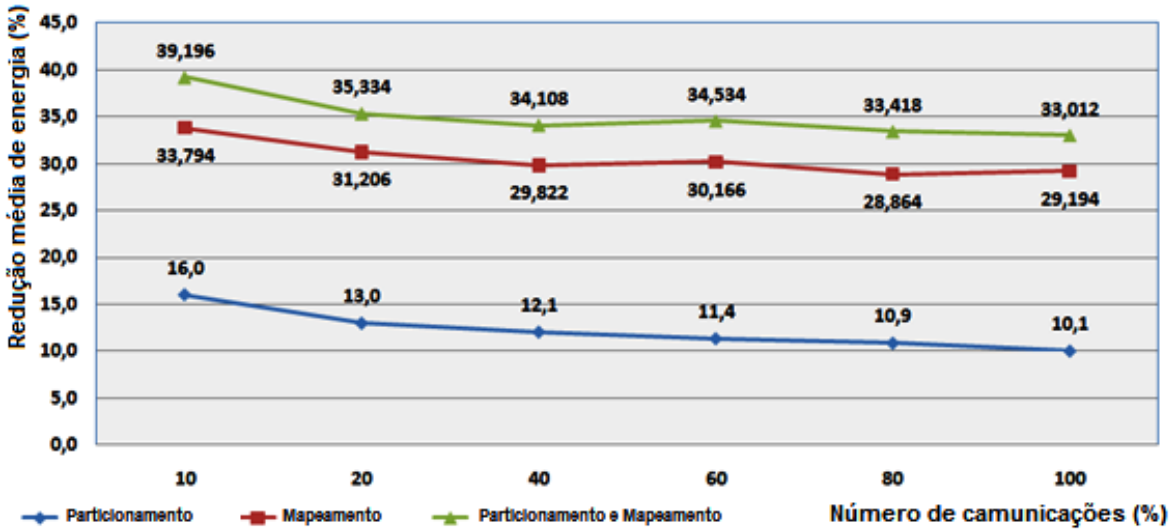


Figura 28 – Influência do particionamento e mapeamento na redução do consumo de energia para uma série de número de comunicações.

Os resultados obtidos com a variação do número de comunicações são semelhantes aos obtidos com a variação do volume de comunicação. No entanto, tanto o mapeamento, quanto o particionamento, são mais eficientes quando há poucas comunicações porque há mais cenários que permite aproximar tarefas comunicantes e processadores.

A Figura 29, juntamente com a Figura 30, mostra um outro conjunto de avaliações. Este considera aplicações sintéticas com 40% do número de comunicações, cada uma com 100 *phits* (volume de comunicação), consumo de energia quando executado no processador igual para todas as tarefas, uso da CPU variando aleatoriamente de 10% a 80%, variação de 6 tamanhos de NoC (2x3, 3x3 3x4, 4x4, 4x5 e 5x5) e variação de 6 número de tarefas (10, 20, 30, 40, 50 e 100). As combinações de tamanhos de NoC e número de tarefas totalizam 36 aplicações sintéticas.

A Figura 29 ilustra o efeito da variação do tamanho da NoC na redução do consumo de energia para 6 tamanhos de NoC: 2x3, 3x3, 3x4, 4x4, 4x5 e 5x5 totalmente preenchidas, o que implica as seguintes quantidades de processadores: 6, 9, 12, 16, 20 e 25, respectivamente. Cada ponto da curva contém a média de todo os valores de consumo de energia calculados com experimentos variando todos os 6 números de tarefas (10, 20, 30, 40, 50 e 100) para um dado tamanho de NoC.

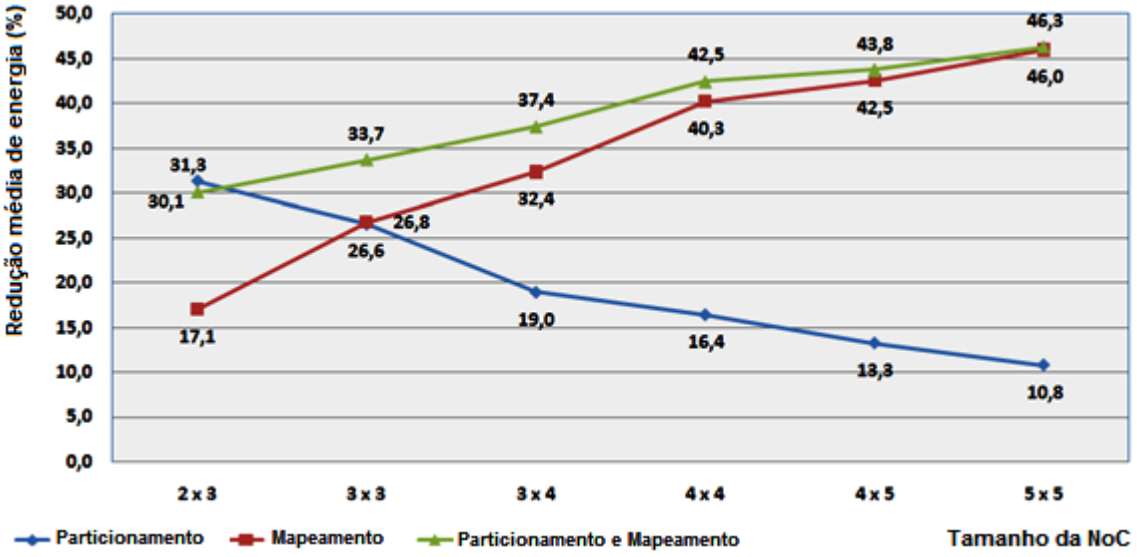


Figura 29 – Influência do particionamento e mapeamento na redução do consumo de energia para uma variedade de tamanhos de NoC.

É possível observar a importância do particionamento, quando um número significativo de tarefas é agrupado em um único processador (por exemplo, 100 tarefas agrupadas em 2x3 processadores, o que implica 16 tarefas / processador). Por outro lado, quando a relação do número de tarefas versus o tamanho da NoC diminui, a eficiência do particionamento também reduz e a eficácia do mapeamento torna-se evidente.

A Figura 30 ilustra o efeito da variação do número de tarefas sobre a redução do consumo de energia. Em oposição a Figura 29, cada ponto concentra os valores de todas as 6 dimensões da NoC para um determinado número de tarefas.

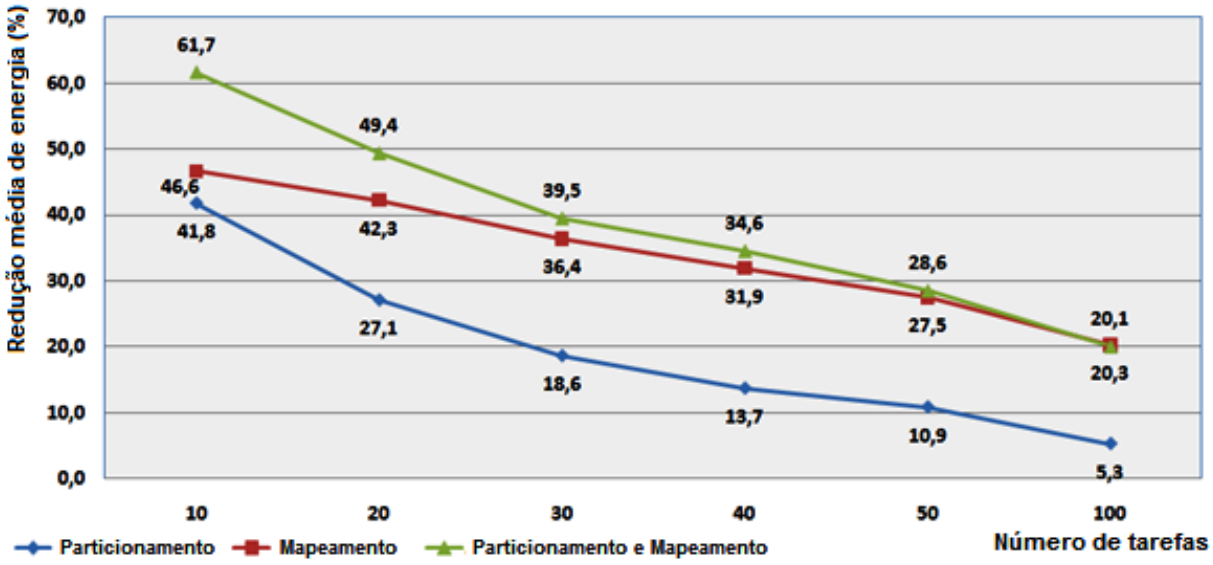


Figura 30 – Influência do particionamento e mapeamento na redução do consumo de energia para uma série de número de tarefas.

O aumento do número de tarefas reduz a eficiência do mapeamento e

particionamento, que é justificada pela redução das possíveis partições - algoritmo de particionamento aplica a técnica de balanceamento de carga evitando mais de 100% de ocupação da CPU. Por outro lado, uma redução do número de tarefas significa mais possibilidades de particionamento e mapeamento, aumentando sua influência sobre a redução de energia média.

Uma ultima avaliação no primeiro conjunto de experimentos verifica a influência do particionamento e mapeamento em 4 aplicações embarcadas: (i) um PBX digital (*Privative Branch Exchange*), (ii) um sistema de reconhecimento de imagem (IRS), (iii) um algoritmo distribuído para o cálculo da integral de *Romberg*, e (iv) um sistema multimídia (MMS). A Tabela 1 mostra algumas características relevantes destas aplicações.

Tabela 3 – Características de quatro aplicações embarcadas.

	PBX	IRS	Romberg	MMS
Tamanho da NoC (linhas x colunas)	2 x 3	2 x 3	3 x 4	4 x 4
Números de processadores	5	6	10	16
Números de tarefas	24	12	30	34
Números de comunicações	142	53	60	182
Quantidade de comunicações médias (*bytes*)	2.334	30.827	35	22.135

A Figura 31 ilustra o efeito do particionamento e do mapeamento na redução do consumo de energia para as 4 aplicações embarcadas caracterizadas na Tabela 3.

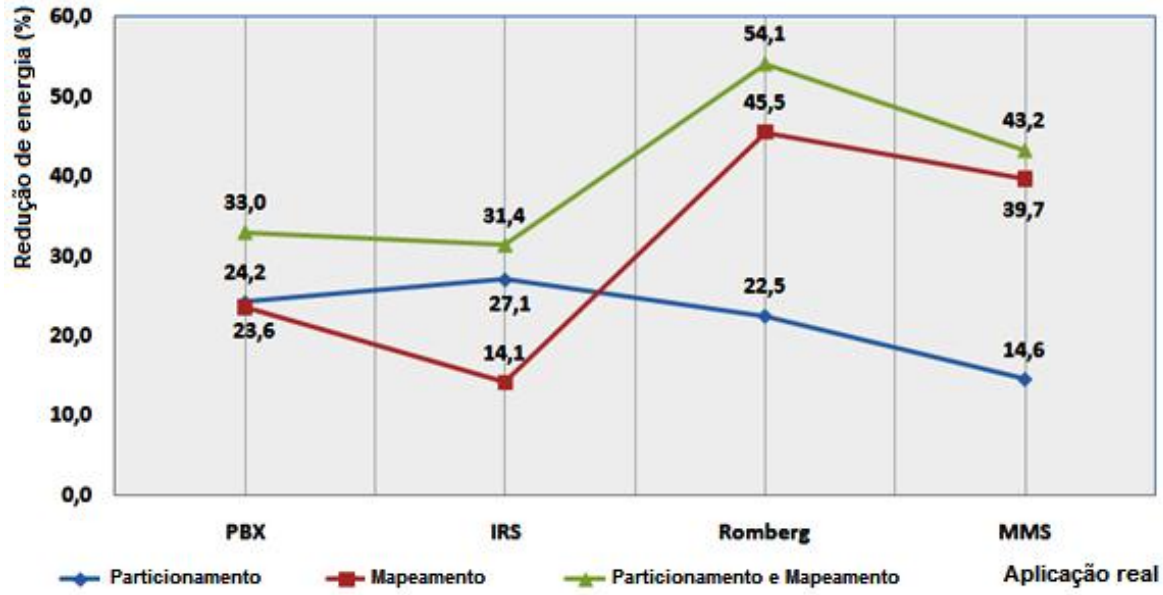


Figura 31 – Influência do particionamento e do mapeamento na redução do consumo de energia para quatro aplicações embarcadas.

Os resultados obtidos ao se considerar as aplicações sintéticas são mantidos. No entanto, dois resultados relevantes são apontados: (i) MMS e Romberg são tipicamente

aplicações de fluxo de dados, e fluxos de dados não são capturados pelo particionamento, uma vez que os fluxos não são especificáveis pelos modelos aqui utilizados, porém o efeito do mapeamento na redução do consumo de energia é evidente, e (ii) o particionamento realizado sobre a aplicação IRS prevê agrupamento das tarefas com comunicação balanceada. Este balanceamento de comunicação reduz a eficiência do mapeamento, uma vez que o volume de comunicação não tem uma variação significativa com o posicionamento dos processadores.

7.2 Influência do particionamento estático como uma etapa prévia do mapeamento dinâmico no consumo de energia

Os experimentos que seguem foram desenvolvidos com o objetivo de avaliar o efeito do uso do particionamento estático como uma etapa anterior ao mapeamento dinâmico na redução do consumo de energia. Estes se referem a comparação das abordagens de particionamento e mapeamento da Figura 1 com outra abordagem utilizada na literatura (Figura 32), que é a associação das tarefas diretamente nos processadores, sem particionamento prévio.

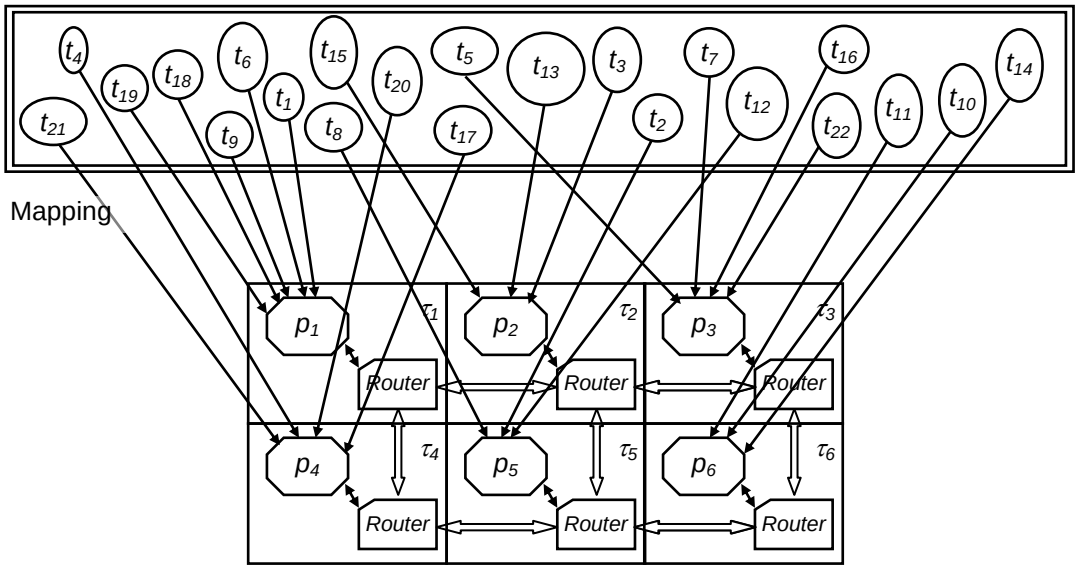


Figura 32 - Mapeamento das tarefas diretamente nos processadores, sem particionamento prévio.

O particionamento, por ser estático, foi realizado com o algoritmo SA, enquanto que o mapeamento por ser dinâmico, foi realizado com um algoritmo mais rápido, que pesquisa o primeiro nodo vizinho mais próximo que esteja “livre”. Aqui, cabe salientar que o algoritmo de mapeamento que mapeia grupo de tarefas (tem particionamento estático prévio) considera “livre” quando um processador não tem grupo de tarefas associado, e uma vez um grupo associado, o mesmo não pode mais receber outras tarefas. Já o

algoritmo de mapeamento que mapeia tarefas (ou seja, não tem o particionamento prévio), considera “livre” os processadores que, ao somar a ocupação da tarefa a ser mapeada ao valor de ocupação do processador não ultrapasse a restrição especificada (parâmetros *Load Balance* da Figura 14).

A Tabela 4 e a Tabela 5 mostram em porcentagem o quanto a abordagem que usa particionamento estático consegue minimizar o consumo de energia em comparação com a abordagem que considera apenas mapeamento dinâmico.

A Tabela 4 resume a primeira avaliação destes experimentos, que é composta por aplicações sintéticas com 50 tarefas, com uso do processador variando aleatoriamente de 10% a 80%, 16 processadores, e um conjunto do número de comunicações (10%, 20%, 30%, 40%, 60%, 80% e 100%) combinado com um conjunto de volume de comunicações (1, 10, 100, 1000 e 10000). O número de comunicações é expresso em porcentagem – 100% significa que todas as tarefas comunicam com todas as outras tarefas, 0% significa a ausência de comunicação e valores intermediários são computados linearmente.

Tabela 4 – Resultado da minimização do consumo de energia para 30 aplicações sintéticas com o número de comunicações variando entre tarefas e quantidade de *bits* de cada comunicação.

Redução de energia (%)		Número de comunicações (%)						
		10	**20**	**40**	**60**	**80**	**100**	**MVC**
Volume de Comunicação	**1**	19,7	18,4	16,4	16,0	15,7	8,8	15,8
	10	19,1	17,3	14,5	15,4	14,8	14,2	15,9
	100	18,2	15,9	16,4	14,3	11,7	15,7	15,4
	1000	20,1	15,7	15,2	13,2	13,5	13,0	15,1
	10000	17,1	19,6	15,9	13,5	13,1	12,5	15,3
MNC		18,8	17,4	15,7	14,5	13,8	12,8	**15,5**

Legenda: MVC - Média considerando as restrições dos volumes de comunicação
MNC - Média considerando as variações dos números de comunicações

A coluna MVC mostra o quanto o número de *bits* transmitidos entre as tarefas afeta na minimização do consumo de energia. É fato que o aumento do volume de comunicação aumenta a energia consumida pela arquitetura alvo. No entanto, a Tabela 4 mostra que a média de redução de energia é semelhante para todos os volumes de comunicação. Isto significa que essa característica não afeta a eficiência das abordagens aqui avaliadas.

A linha MNC mostra o efeito da restrição do número de comunicações entre as tarefas na redução do consumo de energia. Os resultados aqui alcançados são

semelhantes aos descritos no parágrafo anterior. No entanto, a abordagem que considera o particionamento estático é mais eficiente na redução do consumo de energia para um pequeno número de comunicações, porque há mais cenários que permitem aproximar as tarefas fortemente comunicantes.

Por fim, conclui-se que frente aos critérios de restrição de volume e número de comunicações, a abordagem que considera o particionamento estático consegue em média minimizar 15,5%

Tabela 5 – Resultado da minimização do consumo de energia para 36 aplicações sintéticas obtidas através da combinação de 6 quantidades de tarefas e 6 tamanhos de NoCs.

Redução de energia (%)		Quantidade de tarefas						
		10	**20**	**30**	**40**	**50**	**100**	**MTN**
Tamanho da NoC	**2 x 3**	2,4	6,5	10,9	16,6	23,8	48,3	18,1
	3 x 3	3,5	7,7	11,2	19,4	34,6	68,8	24,2
	3 x 4	7,7	12,3	18,3	23,4	35,9	73,6	28,5
	4 x 4	9,3	15,3	19,3	20,1	39,9	79,7	30,6
	4 x 5	12,5	18,6	23,1	28,9	45,6	89,0	36,3
	5 x 5	15,0	25,5	26,8	31,2	49,6	93,1	40,2
MQT		8,4	14,3	18,3	23,3	38,2	75,4	**29,7**

Legenda: MTN - Média considerando diversos tamanhos de NoC
MQT - Média considerando a variação da quantidade de tarefas

A Tabela 5 mostra os resultados de um conjunto composto por aplicações sintéticas, onde, para todas as aplicações, qualquer tarefa comunica com 40% das tarefas restantes da aplicação, cada comunicação entre as tarefas tem 100 *phits* (um *phit* aqui tem 16 *bits*) e o uso do processador varia aleatoriamente entre 10% e 80%. Combinando 6 quantidades de tarefas (10, 20, 30, 40, 50 e 100) com 6 tamanhos de NoC (2x3, 3x3, 3x4, 4x4, 4x5 e 5x5) totaliza 36 aplicações sintéticas. Algumas combinações têm *tile* vazio – não povoadas por tarefas (por exemplo, uma NoC 3x4 com 10 tarefas implica 2 *tiles* vazios), e outras combinações tem mais de uma tarefa por *tile* (por exemplo uma NoC 3x3 com 10 tarefas implica que pelo menos um *tile* tem mais de uma tarefa). Uma vez que cada *tile* contém um processador, há 6 quantidades de processadores: 6, 9, 12, 16, 20 e 25, para cada tamanho de NoC, respectivamente.

A minimização do consumo de energia alcançada com o particionamento estático não é significativa quando analisados aplicações com pequena quantidade de tarefas executadas em pequenas NoCs. Por outro lado, para grandes aplicações ou grandes NoCs a eficiência desta abordagem é evidente, uma vez que lida melhor com estas complexidades.

A linha MQT ilustra o efeito da variação da quantidade de tarefas na minimização do consumo de energia. Esta variação implica a parte mais importante da redução do consumo de energia em relação aos experimentos realizados aqui, que é justificado pelo aumento das possibilidades de agrupamento de tarefa quando a quantidade de tarefas aumenta e essas possibilidades são bem capturadas pela etapa de particionamento estático.

Quatro aplicações embarcadas (as mesmas utilizadas na Seção 7.1) compõem o último conjunto de avaliações, sendo que a Tabela 6 mostra os valores de minimização do consumo de energia para estas.

Tabela 6 – Resultado da minimização do consumo de energia para quatro aplicações embarcadas.

Redução de energia	Aplicações embarcadas				Média
	PBX	IRS	Romberg	MMS	
Diferença (%)	29,1	9,8	51,0	22,8	28,2

Similarmente aos resultados obtidos com aplicações sintéticas, a eficiência da abordagem que utiliza particionamento estático é obtida em aplicações com maior quantidade de tarefas mapeadas em NoC mais complexas, que são os casos de *Romberg* e MMS. Além disto, as aplicações *Romberg* e PBX são mapeadas em NoCs com mais *tile* do que a quantidade de processadores exige, oferecendo assim mais lugares para mapear grupos de tarefas. A abordagem com particionamento estático explora melhor esta característica permitindo a busca de melhores resultados. Além disto, como dito acima, o volume de comunicação não influencia na redução de energia, tanto que os resultados alcançados com a aplicação *Romberg* mostra minimização do consumo de energia muito maior do que os obtidos nas aplicações IRS e MMS.

A redução média do consumo de energia para avaliações feitas no conjunto de experimentos desta Seção é 23,5%. Este resultado foi obtido através da média de consumo de energia proporcional ao número de aplicações, como pode ser visto no cálculo abaixo:

$$\frac{[(15{,}5 \times 30) + (29{,}7 \times 36) + (28{,}2 \times 4)]}{30 + 36 + 4} = \frac{1644}{70} = 23{,}5\%$$

8 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação abordou o tema de particionamento e mapeamento de MPSoCs homogêneos baseados em NoCs. 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, sendo estes processadores do mesmo tipo. As atividades abordadas aqui são o particionamento destas tarefas em grupo de tarefas e o posterior mapeamento deste grupo em processadores posicionados em *tiles* e o mapeamento de tarefas (sem um prévio particionamento) em processadores. A atividade de particionamento é sempre avaliada de forma estática, enquanto que a atividade de mapeamento foi avaliada de forma estática e dinâmica conforme o conjunto de experimentos.

Os experimentos realizados na Seção 7.1 mostram que a atividade de mapeamento é mais eficiente na redução de consumo de energia, quando comparado com atividade de particionamento estático, mas mesmo assim o efeito de particionamento não pode ser negligenciado. Além disso, o efeito em conjunto de ambas as atividades reduz em média 37% o consumo de energia.

Já os experimentos realizados na Seção 7.2 se baseiam no fato que o mapeamento é uma atividade de projeto NP-completa, e quando realizada em tempo de execução pode não obter bons resultados, devido ao tempo exíguo e ao grande número de soluções a serem exploradas. A aplicação do particionamento de tarefas em grupos de forma estática reduz o espaço de busca que o mapeamento necessita realizar, o que permite uma execução mais eficiente de algoritmos de mapeamento dinâmico, podendo este obter um mapeamento de melhor qualidade. Experimentos com várias aplicações sintéticas e 4 aplicações embarcadas mostram que a redução média do consumo de energia é de 23,5%.

8.1 Contribuições do trabalho

Dentre as contribuições do trabalho constam:

- Elaboração de um formato intermediário (XML) para descrever aplicações, permitindo compartilhamento de informações com o propósito de ser menos restrito do que formatos de documentos proprietários. Sendo este formato usado nos *frameworks* PALOMA e CAFES;
- Desenvolvimento de uma ferramenta de geração de aplicações sintéticas

para modelar diversas classes de aplicações (ex. IO-*bounded* ou CPU-*bounded*). Esta ferramenta faz parte do *framework* Paloma;

- Desenvolvimento parcial do *framework* PALOMA, em atividade que envolve o particionamento de tarefas tendo como arquitetura alvo MPSoCs homogêneos;
- Desenvolvimento de 2 algoritmos para cálculo da função custo do particionamento (um para atender o requisito de redução do consumo de energia e outro para atender o requisito de balanceamento de carga);
- Publicações de 2 artigos ([ANT11] e [ANT12]).

8.2 Trabalhos futuros

É importante destacar que a pesquisa realizada neste trabalho, deixa margens para que trabalhos futuros possam ser realizados, incluindo:

- Exploração de novos algoritmos para particionamento de tarefas, como por exemplo, *Largest Communication First*, *Incremental, Tabu Search* e diversos heurísticos;
- Estender o particionamento de tarefas em elementos de processamento para arquitetura do tipo MPSoC heterogêneo;
- Implementar uma multi-função custo que reduza o consumo de energia combinando balanceamento de carga e consumo de energia;
- Estender o particionamento para outros requisitos, como redução do tempo de execução da aplicação e a possibilidade de atender tarefas de tempo real.

REFERÊNCIAS

[ANT11] Antunes, E. B.; Aguiar, A.; Johann, S. F.; Sartori, M.; Hessel, F. P.; Marcon, C. A. M. **Partitioning and Mapping on NoC-Based MPSoC: An Energy Consumption Saving Approach.** In: Workshop on Network on Chip Architectures (NoCArC), Porto Alegre. International Symposium on Microarchitectures (Micro-44). vol. 1. 2011, pp. 1-6.

[ANT12] Antunes, E. B.; Soares, M.; Johann, S. F.; Aguiar, A.; Sartori, M.; Hessel, F. P.; Marcon, C. A. M. **Partitioning and Dynamic Mapping Evaluation for Energy Consumption Minimization on NoC-Based MPSoC.** In: International Symposium on Quality Electronic Design (ISQED), Santa Clara, USA. International Symposium on Quality Electronic Design (ISQED). vol. 1. 2012, pp. 1-7.

[BEU10] Beux, S. L.; Bois, G.; Nicolescu, G.; Bouchebaba, Y.; Langevin, M.; Paulin. P.; **Combining mapping and partitioning exploration for NoC-based embedded systems.** J. Syst. Archit. vol. 56 (7), July 2010, pp. 223-232.

[BJE06] Bjerregaard, T.; Mahadevan, S. **A survey of research and practices of Network-on-chip.** ACM Comput. Surv. vol. 38, New York, USA, June 2006.

[BON07] Bononi, L.; Concer, N.; Grammatikakis, M.; Coppola, M.; Locatelli, R.; **NoC Topologies Exploration based on Mapping and Simulation Models.** In: 10th Euromicro Conference, Digital System Design Architectures, Methods and Tools, August 2007, pp.543-546.

[CAR09] Carvalho, E. **Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC.** Tese Doutorado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, Porto Alegre, Brasil, 2009, 168p.

[CAR10] Carvalho, E.; Calazans, N.; Moraes, F. **Dynamic Task Mapping for MPSoCs**. IEEE Design and Test of Computers, vol. 27 (5), September 2010, pp. 26-35.

[DAL01] Dally, W. & Towles, B. Route Packets, **Not Wires: on Chip Interconnection Networks.** In: 38th Design Automation Conference - DAC'01, 2001, pp. 684-689.

[GOH10] Göhringer, D.; Hübner, M.; Benz, M.; Becker, J.; **A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip.** In: 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), May 2010, pp.259-262.

[HU03] Hu, J.; Marculescu, R. **Energy-Aware Mapping for Tile-Based NoC Architectures under Performance Constraints.** In: Asia and South Pacific Design Automation Conference (ASP-DAC), January 2003, pp. 233-239.

[JER05] Jerraya, A., Tenhunen, H.; Wolf, W. **Guest Editors' Introduction: Multiprocessor Systems-on-Chips.** IEEE Computer, vol. 38 (7). July 2005, pp.36-40.

[KIR83] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. **Optimization by simulated annealing.** Science, 1983, pp. 671–680.

[LEU10] Leupers, R.; Castrillon, J.; **MPSoC programming using the MAPS compiler.** In: 15th Asia and South Pacific , Design Automation Conference (ASP-DAC), January 2010, pp.897-902.

[LIU10] Liu, T.; Zhao, Y.; Li, M.; Xue, C. J.; **Task Assignment with Cache Partitioning and Locking for WCET Minimization on MPSoC.** In: 39th International Conference on Parallel Processing, 2010, pp. 573-582.

[MAN11] Mandelli, M.; Ost, L.; Carara, E.; Guindani, G.; Gouvea, T.; Medeiros, G.; Moraes, F.G.; **Energy-aware dynamic task mapping for NoC-based MPSoCs**. Circuits and Systems (ISCAS), IEEE International Symposium, May 2011, pp.1676-1679.

[MAR01] Martin, G.; Chang, H. **System-on-Chip design**. In: 4th International Conference on ASIC, 2001, pp.12-17.

[MAR05] Marcon, C.; **Modelos para o Mapeamento de Aplicações em Infra-estruturas de Comunicações Intrachip.** Tese Doutorado, Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre, Brasil, 2005, 188p.

[MAR11] 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, 2011.

[MUR04] Murali, S. and De Micheli, G. **Bandwidth-Constrained Mapping of Cores onto NoC Architectures.** Design, Automation and Test in Europe (DATE), February. 2004, pp. 896-901.

[NED11] Nedjah, N.; Silva, M., V., C.; Mourelle, L., M. **Customized computer-aided application mapping on NoC infrastructure using multi-objective optimization.** J. Syst. Archit. 57, 1, January 2011, pp. 79-94.

[RAU94] Rauscher, T. G. **Time to Market Problems - The Organization is the Real Cause.** Proceedings of the IEEE International Engineering Management Conference, 1994, pp. 338-345.

[SAL09] Salminen, E.; Grecu, C.; Hamalainen, T.D.; Ivanov, A. **Application modeling and hardware description for network-on-chip benchmarking.** Computers & Digital Techniques, IET, vol.3 (5), September 2009, pp.539-550.

[SHE99] Sherwani, N. A.; **Algorithms for VLSI Physical Design Automation.** 2nd. Edition. Kluwer Academic Publisher. Boston, USA, 1999.

[TSA10] Tsai, K.; Lai, F.; Pan, C.; Xiao, D.; Tan, H.; Lee, H. **Design of low latency on-chip communication based on hybrid NoC architecture.** In: 8th IEEE International Conference (NEWCAS), June 2010, pp.257-260.

[YE02] Ye, T.; Benini, L.; De Micheli, G. **Analysis of power consumption on switch fabrics in network routers.** Design Automation Conference - DAC, June 2002,

pp.524-529.

[YOU09] Youness, H.; Hassan, M.; Sakanushi, K.; Takeuchi, Y.; Imai, M.; Salem, A.; Wahdan, A. M.; Moness, M.; **A high performance algorithm for scheduling and hardware-software partitioning on MPSoCs.** In: 4th International Conference on Design & Technology of Integrated Systems in Nanoscale Era. April 2009, pp.71-76.

[ZEF03] Zeferino, C. A.; Susin, A. A. **SoCIN: a Parametric and Scalable Network-on-Chip**. In: Symposium on Integrated Circuits and Systems (SBCCI'03), 2003, pp. 169-174.

[ZWI96] Zwillinger, D.; **Standard Mathematical Tables and Formulae.** In: 30th. Edition. CRC Press Inc. NY, USA, 1996.