Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

# Particionamento e Mapeamento de Aplicações em MPSoCs Baseados em NoCs 3D

**Marco Pokorski Stefani**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação na Pós-Graduação em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul

Orientador: Prof. Dr. César Augusto Missio Marcon

Porto Alegre, 2015

**Dados Internacionais de Catalogação na Publicação (CIP)**

S816p Stefani, Marco Pokorski

Particionamento e mapeamento de aplicações em MPSoCs baseados em NoCs 3D / Marco Pokorski Stefani. – Porto Alegre, 2015.
61 p.

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

1. Informática. 2. Arquitetura de Computador.
3. Microprocessadores. 4. Energia Elétrica - Consumo. I. Marcon, César Augusto Missio. II. Título.

CDD 004.35

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

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

# TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Particionamento e Mapeamento de Aplicações em MPSoCs Baseados em NoCs 3D" apresentada por Marco Pokorski Stefani como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 30/03/2015 pela Comissão Examinadora:

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

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

Prof. Dr. Eduardo Augusto Bezerra– — UFSC

Homologada em 18/06/2015, conforme Ata No. 010 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

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

# AGRADECIMENTOS

Primeiramente gostaria de agradecer profundamente ao meu orientador, Prof. Dr. César Marcon, pelo exemplo de ética e dedicação, do qual sem ele dificilmente teria ultrapassado os obstáculos enfrentados.

Agradeço aos meus colegas de mestrados, Ramon Fernandes e Rodrigo Cataldo por todo o auxílio e esforço para o desenvolvimento de artigos relacionados ao tema desta dissertação.

A Dra. Thais Webber, pela ajuda nas pesquisas e pelas contribuições e revisões de artigos.

Ao meu amigo Me. Vinícius Pessil Bohrer, pelo incentivo, apoio e dicas para que eu entrasse no mestrado.

Agradeço especialmente à minha esposa Paula Cidade Moreira Stefani, por me incentivar a continuar os meus estudos. Por abdicar de vários sábados, domingos e férias, para que eu pudesse desenvolver as atividades do mestrado bem como a escrita deste.

Aos meus familiares que sempre acreditaram em mim e confiaram nas minhas capacidades, incentivando a realizar este curso.

Sobre tudo, agradeço a Deus por estar sempre ao meu lado nos bons e maus momentos da vida.

# PARTICIONAMENTO E MAPEAMENTO DE APLICAÇÕES EM MPSOCS BASEADOS EM NOCS 3D

## RESUMO

Sistema multiprocessado intrachip, em inglês *Multiprocessor System-on-Chip* (MPSoC), com comunicação baseada em rede intrachip, em inglês *Network-on-Chip* (NoC), integra grande quantidade de Elementos de Processamento (PEs) com o objetivo de executar aplicações com alto grau de paralelismo/concorrência. Estas aplicações são compostas por diversas tarefas comunicantes, que são mapeadas dinamicamente nos PEs da arquitetura alvo. Quando cresce o número de tarefas da aplicação, a complexidade do mapeamento também cresce, podendo reduzir a eficácia e/ou a eficiência da solução encontrada. Uma abordagem para otimizar o mapeamento é a introdução de uma etapa anterior denominada particionamento, que permite organizar a interação das tarefas através de um agrupamento eficiente, reduzindo o número de alternativas do mapeamento.

Esta dissertação propõe o algoritmo *Partition Reduce* (PR), que é uma abordagem de particionamento de tarefas baseada no algoritmo MapReduce, onde as tarefas são particionadas através de um agrupamento iterativo determinístico. O PR foi analisado quanto a sua eficácia e eficiência para minimizar o consumo de energia causada pela comunicação na arquitetura alvo e para balancear a carga de processamento nos PEs.

Resultados experimentais, contendo um conjunto variado de complexidade de tarefas, demonstram que o PR é mais eficiente na geração de partições com baixo consumo de energia e com um balanceamento de carga eficiente para qualquer nível de complexidade de tarefas, quando comparado com o *Simulated Annealing* (SA). Por outro lado, os resultados mostram que o algoritmo é eficaz apenas para aplicações de média e alta complexidade.

**Palavras chave:** Algoritmo de particionamento; MPSoC baseado em NoC 3D; avaliação de desempenho; eficiência energética; balanceamento de carga.

# PARTITION AND MAPPING OF APPLICATIONS TARGETING 3D NOC BASED MPSOC

## ABSTRACT

Multiprocessor System-on-Chip (MPSoC) based on Network-on-Chip (NoC) incorporates a lot of Processing Elements (PEs) in order to perform applications with high degree of parallelism/concurrence. These applications consist of several communicating tasks that are dynamically mapped into the PEs of the target architecture. When the number of application tasks grows, the complexity of mapping also grows, possibly reducing the effectiveness and/or efficiency of the solution. An approach for the mapping optimization is the introduction of a previous step called partitioning, which allows to organize the tasks interaction through an efficient grouping, reducing the number of mapping alternatives.

This paper proposes the Partition Reduce (PR) algorithm, which is a task partitioning approach inspired on MapReduce algorithm, where tasks are partitioned by a deterministic iterative clustering. The PR was analyzed according to its effectiveness and efficiency to minimize the energy consumption caused by the communication in the target architecture and to balance the processing load on the PEs.

Experimental results, containing a wide range of complex tasks, show that PR is more effective in generating partitions with low power consumption and efficient load balancing at any level of tasks complexity, when compared with the simulated annealing (SA) algorithm. Moreover, the results show that the algorithm is efficient only for medium or high complexity applications.

**Keywords**: Partitioning algorithm; NoC-3D based MPSoC; performance evaluation; energy efficiency; load balance.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS

| | |
|---|---|
| ACPM | Application Communication Pattern Model |
| APCG | Application Characterization Graph |
| ASIC | Application Specific Integrated Circuit |
| CAD | Computer Aided Design |
| CAFES | Communication Analysis for Embedded Systems |
| CDA | Communication Dependence Algorithm |
| CDCG | Communication Dependence and Computation Graph |
| CDCM | Communication Dependence and Computation Model |
| CDF | Control-Data Flow |
| CDFG | Control-Data Flow Graph |
| CDG | Communication Dependence Graph |
| CDL | Communication Dependence List |
| CDM | Communication Dependence Model |
| CF | Control Flow |
| CI | Circuito Integrado |
| CTG | Communication Task Graph |
| CTM | Communication Task Model |
| CWA | Communication Weighted Algorithm |
| CWG | Communication Weighted Graph |
| CWM | Communication Weighted Model |
| ECWG | Extended Communication Weighted Graph |
| ECWM | Extended Communication Weighted Model |
| EMQ | Erro Médio Quadrático |
| FPGA | Field Programmable Gate Array |
| NoC | Network-on-Chip |
| PE | Processing Element |
| PR | Partition Reduce |
| SA | Simulated Annealing |
| SDL | Specification and Description Language |
| SoC | System-on-Chip |
| TSV | Through Silicon Via |

# SUMÁRIO

# 1 INTRODUÇÃO

As indústrias de semicondutores têm sido direcionadas pela demanda de aplicações que requerem muito desempenho com diversas funcionalidades novas. Para implementar tais aplicações, os dispositivos dos Circuitos Integrados (CIs) têm sido continuamente diminuídos num processo de escalamento, pois esta diminuição permite o incremento de funcionalidades com maior eficiência. O escalamento, por sua vez, conduziu a novas classes arquiteturais, tais como (i) o sistema intrachip, em inglês *System-on-Chip* (SoC), onde toda a funcionalidade do sistema está implementada em um único CI [MAR01]; e (ii) o multiprocessador intrachip, em inglês *Multiprocessor SoC* (MPSoC), que é uma especialização do SoC com múltiplos processadores, onde cada processador é definido como um elemento de processamento, em inglês *Processing Element* (PE) [JER05].

Entre os principais requisitos de projeto de CIs atuais estão a redução do consumo de energia e o aumento da capacidade de processamento. Estes requisitos têm direcionado diversas indústrias a projetar CIs tendo SoC como arquitetura alvo. Tal arquitetura permite que o mesmo CI possa realizar diferentes funcionalidades (e.g. temporização e amplificação) implementadas em um ou mais módulos de hardware e/ou software (e.g. processador e memória). A implementação de tais funcionalidades de forma integrada permite aumentar a velocidade de operação, e minimizar o consumo de energia e a dissipação de potência [WOL04].

A comunicação intrachip muitas vezes é implementada com redes compostas por fios dedicados ou barramentos. Estas redes são eficientes em sistemas com pequeno número de núcleos, não se aplicando a SoCs com muitos núcleos comunicantes, principalmente pela falta de escalabilidade. Para executar aplicações que exigem grande quantidade de processamento paralelo, diversas topologias de redes intrachip 2D, em inglês 2D *Networks-on-Chip* (NoCs), principalmente as que implementam roteadores em sua arquitetura, têm sido amplamente pesquisadas [SVA07]. O motivo principal é o alto grau de paralelismo na comunicação, onde diversas conexões, em inglês, *links*, podem operar com diferentes pacotes de dados, provendo assim maior taxa de transferência de dados em comparação a outras topologias de comunicação como ponto-a-ponto e barramento compartilhado. Além disto, NoCs 2D podem ser construídas com topologias escaláveis, facilitando o projeto de SoCs ou MPSoCs com centenas de módulos.

Uma evolução arquitetural da NoC 2D é o seu projeto com uma terceira dimensão. NoCs projetadas para se comunicar em 3 dimensões são chamadas NoCs 3D. A tecnologia de NoC 3D possibilita a redução do consumo de energia devido à esta requerer fios globais

mais curtos (e.g., alimentação e relógio) em relação a uma NoC 2D equivalente. Além disto, NoCs 3D permitem implementar circuitos com maior proteção a ruído, e quando colocado matrizes de memória em planos subjacentes aos planos de processamento, resulta em ganho na comunicação entre memória e processador [AWT06].

Segundo estudos do ITRS [ITRS14], nos próximos 10 (dez) anos será possível a integração de bilhões de transistores e centenas de núcleos em uma mesma pastilha de silício. Para uma integração destes bilhões de transistores, que permitem construir SoCs com centenas a milhares de PEs, torna-se imperativo uma abordagem minuciosa de vários fatores a ser levados em consideração para melhor utilização destes núcleos. Entre estas abordagens estão as atividades de particionamento e mapeamento que são os objetivos principais deste trabalho.

O agrupamento de tarefas de uma aplicação é definido como particionamento, enquanto que o mapeamento é dado pela associação de grupo de tarefas em PEs ou *tiles* de um MPSoC. O mapeamento de uma aplicação (i.e., composta por tarefas, grupo de tarefas ou de núcleos), tendo como alvo uma arquitetura de comunicação do tipo NoC, pode contribuir significativamente para aumentar a vazão e reduzir a latência média das comunicações, uma vez que NoCs tendem a ter alta taxa de comunicação entre os núcleos.

Uma aplicação pode ser decomposta em tarefas, onde cada tarefa é definida como um conjunto de instruções e informações de dados relacionados. As tarefas podem ser realizadas de forma independente ou dependente uma da outra para pré-processar os dados e, em seguida, realizar a troca de dados durante a sua execução em um PE [MAN11]. Quando as tarefas são mapeadas para o mesmo PE, a troca de dados é realizada localmente por compartilhamento de memória. No entanto, se as tarefas são mapeadas em diferentes PEs, a comunicação implicará em troca de mensagens. Além disso, o conjunto de tarefas mapeadas em um mesmo PE pode ser planejado em tempo de projeto, a fim de garantir o mapeamento eficiente em tempo de execução. Este agrupamento de tarefas é denominado particionamento.

O problema de particionamento é NP-completo, o que impede a busca exaustiva de soluções. Assim, os algoritmos com diferentes características exploratórios são usados, principalmente a heurística *Simulated Annealing* (SA). No entanto, a quantidade de dados a serem explorados impede que este tipo de algoritmo heurístico possa ser executado de maneira eficiente, consequentemente, reduzindo a qualidade do resultado. Assim, diversas aplicações recentes de média e alta complexidade requerem o desenvolvimento de uma abordagem de particionamento mais eficiente. Este trabalho propõe o algoritmo *Partition Reduce* (PR), que explora os princípios do *MapReduce* (algoritmo amplamente utilizados

em aplicações distribuídas) [DEA04] no contexto de particionamento de tarefas e mapeamento de grupo de tarefas em tiles de MPSoCs baseado em NoCs. Esta dissertação compara SA com PR usando o consumo de energia e o balanceamento de carga como funções de custo.

## 1.1 MOTIVAÇÃO

Muitos trabalhos de particionamento de tarefas em NoCs levam em consideração a otimização de apenas um requisito de projeto, como por exemplo a minimização da latência das mensagens ou a redução do aquecimento dos processadores. Neste trabalho temos como motivação a exploração de algoritmos onde o projetista possa mapear tarefas ou grupo de tarefas considerando o efeito de um ou mais requisitos de projeto, ponderando características de aplicações que permitem modelar as mesmas de forma a agrupar tarefas para minimizar a função objetivo – aquela que calcula o custo de cada solução em relação aos requisitos de projeto. Este modelo de minimização premia questões de afinidades entre tarefas que permitem aos algoritmos explorar particionamentos e mapeamentos com um número menor de soluções possíveis, reduzindo a complexidade do problema e consequentemente reduzindo o tempo computacional, e favorecendo que sejam obtidas soluções mais próximas de um ótimo. A necessidade destes modelos fica ainda mais evidente com a grande quantidade de módulos e o massivo aumento da comunicação das novas aplicações, onde o particionamento pode permitir que aplicações sejam executadas, por exemplo, no menor tempo possível, reduzindo a latência da comunicação e/ou poupando energia.

## 1.2 OBJETIVOS

Este trabalho possui como principal objetivo a proposição de um algoritmo eficiente para particionamento de tarefas com foco no mapeamento destas em arquiteturas MPSoC (2D/3D) baseadas em NoC. Além disto, um segundo objetivo e contribuição do trabalho é evoluir as ferramentas PALOMA [ANT11] e CAFES [MAR10] para darem suporte ao particionamento e mapeamento de NoCs 3D, visto que as versões anteriores destas ferramentas apenas davam suporte a NoCs 2D. Visando estes objetivos, o trabalho possui os seguintes itens estratégicos:

- Dominar tecnologias de redes de comunicação intrachip 3D (NoCs 3D);
- Dominar arquiteturas de processamento paralelo intrachip 3D (MPSoCs 3D);

- Dominar técnicas de mapeamento e particionamento tendo como alvo arquiteturas MPSoCs baseadas em NoCs 3D;
- Analisar modelos para descrição de aplicações paralelas, que sirvam como entrada para o particionamento;
- Entender fluxos de projeto de sistemas computacionais, bem como, as principais atividades envolvidas.

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

- Explorar trabalhos relacionados, valorizando aqueles que modelam aplicações e redes de comunicações do tipo NoC 3D;
- Analisar e compreender modelos de aplicações utilizadas em outros trabalhos relacionados;
- Refinar a ferramenta de particionamento de tarefas no *framework* PALOMA [ANT11] para particionar uma aplicação em 3D;
- Refinar a modelagem da aplicação na ferramenta de mapeamento de tarefas no *framework* CAFES [MAR10] para modelar em 3D;
- Desenvolver algoritmo de particionamento que permita o projetista tratar múltiplos requisitos de projeto;
- Analisar os algoritmos explorados compreendendo vantagens e limitações de cada um, bem como analisar classes de aplicações adequadas para cada algoritmo.

## 1.3 ORGANIZAÇÃO DO DOCUMENTO

Além deste capítulo introdutório, este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o referencial teórico que fundamenta as atividades de particionamento e mapeamento, bem como a arquitetura base utilizada para o desenvolvimento deste trabalho. O Capítulo 3 apresenta trabalhos relacionados, comparando-os com o aqui proposto. Os Capítulos 4 e 5 descrevem alterações que foram feitas nos frameworks PALOMA e CAFES, respectivamente, para dar suporte à arquitetura de NoC 3D. O Capítulo 6 apresenta a metodologia proposta para particionamento de tarefas em grupos de tarefas e mapeamento de tarefas ou grupo de tarefas em tiles de MPSoCs baseados em NoCs 3D. O Capítulo 7 discute resultados experimentais realizados com os frameworks PALOMA e CAFES para as atividades de particionamento e mapeamento. O

Capítulo 8 descreve as principais conclusões e o Capítulo 9 apresenta a bibliografia referenciada neste trabalho.

# 2 REFERENCIAL TEÓRICO

## 2.1 MODELOS DE ARQUITETURA ALVO

NoC é uma estrutura de comunicação composta por roteadores interligados por conexões físicas. Este trabalho utiliza grafo para representar a topologia de uma NoC, onde os roteadores são representados por vértices e as conexões são representadas por arestas.

As topologias de NoCs são regulares quando é possível definir um padrão de arranjo na estrutura dos roteadores, tal como ilustrado na Figura 1(a) e na Figura 1(b). Caso este padrão não possa ser identificado, a topologia é denominada irregular, como pode ser observado na Figura 1(c) [MAR05].

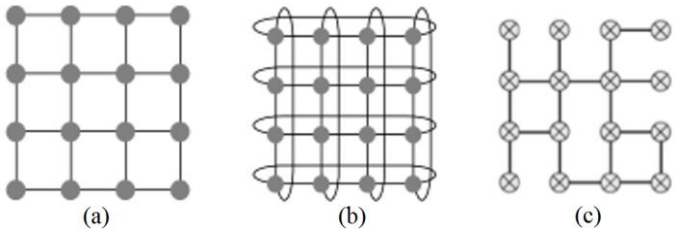



**Figura 1: Exemplo de topologias de rede: regular - (a) malha 2D, (b) toro 2D e (c) irregular ([MAR05]).**

A NoC é dita do tipo direta quando cada PE está conectado diretamente com um roteador apenas. Tanto para arquiteturas 2D, quanto para arquiteturas 3D, as NoCs possuem uma estrutura que inclui PEs, roteadores e conexões. A Figura 2 ilustra um exemplo de uma NoC 2D 2×2 e um exemplo de NoC 3D 2×2×2. Cabe salientar que algoritmicamente – com objetivo dos algoritmos de particionamento/mapeamento – uma NoC 2D pode ser vista como uma implementação de uma NoC 3D onde a terceira dimensão é unitária, gerando uma NoC 3D de apenas um plano.

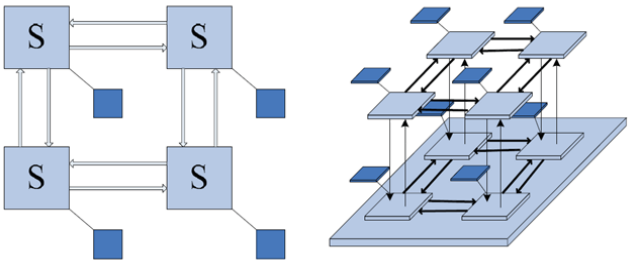



**Figura 2: Exemplo de NoCs. (a) NoC 2D (b) NoC 3D ([CAI10]).**

## 2.2 DEFINIÇÃO DO PARTICIONAMENTO E DO MAPEAMENTO

Particionamento, neste trabalho, é definido como a atividade de agrupar elementos de um conjunto *A*, gerando um novo conjunto *B*, onde cada elemento de *B* é um conjunto

composto por um ou mais elementos de *A*, tal que |*A*| ≥ |*B*|. O particionamento segue um conjunto de regras que definem sua função objetivo.

Aplicações podem ser descritas como um conjunto de tarefas e suas inter-relações. Assim, o particionamento de uma aplicação pode ser definido como o ato de agrupar tarefas conforme regras gerando uma nova descrição da aplicação composta por grupos de tarefas.

O mapeamento, por sua vez, é a atividade que associa elementos de um grupo *A,* a elementos de um grupo *B*. Conforme o tipo de mapeamento, cada elemento do grupo *B* tem associado um ou mais elementos do grupo *A*, mas um elemento de *A* não pode estar associado a mais de um elemento de *B*.

Para este trabalho, os mapeamentos a serem considerados são: (i) mapeamento de tarefas em processadores (ou mais genericamente, PEs); (ii) mapeamento de grupo de tarefas em PEs; (iii) mapeamento de PEs em tiles da arquitetura alvo.

O problema de mapeamento é encontrar associações que atendam requisitos de projeto, tal como a minimização de energia, respeitando as restrições impostas pelo projeto, tal como o limite máximo de área e o máximo de calor dissipável. Dado que podem existir mais de um requisito para o mesmo projeto, o privilégio para atender estes requisitos é normalmente dado por pesos em funções objetivo que permitem relacionar estes requisitos. A função objetivo de mapeamento considera restrições como delimitadores de uma associação, enquanto tentam atender aos requisitos, i.e., minimizar uma composição de custos gerados frente aos requisitos.

Etapas de particionamento e mapeamento são ilustradas na Figura 3. Sendo $T = \{t_1, t_2, ..., t_n\}$ o conjunto de tarefas, com cada tarefa *t* representada por um losango, e $G = \{g_1, g_2, ..., g_n\}$ o conjunto de grupo de tarefas, onde cada grupo de tarefas *g* é representado por um retângulo com cantos arredondados e $g \subseteq T$, então, o particionamento é a atividade que agrupa elementos de T gerando o conjunto *G*, com |*T*| ≥ |*G*|. Sendo $P = \{p_A, p_B, ..., p_n\}$ o conjunto de PEs, onde cada PE *p* é representado por um hexaedro e $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, *o* conjunto de tiles da arquitetura alvo, então define-se três tipos de mapeamento: o (i) *mapeamento de tarefas em PEs* que gera a associação de uma ou mais tarefas a cada um dos PEs, tal que |*T*| ≥ |*P*|; o (ii) *mapeamento de grupo de tarefas em PEs* que associa cada elemento do conjunto de tarefas a cada elemento do conjunto de PEs e |*T*| = |*P*|, ou se |*T*| ≤ |*P*|, então pelo menos um elemento de *P* não tem associação com elemento de *T*; o (iii) *mapeamento de PEs em tiles da arquitetura alvo* que associa cada elemento de *P* com um elemento de $\Gamma$ e $|\Gamma| = |P|$. Esta associação representa o posicionamento físico dos PEs na arquitetura alvo. Tanto particionamento, quanto ao mapeamento, são realizados levando

em consideração uma função custo, formada por um conjunto de regras. Este custo é planejado de forma a atender requisitos e restrições do projeto.

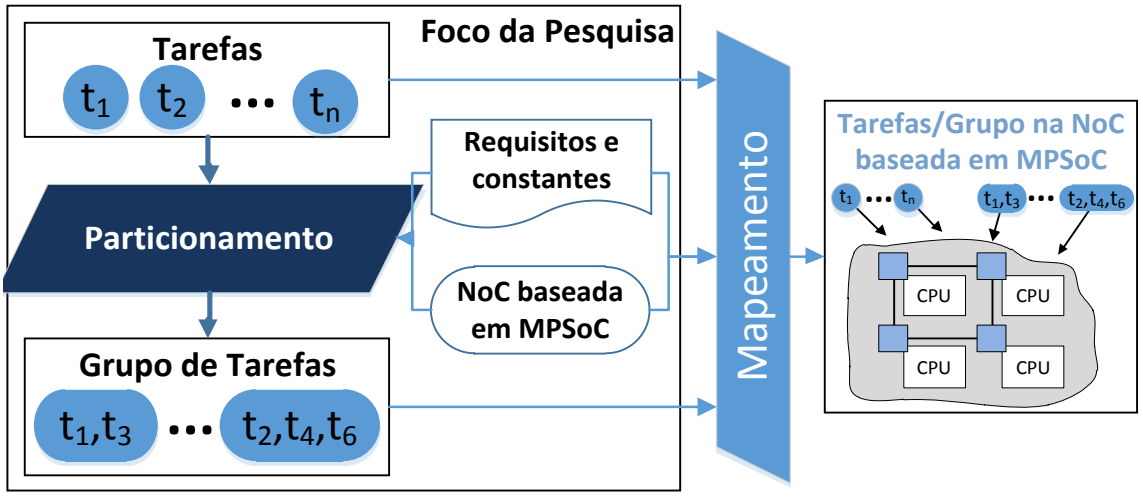



**Figura 3: Particionamento e mapeamento de uma aplicação composta por tarefas para um MPSoC.**

## 2.3 ALGORITMOS DE PARTICIONAMENTO E MAPEAMENTO

Tanto particionamento, quanto mapeamento, possuem natureza NP-completa, o que impede a busca por soluções exaustivas. Assim, outros tipos de algoritmos com diversas características exploratórias devem ser utilizados. Esta seção aborda os algoritmos clássicos como *Simulated Annealing (SA)*, *Taboo Search*, Genético e Kernighan-Lin que nem sempre encontram a melhor solução, mas normalmente encontram uma boa solução em tempo aceitável, ou seja, sacrificam a completude da solução para aumentar a eficiência e ainda assim encontrar resultados de alta qualidade.

O algoritmo SA é uma meta-heurística para otimização que consiste na técnica de busca local probabilística, e tem como base fundamental uma analogia com a termodinâmica [KIR83]. Esta analogia se dá a partir de um processo térmico, dito recozimento, utilizado em metalurgia para obtenção de estados de baixa energia num sólido. O processo consiste de duas etapas: (i) na primeira, a temperatura do sistema é aumentada, e (ii) na segunda, o sistema é resfriado lentamente, de forma que o material se organiza numa estrutura uniforme com mínimo gasto de energia, e reduzindo defeitos do material. De forma análoga, o algoritmo SA busca uma solução aproximada, escolhida de acordo com uma função e com uma variável de temperatura *T*, quanto maior *T*, maior a aleatoriedade, à medida que o algoritmo progride, o valor de *T* é decrementado, assim que o resultado começa a convergir, uma solução é alcançada. Uma das principais vantagens deste algoritmo é permitir o teste de soluções bem diferentes da solução avaliada por último e dar independência do ponto inicial da pesquisa.

O algoritmo *Taboo Search* (TS), ou Pesquisa Tabu, foi criado por Fred W. Glover, em 1986 e formalizado em 1989 [GLF89]. É um método de busca meta-heurística que emprega busca local utilizando otimizações matemáticas. O TS busca um vizinho para mover de forma iterativa de uma potencial solução *X* para uma solução melhorada *X'* na vizinhança de *X*, até que algum critério de parada seja satisfeito. Este algoritmo possui bom desempenho, utilizando-se de estruturas de memória que armazenam as soluções visitadas. Se uma solução potencial foi visitada anteriormente dentro de um determinado período ou se violou uma regra, ela é marcada na estrutura de memória como "*tabu*"(proibida), de modo que o algoritmo não considera essa possibilidade novamente.

O algoritmo genético, em inglês *Genetic Algorithm* (GA), é uma técnica de pesquisa para achar soluções aproximadas em problemas de otimização e busca [BEA93]. Fundamentado principalmente pelo americano John Henry Holland, GA faz parte de uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva com quatro operações principais: avaliação, mutação, seleção natural e recombinação (ou *crossing over*). Iniciando a partir de uma dada população, cada membro é avaliado de acordo com uma função objetivo. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados para formar uma nova população. A nova população, então, é utilizada como entrada para a próxima iteração do algoritmo. Algoritmos genéticos diferem dos algoritmos de busca tradicionais em basicamente quatro aspectos: (i) baseiam-se em uma codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si; (ii) os resultados são apresentados como uma população de soluções e não como uma solução única; (iii) não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado; e (iv) usam transições probabilísticas e não regras determinísticas.

O algoritmo Kernighan-Lin [HEL98] é uma heurística que propõe a geração de soluções ótimas ou quase ótimas para problemas de particionamento de grafos, sendo muito aplicado na geração de layout de circuitos digitais. A ideia básica do algoritmo é: “Partindo de um grafo G (V, E), onde V representa o conjunto de vértices e E o conjunto de arestas. O algoritmo tenta encontrar uma partição de V em dois subconjuntos disjuntos de igual tamanho A e B, tal que a soma dos pesos das arestas entre vértices de A e de B seja minimizada”. O algoritmo pode ser generalizado para várias aplicações, tal como a busca do caminho Euclidiano mais curto (i.e., o problema do caixeiro viajante). No entanto, o algoritmo e sua implementação não é trivial, cada decisão de implementação tem grande impacto no desempenho.

## 2.4 FRAMEWORK PALOMA

PALOMA (**P**artitioning **AL**g**O**rithm for **M**PSoC **A**utomated design) é um framework projetado para automatizar o particionamento, tendo como base três objetivos [ANT11]:

- Descrever classes de aplicações que embora sintéticas, podem ter comportamento similar ao de aplicações reais;
- Realizar testes de desempenho sobre diferentes abordagens algorítmicas empregadas no particionamento de tarefas;
- Analisar o desempenho de usar o particionamento como técnica de pré-mapeamento (i.e., particionamento de tarefas em grupos de tarefas seguido do mapeamento de grupos de tarefas em PEs) e comparar este desempenho com a abordagem de utilizar mapeamento direto (i.e., mapeamento de tarefas em PEs).

O PALOMA tem como entradas, a descrição da arquitetura alvo (i.e., quantidade e tipo de processadores, e tipo de NoC) e da aplicação (i.e., o grafo que representa a aplicação como um conjunto de tarefas comunicantes), a caracterização da aplicação (i.e., o efeito de executar tarefas da aplicação nos processadores da arquitetura alvo, tal como o consumo de energia causada pela execução de uma tarefa em um tipo de processador), e a escolha do algoritmo de particionamento. O resultado do particionamento é um grafo descrevendo a aplicação particionada e com os valores obtidos pela função objetivo de mapeamento.

## 2.5 FRAMEWORK CAFES

CAFES (***C**ommunication **A**nalysis **F**or **E**mbedded **S**ystems*) é um framework que integra um conjunto de modelos de computação usados no mapeamento de (i) PEs em tiles de uma NoC, e de (ii) tarefas em PEs da arquitetura alvo [MAR10]. O principal objetivo do CAFES é permitir que o projetista reduza o consumo de energia e latência da aplicação em um tempo de projeto aceitável.

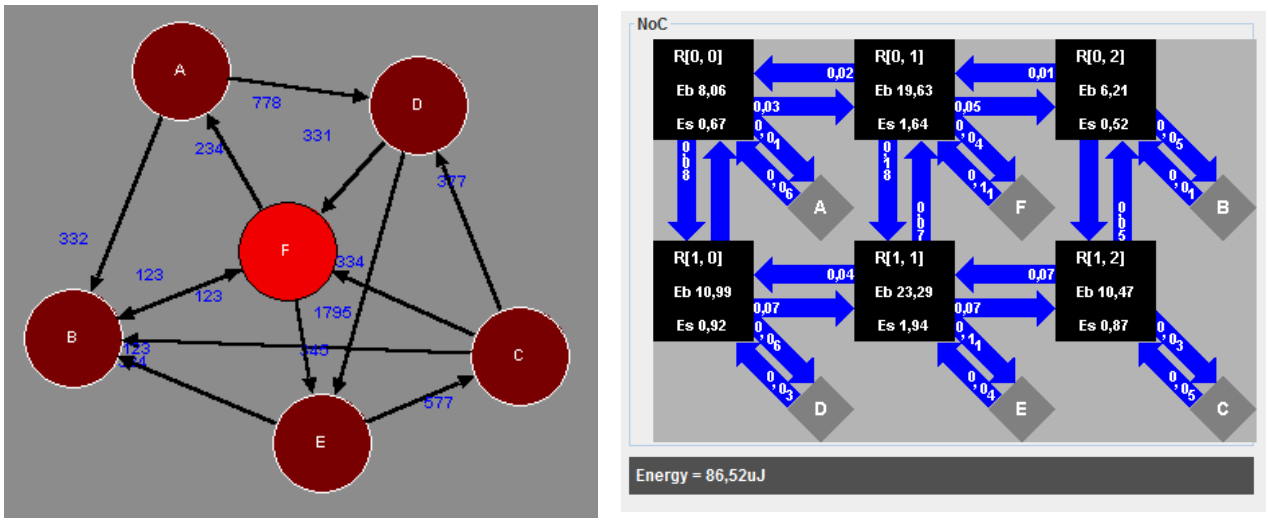
O CAFES é um programa desenvolvido em JAVA, assim permite que o mesmo seja executado em diversos sistemas operacionais, como Windows, Linux e iOS. O framework é *open-source,* permitindo que projetistas avaliem e possam evoluir o mesmo. O CAFES também é extensível, permitindo adicionar novos modelos de computação e de arquitetura alvo. O projetista pode escolher um modelo de aplicação e uma arquitetura alvo, e conforme os requisitos de projeto escolher minimizar o custo de energia gasto na comunicação das tarefas, a minimização da latência das mensagens trocadas entre PEs, entre outros.

### 2.5.1 Modelos de Aplicação

A ferramenta CAFES possui diversos modelos de aplicação, cada modelo com propósito especifico, como analisar aspectos de comunicação, requisitos e restrições do projeto, tempo de computação e consumo de memória.

#### 2.5.1.1 Communication Weight Model (CWM)

O CWM modela uma aplicação pela quantidade de comunicação que ocorre entre PEs ou entre tarefas. A comunicação entre PEs ou tarefas é dada pela soma de todos os bits de todos os pacotes transmitidos de um PE/tarefa para outro/outra. Com este modelo, os projetistas avaliam a qualidade de mapeamentos frente ao requisito de minimização do consumo de energia dinâmica. O modelo é definido por um grafo que descreve a aplicação através de seus/suas PEs/tarefas (i.e., vértices) e as quantidades de comunicação entre estes/estas (i.e., arestas). O mapeamento de PEs/tarefas da aplicação gera a aplicação mapeada em tiles/PEs da arquitetura alvo. A Figura 4 demonstra a entrada gráfica dos dados bem como o resultado gerado, considerando o mapeamento de seis PEs em seis tiles de uma NoC malha 2×3.



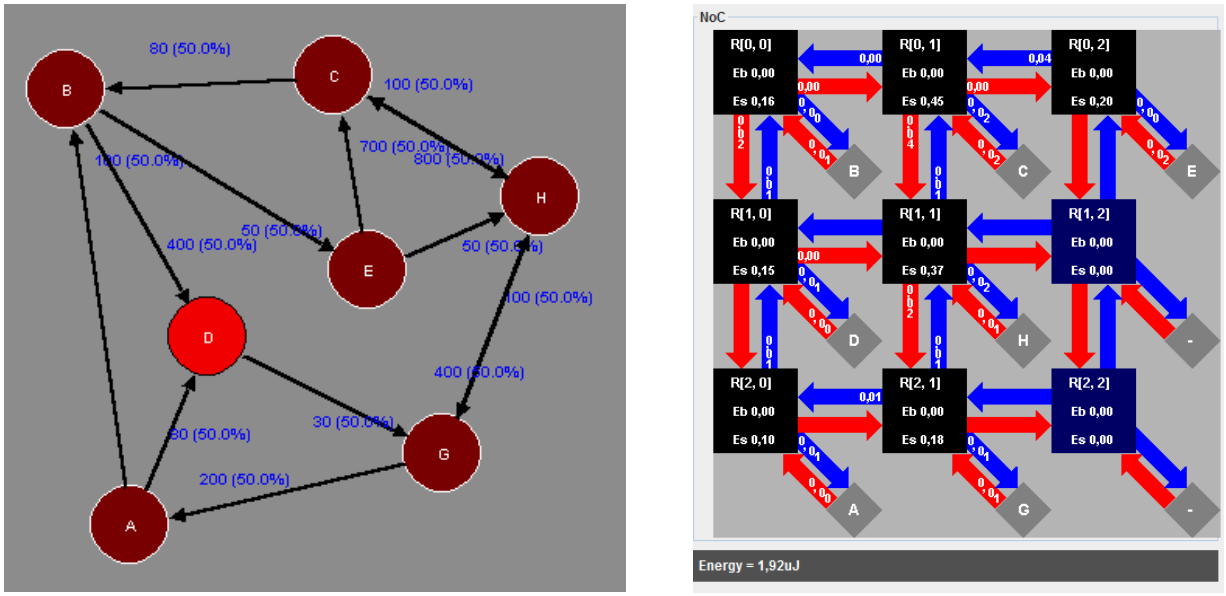

**(a)** **(b)**

**Figura 4: (a) Aplicação descrita por um CWM contendo 6 núcleos e suas comunicações; (b) correspondente mapeamento da aplicação em um NoC malha 2×3. No mapa, aparecem anotados em uJ (micro Joules) os valores de consumo de energia nos buffers (Eb), nos circuitos de chaveamento (Es), nos links (dentro das setas em azul) e o somatório do consumo de energia de todo o sistema.**

#### 2.5.1.2 Extended Communication Weight Model (ECWM)

O ECWM é um refinamento do CWM, acrescentando o número de transições em cada linha de cada canal de comunicação (i.e., a quantidade de bits consecutivos que inverte sua polaridade) como mais uma métrica que é computada junto ao volume de comunicação. Assim como o CWM, o ECWM é representado por um grafo, onde as arestas contêm ambos, o volume de comunicação e o percentual de transições. A Figura 5 ilustra

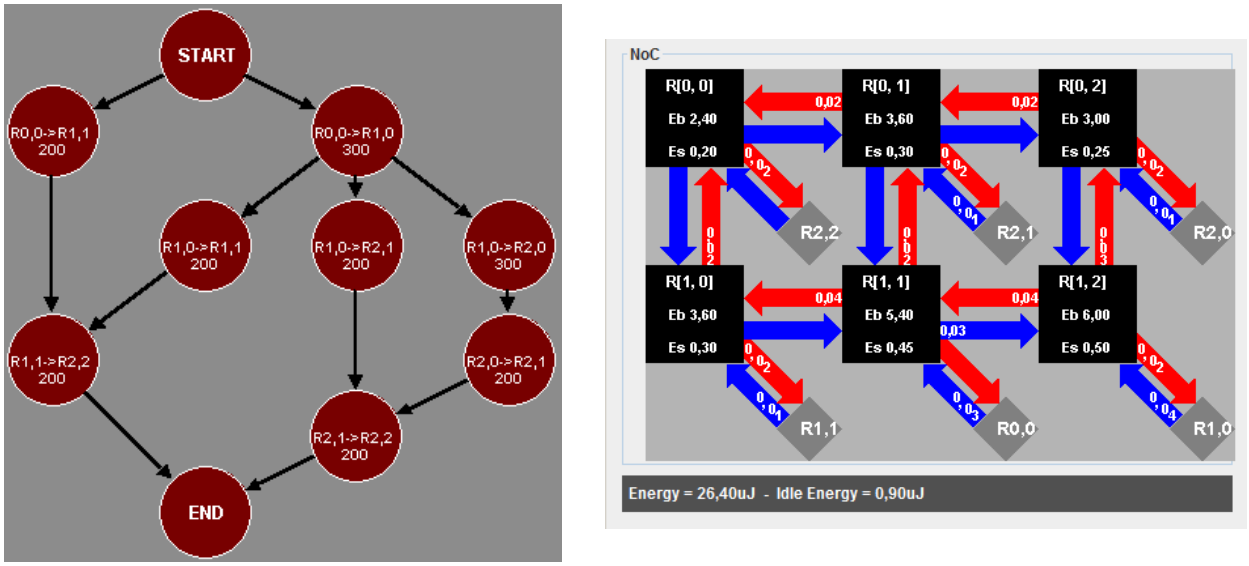
o mapeamento de 7 PEs em 9 tiles de uma NoC malha 2D 3×3, onde se observam dois tiles (marcados com '-') que não tiveram PEs associados.




**(a)** **(b)**

**Figura 5: (a) Aplicação descrita por um ECWM contendo 7 núcleos e suas comunicações; (b) correspondente mapeamento da aplicação em um NoC malha 3x3. No mapa, aparecem anotados em uJ (micro Joules) os valores de consumo de energia nos buffers (Eb), nos circuitos de chaveamento (Es), nos links (dentro das setas em azul) e o somatório do consumo de energia de todo o sistema.**

### 2.5.1.3 Communication Dependence Model (CDM)




**(a)** **(b)**

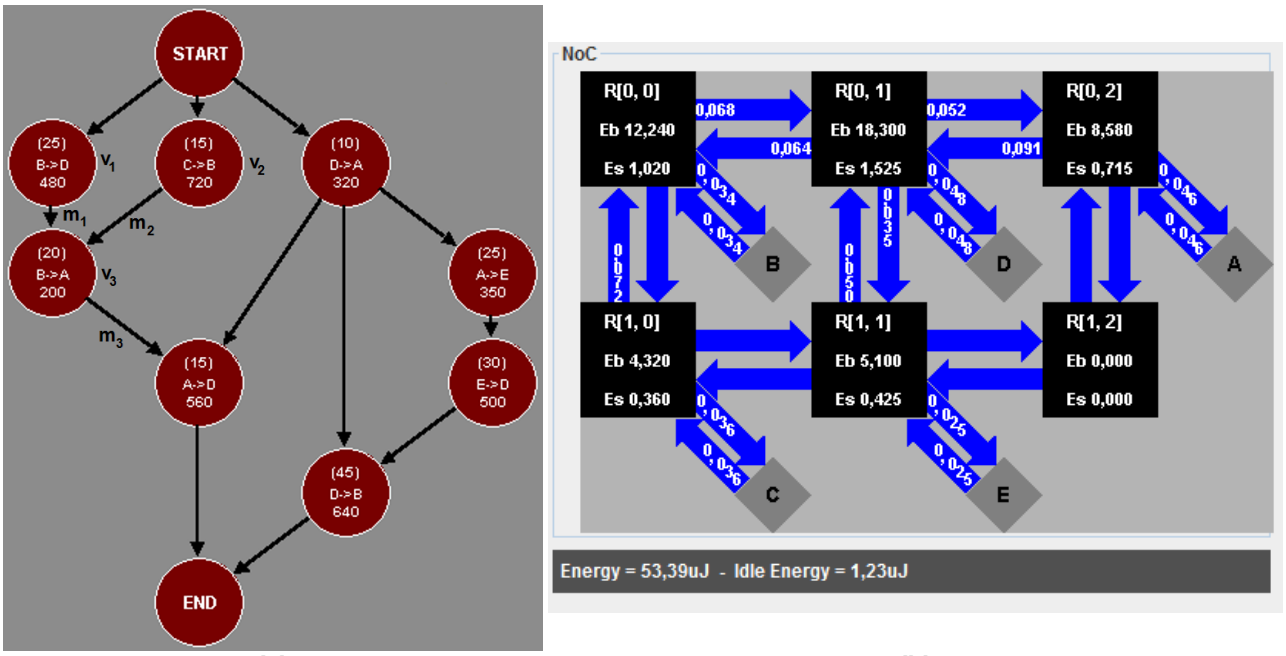
**Figura 6: (a) Aplicação descrita por um CDM contendo 8 comunicações entre PEs; (b) correspondente mapeamento da aplicação em uma NoC malha 2D 2×3. No mapa, aparecem anotados em uJ (micro Joules) os valores de consumo de energia nos buffers (Eb), nos circuitos de chaveamento (Es), nos links (dentro das setas em azul) e o somatório do consumo de energia de todo o sistema, bem como o somatório de energia quando o núcleo está inativo (Idle Energy).**

O CDM modela uma aplicação pela quantidade de dados trafegado na comunicação entre dois PEs/tarefas e pela dependência entre as mensagens. A informação de dependência permite prever quais mensagens podem concorrer pelo mesmo recurso,

de forma a explorar mapeamentos que reduzem esta concorrência. Isto elimina, ou pelo menos, reduz contenções de pacotes na rede de comunicação, consequentemente reduzindo o tempo de execução da aplicação e os consumos de energia estática e dinâmica de todo o circuito. A Figura 6 ilustra uma aplicação sintética descrita com CDM contendo 8 mensagens entre 6 PEs que é mapeada em uma NoC malha 2D 2×3.

#### 2.5.1.4 Communication Dependence and Computation Model (CDCM)

O CDCM é uma evolução do CDM, com o acréscimo do tempo de computação de cada PE. Este tempo é considerado a partir do momento que todas as dependências foram resolvidas até o momento que a mensagem foi disparada na rede. Por exemplo, na Figura 7, o vértice $v_3$ depende de duas arestas, i.e., as mensagens $m_1$ e $m_2$. Uma vez que ambas mensagens chegam ao PE B, este processa 20 unidades de tempo[1] para então lançar a mensagem $m_3$. O tempo de computação permite estimar o tempo de execução da aplicação de maneira mais precisa que o CDM, e consequentemente, as estimativas de consumo de energia estáticas são melhoradas. A Figura 7 exemplifica uma aplicação sintética contendo 5 PEs se comunicando através de 8 mensagens, sendo esta aplicação mapeada em uma NoC malha 2D 2x3.




**(a)** **(b)**

**Figura 7: (a) Aplicação descrita por um CDCM contendo 8 comunicações entre PEs e o processamento que precede a geração de cada comunicação; (b) correspondente mapeamento da aplicação em um NoC malha 2×3. No mapa, aparecem anotados em uJ (micro Joules) os valores de consumo de energia nos buffers (Eb), nos circuitos de chaveamento (Es), nos links (dentro das setas em azul), o somatório do consumo de energia de todo o sistema, bem como o somatório de energia quando os PEs estão inativos (i.e. Idle Energy).**

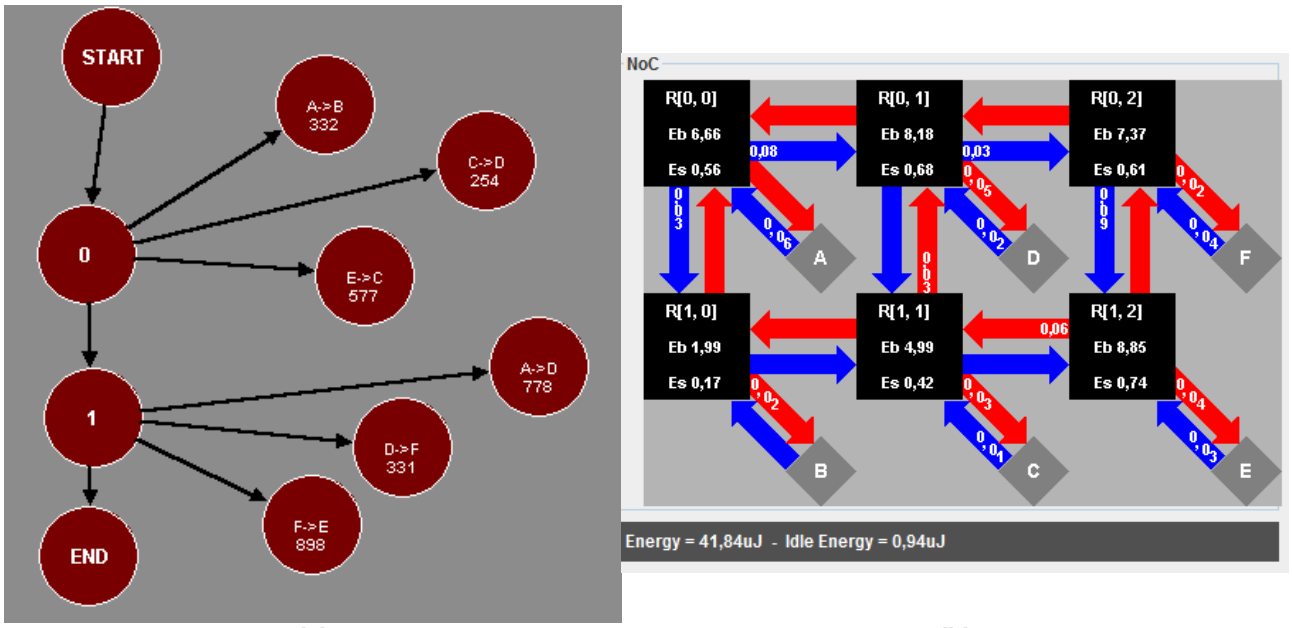
[1] Unidade de tempo é definida na ferramenta CAFES. Esta pode ser, por exemplo, nano segundos, segundos, ciclos de relógio, etc.

**Síntese do CDCM para VHDL**

A partir do mapeamento, tal como ilustrado na Figura 7 (b), é possível gerar um código em VHDL que provê estímulos individuais para os PEs, sendo cada PE uma tarefa ou um conjunto de tarefas. Cada PE é instanciado por uma linha de código mostrando a porta de comunicação, onde cada PE instanciado pode ser conectado a outro. Os estímulos para os PEs são gerados de acordo com a quantidade de dados definidos nas tarefas pelo grafo CDCM, tal como ilustrado na Figura 7 (a). Esta descrição é realizada de forma semiautomática com o auxílio da ferramenta CDCG2VHDL [MAR10].

### 2.5.1.5 Application Communication Pattern Model (ACPM)

O ACPM é um modelo de eventos discreto que descreve uma aplicação pela ordenação total de mensagens. A cada marca temporal é associado um conjunto de eventos (mensagens). Igualmente ao CWM, o ACPM é voltado para modelar grandes aplicações, com a vantagem de inserir mais detalhes temporais pela ordenação de eventos, gerando uma estimativa mais precisa que permite estimar o consumo de energia estática. A Figura 8(a) ilustra uma descrição de aplicação sintética com seis núcleos e 6 mensagens, sendo três geradas no instante 0 e 3 geradas no instante 1. A Figura 8(b) mostra um mapeamento para esta aplicação em uma NoC malha 2D 2×3.




**(a)** **(b)**

**Figura 8: (a) Aplicação descrita por um ACPM contendo 6 mensagens e 2 marcas temporais; (b) correspondente mapeamento da aplicação em um NoC malha 2×3. No mapa, aparecem anotados em uJ (micro Joules) os valores de consumo de energia nos buffers (Eb), nos circuitos de chaveamento (Es), nos links (dentro das setas em azul) e o somatório do consumo de energia de todo o sistema, bem como o somatório de energia quando o núcleo está inativo.**

# 3 TRABALHOS RELACIONADOS

O particionamento e mapeamento de aplicações tendo como alvo arquiteturas de comunicação do tipo NoC tem sido abordado por diversos pesquisadores [UMI05]. Este capítulo analisa trabalhos de particionamento de tarefas em grupo de tarefas, mapeamento de tarefas e grupo de tarefas em PEs e mapeamento de PEs em tiles, todos com foco em arquiteturas alvo baseadas em NoC 3D.

Primeiramente será analisado trabalhos relacionados a infraestruturas de comunicação do tipo NoCs 3D, dando preferência para as com topologia malha. O principal objetivo é analisar qual a técnica mais eficiente para particionar e mapear grande quantidade de tarefas.

Wadhwani et al. [WAD13] apresentam um algoritmo heurístico do tipo *branch-and-bound* para o mapeamento de tarefas em PEs de uma arquitetura de comunicação do tipo NoC malha 3D. Os resultados experimentais mostram que a heurística proposta economizou de 19% a 28% de energia dinâmica da comunicação em comparação a mapeamentos randômicos, e economizou de 42% a 55% quando comparado com mapeamentos em NoC 2D.

Sepúlveda et al. [SEP13] mostram que um MPSoC 3D é caracterizado pela integração de grande quantidade de componentes que executam ampla gama de tarefas em um único chip. No entanto, o aquecimento é um obstáculo para MPSoCs 3D, tornando o mapeamento uma tarefa crítica na construção de uma arquitetura baseada em NoC 3D, influenciando fortemente o desempenho do MPSoC. No trabalho deles, é proposto o uso de um algoritmo de rede do tipo genético com aproximação de Pareto e múltiplos requisitos, onde a minimização da latência e do consumo de energia são os principais requisitos, mas sendo limitados pela restrição de aquecimento. A abordagem proposta reduziu 73% o consumo de energia e 42% a latência quando comparada a outros trabalhos.

Siozios et al. [SIO10] expõem que o problema de comunicação em CIs tornou-se uma questão desafiante e apresentam um algoritmo de mapeamento focado no baixo consumo de energia em NoCs 3D. O algoritmo leva em consideração NoC 3D com diferentes voltagens de alimentação em cada camada e mapeia tarefas nas camadas considerando suas funcionalidades e requisitos. Os resultados experimentais mostraram que a economia de energia chega a 19%, sem qualquer aumento de área ou latência, em comparação com arquiteturas que utilizam apenas uma voltagem de alimentação.

Wang et al. [WAN11] descrevem que arquiteturas de NoCs 3D resolvem problemas como o comprimento do fio e a latência de pacotes que ocorrem com NoCs 2D. O trabalho

deles objetiva o mapeamento com requisito de minimizar latência na NoC 3D, para tanto, o algoritmo de mapeamento utiliza o princípio de Pareto, focando na redução do congestionamento de pacotes. A fim de avaliar o algoritmo proposto, os autores utilizaram como estudo de caso uma aplicação de decodificação de imagem (*Video Object Plane Decoder*). Quando comparado com mapeamentos aleatórios, os resultados mostraram que o algoritmo proposto pode melhorar a latência em 24,4% e 15,4% considerando uma NoC sem e com congestionamento, respectivamente.

Chen et al. [CHE10] relatam que *chips* multiprocessados abrem oportunidades para aplicações paralelas, que originalmente eram executadas em clusters, serem executadas em uma única máquina com muitos núcleos. As diferenças, como a hierarquia de memória e padrões de comunicação entre os clusters e plataformas de múltiplos núcleos levantam novos desafios para projetar e implementar um sistema eficiente. Eles propõem um modelo de desenvolvimento para programar clusters em larga escala chamado *MapReduce*, sendo uma alternativa promissora para aproveitar a plataforma de múltiplos núcleos. O artigo expõe que é mais eficiente processar pequenos pedaços de dados, do que processar um grande dado ao mesmo tempo em plataformas de múltiplos núcleos com memória compartilhada. Assim, o *MapReduce* se torna um modelo ideal, focado em processar pequenas fatias de dados. Adicionalmente, eles propõem um algoritmo de particionamento denominado *Tiling-MapReduce* (TMR) que particiona grandes tarefas em pequenas tarefas de maneira a utilizar eficientemente os recursos da arquitetura alvo. Os resultados experimentais obtiveram uma economia de até 85% de memória, causando menos *cache miss* e uso mais eficiente dos núcleos do processador, resultando aumento de velocidade que varia de 1,2 a 3,3 vezes.

A avaliação de modelos relacionados permite que sejam propostos algoritmos baseados em critérios comuns a todas as aplicações, com foco em minimização do consumo de energia baseado na comunicação (i.e., minimização do consumo de energia unificando tarefas altamente comunicantes) e computação eficiente (i.e., balanceamento de carga das tarefas ativas em cada processador). Tais modelos são discutidos a seguir.

Göhringer et al. [GOH10] relatam que é desafiante o particionamento e mapeamento de tarefas de maneira eficiente para aplicações em MPSoC. A implantação de hardware reconfigurável neste domínio insere mais um grau de dificuldade devido à adaptação no projeto do hardware em tempo de execução. Para explorar este grau de liberdade em MPSoCs, foi utilizado uma abordagem baseada em agrupamento hierárquico para o particionamento da aplicação e configuração em tempo de execução. Além disso, cada módulo da aplicação é particionado a cada processo de reconfiguração do hardware,

a fim de garantir o máximo de desempenho no PE local e, portanto, para o MPSoC em geral.

Zhen Tei el al. [YIN13] relatam que o mapeamento de PEs para uma topologia NoC é considerado um problema NP-Completo. Com o aumento do número de PEs, o mapeamento de aplicações em NoC é ainda mais desafiante. O trabalho propõe um algoritmo genético que incorpora o particionamento da rede com técnicas heurísticas para melhorar o mapeamento de aplicações na NoC. Os resultados experimentais de uma aplicação de decodificação de imagem (i.e., Video Object Plane Decoder) mostraram que o método proposto melhora 6% o custo de comunicação em relação ao algoritmo genético sem a heurística.

Hilbrich et al. [HIL11] descrevem que NoCs baseadas em processadores multicores podem, não só aumentar o desempenho do sistema, mas também permitir a integração de múltiplas funções em um único chip. Em sistemas críticos o particionamento de tarefas se faz importante a fim de evitar a propagação de falhas resultantes da utilização de recursos compartilhados. Para melhorar a utilização de recursos e tolerar falhas, o trabalho utiliza particionamento estático e dinâmico. Como a reconfiguração dinâmica e o particionamento flexível requerem migração de tarefas entre PEs por meio de um recurso compartilhado, a NoC pode se tornar imprevisível, portanto, empiricamente foi analisado uma variedade de transferências de tarefas entre PEs para garantir seu determinismo na reconfiguração.

Pinotti et al. [PIN12] propõem um algoritmo semelhante ao Kernighan-Lin para realizar o particionamento de tarefas. Os autores compararam o algoritmo proposto com o SA e o TS, tendo como requisito a minimização do consumo de energia. Resultados experimentais demonstraram a eficiência do algoritmo proposto com baixo custo computacional.

Marcon et al. [MAR14] descreve que o particionamento estático pode ser usado como uma atividade de pré-mapeamento estático para aumentar a eficiência no mapeamento dinâmico de tarefas. O trabalho propõe um algoritmo baseado na abordagem Kernighan-Lin para executar o particionamento estático, onde as tarefas são agrupadas usando processos de busca em largura e profundidade. Uma vez as tarefas tendo sido agrupadas, o mapeamento dinâmico segue a regra do particionamento estático. Assim, tarefas agrupadas são mapeadas no mesmo PE, resultado e menor espaço de busca em tempo de execução e melhor atendimento de requisitos de projeto. Os resultados experimentais mostram que a abordagem é capaz de alcançar partições equilibradas, com baixo consumo de energia.

Zhang, Hu e Jiang [ZHA14] propõem um algoritmo para resolver o problema de *starvation* de pequenas tarefas através de um escalonamento do tipo *Shortest Remaining Time* (STR). Os resultados experimentais mostram que este escalonamento, junto com o algoritmo Hadoop (uma implementação baseada no MapReduce), diminui o tempo de processamento de pequenas tarefas, sem afetar o desempenho das grandes tarefas.

A Tabela 1 relaciona os trabalhos acima descritos com o trabalho aqui proposto. Os aspectos considerados são: (i) arquitetura de comunicação; (ii) algoritmo básico que realiza a atividade particionamento e/ou mapeamento e o (iii) objetivo da atividade.

**Tabela 1: Resumo dos trabalhos relacionados.**

| Autores | Ano | Tipo MPSoC | NoC | Algoritmo | Objetivo |
|---|---|---|---|---|---|
| **Wadhwani et. al.** | 2013 | Homogêneo | Malha 3D | Heurístico/Branch and Bound | Minimizar energia dinâmica |
| **Sepúlveda et. al.** | 2013 | Heterogêneo | Genérica 3D[2] | Genético/Pareto | Minimizar latência e consumo de energia |
| **Siozios et. al.** | 2010 | Homogêneo | Malha 3D | Indefinido | Minimizar consumo de energia |
| **Wang et. al.** | 2011 | Homogêneo | Malha 3D | Genético/Pareto | Minimizar latência |
| **Chen et. al.** | 2010 | Indefinido | Indefinido | MapReduce | Reduzir congestionamento, consumo de energia, e latência |
| **Göhringer et al** | 2010 | Heterogêneo | Genérica 2D[2] | Hierárquico | Minimizar latência |
| **Zhen Tei el al.** | 2013 | Ambos | Malha 2D | Genético | Melhorar a comunicação e minimizar latência |
| **Hilbrich et al.** | 2011 | Heterogêneo | Malha 2D | Indefinido | Tolerar falhas e minimizar latência |
| **Pinotti et al.** | 2012 | Heterogêneo | Malha 2D | Kernighan-Lin | Minimizar consumo de energia |
| **Marcon et al.** | 2014 | Heterogêneo | Malha 2D | KL*depth e KL*width (Baseado no Kernighan-Lin) | Minimizar consumo de energia e balanceamento de carga |
| **Zhang et al.** | 2014 | Indefinido | Indefinido | Indefinido | Reduzir o tempo de execução |
| **Este trabalho** | **2014** | **Homogêneo** | **Malha 3D** | **Partition Reduce (baseado no MapReduce)** | **Minimizar o consumo de energia e balancear carga** |

2 Permite a construção de uma NoC genérica (e.g. Toro, Malha, ...).

# 4 ALTERAÇÕES NO FRAMEWORK PALOMA PARA DAR SUPORTE A MPSOC 3D E AO PARTITION REDUCE

O framework PALOMA tem como principal atividade particionar tarefas em grupos de tarefas considerando MPSoCs homogêneos ou heterogêneos baseados em NoCs 2D. Mas também permite automatizar a geração de aplicações paralelas sintéticas de forma a modelar várias classes de aplicações. Parte da contribuição deste trabalho está na extensão do PALOMA para permitir o particionamento tendo como foco MPSoCs 3D.

As alterações no PALOMA para dar suporte ao projeto 3D implicaram na inclusão da definição do número de camadas da NoC e do consumo de energia diferenciado que ocorre nas conexões entre planos – implementada normalmente com TSVs (*Through Silicon Vias*) [GUT01]. Estes parâmetros de energia foram denominados EtPhits e EtSPhits, representando a quantidade de phits consumidos em uma TSV com e sem transição de bits, respectivamente. As alterações algorítmicas compreendem o algoritmo de particionamento SA bem como as equações 1 a 7 que permitem estimar o consumo de energia médio da arquitetura de comunicação.

Equação 1: $$\mathrm{totalHops} = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}\sum_{z=0}^{Z-1}\sum_{i=0}^{X-1}\sum_{j=0}^{Y-1}\sum_{k=0}^{Z-1}(|x-i|+|y-j|+|z-k|)$$

Onde: $\mathrm{totalHops}$ contém a quantidade total de saltos de todas as comunicações de uma NoC 3D malha, considerando roteamento determinístico mínimo. Enquanto que X, Y e Z representam as dimensões da NoC, que aparecem como Lines, Columns e Layers na Figura 9, respectivamente.

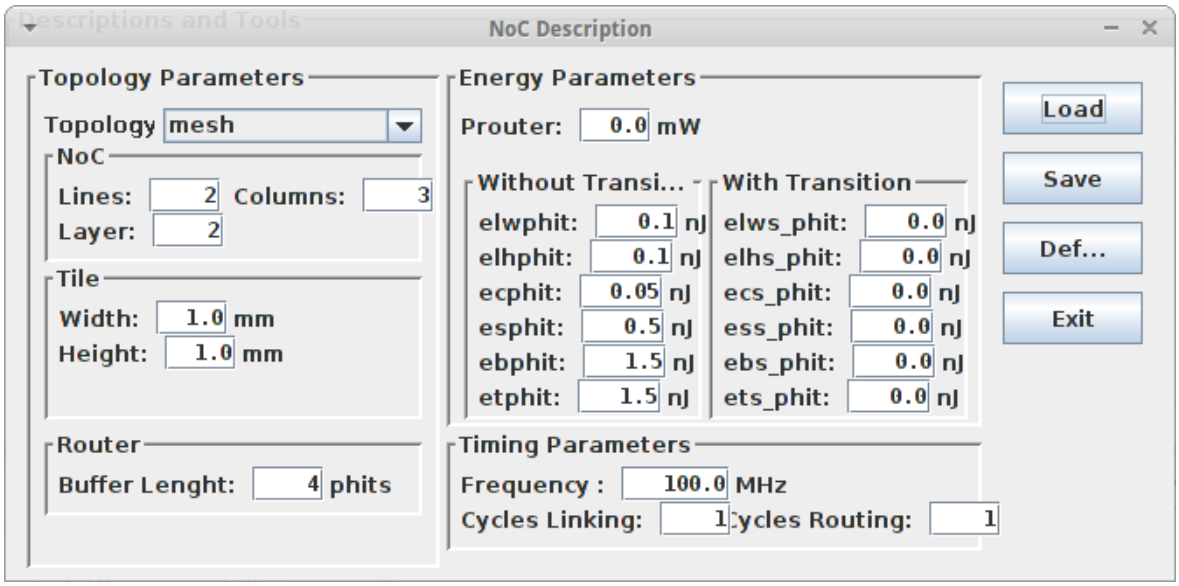



**Figura 9: Janela básica para parametrizar uma NoC 3D. O exemplo ilustra uma NoC malha 2×3×2.**

Equação 2: $\#Processadores = X \times Y \times Z$

Considerando que a arquitetura alvo é do tipo NoC direta (ou seja, cada roteador conecta diretamente um processador), o número de processadores ($\#Processadores$) é igual ao número de roteadores.

Equação 3: $\#MaxCom = \#Processadores \times (\#Processadores - 1)$

O número máximo de comunicações ($\#MaxCom$) para o modelo CWM (descrito na Seção 2.5.1.1) é dado pela comunicação de todos com todos, a exceção ao próprio processador que não envia mensagens para ele próprio.

Equação 4: $\eta = totalHops / \#MaxCom$

Onde $\eta$ representa o número médio de saltos por comunicação.

Equação 5: $E_LPhit = (elwphit + elhphit + etphit) / 3$

Onde: $E_LPhit$ é a média do o consumo de energia das conexões, sejam estas no plano ($elwphit$ e $elhphit$), sejam estas, entre planos ($etphit$). Estes parâmetros de energia são entradas que aparecem na Figura 9.

Equação 6: $EPhit = \eta \times (ebphit + esphit) + (\eta - 1) \times E_LPhit + 2 \times ecphit$

Onde: $EPhit$ representa o consumo de energia de um phit passando por um caminho com $\eta$ saltos de comunicação. Os parâmetros de $ebphit$ e $esphit$ permitem estimar o consumo de energia do armazenamento de um phit em um buffer e do chaveamento de um phit dentro de um roteador. Enquanto que o parâmetro $ecphit$ representa o consumo de energia de um phit passando por uma conexão local entre roteador e processador. Maiores detalhes do modelo podem ser encontrados em [ANT11].

Equação 7: $$ENoC = \sum_{n=1}^{COM} \sum_{i=1}^{COM(n)} EPhit_{ni}$$

Por fim, o consumo de energia da NoC é dado pelo somatório de todas as comunicações ($COM$), sendo $COM(n)$ a quantidade de phits da n-ésima comunicação. $ENoC$ permite estimar o consumo de energia médio, que uma dada partição gera frente a comunicação, antes de efetuar o mapeamento de tarefas/grupo de tarefas na NoC, ou seja, na atividade de particionamento.

A Figura 10 ilustra os parâmetros necessários para a geração de aplicações sintéticas no PALOMA.

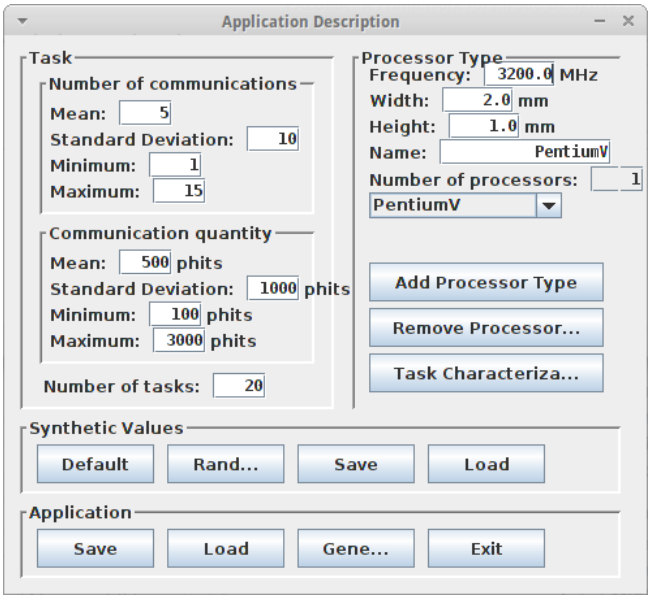



**Figura 10: Exemplo de descrição para geração de aplicação sintética e de uma arquitetura alvo.**

O gerador considera diversos parâmetros para geração de dados estatísticos, tais como média, desvio padrão e valores máximos e mínimos, tanto para o número de comunicações que parte de uma tarefa para outra, como da quantidade de phits que é transmitido em cada comunicação. O gerador pode considerar, também, mais de um tipo de processador, e conforme cada processador, pode existir uma correspondente caracterização de tarefas, contendo por exemplo, o consumo de energia da tarefa executando sobre o processador, e a área de código após compilar a tarefa.

A Figura 11 ilustra os requisitos, constantes e algoritmos que o projetista pode escolher para a geração do particionamento de aplicações sintéticas no PALOMA.

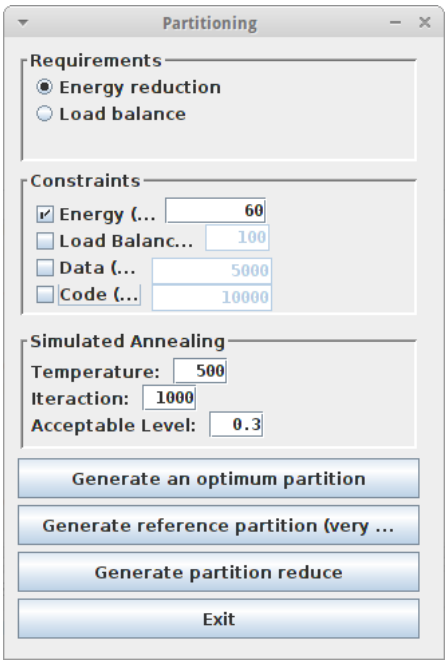



**Figura 11: Exemplo de requisitos, constantes e opção de algoritmo para particionar uma aplicação sintética.**

# 5 ALTERAÇÕES NO FRAMEWORK CAFES PARA DAR SUPORTE A NOC 3D

A ferramenta CAFES integra um conjunto de modelos de computação usados no mapeamento de PEs em *tiles* de NoCs com topologia 2D. Conforme apresentado e discutido no Capítulo 2, a inserção de uma terceira dimensão aumenta o desempenho da arquitetura de comunicação, permitindo dar melhor suporte para o projeto de aplicações mais complexas. Assim, afim de complementar o fluxo do projeto, parte da contribuição deste trabalho está na extensão do CAFES para permitir o mapeamento em NoCs 3D.

O framework CAFES possui diversos modelos de aplicação, cada modelo com um propósito especifico, como analisar aspectos de comunicação, requisitos e restrições do projeto, tempo de computação, consumo de memória e outros. Todos estes modelos são independentes da arquitetura alvo, podendo ser aplicados sobre uma NoC 3D.

As alterações básicas no CAFES para dar suporte ao projeto 3D, assim como no PALOMA, implicaram a inclusão dos parâmetros de energia EtPhits e EtSPhits (ver Capítulo 4). Adicionalmente, na visualização das informações foram adicionados os símbolos ʘ e ⊗ para representar o envio e o recebimento de flits, respectivamente.

As alterações algorítmicas realizadas abrangem os algoritmos de roteamento para as topologias de NoC toros e malha que passaram a ser XYZ ao invés de apenas XY, bem como os algoritmos que executam o mapeamento (i.e., TS, SA e Busca Exaustiva). Além disto, as fórmulas que computam o mapeamento, como o cálculo do consumo de energia, número mínimo de ciclos e consumo de energia do roteador inativo passaram a considerar matrizes 3D para representar as NoCs. Tais alterações se deram em todos os modelos de mapeamento disponíveis no CAFES (i.e., CWM, ECWM, CDCM, CDCM e ACPM) e em suas representações e gerações gráficas e textuais.

A Figura 12 e a Figura 13 ilustram o plano 0 e o plano 1 de uma representação gráfica de uma NoC 3D 2×3×2 com roteamento malha, onde foi mapeada uma aplicação sintética contendo 6 PEs (A, B, C, D, E, F) e diversas comunicações. A aplicação sintética está ilustrada na Figura 14, enquanto que os parâmetros da arquitetura alvo estão descritos na Figura 15.

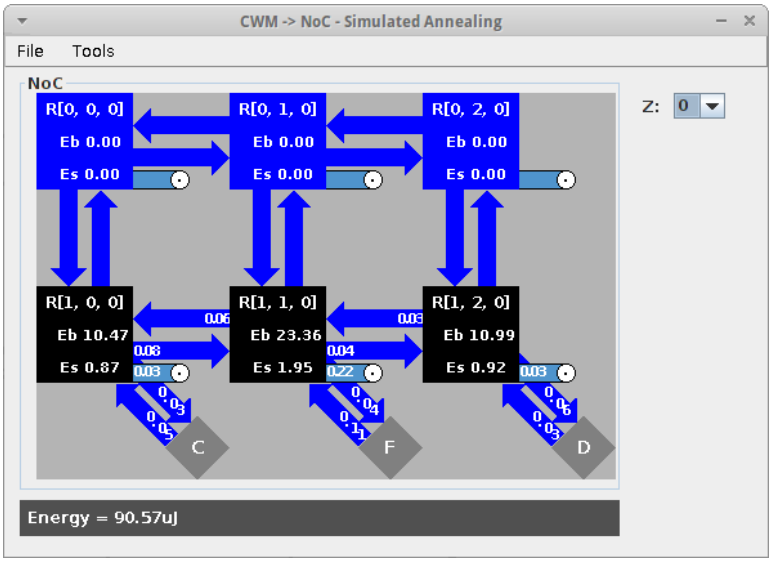



**Figura 12: Resultado do mapeamento visto pelo eixo Z no plano 0.**

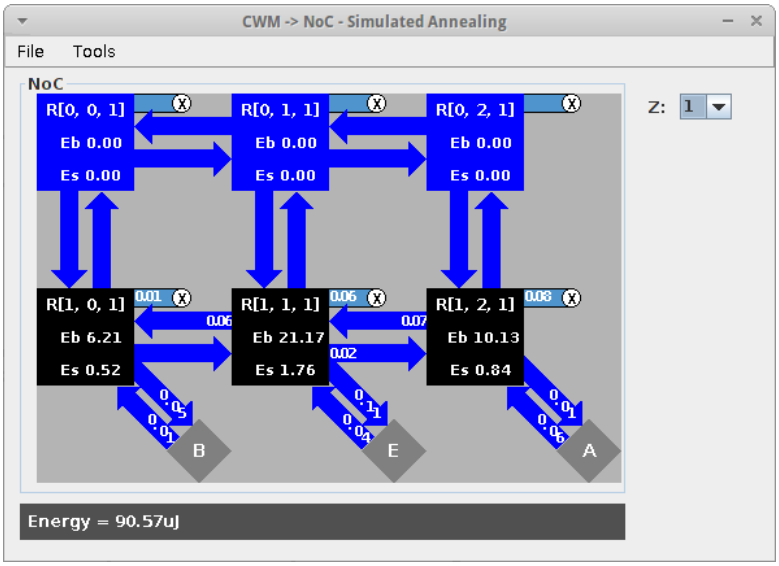



**Figura 13: Resultado do mapeamento visto pelo eixo Z no plano 1.**

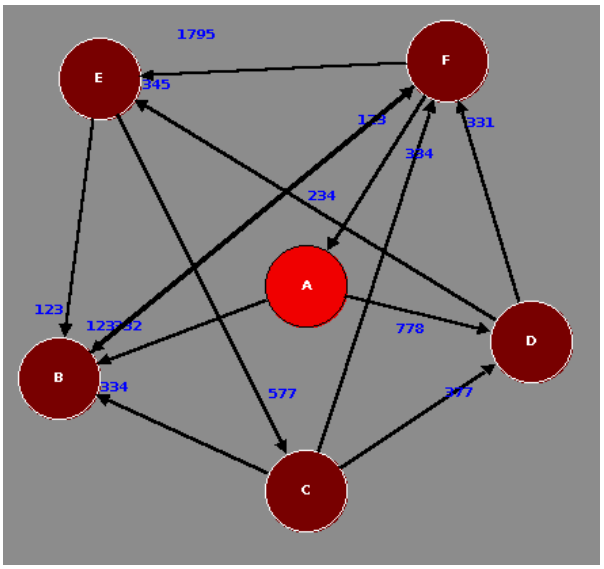



**Figura 14: Exemplo de uma aplicação sintética com 6 PEs modelada com CWM.**

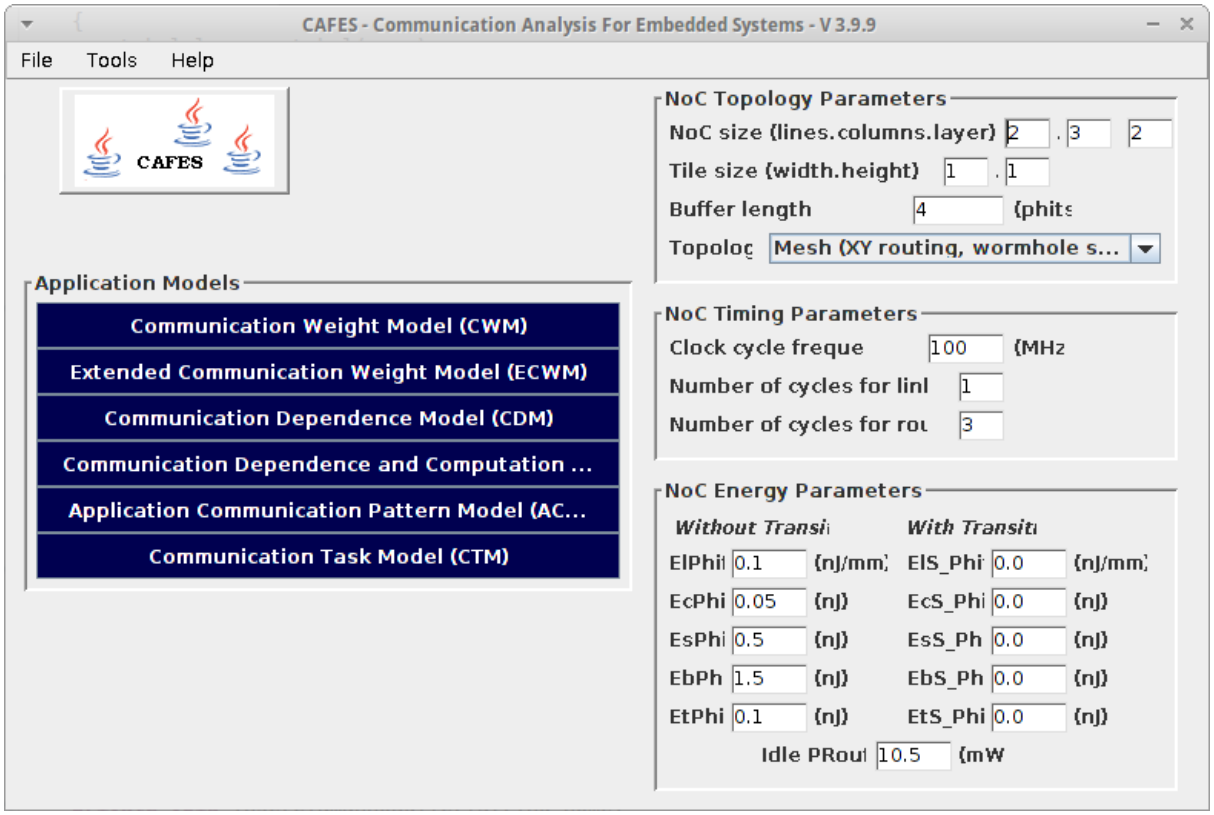



**Figura 15: Tela de parâmetros do framework CAFES.**

# 6 METODOLOGIA APLICADA AO PARTICIONAMENTO E MAPEAMENTO

Este capítulo descreve a metodologia aplicada no particionamento e mapeamento de aplicações, bem como um caso exemplo para compreensão desta metodologia.

## 6.1 DESCRIÇÃO DA METODOLOGIA

A Figura 16 ilustra a metodologia aplicada através de um fluxo de projeto, onde o particionamento de tarefas em processadores homogêneos tem como entradas: (i) o conjunto de tarefas da aplicação. Onde cada tarefa tem definido a ocupação da CPU para evitar a sobrecarga de processamento, o tamanho da tarefa em termos de dados e código, e a energia média gasta pela tarefa durante sua execução; (ii) a descrição da topologia da NoC contendo sua dimensão; (iii) a descrição da aplicação com a quantidade de dados trafegados de uma tarefa para outra; e (iv) a descrição das regras para o particionamento e mapeamento tal como a ocupação de CPU, quantidade de energia gasta, dissipação de calor ou a quantidade máxima de bits transmitida, e ainda estas regras contendo prioridade entre si.

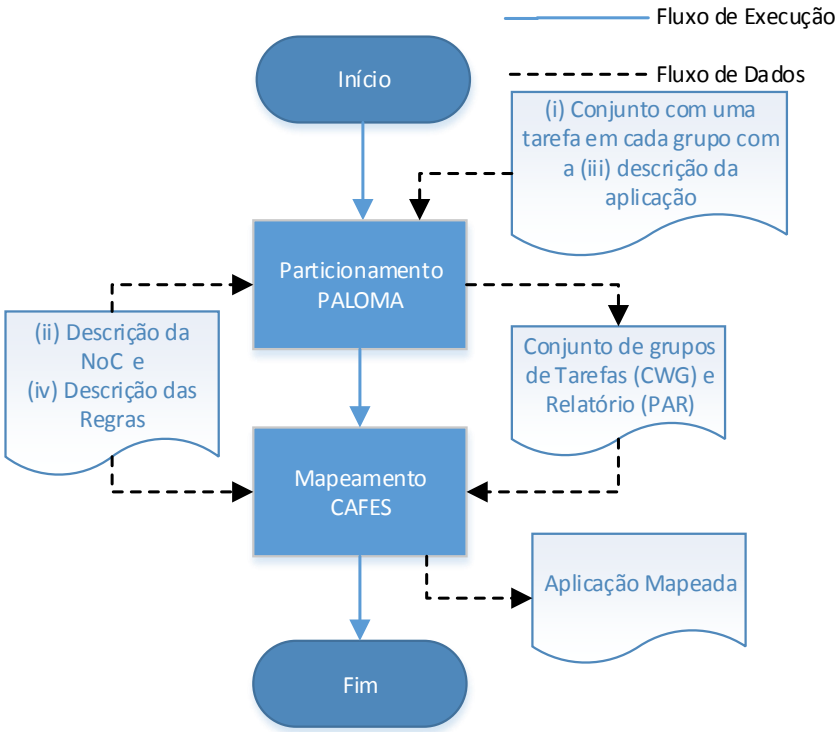



**Figura 16: Fluxo de projeto ilustrando as atividades de particionamento e mapeamento propostas.**

Com as informações acima é possível executar a atividade de particionamento de uma aplicação sintética, onde o projetista pode definir (i) requisitos que pretende explorar (e.g. redução do consumo de energia ou balanceamento de carga) e (ii) restrições (e.g.

limitar o número de tarefas agrupadas em um mesmo processador). Ao final do particionamento é gerado um conjunto de grupo de tarefas descrito em CWG que servirá como base de entrada para o mapeamento no CAFES e um relatório (PAR).

A interface do CAFES permite que o projetista execute o mapeamento dos grupos de tarefas, com as mesmas entradas descritas no particionamento (itens ii e iv) e a saída CWG. A saída do mapeamento (Aplicação mapeada) é um arquivo contendo as associações de tarefas/grupos de tarefas em processadores, objetivando minimizar o consumo de energia de todos os mapeamentos avaliados.

## 6.2 ALGORITMO PARTITION REDUCE (PR)

O algoritmo proposto, chamado Partition Reduce (PR), descreve a aplicação com um modelo semelhante ao CWM, denominado de Communication Weight Graph (CWG). O CWG = <T, W> é um grafo dirigido, onde T = {$t_1$, $t_2$, …, $t_n$} é o conjunto de tarefas da aplicação e W = {($t_a$, $t_b$, $w_{ab}$) | $t_a$, $t_b$ ∈ T, $w_{ab}$ ≠ 0} é o conjunto de todos os bits transmitidos de uma tarefa para outra. Este considera a quantidade de comunicação **unidirecional** entre cada par de tarefas, que é dada pela soma de todos os bits de todos os pacotes transmitidos através da arquitetura de comunicação. A Figura 17 ilustra um exemplo de um CWG contendo seis tarefas e nove comunicações.

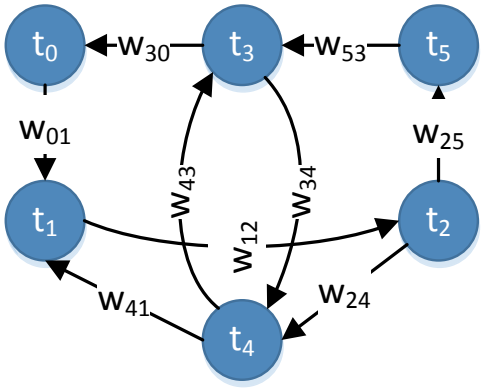



**Figura 17: Descrição de uma aplicação sintética em CWG. Vértices contêm a identificação da tarefa e arestas representam a quantidade de bits transmitidos em uma comunicação unidirecional.**

O PR usa um Grafo Interno (IG) para manipular a aplicação durante a atividade de particionamento. Seja $t_i$ = {$l_i$, $e_i$, $d_i$, $c_i$ | $t_i$ ∈ T} a i-ésima tarefa de T, onde $l_i$ é a carga de processamento necessário para a operação do $t_i$; $e_i$ é a energia consumida por $t_i$ quando executada no PE; e $d_i$ e $c_i$ são as áreas de dados e código de $t_i$ quando a tarefa $t_i$ é compilada para executar no PE. IG = {G, Z} é uma tupla, onde G representa o conjunto de todos os grupos de tarefas. Inicialmente, cada tarefa $t_i$ associada a um grupo unitário $g_i$, tal que |G| = |T|. Porém, durante a execução do PR, as tarefas são agrupadas reduzindo |G|; e Z é

o grafo contendo o agrupamento de tarefas formadas pelo PR. O agrupamento tem a mesma estrutura do CWG; todavia, cada vértice de Z contém um grupo de tarefas, e cada aresta é o conjunto de comunicações entre estes grupos.

### 6.2.1 Detalhamento do Algoritmo

O PR é baseado no algoritmo MapReduce [DEA04], que é largamente utilizado em sistemas distribuídos, uma vez que provê um modelo de programação eficiente para processamento e geração de grande quantidade de dados distribuídos, como por exemplo, construção de índices invertidos, tradução automática e aprendizado de máquina. O MapReduce é executado em duas etapas distintas. A primeira etapa é a de mapeamento, onde um conjunto de elementos é agrupado em tuplas gerando um novo conjunto. A segunda etapa é de redução, onde todas as tuplas são processadas para criar um novo e otimizado conjunto. Estas duas etapas são executadas repetidamente até que todos os conjuntos produzidos estejam completos, respeitando as restrições do projeto. A saída da etapa de redução é a entrada para a próxima etapa de mapeamento. O MapReduce reduz a quantidade de elementos do conjunto a cada ciclo devido à sua característica de agrupamento de tarefas. O algoritmo PR emprega uma abordagem similar, tendo em conta que a etapa de mapeamento é realmente um passo de particionamento, o que implica algumas diferenças que serão discutidas a seguir. A Figura 18 descreve o fluxo de particionamento do PR.

O PR tem como entradas: (i) a descrição da aplicação (ou seja, o CWG que representa a aplicação como um conjunto de tarefas de comunicação); (ii) a caracterização de cada aplicação tarefa no alvo PE (ou seja, o consumo de energia causado pelo tamanho da execução da tarefa, código e a carga de processamento necessário); (iii) a topologia da NoC contendo a quantidade de PEs; e (iv) o conjunto de requisitos e restrições do particionamento.

Diferentemente do algoritmo MapReduce, o PR não é executado de forma distribuída, mas usa o conceito de procurar e gerar associações de tarefas (ou seja, a etapa de mapeamento) e, assim, reduz o conjunto de tarefas original, que equivale a etapa de redução. O PR agrupa tarefas usando um processo cíclico. Cada ciclo é responsável pela seleção e agrupamento de tarefas, equivalente a etapa de mapeamento e redução. O processo de agrupamento de tarefas deve respeitar um conjunto de regras (ou restrições) definidas pelo projetista. O agrupamento é repetido até que todos os grupos de tarefas sejam avaliados, sendo a saída do particionamento um conjunto de grupos no formato

CWG. É importante notar que, estatisticamente, a cada novo ciclo o IG tende a computar mais rápido, pois |G| diminui conforme as tarefas são agrupadas.

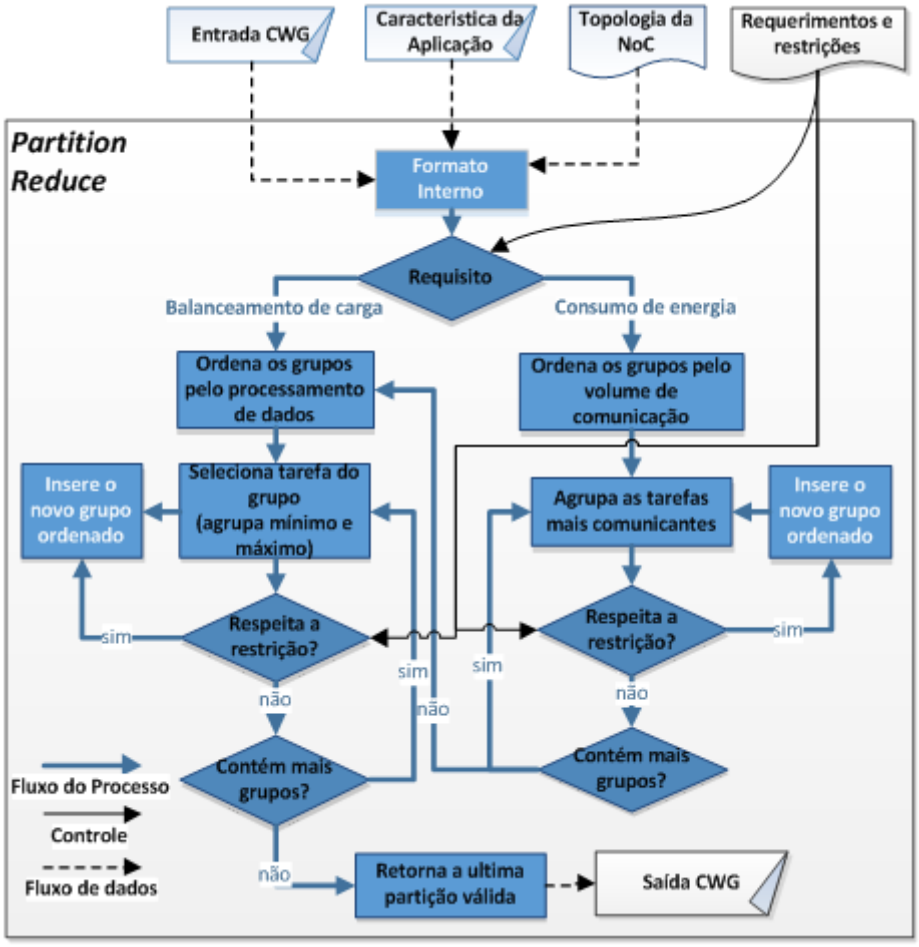



**Figura 18: Fluxo de particionamento do PR, demonstrando os dados de entrada agrupados no formato IG conforme o fluxo de dados e fluxo do processo que o dado IG segue na execução do PR bem como os caminhos alternativos de acordo com os requerimentos e restrições impostas pelo fluxo de controle.**

### 6.2.2 Consumo de Energia como Requisito

O particionamento, tendo como requisito reduzir o consumo de energia, separa os grupos de tarefas de acordo com o consumo de energia da comunicação entre grupos. Os dois grupos que consomem mais energia na comunicação entre si, são agrupados e processados pelo método da redução, que verifica se o agrupamento respeita as restrições determinadas pelo projeto. Se isto acontecer, a associação é efetivada; caso contrário, a associação é marcada como processada e ignorada. Este processo é executado repetidamente até que apenas um grupo de tarefas permaneça, ou não exista mais grupos de tarefas a serem processadas que respeitem as restrições. Ao final do particionamento é executado o balanceamento de carga como requisito, a fim de unificar grupos que não possuem comunicação entre si, visando reduzir a quantidade de PEs necessários e reduzindo assim o consumo de energia por PEs ativos.

O consumo de energia da comunicação é avaliado pelo $E_{Phit}$ (Equação 6), considerando uma NoC do tipo malha com *tiles* formando um cubo para computar a energia dinâmica consumida por um phit que atravessa a NoC do tile $\tau_i$, de coordenadas ($x_i$, $y_i$, $z_i$), para o tile $\tau_j$, de coordenadas ($x_j$, $y_j$, $z_j$). Neste caso, $\eta = (|x_j - x_i| + |y_j - y_i| + |z_j - z_i| + 1)$ é o número de roteadores por onde passa uma determinada comunicação.

### 6.2.3 Balanceamento de Carga como Requisito

O particionamento tendo como requisito o balanceamento de carga, ordena os grupos de tarefas de acordo com os maiores valores de carga de processamento. Inicialmente, cada tarefa é considerada um grupo. O grupo com maior carga de processamento é agrupado com o grupo de menor carga de processamento que respeite as restrições definidas (e.g. limite de carga por processador, limite de área de código). O agrupamento reduz a quantidade de grupos, reduzindo a complexidade dos próximos passos. O particionamento é executado repetidamente até que apenas um grupo de tarefas permaneça ou não haja mais grupo de tarefas a serem agrupados que respeitem as restrições.

## 6.3 Exemplificação da Metodologia usada com o Algoritmo Partition Reduce

Esta seção exemplifica a metodologia utilizada neste trabalho, apresentando a entrada de dados e a execução do algoritmo, descrevendo seu fluxo e exemplificando o particionamento de processadores. Este exemplo representa uma aplicação composta por 6 tarefas (*A, B, C, D, E, F*) ilustrada na Figura 19, onde deve ser feito o particionamento e mapeamento destas tarefas em processadores posicionados em tiles de uma NoC malha 3D com dimensão 2×3×1. O limite de processamento de cada tarefa respeita os limites de no máximo 60% e tem como requisito o balanceamento de carga. Convém salientar que em [STE15] este autor apresenta outro exemplo que auxilia na exemplificação do algoritmo.

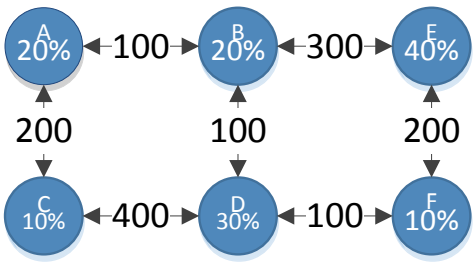



**Figura 19: Aplicação sintética descrita em um grafo. Esta é composta por seis tarefas, onde cada vértice contém o nome da tarefa e a carga de processamento (em percentual), quando a tarefa é executada sobre o PE da arquitetura alvo. Nos vértices está anotado o volume de comunicação bidirecional entre cada tarefa. A aplicação serve como entrada para o *framework* Paloma.**

O exemplo é especificado pelo arquivo de configuração, conforme ilustrado na Figura 20, sendo que o arquivo de configuração é dividido em três grupos: (i) *Target_Architecture* – tipos de processadores, com tamanho e frequência de operação e lista de todos os processadores; (ii) *Application_Caracterization* – contendo a caracterização de cada tarefa sobre cada processador (ver Seção 6.2); e (iii) *Application_Description* – contendo as relações entre as tarefas da aplicação.

```
<SYSTEM_SPECIFICATION>
    <TARGET_ARCHITECTURE>
        <PROCESSOR_TYPE_LIST>
            <PROCESSOR_TYPE type="PentiumV">
                <FEATURES frequency="800.0" width="1.0" height="1.0" depth="1.0" />
                <LIST> A B C D E F</LIST>
            </PROCESSOR_TYPE>
        </PROCESSOR_TYPE_LIST>
    </TARGET_ARCHITECTURE>
    <APPLICATION_CHARACTERIZATION>
        <TASK_LIST>
            <TASK id="A_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="20.0" /></TASK>
            <TASK id="B_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="20.0" /></TASK>
            <TASK id="C_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="10.0" /></TASK>
            <TASK id="D_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="30.0" /></TASK>
            <TASK id="E_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="40.0" /></TASK>
            <TASK id="F_T0"><PROCESSOR_TYPE type="PentiumV" power="18.6" data="609" code="456" cpuOccupation="10.0" /></TASK>
        </TASK_LIST>
    </APPLICATION_CHARACTERIZATION>
    <APPLICATION_DESCRIPTION>
        <COMMUNICATION_TASK_LIST>
            <SOURCE_TASK source="A_T0">
                <COMMUNICATION target="B_T0" volume="100" />
                <COMMUNICATION target="C_T0" volume="200" />
            </SOURCE_TASK>
            <SOURCE_TASK source="B_T0">
                <COMMUNICATION target="A_T0" volume="100" />
                <COMMUNICATION target="D_T0" volume="100" />
                <COMMUNICATION target="E_T0" volume="300" />
            </SOURCE_TASK>
            <SOURCE_TASK source="C_T0">
                <COMMUNICATION target="D_T0" volume="400" />
                <COMMUNICATION target="A_T0" volume="200" />
            </SOURCE_TASK>
            <SOURCE_TASK source="D_T0">
                <COMMUNICATION target="F_T0" volume="100" />
                <COMMUNICATION target="C_T0" volume="400" />
                <COMMUNICATION target="B_T0" volume="100" />
            </SOURCE_TASK>
            <SOURCE_TASK source="E_T0">
                <COMMUNICATION target="B_T0" volume="300" />
                <COMMUNICATION target="F_T0" volume="200" />
            </SOURCE_TASK>
            <SOURCE_TASK source="F_T0">
                <COMMUNICATION target="E_T0" volume="200" />
                <COMMUNICATION target="D_T0" volume="100" />
            </SOURCE_TASK>
        </COMMUNICATION_TASK_LIST>
    </APPLICATION_DESCRIPTION>
</SYSTEM_SPECIFICATION>
```

**Figura 20: Exemplo de descrição de entrada para o particionamento no PALOMA.**

No particionamento, o algoritmo com um conjunto de tarefas ou grupo de tarefas, busca a regra a ser executado em cada ciclo e encaminha estas informações para o processamento, sendo para esta exemplificação o balanceamento de carga no PE. O primeiro processamento de dados ordena todos os valores de acordo com a regra prioritária (i.e., é possível definir mais de um requisito para o particionamento) e agrupa as duas primeiras tarefas. Caso o agrupamento seja uma possibilidade válida retorna este conjunto

gerado senão busca a combinação da tarefa com a próxima tarefa (contendo os valores 20%, 30% e 40%, considerando a regra para o balanceamento de carga (i.e. agrupar maiores cargas de processamento com menores cargas). Para o exemplo, o algoritmo tenta agrupar os nodos com 20% e 40% de carga de processamento, caso este agrupamento não for válido, então parte para um agrupamento de menor carga de processamento (i.e. 20% e 30%). Caso não haja agrupamento que satisfaça a regra, retorna a entrada original que recebeu na entrada do algoritmo e encerra o particionamento. A etapa de particionamento é executada repetidas vezes enquanto encontrar partições que respeitem o conjunto de regras definido pelo projetista no arquivo de configuração.

A execução do algoritmo PR sobre a aplicação da Figura 19 gera um fluxo de dados descrito na Figura 21. Inicialmente, o algoritmo agrupa as tarefas E e F, gerando a partição (EF, A, B, C, D). Como o agrupamento EF respeita a primeira regra é executada a verificação dos demais requisitos e restrições, no caso, a regra da comunicação de dados. Uma vez respeitadas todas as regras, a nova entrada do algoritmo considera agora EF como um agrupamento válido, e este grupo pode receber outras tarefas.

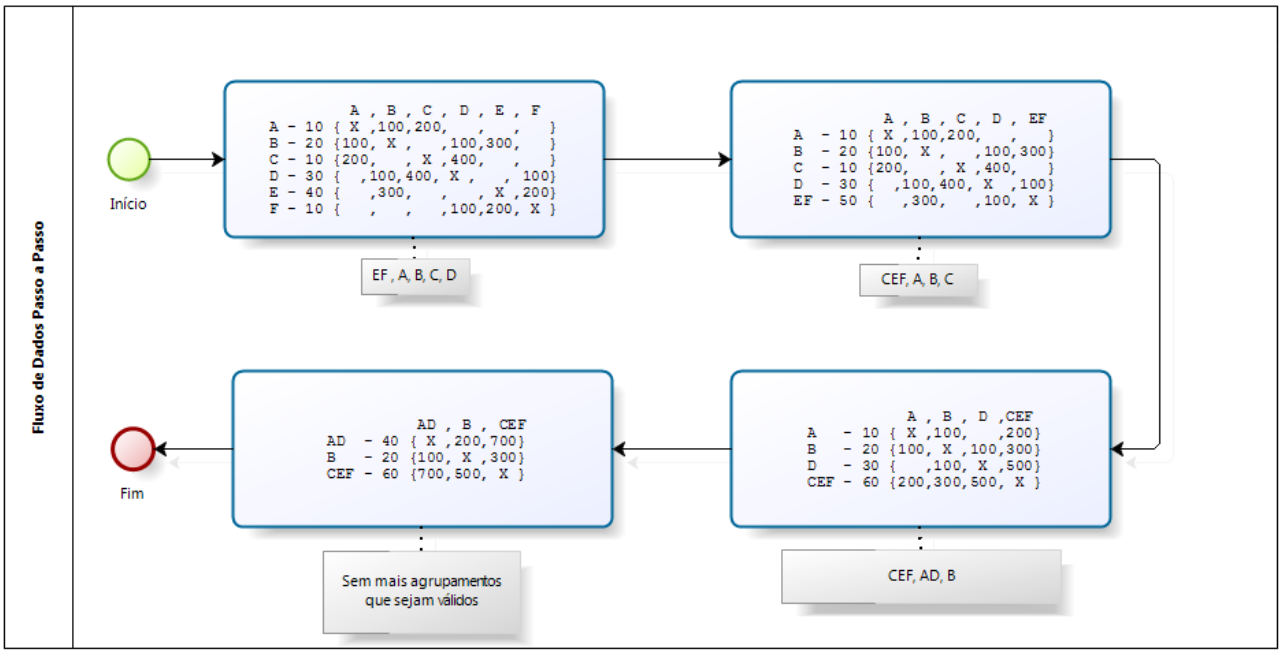



**Figura 21: Fluxo de dados do algoritmo PR para cada etapa da interação do particionamento.**

A consequência de agrupar E com F é gerar um novo grupo com mais carga ainda, de forma a este ser a opção para o próximo agrupamento. Agora existem duas opções de tarefas com 10% para agrupar com EF. A execução do algoritmo faz a escolha pelo último que encontrou, e desta forma agrupa EF com C, gerado o grupo CEF. Este por sua vez também respeita todas as regras. O processo segue até que não sejam possíveis gerar mais agrupamentos. O resultado final está ilustrado na Figura 22.

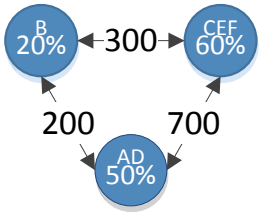



**Figura 22: Particionamento gerado pelo algoritmo proposto, visualizados no CAFES pelo modelo.**

Uma vez gerado o grafo CWG com os conjuntos de tarefas, o CAFES pode mapear o mesmo em uma NoC 3D. No caso, foi escolhida uma NoC 3D com dimensões de 2×2×1. O resultado do mapeamento foram as seguintes associações: (i) grupo com a tarefa *B* no $PE_{000}$ (onde a localidade é representada por $PE_{xyz}$); (ii) grupo com as tarefas *AD* no $PE_{010}$; e (iii) grupo com as tarefas *CEF* no $PE_{110}$, sendo graficamente representado no CAFES conforme a Figura 23.

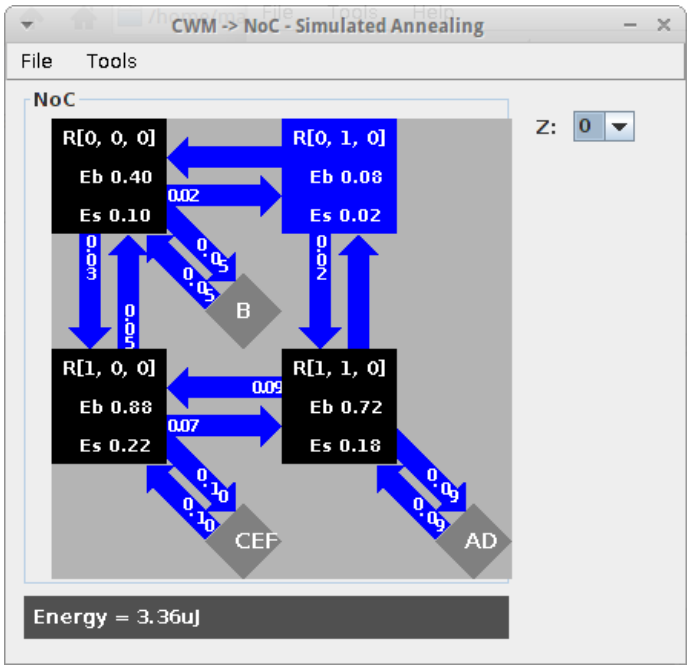



**Figura 23: Mapeamento da aplicação sintética descrita na Figura 22 em uma NoC 3D malha 2×2×1.**

# 7 RESULTADOS EXPERIMENTAIS

Este capítulo descreve resultados experimentais que compara os algoritmos *Simulated Annealing* (SA) e *Partition Reduce* (PR) em relação a requisitos de consumo de energia e balanceamento de carga de processamento. Os experimentos consideram as seguintes métricas:

- Tempo de resposta que os algoritmos levam para particionar a aplicação conforme o requisito escolhido;
- Quantidade de PEs necessária para atender ao particionamento gerado;
- Balanceamento de carga de processamento dos PEs conforme o Erro Quadrático Médio (EQM); e
- Consumo de energia da aplicação (em milijoule - mJ).

Os experimentos realizados assumem dois tipos de cenários:

- Tarefas com caracterização similar se comunicando com no máximo dez outras tarefas e todas as comunicações possuem o mesmo volume de dados, chamado de ***cenário de tarefas homogêneas***; e
- Tarefas geradas com caracterizações completamente randômicas (*power*, *data*, *code*, *cpuOccupation* e *volume*), caracterizando um ***cenário de tarefas heterogêneas***. Adicionalmente, cada tarefa deste conjunto se comunica com no máximo dez outras tarefas onde as comunicações possuem volume de dados randômicos.

## 7.1 COMPARAÇÃO DOS ALGORITMOS PR E SA PARA CENÁRIOS HOMOGÊNEOS

Este conjunto de experimentos tem todas as tarefas com 20% de carga de processamento, com restrição de utilização máxima do PE em 100% de carga, e volume de comunicação de 110 bits. Os experimentos envolvem 12 aplicações sintéticas com as seguintes quantidades de tarefas: 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 8192 e 16384. Este **crescimento exponencial** foi escolhido para permitir analisar situações extremas de operação do algoritmo.

A Figura 24 ilustra o formato de aplicação para cenários de tarefas homogêneas, considerando uma aplicação sintética de 3 tarefas – esta aplicação não foi apresentada nos experimentos.

```
<SYSTEM_SPECIFICATION>
    <TARGET_ARCHITECTURE>
        <PROCESSOR_TYPE_LIST>
            <PROCESSOR_TYPE type="PentiumV">
                <FEATURES frequency="800.0" width="1.0" height="1.0" depth="1.0" />
            </PROCESSOR_TYPE>
        </PROCESSOR_TYPE_LIST>
    </TARGET_ARCHITECTURE>
    <APPLICATION_CHARACTERIZATION>
        <TASK_LIST>
            <TASK id="T0"><PROCESSOR_TYPE type="PentiumV" power="10" data="500" code="100" cpuOccupation="20" /></TASK>
            <TASK id="T1"><PROCESSOR_TYPE type="PentiumV" power="10" data="500" code="100" cpuOccupation="20" /></TASK>
            <TASK id="T2"><PROCESSOR_TYPE type="PentiumV" power="10" data="500" code="100" cpuOccupation="20" /></TASK>
        </TASK_LIST>
    </APPLICATION_CHARACTERIZATION>
    <APPLICATION_DESCRIPTION>
        <COMMUNICATION_TASK_LIST>
            <SOURCE_TASK source="T0">
                <COMMUNICATION target="T1" volume="110" />
                <COMMUNICATION target="T2" volume="110" />
            </SOURCE_TASK>
            <SOURCE_TASK source="T1">
                <COMMUNICATION target="T0" volume="110" />
                <COMMUNICATION target="T2" volume="110" />
            </SOURCE_TASK>
            <SOURCE_TASK source="T2">
                <COMMUNICATION target="T1" volume="110" />
                <COMMUNICATION target="T0" volume="110" />
            </SOURCE_TASK>
        </COMMUNICATION_TASK_LIST>
    </APPLICATION_DESCRIPTION>
</SYSTEM_SPECIFICATION>
```

**Figura 24: Exemplo de aplicação sintética composta por 3 tarefas com comportamento idêntico.**

A Tabela 2 ilustra os resultados dos experimentos, tendo como requisito a minimização do consumo de energia. Estes mesmos resultados são apresentados graficamente na Figura 25 e na Figura 26, com uma subsequente discussão dos resultados mais relevantes.

**Tabela 2: Comparação dos algoritmos SA e PR frente ao requisito de minimização do consumo de energia para aplicações sintéticas compostas por tarefas de comportamento homogêneo.**

| Aplicação (tarefas) | Algoritmo | Tempo (segundos) | Consumo de energia (mJ) |
|---|---|---|---|
| **16384** | SA | N/A | N/A |
| | PR | 333,37 | 199376,28 |
| **8192** | SA | N/A | N/A |
| | PR | 73,70 | 99706,64 |
| **4096** | SA | N/A | N/A |
| | PR | 17,49 | 49762,72 |
| **2048** | SA | N/A | N/A |
| | PR | 3,56 | 24857,76 |
| **1024** | SA | 6132,32 | 12930,98 |
| | PR | 0,89 | 12407,76 |
| **512** | SA | 1259,46 | 6397,86 |
| | PR | 0,32 | 6136,28 |
| **256** | SA | 307,07 | 3132,58 |
| | PR | 0,19 | 3026,67 |
| **128** | SA | 71,35 | 1469,95 |
| | PR | 0,07 | 1441,64 |
| **64** | SA | 18,25 | 667,34 |
| | PR | 0,04 | 668,62 |
| **32** | SA | 5,04 | 267,96 |
| | PR | 0,03 | 278,17 |
| **16** | SA | 1,63 | 88,04 |
| | PR | 0,02 | 102,08 |
| **8** | SA | 1,04 | 19,14 |
| | PR | 0,02 | 26,80 |

A Figura 25 mostra a eficiência de ambos os algoritmos com relação ao tempo necessário para particionar as aplicações. Conforme ilustrado, fica evidente que buscar o melhor consumo de energia pelo algoritmo SA é inviável para um grande número de tarefas, pois o tempo de resposta cresce linearmente e o incremento da quantidade de tarefas cresce quadraticamente, onde a partir de 1024 tarefas executou aproximadamente em uma hora e quarenta minutos. Quando avaliado o PR com 1024 tarefas este levou menos de um minuto e com 16384 tarefas levou apenas 4 minutos, demonstrando ser extremamente eficiente quando comparado ao SA. Uma das principais razões para a redução do tempo é a característica de agrupamento do algoritmo, que a cada laço de iteração reduz em um o número de grupos de tarefas a serem avaliadas.

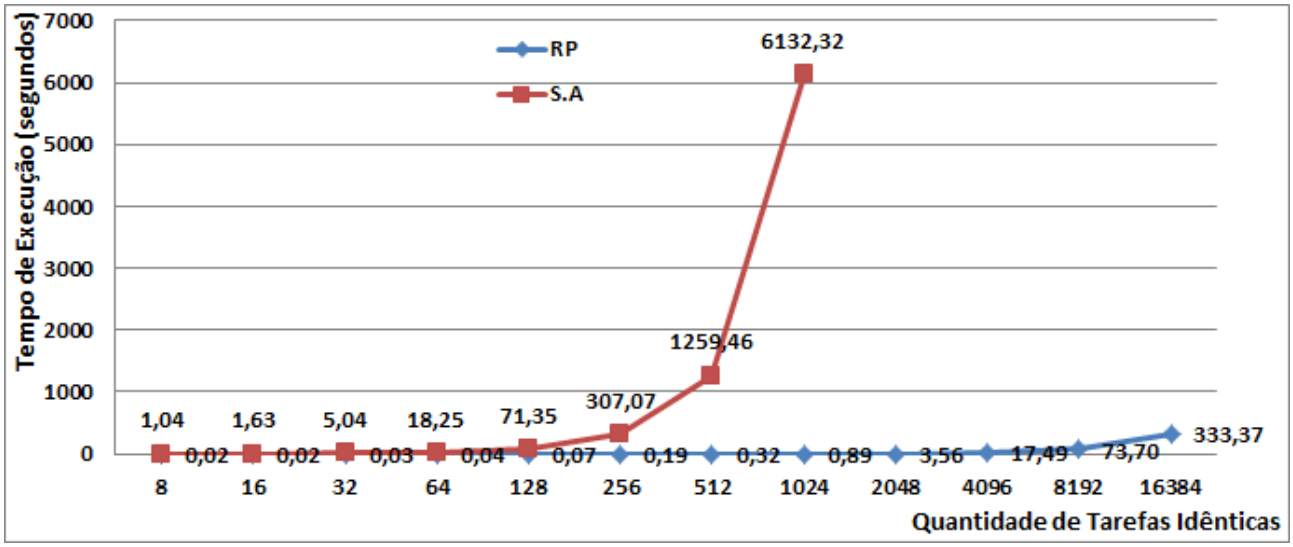



**Figura 25: Experimento da Tabela 2 (cenário homogêneo e requisito de minimização do consumo de energia com crescimento exponencial do número de tarefas) comparando o tempo de execução dos algoritmos SA e PR.**

A Figura 26 mostra que é pequena a diferença dos algoritmos de particionamento na minimização do consumo de energia. Adicionalmente, nota-se que para poucas tarefas (entre 8 a 64 tarefas), o SA apresenta resultados melhores. Enquanto que, a partir de 128 tarefas o PR mostrou ser mais eficaz. Na verdade, proporcionalmente, o SA mostrou ser muito mais eficaz que o PR quando existem poucas tarefas (i.e., menos de 16). Isto ocorre devido a dois fatores, (i) primeiro que o SA faz diversas explorações de particionamento randômicas, que para um conjunto pequeno de tarefas gera uma grande possibilidade de achar um ótimo; (ii) segundo é devido ao determinismo da heurística usada no PR que não explora muitos resultados, podendo levar a mínimos locais. Estes resultados mostram que o algoritmo proposto passa a ser adequado para aplicações de grande complexidade.

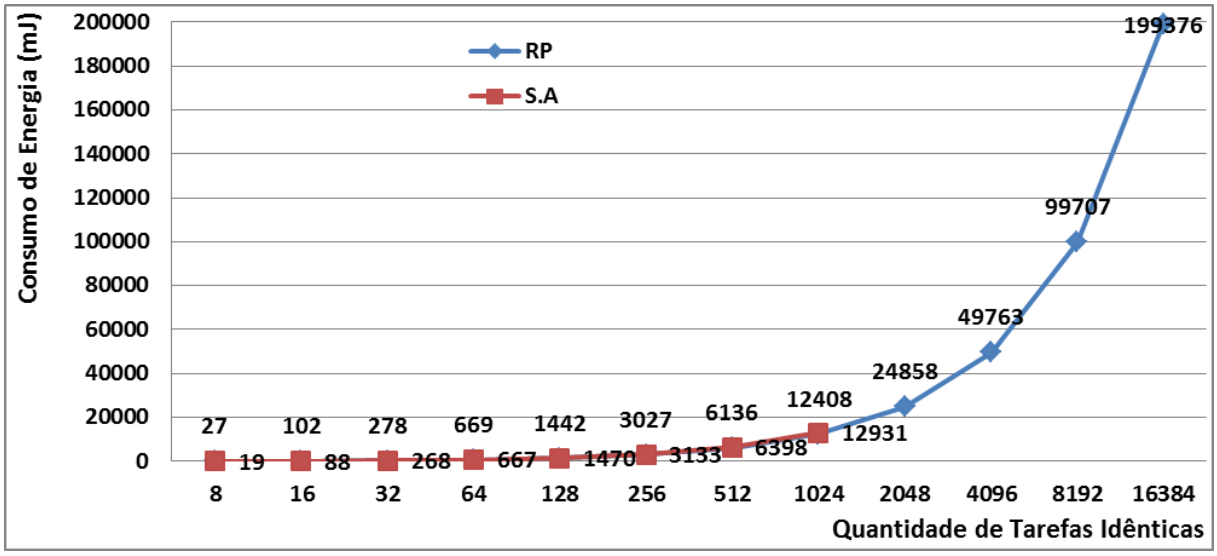



**Figura 26: Experimento da Tabela 2 (cenário homogêneo e requisito de minimização do consumo de energia) destacando o consumo de energia obtido com cada algoritmo frente a variação da quantidade de tarefas.**

A Tabela 3 ilustra os resultados dos experimentos, tendo balanceamento de carga como requisito. Estes mesmos resultados são apresentados graficamente nas três figuras subsequentes, com uma correspondente discussão dos resultados mais relevantes.

**Tabela 3: Comparação da qualidade dos algoritmos SA e PR para particionar aplicações sintéticas compostas por tarefas de comportamento homogêneo, tendo como requisito o balanceamento de carga.**

| Aplicação (tarefas) | Algoritmo | Tempo (segundos) | Número de PEs | Balanceamento de carga (EMQ) |
|---|---|---|---|---|
| **16384** | SA | 471,29 | 4048 | 541,79 |
| | PR | 12,63 | 3277 | 0,12 |
| **8192** | SA | 186,27 | 2025 | 529,35 |
| | PR | 2,65 | 1639 | 2,20 |
| **4096** | SA | 40,91 | 1012 | 492,38 |
| | PR | 0,58 | 820 | 7,80 |
| **2048** | SA | 16,83 | 504 | 447,59 |
| | PR | 0,20 | 410 | 3,89 |
| **1024** | SA | 8,03 | 253 | 375,39 |
| | PR | 0,16 | 205 | 1,94 |
| **512** | SA | 3,87 | 127 | 213,78 |
| | PR | 0,11 | 103 | 34,61 |
| **256** | SA | 2,41 | 64 | 62,50 |
| | PR | 0,08 | 52 | 120,71 |
| **128** | SA | 1,92 | 32 | 0,00 |
| | PR | 0,07 | 26 | 59,17 |
| **64** | SA | 1,52 | 16 | 0,00 |
| | PR | 0,02 | 12 | 28,40 |
| **32** | SA | 1,26 | 8 | 0,00 |
| | PR | 0,03 | 7 | 440,82 |
| **16** | SA | 1,15 | 4 | 0,00 |
| | PR | 0,01 | 4 | 1200,00 |
| **8** | SA | 1,07 | 2 | 0,00 |
| | PR | 0,01 | 2 | 400,00 |

Para os cenários com tarefas homogêneas, a Figura 27, a Figura 28 e a Figura 29 mostram o tempo de execução gasto pelos algoritmos SA e PR, bem como a qualidade dos

algoritmos para obter balanceamento de carga de processamento e o número de PEs necessários para dar suporte às partições geradas, respectivamente.

Conforme ilustra a Figura 27, o SA apresenta um crescimento linear no tempo de particionamento frente ao crescimento exponencial do número de tarefas. Por outro lado, o crescimento do número de tarefas apresentou uma influência menor para o particionamento realizado com o algoritmo PR.

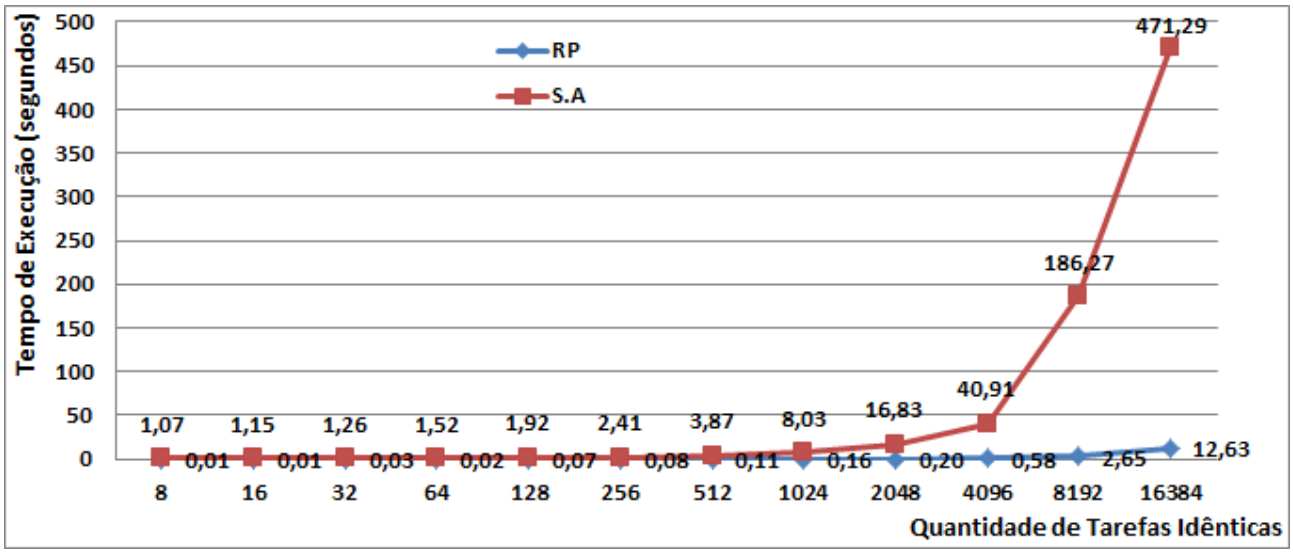



**Figura 27: Experimento da Tabela 3 (cenário homogêneo e requisito balanceamento de carga, com crescimento exponencial do número de tarefas) comparando o tempo de execução dos algoritmos SA e PR.**

Uma medida que destaca a qualidade de um balanceamento de carga é o Erro Médio Quadrático (EMQ).

Equação 8: $$\mathrm{VM} = \frac{\sum_{i=1}^{n} \mathrm{CPU}_i}{n}$$

Onde: Primeiramente é determinado o Valor Médio da carga de processamento (VM), somando-se a carga de processamento de cada processador (CPU) e dividindo pelo número de processadores.

Equação 9: $$\mathrm{EMQ} = \frac{\sum_{t=1}^{n} (VM - \mathrm{CPU}_i)^2}{n}$$

Onde: EMQ é determinado pelo somatório do quadrado da diferença entre o valor médio e a carga de processamento de cada processador, dividido pelo número de processadores.

A Figura 28 ilustra que, de forma similar ao particionamento tendo como requisito a minimização do consumo de energia, para o particionamento com requisito de balanceamento de carga, o algoritmo PR é ineficaz quando realizado em aplicações de

baixa complexidade – no caso com 32 ou menos tarefas. A eficácia do algoritmo é notada apenas para aplicações de alta complexidade – no caso mais de 256 tarefas.

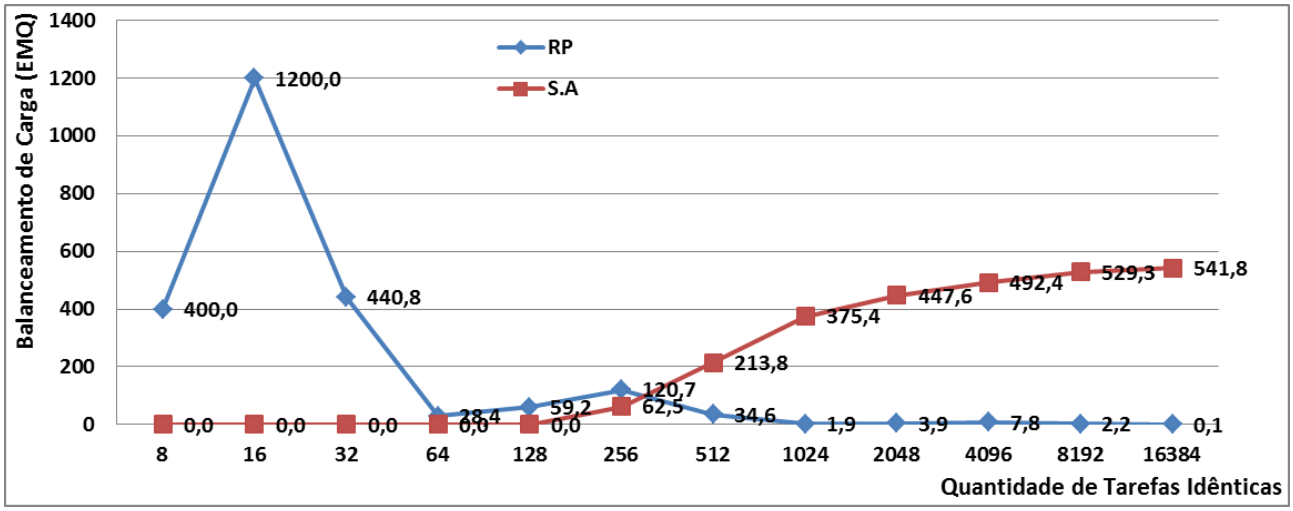



**Figura 28: Experimento da Tabela 3 (cenário homogêneo e requisito balanceamento de carga) destacando o EMQ frente ao crescimento da quantidade de tarefas.**

A Figura 29 mostra que embora o objetivo de balanceamento de carga não seja facilmente alcançável para baixas complexidades, o objetivo de minimizar a quantidade de PEs na arquitetura alvo, que equivale a conseguir agrupar tarefas sem ultrapassar a capacidade de processamento de cada PE, é sempre bem alcançada pelo algoritmo PR, independente da complexidade da aplicação. Note que ao reduzir a quantidade de PEs, o algoritmo pode favorecer a implementações com menor área e com menor consumo de energia.

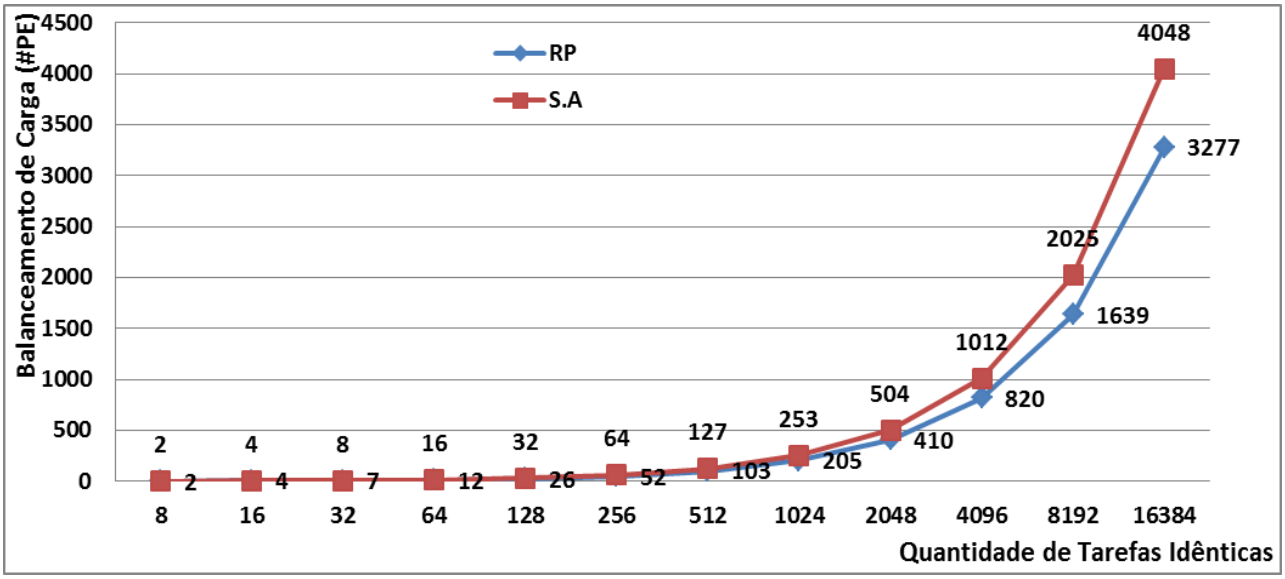



**Figura 29: Experimento da Tabela 3 (cenário homogêneo e requisito balanceamento de carga) destacando a quantidade de PEs necessária para o particionamento frente ao crescimento da quantidade de tarefas.**

## 7.2 COMPARAÇÃO DOS ALGORITMOS PR E SA PARA CENÁRIOS HETEROGÊNEOS

Este conjunto de experimentos tem todas as tarefas com ocupação e volume de dados randômicos, o tamanho do código das tarefas é aleatoriamente escolhido entre 50 e 500 bits, a carga de processamento varia entre 5 e 80%, as tarefas podem comunicar com até no máximo 13 outras tarefas sendo que cada comunicação possui até 100 bits de dados. Os experimentos envolveram 12 aplicações sintéticas com as seguintes quantidades de tarefas: 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 8192 e 16384 considerando como restrição a utilização máxima do PE em 100% de carga.

A Tabela 4 Tabela 2 ilustra os resultados dos experimentos que compram os algoritmos PR e SA, tendo como requisito a minimização do consumo de energia. Estes mesmos resultados são apresentados graficamente na Figura 30 e na Figura 31, com uma subsequente discussão dos resultados mais relevantes.

**Tabela 4: Comparação dos algoritmos SA e PR para particionar aplicações sintéticas em um cenário heterogêneo e tendo como requisito a minimização do consumo de energia.**

| Aplicação (tarefas) | Algoritmo | Tempo (segundos) | Consumo de energia (mJ) |
|---|---|---|---|
| **16384** | SA | N/A | N/A |
| | PR | 314,52 | 559690,66 |
| **8192** | SA | N/A | N/A |
| | PR | 72,46 | 281279,34 |
| **4096** | SA | N/A | N/A |
| | PR | 16,50 | 139618,97 |
| **2048** | SA | N/A | N/A |
| | PR | 3,36 | 69836,27 |
| **1024** | SA | 7879,40 | 38441,15 |
| | PR | 0,84 | 34209,07 |
| **512** | SA | 1731,33 | 18925,01 |
| | PR | 0,30 | 17014,75 |
| **256** | SA | 429,38 | 9539,52 |
| | PR | 0,13 | 8574,21 |
| **128** | SA | 112,47 | 4598,43 |
| | PR | 0,07 | 4234,71 |
| **64** | SA | 28,64 | 2266,90 |
| | PR | 0,03 | 2099,84 |
| **32** | SA | 8,02 | 975,49 |
| | PR | 0,03 | 954,65 |
| **16** | SA | 3,25 | 330,84 |
| | PR | 0,02 | 349,81 |
| **8** | SA | 1,74 | 89,65 |
| | PR | 0,01 | 89,65 |

A Figura 30 compara a eficiência dos algoritmos SA e PR com relação ao tempo necessário para particionar as aplicações de cenário heterogêneo. É possível observar que a tempo de computação do SA tem crescimento linear com o crescimento da quantidade de tarefas, inviabilizando o uso para cenários muito complexos, enquanto que o PR mostra ser adequado para trabalhar com cenários compostos por muitas tarefas. Adicionalmente,

a comparação da Figura 30 com a Figura 25 mostra que a caraterística da aplicação tem pouca influência no tempo de computação, enfatizando que a complexidade está no próprio algoritmo de particionamento.

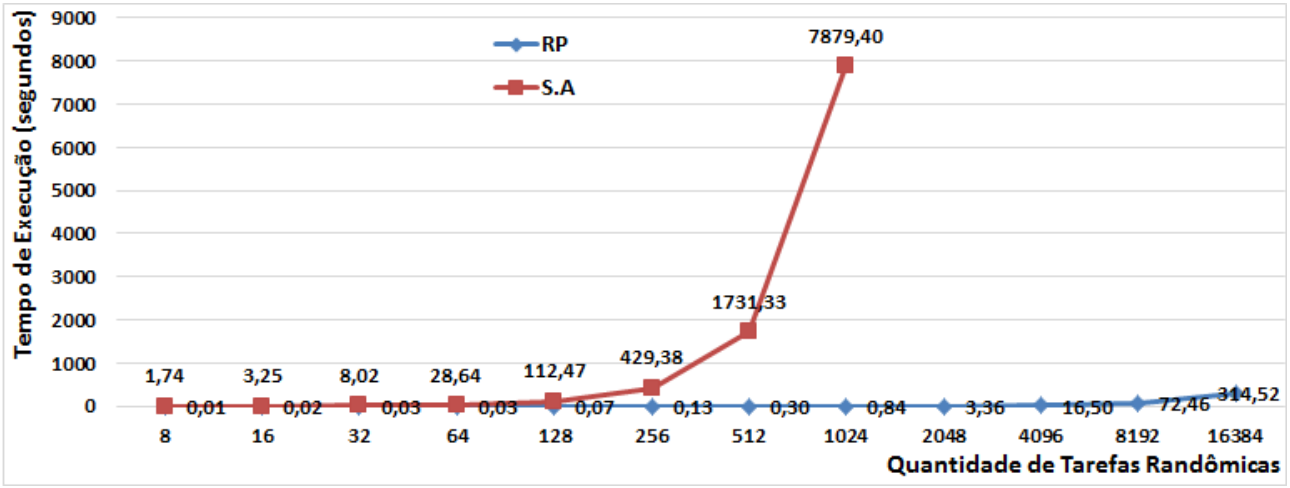



**Figura 30: Experimento da Tabela 4 (cenário heterogêneo e requisito de minimização do consumo de energia) comparando o tempo de execução dos algoritmos SA e PR.**

A Figura 31 ilustra que os algoritmos têm eficácia similar para a obtenção de particionamentos que minimizam o consumo de energia. Adicionalmente, a comparação da Figura 26 com a Figura 31 mostra que, o algoritmo PR obtém resultados muito melhores para cenários heterogêneos do que para cenários homogêneos. Este comportamento ocorre devido às diferenças entre as tarefas permitir ao algoritmo encontrar mais facilmente tarefas com características dispares e selecionar as que mais se adequam para atender ao requisito de particionamento.

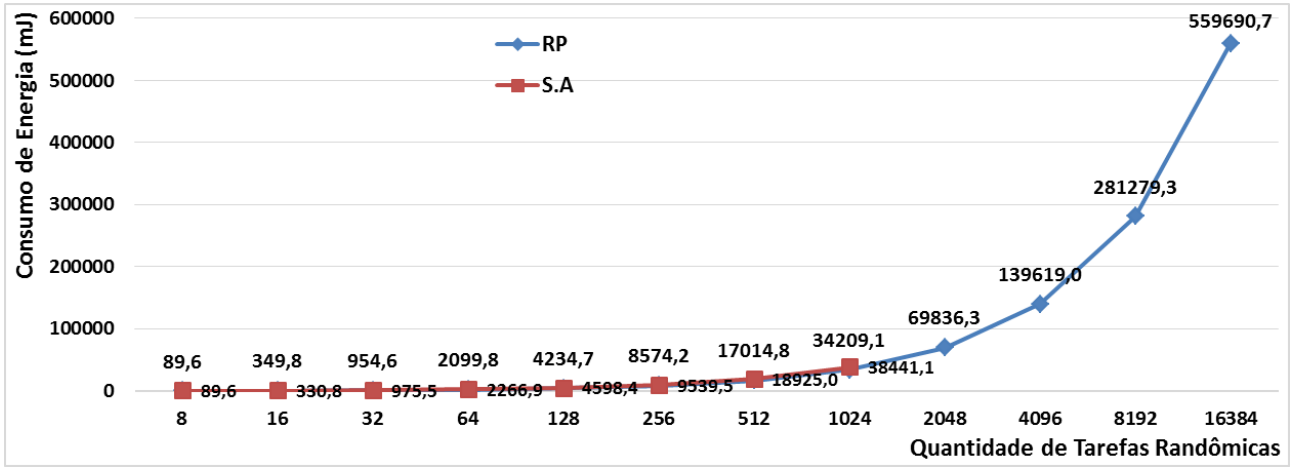



**Figura 31: Experimento da Tabela 4 (cenário heterogêneo e requisito de minimização do consumo de energia) destacando o consumo de energia obtido com cada algoritmo frente a variação da quantidade de tarefas.**

A Tabela 5 ilustra os resultados dos experimentos, tendo como requisito o balanceamento de carga. Estes mesmos resultados são apresentados graficamente nas

três figuras subsequentes, com uma correspondente discussão dos resultados mais relevantes.

**Tabela 5: Comparação dos algoritmos SA e PR para particionar aplicações sintéticas em um cenário heterogêneo e tendo como requisito o balanceamento de carga.**

| Aplicação (tarefas) | Algoritmo | Tempo (segundos) | Número de PEs | Balanceamento de carga (EMQ) |
|---|---|---|---|---|
| **16384** | SA | 1108,57 | 11898 | 537,14 |
| | PR | 16,60 | 7723 | 5,48 |
| **8192** | SA | 451,71 | 5953 | 531,20 |
| | PR | 3,76 | 3889 | 5,85 |
| **4096** | SA | 105,90 | 2980 | 534,88 |
| | PR | 0,50 | 1927 | 6,63 |
| **2048** | SA | 32,27 | 1507 | 543,59 |
| | PR | 0,13 | 956 | 7,92 |
| **1024** | SA | 15,05 | 741 | 512,98 |
| | PR | 0,05 | 481 | 19,54 |
| **512** | SA | 7,16 | 386 | 455,53 |
| | PR | 0,04 | 244 | 7,45 |
| **256** | SA | 3,79 | 191 | 428,67 |
| | PR | 0,03 | 120 | 9,86 |
| **128** | SA | 2,56 | 107 | 354,43 |
| | PR | 0,03 | 61 | 5,87 |
| **64** | SA | 2,00 | 51 | 293,95 |
| | PR | 0,02 | 30 | 71,70 |
| **32** | SA | 1,73 | 25 | 154,26 |
| | PR | 0,01 | 16 | 109,74 |
| **16** | SA | 1,40 | 12 | 216,26 |
| | PR | 0,01 | 9 | 302,17 |
| **8** | SA | 2,15 | 6 | 108,54 |
| | PR | 0,01 | 4 | 461,99 |

Para os cenários heterogêneos, a Figura 32, a Figura 33 e a Figura 34 ilustram o tempo de execução gasto pelos algoritmos SA e PR, bem como a qualidade dos algoritmos para obter balanceamento de carga de processamento e o número de PEs necessários para dar suporte às partições geradas, respectivamente.

Os resultados obtidos para os cenários heterogêneos são muito semelhantes dos obtidos para cenários homogêneos, tal como pode ser observado na comparação entre a Figura 32 e a Figura 27. Também convém ressaltar, que embora o crescimento de tempo de computação seja linear frente ao crescimento exponencial do número de tarefas, o particionamento utilizando a função de balanceamento de carga, consome menos processamento, de forma que, mesmo para os cenários mais complexos o SA consegui computar em tempo aceitável, enquanto que a Figura 25 e a Figura 30 ressaltam a complexidade do algoritmo para particionar com o requisito de consumo de energia.

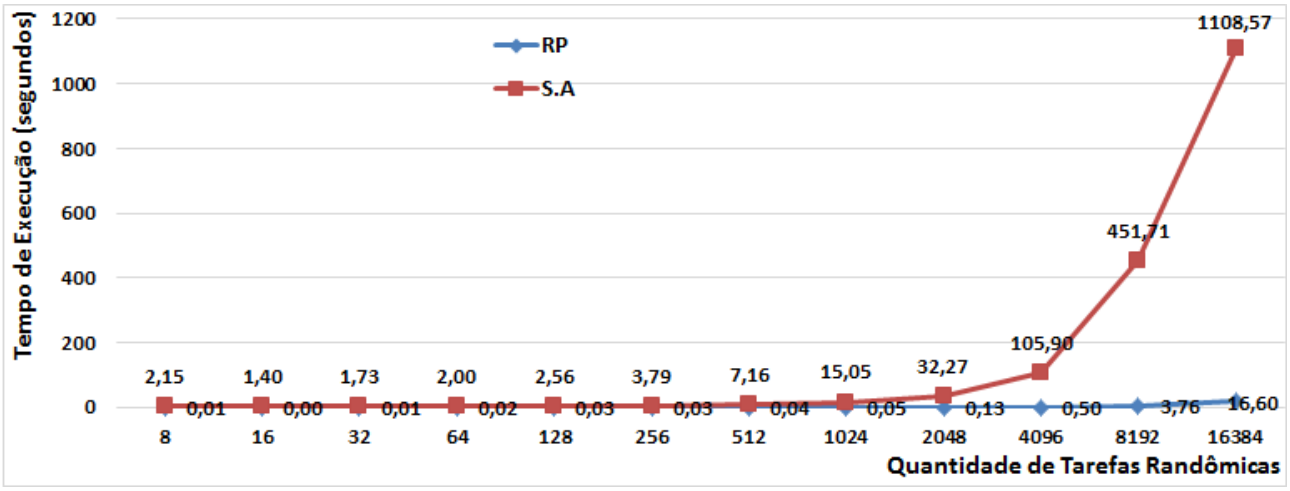



**Figura 32: Experimento da Tabela 5 (cenário heterogêneo e requisito de balanceamento de carga, com crescimento exponencial do número de tarefas) destacando o tempo de execução dos algoritmos frente ao crescimento da quantidade de tarefas.**

A comparação da Figura 33 com a Figura 28 mostra que a vantagem que o SA tinha na obtenção de ótimas partições para um pequeno conjunto de tarefas é bastante reduzida, e já com um cenário com mais de 32 tarefas o PR consegue resultados muito bons, aproximando-se do ótimo (i.e., EMQ igual a 0) a partir de 128 tarefas. Adicionalmente, a Figura 33 mostra que enquanto que o SA tende a piorar o EQM com o crescimento do número de tarefas, o PR tende a melhorar, mostrando uma das principais vantagens desta abordagem algorítmica.

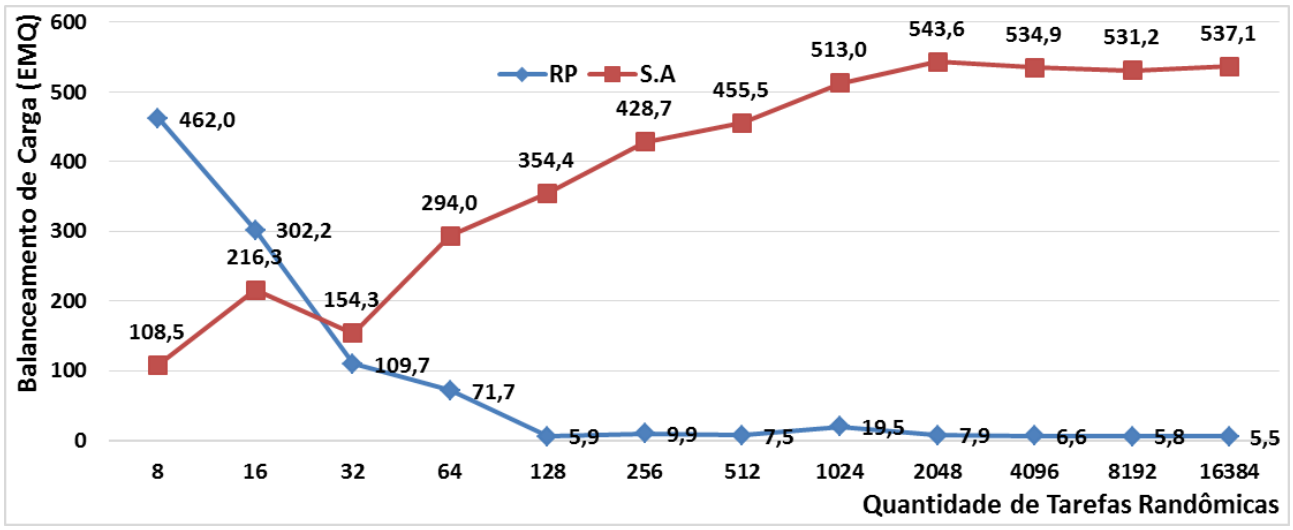



**Figura 33: Experimento da Tabela 5 (cenário heterogêneo e requisito de balanceamento de carga) destacando o EMQ frente ao crescimento da quantidade de tarefas.**

Similarmente ao comportamento apresentado para cenários homogêneos (Figura 29), a Figura 34 mostra que o objetivo de minimizar a quantidade de PEs na arquitetura alvo, é sempre bem alcançada pelo algoritmo PR, independente da complexidade da aplicação. Adicionalmente, a característica da heterogeneidade da aplicação permite que o

PR capture ainda mais detalhes permitido reduzir ainda mais a quantidade de PEs, mesmo para aplicações de baixa complexidade.

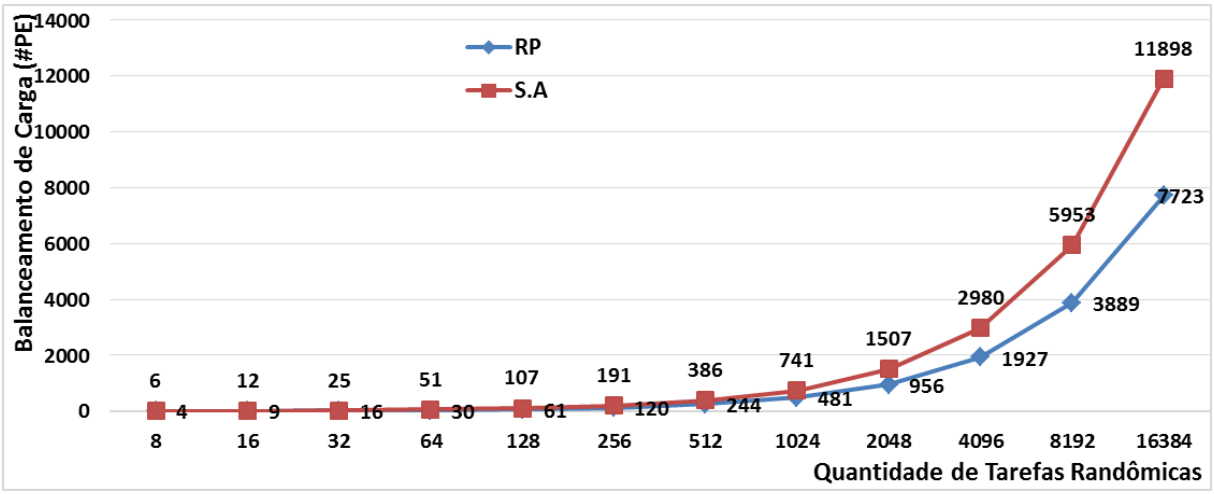



**Figura 34: Experimento da Tabela 5 (cenário heterogêneo e requisito de balanceamento de carga) destacando a quantidade de PEs necessária para o particionamento frente ao crescimento da quantidade de tarefas.**

## 7.3 Conclusões Finais Sobre os Resultados Experimentais

### 7.3.1 Consumo de Energia

Ao avaliarmos o consumo de energia, o PR consome praticamente a mesma quantidade de energia até 64 tarefas, a partir deste ponto, começa a reduzir o consumo de energia em comparação com SA, para o mesmo conjunto de tarefas. No entanto, essa diferença se torna mais evidente no incremento das tarefas. Essas diferenças ocorrem porque o PR busca por mínimos locais, enquanto o SA executa um número arbitrário de passos com pesquisa randômica pela melhor solução. O PR converge rapidamente para uma boa solução, mas não para a melhor solução, enquanto que o SA perde a qualidade quando o número de tarefas cresce devido à natureza NP-completa do problema, aumentando também o tempo de execução. Embora a economia de energia seja semelhante para a maioria dos resultados, os tempos de execução são expressivamente diferente.

O tempo de execução do SA aumenta consideravelmente mais rápido do que o tempo de execução do PR. Ao comparar as duas abordagens, é evidente que se torna inviável a utilização do SA para um número grande de tarefas, uma vez que os experimentos mostraram que para 1024 tarefas o tempo de execução dura cerca de duas horas, fazendo com que os resultados para mais tarefas sejam inviáveis, mesmo utilizando poucas iterações no algoritmo do SA. Em contrapartida, o PR converge em menos de um segundo para um experimento com 1024 tarefas, e em apenas quatro minutos para 16384

tarefas, demonstrando visivelmente a vantagem de usar PR para aplicações maiores nessas configurações.

### 7.3.2 Balanceamento de Carga

Algoritmo PR não fornece bons particionamento considerando o requisito de balanceamento de carga para aplicações com poucas tarefas. Esse comportamento ocorre porque PR otimiza localmente os grupos de tarefas, e os grupos restantes geralmente têm menos carga balanceada em comparação com os anteriores, já que há menos tarefas para inserir nos grupos finais e, portanto, menos carga de processamento. No entanto, quando a quantidade de tarefas aumenta, a abordagem global do SA não consegue um agrupamento adequado devido à crescente complexidade; nota-se que o problema de particionamento cresce linearmente de acordo com a quantidade de tarefas.

Uma das vantagens mais importantes do PR é a redução de grupos de tarefas gerados, o que permite explorar de forma eficiente soluções que minimizem a quantidade de PEs necessárias na arquitetura alvo. O algoritmo PR tem um tempo de computação muito menor que o algoritmo SA para qualquer quantidade de tarefas, sendo mais evidente quando a complexidade da aplicação cresce. Por fim, os resultados experimentais mostram que o PR produz grupos de tarefas mais equilibrados com o EMQ tendendo a zero de acordo com o incremento de tarefas.

# 8 CONCLUSÕES

Algoritmos de particionamento e mapeamento possuem natureza NP-completa, o que impede a busca por soluções exaustivas, sendo comumente usados algoritmos heurísticos clássicos. Um exemplo destes algoritmos é o SA, que normalmente encontra uma boa solução em tempo aceitável para aplicações de baixa e média complexidade.

Este trabalho propõe o algoritmo Partition Reduce (PR), que é derivado da ideia do MapReduce. O PR é um algoritmo heurístico determinístico que tem a vantagem de reduzir a complexidade do problema a cada iteração. Assim, ele consegue tratar de problemas de média e alta complexidade em tempo computacional muito menor que o SA. Conforme descrito inicialmente, mil núcleos por chip é uma perspectiva tecnológica que deverá se tornar realidade dentro de uma década, e ainda demonstrado na Figura 25 e na Figura 28, tornam inviável um possível particionamento pela redução de energia com uma grande escala de tarefas utilizando SA devido ao tempo de resposta, sendo o PR um algoritmo eficiente por causa da redução de uma tarefa a cada iteração (vide que a cada iteração é feito o agrupamento entre duas tarefas).

O particionamento pelo balanceamento de carga pelo SA é mais eficaz para poucas tarefas, ou seja, quando o problema tem baixa complexidade. Uma vez que a complexidade do problema cresce, o algoritmo aqui proposto demonstra ser mais eficaz. Adicionalmente, o PR consegue também reduzir a quantidade mínima de PEs necessária para mapear os grupos de tarefas. Sendo a eficiência medida em termos de tempo de processamento, o PR mostrou ser o mais eficiente para qualquer complexidade de particionamento. Tais resultados mostram a significância do algoritmo proposto para atuais e futuras aplicações de média e alta complexidade.

Por fim, a dissertação também serviu como uma contribuição importante para a implementação de ferramentas inseridas nos frameworks PALOMA e CAFES, que são voltadas ao particionamento de tarefas em grupos e ao mapeamento de tarefas ou grupos de tarefas em PEs, bem como o mapeamento de PEs em tiles, tendo como arquitetura alvo MPSoCs baseados em NoCs 3D.

# 9 REFERÊNCIAS BIBLIOGRÁFICAS


[ANT11] Antunes, E. **Particionamento de MPSoCs Homogêneos Baseados em NoCs.** *Dissertação de Mestrado,* PPGCC/PUCRS, Porto Alegre, Brasil, 2011.

[AWT06] Topol, A. et al. **Three-Dimensional Integrated Circuits**. *IBM J. Research and Development*, vol. 50, n. 4/5, pp. 491-506, 2006.

[BEA93] Beaty, S. **Genetic Algorithms versus Tabu Search for Instruction Scheduling**. *Int. Conf. on Neural Network and Genetic Algorithms*, pp. 496–501, 1993.

[CAI10] Cai, J.; Peng, J.; Lei, Y.; Yue, H.; Zan, L. **Through-silicon via (TSV) capacitance modeling for 3D NoC energy consumption estimation.** *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 815-817, 2010.

[CHE10] Chen, R.; Chen, H.; Zang, B. **Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling**. *International conference on Parallel architectures and compilation techniques (PACT)*, pp. 523-534, 2010.

[DEA04] Dean, J.; Ghemawat, S. **MapReduce: Simplified Data Processing on Large Clusters**. *Symposium on Operating Systems Design and Implementation*, pp. 137-150, 2004.

[GLF89] Glover, F. **Tabu Search, Part I**, *ORSA Journal on Computing*, vol. 1, n. 3, pp. 190-206, 1989.

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

[GUT01] Gutmann, R.; Lu, J.-Q.; Kwon, Y.; McDonald, J.; Cale, T. **Three-dimensional (3D) ICs: A technology platform for integrated systems and opportunities for new polymeric adhesives**. *Conference on Polymers and Adhesives in Microelectronics and Photonics*, pp. 173–180, 2001.

[HEL98] K. Helsgaun. **An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic**. *Datalogiske Skrifter,* 1998, 71p.

[ITR14] ITRS, **International Technology Roadmap for Semiconductors.** *www.itrh.net*, 2014.

[HIL11] Hilbrich, R.; van Kampenhout, J. **Partitioning and Task Transfer on NoC-based Many-Core Processors in the Avionics Domain.** *Journal Softwaretechnik-Trends*, vol. 30, n. 3, 2011, 6 p.

[JER05] Jerraya, A.; Tenhunen, H.; Wolf, W. **Introduction: Multiprocessor Systems-on-Chips**. *IEEE Computer*, vol. 38, n. 7, pp. 36-40, Jul. 2005.

[KIR83] Kirkpatrick, S.; Gelatt, C.; Vecchi, M. **Optimization by simulated annealing**. *Science*, vol. 220, pp. 671-680, 1983.

[MAN11] Mandelli, M.; **Mapeamento dinâmico de aplicações para MPSOCS homogêneos**. *Dissertação de Mestrado*, PPGCC/PUCRS, Porto Alegre, Brasil, 2011.

[MAR01] Martin, G.; Chang, H.; **System-on-Chip Design**. *International Conference on ASIC*, pp. 12-17, 2001.

[MAR05] Marcon, C. **Modelos para o Mapeamento de Aplicações em Infraestruturas de Comunicações Intrachip**. *Tese de Doutorado*, PPGC/UFRGS, Porto Alegre, Brasil, 2005.

[MAR14] Marcon, C.; Webber, T.; Poehls, L.; Pinotti, I. **Pre-mapping Algorithm for Heterogeneous MPSoCs.** *International Conference on VLSI Design,* pp. 252-257, 2014.

[MAR10] Marcon, C.; Calazans, N.; Moreno, E.; Moraes, F.; Hessel, F.; Susin, A. **CAFES: A framework for intrachip application modeling and communication architecture design.** *Journal of Parallel and Distributed Computing*, vol. 71, n. 5, pp. 714-728, May 2011.

[PIN12] Pinotti, I.; Webber, T.; Ribeiro, N.; Fraga, C.; Fagundes, R.; Marcon, C. **Partitioning Algorithms Analysis for Heterogeneous** NoC **Based MPSoC**. *Brazilian Symposium on Computing System Engineering (SBESC)*, pp. 178-183, 2012.

[POS07] Posamentier, A.; Lehmann, I. **The fabulous Fibonacci numbers**. Prometheus Books, 2007.

[WAD13] Wadhwani, P; Choudhary, N.; Singh, D. **Energy Efficient Mapping in 3D Mesh Communication Architecture for NoC.** *Global Journal of Computer Science and Technology Network, Web & Security*, vol. 13, n. 14, pp. 1-5, 2013.

[SEP13] Sepúlveda, J.; Gogniat, G.; Sepúlveda, D.; Pires, R.; Chau, W.; Strum, M. **3DMIA: A Multi-objective Artificial Immune Algorithm for 3D-MPSoC Multi-Application 3D-NoC Mapping.** *Annual conference companion on Genetic and evolutionary computation conference companion*, pp. 167-168, 2013.

[SIO10] Siozios, K.; Anagnostopoulos, I.; Soudris, D. **A High-Level Mapping Algorithm Targeting 3D NoC Architectures with Multiple Vdd.** *IEEE Annual Symposium on Computer Society (ISVLSI)*, pp. 444-445, 2010.

[STA08] Stackhouse, B.; Cherkauer, B.; Gowan, M.; Gronowski, P.; Lyles, C. **A 65nm 2-Billion-Transistor Quad-Core Itanium® Processor.** *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 92-598, 2008.

[STE15] Stefani, M.; Webber, T.; Fernandes, R.; Cataldo, R.; Poehls, L.; Marcon, C. **Task Partitioning Optimization Algorithm for Energy Saving and Load Balance on NoC-based MPSoCs**. *International Symposium on Quality Electronic Design (ISQED)*, 2015, 6p. (artigo aceito para publicação).

[SVA07] Vangal, S.; Howard, J.; Ruhl, G. **An 80-Tile 1.28 TFLOPS Network-on-Chip in 65 nm CMOS.** *IEEE International Solid-State Circuits Conference (ISSCC),* p. 98-589, 2007.

[UMI05] Ogras, U.; Hu, J.; Marculescu, R. **Key research problems in NoC design: a holistic perspective**. *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 69-74, 2005.

[YIN13] Tei, Y.; Marsono, M.; Shaikh-Husin, N.; Hau, Y. **Network partitioning and GA heuristic crossover for NoC application mapping**. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1228-1231, 2013.

[WAN11] Wang, J.; Li L.; Pan, H.; Shuzhuan, H.; Zhang, R. **Latency-aware mapping for 3D NoC using rank-based multi-objective genetic algorithm**. *International Conference on ASIC (ASICON),* pp. 413-416, 2011.

[WOL04] Wolf, W. **The Future of Multiprocessor Systems-on-Chips**. *Design Automation Conference (DAC),* pp. 681-685, 2004.

[ZHA14] Zhang, X.; Hu, B.; Jiang, J. **An Optimized Algorithm for Reduce Task Scheduling**. *Journal of Computers*, v. 9, n. 4, pp. 794-801, Apr. 2014.

# 10 APÊNDICE I

Este apêndice demonstra a geração dos dados de entrada utilizados nos testes de validação do algoritmo PR no caso de redução de energia bem como para o caso de balanceamento de carga.

Uma ferramenta *in-house* foi projetada para gerar aplicações sintéticas contendo um número qualquer de tarefas. Esta ferramenta permitiu gerar arquivos contendo $2^n$ tarefas, com $3 \leq n \leq 16$ (i.e., 8 ... 16384 tarefas).

Os particionamentos das aplicações geradas foram avaliados comparativamente entre os algoritmos SA e PR, levando em consideração os requisitos de consumo de energia e balanceamento de carga.

A geração das aplicações sintéticas levou em consideração dois tipos de tarefas, as idênticas, para testes de validação do algoritmo e tarefas distintas, para tentar simular tarefas reais. As tarefas idênticas são as que possuem todas as funções idênticas, isto é, com mesmo código e tamanho de dados, consomem a mesma quantidade de energia quando executado no PE-alvo, exigindo 20% da carga total de processamento, e as tarefas comunicam com até dez outras tarefas, sendo que cada comunicação tem até 500 bits de dados. As tarefas distintas são geradas aleatoriamente, onde a área de código tem de 5 Kbits a 500 Kbits, o tamanho de dados entre 100 e 10K bits, o consumo de energia entre 10 e 100 phits, e a carga de processamento entre 5 e 80%. Ainda as tarefas se comunica com até 13 outras tarefas e cada uma tem a comunicação entre 5 e 100 bits de dados. O ambiente experimental gerado é ilustrado na Figura 35.

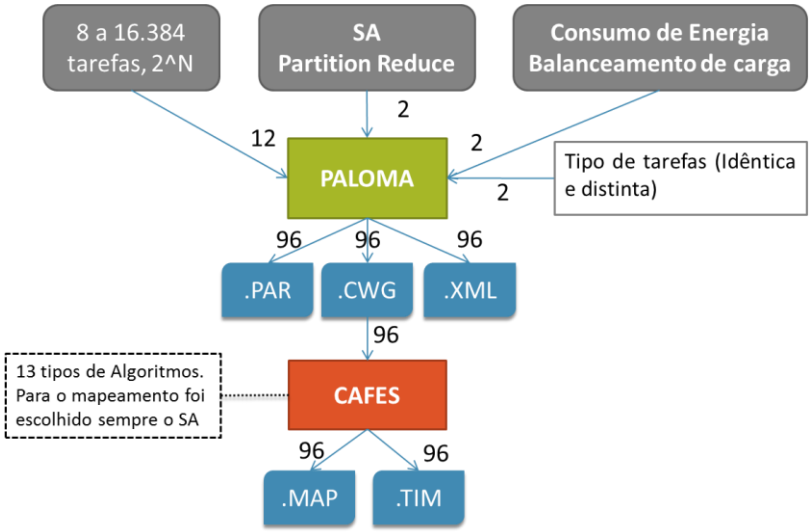



**Figura 35: Configuração do ambiente de teste gerado para a validação do algoritmo PR.**

# 11 APÊNDICE II

Este apêndice apresenta a execução do algoritmo proposto PR para a redução de energia a fim de complementar a explicação da Seção 6.2.2.

Considerando o exemplo sintético ilustrado na Figura 36, a etapa de mapeamento efetua a ordenação pela quantidade de dados transmitidos entre as tarefas, buscando a ligação de maior transferência de dados e a agrupa.

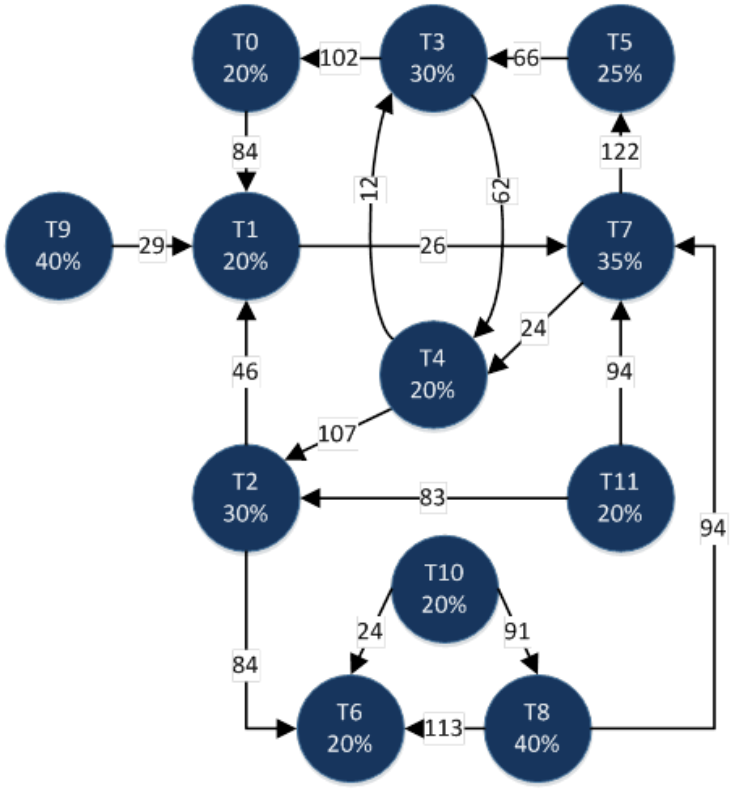



**Figura 36: Exemplo sintético proposto de maior complexidade para a exemplificação mais detalhada do algoritmo PR, quando selecionado o requisito de redução de energia.**

O novo agrupamento gerado é encaminha para a etapa de redução, que é responsável pela verificação das restrições. Caso as restrições sejam atendidas, o agrupamento segue para a etapa de mapeamento, senão o algoritmo marca o agrupamento como inválido antes de encaminhar para o mapeamento. Este processo é executado até não existir mais agrupamentos válidos. Com o encerramento da etapa para a redução com requisito de minimização de consumo de energia, o algoritmo inicia o balanceamento de carga visando reduzir a quantidade de PEs necessários para a execução das tarefas. Assim, a etapa de mapeamento ordena as tarefas ou grupo de tarefas de acordo com a quantidade de processamento necessário para a execução dentro do PE. Adicionalmente, o algoritmo perfaz o agrupamento da tarefa (ou grupo de tarefas) com maior processamento com a tarefa de menor processamento e encaminha para a etapa de redução para validar

as restrições do projeto. A execução desta etapa é detalhada na Tabela 6, onde na linha 11 aparece o resultado final deste particionamento.

**Tabela 6: Fluxo de execução do algoritmo PR para a redução do consumo de energia.**

| Etapas | Mais comunicantes Menos comunicantes |
|---|---|
| **L1** | {T7, T8, T4, T3, T11, T10, T0, T2, T5, T9, T1, T6} |
| **L2** | {T8, T4, T3, T11, T10, T0, T2, (T5, T7), T9, T1, T6} |
| **L3** | {T4, T3, (T8, T6), T11, T10, T0, T2, (T5, T7), T9, T1} |
| **L4** | {T3, (T8, T6), T11, T10, T0, (T4, T2), (T5, T7), T9, T1} |
| **L5** | {(T8, T6), T11, T10, (T3, T0), (T4, T2), (T5, T7), T9, T1} |
| **L6** | {(T8, T6), T11, T10, (T3, T0), (T4, T2), (T5, T7), T9, T1} |
| **L7** | {(T8, T6), T10, (T3, T0), (T4, T2), (T11, T5, T7), T9, T1} |
| **L8** | {(T8, T6, T10), (T3, T0), (T4, T2), (T11, T5, T7), T9, T1} |
| **L9** | {(T8, T6, T10), (T3, T0, T1), (T4, T2), (T11, T5, T7), T9} |
| **L10** | {T9, (T4, T2), (T3, T0, T1), (T8, T6, T10), (T11, T5, T7)} |
| **L11** | {(T3, T0, T1), (T8, T6, T10), (T11, T5, T7), (T4, T2, T9)} |