PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME MACHADO DE CASTILHOS

GERENCIAMENTO TÉRMICO ENERGÉTICO EM MPSOCS

Porto Alegre
2017

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GERENCIAMENTO TÉRMICO E ENERGÉTICO EM MPSOCS

Guilherme Machado de Castilhos

Tese apresentada como requisito parcial à obtenção de grau de Doutor em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre, Brasil
2017

Ficha Catalográfica

C352g Castilhos, Guilherme Machado de

Gerenciamento Térmico e Energético em MPSoCs / Guilherme Machado de Castilhos . – 2017.
94p.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. MPSoC. 2. Gerência Térmica. 3. Gerência Energética. 4. Confiabilidade. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Guilherme Machado de Castilhos

Gerenciamento Térmico e Energético em MPSoCs

Tese apresentada como requisito parcial para obtenção do grau de Doutor em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 10 de agosto de 2017.

BANCA EXAMINADORA:

Prof. Dr. Ney Laert Vilar Calazans (PPGCC/PUCRS)

Prof. Dr. Prof. Dr. Tiago Roberto Balen (UFRGS)

Prof. Dr. Mateus Beck Rutzig (UFSM)

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Orientador)

AGRADECIMENTOS

À minha amada família, Pai, Mãe, Rapha e Gabi por me transformar em quem eu sou.

À Aline, pelas inúmeras vezes que você me enxergou melhor do que eu sou. Te amo.

Ao meu orientador/professor/conselheiro Moraes por me fazer amar a vida acadêmica pelo exemplo.

Gerenciamento Térmico e Energético em MPSoCs

Resumo

As altas variações térmicas e de temperatura de operação podem ter um impacto significativo no desempenho do sistema, consumo de energia e na confiabilidade, uma métrica cada vez mais crítica em sistema multiprocessados. As técnicas de gerenciamento térmico existentes dependem de sensores físicos para fornecer os valores de temperatura para regular a temperatura de operação e a variação térmica do sistema em tempo de execução. No entanto, os sensores térmicos em um chip apresentam limitações (por exemplo, custo extra de potência e de área), o que pode restringir seu uso em sistemas com uma grande quantidade de processadores. Neste contexto, esta Tese propõe um modelo de temperatura baseado em software, realizado em tempo de execução, permitindo capturar informações detalhadas da distribuição de temperatura de sistemas multiprocessados com custo mínimo no desempenho das aplicações. Para validar a proposta, o modelo foi incluído em uma plataforma MPSoC com memória distribuída, descrita no nível RTL. Além disso, os resultados mostram que o erro absoluto médio da estimativa de temperatura, em comparação com a ferramenta HotSpot, é menor do que 4% em sistemas com até 36 elementos de processamento.

O mapeamento de tarefas foi o processo escolhido para atuar no sistema, utilizando as informações de temperatura geradas pelo modelo proposto. O mapeamento de tarefas é o processo de selecionar um elemento de processamento para executar uma determinada tarefa. O número de núcleos em sistemas multiprocessados, aumenta a complexidade do mapeamento de tarefas. As principais preocupações no mapeamento de tarefas em sistemas de grande porte incluem: (*i*) escalabilidade; (*Ii*) carga de trabalho dinâmica; e (*iii*) confiabilidade. É necessário distribuir a decisão de mapeamento em todo o sistema para assegurar a escalabilidade. A carga de trabalho de sistemas multiprocessados pode ser dinâmica, ou seja, novas aplicações podem começar a qualquer momento, levando a diferentes cenários de mapeamento. Portanto, é necessário executar o processo de mapeamento em tempo de execução para suportar carga dinâmica de trabalho.

A atribuição de carga de trabalho desempenha um papel importante na confiabilidade do sistema. O desequilíbrio de carga pode gerar zonas de *hotspot* e consequentemente implicações térmicas. Recentemente, técnicas de mapeamento de tarefas com o objetivo de melhorar a confiabilidade do sistema foram propostas na literatura. No entanto, tais abordagens dependem de decisões de mapeamento centralizado, que não são escaláveis. Para enfrentar esses desafios, esta Tese propõe uma heurística de mapeamento hierárquico realizado em tempo de execução, que ofereça escalabilidade e uma melhor distribuição térmica. A melhor distribuição térmica dentro do sistema aumenta a confiabilidade do sistema a longo prazo, devido à redução das variações térmicas e redução de zonas de hotspot. A heurística de mapeamento proposta considera a temperatura do

PE como uma função custo. A proposta adota um esquema hierárquico de monitoramento de temperatura, capaz de estimar em tempo de execução a temperatura instantânea de cada elemento de processamento. O mapeamento usa a temperatura estimada pelo método de monitoramento para orientar a decisão de mapeamento. Os resultados comparam a proposta com uma heurística de mapeamento cuja principal função de custo minimiza a energia de comunicação. Os resultados obtidos mostram diminuição da temperatura máxima (melhor caso, 8%) e melhora na distribuição térmica (melhor caso, valor 50% menor do desvio padrão das temperaturas dos processadores). Além disso, alcançou-se, no melhor caso, um aumento de 45% no tempo de vida do sistema utilizando o mapeamento proposto.

Palavras-chave: MPSoC, Gerência Térmica, Gerência Energética, confiabilidade.

Thermal and Energy Management in MPSoCs

Abstract

Thermal cycles and high temperatures can have a significant impact on the systems performance, power consumption and reliability, which is a major and increasingly critical design metric in emerging multi-processor embedded systems. Existing thermal management techniques rely on physical sensors to provide them temperature values to regulate the system's operating temperature and thermal variation at runtime. However, on-chip thermal sensors present limitations (e.g., extra power and area cost), which may restrict their use in large-scale systems. In this context, this Thesis proposes a lightweight software-based runtime temperature model, enabling to capture detailed temperature distribution information of multiprocessor systems with negligible overhead in the execution time. The temperature model is embedded in a distributed-memory MPSoC platform, described at the RTL level. Results show that the average absolute temperature error estimation, compared to the HotSpot tool is smaller than 4% in systems with up to 36 processing elements.

Task mapping is the process selected to act in the system, using the temperature information generated by the proposed model. Task mapping is the process of assigning a processing element to execute a given task. The number of cores in many-core systems increases the complexity of the task mapping. The main concerns of task mapping for large systems include (*i*) scalability; (*ii*) dynamic workload; and (*iii*) reliability. It is necessary to distribute the mapping decisions across the system to ensure scalability. The workload of emerging many-core systems may be dynamic, i.e., new applications may start at any moment, leading to different mapping scenarios. Therefore, it is necessary to execute the mapping process at runtime to support dynamic workload.

The workload assignment plays a major role in the many-core system reliability. Load imbalance may generate hotspots zones and consequently thermal implications. Recently, task mapping techniques aiming at improving system reliability have been proposed in the literature. However, such approaches rely on centralized mapping decisions, which are not scalable. To address these challenges, the main goal of this Thesis is to propose a hierarchical runtime mapping heuristic, which provides scalability and fair thermal distribution. Thermal distribution inside the system increases the system reliability in long-term, due to the reduction of hotspot regions. The proposed mapping heuristic considers the PE temperature as a cost function. The proposal adopts a hierarchical thermal monitoring scheme, able to estimate at runtime the instantaneous temperature at each processing element. The mapping uses the temperature estimated by the monitoring scheme to guide the mapping decision. Results compare the proposal against a mapping heuristic whose main cost function minimizes the communication energy. Results obtained in large systems, show a decrease in the maximum temperature (best case, 8%) and improvement in the

thermal distribution (best case, 50% lower standard deviation of processor temperatures). Such results demonstrate the effectiveness of the proposal. Also, a 45% increase in the lifetime of the system was achieved in the best case, using the proposed mapping.

Keywords: MPSoC, Thermal Management, Energy Management, reliability.

LISTA DE FIGURAS

Figura 1 – Variação da taxa de falhas em um chip em relação ao tempo. 20
Figura 2 – Framework de geração de MPSoC proposto por [JOV08]. 23
Figura 3 – Framework de geração de MPSoC proposto por [DIN16]. 24
Figura 4 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09]. 25
Figura 5 – Aplicação modelada como um grafo de tarefas GApp = (T, E). Tarefas Iniciais: t1, t2. Tarefas não-iniciais: t3, t4, t5, t6 27
Figura 6 - Protocolo de comunicação adotado no MPSoC HeMPS. 29
Figura 7 - Grafo da aplicação Produtor-Consumidor (a) e mapeamento desta em uma rede 2x3 (b). 30
Figura 8 - Diagrama de sequência de troca de mensagens entre processadores. 30
Figura 9 – Framework para simulação e fluxo de análise para o *MPSoC*, dividido em cinco fases. . 31
Figura 10 - Abordagem clusterizada para uma gerência distribuída de recursos. 35
Figura 11 – Protocolo para mapeamento de tarefas em um cluster vizinho. SPs brancos são PEs com recursos disponíveis. 37
Figura 12 – Exemplo de troca de mensagens durante a reclusterização 38
Figura 13 – Exemplo de reclusterização usando migração de tarefas. 39
Figura 14 - Protocolo de migração de tarefas. 40
Figura 15 – Exemplo de um MPSoC 9x9, com gerenciamento hierárquico. 41
Figura 16 – Monitoramento hierárquico de energia em um MPSoC [MAR15]. 42
Figura 17 – Protocolo de monitoramento hierárquico. 43
Figura 18 – Fluxo de tráfego para caracterizar o consumo do roteador [MAR14]. 48
Figura 19 – Elemento de Processamento, com o processador e o roteador no centro. O lado do PE é de 1 mm, e o lado do processador e do roteador é 0,3 mm. Tecnologia: 65 nm. 51
Figura 20 – Fluxo do processo de estimação de temperatura [CAS16a]. 57
Figura 21 – (a): Modelo Lateral, com par RC entre o centro de cada bloco e o centro de cada borda compartilhada. (b): Modelo RC completo, incluindo componentes verticais e laterais. 58
Figura 22 – Efeitos da temperatura (^{0}C) ao longo do tempo em um dado PE e nos seus vizinhos [CAS16a]. 59
Figura 23 – Exemplo de decaimento térmico do modelado pelo HotSpot (parte superior) e o decaimento térmico usado nesse trabalho (parte inferior). 60
Figura 24 – Exemplo do comportamento térmico com diferentes valores de potência dissipada no PE. 61
Figura 25 – Posição relativa dos PEs para o modelo baseado em LUT [CAS16a] 61
Figura 26 – Exemplo de uma estrutura de dados de uma matriz térmica [CAS16a]. 62
Figura 27 – Índice de PE para ilustrar o conceito de PE diagonal, lateral e central. 64
Figura 28 – Heurística para atualizar a temperatura atual de todos os PEs. 64
Figura 29 – Distribuição da temperatura em um MPSoC 6x6, considerando o modelo proposto (a) e o modelo HotSpot (b) [CAS16a] 67
Figura 30 – (a) Mapeamento “Energy-Aware”, (b) Mapeamento “Padrão” [CAS16a]. 68
Figura 31 – Overhead do modelo proposto (MPSoC 6x6) [CAS16a] 68

Figura 32 – Protocolo de Seleção de cluster e mapeamento das tarefas iniciais. 73
Figura 33 – Protocolo do mapeamento das tarefas não-iniciais. 74
Figura 34 – Exemplo hipotético de *region_energy.* 76
Figura 35 – Espaço de busca de mapeamento de tarefas não-iniciais, delimitado pelas linhas mais espessas. As linhas tracejadas correspondem à área inicial, sem o aumento de 1 hop. (a) espaço de busca quando uma tarefa que se comunica com t_i já está mapeada (t_1). (b) espaço de busca quando mais de uma tarefa comunicante já está mapeada (t_1 e t_2). 77
Figura 36 – Ciclo do sistema proposto por essa Tese. 84

LISTA DE TABELAS

Tabela 1 – Estado da Arte em *Framework* de Geração de MPSoC. 24
Tabela 2 – Estado da Arte em técnicas de gerência de MPSoC, comparado com a plataforma de referência. 34
Tabela 3 – Avaliação do período de monitoramento. TE: total de energia consumida no cluster (μJ). STDEV: desvio padrão da energia consumida pelos SPs dentro dos clusters (μJ). DIF: diferença entre o máximo e o mínimo consumo entre os clusters. 45
Tabela 4 – Avaliação do período de monitoramento, para o cenário 1, considerando a energia total do sistema, desvio padrão entre SPs e clusters, consumo de energia máximo e mínimo por SPs e tempo de execução. 46
Tabela 5 – Potência média em mW@100Mhz, para cada componente da *NoC*, biblioteca CORE65LPLVT (65nm), 1.2V, 25ºC. 49
Tabela 6 – Conjunto de classes de instruções e resultados obtidos a partir da simulação RTL. 52
Tabela 7 – Potência Média em mW@100MHz, para cada classe de instrução, biblioteca CORE65LPLVT (65nm), 1.2V, 25ºC. 52
Tabela 8 – Estimativa de erro da energia do processador. 53
Tabela 9 – Energia gasta em 36 PEs, em μJ (100MHz). Marcador de A até D correspondem aos PEs executando aplicações. 54
Tabela 10 – Energia gasta em 36 roteadores da NoC, em μJ (100MHz). 55
Tabela 11 – Redução de energia (μJ) usando técnica de baixa potência (LP – Low Power). 55
Tabela 12 – Exemplo de como estimar o efeito da potência média na temperatura, usando a matriz *influência_térmica*. 63
Tabela 13 – Cenários usados para avaliar a heurística que estima a temperatura em tempo de execução 65
Tabela 14 – Erro do modelo proposto em comparação ao modelo HotSpot. 66
Tabela 15 – Estado da Arte em heurísticas de mapeamento dinâmico. 70
Tabela 16 – Características dos cenários avaliados. 79
Tabela 17 – Avaliação da Total de energia/PE (Temp ➔ heurística proposta com função custo a temperatura). 80
Tabela 18 – Resultados de Temperatura dos cenários avaliados 81
Tabela 19 – Resultados de MTTF para os cenários avaliados 83
Tabela 20 – Publicações relacionadas com a Tese de Doutorado 86

LISTA DE SIGLAS

DTW.....Digital Time Warping

DVFS .....Dynamic Voltage and Frequency Scaling

GMP.....Global Manager Processor

HeMPS .....Hermes Multiprocessor System

LEC-DN..... Low Energy Consumption - Dependences Neighborhood

LMP.....Local Manager Processor

MPSoC .....Multiprocessor System on Chip

MTTF.....Mean Time to Failures

NoC.....Network-on-Chip

PE.....Processing Element

RTL.....Register Transfer Level

RAMP.....Reliability Aware Microprocessor

SP .....Slave Processor

TE .....Total de energia consumida por um elemento de processamento

VOPD .....Video Object Plane Decoder

SUMÁRIO

1 INTRODUÇÃO 19
1.1 HIPÓTESES À SEREM DEMONSTRADAS COM A TESE 21
1.2 OBJETIVOS 21
1.3 ORIGINALIDADE E CONTRIBUIÇÕES DA TESE 22
1.4 ESTRUTURA DO DOCUMENTO 22

2 PLATAFORMA DE REFERÊNCIA – MPSOC HEMPS 23
2.1 ESTADO DA ARTE 23
2.2 ARQUITETURA HEMPS (HERMES MULTIPROCESSOR SYSTEM) 25
2.2.1 Plasma-IP 26
2.2.2 NoC Hermes 27
2.2.3 Modelagem das Aplicações 27
2.2.4 Repositório de Tarefas 28
2.2.5 Comunicação entre Tarefas 28
2.3 FRAMEWORK PARA GERAÇÃO DE MPSOC 31

3 GERÊNCIA DISTRIBUÍDA DE RECURSOS 33
3.1 ESTADO DA ARTE 33
3.2 ARQUITETURA COM CONTROLE DISTRIBUÍDO DE RECURSOS 34
3.3 AUMENTO DINÂMICO DO TAMANHO DOS CLUSTERS 36
3.4 MIGRAÇÃO DE TAREFAS 38

4 MONITORAMENTO HIERÁRQUICO DE ENERGIA 41
4.1 MÉTODO HIERÁRQUICO DE MONITORAMENTO 41
4.2 MÉTODO DE MONITORAMENTO DE ENERGIA 42
4.2.1 Monitoramento de Energia em nível de PE 43
4.2.2 Monitoramento de Energia em nível de Cluster 44
4.2.3 Monitoramento de Energia em nível de Sistema 44
4.3 RESULTADOS 44

5 MODELAGEM ENERGÉTICA E TÉRMICA 47
5.1 MODELAGEM ENERGÉTICA 47
5.1.1 Estado da Arte 47
5.1.2 Modelagem energética da NoC e dos Links 48
5.1.3 Modelagem energética do Processador 51
5.1.4 Resultados 53
5.2 MODELAGEM TÉRMICA 55
5.2.1 Estado da Arte 56
5.2.2 Fluxo do Processo de Estimação de Temperatura 57
5.2.3 Calibração de temperatura 58
5.2.4 Heurística de estimação de temperatura 62
5.2.5 Resultados 65

6 ESTUDO DE DIFERENTES ESTRATÉGIAS DE MAPEAMENTO 69
6.1 ESTADO DA ARTE 69
6.2 ESTRATÉGIA DE MAPEAMENTO HIERÁRQUICO 72
6.2.1 Heurística de Seleção de Cluster 74

6.2.2 Heurística de Mapeamento das Tarefas iniciais 75
6.2.3 Heurística de Mapeamento de Tarefas não-iniciais 77
6.3 DIFERENÇAS ENTRE MAPEAMENTO DE TAREFAS PROPOSTOS 78
6.4 RESULTADOS 79
6.4.1 Avaliação de Energia 80
6.4.2 Avaliação de Temperatura 81
6.4.3 Confiabilidade 82
7 CONCLUSÃO E TRABALHOS FUTUROS 84
7.1 PUBLICAÇÕES 85
REFERÊNCIAS 87

1 INTRODUÇÃO

Com o avanço da tecnologia de fabricação de circuitos integrados, é possível criar transistores cada vez menores, possibilitando o desenvolvimento de sistemas completos em um único chip, denominados *Systems-on-Chip* (SoCs). Um SoC é um circuito integrado que implementa a maioria ou todas as funções de um sistema eletrônico completo [JER05].

Muitas aplicações requerem SoCs com vários processadores a fim de suprir seus requisitos de desempenho. Um SoC que contém diversos elementos de processamento (PEs, em inglês, *processing element*) é chamado de *Multiprocessor System-on-Chip* (MPSoC). A utilização de MPSoCs é atualmente o principal paradigma de projeto de sistemas embarcados [ZHO16]. Um MPSoC consiste de uma arquitetura composta por recursos heterogêneos, que podem incluir múltiplos processadores, módulos de hardware dedicados, memórias e um meio de interconexão [WIL09].

Os MPSoCs estão presentes em várias aplicações comercias, sendo amplamente utilizados na área de redes, telecomunicação, processamento de sinais, multimídia, entre outras [ZHO16][HOW10]. O desempenho destas aplicações pode ser otimizado se estas forem particionadas em tarefas, as quais são executadas em paralelo nos diversos recursos do MPSoC. Define-se tarefa como um conjunto de instruções e dados, com informações necessárias à sua correta execução em um dado PE.

Garantir a confiabilidade a longo prazo é um objetivo essencial para todos os fabricantes de MPSoCs. Ao mesmo tempo em que a tecnologia de semicondutores foi capaz de integrar dezenas de *cores* no mesmo chip [INT13], permitindo o aumento no desempenho computacional, ela também está acelerando o aparecimento de problemas de confiabilidade, o que resulta em uma redução na vida útil do chip [BOR1BOR18].

A introdução de novos materiais, processos e dispositivos, aliada às limitações de aumento de tensão e ao crescimento do consumo de energia no chip [COS09][EBI12][CHA13], impõem muitos desafios em relação a confiabilidade. Especificamente, os dois principais desafios para manter a confiabilidade são: (1) o aumento contínuo no tamanho do *die* e no número de transistores; (2) a constante redução do tamanho dos transistores visando desempenho [SHA15]. Mais transistores resultam em uma maior probabilidade de ocorrerem falhas, que resultam em menos tempo de vida do chip. O aumento do consumo de energia e o aumento na densidade de transistores estão causando temperaturas mais altas nos circuitos integrados, resultando na aceleração do surgimento de falhas.

A confiabilidade em processadores está diretamente relacionada com a sua temperatura de operação e espera-se que muitos problemas de confiabilidade venham a surgir devido a altas temperaturas dos mesmos. A relação entre confiabilidade e falha é tipicamente dada pela relação de Arrhenius, que é derivada da dependência observada nas taxas de reação química nas mudanças de temperatura. Tal relação modela a redução de vida de um circuito quando opera sob valores de temperaturas mais elevados em comparação com sua temperatura normal de funcionamento [DIA12].

O modelo de Arrhenius nos mostra que a vida útil de um processador diminui exponencialmente com o aumento de sua temperatura. Por exemplo, hotspots no processador irão resultar em uma maior taxa de falhas nessas áreas, o que influencia diretamente no envelhecimento e desempenho do sistema [COS09][EBI12][CHA13][KUM11][HEN13]. As falhas relacionadas ao desgaste de circuitos integrados são fenômenos já extensivamente abordados na literatura [JSS11].

A confiabilidade de um chip a longo prazo pode ser representada pela curva da banheira, ilustrada na Figura 1, a qual mostra a taxa de falhas de um chip ao longo do tempo.

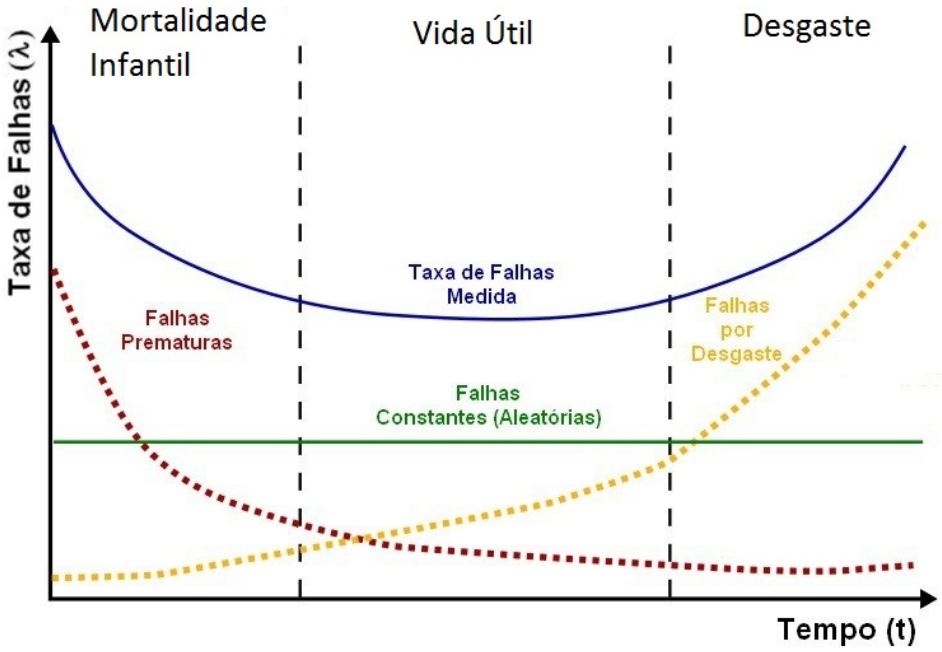

Figura 1 – Variação da taxa de falhas em um chip em relação ao tempo.

A curva da banheira é composta por três curvas individuais, relacionadas à mortalidade infantil (início de vida), vida útil e desgaste. Cada região é caracterizada por diferentes tipos de falhas. As falhas no início de vida são resultado de defeitos relacionados ao processo de fabricação, as quais reduzem com o passar do tempo. No tempo de vida útil, os defeitos são causados por falhas aleatórias. As falhas no período de desgaste são causadas basicamente por desgaste dos materiais. São quatro os principais efeitos do desgaste nos circuitos integrados: eletromigração; migração por estresse; ruptura dielétrica dependente do tempo (TDDB); e ciclos térmicos.

Pelo efeito que a temperatura exerce no chip, o gerenciamento térmico é reconhecido como uma das características essenciais em um MPSoC [YI14]. Um sistema sem gerenciamento térmico faz com que o chip fique exposto a variações térmicas sem controle, o que pode significar uma diminuição do desempenho devido ao atraso de interconexões [AJA05] e redução da confiabilidade devido ao alto gradiente térmico [VIS00] [SKA04]. A variação de temperatura não é somente uma preocupação espacial, mas também temporal, pois frequentes mudanças abruptas na frequência e na tensão, produzidas por ciclos térmicos, podem aumentar a taxa de falhas do sistema [HAA06]. Além disso, o aumento da temperatura conduz ao aumento do custo de resfriamento e

encapsulamento [GUN01].

O gerenciamento térmico em MPSoCs é realizado tanto em hardware como em software. No nível de hardware pode-se obter as informações necessárias para o gerenciamento térmico utilizando técnicas de monitoramento do sistema. Tais informações podem ser obtidas por meio de sensores de temperatura. Utilizando tais sensores, se obtém o valor de temperatura do sistema mediante acréscimo de hardware, sendo que a precisão de tal técnica depende da qualidade e da quantidade de sensores. Outra forma de se obter informações térmicas do sistema é usando um modelo em software, que através de um monitoramento de energia consumida, é capaz de estimar e informar em tempo de execução os valores térmicos de cada unidade do sistema, técnica utilizada nesta Tese. Já no nível de software, o sistema pode usar técnicas como escalonamento de tarefas, migração de tarefas entre PEs ou mapeamento de tarefas, para atuar e reduzir os valores de temperatura.

Por esses motivos, o gerenciamento térmico e energético são desafios críticos e um dos principais fatores que devem ser levados em conta no momento de realizar o projeto de um MPSoC [ZAN12].

1.1 Hipóteses à serem demonstradas com a Tese

Esta Tese busca demonstrar duas hipóteses: *(i)* é possível estimar a temperatura de um MPSoC em tempo de execução com precisão e baixo custo computacional; e *(ii)* mapeamento de tarefas tendo como função custo a temperatura instantânea do chip pode aumentar a confiabilidade do sistema.

1.2 Objetivos

Com a intenção de demonstrar as hipóteses definidas acima, o objetivo estratégico desta Tese é, primeiramente, definir uma técnica de estimativa de temperatura a qual seja possível implementá-la em um MPSoC, com baixo custo computacional para operar em tempo de execução. Tendo uma técnica definida, o segundo objetivo estratégico desta Tese é utilizar uma heurística de mapeamento de tarefas que tenha como função custo a temperatura instantânea do chip com o intuito de aumentar a confiabilidade do sistema.

Para atingir os objetivos estratégicos, os seguintes objetivos específicos devem ser atendidos:

- Definir um monitoramento capaz de obter informações de dissipação de potência de forma eficiente e com baixo custo;
- Criar um modelo térmico que seja capaz de estimar a temperatura do chip em tempo de execução;
- Criar uma heurística de mapeamento de tarefas que seja capaz de utilizar os dados do modelo térmico como função custo, a fim de buscar uma melhor distribuição térmica no MPSoC.

1.3 Originalidade e Contribuições da Tese

A originalidade desta Tese baseia-se na criação de um modelo térmico para MPSoC, capaz de, através de um monitoramento hierárquico, fornecer informações quanto à temperatura instantânea do chip com baixa granularidade (i.e., no nível de PE) e precisão, assim como baixo custo computacional, e com tais informações, agir sobre o sistema a fim de aumentar sua confiabilidade.

A presente Tese apresenta duas principais contribuições: *(i)* criação de um modelo térmico baseado em software, que auxilia o processador gerente do MPSoC, em tempo de execução, a tomar decisões sem a inserção de hardware adicional e com baixo impacto computacional [CAS16a]; *(ii)* propor uma heurística de mapeamento de tarefas que atue no MPSoC a fim de aumentar sua confiabilidade, usando como função custo as informações térmicas obtidas pelo modelo citado anteriormente [CAS16b]. Essas contribuições são detalhadas a seguir:

- Desenvolvimento de um framework para a criação e simulação de MPSoCs, com o intuito de facilitar o desenvolvimento e os testes das técnicas propostas por essa Tese.
- Desenvolvimento de um sistema de monitoramento hierárquico, capaz de obter informações de consumo energético de cada PE do MPSoC, e enviá-las para processadores gerentes, sem sobrecarregá-los. Para isso, foi utilizada uma abordagem com três diferentes tipos de PEs: escravo, gerente local e gerente global. Essa diferença entre as funções de cada PE permite ao sistema trabalhar hierarquicamente.
- Desenvolvimento de um modelo térmico para MPSoC, que usa o sistema de monitoramento hierárquico descrito anteriormente para colher dados, capaz de estimar a temperatura instantânea do chip em tempo de execução. A precisão de tal modelo foi comparada com o modelo térmico HotSpot e obtidos resultados similares.
- Desenvolvimento de uma heurística de mapeamento de tarefas, que é alimentada com os dados obtidos pelo modelo térmico desenvolvido nessa Tese, com o objetivo de testar a usabilidade do modelo proposto e criar uma ferramenta que auxilie o gerenciamento de MPSoC a aumentar sua confiabilidade.

1.4 Estrutura do Documento

O Capítulo 2 descreve a plataforma HeMPS, utilizada como MPSoC de referência nesta Tese. O Capítulo 3 apresenta a inserção de uma gerência distribuída de controle no MPSoC de referência. O Capítulo 4 apresenta o mecanismo de monitoramento hierárquico de energia. O Capítulo 5 apresenta a modelagem de energia e de temperatura. O Capítulo 6 apresenta as diferentes estratégias de mapeamento. Por fim, o Capítulo 7 conclui este manuscrito. Os Capítulos 4, 5 e 6 correspondem às contribuições originais desta Tese.

2 PLATAFORMA DE REFERÊNCIA – MPSOC HEMPS

Neste Capítulo é feita uma revisão do estado da arte em técnicas de gerência de MPSoC e dos *frameworks* de geração de MPSoC (Seção 2.1). Além disso, o Capítulo apresenta a arquitetura do MPSoC homogêneo HeMPS, utilizado como plataforma inicial deste trabalho (Seção 2.2) bem como o *framework* de geração de MPSoC desenvolvido [CAS14] (Seção 2.3).

O MPSoC HeMPS [WOS07] [CAR09], foi utilizado como referência de arquitetura, e sobre o mesmo foram realizadas modificações detalhadas no Capítulo 3. Já o *framework* utilizado foi totalmente desenvolvido no contexto desta Tese.

2.1 Estado da Arte

Nessa Seção é feita uma revisão no estado da arte em *frameworks* de geração de *MPSoC*, realizando também um comparativo com o *framework* desenvolvido.

Em [JOV08], usa-se como entrada para o framework que gera o MPSoC um arquivo em linguagem XML, que gera o hardware desejado. A Figura 2 mostra o fluxo desse framework. Esse framework tem como vantagem a possibilidade de escolha de diversos tipos de rede de intercomunicação, tais como, *Mesh*, *Torus* e *Spidergon*. Como desvantagens, pode-se citar o tempo de simulação, que no caso do VHDL é maior do que do SistemC RTL, e a falta de um mecanismo de análise e depuração.

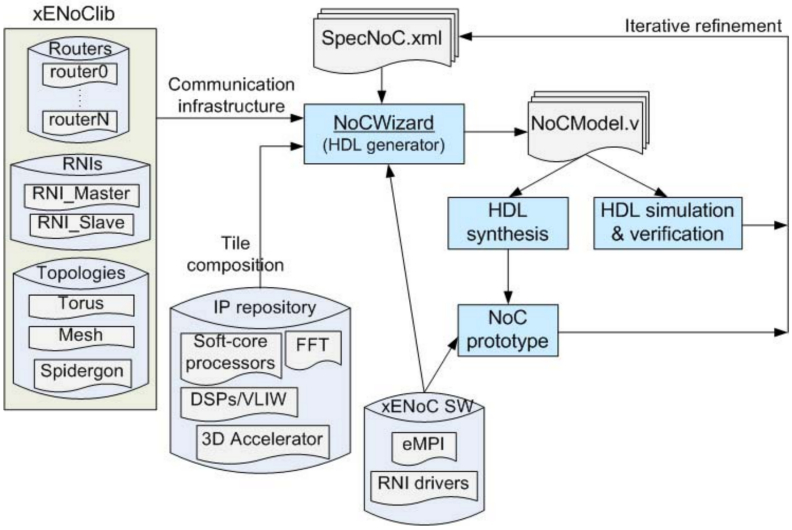

Figura 2 – Framework de geração de MPSoC proposto por [JOV08].

Em [DIN16], usa-se a linguagem XML em conjunto com a linguagem JAVA para gerar um modelo de MPSoC. Esse modelo é abstrato, e usa os conceitos de orientação à objetos para criar cada elemento do MPSoC, como podemos ver na Figura 3. A vantagem dessa abordagem é a facilidade de se criar configurações novas alterando parâmetros relativos a memórias, processadores e rede. A desvantagem deste modelo abstrato é não possuir a precisão de ciclo encontrada em um sistema baseado em SystemC RTL.

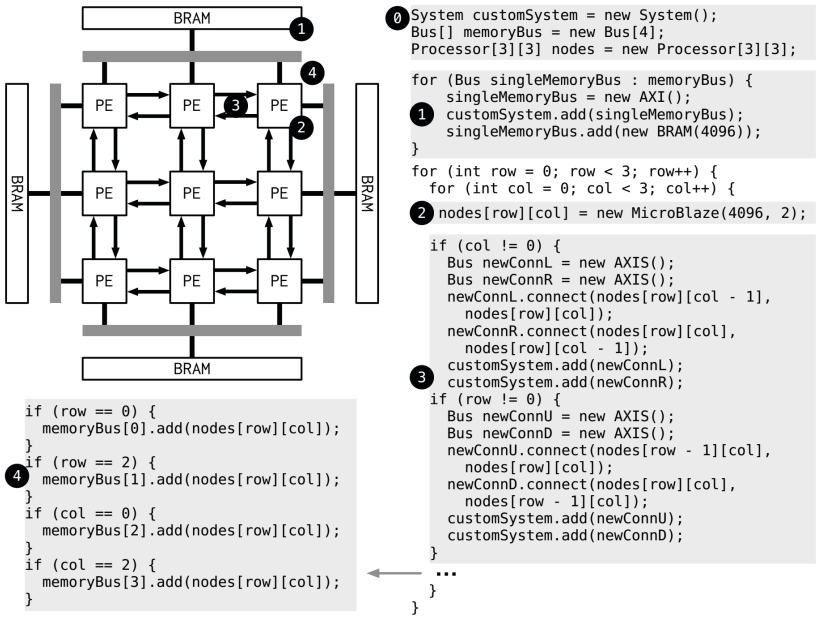

Figura 3 – Framework de geração de MPSoC proposto por [DIN16].

A Tabela 1 compara características relevantes dos trabalhos revisados, tais como: linguagem de descrição da plataforma; arquitetura de interconexão; tipo de processador; forma de depuração; precisão; dentre outras. A maioria dos trabalhos revisados, adotam a linguagem SystemC para modelar os seus sistemas. Entretanto, nenhuma das propostas gera um sistema com precisão em nível de ciclo, o que é um diferencial da plataforma de referência desse trabalho, já que o modelo SystemC foi derivado de um modelo VHDL sintetizável, o que garante a precisão em nível de ciclo. Dois trabalhos revisados usaram para descrever o modelo da plataforma a linguagem XML, em conjunto com outras, como VHDL ou JAVA.

Tabela 1 – Estado da Arte em *Framework* de Geração de MPSoC.

Ref.	Ling. de descrição	Interconexão	Tipo de processador	Depuração	Precisão	Mecanismo do Modelo	Tamanho máximo da rede	Tempo de Simulação
[ANG06]	SystemC	Memória compartilhada	Modelo de processado LISA 2.0	Interface gráfica		Baseado em eventos	Depende da dimensão da memória compartilhada	
[ROT11]	SystemC	NoC	MIPS		Quasi Cycle Accurate Level	Baseado em eventos		1797s para um MPSoC 6x6 com 4 tarefas
[JOV08]	VHDL/XML	NoC	NIOSII soft-core			Baseado em ciclos	Tamanho parametrizável de acordo com as dimensões da NoC	
[LEM12]	SystemC TLM	NoC	MIPS e 2 VLIW		7% comparada ao RTL	TLM		0,35s para 1 PE executando uma FFT
[DIN16]	JAVA/XML	Barramento/ NoC	ARM				6x6	
Framework de Referência [CAS14]	**SystemC RTL**	**NoC**	**MIPS**	**Relatórios de Desempenho**	**Precisão de ciclo**	**Baseado em eventos**	**Até 48x48**	**286s para um MPSoC 8x8.**

O Framework proposto nesta Tese permite alterar diversas configurações do MPSoC a fim de proporcionar opções de parametrização ao projetista. Exemplos de tais parâmetros são: o tamanho do sistema; número de tarefas por PE; técnicas de gerenciamento; heurísticas de mapeamento; entre outros parâmetros.

Além disso, diferentemente dos demais trabalhos revisados, o framework proposto permite avaliar com rapidez plataformas com um grande número de PEs. Por exemplo, uma simulação de um MPSoC com 64 PEs por 20 ms de tempo de execução requer em torno de 286 segundos de tempo de máquina para terminar a simulação (Intel Xeon 2.93 GHz de 8 cores, com 32GB de RAM). Assim, as características que destacam o framework proposto em relação aos trabalhos revisados incluem a precisão de ciclo do modelo (ciclo de relógio) e o baixo tempo para avaliar o desempenho de plataformas com um grande número de PEs.

2.2 Arquitetura HeMPS (Hermes Multiprocessor System)

Os principais componentes da arquitetura HeMPS são os elementos de processamento, denominados Plasma-IP, que são interconectados pela NoC Hermes [MOR04], utilizada como infraestrutura de comunicação. Além disso, tem-se uma memória externa, denominada de repositório de tarefas. Na Figura 4 é ilustrada uma instância da arquitetura HeMPS, utilizando uma NoC Hermes de dimensão 2x3 interconectando os Plasmas-IP.

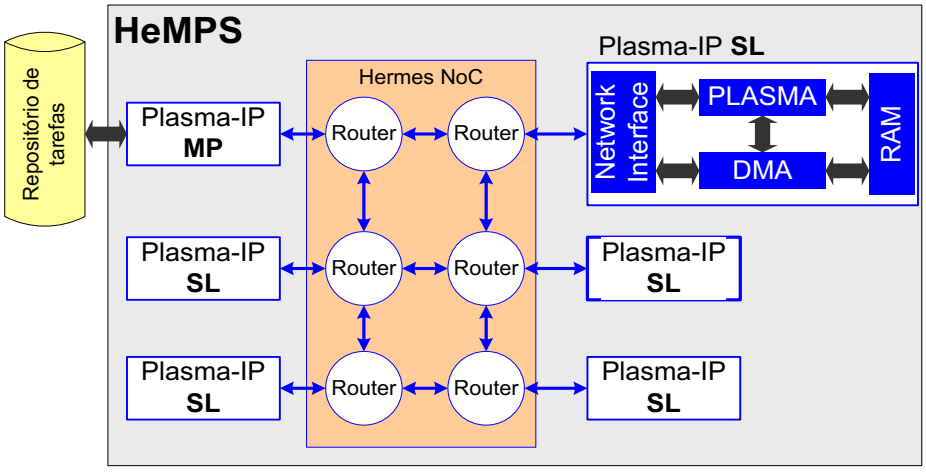

Figura 4 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09].

As características relevantes do MPSoC HeMPS incluem:

- *Processamento homogêneo*. Todos os PEs possuem a mesma arquitetura. Justifica-se o emprego de um MPSoC homogêneo para simplificar a geração dos códigos objetos e a migração de tarefas. Um MPSoC heterogêneo implica em diferentes códigos objetos para a mesma aplicação, ou tradução dinâmica de código [OST12]. Estudo relativo a MPSoCs heterogêneos foi o tema abordado da Dissertação "*Integração de Novos Processadores em Arquiteturas MPSoC: Um Estudo de Caso*" [WAC11].
- *Utilização de rede intrachip (NoC)*. Justifica-se o emprego desta arquitetura de comunicação dada a escalabilidade da mesma e à possibilidade de múltiplas

comunicações entre PEs ocorrerem simultaneamente [PAS08].

- *Memória distribuída*. Cada processador possui uma memória privada, responsável por armazenar instruções e dados. Optou-se por esta arquitetura, pois a mesma reduz o tráfego de dados no meio de comunicação, e não requer memórias cache externas [WOL12].
- *Comunicação por troca de mensagens*. Justifica-se o emprego de troca de mensagens, pois este é o mecanismo natural de comunicação em um sistema com memórias distribuídas. A utilização de memória compartilhada é mais adequada a sistemas que utilizam barramento, com poucos PEs [PAT11].
- *Organização de memória paginada*. Este mecanismo possui as seguintes desvantagens, comparado à organização de memória segmentada: restrição do tamanho máximo das tarefas ao tamanho da página, desperdício de memória caso as tarefas possuam tamanho menor que a página (fragmentação interna). Porém esta escolha simplifica tanto o processo de mapeamento quanto o de migração de tarefas, pois uma página de memória é vista como um recurso de processamento de tarefas, podendo a tarefa ser mapeada em qualquer página livre. Não são necessários mecanismos complexos de gerência de memória.

2.2.1 Plasma-IP

O Plasma-IP, elemento de processamento do MPSoC HeMPS, possui duas instâncias distintas: gerente, denominado de Plasma-IP MP, responsável pela gerência dos recursos do sistema; e escravo, denominado de Plasma-IP SL. O MPSoC HeMPS contém apenas um Plasma-IP MP, pois possui uma gerência de recursos centralizada. O Plasma-IP contém os seguintes componentes:

- processador Plasma [PLA11]: é um processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS. O processador Plasma do MPSoC HeMPS possui algumas modificações em relação ao processador Plasma original, como por exemplo, a criação do mecanismo de interrupção, exclusão de módulos (UART) e inclusão de novos registradores mapeados em memória.
- memória privada (RAM): contém o sistema operacional, denominado *µkernel,* executado pelo processador Plasma. No caso dos Plasma-IP SL, a memória é dividida em páginas de tamanho fixo, onde é feita a alocação de tarefas.
- interface de rede (NI): realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.
- módulo de acesso direto à memória (DMA): possibilita o processador continuar a execução de tarefas sem controlar diretamente a troca de mensagens com a rede. O DMA (do inglês, *Direct Memory Access*) tem como principal função transferir o código-objeto de tarefas que chegam à interface de rede para a memória do processador e enviar para o processador gerente mensagens de depuração.

2.2.2 NoC Hermes

A interconexão dos elementos de processamento do MPSoC HeMPS é realizada através da NoC Hermes. Esta NoC é parametrizável e possui topologia malha 2D. O mecanismo de comunicação é realizado por chaveamento de pacotes, utilizando o modo *wormhole*, no qual um pacote é transmitido entre os roteadores em *flits*.

Os roteadores da NoC possuem *buffers* de entrada, uma lógica de controle compartilhada por todas as portas do roteador, um *crossbar* interno e até cinco portas bidirecionais. Estas portas são: *East*, *West*, *North*, *South* e *Local*. A porta *Local* estabelece a comunicação entre o roteador e seu núcleo local, sendo as demais portas utilizadas para conectar o roteador aos roteadores vizinhos. A arbitragem *round-robin* é utilizada pelo roteador da NoC Hermes. Essa política utiliza um esquema de prioridades dinâmicas, proporcionando um serviço mais justo que a prioridade estática. O algoritmo de roteamento utilizado é o XY, que envia os pacotes na rede primeiramente horizontalmente até chegar à coordenada X do roteador destino, e depois percorre verticalmente, até encontrar o roteador destino.

2.2.3 Modelagem das Aplicações

Uma aplicação é modelada com um grafo GApp = (T, E), onde cada vértice $t_i \in T$ representa uma tarefa da aplicação e cada aresta representa a comunicação entre tarefa t_i e t_j . O peso de uma aresta e_{ij} é denominado por $comm_{ij}$, representando o volume total de dados transferidos entre as tarefas t_i e t_j. A Figura 5 representa um exemplo de uma aplicação modelada como um grafo de tarefas. Aplicações podem ser periódicas ou aperiódicas. Se a aplicação é periódica (ex. decodificação de vídeo), o grafo de tarefas representa uma interação da aplicação.

Uma aplicação tem *tarefas inicias* (ex. t_1 e t_2) e *tarefas não-inicias*. Tarefas inicias são aquelas que inicializam a execução do aplicativo quando mapeadas no sistema. Tais tarefas não têm dependências por outras tarefas para começar a executar. Uma tarefa $t_i \in T$ contém um conjunto C_i chamado de lista de comunicação de tarefas. Esse conjunto é definido como $C_i = \{(t_j, comm_{ij}); (t_k, comm_{ik}); \ldots (t_n, comm_{in})\}$, onde cada elemento é uma tupla contendo uma tarefa t_j que comunica com t_i e o valor $comm_{ij}$, correspondendo ao total do volume transferido entre t_i e t_j em ambas as direções (ex. t_i para t_j e t_j para t_i).

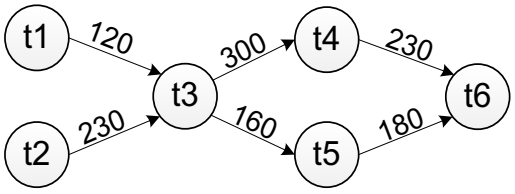

Figura 5 – Aplicação modelada como um grafo de tarefas GApp = (T, E). Tarefas Iniciais: t1, t2. Tarefas não-iniciais: t3, t4, t5, t6

Cada aplicação tem um arquivo *descritor de aplicação* que contém informações usadas para guiar as decisões de mapeamento. Tal arquivo contém: (*i*) tamanho da aplicação, que corresponde ao número total de tarefas da aplicação; (*ii*) lista das tarefas inicias; (*iii*) o conjunto C_i para cada tarefa t_i da aplicação.

2.2.4 Repositório de Tarefas

No momento de inicialização de uma dada aplicação, a heurística de mapeamento carrega no MPSoC somente as tarefas iniciais. As demais tarefas são inseridas dinamicamente no sistema em função das requisições de comunicação e dos recursos disponíveis.

O repositório de tarefas é uma memória externa ao MPSoC que contém o código-objeto de todas as tarefas que executarão no sistema. As primeiras posições do repositório de tarefas contêm os descritores de cada tarefa, com informações como identificador único, posição inicial no repositório de tarefas, tamanho da tarefa, dentre outras informações.

2.2.5 Comunicação entre Tarefas

A comunicação entre tarefas no sistema é realizada através de trocas de mensagens que ocorrem através de *pipes*. Um *pipe* é uma área de memória reservada para a troca de mensagens, alocado no *µkernel*. Nesta área são armazenadas todas as mensagens das tarefas que executam em um determinado PE. As mensagens são armazenadas de forma ordenada, e consumidas na mesma ordem.

Duas chamadas de sistema são utilizadas para a troca de mensagens: *WritePipe()* e *Readpipe().* No nível de aplicação estas chamadas de sistema são utilizadas através das primitivas *Send()* e *Receive(),* que respectivamente chamam as rotinas de *WritePipe()* e *Readpipe()* contidas no *µkernel* de um Plasma-IP.

Na implementação do *µkernel* do MPSoC HeMPS, o *Send()* é assíncrono e o *Receive()* síncrono. A principal vantagem desse método é que uma dada mensagem só é injetada na NoC se esta foi requisitada pelo receptor, reduzindo o congestionamento da rede, pois não há a possibilidade de pacotes ficarem bloqueando a rede para serem posteriormente consumidos. Para implementar um *Send()* assíncrono, um espaço dedicado de memória no *µkernel*, chamado *pipe*, armazena cada mensagem que foi escrita pelas tarefas.

A Figura 6 ilustra a comunicação entre duas tarefas, A e B, mapeadas em diferentes PEs. Quando a tarefa A executa um *Send()* (1), a mensagem é armazenada no *pipe* e a execução da tarefa A continua (2). Isso caracteriza uma escrita assíncrona não-bloqueante. Quando a tarefa B executa um *Receive()* (3), duas situações podem ocorrer. Se a tarefa destino está mapeada no mesmo processador, a tarefa executa uma leitura no *pipe* local. Se a tarefa está mapeada em outro PE, como nesse exemplo, o *µkernel* envia uma requisição de mensagem através da NoC (4) e a tarefa entra em estado de espera (estado *waiting*) (5), caracterizando uma leitura bloqueante. A tarefa A recebe a requisição de mensagem (6) e envia a mensagem através da NoC, liberando espaço no *pipe* (6). Quando a mensagem chega no PE da tarefa B (7), o *µkernel* armazena a mensagem no espaço de memória da tarefa B, habilitando a execução da tarefa B e mudando o seu estado (estado *ready*) (8).

Também existe a rotina chamada *Handler_NI()*, executada pelo *µkernel*, que trata a interrupção de *hardware* responsável pelo tratamento dos pacotes recebidos pela Interface de Rede. Assim, quando um pacote é recebido, a rotina *Handler_NI()* analisa o pacote verificando seu

serviço, desmontando-o, e utilizando suas informações para o tratamento do serviço.

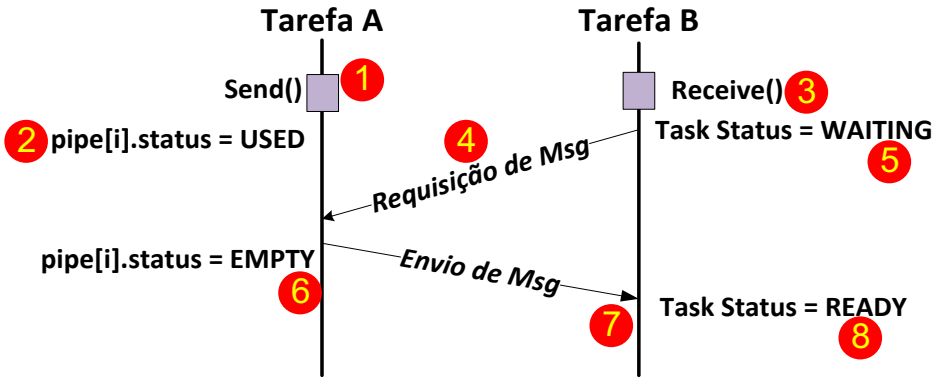

Figura 6 - Protocolo de comunicação adotado no MPSoC HeMPS.

Os serviços que o *µkernel* podem tratar incluem:

- MESSAGE_REQUEST: é enviado para realizar a requisição de uma mensagem a uma tarefa que está alocada em outro PE do sistema (direção Escravo → Escravo).
- MESSAGE_DELIVERY: contém a mensagem a ser entregue, a qual foi requisitada por um pacote de MESSAGE_REQUEST (direção Escravo → Escravo).
- TASK_ALLOCATION: é utilizado para a alocação de uma tarefa solicitada em determinado PE da rede (direção Gerente → Escravo).
- TASK_ALLOCATED: é utilizado para informar que uma determinada tarefa foi alocada no sistema (direção Gerente → Escravo).
- TASK_REQUEST: é utilizado para solicitar o mapeamento de uma tarefa. Caso a tarefa solicitada já esteja mapeada, um pacote de resposta de LOCATION_REQUEST é enviado à tarefa solicitante contendo a localização da tarefa solicitada (direção Escravo → Gerente).
- TASK_TERMINATED: é utilizado para avisar que uma tarefa terminou sua execução (direções Escravo → Gerente e Gerente → Escravos).
- TASK_DEALLOCATED: é utilizado para avisar que a tarefa terminou sua execução e pode ser liberada a página do PE em que a mesma estava executando (direção Escravo → Gerente).
- LOCATION_REQUEST: é utilizado para requisitar e responder a localização de uma determinada tarefa (direções Escravo → Gerente e Gerente→ Escravo).

Para exemplificar a operação de trocas de mensagens e os respectivos serviços, foi utilizada uma rede de tamanho 2x3, com uma aplicação produtor-consumidor (Figura 7(a)). Ambas as tarefas são alocadas estaticamente, nos processadores 02 e 12. O mapeamento desta aplicação é ilustrado na Figura 7(b). A Tarefa A executa somente a primitiva *Receive()* da Tarefa B, e a Tarefa B executa a primitiva *Send()* para a Tarefa A.

No diagrama de sequência da Figura 8, pode-se acompanhar a troca de mensagens entre os processadores. Neste diagrama, no primeiro momento, o processador Gerente, envia para os

processadores 02 e 12 o código objeto das tarefas A e B para serem alocadas (assume-se neste exemplo mapeamento estático de tarefas). O envio dessas tarefas é realizado através da mensagem contendo o serviço “TASK_ALOCATION”. Após as tarefas serem alocadas, assume-se que a Tarefa A executa um *Receive()* para a Tarefa B. Como ainda não houve comunicação entre as tarefas, o *µkernel* do processador executando a Tarefa A ainda não contém o endereço da Tarefa B. Com isso, a Tarefa A envia um “LOCATION_REQUEST” para o processador Gerente, requisitando a localização da Tarefa B. Então o processador Gerente envia para o processador 02 a localização da Tarefa B, através de outro “LOCATION_REQUEST”.

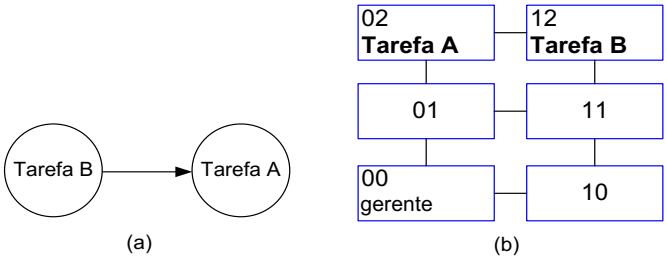

Figura 7 - Grafo da aplicação Produtor-Consumidor (a) e mapeamento desta em uma rede 2x3 (b).

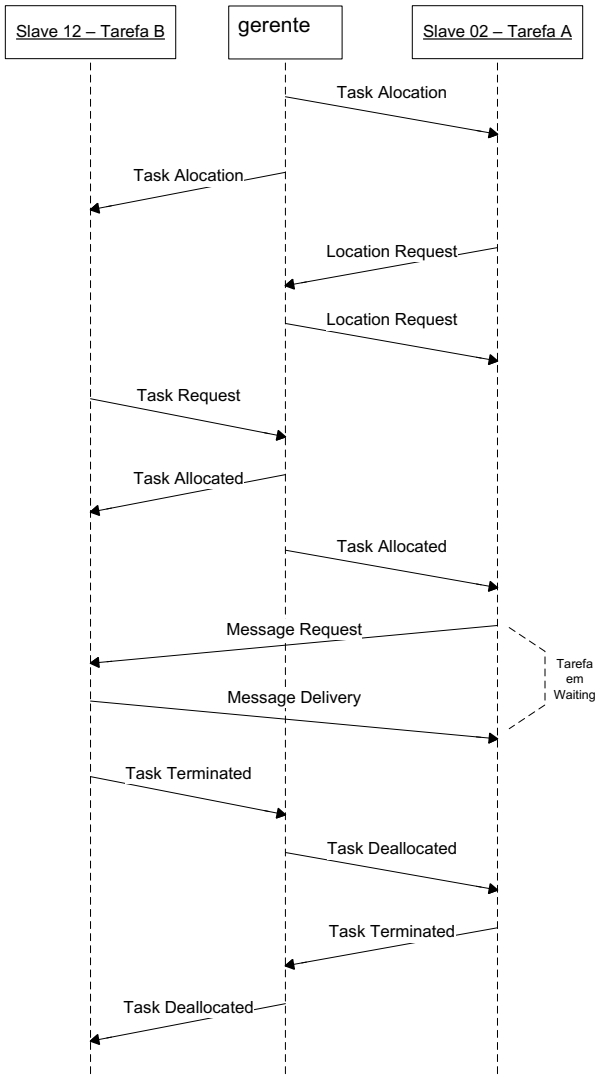

Figura 8 - Diagrama de sequência de troca de mensagens entre processadores.

Após isso, a Tarefa B faz um *Send()* para a Tarefa A. Como no caso anterior, ele não sabe onde a Tarefa A se encontra. Por convenção, no *Send()*, ou *WritePipe()*, ao invés de se enviar um "LOCATION_REQUEST", se envia um "TASK_REQUEST" para o processador gerente. Como resposta, o Gerente envia para o processador 02 e para o processador 12 uma mensagem de "TASK_ALLOCATED" com a localização de ambas as tarefas. Notar que neste caso a mensagem "TASK_ALLOCATED" para o processador 02 é redundante, pois o endereço da tarefa B já foi recebido no "LOCATION REQUEST". Isso ocorre pela ordem de execução adotada no exemplo, onde a Tarefa A iniciou primeiro. Caso fosse a Tarefa B a que iniciasse primeiro, a mensagem "TASK_ALLOCATED" eliminaria a necessidade do envio e do recebimento do serviço de "LOCATION_REQUEST", pois quando a Tarefa A fosse requisitar a localização da Tarefa B, ela já saberia a sua localização.

Na sequência, a Tarefa A envia um "MESSAGE_REQUEST" para a Tarefa B, requisitando dados. Enquanto a Tarefa A não receber a resposta da mensagem, esta tarefa fica bloqueada no estado *Waiting*. Se a Tarefa B já fez um *Send()* para Tarefa A, ele envia a reposta para Tarefa A através da mensagem de "MESSAGE_DELIVERY". Quando as tarefas terminam, estas enviam para o Gerente uma mensagem "TASK_TERMINATED", como resposta o Gerente envia para todos os processadores uma mensagem de "TASK_DEALLOCATED", informando que a tarefa terminou.

2.3 Framework para geração de MPSoC

Do ponto de vista do projetista, o *framework* é importante para validar uma nova funcionalidade inserida no sistema (ex. heurística de mapeamento) em um grande número de configurações de *MPSoC*. Já para o desenvolvedor de aplicações, o desempenho de um conjunto de aplicações executando simultaneamente na plataforma deve ser também avaliada em diferentes configurações de *MPSoC*. A Figura 9 apresenta o fluxo do Framework desenvolvido [CAS14], que permite a automatização da geração, simulação e análise do *MPSoC* de referência.

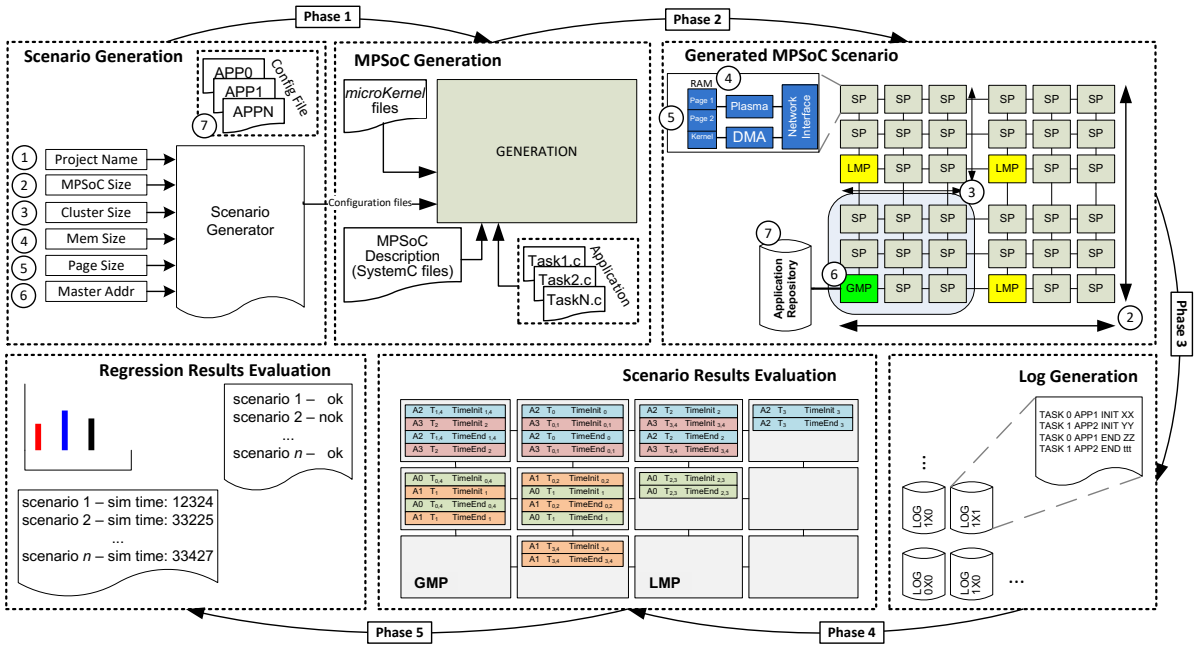

Figura 9 – Framework para simulação e fluxo de análise para o *MPSoC*, dividido em cinco fases.

Esse fluxo de geração e análise é divido em 5 fases:

Fase 1: Geração de Cenários – A primeira fase gera cenários com diferentes aplicações e diferentes configurações de *MPSoC*. A ferramenta tem por entrada: (1) nome do projeto; (2) tamanho do *MPSoC*; (3) tamanho do cluster; (4) tamanho da memória; (5) tamanho da página; (6) endereço do *GMP* (processador que tem a função de gerência global do sistema); (7) arquivo de configuração contendo o conjunto de aplicações que serão inseridas em tempo de execução.

Fase 2: Geração do MPSoC – Essa fase é responsável pela geração e compilação do hardware e software do *MPSoC*. Tal tarefa é realizada por uma ferramenta que possui as seguintes entradas: arquivos de descrição de hardware do *MPSoC, descritos em SystemC*; arquivos do *µkernel*; código em linguagem C das aplicações; e um arquivo de configuração para cada cenário. O arquivo de configuração contém os parâmetros definidos na fase 1. O resultado dessa segunda fase é um arquivo executável para cada cenário. Os números na Figura 9, que aparecem no resultado da segunda fase correspondem aos parâmetros da primeira fase. Por exemplo: (2) é o tamanho do *MPSoC*; (3) tamanho do cluster; (4) tamanho da memória; etc.

Fase 3: ***Simulação*** – A terceira fase simula cada cenário usando seu respectivo arquivo executável. Cada simulação pode ser executada sequencialmente em uma mesma estação de trabalho ou distribuídas em um *grid*. O resultado dessa fase é um relatório de desempenho para cada um dos SPs (processadores escravos), identificando o tempo inicial e final de cada tarefa/aplicação. Já o GMP, possui um relatório diferente, que contém também o tempo total de execução para o cenário simulado.

Fase 4: ***Avaliação de Resultados*** – Nessa fase, um *script* lê os relatórios de desempenho de cada cenário, ilustrando graficamente onde cada tarefa foi mapeada, juntamente com o seu tempo de execução inicial e final. Essa fase provê ao usuário uma visão geral de como as tarefas foram mapeadas no MPSoC, e o desempenho de cada aplicação. É possível também indicar para o script o tempo de simulação, para obter uma "fotografia" do sistema em um determinado momento da simulação.

Fase 5: ***Comparação entre cenários e teste de regressão*** – Na última fase, outro *script* extrai informações de desempenho dos relatórios e verifica se todas aplicações terminaram suas execuções. Essa fase também gera relatórios, gráficos e tabelas usados para realizar comparações entre diferentes cenários. Esse *script* usado na última fase é também capaz de executar automaticamente as fases 2, 3 e 4, usando como entrada os arquivos de configuração gerados pela fase 1. Por isso, esse *script* permite fazer testes de regressão, o que torna possível gerar simular inúmeras configurações de *MPSoC* sem a interferência do usuário. No final da regressão, analisando os relatórios é possível verificar se todos os cenários executaram corretamente (permite depurar a plataforma, analisando o cenário com aplicações não finalizadas), ou analisar o desempenho de um conjunto de aplicações usando diferentes configurações.

Com esse framework para a geração e simulação de MPSoC, é possível criar e analisar grandes plataformas com um tempo aceitável, tanto da criação quando da simulação.

3 GERÊNCIA DISTRIBUÍDA DE RECURSOS

Este Capítulo apresenta inicialmente uma revisão do estado da arte sobre técnicas de gerência em MPSoCs, seção 3.1. Na sequência, apresenta-se o mecanismo de gerência distribuída de recursos [CAS13], seção 3.2. Após, seção 3.3, são apresentados os mecanismos de ajuste dinâmico do tamanho do cluster, e o processo de migração de tarefas, seção 3.4.

A gerência distribuída de recursos visa garantir a escalabilidade em sistemas de grande porte. Os mecanismos de migração de tarefas e reclusterização são necessários para o desenvolvimento das técnicas que serão apresentadas nos capítulos subsequentes.

A plataforma apresentada na seção 2.2, foi modificada a fim de passar de uma plataforma com controle centralizado, executado por um único PE (gerente global), para uma plataforma com controle distribuído, controlado por gerentes locais, posicionados em regiões do MPSoC, regiões estas denominadas *clusters* [CAS13].

3.1 Estado da Arte

Nessa seção é feita uma revisão no estado da arte em técnicas de gerência de *MPSoC*, fazendo um comparativo com a plataforma de referência. No processo de revisão do estado da arte de técnicas de gerência de MPSoCs, foram encontrados trabalhos com objetivos distintos, sendo:

- Trabalhos com foco em desenvolver um controle distribuído a fim de tornar o MPSoC escalável, e compará-lo com arquiteturas centralizas ([KOB11], [SHA11], [FAT11] e [MAS16]).
- Trabalhos que exploram mapeamento distribuído de tarefas em tempo de execução, visando evitar que um único agente tenha a função de mapear tarefas ([ALF08], [CUI12], [ANA12] e [WEI11]).
- Trabalhos relacionados a migração de tarefas ([WIL15] [GOO11], [ALM10] e [WIL15]), com foco em proporcionar uma menor latência para o fluxo de comunicação criada pelo mecanismo de migração de tarefa [GOO11], apresentar uma estratégia parcialmente distribuída de migração de tarefas [ALM10] e balancear carga no sistema.

As lacunas identificadas nos trabalhos revisados em gerência e mapeamento distribuído incluem:

1. Ausência de informações relacionadas à arquitetura utilizada.
2. Modelagem abstrata do sistema não permitindo avaliação detalhada do desempenho, ou mesmo ausência de informações relacionadas à modelagem
3. Nenhum trabalho detalhou o processo de migração de tarefas, assumindo uma modelagem abstrata para o mesmo, além de nenhum método possuir uma técnica de migração de tarefas que faça a migração do contexto da mesma.

A Tabela 2 posiciona a plataforma de referência em relação ao estado da arte. A gerência de recursos é distribuída, havendo um gerente por *cluster* (terceira coluna). O sistema é subdivido em regiões (*clusters*) onde seus tamanhos são dinamicamente adaptados em função das aplicações. Há também um monitoramento de tarefas visando a desfragmentação do sistema, utilizando como mecanismo a migração de tarefas. O sistema foi implementado utilizando uma modelagem com precisão de ciclo de relógio.

Tabela 2 – Estado da Arte em técnicas de gerência de MPSoC, comparado com a plataforma de referência.

Ref.	Tipo de gerente	Tipo de gerente distribuído	Clusters	Monitoramento	Migração de tarefa	Mapeamento dinâmico	Modelagem	Objetivo do trabalho
[KOB11]	Distribuído	Aplicação	Não	Sim	Sim	Sim	C	Balanceamento de carga
[SHA11]	Distribuído	Aplicação	Não	Sim	Não	Sim	POOSL [VOE97]	Escalabilidade
[FAT11]	Distribuído	Aplicação	Não	Sim	Não	Sim	Não implementado	Escalabilidade
[ALF08]	Distribuído	Cluster	Sim	Não	Não	Sim		Escalabilidade
[CUI12]	Distribuído	Cluster	Sim	Não	Não	Sim		Escalabilidade
[ANA12]	Distribuído	Aplicação	Sim	Não	Não	Sim		Melhorar desempenho do sistema
[WEI11]	Distribuído	Implementado em todos os PEs	Não	Não	Não	Sim	Java	Reduzir congestionamento da NoC
[GOO11]	Centralizado		Não	Não	Sim (código/dado)		Xmulator [NAY07]	Consumo de energia
[ALM10]	Centralizado		Não	Não	Sim (código)		RTL (SystemC e VHDL)	Melhorar desempenho do sistema
[MAS16]	Distribuído	Aplicação	Não	Sim	Não	Sim		Melhorar desempenho do sistema
[WIL15]	Centralizado		Não	Sim	Sim (código)	Sim		Balanceamento de carga
Plataforma de Referência [CAS13]	Distribuído	Cluster	Sim	Sim	Sim (código/dado/contexto)	Sim	RTL (SystemC)	Escalabilidade / Balanceamento de carga / Consumo de energia / reclusterização

3.2 Arquitetura com Controle Distribuído de Recursos

A revisão do estado da arte da seção 3.1, identificou 4 abordagens distintas para realizar a gerência distribuída de recursos:

I. Sistema dividido em *clusters*, onde cada *cluster* é controlado por um gerente. Os *clusters* podem ser responsáveis por mais que uma aplicação e eles também podem ser reorganizados, modificando seus tamanhos em tempo de execução através de um método de reclusterização [ALF08] [CUI12].

II. O sistema é divido em *clusters* baseados no tamanho das aplicações. Um *cluster* contém somente uma aplicação [ANA12] [MAS16].

III. Um gerente de aplicação é usado como na segunda abordagem, mas esse método não usa *clusters* com tamanho predefinidos, permitindo assim o espalhamento de aplicações em PEs não contínuos [KOB11] [SHA11].

IV. Uma função global do sistema é implementada em cada PE do sistema, tal como a heurística de mapeamento de tarefas [WEI11].

A arquitetura com controle distribuído adotada na presente Tese divide o MPSoC em *n clusters* de tamanhos iguais, definidos em tempo de projeto. Em tempo de execução, se uma dada

aplicação não couber no *cluster* por falta de recursos disponíveis, o *cluster* pode requisitar recursos aos *clusters* adjacentes, sendo o processo denominado reclusterização. Tal abordagem é preferível às demais abordagens pois:

I. O número de PEs dedicados à função de gerência é limitado ao número de *clusters*. Uma abordagem com um gerente por aplicação pode implicar em uma maior sobrecarga, já que o número de aplicações que irão executar no sistema é desconhecido em tempo de execução.

II. A abordagem clusterizada reduz o número de *hops* entre tarefas pertencentes a uma mesma aplicação, reduzindo o tráfego global da NoC.

III. Não é necessário criar/destruir agentes toda a vez que uma nova aplicação entra/deixa o sistema, aumentando desse modo o desempenho global do sistema.

A Figura 10 mostra um MPSoC 9x9 contendo nove *clusters* de tamanho 3x3. O MPSoC contém três tipos de PEs: o Gerente Global (GMP), Gerente Local (LMP) e Escravo (SP). O LMP é responsável pelo controle do *cluster*, executando funções como o mapeamento de tarefas, migração de tarefas, monitoramento, verificação de *deadlines* e comunicação entre outros LMPs e GMP. O GMP contém todas as funções do LMP, e as funções relacionadas com a gerência global do sistema. Exemplos dessas funções incluem a escolha de qual *cluster* uma dada aplicação irá ser mapeada, o controle de recursos disponíveis em cada *cluster*, o recebimento de mensagens de controle e de *debug* dos LMPs, acesso ao repositório de aplicações (memória externa contendo os códigos objetos de todas as tarefas) e recebimento de requisições de novas aplicações de uma interface externa. Os SPs são responsáveis pela execução das tarefas.

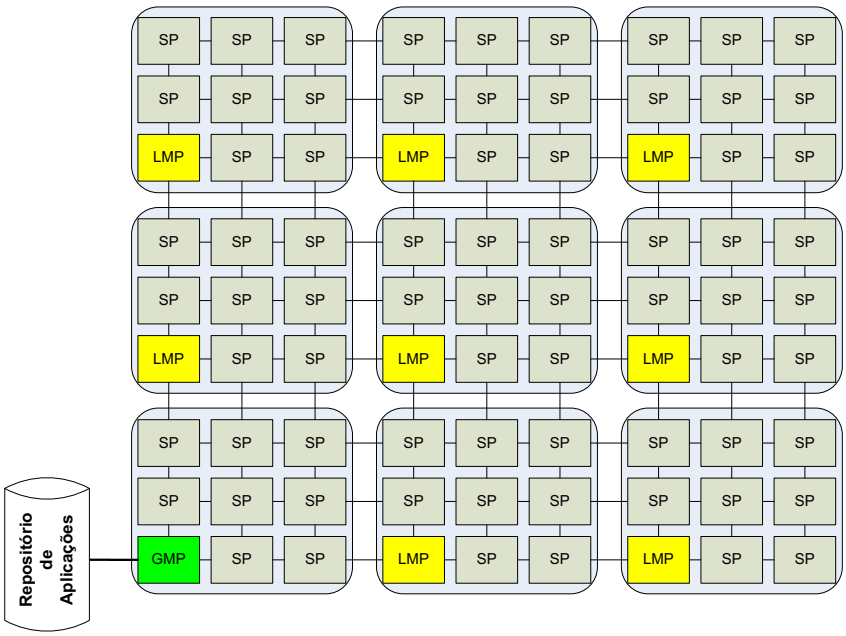

Figura 10 - Abordagem clusterizada para uma gerência distribuída de recursos.

Observar que a memória externa, anteriormente denominada "*repositório de tarefas*", denomina-se na arquitetura com controle distribuído de recursos de "*repositório de aplicações*". Esta alteração deve-se ao fato que a granularidade do mapeamento mudou de tarefa para

aplicação. Anteriormente, no controle centralizado, o processador gerente recebia requisições de mapeamento de tarefas. Na arquitetura com controle distribuído de recursos, aplicações são inseridas no sistema, e o GMP envia o descritor das novas aplicações a um determinado LMP, o qual inicia o mapeamento das tarefas.

Quando o sistema inicia, o GMP é também responsável pela inicialização dos clusters, informando aos LMPs quais regiões eles irão gerenciar. Após o LMP conhecer a região que irá controlar, ele informa a todos os SPs dessa região que ele será seu gerente. Este mecanismo de inicialização de cluster e SPs torna o sistema flexível, permitindo que o sistema mude o tamanho do cluster em tempo de execução.

O GMP é o único PE com acesso a dispositivos externos, como o repositório de aplicações. Na Figura 10, nove PEs são reservados para as funções de gerência, representando 11,1% dos PEs sem executar aplicações de usuário. Usando *clusters* de tamanho 4x4 em um MPSoC com tamanho de 16x16, esse número cai para 6,25%, o qual é um custo aceitável, considerando os benefícios obtido.

3.3 Aumento Dinâmico do Tamanho dos Clusters

Essa seção apresenta o mecanismo de reclusterização. O processo de reclusterização implica no ajuste em tempo de execução do tamanho de um determinado *cluster*. Em um primeiro momento o processo de reclusterização implica no aumento do tamanho do *cluster*, por indisponibilidade de recursos. Em um segundo momento o processo de reclusterização pode migrar tarefas que foram mapeadas em *clusters* vizinhos (*clusters* diferentes do qual a aplicação é gerenciada), com o objetivo de melhorar o desempenho das aplicações. Este segundo processo é condicionado não apenas à existência de recursos livres, mas também ao ganho de desempenho pela redução do número de *hops* entre as tarefas comunicantes. O processo de migração de tarefas é detalhado na seção 3.4

Considerar a Figura 11 como exemplo para ilustrar o processo de reclusterização. Nesta figura, o *cluster* posicionado na parte superior esquerda não possui recursos livres, enquanto os demais *clusters* possuem 1 SP disponível.

O *cluster* sem recursos disponíveis recebe em um dado momento uma solicitação de mapeamento de tarefa oriundo de um SP do seu *cluster*. Dada a ausência de recursos livres, o LMP desse *cluster* envia a mensagem *"Requisição de Empréstimo"* requisitando recursos a todos os seus *clusters* vizinhos (passo 1 da Figura 11). A mensagem de pedido de recursos, *"Requisição de Empréstimo"*, contém também o endereço do SP que requisitou o mapeamento de uma nova tarefa. Este processo de requisição é enviado em um primeiro momento para até 8 *clusters* vizinhos (quadrado envolvente do *cluster*). Caso não haja sucesso na primeira exploração, o quadrado envolvente cresce e a exploração pode continuar até todos os *clusters* serem requisitados a responderem se possuem recursos livres. Como já mencionado anteriormente, sempre haverá recurso disponível, pois a aplicação só é inserida no MPSoC se esta condição for verdadeira.

Os LMPs vizinhos ao receberam uma mensagem *"Requisição de Empréstimo"* buscam por recursos disponíveis em seus *clusters*. Se houver apenas um recurso disponível, este recurso é

reservado para ser emprestado, caso contrário, se existir mais do que um recurso disponível, o LMP irá reservar o recurso mais próximo possível, em número de *hops*, entre a tarefa a ser mapeada e a tarefa que solicitou o mapeamento. Após a reserva, todos os LMPs vizinhos, notificam a posição do recurso (passo 2 da Figura 11, SPs azuis são reservados), caso esse existir.

O LMP que requisitou recurso externo ao *cluster* escolhe o SP mais próximo de quem requisitou a tarefa, enviando uma mensagem de "*Liberar Recursos*" para liberar os recursos dos LMPs que não foram selecionados (passo 3 da Figura 11). Em seguida, o LMP envia uma mensagem de "*Requisição de alocação de tarefa*" para o GMP requisitando o mapeamento da tarefa no recurso emprestado (passo 4 da Figura 11). Portando o tamanho do *cluster* aumenta em tempo de execução, pois o recurso emprestado faz parte agora desse *cluster*. Esse processo otimiza a gerência do sistema, uma vez que as aplicações podem ser mapeadas, mesmo se o *cluster* não possuir recursos disponíveis.

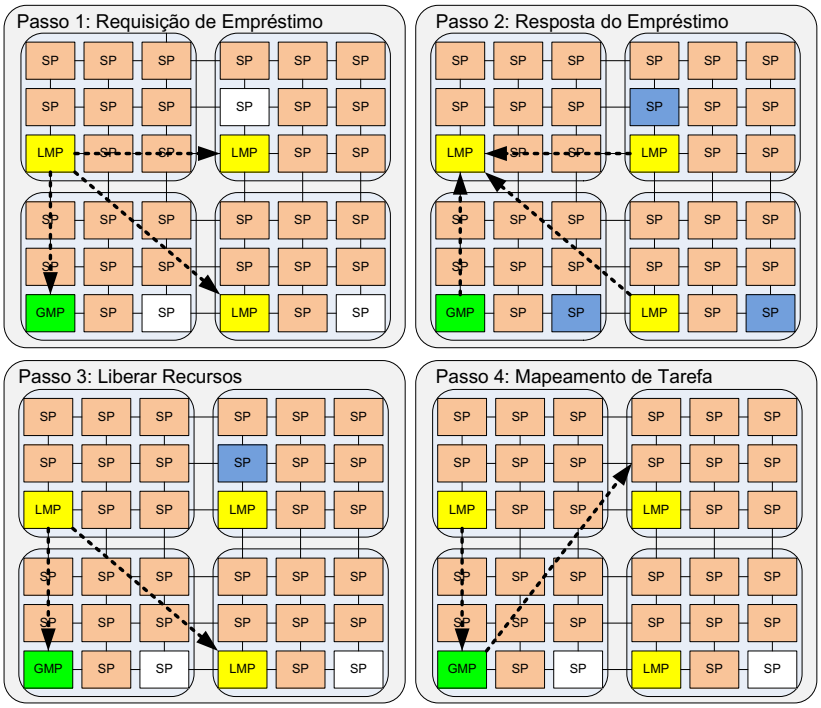

Figura 11 – Protocolo para mapeamento de tarefas em um cluster vizinho. SPs brancos são PEs com recursos disponíveis.

A Figura 12 mostra um exemplo de troca de mensagens durante o processo de reclusterização. Nesse exemplo o LMP1 não possui recursos disponíveis para mapear uma tarefa, o que o faz iniciar o processo de empréstimo de recursos. Ele faz uma requisição de empréstimo para os seus *clusters* vizinhos. O LMP de cada *cluster* vizinho busca por um PE livre que minimize a distância *Manhattan* até a tarefa requisitante (ação "mapeamento do empréstimo"). Os *clusters* vizinhos ao terminarem o "*mapeamento do empréstimo*" respondem com a localização do recurso disponível, se houver. Nesse exemplo, o LMP1 escolheu o recurso do *cluster* gerenciado pelo LMP4, liberando os recursos dos demais *clusters*. Após essa escolha, o LMP1 requisita a alocação da tarefa para o GMP que faz a alocação no SP escolhido. Lembrando que esta alocação de tarefa corresponde apenas à transmissão do código objeto, através do DMA do GMP.

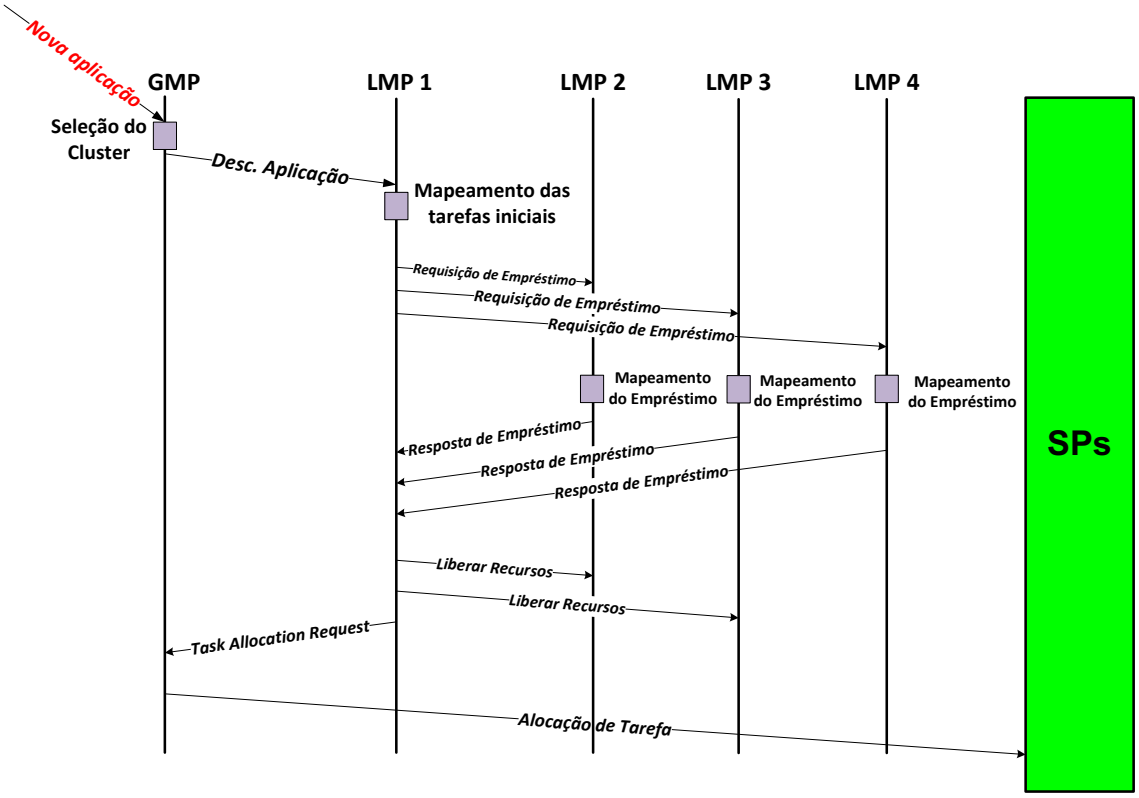

Figura 12 – Exemplo de troca de mensagens durante a reclusterização

Todas as tarefas alocadas em *clusters* vizinhos são gerenciadas pelos LMPs que gerenciam suas aplicações e não o LMP do *cluster* que a tarefa está mapeada. Quando uma tarefa termina sua execução, ela informa ao LMP que a gerencia o seu término e também informa ao LMP do *cluster* do qual está mapeada para que ele possa liberar esse recurso e redimensionar o seu *cluster*.

Apenas os LMPs têm o controle das localizações das tarefas de suas aplicações e do número de tarefas que já terminaram de uma determinada aplicação. O GMP somente tem a visão do total de recursos utilizados nos *clusters* e no sistema, sendo que somente atualizam essas informações quando alguma aplicação finaliza e não alguma tarefa. Quando todas as tarefas de uma aplicação terminam, o LMP que a gerencia informa ao GMP que essa aplicação terminou sua execução e o GMP atualiza sua tabela de recursos disponíveis no sistema.

3.4 Migração de Tarefas

Migração de tarefas pode ser usada para balanceamento de carga, tolerância a falhas e para restaurar o desempenho de uma determinada aplicação devido, por exemplo, a inserção de uma nova aplicação no sistema. No caso do presente trabalho, a migração de tarefas foi utilizada para desfragmentar o sistema, migrando as tarefas mapeadas em clusters vizinhos, com o objetivo de melhorar o desempenho das aplicações.

A migração de tarefas faz parte do processo de reclusterização e é condicionado não apenas à existência de recursos livres, mas também ao ganho de desempenho pela redução do número de hops entre as tarefas comunicantes.

A Figura 13 mostra um exemplo do uso da migração de tarefas no protocolo de reclusterização. A Figura 13(a) apresenta o mapeamento de uma determinada aplicação composta de 6 tarefas (A-F), em um cenário onde as aplicações são inseridas/removidas em tempo de execução. O MPSoC pode ter a maioria de seus recursos ocupados no momento da inserção da aplicação (PEs com a cor laranja), restringindo a procura por recursos livres pela heurística de mapeamento, o que ocasionou no mapeamento de algumas tarefas em clusters vizinhos ao cluster que gerencia a aplicação (cluster 2).

Em um dado momento, uma tarefa dentro do cluster termina a sua execução, informando ao LMP o seu término e liberando um recurso do cluster (Figura 13(b)). Após o recebimento da mensagem de término de tarefa, caso haja alguma tarefa que o LMP gerencie fora do seu cluster, ele executa uma função que calcula se é viável a migração. Essa função compara o número de hops da localização atual da tarefa até suas comunicantes com o número de hops da localização, caso ela seja migrada. Se o número de hops se tornar menor após a realização da migração, essa migração é viável.

Com isso, o LMP envia uma mensagem para a tarefa que será migrada, solicitando sua migração para o PE com o recurso livre. A tarefa então migra para o PE destino e informa ao LMP do *cluster* que está mapeada, que está migrando e que desocupará o recuso do *cluster* (Figura 13(c)).

Figura 13 – Exemplo de reclusterização usando migração de tarefas.

O protocolo de migração de tarefas é ilustrado na Figura 14 com um exemplo e pode ser resumido da seguinte forma:

1. Uma tarefa alocada no SP1 termina sua execução e avisa o LMP do seu cluster o seu término.
2. O LMP verifica se há alguma tarefa que ele gerencia fora do cluster. Caso exista, ele verifica se

a migração dessa tarefa é viável, ou seja, há uma redução no número de hops entre as tarefas comunicantes.

3. Caso a migração seja viável, o LMP envia uma mensagem de requisição de migração para essa tarefa, nesse exemplo, localizada no SP2.
4. A tarefa só pode ser migrada se está no estado de execução. O escalonador do *µkernel* verifica essa condição. Se a tarefa pode ser migrada, o *µkernel* envia para o PE destino, um pacote com o conteúdo completo da página da tarefa e seus descritores de tarefas (TCB – Task Control Block). O detalhamento do processo de migração de tarefas é descrito em [RUA17].
5. A tarefa migrada é escalonada uma vez que o código e o TCB forem completamente recebidos.
6. No exemplo, uma tarefa comunicante envia uma requisição de leitura para a tarefa que foi migrada, mas para o SP onde ela não está mais alocada.
7. Quando essa requisição de leitura chega, ela é repassada para o SP onde se encontra a tarefa.
8. O SP1 então recebe a requisição de leitura e responde à requisição, já com a sua nova posição.

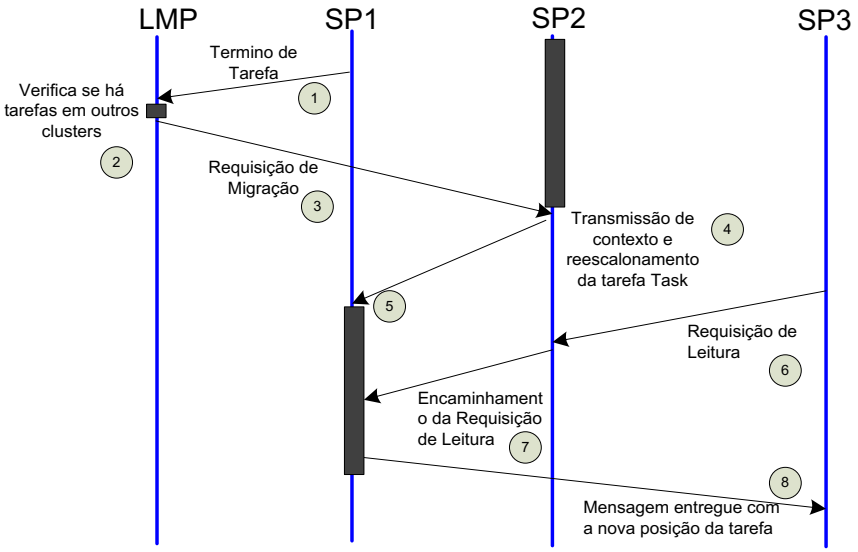

Figura 14 - Protocolo de migração de tarefas.

O passo ‘4’ do processo de migração pressupõe que a tarefa esteja no estado de execução, ou seja, não há *receive*() pendente. Esta é uma condição importante, pois garante que a tarefa a ser migrada não está esperando dados de outra tarefa. Caso estivesse aguardando dados de outra tarefa, o PE da tarefa poderia receber dados durante a migração, levando à perda de informações.

Ainda neste contexto, há o controle de entrega de mensagens. O processo que garante que as mensagens serão consumidas na ordem correta é o armazenamento local das tarefas produzidas. Quando a tarefa é migrada, as mensagens produzidas pelos *send()* permanecem armazenadas no *pipe* do PE original. Portanto, as tarefas que se comunicarem com a tarefa migrada ainda utilizam o endereço da tarefa antes da mesma migrar. Uma vez que todas as mensagens sejam consumidas, as requisições de comunicação são encaminhadas para a nova posição da tarefa, que responde essas requisições com o novo endereço. A partir deste momento, as tarefas que se comunicam com a tarefa migrada utilizam o novo endereço.

4 MONITORAMENTO HIERÁRQUICO DE ENERGIA

O gerenciamento hierárquico em MPSoCs, consiste em ter níveis diferentes de gerenciamento, com o intuito de limitar o escopo de cada gerente, reduzindo o custo computacional e aumentando a escalabilidade do sistema. Este capítulo apresenta inicialmente o método hierárquico de monitoramento (seção 4.1), o método de monitoramento de energia (seção 4.2) e a avaliação deste método (seção 4.3).

4.1 Método Hierárquico de Monitoramento

Conforme apresentado no Capítulo anterior, para permitir que o sistema de gerenciamento hierárquico opere corretamente, o sistema é divido em regiões virtuais denominadas *clusters*, conforme apresentado na Figura 15.

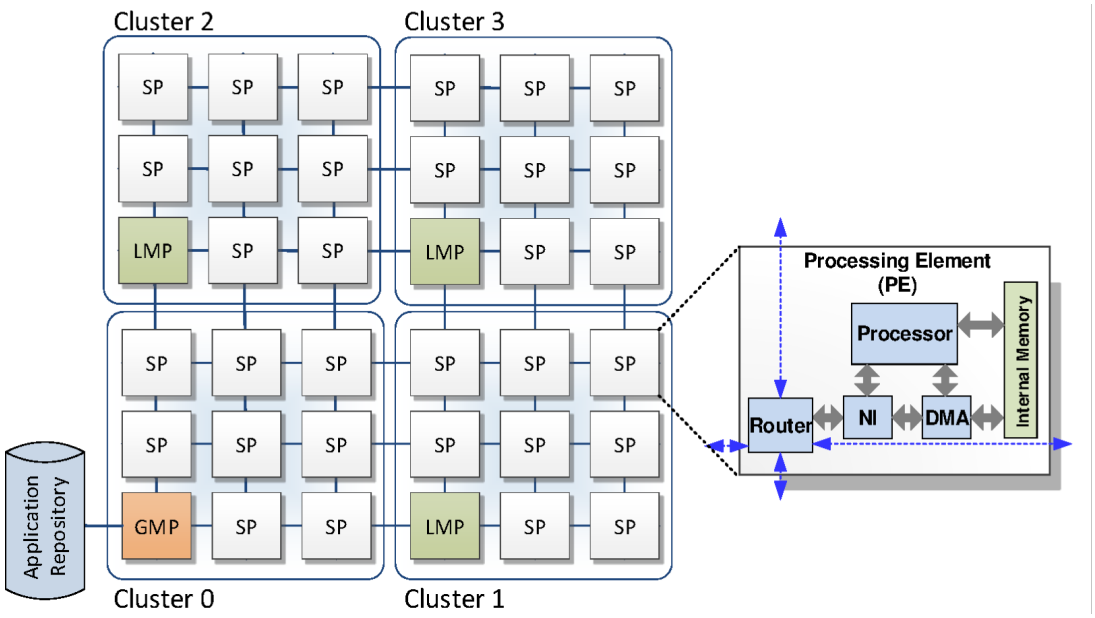

Figura 15 – Exemplo de um MPSoC 9x9, com gerenciamento hierárquico.

Para isso, os elementos de processamento assumem três papéis distintos:

- **Elemento de processamento Escravo (SP)**. SPs executam tarefas de aplicações. Cada SP executa um *µkernel*, que suporta a comunicação entre PEs, execução de tarefas simultaneamente (multi-tarefa) e software de interrupção (*traps*). Cada SP pode executar um número máximo de tarefas simultaneamente (MAX_SP_TASKS), que corresponde ao número de páginas de memória no SP menos 1 (SP_PAGES -1).
- **Elemento de processamento Gerente Local (LMP)**. Responsável pelo controle do cluster, executando funções tal como mapeamento de tarefas, migração de tarefas e reclusterização (processo detalhado na Seção 3.3).
- **Elemento de processamento Gerente Global (GMP)**. Um único PE é responsável pela gerência global do sistema. Essa gerência inclui a definição do mapeamento das aplicações no cluster, controle de acesso a dispositivos externos (ex. repositório de aplicações). Além

disso, o GMP gerência um cluster (por exemplo, o cluster inferior a esquerda da Figura 15), executando todas as funções de um LMP.

4.2 Método de Monitoramento de Energia

O método de monitoramento de energia é hierárquico, como mostra a Figura 16, de acordo com a arquitetura anteriormente apresentada. O esquema proposto resulta em um baixo tráfego na NoC, com uma pequena intrusividade no hardware e no software do PE.

O monitoramento é dividido em três níveis. O nível inferior de monitoramento é aplicado nos SPs, que calculam a energia consumida, enviando periodicamente a energia por processador para o seu LMP. No próximo nível, o LMP recebe os dados de energia dos seus SPs correspondentes para calcular a energia do cluster. Finalmente, os LMPs enviam para o GMP, no nível superior da hierarquia, a energia por cluster. Os SPs do cluster do GMP (Figura 16) enviam os dados de energia por processador diretamente para o GMP. Assim, o GMP pode estimar o total de energia gasta pelo sistema, porque recebe dados de energia de todos os LMPs e dos SPs que ele controla.

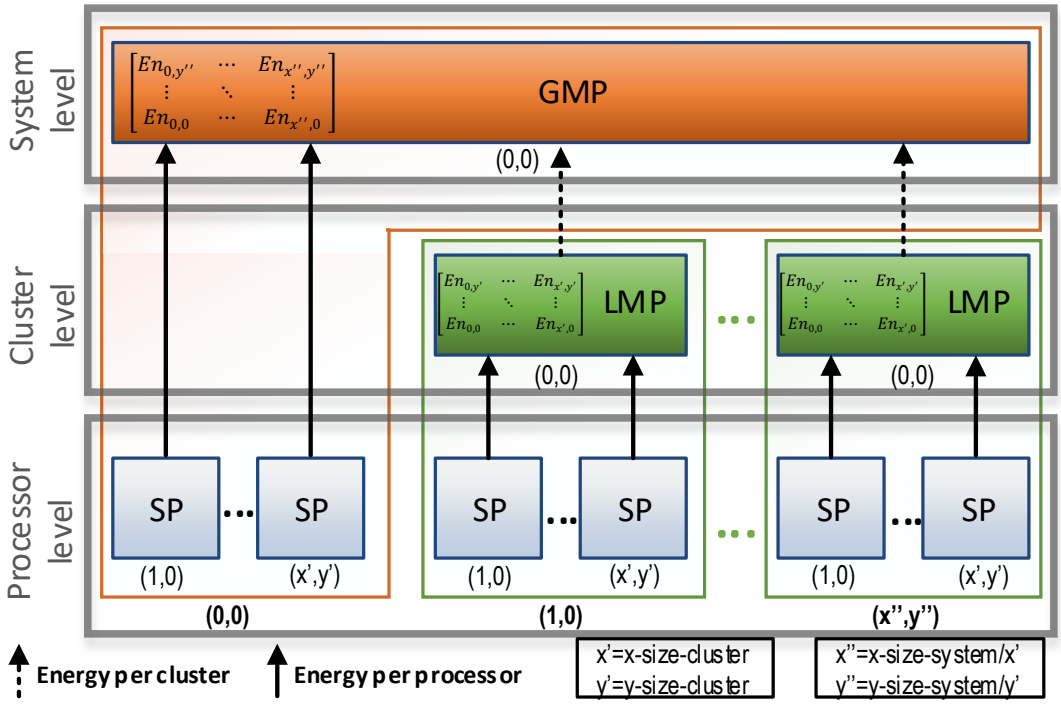

Figura 16 – Monitoramento hierárquico de energia em um MPSoC [MAR15].

A Figura 17 ilustra o protocolo de monitoramento hierárquico, onde, os SPs do MPSoC enviam periodicamente para os seus respectivos LMP, pacotes de monitoramento contendo o quanto o PE (processor e roteador) consumiu, e o LMP atualiza sua tabela de energia. Este processo por ocorrer internamente ao cluster, é chamado de monitoramento intra-cluster.

Paralelamente, o LMP informa para o GMP a quantidade total de energia consumida pelo cluster. Esta atualização ocorre quando uma tarefa requisita um mapeamento, quando uma aplicação terminar ou periodicamente a critério do desenvolvedor. Este processo de monitoramento por ocorrer além das fronteiras do cluster, é chamado de inter-cluster.

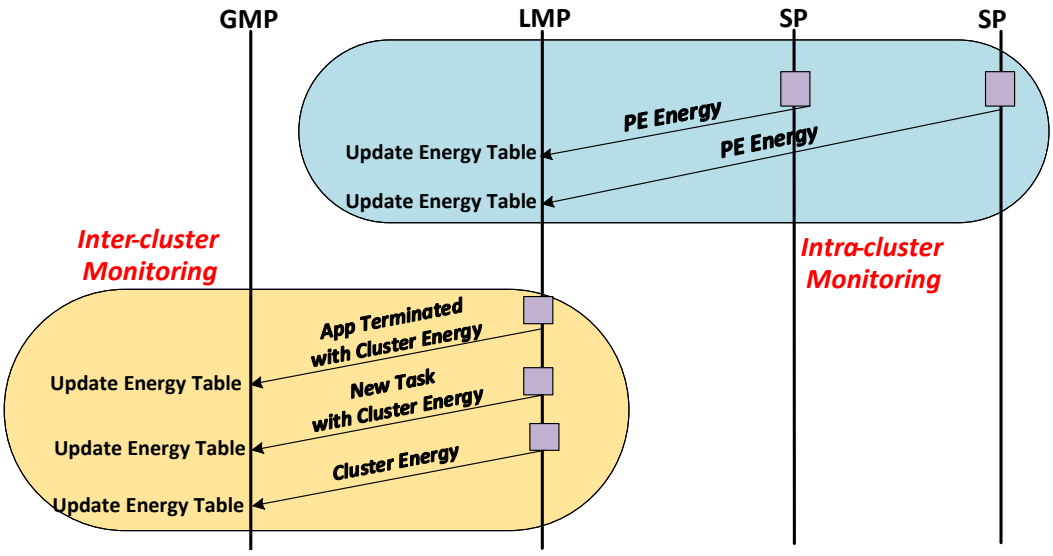

Figura 17 – Protocolo de monitoramento hierárquico.

4.2.1 Monitoramento de Energia em nível de PE

A seção 5.1 apresenta o método para estimar potência e energia, através da caracterização do processador por um processo de calibração. O conjunto de instruções é agrupado em classes e para cada classe de instrução, uma simulação RTL do código *assembly* da classe correspondente é executado no *netlist* do processador. Esse método apresenta um pequeno erro em comparação a medições reais (abaixo de 5%), fornecendo uma medida de energia por instrução para todas as classes de instrução. O método de monitoramento de energia proposto usa esta abordagem nos PE, requerendo pequenas adaptações no hardware e no software.

- **Requisitos de hardware**. Contadores de instruções são adicionados ao hardware do PE. Na implementação atual, esses contadores estão incluídos na parte de controle do processador. Se o hardware do processador não puder ser modificado, um *sniffer* pode ser adicionado no barramento de acesso à memória de instruções. Registradores de propósito específico armazenam o número de instruções executadas por classe, sendo continuamente atualizados.
- **Requisitos de software:** No *µkernel* novas funções leem os contadores de instruções periodicamente, de acordo com uma janela de amostragem, para calcular a energia consumida pelo processador. Multiplicando a energia por instrução pelo número de instruções, o *µkernel* estima a energia do processador nesta janela. Potência média, energia acumulada, temperatura são exemplos que podem ser estimados a partir da energia por instrução e amostragens de tempo.

Observe que o método proposto não assume a existência de nenhum sensor no hardware. Além disso, o requisito de hardware é restrito aos contadores de instruções, tornando este método pouco intrusivo e facilmente adaptável a diferentes processadores.

4.2.2 Monitoramento de Energia em nível de Cluster

O LMP recebe informações relativas à quantidade de energia que cada SP consumiu durante a *janela de monitoramento*. O *µkernel* periodicamente contabiliza a energia gasta por cada SP, transmitindo o valor obtido para o LMP. Note que os LMPs conhecem a carga de trabalho (energia consumida) de cada SP, o que permite que os LMPs executem heurísticas para distribuir a carga de trabalho uniformemente ao longo do tempo.

Esse processo provoca uma pequena quantidade de tráfego na NoC, sendo esse tráfego, local em cada cluster. Além disso, como o número de SPs em cada cluster é pequeno (tipicamente entre 16 e 20), a carga computacional para tratar os pacotes de monitoramento em cada LMP é pequena. Por um lado, o número de pacotes de controle aumente com a diminuição da *janela de monitoramento*, sobrecarregando o LMP. Por outro lado, grandes janelas de monitoramento, atrasam o cálculo de energia consumida pelos SPs, podendo levar o LMP a tomar decisões erradas, por exemplo de mapeamento. A seção 4.3 discute este compromisso, avaliando diferentes *janelas de monitoramento*.

O *µkernel* do LMP executa o monitoramento de energia no nível de cluster e não há necessidade de modificações de hardware nesse nível.

4.2.3 Monitoramento de Energia em nível de Sistema

No topo do esquema de monitoramento de energia, o GMP recebe informações relativas a quantidade de energia consumida por cada cluster. Sempre que uma comunicação entre LMP e o GMP ocorrer, a informação sobre a energia do cluster é inserida no pacote à ser enviado. Tal abordagem evita a sobrecarga do GMP com as mensagens de monitoramento. Duas mensagens em que a informação de monitoramento é inserida são:

- *NewTask* – o LMP solicita a alocação de uma nova tarefa;
- *AppTerminated* – o LMP informa o fim de uma dada aplicação. O LMP envia essa mensagem quando todas as tarefas de uma dada aplicação terminam suas execuções.

A execução de tarefas por períodos longos de tempo, poderiam levar a um longo tempo sem atualizar o GMP, subestimando a quantidade de energia dos clusters. Portanto, cada LMP notifica periodicamente o GMP, com o consumo de energia de cada cluster. Esse período de tempo do monitoramento inter-cluster é maior que do monitoramento intra-cluster. Note que o GMP conhece somente o total de energia gasta por cada cluster, não possuindo uma visão detalhada da distribuição de energia dentro dos cluster.

4.3 Resultados

Foram realizados experimentos para avaliar o consumo de energia variando os períodos de monitoramento. Para isso, foram realizados testes com 7 cenários diferentes, com 5 variações de período de monitoramento, totalizando 35 cenários. A Tabela 3 apresenta a avaliação do consumo de energia de cada cluster, variando o período de monitoramento, usando um cenário com tamanho de MPSoC 8x8 (60 SPs), tamanho de cluster de 4x4 e 780 tarefas. Todos os demais cenários

avaliados obtiveram o mesmo padrão de resultados.

Com o período de monitoramento intra-cluster pequeno, o número de pacotes de monitoramento cresce, sobrecarregando o LMP. Nesse caso, alguns pacotes de monitoramento sofrem atrasos, e o LMP pode tomar uma decisão levando em conta dados do passado (ex. alguns SPs não foram atualizados pois alguns pacotes de monitoramento não foram tradados), fazendo o LMP tomar decisões erradas sobre mapeamento de tarefas. Por outro lado, com períodos de monitoramento grandes, SPs podem receber novas tarefas e a energia consumida ainda não foi atualizada. Com um período de monitoramento intermediário, todos os pacotes de monitoramento são recebidos e tratados, sem incorrer em uma atualização demorada induzida por períodos longos de monitoramento. Observe a linha DIF, que corresponde a diferença entre o consumo máximo e mínimo entre clusters. O período de monitoramento 1ms/3ms leva a uma melhor distribuição de carga entre os clusters.

Tabela 3 – Avaliação do período de monitoramento. TE: total de energia consumida no cluster (μJ). STDEV: desvio padrão da energia consumida pelos SPs dentro dos clusters (μJ). DIF: diferença entre o máximo e o mínimo consumo entre os clusters.

	Variação de períodos de monitoramento intra/inter cluster									
	0.25ms / 3ms		0.5ms / 3ms		**1ms / 3ms**		2ms / 3ms		4ms / 8ms	
	TE	STDEV	TE	STDEV	**TE**	**STDEV**	TE	STDEV	TE	STDEV
Cluster 0	4.247	46	3.818	51	**2.607**	**30**	2.609	56	2.567	34
Cluster 1	2.512	28	2.215	31	**2.479**	**22**	2.196	31	2.412	22
Cluster 2	2.408	37	2.434	33	**2.433**	**36**	2.541	40	2.788	31
Cluster 3	1.676	26	2.083	15	**2.476**	**33**	2.390	27	2.592	42
DIFF	2.571		1.735		**174**		413		376	

A Tabela 4 avalia diferentes parâmetros de desempenho para diferentes períodos de monitoramento. Essa tabela usa o mesmo cenário da tabela anterior, e novamente, ela representa o comportamento dos demais cenários. Os resultados nessa tabela mostram:

- *Distribuição da carga de trabalho* (linhas 1 até 3). O desvio padrão de energia entre todos os SPs varia de 31 μJ no melhor caso à 72 μJ no pior caso, enquanto a energia máxima consumida varia entre 234 e 390 μJ. Vale notar, que no período de monitoramento 1ms/3ms a energia mínima consumida é maior do que os demais, o que significa que o balanceamento de carga foi melhor realizado.
- *Tempo de Execução* (linha 4). Pequena diferença entre as variações de período.
- *Energia Consumida* (linha 5). Quanto menor o período de monitoramento, maior a energia consumida.
- *Tráfego da NoC* (linha 6). Quanto menor o período de monitoramento, maior o número de flits.

Tabela 4 – Avaliação do período de monitoramento, para o cenário 1, considerando a energia total do sistema, desvio padrão entre SPs e clusters, consumo de energia máximo e mínimo por SPs e tempo de execução.

	Variação de períodos de monitoramento intra/inter cluster				
	0.25ms / 3ms	0.5ms / 3ms	**1ms / 3ms**	2ms / 3ms	4ms / 8ms
STDEV entre todos SPs (μJ)	72	58	**31**	41	34
Consumo Max SP (μJ)	390	372	**234**	269	249
Consumo Min SP (μJ)	66	98	**111**	88	68
Tempo de Execução (ms)	260	234	**234**	240	233
Total System Energy (μJ)	10.842	10.549	**9.996**	9.736	10.358
N# of flits (10^6)	18,815	16,655	**15,666**	15,539	14,887

Esse trabalho adota 1 e 3 ms como período de monitoramento intra-cluster e inter-cluster, respectivamente. Esses valores foram adotados porque eles apresentam uma melhor relação entre distribuição de carga de trabalho e energia consumida.

5 MODELAGEM ENERGÉTICA E TÉRMICA

Esse capítulo apresenta os métodos propostos para a modelagem energética e modelagem térmica. A proposta é obter dados de energia dos processadores, e em posse desses dados, estimar as temperaturas de cada PE ao longo do tempo usando um modelo térmico.

5.1 Modelagem energética

Essa subseção apresenta o modelo energético da NoC e do PE, com a análise da estimativa de erro apresentada em [MAR14].

5.1.1 Estado da Arte

Diversos trabalhos propõem diferentes modelos para estimar o consumo de potência e energia em uma NoC. Ye et al. [YE02] apresentam um framework para estimar a potência dissipada considerando o custo de energia para transmitir um bit de uma porta de entrada para uma porta de saída de um roteador. Chan et al. [CHA05] utilizam regressão linear para estabelecer uma relação entre os eventos que ocorrem em cada componente do roteador com o consumo de energia. Já Meloni et al. [MEL07] mostram um método para formular a dissipação de energia com base em componentes arquiteturais, implementação e parâmetros de tráfego. Enquanto Guindani et al. [GUI08] propõe um modelo para estimar a potência com base nas taxas de recepção médias em cada buffer do roteador, constituído por um passo de calibração seguido por sua aplicação. O objetivo desses trabalhos é fornecer métodos para explorar o projeto, visando a minimização de energia.

No nível do processador, Tiwari et al. [TIW94] descrevem uma metodologia para estimar a dissipação de energia do processador de acordo com a energia consumida por cada instrução. Gupta et al. [GUP10] extraem os valores de potência das instruções, de uma caracterização feita no nível de portas lógicas, integrando-os em um Simulador de Conjunto de Instruções (ISS). Ambos os trabalhos estabelecem a dissipação de potência do processador como uma função da energia consumida por cada instrução.

Atilallah et al. [ATI07] fornece um modelo genérico para estimar a dissipação de potência no início do fluxo de projeto para MPSoCs. Os Autores combinam os modelos de potência em um simulador da arquitetura. A modelagem do processador considera dois estados, executando e aguardando, sendo a dissipação de potência para esses dois estados diferentes. Os Autores também modelam a memória cache e um *crossbar* como infraestrutura de interconexão. Os testes são apresentados usando uma aplicação de codificador H.263 com sistemas com 4 até 16 processadores. O objetivo de sua experiência é avaliar o trade-off entre tamanhos de cache, tempo de execução e dissipação total de potência.

Mediante essa pesquisa, há uma lacuna na literatura sobre modelos de energia para MPSoCs baseadas em NoC, usando troca de mensagens como o modo de comunicação entre aplicações. O presente trabalho supre esta lacuna, modelando o consumo de energia para o NoC e os elementos de processamento.

5.1.2 Modelagem energética da NoC e dos Links

O modelo energético de uma NoC é baseado em um processo de caracterização. Este é processo de caracterização da NoC compreende 4 passos.

5.1.2.1 Geração de tráfego

Os componentes que fazem parte do roteador são o buffer de entrada, *crossbar* e a lógica de controle (responsável pelo roteamento e pela arbitragem). Para caracterizar um determinado componente, é necessário ter o máximo de atividade de chaveamento nas portas lógicas. Portanto, para caracterizar o perfil de potência de um determinado roteador é necessário excitar todos os componentes internos, e fornecer um *payload* com uma grande distância de *Hamming* entre os *flits* para induzir uma grande atividade de chaveamento nas portas lógicas do roteador. A Figura 18 mostra o fluxo de tráfego usado para caracterizar o roteador 11 (roteador central), em uma NoC 3x3. Esse roteador tem cinco buffers de entrada, cada um recebendo *flits* de um determinado fluxo.

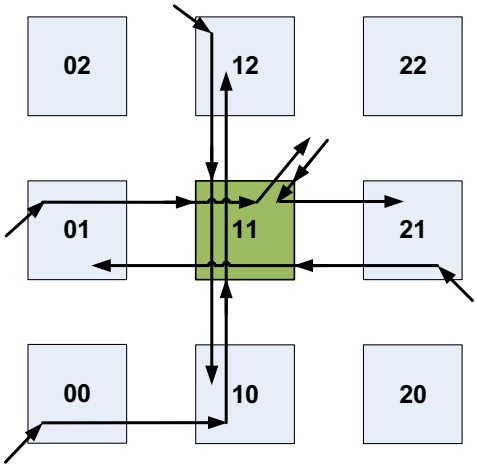

Figura 18 – Fluxo de tráfego para caracterizar o consumo do roteador [MAR14].

Cada fonte de fluxo de tráfego injeta 1000 pacotes de 32-flits, com uma distância de *Hamming* entre *flits* superior a 80%. A dissipação de potência de um roteador é uma função da taxa de recepção nos buffers [GUI08]. Foram realizadas 6 simulações como casos de testes, com taxas de injeção variando entre 0% (*idle*) à 50% da largura de banda do *link*. Por exemplo, para uma taxa de injeção igual a 50%, para pacotes de 32-bits, cada pacote é injetado a cada 64 ciclos de relógio.

5.1.2.2 Síntese Lógica

O roteador central é sintetizado (usando a ferramenta *rc* da Cadence), aplicando restrições de temporização e otimização de potência para uma determinada tecnologia. É também importante considerar na síntese lógica o modelo dos fios, para obter uma estimativa correta das capacitâncias parasitas devido ao roteamento. O resultado da síntese é o *netlist* do roteador central, o qual substitui a descrição *VHDL RTL* original do roteador.

5.1.2.3 Simulação com diferentes taxas de injeção

O terceiro passo do processo é a simulação da *NoC* (usando a ferramenta *incisive* da Cadence), com o *netlist* do roteador central e o arquivo *SDF* (contendo os dados de atraso e temporização obtidos após da síntese lógica). Cada caso de teste simulado gera um arquivo *TCF* (*Toggle Count Format*), com a atividade de chaveamento do roteador central.

5.1.2.4 Análise de potência

O quarto passo corresponde à análise de dissipação de potência (usando a ferramenta *rc* da Cadence), utilizando os arquivos *TCF*. A análise de potência reportou 97,35% de atividade de chaveamento, demonstrando uma correta geração de tráfego. O relatório de potência contém detalhes de dissipação de potência para cada componente interno da *NoC*. Um exemplo de análise de potência é resumido na Tabela 5, com a potência dissipada para os componentes internos do roteador, e a potência total média dissipada para o roteador de 5 portas.

Tabela 5 – Potência média em mW@100Mhz, para cada componente da *NoC*, biblioteca CORE65LPLVT (65nm), 1.2V, 25ºC.

Rate (%)	Buffer	Crossbar	Control Logic	5 port router
0	30,25	0,31	27,08	**205,28**
10	49,33	4,51	32,49	**322,03**
20	68,27	8,75	37,85	**437,93**
30	87,32	12,63	43,19	**554,39**
40	106,02	16,62	48,44	**668,76**
50	124,45	20,46	53,56	**781,16**

Para completar o processo de caracterização, uma regressão linear é feita usando os resultados acima, resultando em uma equação para cada coluna da Tabela 5. A equação de potência para o roteador resultou em um r^2=0,99995, validando a dependência linear da potência média com a taxa de injeção. Note que o *buffer* é o componente responsável pela maior dissipação de potência no roteador. Para um roteador de 5 portas, a dissipação dos buffers é maior do que 80% da potência total.

No contexto do MPSoC, a seguinte suposição é feita: quando um pacote é injetado na rede, ele é transmitido em rajada, com taxa de injeção igual a 100%, caso contrário o link está em *idle*, usando uma taxa igual a 0%. Essa suposição é correta desde que os PEs injetem pacotes na NoC usando um módulo de DMA, resultando na injeção de um *flit* por ciclo de relógio. Essa suposição pode introduzir um erro nas estimativas, uma vez que o congestionamento não está sendo levado em consideração. Esse erro pode ser negligenciado porque a atual taxa de injeção no MPSoC é pequena (abaixo de 10%). Por exemplo, em [YUH11] a carga de tráfego observado pela simulação dos benchmarks SPLASH-2 foi de apenas 0,55%.

Essa suposição resulta em dois valores de energia. A equação 1 corresponde à energia ativa correspondente para a transmissão de um *flit* através de um *buffer*, enquanto os outros permanecem em *idle*. A equação 2 corresponde à energia em *idle* para o buffer em estado de *idle*.

Notar que as equações 1 e 2 calculam a energia por ciclo de relógio.

$$E_{active} = \left[\left(n_{ports} - 1\right) * P_{buffer}(0) + P_{buffer}(1) + P_{crossbar}(1) + P_{control_{logic}}(1)\right] * T \quad (1)$$

$$E_{idle} = \left[\left(n_{ports}\right) * P_{buffer}(0) + P_{crossbar}(0) + P_{control_{logic}}(0)\right] * T \quad (2)$$

onde: n_{ports} é o número de portas do roteador, $P_{componente}(0)$ é a potência média sem tráfego, $P_{componente}(1)$ é a potência média com taxa de injeção igual a 100% e T o período do relógio.

Aplicando as equações 1 e 2, usando os dados da Tabela 5, para o roteador de 5 portas obtém-se:

$$E_{active} = 4{,}610\ pJ \quad (3)$$

$$E_{idle} = 1{,}786\ pJ \quad (4)$$

Para obter a energia consumida em cada roteador, é necessário determinar a quantidade de ciclos de relógio que o roteador transmitiu dados (ativo), e a quantidade de ciclos de relógio em *idle*. Modelos baseados em volume, como em [HU03], consideram somente os *flits* transmitidos. O modelo criado considera toda a potência dissipada do roteador, nos modos ativo e *idle*.

A equação 5 calcula o número de ciclos de relógio usado por um determinado roteador na transmissão de *flits*. A quantidade de ciclos é proporcional à quantidade de *flits* que atravessam um determinado roteador, mais o número de pacotes multiplicado pela constante k, representando o número de ciclos consumido para rotear e arbitrar um pacote entrando no roteador (na *NoC* utilizada esse valor corresponde a 5 ciclos de relógio). O período restante é o tempo que o roteador está em *idle*, equação 6, onde $Ciclos_{sim}$ é o número total de períodos de relógio da simulação.

$$Ciclos_{active} = \sum flits + (\sum pacotes) * k \quad (5)$$

$$Ciclos_{idle} = Ciclos_{sim} - Ciclos_{active} \quad (6)$$

Finalmente, as equações 7 e 8 computam a energia total consumida e a potência dissipada por um roteador *k*, respectivamente.

$$E_{rot_k} = (E_{active} * Ciclos_{active} + E_{idle} * Ciclos_{idle}) \quad (7)$$

$$\overline{P_{rot_k}} = \frac{E_{rot_k}}{n\_ciclos\_simulados * T} \quad (8)$$

Para validar o método de estimativa, um cenário de tráfego com as seguintes características foi inserido no NoC: distribuição temporal Pareto On-Off (momentos de rajada e de *idle* caracterizam a Distribuição Pareto), 1000 rajadas com pacotes de 34-flits. O Pareto On-Off foi escolhido já que corresponde ao comportamento esperado no MPSoC. O número de ciclos simulados foi de 173.733 ciclos, resultando em $Ciclos_{active}$ = 39.000 e $Ciclos_{idle}$ = 139.733. Segundo a equação 7 a energia estimada é 429.393,75 pJ (58,13% corresponde ao consumo em idle). Usando a equação 8, a potência média estimada é 240,2431 μW, enquanto a potência média obtida pela ferramenta *rc* foi de 240,26 μW, demonstrando a precisão do método.

Para calcular a energia consumida nos links (fios), é necessário determinar a capacitância do fio entre dois roteadores. Usando a mesma tecnologia, o layout de um PE completo foi sintetizado, conforme ilustrado na Figura 19.

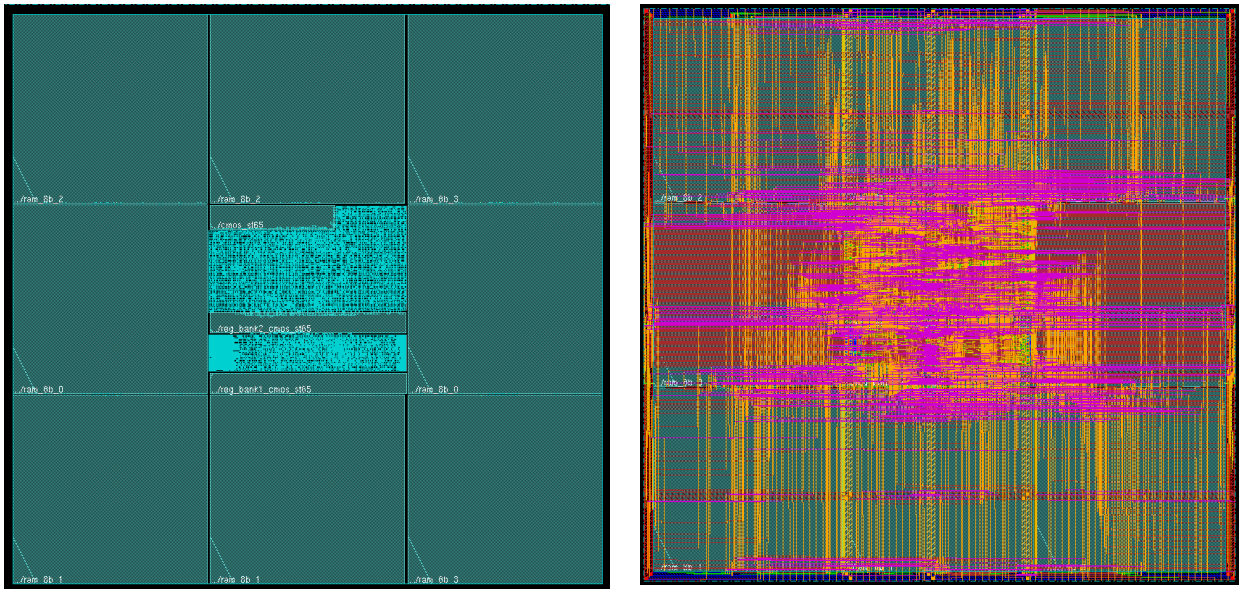

Figura 19 – Elemento de Processamento, com o processador e o roteador no centro. O lado do PE é de 1 mm, e o lado do processador e do roteador é 0,3 mm. Tecnologia: 65 nm.

Cada fio da NoC tem aproximadamente 1 mm de comprimento, correspondendo a uma capacitância igual a 2194 fF. Com a tensão de alimentação igual a 1,2V e largura de flit de 16-bit, cada link consome 4,21248 pJ (E_{link}) para carregar totalmente ou descarregar. Para calcular a energia de cada link de um determinado roteador, é necessário multiplicar E_{link} pelo número de flits transmitidos em cada porta de saída do roteador (exceto a porta local, que possui fios muito curtos), e pela atividade de chaveamento médio estimado nos links (α) – equação 9. O valor assumido de α é 0,4 (o pior caso medido em tráfego real).

$$E_{wire_k} = E_{link} * \left(\sum_{norte,sul,leste,oeste} n_{de\_flits} \right) * \alpha \tag{9}$$

5.1.3 Modelagem energética do Processador

O processo para caracterizar a energia do processador também depende de calibração. Inicialmente, o conjunto de instruções é dividido em classes [TIW94], e um programa é escrito (em linguagem *assembly*) para cada classe de tal forma de excitar todas as instruções da classe e os registradores internos.

A modelagem de energia do processador inicia com uma simulação RTL, com um *testbench* capaz de contar o tipo e o número de instruções executadas, bem como contar o número de ciclos de relógio para executar o programa que está sendo simulado.

A Tabela 6 mostra os resultados obtidos a partir da simulação RTL do processador Plasma (MIPS ISA). O *testbench* gerou os resultados da segunda e terceira colunas da Tabela. A quarta coluna da Tabela 6 corresponde ao CPI (ciclo por instrução) médio para cada classe de instrução,

permitindo estimar o tempo de execução em um nível de abstração maior. A quinta coluna é a medida da *qualidade* do programa para estimar a potência/energia para uma determinada classe de instruções. As classes *Branch* e *Jump* tem o menor percentual de instruções nos programas devido à inserção de instruções *NOP*.

Tabela 6 – Conjunto de classes de instruções e resultados obtidos a partir da simulação RTL.

Classes	Número de instruções	Ciclos simulados	CPI	% de inst. por classe
arithmetic	73.643	73.657	1,0002	98,90
logical	108.643	108.657	1,0001	99,41
shift	46.417	46.431	1,0003	99,53
move	60.025	60.039	1,0002	98,96
load_store	70.745	137.259	1,9402	94,99
mult_div	159.757	160.021	1,0017	94,52
nop	25.457	25.471	1,0005	99,58
branch	220.017	220.031	1,0001	47,72
jump	220.03	220.031	1,0000	45,45

O processador é então sintetizado (usando a ferramenta *rc* da Cadence), seguido por uma simulação de com atraso de porta (usando a ferramenta *incisive* da Cadence) e estimativa de potência (usando a ferramenta *rc* da Cadence), como no processo de calibração da NoC. A tabela Tabela 7 mostra para cada classe de instrução a medida da potência média, energia total e energia por instrução (obtida pela divisão da energia pelo número de instruções simuladas). Para as classes *branch* e *jump* a energia é calculada considerando a energia da instrução *NOP*.

Tabela 7 – Potência Média em mW@100MHz, para cada classe de instrução, biblioteca CORE65LPLVT (65nm), 1.2V, 25^{o}C.

Classes	Potência Média (mW)	Energia Total (nJ)	Energia por inst.
arithmetic	2,605	1.918,76	26,05
logical	2,269	2.465,43	22,69
shift	2,175	1.009,87	21,76
move	2,110	1.266,82	21,10
load_store	2,293	3.147,35	44,49
mult_div	2,263	3.621,28	22,67
nop	1,467	373,70	14,68
branch	3,114	6.851,77	31,24
jump	1,851	4.072,77	20,30

A equação 10 calcula a energia para uma determinada classe de instruções.

$$E_{classe} = P_{classe} * Ciclos * T \quad (10)$$

onde: P_{classe} é a potência média para uma determinada classe de instruções (segunda coluna da Tabela 7), *Ciclos* é o total de ciclos simulado (terceira coluna da Tabela 6), T o período do relógio.

As equações 11 e 12 calculam a energia total consumida e a potência dissipada para um processador, respectivamente.

$$E_{processador} = \sum_{i=0}^{n\_classes} n\_instruções_i * E_{classe_i} \quad (11)$$

$$\overline{P_{processador}} = \frac{E_{processador}}{\sum_{i=0}^{n\_classes} n\_instruções_i * CPI_{classe_i}} \quad (12)$$

onde: $n\_instruções_i$ é o número de instruções executadas para uma dada classe, E_{classe} é a energia por instrução para uma determinada classe, CPI_{classe} é o CPI para uma determina classe.

Para validar o método de estimativa, diferentes benchmarks são usados (Tabela 8), e a potência medida é comparada com modelo de estimativa usando as equações 11 e 12. A estimativa de tempo de execução é baseada no CPI por classe de instrução, resultando em um erro inferior a 3%. A Tabela 8 mostra que o modelo de estimativa de energia é preciso, com um erro médio inferior do que 10%. As diferenças observadas vêm da: (*i*) atividade de chaveamento usada no modelo de energia por instrução não é o mesmo de uma aplicação real; (*ii*) em aplicações reais existem dependências de dados, afetando os tempos de execução e a energia consumida.

Tabela 8 – Estimativa de erro da energia do processador.

Benchmark	Potência Média Medida (mW)	Energia estimada (nJ)	Potência estimada (mW)	Erro (%)
Insert sort	2,236	4.185.267	2,39	6,98
Bubble sort	2,548	1.716.722	2,42	-5,14
fft	2,327	838.921	2,25	-3,36
matrix	2,214	6.734.430	2,25	1,77
Binary	2,218	1.712.422	2,34	5,71
compress	2,218	4.398.619	2,29	3,27
usqrt	2,407	2.013.805	2,40	-0,11
factorial	2,093	7.404.168	2,24	6,83
switch-case	2,236	5.626.256	2,26	0,96

5.1.4 Resultados

Quatro aplicações reais foram executadas simultaneamente em um MPSoC HeMPs 6x6 (descrição SystemC nível RTL com precisão de ciclo de relógio), uma aplicação MPEG (5 tarefas), uma DTW (6 tarefas), uma Dijkstra (6 tarefas) e uma aplicação *multispectral image analysis* (14 tarefas). O tempo de execução de um cenário é 10,5 minutos, em um processador Intel Xeon 2.93 GHz de 8 cores, com 32GB de RAM.

O projeto do MPSoC adotou duas estratégias para baixa potência. No nível do PE, o relógio do processador é desligado quando não há tarefas para executar. Quando um processador recebe uma tarefa para executar, a interface de rede detecta a requisição de mapeamento e o relógio do processador é ativado. Quando a tarefa termina sua execução, o relógio do processador é novamente desligado. Usando essa abordagem, a potência dissipada pelo processador em *idle* é somente sua corrente de fuga (*leakage*), sendo igual à 0,02 mW. No roteador da NoC, duas

frequências são usadas, 100 MHz quando flits estão sendo transmitidos, e 10 MHz quando o roteador está em modo *idle*. A frequência de chaveamento é controlada no buffer de entrada, que sinaliza quando flits estão entrando para a frequência ser elevada. Tal abordagem permite reduzir o consumo de energia em modo *idle*.

A Tabela 9 apresenta a energia gasta por cada processador do MPSoC (cada elemento da tabela corresponde a um processador). Os PEs amarelos correspondem aos processadores gerentes, nesses processadores o relógio não é desligado. Os PEs brancos contêm processadores que não receberam nenhuma tarefa para executar. Eles executam o processo de *boot* executado pelo OS, e seus relógios são desligados. Os PEs verdes contêm processadores que executaram tarefas. Com tais resultados, se torna possível estimar a energia (e a potência) de cada aplicação executada no MPSoC. As aplicações consomem, respectivamente: DTW (A) 149,5 μJ, MPEG (B) 193,16 μJ, *multispectral image analysis* (C) 25,02 μJ, Dijkstra (D) 259,25 μJ.

Tabela 9 – Energia gasta em 36 PEs, em μJ (100MHz). Marcador de A até D correspondem aos PEs executando aplicações.

X \ Y	0	1	2	3	4	5
5	1,41	1,41	1,41	150,15 B	1,41	1,41
4	35,01 A	1,41	1,41	35,03 B	7,98 B	1,41
3	171,66	79,53 A	34,96 A	171,69	1,41	1,41
2	3,89 C	9,80 C	2,88 C	82,72 D	1,41	1,41
1	3,69 C	2,96 C	1,41	93,32 D	1,41	1,41
0	163,48	1,80 C	1,41	171,69	83,22 D	1,41

A Tabela 10 mostra a energia gasta pelos roteadores da NoC (cada elemento da tabela corresponde a um roteador). Essa tabela mostra que:

i. A energia gasta por cada roteador é uma função do número de buffers: roteadores brancos possuem 3 buffers, consumindo 1,25 μJ, exceto o roteador do canto inferior esquerdo que transmite flits para outros roteadores; roteadores azuis possuem 4 buffers (1,57 até 1,67 μJ); roteadores verdes possuem 5 buffers (1,89 até 2,01 μJ).
ii. A taxa de injeção é pequena, tipicamente 5%, o que explica a pequena variação observada no consumo dos roteadores. Essa pequena taxa de injeção é devido ao comportamento dos benchmarks já que a maior parte do tempo, os PEs estão executando e não comunicando. A segunda razão que explica esse pequeno tráfego vem da ausência de memória compartilhada.
iii. Os PEs consomem até 100 vezes mais, comparado aos roteadores. Em tais experimentos, a energia consumida pela NoC corresponde à 4,6% do total da energia consumida.

A energia consumida nos links da NoC foi menor, sendo o consumo no pior caso igual a 0,0402 μJ. Apesar do pequeno consumo observado nos fios, é importante analisar os links com maiores cargas, uma vez que esses links são mais suscetíveis aos efeitos do envelhecimento, como

eletromigração.

Tabela 10 – Energia gasta em 36 roteadores da NoC, em μJ (100MHz).

X \ Y	0	1	2	3	4	5
5	1,24	1,57	1,57	1,62	1,57	1,24
4	1,61	1,90	1,89	2,01	1,94	1,57
3	1,61	1,98	1,94	1,92	1,90	1,57
2	1,60	1,91	1,91	1,95	1,90	1,57
1	1,61	1,92	1,91	1,98	1,90	1,57
0	1,38	1,67	1,64	1,63	1,60	1,24

A Tabela 11 mostra o impacto da estratégia de baixa potência (LP – Low Power) adotada. No nível de processador, o total de energia consumida teve uma redução de 65% e no nível da NoC a redução foi de 90%. Sem a técnica de LP, todos os processadores executam quase o mesmo número de instruções (menor valor de desvio padrão), enquanto com a técnica de LP somente os processadores com tarefas sendo executadas (alto valor do desvio padrão, Tabela 11). A NoC sem LP consome dez vezes mais comparada a NoC com LP, devido a relação entre a frequência de quando o roteador está ligado ou em *idle* (100 e 10 MHz, respectivamente).

Tabela 11 – Redução de energia (μJ) usando técnica de baixa potência (LP – Low Power).

	Processadores			**NoC**		
	Total	Média	Dev. Padrão	Total	Média	Dev. Padrão
Sem LP	3,777.2	128.3	18.0	604.7	16.8	2.2
Com LP	1,329.4	36.9	58.9	61.5	1.7	0.2

5.2 Modelagem Térmica

A evolução tecnológica aliada à crescente capacidade de processamento de sistemas multiprocessados pode causar maior variação térmica e de temperatura de operação no chip. O gerenciamento de variação térmica e temperatura de operação é fundamental para realizar a operação do sistema de forma confiável e eficiente [HEN13]. Temperaturas elevadas no chip podem levar a uma degradação geral do desempenho do sistema (por exemplo, eficiência energética), falhas transitórias devido a violações de temporização, bem como falhas físicas permanentes [SEM06] [HRU08].

Para equilibrar a variação de temperatura em tempo de execução, ao mesmo tempo em que satisfaçam as restrições de potência de sistemas multi-core, os pesquisadores utilizam diferentes técnicas, como DVFS, escalonamento de tarefas, mapeamento e migração. DVFS reduz os picos de alta temperatura, baixando a tensão de alimentação e a frequência de operação do sistema. Os efeitos da alocação das aplicações e sua carga de trabalho no comportamento térmico do sistema foram investigados nos trabalhos [CHA13] e [RUD14], mostrando que as decisões de desequilíbrio

de carga podem gerar zonas *hotspots* e consequentes implicações térmicas, o que pode resultar em sistemas não confiáveis. Enquanto o mapeamento de tarefas e as técnicas de migração visam equilibrar as cargas de trabalho das aplicações em vários elementos de processamento, o escalonamento de tarefas se concentra em otimizar tarefas/threads locais para atender às restrições de temperatura. As técnicas acima mencionadas assumem a existência de vários sensores de temperatura no chip para gerir a temperatura do sistema [WAN13].

Embora a coleta de valores reais de temperatura no chip possa ser considerada ideal, essa abordagem apresenta várias limitações. Os sensores físicos podem estar localizados longe das zonas de *hotspots*, o que afeta tanto a precisão da medição de temperatura quanto o tempo de resposta [KON12]. Para minimizar tais limitações, a utilização de inúmeros sensores é uma opção. Conforme relatado em [WAR10], para conseguir uma gestão térmica eficiente, o microprocessador IBM combina 44 sensores térmicos digitais em um chip. A incorporação de múltiplos sensores no mesmo *die* pode reduzir a subestimação da temperatura e o atraso na resposta, mas incorre em custos adicionais em relação à energia e à área, reduzindo seu uso em sistemas de grande escala que requerem informações detalhadas sobre distribuição de temperatura em tempo de execução.

O objetivo desta seção é propor um modelo de monitoramento de temperatura em tempo de execução baseado em software, que pode ser usado para coletar informações detalhadas da distribuição de temperatura de sistemas embarcados multi-core, sem penalizar o tempo de execução das aplicações e a operação do sistema.

As principais contribuições desta seção são as seguintes:

- um modelo de monitoramento de temperatura em tempo de execução baseado em software que possa ser usado para gerenciamento de temperatura em sistemas de grande escala;
- validação do modelo proposto, integrando-o a uma plataforma MPSoC baseada em NoC, considerando diversos cenários de referência;
- comparação do modelo proposto com uma ferramenta de temperatura de referência.

5.2.1 Estado da Arte

Além de modelos e ferramentas, como o HotSpot [WEI06], que permitem a análise térmica em tempo de projeto, os pesquisadores têm investigado técnicas para gerenciar em tempo de execução, sistema com restrição de potência e temperatura.

Por exemplo, em [CHA13] [RUD14], as tarefas são mapeadas de acordo com a condição térmica dos núcleos, que são coletadas em tempo de execução por sensores físicos. Em [GE10], os sensores também são utilizados para monitorar a temperatura do sistema e, com base nas informações recolhidas, as migrações de tarefas podem ser disparadas, com o objetivo de equilibrar a temperatura do sistema. Um esquema de migração de tarefas semelhante é proposto em [LIU15], com o objetivo de reduzir *hotspots* em sistemas multi-core. Outro trabalho que visa o equilíbrio térmico em sistemas multi-core é apresentado em [KUR08].

Wu et al. [WU11] apresentam um método de DVFS baseado em sensor de temperatura, que

permite regular a frequência do sistema de acordo com sua condição de temperatura.

Tais técnicas dependem de uma abordagem de monitoramento, que forneça uma informação de potência/temperatura e são usadas para disparar uma técnica de gerenciamento térmico quando necessário. Tais abordagens consideram a presença de sensores no chip. Apesar do aumento da potência do chip e do seu custo da área, os sensores físicos são vulneráveis a ruídos e à variação do processo. Na tentativa de substituir ou pelo menos reduzir o número de sensores térmicos no chip, uma abordagem baseada em software é descrita em [LEE05]. Esta abordagem amplia o modelo HotSpot e usa contadores de desempenho físico disponíveis no Pentium 4 para coletar sua potência em tempo de execução. A principal limitação dessa abordagem é a sobrecarga de desempenho relacionada à computação do modelo de temperatura HotSpot. Os resultados mostram que a sobrecarga de desempenho dessa abordagem é superior a 50%, dependendo do perfil do benchmark.

Diferente dos trabalhos revistos, esta Tese propõe um modelo de monitoramento térmico baseado em software, que foi integrado em um sistema multi-core baseado em NoC. O modelo proposto permite coletar informações detalhadas sobre a distribuição de temperatura de sistemas multi-core de grande escala com pouco tempo de execução (menos de 5%) e sobrecarga de potência (menos de 4%). Além disso, devido ao baixo uso de memória (menos de 2 KB), a abordagem proposta pode ser facilmente integrada em outros sistemas operacionais, o que abre novas possibilidades para melhorar o gerenciamento dinâmico de temperatura de outras plataformas multiprocessadas.

5.2.2 Fluxo do Processo de Estimação de Temperatura

Essa subseção descreve o fluxo de duas fases usado para estimar a temperatura do MPSoC, no nível de PE. A Figura 20 ilustra a fase "*tempo de projeto*" e a fase "*tempo de execução*".

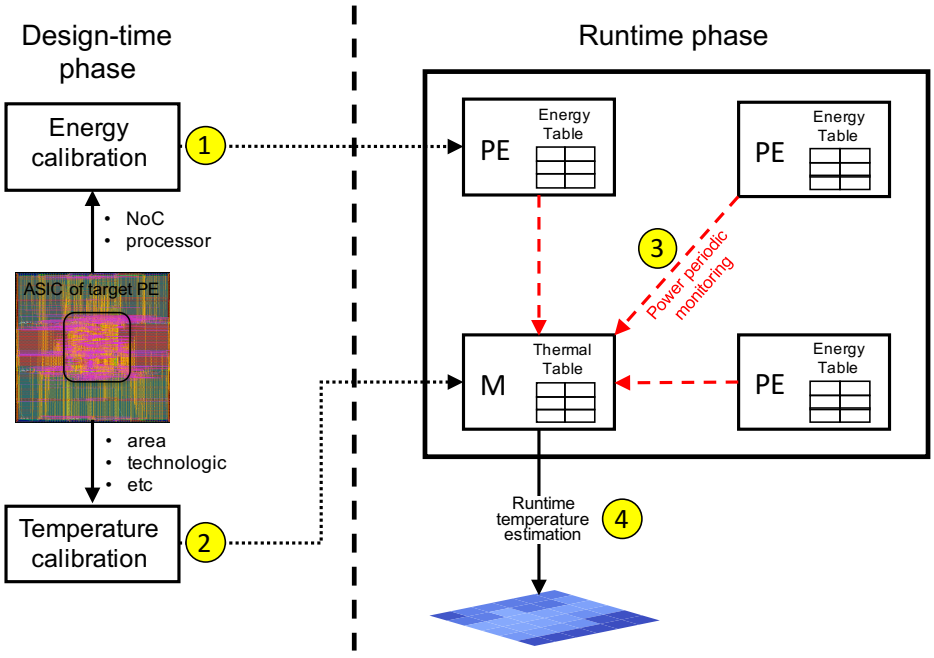

Figura 20 – Fluxo do processo de estimação de temperatura [CAS16a].

A fase “*tempo de projeto*” consiste em duas etapas:

(1) Calibração de energia (‘1’ na Figura 20). Essa etapa gera uma *tabela de energia*, a qual inclui o custo energético do processor e do roteador da NoC, para uma dada tecnologia. Essa fase foi detalhada na subseção 5.1.

(2) Calibração de temperatura (‘2’ na Figura 20). Essa etapa cria um modelo de temperatura usando a ferramenta Hotspot como referência. Como saída, uma tabela térmica é gerada com as influencias térmicas. Essa etapa é detalhada na subseção 5.2.3.

A fase “tempo de execução” também contém duas etapas:

(1) Monitoramento periódico de potência (‘3’ na Figura 20). Cada PE monitora a energia do seu processador e roteador, de acordo com uma janela de monitoramento parametrizável. Valores de potência obtidos pelo monitoramento são enviados para o processador gerente (*M* na Figura 20).

$$P_{PE} = \frac{E_{router} + E_{processor}}{monitoring\ period} \tag{13}$$

(2) Estimativa de temperatura em tempo de execução (‘4’ na Figura 20). O processador gerente recebe a potência dissipada por cada PE e calcula em tempo de execução a temperatura corrente de todos os PEs. Essa etapa é detalhada na subseção 5.2.4.

5.2.3 Calibração de temperatura

A calibração de temperatura usa como referência a ferramenta HotSpot [WEI06], que fornece um modelo térmico preciso, amplamente utilizado na comunidade de pesquisa de arquitetura de computadores. Os valores de erro de temperatura obtidos no pior caso, usando o modelo da ferramenta HotSpot é menor do que 5%, o que justifica a sua utilização como modelo de referência.

A caracterização térmica da ferramenta HotSpot é produzida de acordo com um *floorplan* que representa o circuito alvo, considerando propriedades físicas e dissipação de energia de componentes (por exemplo, processadores), modelados como um par RC, no *die* (a Figura 21 ilustra o modelo RC). No nosso caso, HotSpot é usado para capturar o fluxo de calor dentro do PE e dentro do encapsulamento térmico de cada PE ativo, bem como o fluxo de calor lateral entre os PEs.

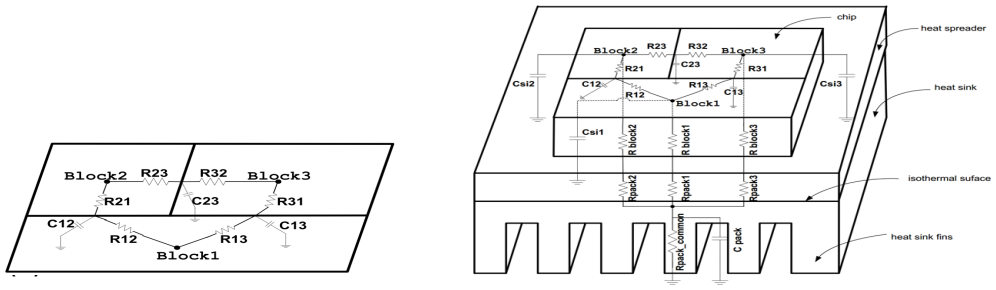

Figura 21 – (a): Modelo Lateral, com par RC entre o centro de cada bloco e o centro de cada borda compartilhada. (b): Modelo RC completo, incluindo componentes verticais e laterais.

Para iniciar a calibração de temperatura, é aplicada na ferramenta HotSpot, o máximo de potência dissipada em um dado PE, com um conjunto de PEs circundando tal PE. Existem dois objetivos para essa simulação. O primeiro, é avaliar como a dissipação de potência afeta a temperatura ao longo do tempo. O segundo objetivo, é avaliar o como a temperatura do PE sob teste afeta a temperatura dos seus vizinhos.

A Figura 22 ilustra o impacto da temperatura de um PE alvo (PE sob avaliação) sobre seus vizinhos diretos (laterais e diagonais), e nos PEs mais afastados do PE alvo (outros PEs vizinhos). Nessa figura, vemos que quando um dado processador dissipa uma certa quantidade de potência, sua temperatura sobe, afetando de formas diferentes a temperatura dos restantes dos PEs. Os seus vizinhos laterais, são os que sofrem maior influência, pois há mais área de contato entre eles. O segundo tipo de vizinho mais afetado é o vizinho diagonal, que tem uma área de contato menor com o PE alvo, o que consequentemente diminui a influência térmica. Por último, o restante dos PEs do MPSoC, que não são vizinhos diretos, sofrem uma influência térmica praticamente desprezível.

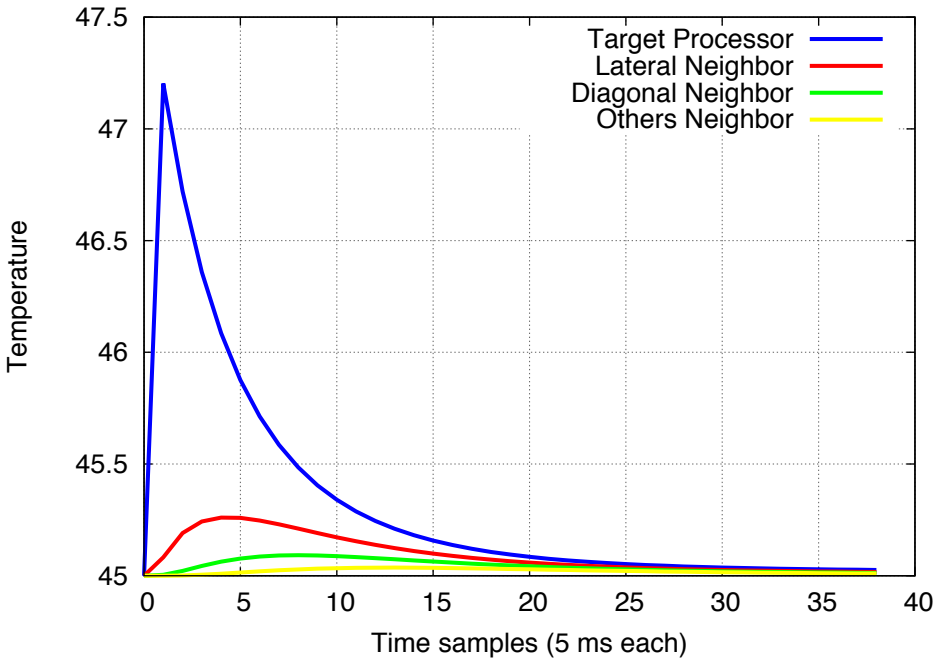

Figura 22 – Efeitos da temperatura (^{0}C) ao longo do tempo em um dado PE e nos seus vizinhos [CAS16a].

Para criar o modelo de temperatura, um conjunto de premissas são estabelecidas com o intuito de reduzir a complexidade do modelo. Tal redução é necessária para o modelo ser factível de ser executado em tempo de execução.

- Premissa 1. É considerado que a influência térmica de um PE afeta somente seus vizinhos diretos. Como visto na Figura 22, a temperatura de um dado PE tem um impacto mínimo nos seus vizinhos distantes.
- Premissa 2. O efeito do decaimento da temperatura (inércia térmica) que é modelado pelo HotSpot é muito longo. Esse trabalho assume que, quando um PE dissipa uma dada quantidade de potência, o efeito dessa dissipação de potência sobre a

temperatura, ocorre somente durante um terminado tempo (TAMANHO_INERCIA). No final desse período, o efeito térmico da potência dissipada termina. O período assumido por este trabalho é 100 ms, correspondendo à 20 janelas de amostragens de 5 ms. A parte superior da Figura 23 mostra o gráfico do decaimento térmico modelado pelo HotSpot e a parte inferior da mesma figura, mostra o gráfico desse mesmo decaimento, mas com uma duração menor do seu efeito.

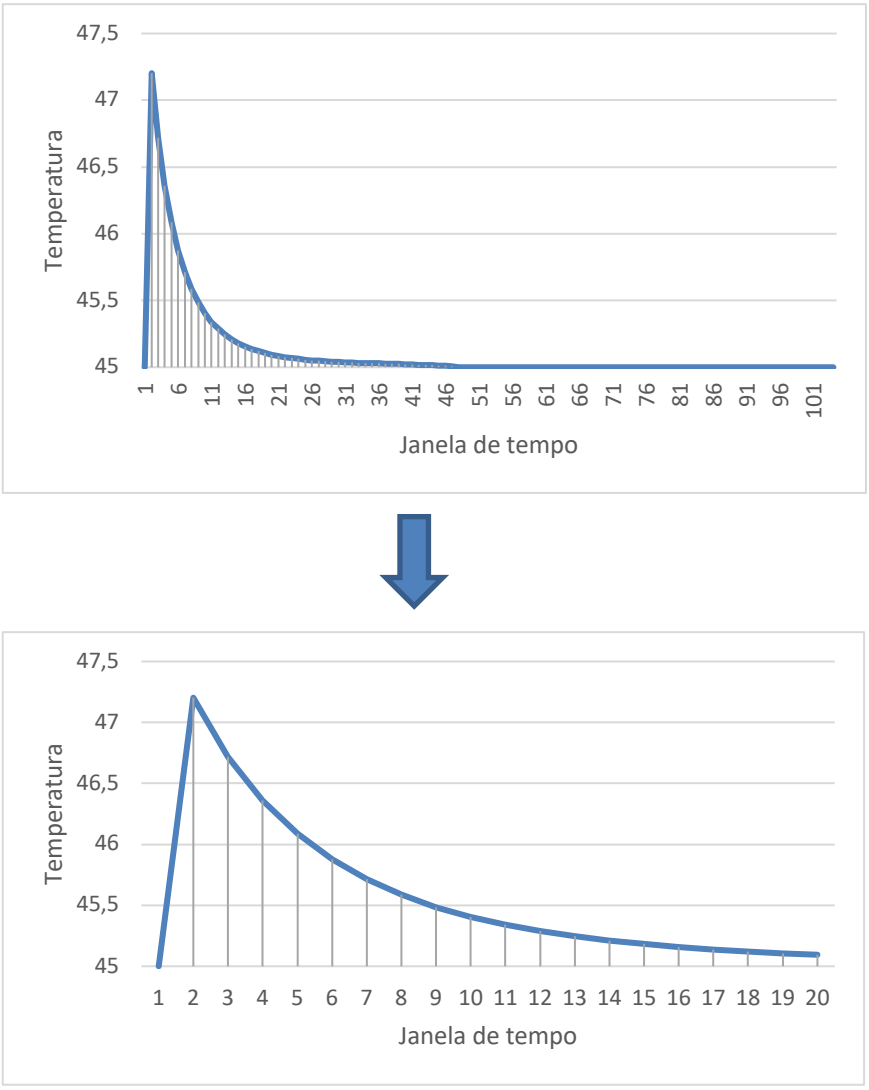

Figura 23 – Exemplo de decaimento térmico do modelado pelo HotSpot (parte superior) e o decaimento térmico usado nesse trabalho (parte inferior).

- Premissa 3. Como a temperatura é linear em função da potência, é possível discretizar os valores de potência em intervalos, atribuindo uma temperatura para cada intervalo de potência. A Figura 24 apresenta um exemplo de um comportamento térmico de um processador ao longo do tempo para sua potência máxima, e outros dois valores de potência, 75% e 50%.

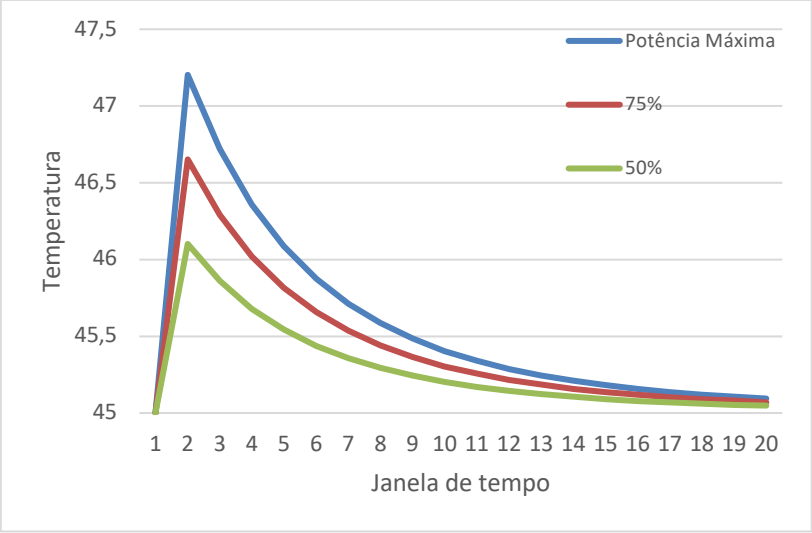

Figura 24 – Exemplo do comportamento térmico com diferentes valores de potência dissipada no PE.

- Premissa 4. Somente valores inteiros são usados ao invés de números em ponto flutuante. Essa premissa permite maior rapidez na execução do modelo.

Essas premissas podem induzir a um erro na estimativa de temperatura. Esse erro do modelo é avaliado na subseção 5.2.5.1.

Usando as premissas acima, é criado um modelo baseado em tabelas (LUTs ou *Lookup Tables*). Esse modelo contém 3 *matrizes térmicas*. Cada matriz térmica corresponde ao efeito da potência ao longo do tempo para os PEs na fronteira do sistema (*lateral*), nas diagonais do sistema (*diagonal*) ou no centro do sistema (*central*). A Figura 25 mostra o posicionamento desses PEs em um sistema com tamanho 6x6.

diagonal	lateral	lateral	lateral	lateral	diagonal
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
lateral	central	central	central	central	lateral
diagonal	lateral	lateral	lateral	lateral	diagonal

Figura 25 – Posição relativa dos PEs para o modelo baseado em LUT [CAS16a].

A Figura 26 apresenta uma abstração de uma estrutura de dados usada para uma matriz térmica. Cada matriz possui 3 dimensões:

- coordenada *x*: índice de potência, com a potência discretizada em 10 faixas de acordo com a potência máxima dissipada em uma janela de amostragem;
- coordenada *y*: inércia térmica (20 valores);
- coordenada *z*: efeito da potência na temperatura de um PE sob avaliação e em seus vizinhos (3 valores). Para o índice *z*=0, o efeito é do próprio PE sob avaliação, *z*=1 o efeito é nos PEs laterais e *z*=2 é nos PEs diagonais.

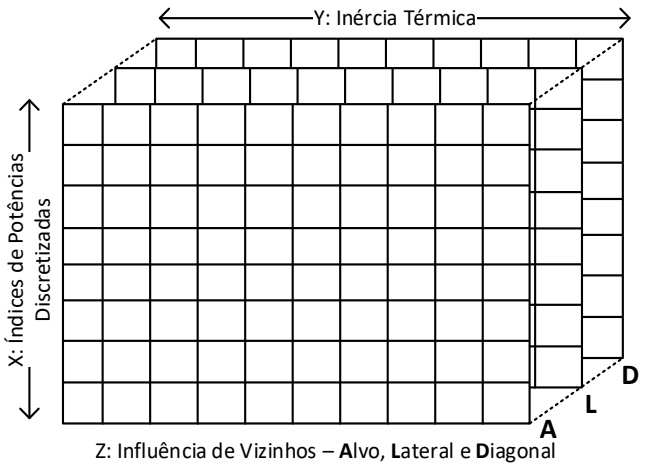

Figura 26 – Exemplo de uma estrutura de dados de uma matriz térmica [CAS16a].

Cada matriz térmica requer 600 inteiros (10 * 20 * 3), o que representa uma baixa quantidade memória necessária, menos do que 2 KB, para gravar o conjunto de tabelas para estimar a temperatura.

5.2.4 Heurística de estimação de temperatura

Para computar em tempo de execução a temperatura corrente de todos os PEs do MPSoC, a heurística de estimação de temperatura deve ser o mais simples possível, mas sem incorrer em imprecisão. A base para a computação da temperatura são as 3 *matrizes térmicas*, obtidas na fase de tempo de projeto.

A heurística proposta adota 3 estruturas de dados para computar a temperatura de cada elemento de processamento:

- temperatura[|PE|]: vetor que contém a temperatura de cada PE, onde cada elemento é inicializado com a temperatura ambiente (45ºC).
- potência[|PE|]: vetor que contém a potência média de cada PE, obtida no final de cada janela de monitoramento (5 ms na atual implementação). Se um dado PE não envia o pacote de monitoramento, é assumido que a potência média no período é zero (por exemplo, o processador pode estar em estado *hold*).
- influência térmica [TAMANHO_INERCIA][|PE|]: matriz usada para armazenar o efeito da temperatura ao longo do tempo. Essa matriz funciona como uma FIFO circular.

A fim de exemplificar e ajudar na compreensão do uso da matriz *influência térmica*, vamos assumir que: TAMANHO_INERCIA=4, |PE|=1. Também vamos assumir que, para 1W de potência dissipada em um processador, o efeito que essa potência irá causar na temperatura é de 10ºC, -5ºC, -3ºC e -2ºC respectivamente ao longo do tempo, e para 0,5W de potência dissipada o efeito térmico é de 6ºC, -3ºC, -2ºC e -1ºC. A Tabela 12 apresenta um exemplo hipotético. No topo da tabela apresenta-se a temperatura atual (T_{atual}), e a potência média obtida através do monitoramento.

Para cada janela de amostragem a matriz *influência_térmica* ilustra a influência atual da temperatura (coluna *corr.*), o efeito que a potência aplicada irá causar (coluna *aplic.*), e o efeito resultante, obtido pela soma das colunas *corr.* e *aplic.* (coluna *res.*), Observar:

- linhas cinzas indicam a atual posição de acesso a matriz *influência_térmica*;
- a amostragem seguinte à amostragem corrente tem o valor anterior à posição atual de acesso zerado, pois terminou o efeito da inércia térmica (valores zero em vermelho);

A última linha da Tabela é a nova temperatura, considerando o efeito da potência sobre a temperatura, bem como o efeito dos valores de potência previamente amostrados, conforme a equação abaixo.

$$Nova\ T = T_{atual} + Efeito_{res.} \tag{14}$$

Tabela 12 – Exemplo de como estimar o efeito da potência média na temperatura, usando a matriz *influência_térmica*.

T_{atual}	45ºC				55ºC				60ºC				58ºC				56ºC		
Potência	1W				1W				0,5W				0,5W				1W		
Matriz influência térmica	*corr.*	*aplic.*	*res.*		*corr.*	*aplic.*	*res.*		*corr.*	*aplic.*	*res.*		*corr.*	*aplic.*	*res.*		*corr.*	*aplic.*	*res.*
	0	**10**	**10**		0	-2	-2		-2	-2	-4		-4	-3	**-7**		**-7**	**10**	**3**
	0	-5	-5		**-5**	**10**	**5**		0	-1	-1		-1	-2	-3		-3	-5	-8
	0	-3	-3		-3	-5	**-8**		**-8**	**6**	**-2**		0	-1	-1		-1	-3	-4
	0	-2	-2		-2	-3	-5		-5	-3	**-8**		**-8**	**6**	**-2**		0	-2	-2
Nova T	55ºC				60ºC				58ºC				56ºC				59ºC		

A primeira potência amostrada adiciona 10ºC na temperatura do PE. A segunda amostra de potência adiciona 5ºC na temperatura do PE (-5ºC devido a influência térmica prévia mais 10ºC da influência atual (potência atual)). A terceira amostra de potência, 0,5W, acrescentaria mais 6ºC a temperatura do PE, mas devido a influência térmica das amostras de potências anteriores, a temperatura reduz 2ºC ($(-3ºC)_{amostra\_1}$ + $(-5ºC)_{amostra\_2}$ + $(6ºC)_{amostra\_3}$). A Tabela 12 também demonstra o comportamento circular que a matriz *influência_térmica* possui.

Observar que a temperatura final é de 59 ºC, e se não houverem mais amostras de temperatura, o valor de temperatura do PE retorna para 45ºC (somando os valores da última coluna: -8, -4 e -2).

A Figura 28 apresenta a heurística proposta para atualizar a temperatura atual de todos os PEs. Além dos vetores de *temperatura* e de *potência*, o procedimento recebe o valor de *idx*, que corresponde à posição de acesso da matriz *influência_térmica* (células cinzas do exemplo apresentado na Figura 9). Essa heurística utiliza 3 funções:

- **get_tipo_matriz**(PE) (nas linhas 3 e 7) – de acordo com a posição no sistema, o PE sob avaliação pode ser diagonal (PEs 1, 3, 7, 9 na Figura 27), lateral (PEs 2, 4, 6, 8) ou central

(PE 5). Esta função retorna um ponteiro para uma das matrizes térmicas.

PE 7	PE 8	PE 9
PE 4	PE 5	PE 6
PE 1	PE 2	PE 3

Figura 27 – Índice de PE para ilustrar o conceito de PE diagonal, lateral e central.

- **vizinhos**(PE) (linha 5) – essa função retorna uma lista com os vizinhos diretos de um dado PE. Por exemplo (Figura 27), se o PE sob avaliação é o 1, a função irá retornar o conjunto {2,4,5}, se o PE=5 a função irá retornar o conjunto {1,2,3,4,6,7,8,9}
- **get_viz_influência_sobre_pe**(PE) (linha 8) – retorna como um dado PE (viz) influencia o PE sob avaliação. No exemplo anterior, se o PE=1 e viz=2, a função retornará *lateral*. Se viz=5, a função retornará *diagonal*.

```
void atualiza_temperatura(idx, temperatura[], potência[])
    // idx: posição atual na matriz influência_térmica
    // temperatura[|PE|]: temperatura atual de todos os PEs, atualizada por esse procedimento
    // potência[|PE|]: potência média monitorada, durante a janela de monitoramento, de todos os PEs
1.  FOR EACH PE pe no sistema
2.      FOR inercia = 1 até TAMANHO_INERCIA
3.          *m = get_tipo_matriz(pe)                    // TIPO DE MATRIZ (LATERAL/DIAGONAL/CENTRAL)
4.          influência_térmica(idx,pe) += m [potência(pe)] [inercia] [0]
5.          lista_vizinhos ← vizinhos(pe)
6.          FOR EACH PE viz na lista_vizinhos
7.              *m = get_tipo_matriz(viz)
8.              pos = get_viz_influência_sobre_pe(pe,viz)    // LATERAL OU DIAGONAL
4.              influência_térmica(idx,pe) += m [potência(viz)] [inercia] [pos]
10.         END FOR
11.         idx ← (idx +1) mod TAMANHO_INERCIA
12.     END FOR
13.     temperatura(pe) += influência_térmica(idx,pe)
14.     influência_térmica(idx,pe) ← 0
15. END FOR
16. idx ← (idx +1) mod TAMANHO_INERCIA
```

Figura 28 – Heurística para atualizar a temperatura atual de todos os PEs.

O laço externo (linha 1 até 15 na Figura 28) atualiza a temperatura de todos os PEs. O laço entre as linhas 2 e 12, atualiza a matriz *influência_térmica*, como no exemplo da Tabela 12, agora considerando a influência dos vizinhos diretos. As linhas 3 e 4 atualizam a matriz *influência_térmica* considerando somente o PE sob avaliação. Seguindo, entre as linhas 5 e 10, a matriz *influência_térmica* é atualizada considerando os vizinhos diretos.

A linha 11 incrementa o índice atual da matriz *influência_térmica*. Note que o laço 2-12 tem, na verdade, dois índices: inercia, que corresponde ao efeito da temperatura ao longo do tempo; idx, usado para preencher a matriz *influência_térmica*. Quando o laço 2-12 termina, *idx* retorna para

o valor original. Na linha 13, a temperatura do PE sob avaliação é atualizada. Na linha 14 a influência térmica na atual matriz *influência_térmica* é zerada, como mostra o exemplo da Tabela 12. Quando todos os PEs têm duas temperaturas atualizadas, o índice *idx* é incrementado (linha 16).

Essa heurística tem complexidade computacional O(*n*), onde **n** é o número de PEs. Nos experimentos realizados para um MPSoC executando a 100 MHz e com tamanho de 3x3, o pior caso para o tempo de execução da heurística foi de 325 µs.

5.2.5 Resultados

Os experimentos para avaliar o modelo térmico, foram realizados na plataforma de referência desse trabalho.

Cinco *benchmarks*, descritos na linguagem C, foram usados: *(i)* DTW – *Digital Time Warping*, com 10 tarefas; *(ii)* decodificador MPEG, com 5 tarefas; *(iii)* DJK – Dijkstra, com 6 tarefas; *(iv)* SYN1, aplicação sintética, com 12 tarefas, que emula o comportamento da comunicação de um decodificador MPEG4 completo; *(v)* SYN2, aplicação sintética, com 12 tarefas, que emula o comportamento da comunicação de um decodificador VOP (*Video Object Plane*).

Os experimentos foram conduzidos usando os cenários apresentados na Tabela 13, com três diferentes tamanhos de MPSoC: 3x3, 4x4 e 6x6. Em todos os tamanhos de MPSoC, um PE é reservado com o propósito de gerenciar o sistema (ex. executar a heurística proposta). O tamanho limite do sistema nos experimentos foi de 36 PEs, uma vez que sistemas com grande número de PEs adotam uma organização baseada em cluster, onde o MPSoC é particionado em clusters, com um gerente por cluster, garantindo sua escalabilidade [CAS13].

Tabela 13 – Cenários usados para avaliar a heurística que estima a temperatura em tempo de execução

Cenário	Aplicações	Número de tarefas
1	20 x MPEG, 20 x DJK, 20 x SYN1, 20 x SYN2, 20 x DTW	780
2	10 x MPEG, 10 x DJK, 10 x SYN1, 10 x SYN2, 10 x DTW	390
3	50 x MPEG	250
4	100 x DTW	1000
5	100 x MPEG	500

5.2.5.1 Precisão do modelo proposto

Essa subseção compara o modelo de temperatura proposto com o modelo usado como referência, o HotSpot. A Tabela 14 mostra o erro absoluto do modelo proposto, considerando a ferramenta HotSpot como referência.

Duas heurísticas de mapeamento foram usadas na avaliação: “Padrão”, onde a função custo do mapeamento é minimizar a energia de comunicação na NoC; “Energy-Aware”[CAS16a], cuja a função custo é a distribuição da carga de trabalho ao longo do tempo.

Tabela 14 – Erro do modelo proposto em comparação ao modelo HotSpot.

Cenário/ Tamanho do MPSoC		Mapeamento "Padrão"			Mapeamento "Energy-Aware"		
		MÉDIA	DESVIO PADRÃO	MAX	MÉDIA	DESVIO PADRÃO	MAX
3x3	1	2,33%	2,19%	8,88%	1,34%	1,33%	5,71%
	2	1,98%	2,12%	8,88%	1,14%	1,23%	5,71%
	3	1,46%	1,61%	8,00%	1,08%	1,07%	5,44%
	4	1,21%	0,81%	5,14%	1,13%	1,06%	5,50%
	5	2,42%	2,44%	9,74%	1,44%	1,57%	6,69%
4x4	1	**3,52%**	**3,31%**	11,99%	1,87%	1,84%	**6,94%**
	2	2,53%	2,53%	10,86%	1,38%	1,41%	6,18%
	3	1,92%	1,66%	7,70%	0,87%	0,80%	3,66%
	4	1,79%	1,88%	9,02%	1,14%	0,98%	4,05%
	5	3,13%	3,03%	12,14%	1,43%	1,41%	5,75%
6x6	1	3,18%	3,16%	14,20%	1,91%	1,64%	5,89%
	2	2,17%	2,22%	12,05%	1,05%	0,99%	4,43%
	3	0,95%	0,88%	5,00%	0,52%	0,45%	2,49%
	4	0,57%	0,50%	2,06%	0,68%	0,53%	2,28%
	5	1,66%	1,78%	10,07%	0,94%	0,88%	3,87%

Analisando a Tabela 14, podemos observar que o erro absoluto médio (representado pelas colunas MÉDIA) é menor do que 3,6%, e o desvio padrão (colunas DESVIO PADRÃO) é menor do que 3,4%. A heurística de mapeamento "padrão" cria zonas de hotspot, induzindo ao aumento de temperatura em alguns PEs, o que evidencia os efeitos das premissas usadas para criar o modelo. Uma heurística de mapeamento que distribui a carga de trabalho uniformemente impede a criação de hotspots, resultando em um menor erro médio absoluto e desvio padrão, em comparação com o mapeamento "Padrão". Na maioria dos casos de teste, o erro absoluto máximo (colunas MAX) constatado foi menor do que 10%. O maior erro foi observado com o mapeamento "padrão" devido às zonas de hotspot. O erro máximo observado, usando o mapeamento "energy-aware", foi de 6,94%.

O modelo proposto de temperatura baseado em software (executado em 0,35 ms na frequência de 100MHz) permite estimar com precisão a temperatura de cada PE em tempo de execução. A estimativa de temperatura com um erro menor que 10% em relação à ferramenta HostSpot, permite gerenciar com segurança a temperatura do sistema em tempo de execução. A temperatura está disponível para cada PE (dados de grão fino), diferentemente das abordagens com sensores térmicos, que fornecem dados de temperatura para todo o sistema ou áreas maiores.

5.2.5.2 *Mapas Térmicos*

A Figura 29 mostra a distribuição térmica para o cenário 4 em uma MPSoC de 6x6. Cada quadrado representa um processador no MPSoC (eixo x e y), e o eixo temporal representa diferentes momentos da simulação (5%, 25%, 50%, 75% e 100% do tempo de execução). Conforme ilustrado na Figura 29(a), o modelo proposto produz uma distribuição térmica semelhante, em relação ao modelo Hotspot (Figura 29(b)) durante todas as fatias de tempo. Resultados semelhantes são observados para todos os outros cenários testados, o que corrobora com a validação do modelo proposto.

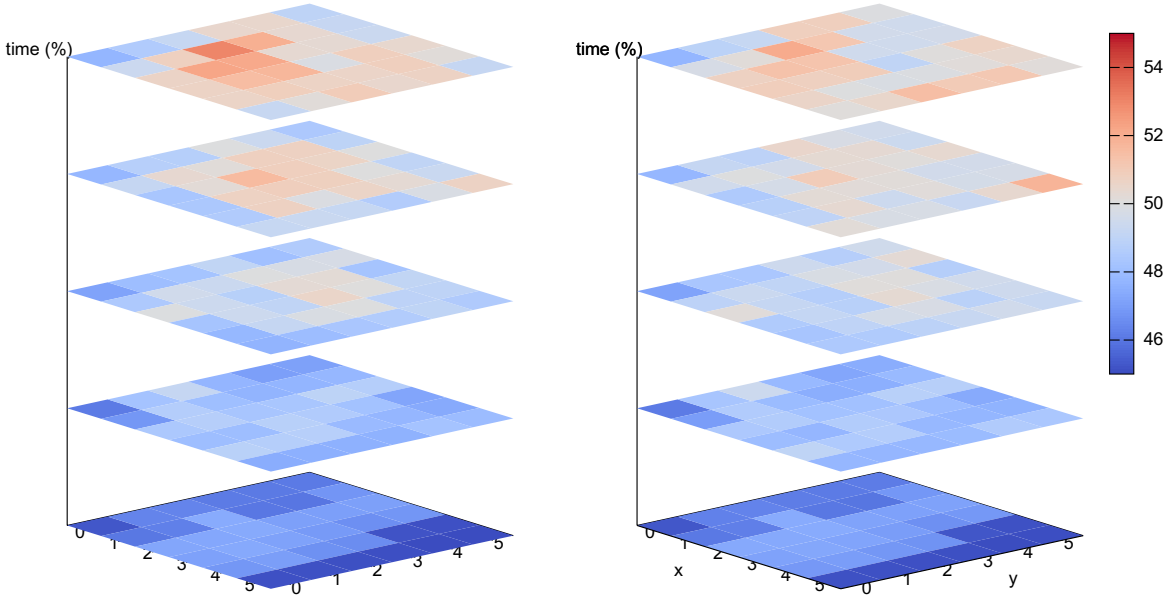

Figura 29 – Distribuição da temperatura em um MPSoC 6x6, considerando o modelo proposto (a) e o modelo HotSpot (b) [CAS16a].

A Figura 30 compara mapas térmicos, considerando as duas heurísticas de mapeamento. O objetivo é apresentar a ocorrência de hotspots (Figura 30(b)). Com a presença de hotspots, a influência dos processadores quentes aumenta, aumentando o erro do modelo, conforme apresentado na Tabela 14. Mesmo com hotspots, o erro médio é pequeno (<3,6%). Somente os processadores nas zonas de hotspot tem a estimativa de erro aumentada (pior caso: 14,2%). Esta comparação térmica ajuda no desenvolvimento de heurísticas, como mapeamento ou outras heurísticas de gerenciamento de energia (por exemplo, DVFS).

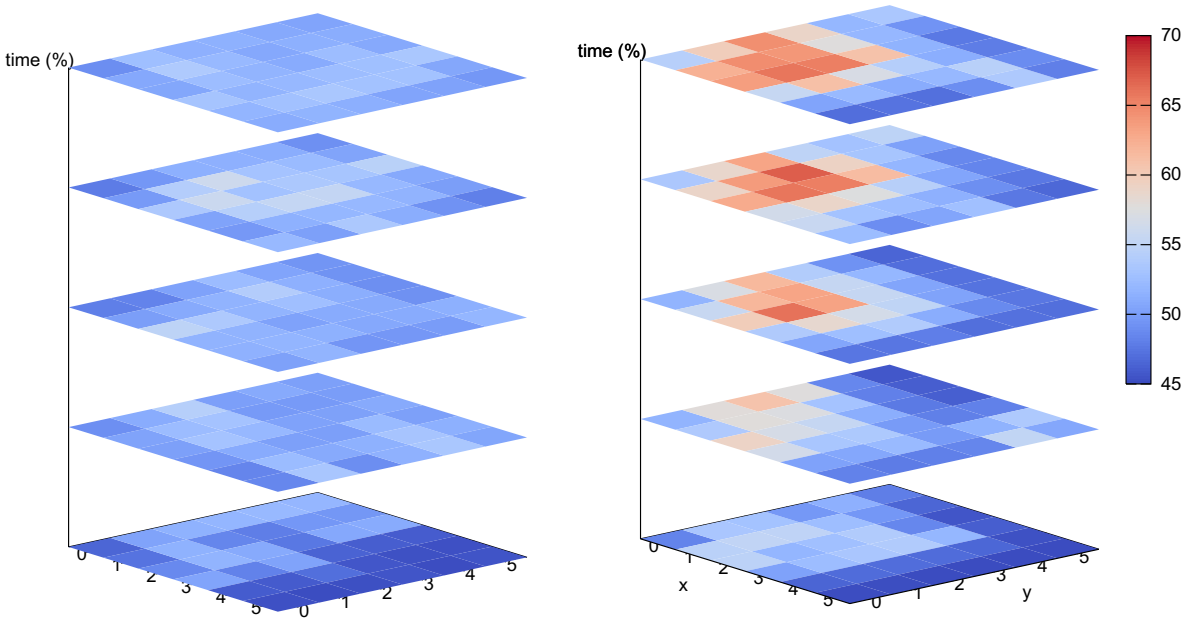

Figura 30 – (a) Mapeamento “Energy-Aware”, (b) Mapeamento “Padrão” [CAS16a].

5.2.5.3 Estimativa de custo computacional

Esta subseção avalia dois overheads induzidos pela inclusão de monitoramento e estimativa de temperatura no sistema. O gráfico apresentado na Figura 31 avalia o tempo de execução e o overhead de energia. Todos os cenários foram executados com e sem o método proposto. O overhead de tempo total de execução do modelo proposto é inferior a 4,9% no pior dos casos. À medida que o tempo de execução aumenta, a energia consumida pelo MPSoC também aumenta. Os resultados mostram que a sobrecarga de energia do modelo varia de 0,87% a 3,64% dependendo do cenário. Esses resultados demonstram o baixo custo do modelo proposto.

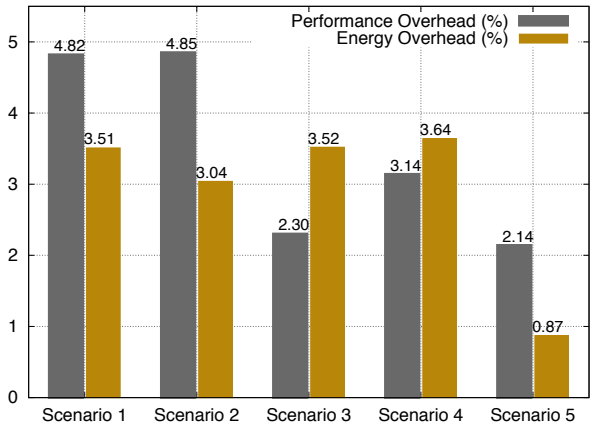

Figura 31 – Overhead do modelo proposto (MPSoC 6x6) [CAS16a].

6 ESTUDO DE DIFERENTES ESTRATÉGIAS DE MAPEAMENTO

Este Capítulo apresente três diferentes estratégias de mapeamento baseadas em um mesmo método proposto, o método de mapeamento hierárquico. O que diferencia essas estratégias é a função custo utilizada na heurística de mapeamento. Além disso, este Capítulo apresenta o estado-da-arte em mapeamento de tarefas.

6.1 Estado da Arte

A literatura sobre mapeamento de tarefas é ampla, exigindo uma taxonomia considerando diferentes critérios de mapeamento. Autores em [MAN11] e [OST13] classificam o processo de mapeamento de acordo com quatro critérios:

i. *Arquitetura alvo*: Mapeamento de tarefas pode ser executado em sistemas homogêneos (elementos de processamento idênticos) ou heterogêneos (ex. DSPs, IPs dedicados, aceleradores).

ii. *Número de tarefas por PE*: monotarefa ou multitarefa. Monotarefa assume que somente uma tarefa será atribuída por PE, enquanto a multitarefa permite mapear mais de uma tarefa por PE de acordo com critérios específicos. Uma abordagem multitarefa pode explorar melhor os recursos do sistema, permitindo a execução de um maior número de aplicações em paralelo.

iii. *O momento em que é executado:* em tempo de projeto ou em tempo de execução (*mapeamento dinâmico*). Abordagens que são executadas em tempo de projeto não são adequadas em cenários com carga de trabalho dinâmica imposta pela execução de diferentes aplicações ao longo do tempo de vida do sistema.

iv. *Gerência de mapeamento:* centralizada ou hierárquica. Mapeamento centralizado usa um único PE como responsável por todo o gerenciamento, o que é adequado para sistema pequenos devido a questão de escalabilidade. Na abordagem hierárquica, a gerenciamento do mapeamento é distribuído entre diferentes PEs, aumentando a confiabilidade e a escalabilidade do sistema.

Esse trabalho concentra-se em sistemas de propósito geral, capazes de executar várias aplicações que não são conhecidas previamente. Esse trabalho também pressupõe que as aplicações subjacentes podem ser inseridas no sistema de forma não determinística, de acordo com os requisitos do usuário. A literatura contém várias abordagens de mapeamento em tempo de execução. A Tabela 15 sintetiza os trabalhos de acordo com a taxonomia de mapeamento, para propostas voltadas para mapeamento dinâmico.

Apenas alguns trabalhos relacionados ao mapeamento multitarefa foram encontrados na literatura, propostos por Singh et al. [SIN09][SIN10][SIN13] e Mandelli et al. [MAN15]. Técnicas de mapeamento multitarefa incluem o agrupamento de tarefas, que fazem com que as tarefas sejam executadas em um mesmo PE, e escalonadas por um μ*kernel*. Um método de agrupamento de tarefas não otimizado pode levar a *hotspots*, reduzindo o tempo de vida do sistema e acelerando o desgaste do mesmo devido ao aumento de temperatura. Sistemas heterogêneos podem ter melhor

desempenho para uma especifica aplicação, e sistemas homogêneos são plataformas de propósito geral. De forma similar a exemplos industriais [INT16][TIL16], o foco do presente trabalho é em arquiteturas homogêneas.

Outra importante característica é a abordagem usando gerenciamento hierárquico do sistema [KOB11][CUI12][MAN15]. Tal abordagem é escalável e pode reduzir o esforço computacional do algoritmo de mapeamento, aumento o desempenho do sistema.

Tabela 15 – Estado da Arte em heurísticas de mapeamento dinâmico.

Autor	**Multi/ Monotarefa**	**Modelo de arquitetura**	**Gerência**	**Objetivo**
Smit et al. [SMI05]	Monotarefa	Homogêneo	Centralizada	Consumo de energia e requisitos de QoS
Ngouanga et al. [NGO06]	Monotarefa	Homogêneo	Centralizada	Volume de comunicação e carga de computação
Coskun et al. [COS09]	Monotarefa	Homogêneo	Centralizada	Confiabilidade do Sistema
Chou et al. [CHO10]	Monotarefa	Homogêneo	Centralizada	Consumo de Energia, contenção interna e externa da rede
Hölzenspies et al. [HOL08]	Monotarefa	Heterogêneo	Centralizada	Consumo de energia e requisitos de QoS
Al Faruque et al. [ALF08]	Monotarefa	Heterogêneo	**Hierárquica**	Tempo de execução, tempo de mapeamento e monitoramento de trafego
Wildermann et al. [WIL09]	Monotarefa	Homogêneo	Centralizada	Latência da comunicação e consumo de energia
Schranzhofer et al. [SCH10]	Monotarefa	Homogêneo	Centralizada	Consumo de energia
Lu et al. [LU10]	Monotarefa	Homogêneo	Centralizada	Latência da comunicação e consumo de energia
Carvalho et al. [CAR10]	Monotarefa	Heterogêneo	Centralizada	Contenção de rede e volume de comunicação
Singh et al. [SIN09] [SIN10] [SIN13]	**Multitarefa**	Heterogêneo	Centralizada	Contenção de rede e volume de comunicação
Kobe et al. [KOB11]	Monotarefa	Homogêneo	**Hierárquica**	Tempo de execução e trafego de comunicação
Cui et al. [CUI12]	Monotarefa	Homogêneo	**Hierárquica**	Tráfego de comunicação e consumo de energia
Hartman et al. [HAR12]	Monotarefa	Homogêneo e Heterogêneo	Centralizada	Confiabilidade do Sistema
Chantem et al. [CHA13]	Monotarefa	Homogêneo	Centralizada	Confiabilidade do Sistema
Bolchini et al. [BOL13]	Monotarefa	Homogêneo	Centralizada	Consumo de energia e tempo de vida do sistema
Das et al. [DAS14]	Monotarefa	Homogêneo	Centralizada	Requisitos das aplicações e tempo de vida do sistema
Mandelli et al. [MAN15]	**Multitarefa**	Homogêneo	**Hierárquica**	Redução da energia de comunicação
Trabalho Proposto	**Multitarefa**	**Homogêneo**	**Hierárquica**	**Carga de trabalho, volume de comunicação, consumo de energia, redução da temperatura e tempo de vida do sistema**

A literatura apresenta diferentes abordagens de mapeamento de tarefas em tempo de execução visando aumento da confiabilidade. Todos os trabalhos revisados com o enfoque em confiabilidade usam uma abordagem de gerenciamento centralizado do sistema [CHA13][COS09] [HAR12][BOL13][DAS14]. Entre estes, alguns trabalhos [BOL13][DAS14] produzem decisões de mapeamento em tempo de projeto que são armazenadas em um banco de dados e usadas em tempo de execução. Esta abordagem pode reduzir o desempenho do sistema, devido à sua incapacidade de lidar com variações imprevisíveis da carga de trabalho. As abordagens de mapeamento de tarefas propostas por [CHA13][HAR12], empregam sensores físicos para capturar as condições térmicas ou de desgaste dos PEs em tempo de execução. A inclusão de sensores fornece informações precisas para a decisão de mapeamento com um custo de área e de consumo de energia adicional.

A literatura apresenta abordagens hierárquicas para aumentar a confiabilidade do sistema. Entretanto, tais abordagens usam outras técnicas ao invés de mapeamento de tarefas [GE10][WU11][LIU15]. Ge et al. [GE10] propõem um método de migração de tarefas para balanceamento térmico. Esse método usa sensores térmicos, que aumentam o custo de hardware. Wu et al. [WU11] apresentam um método de *frequency scaling* dinâmico para gerenciamento térmico, o que pode impor custos de hardware adicionais. Liu et al. [LIU15] também apresentam uma abordagem de migração de tarefas para gerenciamento térmico, mas não consideram os custos de desempenho.

Mandelli et al. [MAN15] propõem a heurística LEC-DN (*Lower Energy Consumption based on Dependencies-Neighborhood*), uma abordagem de mapeamento hierárquico cuja função principal é reduzir a energia de comunicação. Para minimizar a energia de comunicação, a heurística LEC-DN visa reduzir a distância em *hops* entre as tarefas comunicantes. Quando uma determinada tarefa t_i é requerida para ser mapeada, esta heurística primeiramente analisa o conjunto de tarefas comunicantes com o t_i já mapeado. Então, a heurística aproxima t_i das tarefas comunicantes com um maior volume de comunicação.

Esta Tese propõe uma abordagem de mapeamento de tarefas que difere da literatura, uma vez que inclui as seguintes características:

- *Dinâmico*: A abordagem proposta pode melhor gerenciar as variações de carga de trabalho ao longo do tempo.
- *Hierárquico*: A abordagem proposta é implementada em diversos PEs gerenciados de forma hierárquica. Esse gerenciamento de sistema hierárquico melhora a escalabilidade do sistema, dividindo o sistema em regiões, cada um com um gerente responsável por ações dentro dele. Além disso, reduz o esforço computacional de decisão de mapeamento, não comprometendo o desempenho do sistema.
- *Melhor confiabilidade do sistema*: A abordagem proposta visa melhorar o balanceamento energético, que está diretamente relacionado a uma melhor confiabilidade do sistema [CHA13] [WAN14].
- *Monitoramento de energia hierárquica*: A abordagem proposta não emprega sensores físicos na decisão de mapeamento. Os dados de energia são obtidos em tempo de execução usando uma

abordagem de monitoramento hierárquico (descritos no Capítulo 4).

- *Modelo com precisão de ciclo de relógio usado para a validação*: A abordagem de mapeamento proposto é validada em um MPSoC com uma grande quantidade de elementos de processamento (até 256 elementos de processamento), modelado em RTL SystemC com precisão de ciclo de relógio.

6.2 Estratégia de mapeamento hierárquico

O mapeamento de um conjunto de tarefas T = $\{t_1, t_2, ..., t_n\}$ de GApp para o conjunto de SP = $\{sp_1, sp_2, ..., sp_k\}$ de GMPSoC, é definido pela função de mapeamento T → SP, onde $\forall t_i \in T, \exists\ sp_j \in$ SP. O mapeamento hierárquico de tarefas é dividido em três etapas principais:

(1) ***seleção de cluster***, define o cluster para mapear uma aplicação requerida;

(2) ***mapeamento das tarefas iniciais,*** seleciona SPs para mapear as tarefas iniciais da aplicação dentro do cluster;

(3) ***mapeamento das tarefas não-iniciais,*** seleciona SPs para mapear as tarefas não- iniciais.

A Figura 32 apresenta o protocolo de seleção de cluster e mapeamento das tarefas iniciais. O GMP recebe do mundo externo requisições para executar uma nova aplicação no sistema (‘1 – Nova aplicação’, Figura 32). O GMP verifica se o sistema tem recursos disponíveis para mapear a aplicação. Se ele não tiver recursos disponíveis, a aplicação é escalonada para ser mapeada depois. Caso contrário, o GMP seleciona o cluster para mapear a aplicação requisitada (‘2 – Seleção de Cluster’, Figura 32). A heurística usada para selecionar o cluster pode variar em função da política de mapeamento utilizada.

Uma vez que o cluster foi selecionado, o GMP obtém o *descritor da aplicação* do repositório de aplicação (seção 2.2.3), transmitindo essas informações para o LMP do cluster selecionado (‘3 – App. Desc.’, Figura 32). O LMP do cluster selecionado recebe e grava o descritor da aplicação. Após, o LMP mapeia as tarefas inicias dentro do cluster (‘’4 – Mapeamento das Tarefas Inicias’, Figura 32). O mapeamento das tarefas iniciais inicia a execução da aplicação. Bem como na seleção dos cluster, a heurística de mapeamento de tarefas iniciais pode variar em função da política de mapeamento utilizada.

Depois de selecionar um SP para receber uma tarefa inicial, o LMP envia uma mensagem para o GMP com o serviço “*requisição de alocação de tarefas*” (‘5 – Nova Tarefa’, Figura 32). Tal mensagem requisita a alocação de um código objeto de uma tarefa inicial no SP selecionado. Isso acontece porque o GMP é o único PE com acesso ao repositório de aplicações. Então, o GMP obtém o código objeto da tarefa do repositório de aplicações e transmite esse código para o SP selecionado (‘6 – Alocação de Tarefa’, Figura 32). O SP irá escalonar a nova tarefa no final da recepção do pacote de “alocação de tarefa”. Além disso, o LMP mantém uma estrutura de dados, chamado *tabela de tarefas*, com o endereço de todas as tarefas mapeadas no cluster.

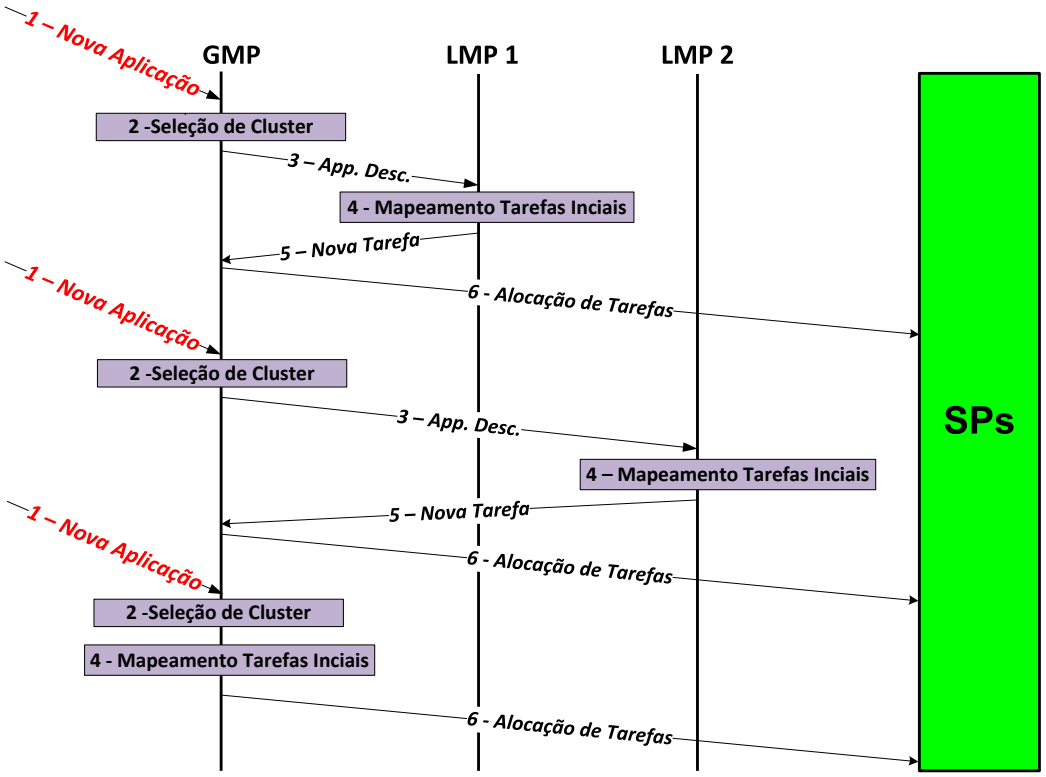

Figura 32 – Protocolo de Seleção de cluster e mapeamento das tarefas iniciais.

Considere na Figura 32 a terceira inserção de aplicação. Essa situação ilustra um cenário onde o cluster selecionado é gerenciado pelo próprio GMP. Nesse caso, o GMP também executa o algoritmo de mapeamento das tarefas inicias.

Como explicado anteriormente, o mapeamento de tarefas não-iniciais ocorre sempre que uma determinada tarefa t_i precisa se comunicar com uma tarefa t_j ainda não mapeada. Suponha no exemplo da Figura 33, onde a tarefa t_1, mapeada no SP_1, precisa se comunicar com uma tarefa t_2 ainda não mapeada. Nesse caso, a tarefa t_1 requisita o mapeamento da t_2 para o LMP do cluster, enviando uma mensagem com o pacote de "*Requisição de Tarefa*" ('1 – Requisição de Tarefas', Figura 33). O LMP recebe a requisição de tarefa e executa a heurística de mapeamento para selecionar um SP para mapear a tarefa t_2 ('2 – Heurística de mapeamento de tarefas', Figura 33).

No exemplo da Figura 33, a heurística de mapeamento, seleciona SP_2 para mapear a tarefa t_2. Em seguida, o LMP requisita o mapeamento da tarefa t_2 no SP_2 para o GMP, enviando um pacote de serviço de "requisição de alocação de tarefas" ('3 – Nova Tarefa', Figura 33). O LMP também usa o pacote de serviço de "alocação de tarefa" para informar o SP_1 a localização de t_2, e para o SP_2 a localização da tarefa t_1 ('4 – Localização de tarefa', Figura 33). Essas localizações são gravadas nas tabelas de tarefas dos SPs. Finalmente, o GMP obtém do repositório de aplicações o código objeto da tarefa t_2 e transmite para o SP2 ('5 – Alocação de tarefa', Figura 33).

As duas heurísticas de mapeamento propostas na Seção 6.3, usam o conceito de mapeamento hierárquico apresentado nas subseções a seguir. Nestas subseções são apresentadas as heurísticas de seleção de cluster, mapeamento das tarefas não-iniciais e mapeamento das tarefas inicias.

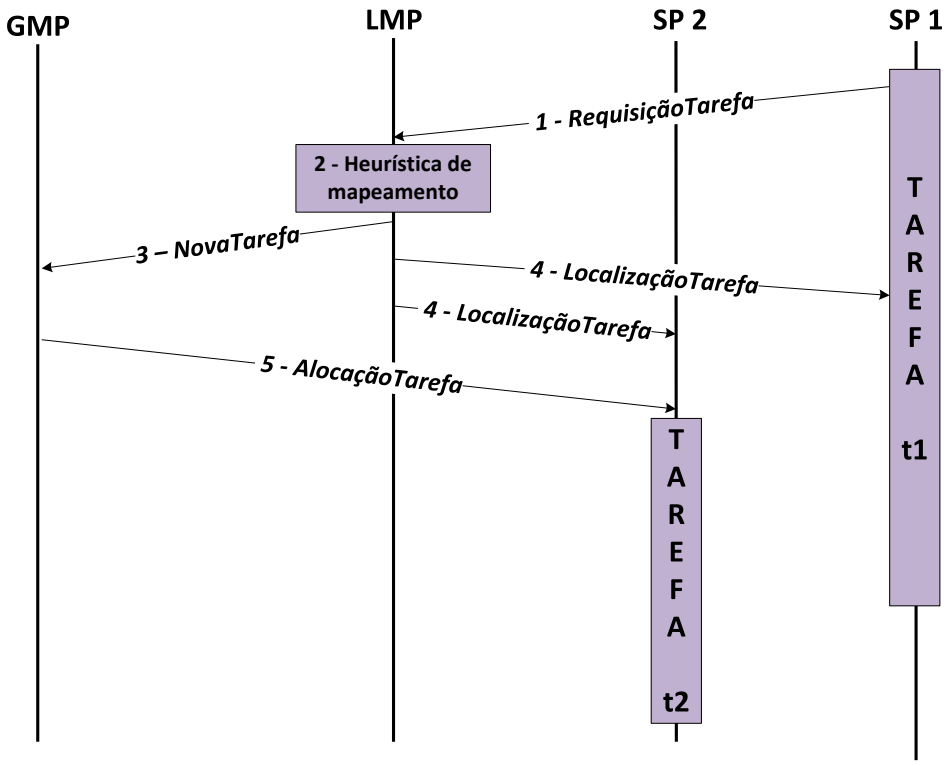

Figura 33 – Protocolo do mapeamento das tarefas não-iniciais.

6.2.1 Heurística de Seleção de Cluster

Para explicar a heurística de seleção de cluster, iremos usar a função custo "energia" como exemplo. Tal função custo pode ser alterada, dependendo do tipo de mapeamento de tarefas a ser utilizado. Esta heurística calcula a energia consumida em cada cluster c_k, *cl_energy*(c_k), usando dados enviados pelos pacotes de monitoramento. O cluster com o menor *cl_energy*(c_k) é selecionado. Este procedimento evita mapear um aplicativo em um cluster sobrecarregado, o que melhora a distribuição da carga de trabalho. O Algoritmo 1 apresenta o pseudocódigo da heurística de seleção de cluster.

O Algoritmo 1 inicialmente verifica se o sistema possui recursos disponíveis para mapear a aplicação (linha 3). Se não houver recursos suficientes no sistema, a aplicação é escalonada para ser mapeada posteriormente. O primeiro loop (linhas 4-9) analisa todos os clusters que possuem recursos disponíveis para mapear a aplicação, selecionando aquele com a menor energia acumulada. Se não houver um cluster com recursos disponíveis para mapear a aplicação, um cluster com a menor energia acumulada é selecionado, independentemente do número de recursos disponíveis (linhas 11-16). Observe que a aplicação é mapeada no MPSoC se e somente se o sistema tiver recursos disponíveis para a aplicação.

Esta heurística visa distribuir a energia de forma homogênea quando uma nova aplicação for admitida no sistema. A longo prazo, este procedimento evita hotspots e o desgaste dos processadores.

Algoritmo 1 - pseudocódigo da heurística de seleção de cluster.

```
Entrada: application size app.size
Saída: selected_cluster
1.  selected_cluster ← -1
2.  selected_cluster_energy ← ∞
3.  IF available_resources(system) >= APP.size THEN
4.      FOR EACH cluster c_k in the system
5.          IF available_resources(c_k) >= APP.size AND cl_energy (c_k)< selected_cluster_energy THEN
6.              selected_cluster ← c_k
7.              selected_cluster_energy ← cl_energy (c_k)
8.          END IF
9.      END FOR
10.     IF selected_cluster = -1 THEN
11.         FOR EACH cluster c_k in the system
12.             IF cl_energy (c_k)< selected_cluster_energy THEN
13.                 selected_cluster ← c_k
14.                 selected_cluster_energy ← cl_energy (c_k)
15.             END IF
16.         END FOR
17      END IF
18. END IF
19. return selected_cluster
```

6.2.2 Heurística de Mapeamento das Tarefas iniciais

O heurística de mapeamento das tarefas iniciais busca uma região com menor energia consumida no cluster. O espaço de busca é limitado pelo parâmetro *n_hops*, obtido de sqrt(|$PE_{cluster}$|/2), onde |$PE_{cluster}$| é o número de PEs no cluster. O raciocínio deste procedimento é mapear tarefas comunicantes próximas umas das outras, em um conjunto de PE com a menor energia acumulada.

Esta heurística divide o processo de mapeamento das tarefas iniciais em duas fases. A primeira fase seleciona um SP com a menor *region_energy* para receber uma tarefa inicial. Uma segunda fase é executada quando a aplicação possui mais de uma tarefa inicial. Nessa fase, é criado um conjunto com todos os SPs até *n hops* do SP selecionado, selecionando o SP deste conjunto com o menor energia total consumida (*TE*).

A função *region_energy(*sp_i*, n_hops)* retorna o TE média do conjunto contendo sp_i e todos os SPs até *n_hops hops* de sp_i. A Figura 34 mostra um exemplo hipotético usando um cluster 7x7, onde sp_i é o SP central $sp_{central}$ (em verde); e *n_hops* é 3 saltos. Na Figura 34, os números dentro de cada retângulo representam a TE de cada SP. O valor de *region_energy(*$sp_{central}$*, 3)* corresponde a 64, uma vez que: (*i*) dentro de uma região com 3 de *hops* de distância do $sp_{central}$ existem 25 SPs; (*ii*) a soma dos TEs dos SPs nesta área é igual a 4100; (iii) a TE média nesta área é igual a 4100/25 = 64.

Suponha que uma aplicação tenha duas tarefas iniciais: t_i e t_j. A primeira tarefa inicial t_i é

mapeada no $sp_{central}$ da Figura 34. Para o mapeamento da t_j é definida uma região de 3 *hops* a partir do $sp_{central}$, conforme delimitado pelos SPs numerados na Figura 34. O SP com o menor TE nesta região é escolhido para receber t_j. No exemplo, esse SP tem TE igual a 66, em destaque na Figura.

			123			
		66	178	280		
	114	200	80	109	77	
120	210	120	200	110	350	327
	124	156	85	413	95	
		149	123	189		
			102			

Figura 34 – Exemplo hipotético de *region_energy.*

O pseudocódigo da primeira fase do mapeamento de tarefas iniciais é detalhado no Algoritmo 2. O loop principal (linhas 3-8) seleciona um SP (*selected_sp*) com o menor *region_energy*. Este procedimento garante que as tarefas da aplicação que serão mapeadas posteriormente, serão mapeadas mais perto do SP selecionado e em SPs com menor energia acumulada.

Algoritmo 2 - Pseudocódigo da heurística de mapeamento das tarefas iniciais, primeira fase.

Input: *n_hops*
Output: *selected_sp*
1. selected_sp ← *-1*
2. selected_region_energy ← *+∞*
3. **FOR EACH** SP sp_i in the *cluster*
4. **IF** available(sp_i) **AND** region_energy(sp_i, n_hops)< selected_region_energy **THEN**
5. selected_sp ← sp_i
6. selected_region_energy ← region_energy(sp_i, n_hops)
7. **END IF**
8. **END FOR EACH**
9. return selected_sp

Se a aplicação tiver apenas uma tarefa inicial, o SP escolhido pela heurística do Algoritmo 2 é selecionado para executar a tarefa. Caso contrário, a heurística apresentada no Algoritmo 3 é executada para cada tarefa inicial não mapeada. Na linha 4 é criado um conjunto *neighbors_list* com todos os SPs até n_*hops* de *selected_sp* computado na fase anterior. O loop entre as linhas 6-11 seleciona um SP disponível da lista de vizinhos com o menor TE. Se não houver SP disponível dentro da lista, o espaço de busca aumenta 1 *hop* (linhas 12-15), até visitar todos os SPs do cluster (linha 5).

Algoritmo 3 - Pseudocódigo da heurística de mapeamento das tarefas iniciais, segunda fase.

```
Input: SP_address, n_hops          // SP_address is the selected_sp address obtained in the 1st phase
Output: selected_sp
1.  selected_sp ← -1
2.  selected_sp_energy ← +∞
3.  // Get all neighbors of selected_sp within a distance n_hops
4.  neighbors_list ← neighbors(SP_address, n_hops)
5.  WHILE all SPs in the cluster not evaluated AND selected_sp=-1 DO
6.      FOR EACH SP sp_i IN neighbors_list
7.          IF available(sp_i) = true AND TE(sp_i) < selected_sp_energy THEN
8.              selected_sp ← sp_i
9.              selected_sp_energy ← TE (sp_i)
10.         END IF
11.     END FOR
12.     IF selected_sp = -1 THEN
13.         n_hops ← n_hops +1
14.         neighbors_list ← neighbors(SP_address, n_hops)
15.     END IF
16. END WHILE
17. return selected_sp
```

6.2.3 Heurística de Mapeamento de Tarefas não-iniciais

Suponha que uma tarefa não-inicial t_i é requerida para ser mapeada. A heurística de mapeamento de tarefas não iniciais avalia o conjunto C(t_i) (tarefas comunicantes com t_i), e cria uma caixa delimitadora contendo todas as tarefas comunicantes com t_i mapeadas no cluster (*boundig box*). Esta caixa delimitadora é aumentada em um *hop* oferecendo um maior espaço de busca à heurística. Os limites do cluster limitam o espaço de busca. A Figura 35 ilustra o espaço de busca de mapeamento no cluster. Esta heurística seleciona o SP dentro da caixa delimitadora com o TE mais baixo. Esta heurística faz um *trade-off* entre balanceamento de carga de trabalho e redução de volume de comunicação. O Algoritmo 4 apresenta o pseudocódigo usado para selecionar um SP para receber uma tarefa não-inicial t_i.

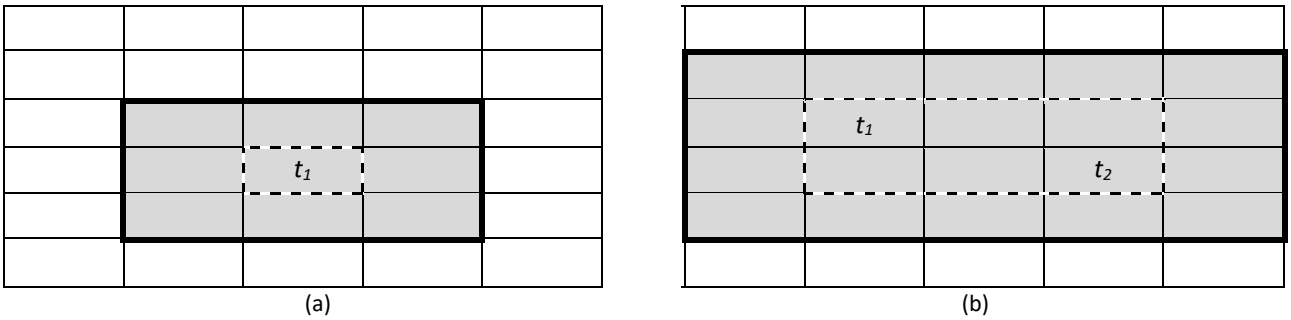

Figura 35 – Espaço de busca de mapeamento de tarefas não-iniciais, delimitado pelas linhas mais espessas. As linhas tracejadas correspondem à área inicial, sem o aumento de 1 hop. (a) espaço de busca quando uma tarefa que se comunica com t_i já está mapeada (t_1). (b) espaço de busca quando mais de uma tarefa comunicante já está mapeada (t_1 e t_2).

Algoritmo 4 - Pseudocódigo da heurística de mapeamento das tarefas não iniciais.

```
Input: t_i , set C(t_i)
Output: selected_sp
1.  selected_sp ←-1
2.  selected_sp_energy ← +∞
3.  MC(t_i) ← mapped_tasks(C(t_i))          // all tasks communicating with t_i already mapped
4.  bounding_box ← area(MC(t_i))
5.  increase(bounding_box, 1 hop)
6.  WHILE all SPs in the cluster were not evaluated AND selected_sp=-1 DO
7.      neighbors_list ← search_SPs(bounding_box)
8.      FOR EACH SP sp_i IN neighbors_list
9.          IF available(sp_i) = true AND TE(sp_i)< selected_sp_energy THEN
10.             selected_sp ← sp_i
11.             selected_sp_energy ← TE(sp_i)
12.         END IF
13.     END FOR
14.     IF selected_sp = -1 THEN
15.         increase(bounding_box, 1 hop)
16.     END IF
17. END WHILE
18. return selected_sp
```

A heurística cria uma lista com todas as tarefas que se comunicam com t_i já mapeadas nos SPs do cluster (linha 3). Na sequência, é definido uma caixa delimitadora retangular (linha 4), com todas as tarefas comunicantes mapeadas. Esta caixa delimitadora é aumentada por um *hop* (linha 5), oferecendo um espaço de busca maior para mapear a t_i. É criada uma lista com SPs candidatos (linha 7). O SP disponível na lista com o menor TE é selecionado (linhas 8-13). Se nenhum SP puder ser selecionado, a caixa delimitadora é aumentada em um *hop* (linhas 14-16). Este processo continua até encontrar um SP ou até que todos os SPs do cluster tenham sido visitados.

Os Algoritmo 3 e Algoritmo 4 podem retornar -1, o que significa que o cluster não possui um SP disponível para receber a tarefa. Nessa situação, o *µkernel* toma emprestado um SP de um cluster vizinho (processo chamado reclusterização), mapeando a tarefa no SP emprestado.

6.3 Diferenças entre Mapeamento de Tarefas Propostos

Utilizando o método de mapeamento hierárquico proposto por essa Tese, foi possível criar duas variações, basicamente mudando a função custo do mapeamento. Essas variações são:

- ***Energy:*** Essa heurística usa como função custo a energia monitorada em tempo real nos PEs. Essa heurística faz uma relação custo-benefício entre distribuir a carga de trabalho (energia do processor e do roteador) e reduzir o volume de comunicação. [MAN15a]

- ***Temperature:*** Essa heurística usa como função custo a temperatura estimada pelo modelo proposto por essa Tese. O intuito dessa heurística, é obter um melhor balanceamento término ao longo do tempo no MPSoC, melhorando sua confiabilidade.

6.4 Resultados

Nessa seção, é realizada a análise de resultados visando o balanceamento energético e térmico e a confiabilidade do sistema.

Os experimentos foram executados no MPSoC de referência, utilizando um modelo com precisão de ciclo, descrito em SystemC. Cada SP pode executar até duas tarefas simultaneamente. Foi utilizado as duas variações do mapeamento hierárquico proposto nesta Tese, *Energy* e *Temperature*, e como mapeamento de referência para os testes, foi utilizado a heurística de mapeamento LEC-DN proposto por [MAN15b].

A heurística LEC-DN considera as dependências entre todas as tarefas comunicantes, utilizando como função custo a minimização da energia de comunicação da NoC. Para minimizar a energia de comunicação, esta heurística usa o volume de comunicação entre as tarefas, uma vez que o número de flits transmitidos define a energia de comunicação. Esta heurística foi selecionada como referência, pois sua função de custo é a adotada na maioria dos sistemas baseados em NoC: minimizar a energia de comunicação

Cinco benchmarks, descritos em linguagem C, foram utilizados: (*i*) DTW - Digital Time Warping (DTW), com 10 tarefas; (*ii*) MPEG decoder, com 5 tarefas; (*iii*) DJK - Dijkstra, com 6 tarefas; (*iv*) SYN1, aplicação sintética, com 12 tarefas, que emula o comportamento da comunicação de um decodificador completo MPEG4; (*v*) SYN2, aplicação sintética, com 12 tarefas, que emula o comportamento da comunicação de um decodificador VOP (Video Object Plane).

Os experimentos foram conduzidos utilizando os cenários apresentados na Tabela 16. Os cenários 1 e 2 contém um mix de aplicações, quanto os cenários de 3 a 5 tem aplicações idênticas. Espera-se que cenários com aplicações idênticas gerem uma solução de mapeamento com a mesma carga. A última coluna da Tabela 16 corresponde à média do número de tarefas por SP. Cenários com valores maiores nessa coluna correspondem a carga de trabalho mais pesada.

Tabela 16 – Características dos cenários avaliados.

Cenários	Tamanho do MPSoC	Tamanho do Cluster	Aplicações	Número total de tarefas	Número de tarefas por SP
1	8x8 (60 SPs)	4x4	20 x MPEG, 20 x DJK, 20 x SYN1, 20 x SYN2, 20 x DTW	780	13
2			10 x MPEG, 10 x DJK, 10 x SYN1, 10 x SYN2, 10 x DTW	390	6.5
3			50 x MPEG	250	4.17
4			100 x DTW	1000	16.67
5			100 x MPEG	500	8.33

6.4.1 Avaliação de Energia

A Tabela 17 apresenta a avaliação do total de consumo de energia por PE, para cada cenário testado. Nela podemos observar 4 colunas, as quais são:

- **Máxima**: Nessa coluna, se tem a informação de energia máxima consumida por um único PE. Podemos observar que a heurística LEC-DN obtém um valor superior as demais heurísticas em todos os cenários. Isso é devido ao fato da mesma ter como objetivo, a aproximação das tarefas comunicantes para diminuir a energia de comunicação. Vale ressaltar que mesmo a heurística "*Energy*" buscando o melhor balanceamento energético, a heurística "*Temp.*" obteve melhores resultados em todos os cenários.
- **Mínima**: Nessa coluna, se tem a informação de energia mínima consumida por um único PE. Nota-se que em todos os cenários, a heurística LEC-DN obteve valor '0', o que significa que em todos os cenários testados, algum PE do MPSoC não executou nenhuma tarefa. Esse comportamento faz com que se crie regiões frias no chip, sobrecarregando as demais regiões. Observando a coluna referente a heurística de mapeamento "*Temp.*", ressalta-se que em todos os cenários, a energia mínima consumida por um PE foi superior às demais heurísticas. Esse fato é positivo, pois demonstra que todos os PEs do sistema, estiveram em funcionamento.
- **Média**: Considerando que a carga de trabalho aplicada para todas as heurísticas de mapeamento é a mesma para cada cenário, uma pequena variação é esperada. Ambas as heurísticas propostas apresentam um pequeno aumento no total de energia média, isso é explicado pelo fato de mais processadores serem atribuídos para executar tarefas, levando a uma execução adicional de instruções. Quando um determinado processador não está executando nenhuma tarefa, ele entra em um estado de espera, dissipando apenas energia estática.
- **Desvio Padrão**: Analisando a coluna de Desvio Padrão, podemos observar que a heurística "*Temp.*" apresenta um desvio padrão inferior aos demais, caracterizando uma melhor distribuição energética.

Tabela 17 – Avaliação da Total de energia/PE (Temp ➔ heurística proposta com função custo a temperatura).

Cenário	Máxima (uJ)			Mínima (uJ)			Média (uJ)			Desvio Padrão (uJ)		
	LEC-DN	Energy	***Temp***.	LEC-DN	Energy	***Temp***.	LEC-DN	Energy	***Temp***.	LEC-DN	Energy	***Temp***
1	489,60	396,02	394,056	0,00	4,31	55,70	192,05	198,28	207,62	115,96	94,86	71,15
2	215,85	200,33	187,32	0,00	4,31	5,23	92,42	96,62	106,91	61,43	50,79	39,55
3	159,25	130,36	87,44	0,00	2,34	14,76	39,79	40,74	42,62	47,29	41,42	16,21
4	111,17	108,16	96,20	0,00	26,59	46,70	70,41	71,21	73,96	32,58	19,42	12,09
5	316,49	193,02	174,644	0,00	13,96	30,62	82,76	86,19	88,58	93,46	43,24	32,58

6.4.2 Avaliação de Temperatura

A Tabela 18 apresente a avaliação de temperatura, para cada cenário testado. Nela podemos observar 3 colunas, as quais são:

- **Temperatura Máxima**: Podemos observar, que utilizando as heurísticas de mapeamento "*Energy*" ou "*Temp.*" obteve-se resultados de temperatura máxima inferiores aos registrados nos cenários utilizando a heurística de mapeamento LEC-DN. Isso ocorreu devido ao fato da LEC-DN agrupar as tarefas em um mesmo processador ou região, aumentando consequentemente a sua temperatura. Podemos ressaltar também, que mesmo a heurística "*Energy*" conseguindo obter uma redução de temperatura máximo no melhor caso de 3,4% em relação a LEC-DN, pode-se observar que a heurística "*Temp.*" melhorou ainda mais, no melhor caso 7,6%. Essa diferença é obtida, pois a função custo da "*Temp.*" é a temperatura estimada pelo modelo proposto nessa Tese, e tal função custo, leve em conta não somente a temperatura gerada pela energia consumida no próprio PE, mas também pela influência térmica dos seus PEs vizinhos, o que acarreta uma maior precisão e logo uma melhor distribuição térmica.
- **Temperatura Média**: A diferença da temperatura nas três heurísticas é muito pequena em todos os cenários. Isso é devido ao fato que estes possuem a mesma carga de trabalho o que faz com que a energia consumida seja muito similar entre eles, e consequentemente espera-se que a temperatura (que é uma função da energia), na média seja similar também.
- **Desvio Padrão da Temperatura**: O fato da temperatura média dos cenários avaliados serem muito similares, não obrigatoriamente quer dizer que estes possuem uma distribuição térmica similar. Como podemos obsevar na coluna em questão, a heurística "*Temp.*" chega a ter, no melhor caso, um desvio padrão da temperatura 49,9% inferior ao registrado na heurística LEC-DN, o que se pode concluir que mesmo com uma temperatura média similar, o balanceamento térmico atingido pela heurística proposta baseada em temperatura é superior ao LEC-DN e a heurística baseada em energia.

Tabela 18 – Resultados de Temperatura dos cenários avaliados

Cenário	Temperatura Máxima (ºC)			Temperatura Média (ºC)			Desvio Padrão da Temperatura (ºC)		
	LEC-DN	Energy	***Temp***.	LEC-DN	Energy	***Temp***.	LEC-DN	Energy	***Temp***.
1	58,168	57,406	56,021	49,207	49,107	49,032	2,499	2,223	1,821
2	55,465	55,077	53,445	48,279	48,157	48,039	2,224	2,193	1,487
3	53,939	52,086	52,027	47,421	47,263	47,126	2,082	1,725	1,271
4	53,748	52,844	51,602	48,724	48,617	48,537	2,016	1,663	1,374
5	56,308	55,080	52,042	48,369	48,089	48,011	2,575	1,923	1,290
Razão/LEC-DN:		**-3,4%**	**-7,6%**		**-0,6%**	**-0,7%**		**-25,3%**	**-49,9%**

6.4.3 Confiabilidade

Para efetuar a análise da confiabilidade de um processador, foi utilizado uma ferramenta chamada *RAMP* (*Reliability Aware Micropocessor*) [SRI04]. Esta ferramenta expressa a confiabilidade em termos de MTTF (tempo médio entre falhas ou exceptiva de tempo de vida de um processador).

O tempo de vida, ou confiabilidade, de um processador está diretamente relacionado à temperatura de operação, da onde se esperam problemas caso a temperatura do processador seja drasticamente alterada. Usualmente, a relação entre as taxas de falhas de um processador em diferentes temperaturas é expressa numa forma semelhante à equação de Arrhenius. Esse tipo de equação modela a dependência observada que a taxa de uma reação química tem em relação à temperatura ou mudanças de temperatura [JED16]. [SRI04] afirmou que os mecanismos críticos de falhas intrínsecas de processadores são:

- **Eletromigração:** é a consequência da depleção de metal de alguma região do condutor e seu acúmulo em outras regiões. O resultado é uma abertura de circuito na região onde o metal sai. [SRI04] propõe um modelo para o MTTF devido à eletromigração. Com base nesse modelo, uma temperatura média menor aumenta o MTTF devido à eletromigração.
- **Migração por Estresse:** é ocasionada pelo movimento de átomos de metal, que por sua vez é ocasionado por estresse mecânico [JED16]. Tal estresse é devido a diferentes taxas de expansão térmica de diferentes matérias no dispositivo. [JED16] propõe um modelo para o MTTF devido à migração por estresse. Assim como ocorre na eletromigração, uma menor temperatura média de operação, aumenta o MTTF devido à migração por estresse.
- **Ruptura dielétrica dependente do tempo:** dielétricos são usados em microeletrônica devido a sua propriedade isoladora. A ruptura do dielétrico é a formação de um caminho de baixa resistência condutiva através do dielétrico, formação a qual compromete as propriedades isoladoras, ocasionando falhas irreversíveis no sistema. Nesse mecanismo de falha, a temperatura desempenha uma influência mais que exponencial [SRI04]. [WU02] modela o MTTF devido à ruptura dielétrica dependente do tempo.
- **Ciclos Térmicos:** a variação de temperatura também causa fadiga. O acúmulo dessa fadiga toda vez que há um ciclo térmico eventualmente pode ocasionar uma falha. [SRI04] modela o MTTF devido à ciclos térmicos.

A ferramenta RAMP usa os 4 modelos citados acima para calcular o MTTF instantâneo do sistema, baseado na temperatura e utilização atual. O método de análise utilizado no RAMP parte do modelo de soma de taxa de falhas, SOFR (Sum-of-Failure Rates). Esse modelo é útil para determinar o MTTF de um sistema, como apontado em [ROS07]. Ele trata a falha de cada estrutura do sistema como independente das falhas das demais estruturas.

O modelo utilizado na ferramenta RAMP, possibilidade dividir o sistema em múltiplas

estruturas, o que é fundamente na análise de confiabilidade de um MPSoC.

O modelo RAMP requer como entrada, a temperatura de cada estrutura do sistema, ao longo do tempo, e fornece como saída, o MTTF (em anos) do MPSoC para um dado cenário testado.

Foram analisados os 5 cenários em um MPSoC 8x8, e comparado o resultado do teu MTTF, como mostra a Tabela 19.

Como podemos observar nessa tabela, tanto a heurística de mapeamento "*Energy*", que visa o balanceamento enérgico, quanto a heurística de mapeamento "*Temp.*", que visa o balanceamento térmico obtiveram resultados superiores de MTTF em comparação com a heurística de mapeamento LEC-DN. No cenário 3, se obteve o melhor ganho, passando de uma expectativa de vida de 39,04 anos, usando a heurística LEC-DN para uma expectativa de vida de 56,64 anos usando a heurística "*Temp.*". Já comparando a heurística LEC-DN com a heurística "Energy", obteve-se o maior ganho em expectativa de vida no cenário 4, com um acréscimo de 24,19% em relação ao LEC-DN.

Esses resultados são obtidos, devido a forma que a heurística LEC-DN trabalha. Como ela tem o objetivo de minimizar a comunicação entre tarefas, ela acaba agrupando as tarefas comunicantes em um mesmo processador ou muito próximas, criando regiões muito quentes (hotspots) o que acelera o desgaste do MPSoC e diminui a sua confiabilidade. Já os dois mapeamentos propostos, visam balancear o sistema (ou energeticamente ou termicamente), o que dente a evitar regiões quentes, o que aumente a confiabilidade do sistema.

Tabela 19 – Resultados de MTTF para os cenários avaliados

Cenário	Anos		
	LEC-DN	Energy	Temp.
1	14,020	14,180	18,010
2	23,010	26,980	29,100
3	39,040	44,450	56,640
4	21,820	27,100	32,200
5	25,850	26,180	29,830
Razão/LEC-DN:		+24,19%	+45,08%

7 CONCLUSÃO E TRABALHOS FUTUROS

A Introdução da Tese afirmou a seguinte hipótese: “Esta Tese busca demonstrar duas hipóteses: *(i)* é possível estimar a temperatura de um MPSoC em tempo de execução com precisão e baixo custo computacional; e *(ii)* mapeamento de tarefas tendo como função custo a temperatura instantânea do chip pode aumentar a confiabilidade do sistema.

A proposição de um modelo de estimativa de temperatura baseado em software, que usa como entrada um modelo de monitoramento hierárquico para estimar a energia consumida por cada PE, permitiu estimar a temperatura no nível de PE em MPSoCs com precisão, em relação ao um modelo de referência e com baixo custo computacional. Consequentemente, a primeira parte da hipótese acima está corroborada.

Foi proposta uma heurística de mapeamento de tarefas hierárquico, que usou como função custo os valores obtidos pelo modelo de temperatura previamente proposto. Com isso, e em comparação com outras heurísticas de mapeamento, obteve-se um melhor balanceamento térmico e consequentemente um aumento na confiabilidade do sistema. Assim, a segunda parte da hipótese acima também está demonstrada.

O presente trabalho explorou três eixos principais, como mostrado na Figura 36:

1. Monitoramento. O primeiro eixo é o monitoramento do sistema. Nessa fase foi proposto um monitoramento hierárquico capaz de estimar a energia consumida por cada processador.
2. Heurísticas: O segundo eixo é a execução de heurísticas. Foi proposto duas heurísticas para modelar o comportamento energético e o comportamento térmico do sistema, com o intuído de utilizar os valores calculados na fase posterior.
3. Atuações: O terceiro eixo é a execução do processo de atuação, utilizando os valores calculados na fase anterior. Para isso, foi proposto uma heurística de mapeamento, que tem como objetivo melhorar o balanceamento térmico e consequentemente a confiabilidade e o tempo de vida do sistema.

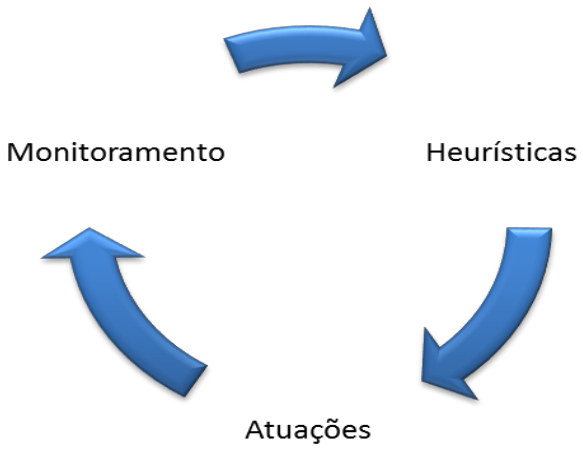

Figura 36 – Ciclo do sistema proposto por essa Tese.

Essa Tese começou apresentando uma alteração na plataforma de referência, para que ela utilize uma gerência distribuída de recursos. Tal alteração foi necessária para o restante do trabalho. Também foi apresentado, um *framework* de geração de MPSoCs, o qual foi utilizado em todas as etapas de testes dessa Tese, facilitando as mudanças de características dos diferentes cenários testados e a analise de resultados.

Foi proposto um método para definir os parâmetros de energia/potência para a NoC e o processador, aplicando isso em um modelo em alto-nível. Os resultados mostram que: (*i*) é possível estimar a energia por aplicação; (*ii*) a maior parte da energia consumida vem dos processadores (aproximadamente 90%); (*iii*) mesmo que a NoC esteja subutilizado (normalmente as cargas do link estão abaixo de 5%), a NoC permite transações paralelas, sendo escaláveis em comparação com os barramentos; (*iv*) a energia consumida nos fios é pequena (cerca de 1%). No entanto, várias direções para trabalhos futuros são identificadas: (*i*) modelar a energia consumida nas memórias locais, módulos DMA e NI; (*ii*) modelar o efeito do congestionamento no NoC quando dois PE tentam se comunicar com o mesmo PE alvo.

Essa Tese também propôs um modelo de temperatura para monitorar a temperatura do sistema em tempo de execução. As características desse modelo de temperatura incluem escalabilidade, baixo custo de computação e execução em tempo de execução. A ferramenta HotSpot é usada para gerar o comportamento térmico do sistema (fase de tempo de projeto), que é usado para calcular a temperatura do sistema em tempo de execução (fase de tempo de execução). O modelo proposto alcançou um baixo valor erro, com impacto mínimo no consumo de energia (<4%) e sobrecarga de desempenho (<5%). Os resultados mostram que o erro absoluto médio da estimativa de temperatura, em comparação com o HotSpot, é menor do que 4% em sistemas com até 36 elementos de processamento.

A proposta de atuação dessa Tese foi de um mapeamento hierárquico de tarefas, tendo como função custo a temperatura de cada PE. Obteve-se no melhor caso, aproximadamente 8% de redução da temperatura máxima em comparação com a heurística de referência, e um desvio padrão 50% inferior, o que caracteriza um ganho substancial em balanceamento térmico. Como trabalhos futuros na parte de atuação, incluem usar técnicas de DVFS, que seria ativada somente quando uma violação de temperatura é atingida.

7.1 Publicações

Os trabalhos publicados referentes a essa Tese de Doutorado são mostrados na Tabela 20. Cada tema proposto por essa Tese possui um trabalho publicado respaldando o mesmo. Iniciando pela mudança da plataforma de referência para uma plataforma com gerência distribuída [CAS13] e a criação de uma plataforma de geração de MPSoCs [CAS14], que serviram para dar suporte ao restante da tese.

Após, temos o monitoramento hierárquico de energia [CAS16b], que serve de motor para as modelagens propostas. Tais modelagens, tanto energética [MAR14] quanto térmica [CAS16a] usam

como suporte os trabalhos previamente publicados. Para finalizar, as heurísticas de mapeamento [CAS16b] [MAN15a] que utilizam os conceitos propostos por essa tese, para atuar no MPSoC e proporcionar um ganho, tanto energético quanto térmico.

Tabela 20 – Publicações relacionadas com a Tese de Doutorado

Título	Descrição	Publicação	Localização
Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes [CAS13]	Plataforma de Referência – Gerência Distribuída	ISVLSI 2013	Capítulo 3
A Framework for MPSoC Generation and Distributed Applications Evaluation [CAS14]	Framework para Geração de MPSoC	ISQED 2014	Seção 2.3
Hierarchical Energy Monitoring for Task Mapping in Many-core Systems [CAS16b]	Monitoramento Hierárquico de Energia, Mapeamento hierárquico de tarefas	JSA 2016	Capítulo 4 e Capítulo 6
A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems [MAN15a]	Mapeamento de Tarefas utilizando a energia como função custo	SBCCI 2015	Seção 6.3
A Method for NoC-based MPSoC Energy Consumption Estimation [MAR14]	Modelagem Energética	ICECS 2014	Seção 5.1
A Lightweight Software-based Runtime Temperature Monitoring Model for Multiprocessor Embedded Systems [CAS16a]	Modelagem Térmica	SBCCI 2016	Seção 5.2

REFERÊNCIAS

[AJA05] Ajami, A. H.; Banerjee, K.; Pedramm, M. “Modeling and analysis of nonuniform substrate temperature effects on global ULSI interconnects”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.24(6), 2005, pp. 849-861.

[ALF08] Al Faruque, M.; Krist, R.; Henkel, J. “ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication”. In: DAC, 2008, pp. 760-765.

[ALM10] Almeida, G. M.; Varyani, S.; Busseuil, R.; Sassatelli, G.; Benoit, P.; Torres, L.; Carara, E.; Moraes, F.G. “Evaluating the Impact of Task Migration in Multi-Processor Systems-on-Chip”. In: SBCCI, 2010, pp. 73-78.

[ANA12] Anagnostopoulos, I.; Bartzas, A.; Kathareios, G.; Soudris, D. “A divide and conquer based distributed run-time mapping methodology for many-core platforms”. In: DATE, 2012, pp. 111-116.

[ANG06] Angiolini, F.; Ceng, J.; Leupers, R.; Ferrari, F.; Ferri, C.; Benini, L. “An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration”. In: DATE, 2006, 6p.

[ATI07] Atitallah. R.; Niar. S.; Dekeyser. J. “MPSoC power estimation framework at transaction level modeling”. In: ICM, 2007, pp. 245–248.

[BOL13] Bolchini, C.; Carminati, M.; Miele, A.; Das, A.; Kumar, A.; Veeravalli, B. “Runtime mapping for reliable many-cores based on energy/performance trade-offs”. In: DFT, 2013, pp. 58–64.

[BOR18] Boraten, T.; Kodi, A. “Runtime Techniques to Mitigate Soft Errors in Network-on-Chip (NoC) Architectures”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.37(3), 2018, pp. 682-695.

[CAR09] Carara, E. A.; Oliveira, R. P.; Calazans, N. L. V.; Moraes, F. G. “HeMPS - A Framework for Noc-Based MPSoC Generation”. In: ISCAS, 2009, pp. 1345-1348.

[CAR10] Carvalho, E.; Calazans, N.; Moraes, F. G. “Dynamic Task Mapping for MPSoCs”. IEEE Design & Test of Computers, v.27(5), 2010, pp. 26-35.

[CAS13] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. G. “Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes”. In: ISVLSI, 2013, pp. 153-158.

[CAS14] Catilhos, G.; Wachter, E.; Madalozzo, G.; Erichsen, A.; Monteiro, T.; Moraes, F. G. “A Framework for MPSoC Generation and Distributed Application Evaluation”. In: ISQED, 2014, pp. 408-411.

[CAS16a] Castilhos, G.; M.; Ost, L.; Moraes, F. G. “A Lightweight Software-based Runtime Temperature Monitoring Model for Multiprocessor Embedded Systems”. In: SBCCI, 2016, 6p.

[CAS16b] Castilhos, G.; Mandelli, M.; Ost, L.; Moraes, F. G. “Hierarchical Energy Monitoring for Task Mapping in Many-core Systems”. Journal of Systems Architecture, v.63, 2016, pp. 80–92.

[CHA05] Chan. J.; Parameswaran. S. “NoCEE: energy macro-model extraction methodology for network on chip routers”. In: ICCAD, 2005, pp. 254- 259.

[CHA13] Chantem, T.; Xiang, Y.; Hu, X. S.; Dick, R. P. "Enhancing multicore reliability through wear compensation in online assignment and scheduling". In: DATE, 2013, pp. 1373 -1378.

[CHO10] Chou, C-L.; Marculescu, R. “Run-Time Task Allocation Considering User Behavior in Embedded Multiprocessor Networks-on-Chip". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.29(1), 2010, pp. 78–91.

[COS09] Coskun, A. K.; Ayala, J. L.; Atienza, D.; Rosing, T. S.; Leblebici, Y. “Dynamic thermal management in 3D multicore architectures”. In: DATE, 2009, pp. 1410-1415.

[CUI12] Cui, Y.; Zhang, W.; Yu, H. “Decentralized Agent Based Re-Clustering for Task Mapping of Tera-Scale Network-on-Chip System”. In: ISCAS, 2012, pp. 2437-2440.

[DAS14] Das, A.; Kumar, A.; Veeravall, B. “Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs”. In: DATE, 2014, 6p.

[DIA12] Diamantopoulos, D.; Siozios, K.; Xydis, S.; Soudris, D. “A Systematic Methodology for Reliability Improvements on SoC-Based Software Defined Radio Systems”. VLSI Design, v.2012, Article ID 784945, 2012, 15 p.

[DIN16] Ding, H.; Ma, S.; Huang, M.; Andrews, D. "OOGen: An Automated Generation Tool for Custom MPSoC Architectures Based on Object-Oriented Programming Methods". In: IPDPSW, 2016, pp. 233-240.

[EBI12] Ebi, H. A.; Amrouch, H.; Henkel, J. “COOL: control-based optimization of load-balancing for thermal behavior”. In: CODES+ISSS, 2012, pp. 255-264.

[FAT11] Fattah, M.; Daneshtalab, M.; Liljeberg, Pasi; Plosila J. “Exploration of MPSoC Monitoring and Management Systems”. In: ReCoSoC, 2011, 3p.

[GE10] Ge, Y.; Malani, P.; Qiu, Q. “Distributed task migration for thermal management in many-core systems”. In: DAC, 2010, pp. 579–584.

[GOO11] Goodarzi, B.; Sarbazi-Azad, H. “Task Migration in Mesh NoCs over Virtual Point-to-Point Connections”. In: Euromicro, 2011, pp. 463-469.

[GUI08] Guindani, G.; Reinbrecht, C.; Raupp, T.; Calazans, N.; Moraes, F.G. “NoC Power Estimation at the RTL Abstraction Level”. In: ISVLSI, 2008, pp. 475-478.

[GUN01] Gunther, S.; Binns, F.; Carmean, D. M.; Hall, J. C. “Managing the impact of increasing microprocessor power consumption”. Intel Technology Journal, V.5(1), 2001, 9p.

[GUP10] Gupta, T.; Bertolini, C.; Heron, O.; Ventroux, N.; Zimmer, T.; Marc, F. “High level power and energy exploration using ArchC”. In: SBAC-PAD. 2010. pp. 25–32.

[HAA06] Haase, J.; Damm, M.; Hauser, D.; Waldschmidt, K. “Reliability-aware power management of multi-core processors”. From Model-Driven Design to Resource Management for Distributed Embedded Systems, Springer, 2006, PP. 205-214.

[HAR12] Hartman, A.; Thomas, D. “Lifetime improvement through runtime wear-based task mapping”. In: CODES+ISSS, 2012, pp. 13-22.

[HEN13] Henkel, J.; et al. “Reliable on-chip systems in the nano-era: Lessons learnt and future trends”. In: DAC, 2013, 10p.

[HOL08] Hölzenspies, P. K. F.; Hurink, J. L.; Kuper, J.; Smit, G. J. M. “Runtime Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)”. In: DATE, 2008, pp. 212-217.

[HOW10] Howard, J; et al. “A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS”. In: ISSCC, 2010, pp. 108-109.

[HRU08] Hruska, J. “NVIDIA denies rumors of faulty chips, mass GPU failures”. Capturado em: https://arstechnica.com/gadgets/2008/07/nvidia-denies-rumors-of-mass-gpu-failures, Julho, 2008.

[HU03] Hu, J.; Marculescu, R. “Energy-aware mapping for tile-based NoC architectures under performance constraints”. In: ASP-DAC, 2003, pp. 233-239.

[INT16] Intel. “The Intel® Xeon Phi™ Coprocessor”, 2016.

[INT13] International Technology Roadmap for Semiconductors. Capturado em: http://www.itrs.net/reports.html, 2013.

[JED16] JEDEC. “Failure mechanisms and models for semiconductor devices”. Capturado em: https://www.jedec.org/standards-documents/docs/jep-122e, Setembro, 2016.

[JER05] Jerraya, A. A.; Wolf, W. “Multiprocessor Systems-on-Chips”. Morgan Kaufmann Publishers Inc, 2005, 602p.

[JOV08] Joven, J.; Font-Bach, O.; Castells-Rufas, D.; Martinez, R.; Teres, L.; Carrabina, J. “xENOC – An eXperimental Network-on-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures”. In: Euromicro, 2008, pp. 141-148.

[JSS11] J. S. S. T. Association et al., “Failure mechanisms and models for semiconductor devices”. In: JEDEC Publication 2013.

[KOB11] Kobbe, S.; Bauer, L.; Lohmann, D.; Schroder-Preikschat, W.; Henkel, J. “DistRM: Distributed Resource Management for On-Chip Many-Core Systems”. In: CODES+ISSS, 2011, pp. 119-128.

[KON12] Kong, J.; Chung, S. W.; Skadron, K. “Recent Thermal Management Techniques for Microprocessors”. ACM Computing Surveys, v.44(3), 2012, 42 p.

[KUM11] Kumar, P.; Thiele, L. “Thermally optimal stop-go scheduling of task graphs with real-time constraints”. In: ASP-DAC, 2011, pp. 123-128.

[KUR08] Kursun, E.; Cher, C.-Y. “Variation-aware thermal characterization and management of multi-core architectures”. In: ICCD, 2008, pp. 280–285.

[LEE05] Lee, K. J.; Skadron, K. “Using performance counters for runtime temperature sensing in high-performance processors”. In: IPDPS, 2005, 8p.

[LEM12] Lemaire, R.; Thuries, S.; Heiztmann, F. “A flexible modeling environment for a NoC-based multicore architecture”. In: HLDVT, 2012, pp. 140-147.

[LIU15] Liu, Z.; Tan, S. X. D.; Huang, X.; Wang, H. “Task Migrations for Distributed Thermal Management Considering Transient Effects”. IEEE Trans. on Very Large-Scale Integration Systems, v.23(2), 2015, pp. 397–401.

[LU10] Lu, S.; Lu, C.; Hsiung, P. “Congestion- and energy-aware runtime mapping for tile-based network-on-chip architecture”. In: Frontier Computing: Theory, Technologies and Applications, 2010, pp. 300–305.

[MAN11] Mandelli, M. “Mapeamento Dinâmico de Aplicações para MPSoCs Homogêneos”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 106p.

[MAN15a] Mandelli, M.; Castilhos, G. M.; Sassatelli, G.; Ost, L.; Moraes, F. G. “A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems”. In: SBCCI, 2015, 6p.

[MAN15b] Mandelli, M.; Ost, L.; Sassatelli, G.; Moraes, F. "Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability". In: ISQED, 2015, pp. 392-396.

[MAR14] Martins, A.; Silva, D.; Castilhos, G.; Monteiro, T.; Moraes, F. G. "A Method for NoC-based MPSoC Energy Consumption Estimation". In: ICECS, 2014, pp. 427-430.

[MAR15] Martins, A.; Ruaro, M.; Moraes, F. "Hierarchical Energy Monitoring for Many-Core Systems". In: ICECS, 2015, pp. 657-660.

[MAS16] Massari. G.; et al. “Combining application adaptivity and system-wide Resource Management on multi-core platforms”. In: SAMOS, 2016, pp. 26-33.

[MEL07] Meloni. P.; Loi, I.; Anglioni, F.; Carta, S.; Barbaro, M.; Raffo, L., Benini, L. “Area and power modeling for networks-on-chip with layout awareness”. VLSI Design, v.2007, Article ID 50285, 2007. 12 pages.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. In: Integration, the VLSI Journal, 2004, vol. 38(1), pp. 69-93.

[NAY07] Nayebi, A.; Meraji, S.; Shamaei, A.; Sarbazi-Azad, H. "XMulator: A Listener-Based Integrated Simulation Platform for Interconnection Networks". In: AMS, 2007, pp. 128-132.

[NGO06] Ngouanga, A.; Sassatelli, G.; Torres, L.; Gil, T.; Soares, A.; Susin, A. “A contextual re-sources use: a proof of concept through the APACHES platform”. In: DDECS, 2006, pp.42-47.

[OST12] Ost, L.; Varyani, S.; Mandelli, M; Almeida, G.; Indrusiak, L.; Wachter, E.; Moraes, F. G.; Sassatelli, G. “Enabling Adaptive Techniques in Heterogeneous MPSoCs based on Virtualization”. ACM Transactions on Reconfigurable Technology and Systems, v. 5(3), 2012, 12p.

[OST13] Ost, L. C.; Mandelli, M; Almeida, G. M.; Moller, L. S.; Indrusiak, L. S.; Sassatelli, G.; Benoit, P.; Glesner, M.; Robert, M.; Moraes, F. G. “Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach”. ACM Transactions on Embedded Computing Systems, v. 12(3), 2013, 22p.

[PAS08] Pasricha, S.; Dutt, N. “On-Chip Communication Architectures”, Morgan Kauffman, 2008, 544p.

[PAT11] Patterson, D.; Hennessy, J. “Computer Architecture: A Quantitative Approach”. Morgan Kaufmann, 2011, 856p.

[PLA11] Processador PLASMA. Capturado em: http://plasmacpu.no-ip.org:8080/, 2011.

[ROS07] Rosing, T.; Mihic, K.; Micheli, G. “Power and Reliability Management of SoCs”. IEEE Transaction on VLSI Systems, v. 15(4), 2017, pp 391-403.

[ROT11] Roth, C.; et al. “Modular Framework for Multi-level Multi-device MPSoC Simulation”. In: IPDPSW, 2011, 6p.

[RUA17] Ruaro, M.; Moraes, F. G. “Demystifying the Cost of Task Migration in Distributed Memory Many-Core Systems”. In: ISCAS. 2017. 4p.

[RUD14] Rudi, A.; Bartolini, A.; Lodi, A.; Benini, L. “Optimum: Thermal aware task allocation for heterogeneous many-core devices”. In: HPCS, 2014, pp. 82–87

[SCH10] Schranzhofer, A.; Chen, J.-J.; Thiele, L. "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms". IEEE Transactions on Industrial Informatics, v. 6(4), 2010, pp. 692-707.

[SEM06] Semenov, O.; Vassighi, A.; Sachdev, M. “Impact of self-heating effect on long-term reliability and performance degradation in CMOS circuits”. IEEE Transactions on Device and Materials Reliability, v. 6(1), 2006, pp. 17–27.

[SHA11] Shabbir, A.; Kumar, A.; Mesman, B.; Corporaal, H. “Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms”. In: SAMOS, 2011, pp. 132-139.

[SHA15] Shafique, M.; Gnad, D.; Garg, S.; Henkel, J. “Variability-aware dark silicon management in on-chip many-core systems”. In DATE, 2015, pp. 387-392.

[SIN09] Singh, A.K.; Jigang, W.; Prakash, A.; Srikanthan, T. "Efficient Heuristics for Minimizing Communication Overhead in NoC-based Heterogeneous MPSoC Platforms". In: RSP, 2009, pp. 55-60.

[SIN10] Singh, A.K.; Srikanthan, T.; Kumar, A.; Jigang, W. "Communication-aware heuristics for runtime task mapping on NoC-based MPSoC platforms". Journal of Systems Architecture, v. 56(7), 2010, pp. 242-255.

[SIN13] Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. “Mapping on multi/many-core systems: survey of current and emerging trends”. In: DAC, 2013, 10p.

[SKA04] Skadron, K.; Stan, M.; Sankaranarayanan, K.; Huang, W.; Velusamy, S.; Tarjan, D. “Temperature-aware microarchitecture: Modeling and Implementation”. ACM Transactions on Architecture and Code Optimization, v. 1(1), 2004, p. 94-125.

[SMI05] Smit, L.T.; Hurink, J.L.; Smit, G.J.M. "Runtime mapping of applications to a heterogeneous SoC". In: SoC, 2005, pp. 78-81

[SRI04] Srinivasan, J.; Adve, S. V.; Bose, P.; Rivers, J. A. "The case for lifetime reliability-aware microprocessors". In: ISCAS 2004, pp. 276-287.

[TIL16] Tilera Corporation, "Tile-GX Processor Family". Capturado em: http://www.tilera.com/products/processors/TILE-Gx_Family, 2016.

[TIW94] Tiwari, V.; Malik, S.; Wolfe, A. "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization". In: ICCAD, 1994, pp. 384-390.

[VIS00] Viswanath, R.; et al, “Thermal performance challenges from silicon to systems”. Intel Technology Journal, 2000.

[VOE97] Voeten J. P. M.; Van der Putten P. H. A., “Speciation of reactive hardware/software systems”. Ph.D. Dissertation, Eindhoven University of Technology, 1997, 459p.

[WAC11] Wachter, W. E. “Integração de Novos Processadores em Arquiteturas MPSoC: Um Estudo de Caso”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 86p.

[WAN13] Wanner, L.; Apte, C.; Balani, R.; Gupta, P.; Srivastava, M. “Hardware Variability-Aware Duty Cycling for Embedded Sensors”. IEEE Transactions on Very Large-Scale Integration Systems, v. 21(6), 2013, pp. 1000–1012.

[WAN14] Wang, Z.; Chen, C.; Sharma, P.; Chattopadhyay, A. “System-level reliability exploration framework for heterogeneous MPSoC”. In: GLSVLSI, 2014, pp. 9-14.

[WAR10] Ware, M.; Rajamani, K.; Floyd, M.; Brock, B.; Rubio, J. C.; Rawson, F.; Carter, J. B. “Architecting for power management: The IBM POWER7 approach”. In: HPCA, 2010, 11 p.

[WEI06] Wei Huang; et al. “HotSpot: a compact thermal modeling methodology for early-stage VLSI design”. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, v. 14(5), 2006, pp. 501-513.

[WEI11] Weichslgartner, A.; Wildermann, S.; Teich, J. “Dynamic decentralized mapping of tree-structured applications on NoC architectures”. In: NOCs, 2011, pp. 201-208.

[WIL09] Wildermann, S.; Ziermann, T.; Teich, J. "Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures". In: FPT, 2009, pp. 514 - 517.

[WIL15] Wildermann, S.; Ziermann, T.; Teich, J. "Design Methodology and Run-time Management for Predictable Many-Core Systems". In: ISORCW, 2015, 6p.

[WOL12] Wolf, M. “Computers as components: principles of embedded computing system design”. Morgan Kaufmann Publishers, 2012, 528p.

[WOS07] Woszezenki, C. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCS”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2007.

[WU02] Wu, E.; et al “Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides”. In: Solid-state Electronics Journal, v. 46(11), 2003, pp. 1787-1798.

[WU11] Wu, Y. K.; Sharifi, S.; Rosing, T. S. “Distributed thermal management for embedded heterogeneous MPSoCs with dedicated hardware accelerators”. In: ICCD, 2011, pp. 183–189.

[YE02] Ye, T.; Benini, L.; De Micheli, G. “Analysis of power consumption on switch fabrics in network routers”. In: DAC, 2002, pp. 524–529.

[YI14] Yi, J.; et al. “An Improved Thermal Model for Static Optimization of Application Mapping and Scheduling in Multiprocessor System-on-Chip”. In: ISVLSI, 2014, 6p.

[YUH11] Yu-Hsiang, K.; Yang, M; Artan, N.S.; Chao, H.J. “CNoC: High-Radix Clos Network-on-Chip”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 30(12), 2011, pp. 1897-1910.

[ZAN12] Zanini, F.; et al, “Online Thermal Control Methods for Multiprocessor Systems”. ACM Transactions on Design Automation of Electronic Systems, v. 18(1), 2012, 26p.

[ZHO16] Zhou, J.; et al. “Thermal-Aware Task Scheduling for Energy Minimization in Heterogeneous Real-Time MPSoC Systems”. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 35(8), 2016, pp. 1269-1282.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br