UNIVERSIDADE FEDERAL DO PAMPA

Ezequiel Luís Vidal

Investigando Heurísticas de Mapeamento de Tarefas sobre a Plataforma HeMPS

Alegrete
2018

Ezequiel Luís Vidal

Investigando Heurísticas de Mapeamento de Tarefas sobre a Plataforma HeMPS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Profª. Drª. Aline Vieira de Mello

Coorientador: Prof. Dr. Ewerson L. de Souza Carvalho

Alegrete
2018

Ezequiel Luís Vidal

Investigando Heurísticas de Mapeamento de Tarefas sobre a Plataforma HeMPS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de Junho de 2018
Banca examinadora:

Aline Vieira de Mello

Profª. Drª. Aline Vieira de Mello
Orientador
UNIPAMPA

p/ Aline Vieira de Mello

Prof. Dr. Ewerson L. de Souza Carvalho
Coorientador
FURG

Prof. Dr. Alessandro Gonçalves Girardi
UNIPAMPA

Claudio Schepke

Prof. Dr. Claudio Schepke
UNIPAMPA

Dedico este trabalho à minha família, em especial à minha amada esposa Driely.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus que esteve comigo nesta caminhada, me dando capacidade para chegar até aqui. Obrigado aos colegas de curso e amigos que fiz na universidade e na cidade de Alegrete, sempre os levarei comigo. Agradeço a todos os professores que me acompanharam durante a graduação, em especial à Profa. Aline Mello e ao Prof. Ewerson Carvalho, responsáveis pela orientação deste trabalho. Agradeço aos membros da banca examinadora, pela disponibilidade de participar e pelas contribuições acerca deste trabalho. Obrigado Unipampa! Meu agradecimento especial à minha família por todo o apoio e compreensão e à minha esposa Driely, pois sem ela ao meu lado eu não conseguiria.

“Pois, que adianta ao homem ganhar o mundo inteiro e perder a sua alma?”
(Marcos 8:36)

RESUMO

Um MPSoC é um sistema integrado em um único chip (SoC), composto por vários elementos de processamento, permitindo assim a execução de múltiplas tarefas em paralelo. Eles são tendência no projeto de circuitos, pois permitem minimizar a crise de produtividade de projeto, representada pela lacuna entre a capacidade da tecnologia do silício e a capacidade atual de projeto de SoCs. Cita-se como exemplos de MPSoCs o TILE-Gx72 e o Kilo-Core, compostos respectivamente por 72 e 1000 núcleos de processamento. Diante desta quantidade abundante de núcleos, estratégias de alocação de recursos eficientes precisam ser desenvolvidas. O mapeamento de tarefas é responsável por atribuir elementos de processamento do MPSoC para cada uma das tarefas de uma dada aplicação. O mapeamento estático é concebido em tempo de projeto, e por isso não pode lidar com cargas de trabalho dinâmicas. Já o mapeamento dinâmico consegue lidar com novas tarefas ou aplicações inseridas no MPSoC em tempo de execução. No entanto, essa estratégia exige o emprego de heurísticas simples e rápidas. Este trabalho investiga o desempenho de heurísticas para mapeamento dinâmico de tarefas. Para atingir este objetivo, primeiramente foi realizada uma pesquisa sobre MPSoCs, e a partir dela foram discutidos os modelos de sistemas encontrados bem como plataformas que simulam estes sistemas. A HeMPS foi selecionada como plataforma alvo deste trabalho. Em seguida, foram investigadas heurísticas de mapeamento de tarefas e as heurísticas *First Free*, *Nearest Neighbor*, *Path Load* e *Best Neighbor* foram selecionadas para implementação na plataforma de simulação. Após esta etapa, foram realizadas simulações de casos de teste com aplicações executando na plataforma. O desempenho e a ocupação dos canais de comunicação da HeMPS foram utilizados como métricas para comparar os resultados das heurísticas. Os resultados das simulações evidenciaram a vocação das heurísticas *Path Load* e *Best Neighbor* em controlar os congestionamentos nos canais de comunicação, tendo a heurística *Best Neighbor* apresentado a menor média de ocupação dos canais de comunicação. Em relação ao desempenho, as heurísticas *Path Load* e *Best Neighbor* obtiveram tempo de mapeamento e simulação até 6,24% superiores as demais heurísticas, porque possuem algoritmos mais complexos em comparação as heurísticas *First Free* e *Nearest Neighbor* que não consideram a ocupação dos canais de comunicação.

Palavras-chave: SoC, MPSoC, Mapeamento de Tarefas, Heurísticas.

ABSTRACT

An MPSoC is an integrated system on a single chip (SoC), composed of multiple processing elements, allowing the execution of multiple tasks in parallel. They are trend in circuit design since allow to minimize the project productivity crisis, represented by the gap between the silicon technology capacity and the SoCs' current design capacity. Examples of MPSoCs are TILE-Gx72 and Kilo-Core, composed respectively by 72 and 1000 processing cores. Faced with this abundant amount of cores, efficient resource allocation strategies need to be developed. Task Mapping is responsible by assigning the MPSoC's Processing for each task of a given application. The Static mapping is performed at design time, so it can not handle dynamic workloads. On the other hand, the dynamic mapping strategies can handle new tasks or applications inserted in MPSoC at run time. However, this strategy requires the employment of simple and fast heuristics. The purpose of this paper is to investigate the performance of heuristics for dynamic task mapping. In order to reach this objective, a research was first carried out on MPSoCs, and from there the models of systems found as well as platforms that simulate these systems were discussed. HeMPS was selected as the target platform for this work. Next, task mapping heuristics were investigated and heuristics *First Free*, *Nearest Neighbor*, *Path Load* and *Best Neighbor* were selected for implementation in the simulation platform. After this step, we performed simulations of test cases with applications running on the platform. The performance and occupation of the HeMPS communication channels were used as metrics to compare the results of the heuristics. The results of the simulations evidenced the vocation of the *Path Load* and *Best Neighbor* heuristics in controlling congestion in the communication channels, with the *Best Neighbor* heuristic presented the lowest average occupation of the communication channels. Regarding the performance, the *Path Load* and *Best Neighbor* heuristics obtained mapping and simulation time up to 6.24% higher than the other heuristics, because they have more complex algorithms in comparison to the *First Free* and *Nearest Neighbor* heuristics that do not consider the occupation of communication channels.

Key-words: SoC, MPSoC, Task Mapping, Heuristics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CI Circuito Integrado

CNN Communication-aware Nearest Neighbor

DDR Double Data Rate

DMA Direct Memory Access

DN Dependences Neighborhood

FIFO First In First Out

HeMPS Hermes Multiprocessor System

IP Intellectual Property Core

LEC-DN Low Energy Consumption-Dependences Neighborhood

MIPS Microprocessor Without Interlocked Pipeline Stages

MPSoC Multi-processor System-on-Chip

NI Network Interface

NN Nearest Neighbor

NoC Network-on-Chip

PBD Platform-Based Design

PCB Printed Circuit Board

PE Processing Element

QAP Quadratic Assignment Problem

RISC Reduced Instruction Set Computer

RTL Register Transfer Level

SoC System-on-Chip

SRAM Static Random Access Memory

TLM Transaction-level Modeling

VLSI Very-Large-Scale Integration

SUMÁRIO

1 INTRODUÇÃO

Sistemas Embarcados são sistemas computacionais projetados para um propósito específico. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado é projetado para realizar um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Como pode-se observar na Figura 1, estes sistemas estão em diversos produtos. Dentre eles pode-se citar: eletrodomésticos, carros, aeronaves, sistemas de monitoramento médico, videogames, etc. Tal tecnologia foi alcançada pelo aumento na capacidade de se integrar transistores, os componentes mais básicos de um sistema computacional.

Figura 1 – Exemplos de sistemas embarcados.

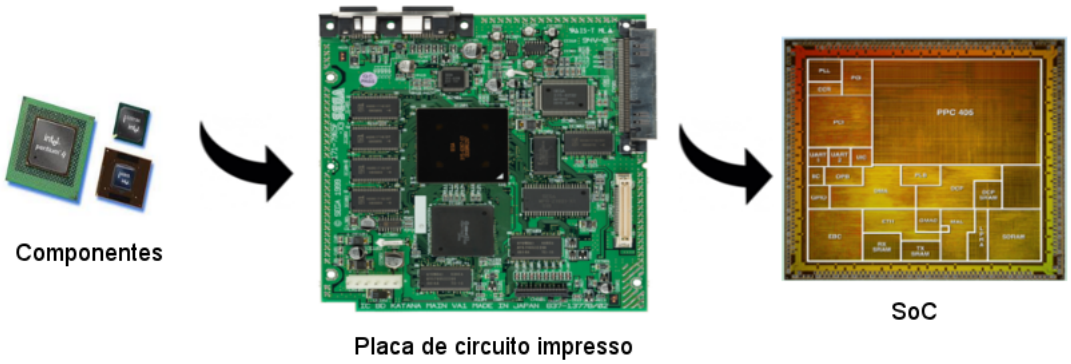
O transistor foi concebido nos laboratórios da Bell Telephone em 1947 e hoje é um componente fundamental na microeletrônica. Com o avanço da tecnologia submicrônica, o transistor teve uma considerável redução nas suas dimensões. Isso fez com que se produzissem Circuitos Integrados (CIs) mais complexos e com alto grau de integração, com um custo relativamente baixo. Com o passar dos anos, a indústria se deparou com a *Crise de Produtividade de Projeto*, pois enquanto a produtividade cresce, a complexidade apresenta um crescimento quase três vezes superior (SIA, 1999).

Neste contexto, o SoC (do inglês *System-on-Chip*) surgiu como uma solução, cuja estratégia consiste em integrar sistemas complexos em uma única pastilha de silício. Um SoC implementa todas as funcionalidades de um sistema eletrônico completo e seus componentes variam de acordo com a aplicação a qual ele irá executar. Os SoCs podem ser compostos por unidades de processamento, memórias e controladores. A característica

de ter seus componentes muito próximos um dos outros faz com que a comunicação entre eles ocorra mais rapidamente, diminuindo o tempo de execução das aplicações.

A principal vantagem do emprego de SoC é o tempo reduzido de projeto. Geralmente, o projeto de SoC é baseado no reuso de núcleos de hardware (do inglês, *Intellectual Property Cores* ou IPs). Como os IP-Cores são módulos pré-projetados e pré-validados, a técnica de reuso pode garantir um tempo menor para o produto chegar ao mercado, fator imprescindível no cenário competitivo atual (CARVALHO, 2009). Conforme representado na Figura 2, a estratégia de SoC pode ser vista como uma mudança no paradigma de projeto baseado em PCB (do inglês *Printed Circuit Board*), onde os componentes são conectados através de uma placa de circuito impresso. Agora, estes componentes podem ser inseridos dentro de um mesmo CI, agrupando IP-Cores na forma de um SoC.

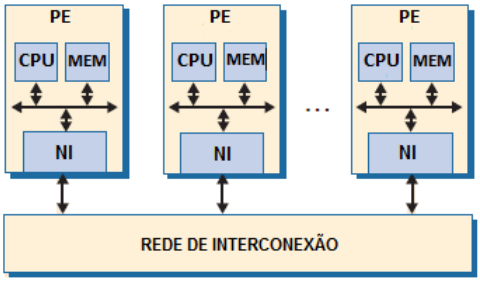
Figura 2 – Mudança no paradigma de projeto baseado em PCB para SoC.



O aumento na complexidade das aplicações exigiram uma maior capacidade de processamento, além disso surgiram problemas causados pelo aumento de potência dos processadores. Buscando uma solução para este problema foram desenvolvidos sistemas computacionais compostos por vários processadores, outrora implementados em clusters. Hoje, estes sistemas são implementados na forma de um SoC e recebem o nome de MPSoC (do inglês, *Multi-Processor System-on-Chip*). Em geral, estes sistemas são compostos por vários elementos de processamento, memórias e IP-Cores específicos. Cabe destacar que em se tratando do número de núcleos, é comum encontrar na literatura termos como *Multicore* e *Manycore*. Um processador Multicore normalmente se refere a dispositivos com 2-8 núcleos, como é o caso dos processadores atuais (*e.g. dual core, quad core, octa core*). Enquanto isso, o termo Manycore geralmente é empregado quando o sistema é composto por um número maior de núcleos, na casa de dezenas ou centenas (SINGH et al., 2013). Por isso, algumas vezes na literatura o MPSoC é chamado de Mar de Processadores (do inglês, *Sea of Processors*).

MPSoCs são arquiteturas que buscam um compromisso entre restrições da tecnologia *Very-Large-Scale Integration* (VLSI) e as necessidades da aplicação (JERRAYA; WOLF, 2005). O MPSoC está se tornando um estilo de projeto cada vez mais utilizado por reduzir o tempo de chegada do produto ao mercado; maximizar a reutilização de projetos; simplificar o processo de verificação; e proporcionar flexibilidade e programabilidade para reuso de plataformas complexas depois de fabricadas (MARTIN, 2006). Conforme ilustra a Figura 3, um MPSoC pode ser composto por vários elementos de processamento, ou *Processing Elements* (PE). Cada PE possui um ou mais processadores, uma ou mais memórias, vários periféricos, e uma interface de rede NI (do inglês *Network Interface*), que permite o acesso à rede de interconexão.

Figura 3 – Estrutura genérica de um MPSoC.



Fonte – adaptado de Nava et al. (2005)

Um MPSoC é chamado de homogêneo quando seus PEs são todos iguais. Já quando utiliza diferentes elementos de processamento, diz-se que é um MPSoC heterogêneo. Um MPSoC homogêneo facilita o desenvolvimento de software e o mapeamento da aplicação, sendo capaz também de simplificar a aplicação de técnicas como migração de tarefas. Enquanto isto, as plataformas heterogêneas tendem a suportar uma variedade maior de aplicações, além de melhorar o desempenho (JERRAYA; WOLF, 2005). Esta melhora de desempenho está relacionada ao fato de que MPSoCs heterogêneos permitem a utilização de um módulo dedicado para cada uma das necessidades de uma aplicação.

O projeto de interconexão entre os todos os elementos de um MPSoC é de suma importância, pois decide como será a infraestrutura de comunicação do sistema. A conexão ponto a ponto apresenta o melhor desempenho por proporcionar alto grau de paralelismo ao sistema, porém apresenta pouca escalabilidade, reusabilidade e flexibilidade por necessitar de um número excessivo de fios para a interligação. Outra alternativa é o emprego de barramento, pois ele permite maior escalabilidade e reusabilidade comparado à conexão ponto a ponto. Contudo, este apresenta um baixo grau de paralelismo, pelo fato de que apenas uma transação pode ser realizada por instante de tempo (MANDELLI, 2011). O emprego de barramento é mais adequado para sistemas Multicore. Para sistemas Manycore, outra abordagem faz-se necessária. Neste caso, pode-se fazer uso de uma

rede intra-chip, comumente chamada de NoC (do inglês *Network-on-Chip*). Esta estratégia oferece uma infraestrutura de comunicação escalável e flexível com alto suporte a modularidade e seu desempenho é poderoso (HESHAM et al., 2017). NoC é um assunto muito pesquisado e seu desempenho se reflete diretamente sobre o desempenho do MPSoC como um todo.

As próximas seções estão organizadas como segue: a seção 1.1 aborda os tópicos de maior interesse em MPSoC. A seção 1.2 trata da motivação deste trabalho, evidenciando a crescente complexidade das atuais aplicações e dos sistemas multiprocessados, e também a busca por melhores estratégias no mapeamento das aplicações em tais sistemas. Na seção 1.3, os objetivos estratégicos e específicos deste trabalho são apresentados. Por fim, a seção 1.4 detalha como o restante deste documento está organizado.

1.1 Tópicos em MPSoC

Esta seção apresenta os tópicos de maior interesse na pesquisa de MPSoCs. Mais detalhes sobre a NoC são apresentados, bem como as vantagens de adotar este tipo de interconexão. Além disso, o problema do mapeamento de tarefas e seus detalhes são introduzidos. Por fim, é apresentada a modelagem de plataformas MPSoC e como esse tipo de abordagem trouxe benefícios na fase de projeto destes complexos sistemas.

1.1.1 Redes Intra-Chip - NoC

Uma NoC é composta basicamente por um conjunto de roteadores e canais de comunicação que interconectam os núcleos de um sistema integrado. Sua funcionalidade é suportar a comunicação entre esses núcleos, que ocorre através da troca de mensagens, geralmente transmitidas na forma de pacotes ao longo da rede (ZEFERINO, 2003). A flexibilidade da NoC deve permitir a conexão de núcleos de diferentes naturezas, como por exemplo: processadores, memórias, dispositivos de entrada/saída ou ainda IP-Cores específicos.

Uma NoC pode ser caracterizada por sua topologia, que corresponde à organização dos seus roteadores de acordo com um grafo, onde os roteadores são representados pelos vértices, e os canais de comunicação pelas arestas. Os tipos mais tradicionais de topologia para NoCs são malha e toro, principalmente por serem regulares e planares, portanto mais facilmente implementados em circuitos 2D. Outra característica importante diz respeito aos mecanismos de comunicação que a NoC adota, pois eles definem a forma como os pacotes trafegam através dela (ZEFERINO, 2003). O *controle de fluxo* realiza a regulação do tráfego nos canais da rede, alocando os recursos necessários para fazer as mensagens trafegarem na NoC. Essa política de regulação é implementada em nível de canal, fazendo o armazenamento dos dados em transferência nos *buffers*. Os tipos de controle de fluxo mais utilizados são controle de fluxo baseados em créditos, em canais virtuais ou em

um protocolo de *handshake*. O *roteamento* é descrito por um algoritmo, que leva em consideração a topologia da rede adotada. Sua função é definir o caminho a ser percorrido por uma mensagem do seu destino até a sua origem. Cita-se aqui como exemplos os algoritmos XY e YX, que são mais utilizados na topologia malha. A *arbitragem* resolve concorrências na rede, gerenciando conflitos internos, quando duas ou mais mensagens tentam acessar o mesmo recurso, que pode ser um buffer ou um canal de saída. O mecanismo de *chaveamento* define como é realizada a transferência de uma mensagem da entrada de um roteador para um de seus canais de saída, e pode ser implementado por circuito ou por pacote. A *memorização* determina o esquema de filas utilizado para armazenar as mensagens. Como exemplo, pode ser utilizado buffers nas entradas, nas saídas ou em ambos os tipos de portas dos roteadores.

Existem contribuições científicas que envolvem os aspectos de suporte à qualidade de serviço, adaptatividade e eficiência energética em NoCs. Várias pesquisas focam em desafios específicos no projeto, como Abbas et al. (2014) e Haiyun (2011). O trabalho de Ruaro, Carara e Moraes (2015) investiga e propõe técnicas de adaptação em tempo de execução dos recursos da NoC, de acordo com os requisitos de qualidade de serviço de cada aplicativo executado no MPSoC. As técnicas propostas reduziram em média 60% as violações de latência, garantindo menor *jitter* e *throughput*, sem afetar o desempenho da rede. Por sua vez Sayuti e Indrusiak (2013) focam na minimização de energia através da alocação de tarefas como uma das mais eficazes formas de abordar a eficiência energética em NoCs em tempo real. Eles propõem um algoritmo de otimização genética multi-objetivo capaz de garantir uma alocação com um consumo de energia minimizado.

1.1.2 Modelagem de MPSoCs

O desempenho e o consumo de energia de um MPSoC dependem não somente das características de hardware como o número de processadores que possui, o tamanho das suas memórias ou o tipo de rede de interconexão empregada, mas também de suas interações, que dependem do mapeamento e da migração das tarefas (GRüTTNER et al., 2011). Em consequência disso, fica evidente o amplo espaço de projeto do MPSoC. Mohanty et al. (2002) dizem que a exploração do espaço de projeto é o processo de analisar as muitas alternativas de implementação para identificar a melhor solução. O amplo espaço de projeto para desenvolver a infra-estrutura de hardware/software, ou para avaliar o desempenho das aplicações requer estruturas adequadas para gerar e simular o MPSoCs.

Os métodos tradicionais de projeto trazem dificuldades nas tomadas de decisão e encarecem o produto, pois demandam simulação em nível de hardware, estando disponível apenas no final do fluxo de projeto (MADALOZZO, 2017). Visando solucionar os problemas apresentados pelos métodos tradicionais de projeto, foi adotada a técnica de projeto baseado em plataforma (PBD, do inglês *Platform-Based Design*). O método

de projeto PBD adota a modelagem de plataformas virtuais em nível de sistema. Isso possibilita obter rápidas simulações, depuração de software e reuso de componentes de hardware.

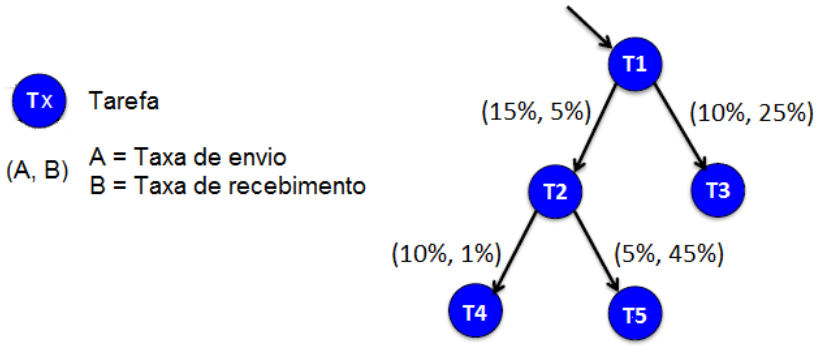
As plataformas virtuais são meios eficientes no qual as funcionalidades do software e do hardware desejado podem ser projetadas e verificadas em conjunto nas fases iniciais do projeto (REKIK et al., 2013). Ao contrário do desenvolvimento tradicional onde o software era implementado após o hardware ter sido finalizado, neste tipo de abordagem a integração do hardware com o software é realizada desde o início do processo de implementação. As plataformas virtuais no desenvolvimento de software embarcado trazem benefícios como diminuir custo de desenvolvimento do produto, aumentar a qualidade do software pelo fato de permitir alto poder de teste e de reduzir os riscos inerentes ao desenvolvimento de software. Para fins científicos, é interessante modelar um MPSoC, pois várias simulações e testes podem ser realizados sem a necessidade da plataforma física. Os MPSoCs podem ser modelados através de ferramentas como ArchC (RIGO et al., 2004), OVP (*Open Virtual Platform*) (OVP, 2016) e SystemC. Vários modelos de MPSoC foram propostos. Podem ser citados os trabalhos de Paulin et al. (2006), Carara et al. (2009), Beserra, Niaki e Sander (2012), Benini et al. (2012) e Zhang et al. (2013). Alguns modelos de plataforma serão discutidos em maiores detalhes no Capítulo 2.

1.1.3 Mapeamento de Tarefas

O mapeamento de tarefas é definido por Mandelli (2011) como o ato de escolher o melhor PE de um MPSoC para alocar uma dada tarefa. A escolha do melhor recurso do sistema para determinada tarefa deve levar em conta os requisitos inerentes a cada aplicação. Para aplicações em tempo real, o atendimento dos prazos ou *deadlines* é fundamental. Em contrapartida, em sistemas portáteis, reduzir o consumo de energia é o objetivo maior. De maneira simplificada, pode-se definir uma aplicação como um conjunto de tarefas que se comunicam entre si. Conforme a Figura 4, uma aplicação pode ser representada através de um grafo dirigido, onde os vértices representam tarefas e as arestas representam comunicações entre as tarefas da aplicação. No exemplo, os pesos nas arestas significam a taxa de comunicação em ambos os sentidos entre cada par de tarefas comunicantes.

Com relação ao momento em que o mapeamento é realizado, ele é dito *estático* ou *dinâmico*. O mapeamento estático é concebido em tempo de projeto, tirando proveito do fato de poder utilizar algoritmos mais complexos e avaliar um maior número de alternativas de mapeamento. Já o mapeamento dinâmico acontece em tempo de execução. Para isso, exige uma heurística simples e rápida tendo em vista que esta pode interferir no desempenho das aplicações. Neste tipo de mapeamento é preciso ter um PE que controle ou gerencie o processo de mapeamento das tarefas. Este controle pode ser *centralizado*, quando um PE é responsável por executar a heurística de mapeamento, ou pode ser *dis-*

Figura 4 – Exemplo de aplicação modelada na forma de um grafo.



tribuído, quando o MPSoC é dividido em *clusters* que possuem um PE que faz o controle naquela região.

O mapeamento também pode ser classificado quanto a característica da arquitetura do MPSoC, se ele é *homogêneo* ou *heterogêneo*. No mapeamento heterogêneo, é necessário realizar um passo de ligação (do inglês, *binding*) antes que a tarefa seja mapeada, com o objetivo de assegurar a atribuição das tarefas apenas aos elementos de processamento adequados (CARVALHO, 2009). Além disso, as tarefas podem ser mapeadas nos PEs segundo duas abordagens, sendo elas a *monotarefa* e a *multitarefa*. Na abordagem monotarefa apenas uma tarefa é mapeada em cada PE, enquanto que a multitarefa admite mais de uma.

A maioria da literatura existente recai sob alocação de recursos estáticos, por exemplo Thiele et al. (2011), Meyer, Hartman e Thomas (2010), Choi et al. (2012) e Castrillon et al. (2012)). Para processamento em tempo real, heurísticas eficientes foram concebidas para atribuir novas tarefas chegando aos recursos do sistema, como Nollet et al. (2008), Schranzhofer, Chen e Thiele (2009), Carvalho, Calazans e Moraes (2010), Wang et al. (2011), Kaushik et al. (2011) e Chen, Marconi e Mitra (2012). Fattah et al. (2014) propõem um algoritmo de mapeamento dinâmico para sistemas com vários núcleos conectados em rede. Este diminui a latência e a dissipação de energia da rede e melhora o tempo de execução das aplicações em 6%. O algoritmo proposto por Ng et al. (2015) trata da fragmentação no sistema, que acontece quando núcleos ficam ociosos devido à falta de conhecimento em tempo de projeto do tempo de término de uma aplicação. Como o mapeamento dinâmico de tarefas será o foco deste trabalho, um revisão mais completa do estado da arte será tratada no Capítulo 2 deste documento.

1.2 Motivação do Trabalho

Uma mudança de paradigma para a adoção de sistemas *Manycore* pode ser observada em vários domínios, como computação embarcada e computação de alto desempe-

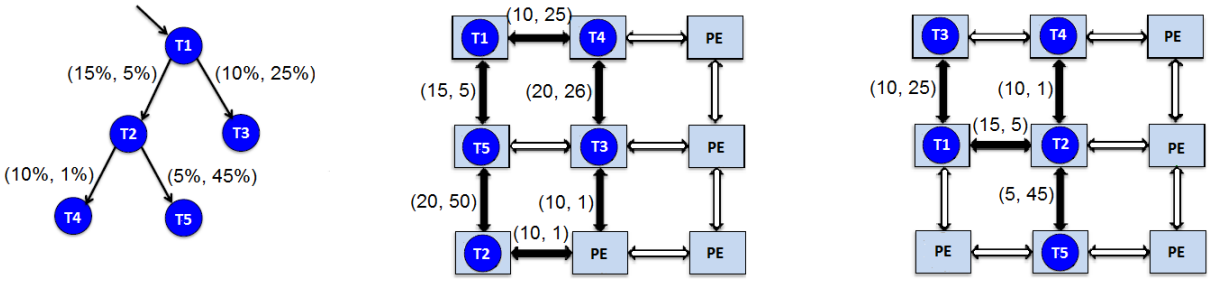
nho. A mudança é necessária dado que os requisitos de desempenho das aplicações não podem ser satisfeitos simplesmente aumentando o número de núcleos ou a frequência de um processador de núcleo único. Os MPSoCs visam superar esses desafios integrando múltiplos núcleos operando em baixas frequências, onde os núcleos podem se comunicar de forma coesa uns com os outros. MPSoCs são realidade, visto que já são construídos por alguns fabricantes. Alguns exemplos incluem o processador de 72 núcleos Tile-Gx72 (MELLANOX, 2015), o chip de 80 núcleos da Intel (VANGAL et al., 2007), o processador MPPA de 256 núcleos da Kalray (DINECHIN et al., 2014) e mais recentemente o processador de 1000 núcleos desenvolvido em parceria entre a IBM e a universidade UC Davis (BOHNENSTIEHL et al., 2016).

Conforme Ding et al. (2014), estratégias de alocação de recursos eficientes precisam ser desenvolvidas tomando em consideração uma quantidade abundante de núcleos e heterogeneidade. O mapeamento de tarefas se encarrega de alocar os recursos necessários de um sistema para que determinada aplicação seja executada, sendo um tópico muito importante em MPSoCs. O problema do mapeamento está na classe de problemas tais como o Problema Quadrático de Alocação (do inglês, *Quadratic Assignment Problem* - QAP) (GAREY; JOHNSON, 1979), um dos problemas de otimização combinatória mais desafiadores que existem. Este problema é considerado NP-completo, ou seja, não é conhecido um algoritmo que o resolva de maneira ótima em tempo polinomial. Quando o MPSoC possui uma rede pequena, como uma NoC 2x2 por exemplo, é possível obter soluções ótimas em tempo consideravelmente curto. Porém, com um número de processadores cada vez maior, o tempo de processamento se torna inviável se não forem empregadas heurísticas eficientes.

A estratégia de mapeamento pode focar na melhoria do tempo de computação de uma aplicação (CHEN; MARCONI; MITRA, 2012). Alguns trabalhos levam em consideração o tempo de computação e também a comunicação das tarefas no MPSoC (NG et al., 2015). Carvalho, Calazans e Moraes (2010) empregam como métrica a taxa de ocupação dos canais da NoC. Levando em consideração essa métrica, a Figura 5 (a) apresenta uma aplicação composta por suas tarefas e as taxas de comunicação entre elas. A Figura 5 (b) ilustra um mapeamento onde as tarefas são mapeadas aleatoriamente no MPSoC. Enquanto isto, a Figura 5 (c) apresenta um possível mapeamento onde as tarefas que se comunicam estão o mais próximo uma das outras. Em ambos os casos o MPSoC possui roteamento XY. Note que as taxas de ocupação dos canais diferem nos dois mapeamentos, evidenciando que por gerar ocupações menores, a primeira estratégia de mapeamento é melhor em comparação com a segunda, propiciando um uso mais eficiente dos canais de comunicação da NoC.

Conforme mencionado anteriormente, o mapeamento estático não pode lidar com cargas de trabalho dinâmicas chegando no MPSoC. Já o mapeamento dinâmico consegue lidar com novas tarefas ou aplicações inseridas no sistema em tempo de execução. Para

Figura 5 – Dois possíveis mapeamentos e a ocupação dos canais resultante.



(a) Exemplo de Aplicação. (b) Mapeamento Aleatório. (c) Mapeamento Otimizado.

isso, exige uma heurística simples e rápida tendo em vista que esta pode interferir no desempenho das aplicações. O desempenho de uma tarefa pode se degradar devido, por exemplo, a sobrecarga de processamento do PE onde esta está executando. A migração de tarefas, que difere do mapeamento de tarefas, é o ato de transferir uma tarefa já mapeada de um processador para outro. Existem trabalhos que abordam este tema, como estratégia de melhorar o mapeamento (MUNK et al., 2015). Embora seja importante, a migração de tarefas não é abordada neste trabalho, que foca o problema do mapeamento de tarefas.

1.3 Objetivos do Trabalho

O objetivo deste trabalho consiste em avaliar o desempenho de estratégias de mapeamento de tarefas. Para isto, uma pesquisa aprofundada na literatura será realizada e as diferentes estratégias serão classificadas. Além disso, heurísticas de mapeamento serão implementadas e integradas em uma plataforma MPSoC para que um estudo comparativo seja realizado. Este comparativo se concentrá nas métricas importantes para sistemas de tempo real baseados em múltiplos núcleos, como desempenho de computação e ocupação dos canais de comunicação. Dentre os objetivos estratégicos do presente trabalho encontram-se:

- Investigar tópicos relativos a MPSoCs;
- Revisar as plataformas de MPSoC disponíveis;
- Investigar MPSoCs existentes atualmente; e
- Revisar a literatura sobre Mapeamento de Tarefas.

Os objetivos específicos deste trabalho são:

- Classificar estratégias de mapeamento encontradas na literatura;
- Implementar heurísticas de mapeamento;

- Integrar heurísticas implementadas em uma plataforma de MPSoC; e
- Avaliar o desempenho das heurísticas implementadas.

1.4 Organização do Documento

Neste primeiro Capítulo foi apresentada a introdução ao assunto, o que incluiu tópicos que abordaram as rede intra-chip, a modelagem de plataformas MPSoCs, o mapeamento de tarefas, além da motivação e dos objetivos deste trabalho.

A sequência deste documento encontra-se organizado da seguinte forma:

O Capítulo 2 aborda com mais detalhes alguns assuntos já apresentados resumidamente na introdução. Este Capítulo apresenta propostas de plataformas de MPSoCS, assim como alguns MPSoCs. Além disso, os trabalhos sobre Mapeamento de Tarefas encontrados na literatura são discutidos.

O Capítulo 3 apresenta a proposta deste trabalho. Para isso, o MPSoC e as heurísticas alvo são apresentados, bem como as métricas que servirão de avaliação para realizar um comparativo entre as heurísticas.

O Capítulo 4 é composto pelo desenvolvimento deste trabalho. Para isso, as implementações das heurísticas alvo são apresentadas, e uma análise delas é realizada.

O Capítulo 5 apresenta os detalhes das simulações realizadas, bem como seus resultados e discussão.

Por fim, o Capítulo 6 traz as considerações finais deste trabalho, discutindo sobre os desafios e limitações encontrados na sua execução, bem como propostas para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Neste Capítulo, são apresentados os produtos baseados na tecnologia MPSoC, na seção 2.1. Seguindo, trabalhos que introduzem diferentes plataformas que simulam um MPSoC, em diferentes níveis de abstração e com diferentes métodos para modelagem são apresentados na seção 2.2. Por fim, algumas abordagens de mapeamento dinâmico de tarefas são apresentadas na seção 2.3.

2.1 MPSoCs

Nas subseções seguintes, serão apresentados alguns produtos de MPSoCs. São eles: TILE-Gx72 na subseção 2.1.1, o MPSoC da Intel na subseção 2.1.2, MPPA-256 na subseção 2.1.3, Kilo-Core na subseção 2.1.4, e o Epiphany-V na subseção 2.1.5.

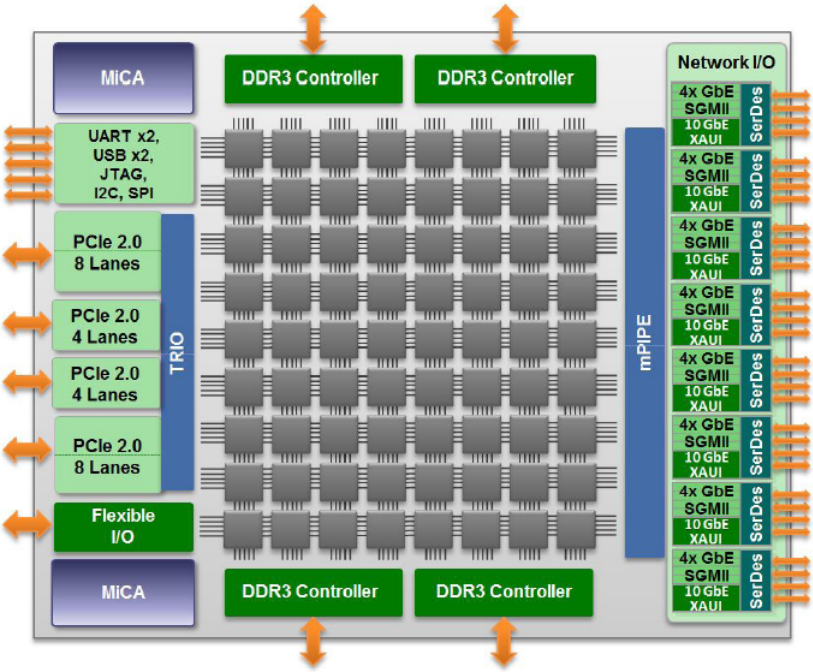
2.1.1 TILE-Gx72

A Mellanox Technologies apresentou o TILE-Gx72 (MELLANOX, 2015), um MPSoC composto por 72 PEs homogêneos conectados pela NoC malha iMESH. O processador é otimizado para redes inteligentes, aplicações multimídia e em nuvem. Cada núcleo possui um processador de 64 bits, além de memórias cache L1 e L2, seis portas PCI Express, oito portas Ethernet 10Gbps e quatro controladores de memória DDR3. Seus PEs operam em uma freqüência entre 1.0GHz e 1.2GHz com consumo de potência entre 51 e 65W a 1.2GHz. Para economizar energia, um dado PE pode entrar no modo *sleep* quando estiver ocioso. Este MPSoC executa uma versão do Linux que pode lidar com todos os núcleos, alocando núcleos específicos para tarefas específicas ou aproveitando-os como um recurso agregado. A programação de aplicativos para o TILE-Gx72 pode ser realizada através do *Multicore Development Environment* (MDE), um ambiente que além da descrição de código em C, permite ainda simulação, análise de desempenho e depuração. A Figura 6 apresenta uma visão geral deste MPSoC.

2.1.2 Terascale

O projeto Terascale da Intel apresentou um MPSoC com 80 PEs idênticos que são conectados utilizando uma NoC (VANGAL et al., 2007) (VANGAL et al., 2008). A NoC apresenta interfaces assíncronas e sua topologia é do tipo malha 2D. A frequência com que o sistema opera é 4 GHz. Cada elemento de processamento possui duas unidades independentes de ponto flutuante de precisão simples (FPMAC), memória de instruções (IMEM), e memória de dados (DMEM). O roteador na NoC apresenta chaveamento *wormhole* e dois canais físicos para transmissão de pacotes. O MPSoC alcança o pico de desempenho de 1.0Tflops à 1V e 1.28Tflops à 1.2V. O consumo de potência estimado é de 98W à 1V e 181W à 1.2V. Todo MPSoC possui em torno de 100 milhões de transistores. A Figura 7 apresenta uma visão geral deste MPSoC.

Figura 6 – MPSoC TILE-Gx72 composto por 72 PEs.



Fonte – Mellanox (2015).

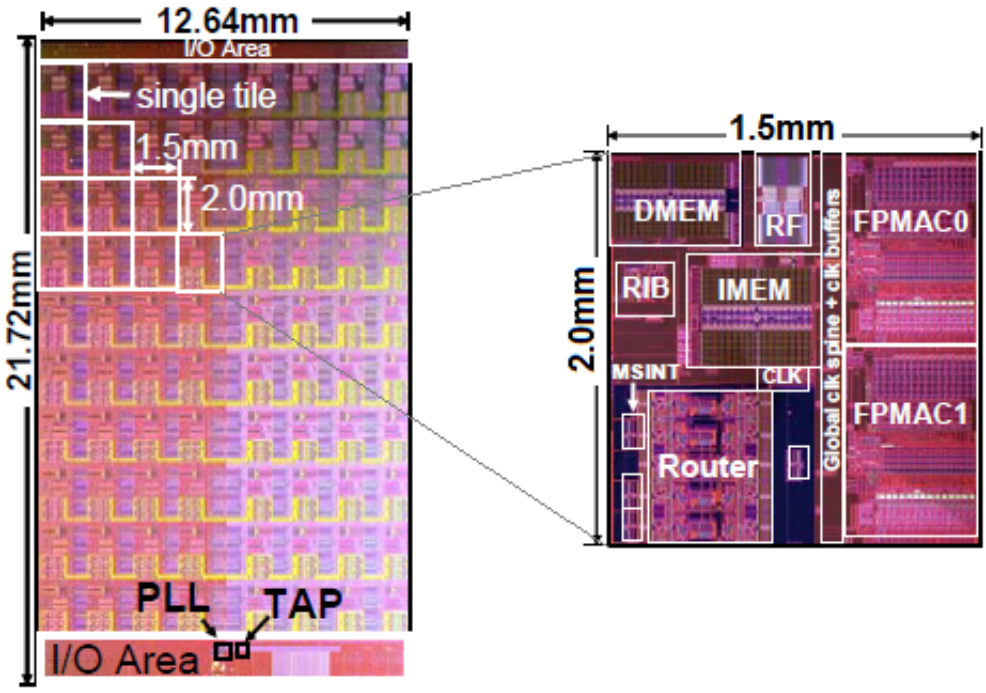
2.1.3 MPPA-256

O MPSoC MPPA-256 (KALREY, 2017) é composto por uma série de 16 clusters e 4 subsistemas de entrada e saída, eles próprios conectados por NoC. A NoC é uma estrutura de topologia *torus* em 2D envolvendo largura de banda duplex total de até 3,2 GBps entre cada cluster adjacente. Cada cluster é composto por 16 núcleos idênticos de 32 bits, um espaço de memória compartilhada, memórias cache L1 de dados e L1 de instruções por núcleo, um controlador DMA, dentre outros. Os subsistemas de E/S implementam vários interfaces padrão como: dois canais DDR de 64 bits de até 12,8 GBps; duas interfaces PCIe; dois controladores Ethernet; um controlador de memória estática universal que permite conectar cinco dispositivos externos como memória serial e memórias SRAM assíncronas; dois bancos de E/S de propósito geral e a interface NoCX, que fornece uma largura de banda agregada de 40Gbps e permite escalar facilmente o número de núcleos. A Figura 8 apresenta um diagrama de blocos do MPPA-256.

2.1.4 Kilo-Core

O chip Kilo-Core (BOHNENSTIEHL et al., 2016) possui 1000 processadores programáveis e 12 módulos de memória independentes capazes de atender simultaneamente os dados e as solicitações de instruções, estes são integrados a um dispositivo CMOS PD-SOI de 32nm. A maioria dos processadores estão dispostos em 32 colunas e 31 linhas,

Figura 7 – MPSoC Terascale composto de 80 PEs.



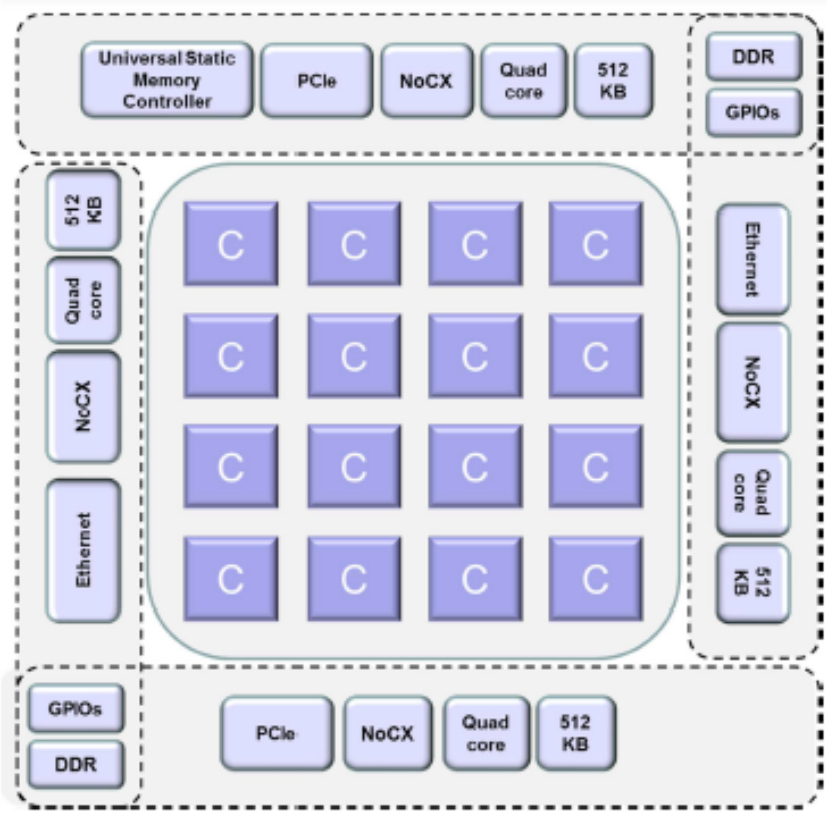
Fonte – Vangal et al. (2007).

sendo 8 processadores e 768 KB em 12 memórias independentes na 32ª linha, conforme ilustra a Figura 9. As memórias de dados do processador são implementadas como dois bancos de 128x16 bits. A comunicação é realizada por uma rede de comutação de circuitos de alto vasão e uma rede complementar menor de comutação de pacotes. Cada um dos 12 módulos de memória independentes contém uma SRAM de 64KB, que serve a dois processadores e suporta 28,4 Gbps de largura de banda de E/S. Em 1.1 volts, os processadores operam até uma média de 1,78 GHz, obtendo uma taxa de computação de chip total máxima de 1,78 trilhão de instruções/seg. Em 0,84 volts, os 1000 núcleos executam 1 trilhão de instruções/seg enquanto dissipam 13,1 watts.

2.1.5 Epiphany-V

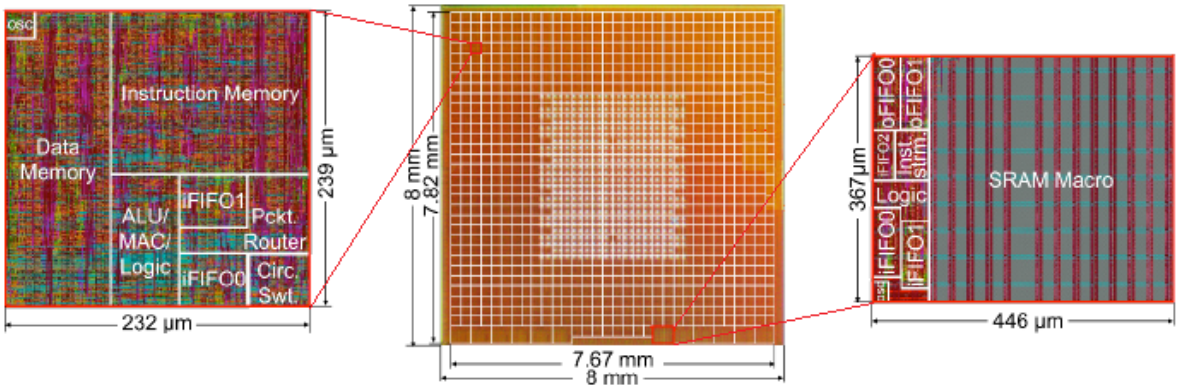
O Epiphany-V (OLOFSSON, 2016) é um MPSoC que contém uma matriz de 1024 processadores, uma NoC com topologia malha 2D e 1024 pinos programáveis para E/S. Ele possui uma organização de memória compartilhada distribuída e é composto por processadores RISC de 64 bits que se comunicam através de uma rede de baixa latência. Cada nó na matriz é um processador RISC completo capaz de executar um sistema operacional. A NoC *emesh* do Epiphany-V consiste em três redes independentes, cada uma com diferentes propósitos: a *rmesh*, que trata dos pacotes de solicitação de leitura;

Figura 8 – Diagrama de blocos do MPSoC MPPA-256, composto por 256 PEs.



Fonte – KALREY (2017).

Figura 9 – MPSoC KiloCore composto por 1000 PEs.

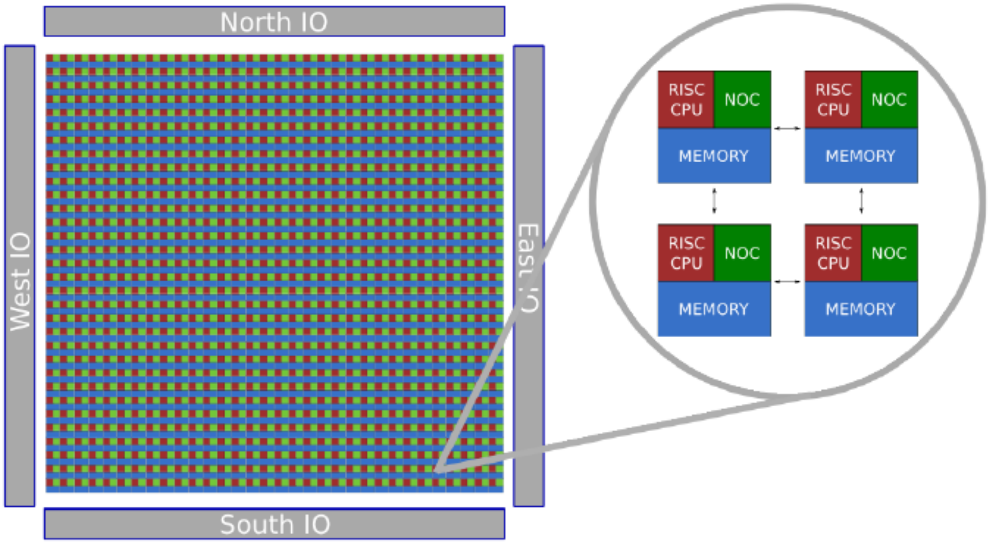


Fonte – Bohnenstiehl et al. (2016).

a *cmesh*, que foca nos pacotes de gravação no chip; e a *xmesh*, que por sua vez cuida dos pacotes de gravação externos ao chip. A tecnologia de fabricação é de 16nm, e o chip possui 4,5 bilhões de transistores. Em comparação com os principais processadores de alto

desempenho, o chip demonstra um ganho de 80x na densidade de área do processador e de 3.6x na densidade de área da memória. Este processador é capaz de executar 2048 operações de ponto flutuante de 64 bits e 4096 operações de ponto flutuante de 32 bits por segundo. A Figura 10 apresenta uma visão geral do Epiphany-V.

Figura 10 – MPSoC Epiphany-V composto por 1024 PEs.



Fonte – Olofsson (2016).

Como ilustrado na Tabela 1, todos os MPSoCs investigados possuem arquitetura homogênea. Todos também possuem uma infraestrutura de comunicação baseada em NoC, o que é visto como tendência neste tipo de sistema. É evidenciado também que o número de PEs em MPSoCs está aumentando com o passar dos anos, como previsto por Borkar (2007). O Kilo-Core com 1000 PEs e o Epiphany-v com 1024 confirmaram esta previsão. Devido ao fato de não ter acesso a estes MPSoCs, o estudo aqui proposto vai se basear em um modelo de MPSoC.

2.2 Plataformas de Simulação

A seguir, são apresentadas algumas propostas de plataformas de MPSoCs acadêmicas. Dentre eles: o MultiFlex na subseção 2.2.1, MCVP-NoC na subseção 2.2.2, MultiX-Simulation na subseção 2.2.3. Por fim, a subseção 2.2.4 apresenta a plataforma HeMPS.

2.2.1 MultiFlex

O sistema MultiFlex (PAULIN et al., 2006) é uma ferramenta de mapeamento de aplicações para plataformas multiprocessadas que integra componentes paralelos heterogê-

Tabela 1 – MPSoCs Investigados.

MPSoC	Arquitetura	Comunicação	Núm. PEs	Operação	Desempenho
TILE-Gx72	Homogênea	NoC Malha	72	1.2 GHz	Não informou
Terascale	Homogênea	NoC Malha	80	4 GHz	1.28 Tflops
MPPA-256	Homogênea	NoC Toro	256	600 MHz	4.5 Gflops
Kilo-Core	Homogênea	NoC Malha	1000	1.78 GHz	1 trilão instruções/s
Epiphany-V	Homogênea	NoC Malha	1024	800 MHz	100 Gflops

neos (hardware ou software) em um ambiente de programação de plataforma homogênea. O MultiFlex é desenvolvido no nível RTL (do inglês *Register Transfer Level*) sintetizável, e permite a modelagem de plataformas através de encapsulamento e abstração de informações. MPSoCs podem ser implementados para diversas fins podendo variar o grau de eficiência.

Dois modelos de programação paralela de alto nível são suportados pelas seguintes ferramentas de mapeamento de plataforma do MultiFlex. O DSOC (*Distributed System Object Component*) que é um modelo de passagem de mensagem orientado a objetos e o SMP (*Symmetrical Multiprocessing*), que é baseado em memória compartilhada. O modelo DSOC tem suporte à computação distribuída heterogênea baseada em troca de mensagens e o SMP tem suporte a *multithreading* acessando memórias compartilhadas. O uso da ferramenta de mapeamento proposta, disponibiliza uma rápida exploração de algoritmos descritos em SMP e DSOC, automaticamente mapeados em arquiteturas paralelas. Uma plataforma desenvolvida foi o StepNP (PAULIN et al., 2004), que inclui modelos de processadores reconfiguráveis, uma arquitetura de comunicação baseada em NoC e periféricos. Os autores utilizaram o StepNP para o mapeamento de uma aplicação de gerenciamento de tráfego de internet a 2.5Gb/s e uma aplicação de vídeo MPEG4 de resolução VGA a 30fps.

2.2.2 MCVP-NoC

A ferramenta de modelagem MCVP-NoC (*Plataforma Virtual de Muitos Núcleos com Suporte de Redes em Chip*) (ZHANG et al., 2013) foi projetada para modelagem de MCSoCs de grande escala e pode ser utilizada na exploração de espaço de projeto, desenvolvimento de software inicial e verificação de sistema. A MCVP-NoC foi construída em SystemC no nível TLM (do inglês, *Transaction Level Modeling*), sendo integrada com Orion2.0 para estimativa de energia e área. Uma camada OVP foi integrada para prover um rápido tempo de simulação dos processadores, memória e barramento.

A ferramenta pode executar código de software real e ainda permite avaliar o de-

sempenho do projeto de um MCSoC sob carga de aplicações reais. Quando uma simulação é finalizada, além das informações de desempenho, informações de potência e área também são disponibilizadas. A integração OVP possibilita um alto desempenho de simulação, análise e validação das plataformas. A MCVP-NoC também suporta a modelagem de uma NoC totalmente personalizada. Os autores modelaram um sistema com 16 núcleos conectado por uma NoC malha como demonstração. Este pôde executar aplicações de decodificação. Os resultados simulados em alto nível de abstração (plataformas virtuais) alcançaram um ganho de 40 vezes em relação à modelagem RTL.

2.2.3 MultiX-Simulation

Roth et al. (2011) apresentaram uma estrutura para multi-resolução e multi-dispositivo de MPSoCs, chamada MultiX-Simulation. A estrutura é baseada na simulação da biblioteca do SystemC, que é fundamentada em um *backbone* de simulação genérico chamado Arquitetura de Alto Nível (do inglês, *High Level Architecture* ou HLA). Esta biblioteca implementa um simulador baseado em eventos onde a representação lógica de uma interconexão de simuladores diferentes é chamada de Federação e inclui vários módulos (Federados) que se comunicam através de uma infra-estrutura de tempo de execução (do inglês, *Runtime Infrastructure* ou RTI).

O RTI oferece diferentes serviços de gerenciamento, que são relevantes para controle de simulação, sincronização e troca de dados. Para interconectar PEs, uma NoC executa um esquema de roteamento XY com canais dedicados com uma lógica FIFO (*First In First Out*), tudo implementado em SystemC. O MPSoC é particionado atribuindo um PE a cada federado e dividindo os canais FIFO, o que resulta em carga de trabalho similar para cada federado. Os autores apresentam resultados comparando quatro aplicativos execução em tamanhos MPSoC de até 6x6, comparando a velocidade do modelo RTL com o modelo misto proposto. Seus resultados indicam uma simulação com aumento de velocidade de até 229 vezes no modelo proposto em relação ao modelo RTL, em um MPSoC com tamanho 6x6.

2.2.4 HeMPS

A HeMPS (*Hermes Multiprocessor System*) (CARARA et al., 2009) é uma plataforma para a geração de MPSoCs baseados em NoC com suporte para mapeamento de tarefas estáticas e dinâmicas. Esta plataforma gera uma descrição do sistema RTL VHDL sintetizável junto com modelos de simulação C/SystemC para processadores e memórias. Os principais componentes de hardware são o processador Plasma e na NoC Hermes. A partir da interconexão de Plasma-IPs (PEs) através da NoC Hermes, são criadas instâncias deste MPSoC. Cada Plasma-IP contém, além de um processador Plasma, uma memória privada, uma interface de rede e um módulo DMA (do inglês, *Direct Memory Acces*). A HeMPS também possui uma memória externa que armazena as tarefas das aplicações a

serem executadas, chamada de repositório de tarefas (*Task Repository*). Além da infra-estrutura de hardware, a estrutura fornece uma infra-estrutura de software que inclui um *microkernel* multitarefa, primitivas de comunicação entre tarefas e suporte para cargas de trabalho dinâmicas. O *microkernel* executa em cada processador permitindo multitarefa no nível do processador.

A HeMPS possui recursos para avaliar o desempenho de aplicações em execução. Ferramentas gráficas permitem a depuração e verificação do sistema, individualizando dados para cada tarefa. Experimentos realizados pelos autores demonstram a eficácia de alguns recursos da plataforma, como modelos de simulação e estratégias de mapeamento dinâmico. Os modelos de simulação reduziram o tempo total de simulação em 91% em comparação com modelos RTL puro, com um erro preciso de ciclo inferior a 3%. O mapeamento dinâmico foi melhor que o mapeamento estático tradicional, devido à quantidade reduzida de simultaneidade de solicitação de mensagens.

A Tabela 2 apresenta os trabalhos investigados. Uma característica comum a todos é a descrição da arquitetura de comunicação, que implementam uma NoC como infraestrutura. Nota-se que nos trabalhos analisados temos diversos níveis de abstração. Zhang et al. (2013) utilizam SystemC e a plataforma virtual OVP. Enquanto que Paulin et al. (2006) descrevem a sua plataforma em RTL sintetizável, um nível mais baixo. Outros trabalhos utilizam o SystemC juntamente com RTL, como Roth et al. (2011) e Carara et al. (2009). A maioria dos trabalhos fez um comparativo dos níveis abstratos. Em comparação com RTL puro, os trabalhos de Zhang et al. (2013) e Roth et al. (2011) foram mais rápidos, 40 e 229 vezes respectivamente. O trabalho de Carara et al. (2009) foi 91% mais rápido. O trabalho de Paulin et al. (2006) não realizou este tipo de comparação. Um item importante analisado foi quanto ao acesso a plataforma. A únicas plataformas que permitem acesso foram dos trabalhos de Carara et al. (2009) e Paulin et al. (2006), sendo a segunda somente após aquisição de uma licença paga.

Tabela 2 – Plataformas MPSoC Investigadas.

Ref.	**Arquitetura Comunicação**	**Modelagem**	**Hierarquia Memória**	**Comparado com RTL**	**Acesso**
Paulin et al.(2006)	Network-on-Chip	RTL-Sintetizável	Distribuída e Compartilhada	Não comparou	Pago
Zhang et al.(2013)	Network-on-Chip	OVP, SystemC e TLM	Distribuída e Compartilhada	40X mais rápido	Fechado
Roth et al.(2011)	Network-on-Chip	SystemC	Distribuída	229X mais rápido	Fechado
Carara et al.(2009)	Network-on-Chip	SystemC e RTL	Distribuída	91% mais rápido	Disponível

2.3 Mapeamento de Tarefas

Após os MPSoCs terem sido apresentados, são listados alguns trabalhos relacionados ao Mapeamento de Tarefas, que é o tópico principal deste trabalho. A subseção 2.3.1 apresenta o trabalho de Carvalho, Calazans e Moraes. A subseção 2.3.2 traz o trabalho de Ost et al. O trabalho de Fattah et al. é apresentado na subseção 2.3.3. A subseção 2.3.4 traz o trabalho de Mendis, Indrusiak e Audsley. Na subseção 2.3.5, é abordado o trabalho de Quan e Pimentel. A subseção 2.3.6 apresenta o trabalho de Huang et al. Na subseção 2.3.7 o trabalho de Ng et al. é abordado. A subseção 2.3.8 traz o trabalho de Singh et al. Por fim, na subseção 2.3.9 é apresentado o trabalho de Seidipiri et al.

2.3.1 Carvalho, Calazans e Moraes.

Carvalho, Calazans e Moraes (2010) investigaram o desempenho de algoritmos de mapeamento para MPSoCs heterogêneos com cargas de trabalho dinâmicas. Neste trabalho, o principal objetivo foi minimizar o congestionamento dentro da NoC através da otimização do uso do canal. Os parâmetros de desempenho observados foram tempo de execução, carga no canal da NoC e latência de pacote. Algumas heurísticas de mapeamento foram propostas pelos autores. A First Free (FF) é usada como referência e apenas para fins de comparação. Essa abordagem começa no endereço (0,0) da NoC e seleciona o primeiro nó livre capaz de executar a tarefa solicitada, percorrendo o MPSoC da esquerda para a direita e em cada coluna de baixo para cima. O mapeamento Nearest Neighbor (NN) é semelhante à estratégia FF já que também não possui avaliação de custo. Ele começa a procurar por um nó livre em torno do nó que faz a solicitação. Sua procura testa todos os vizinhos n-hops, com n variando entre 1 e os limites da NoC.

Enquanto isso, a heurística Path-Load (PL) considera apenas o peso de comunicação dos canais no caminho a ser usado pela tarefa no mapeamento. Um caminho agrupa um conjunto de canais de rede, definidos como uma função do posicionamento da tarefa e do algoritmo de roteamento. Já o mapeamento Best Neighbor (BN) combina a estratégia NN de busca e a abordagem de computação PL. A heurística Minimum Maximum Channel Load (MMC) tenta reduzir a ocupação máxima dos canais da NoC, com o objetivo de evitar o congestionamento na NoC, melhorando o desempenho geral. E por fim, semelhante a MMC, a heurística Minimum Average Channel Load (MAC) visa reduzir a ocupação média dos canais da NoC. Enquanto a heurística MMC minimiza o pico no uso do canal de comunicação, a heurística MAC distribui homogeneamente a carga de comunicação na NoC. As heurísticas propostas alcançam até 31% de carga de canal menor e até 22% menor de latência de pacote que outras heurísticas.

2.3.2 Ost et al.

Ost et al. (2013) desenvolveram uma heurística de mapeamento dinâmico multitarefa e consciente de consumo de energia. Esta emprega um algoritmo de busca otimizado para definir a alocação de posição de tarefas em tempo de execução. A heurística de mapeamento proposta possui uma etapa de pré-mapeamento para agrupar várias tarefas antes de mapear estes grupos para os elementos de processamento. O algoritmo proposto chamado Premap-DN integra a heurística LEC-DN e o método de agrupamento *premap*.

A heurística DN (do inglês, *Dependences Neighborhood*) considera todas as dependências entre tarefas, mapeando a tarefa solicitada o mais próximo possível das tarefas já mapeadas com as quais se comunica, por meio de uma função de custo de proximidade (em número de saltos). A heurística LEC-DN (do inglês, *Low Energy Consumption-Dependences Neighborhood*) amplia o DN, empregando duas funções custo: a proximidade em número de saltos e a energia de comunicação envolvida na transferência/recepção de dados entre tarefas. Este segundo critério é utilizado quando uma determinada tarefa se comunica com pelo menos duas tarefas mapeadas. Nessa situação, a nova tarefa é mapeada mais perto da tarefa com maior volume de comunicação. Quando a tarefa solicitada possui apenas uma tarefa de comunicação já mapeada, o LEC-DN emprega o método de busca NN (do inglês, *Nearest Neighbor*) uma busca em espiral. Se houver mais de uma tarefa de comunicação já mapeada, o LEC-DN procura um PE dentro da caixa delimitadora definida pela posição dessas tarefas.

O objetivo do método de clusterização *premap* é agrupar um conjunto de tarefas de comunicação no mesmo PE, sem reservar recursos para toda a aplicação. Assim, quando uma determinada tarefa é pré-mapeada, apenas sua colocação é reservada. Assim, o mapeamento efetivo das tarefas pré-mapeadas é executado quando solicitado. Ao comparar o Premap-DN com a abordagem CNN (do inglês, *Communication-aware Nearest Neighbor*), o tempo de total de execução é reduzido em até 18,37%, e a redução no consumo de energia chega a 30,36%.

2.3.3 Fattah et al.

Fattah et al. (2014) propõem um algoritmo de mapeamento dinâmico de tarefas, chamado CASqA (do inglês, *Contiguity Adjustable Square Allocation*). Este foca no tempo de execução e no consumo de energia do MPSoC. Neste algoritmo, o nível de *contiguidade* (α) dos processadores alocados é ajustado. Os autores comentam que uma alocação contígua acontece quando as tarefas de uma aplicação são resolvidas em proximidade. Uma alocação estritamente contígua ($\alpha = 0$) diminui a latência e a dissipação de energia da rede e melhora o tempo de execução das aplicações. No entanto, ela limita o *throughput* e aumenta o tempo de resposta das aplicações. Uma alocação não-contígua ($\alpha = 1$) melhora o *throughput*, mas degrada o tempo de execução das aplicações e métricas de

rede.

No algoritmo CASqA, o nível de contiguidade desejada pode ser ajustado em entre soluções estritamente contíguas a ilimitadas não contíguas. Quando uma solução contígua não é possível, a aplicação é mapeada somente se respeitar a contiguidade desejada. Isso estabelece um equilíbrio entre o aumento do tempo de espera das aplicações (mapeamento contíguo) e o desempenho fraco (soluções não-contíguas). Os resultados indicam que o *throughput* máximo não é alcançado na alocação não-contígua ($\alpha = 1$), mas sim quando se utilizam valores intermediários ($0 < \alpha < 1$). Um maior *throughput* (em 3%) com desempenho de rede melhorado pode ser alcançado com o emprego de valores de α intermediários. Mais precisamente, pode-se obter uma queda de até 35% nos custos da rede ajustando o nível de contiguidade em comparação aos casos não-contíguos, enquanto o *throughput* alcançado é mantido constante. Além disso, CASqA fornece pelo menos 32% de economia de energia na rede em comparação com outras abordagens.

2.3.4 Mendis, Indrusiak e Audsley

Mendis, Indrusiak e Audsley (2015) adaptam duas técnicas de gerenciamento de recursos distribuídas existentes para reduzir o atraso de múltiplos fluxos de decodificação de vídeo em uma plataforma MPSoC baseada em NoC. A novidade no trabalho apresentado é dupla. Em primeiro lugar, o algoritmo de balanceamento de carga bio-inspirado introduzido por Caliskanelli et al. (2013) foi adaptado para permitir o remapeamento distribuído dinâmico dentro de uma NoC. A técnica proposta é totalmente distribuída, não contém um gerente global e cada PE periodicamente executa um conjunto leve de regras que lhe confere a capacidade de realizar decisões de reatribuição de tarefas, baseadas em um conhecimento local. Isso introduz redundância controlada no sistema, onde não existe um único ponto de falha ou entidade de gerenciamento.

Em segundo lugar, a abordagem de gerenciamento de recursos baseada em *cluster* de Castilhos et al. (2013) foi adaptada, introduzindo a prioridade da tarefa e a consciência de distância de remapeamento. Os resultados experimentais demonstram que o *remake* bio-inspirado proporciona uma melhoria marginal (2%-4%) na redução do atraso, mas que envolve uma sobrecarga de comunicação 10% a 30% menor. A melhoria foi menor na distribuição da carga de trabalho frente aos esquemas de gerenciamento baseados em *cluster* e centralizados.

2.3.5 Quan e Pimentel

Quan e Pimentel (2015) apresentaram um algoritmo de mapeamento de tarefas híbrido que combina uma exploração de mapeamento estático e uma otimização de mapeamento dinâmico para alcançar uma melhoria geral da eficiência do sistema. Primeiro, o estágio de tempo de projeto realiza a exploração de espaço de projeto para encontrar dois mapeamentos ótimos para cada aplicativo, com os objetivos de maximizar o *through-*

put e maximizar a produção sob um orçamento de energia predefinido. Em segundo lugar, o estágio de tempo de execução otimiza dinamicamente o mapeamento das aplicações em execução, com base nos mapeamentos ótimos das aplicações correspondentes explorados no primeiro estágio (tempo de projeto) e no estado de execução do sistema.

Os autores avaliaram o algoritmo com base em um sistema MPSoC heterogêneo com três aplicações reais. Os resultados experimentais revelam a eficácia do algoritmo proposto, comparando soluções derivadas com as obtidas de vários outros algoritmos de mapeamento de tempo de execução. Em casos de teste com três aplicativos ativos simultaneamente, as soluções de mapeamento derivados da abordagem possuem melhorias médias de desempenho que variam de 45,9% a 105,9% e economia de energia média variando de 14,6% a 23,5%.

2.3.6 Huang et al.

O trabalho de Huang et al. (2015) propôs um algoritmo de mapeamento dinâmico chamado WeNA (*Weighted-based Neighborhood Allocation*), destinado a MPSoCs com comunicação baseada em NoC. Este algoritmo é resultante do aprimoramanto de um algoritmo não contíguo, ou seja, que não levava em consideração a proximidade das tarefas mapeadas nos recursos de um MPSoC. O WeNA leva em consideração esta métrica, além de outras. Este algoritmo tem como proposta diminuir a sobrecarga de comunicação do interprocessador e melhorar o desempenho da rede e da aplicação.

Para isso, os volumes de comunicação entre tarefas são utilizados para organizar a ordem de mapeamento das tarefas pertencentes a mesma aplicação. Um parâmetro para cada tarefa é desenvolvido, este direciona a decisão de mapeamento otimizada em comparação com a vizinhança atual e informações de ocupação. Algumas características de uma tarefa alvo incluem volumes de comunicação e o número de suas tarefas filhas que serão mapeadas posteriormente. Após isso, é realizada uma escolha determinística para o melhor nó, de acordo com o estado atual do sistema. Os resultados experimentais indicam que o algoritmo proporciona melhorias consideráveis em comparação com um algoritmo de última geração, com redução na Distância Média Ponderada de Manhattan (8.06%) e na latência da rede (9,8%). Além disto, obteve uma melhora de 8,3% no consumo de energia, o que significa que o custo da comunicação e as probabilidades de congestionamento foram reduzidas através das melhorias, especialmente sob a situação de ampla gama de volume de comunicação e alta taxa de utilização do sistema.

2.3.7 Ng et al.

Ng et al. (2015) propuseram uma abordagem para otimizar o desempenho e o consumo de energia do sistema em situações de fragmentação. O que acarreta tal fragmentação são as alocações e desalocações frequentes das aplicações que podem deixar núcleos livres e dispersos no MPSoC. Isso ocorre devido à falta de conhecimento em

tempo de projeto dos tempos de conclusão das tarefas. Esta abordagem trata da desfragmentação em tempo de execução que reúne e redimensiona os núcleos dispersos nas proximidades. Os autores propõem também uma métrica de fragmentação que é capaz de avaliar a dispersão dos núcleos livres. Com base nisso, o algoritmo proposto é executado para reunir os núcleos livres dispersos quando a métrica for superior a um determinado limite predefinido. Desta forma, uma região contígua do núcleo livre é formada para facilitar o mapeamento eficiente das aplicações recebidas.

Além disso, o algoritmo também é consciente da natureza das aplicações existentes e minimiza seu impacto no desempenho. A abordagem de desfragmentação proposta reduz o tempo total de execução e o consumo de energia em 42% e 41%, respectivamente, quando comparado com algumas das abordagens existentes. Além disso, segundo os autores uma carga sem muito impacto, que representa apenas menos de 2,6% do tempo de execução geral, é necessária para o processo de desfragmentação.

2.3.8 Singh et al.

Singh et al. (2016) apresentaram uma nova abordagem de mapeamento em tempo de execução, onde a identificação de um mapeamento eficiente para um caso de uso (um conjunto de aplicações ativas simultaneamente) é realizada através da análise de rastreamento de execução *online* das aplicações ativas. Para superar o gargalo de processamento de tempo de execução, uma análise intensiva de computação é executada em tempo de projeto. Os resultados analisados são considerados no tempo de execução para permitir um mapeamento mais eficiente. A análise de rastreamento facilita a identificação rápida do mapeamento, enquanto otimiza o uso de recursos do sistema e a produção das aplicações ativas, levando a um menor consumo de energia.

Ao identificar rapidamente o mapeamento eficiente em tempo de execução, a abordagem proposta supera o gargalo do tempo de exploração para mapeamento de problemas de grande porte, e seu problema de sobrecarga de armazenamento em comparação com as abordagens tradicionais. Nesta abordagem, em média, o tempo de exploração para identificar o mapeamento é reduzido em 14 vezes quando comparado com abordagens de última geração. Além disso, as despesas gerais de armazenamento são reduzidas em 92%. Segundo os autores, a economia de energia e recursos é alcançada juntamente com a identificação de mapeamento de alta qualidade.

2.3.9 Seidipiri et al.

Neste trabalho, Seidipiri et al. (2016) propuseram uma heurística de mapeamento eficiente, chamado RASMAP, para NoCs com topologia malha. O principal objetivo da proposta é reduzir o atraso médio do pacote e a taxa de comunicação nos canais da NoC. Para atingir esse objetivo, as tarefas que trocam dados entre si são colocadas o mais

próximo possível e também as tarefas com maior grau de relação são colocadas em núcleos com alta acessibilidade para a maioria das tarefas.

A proposta primeiro prioriza as tarefas do gráfico de aplicação fornecido com base no tráfego de comunicação total de entrada/saída. Então, uma tarefa com mais tráfego de comunicação é selecionada e mapeado na parte central da topologia de malha, isto é, em um núcleo com os canais de comunicação mais disponíveis no MPSoC. Depois disso, repetidamente, a próxima tarefa é selecionada de uma maneira que tenha mais comunicações com as tarefas já mapeadas. Tal tarefa é mapeada no núcleo cujo grau de vizinhança é proporcional ao grau de ligação das tarefas. O método proposto é comparado com vários algoritmos de mapeamento e resultados indicam que a heurística oferece melhor desempenho e consome menos energia na NoC.

Conforme o resumo apresentado na Tabela 3, a maioria dos trabalhos listados tem como objetivo a otimização de consumo de energia (CE) e tempo de execução (TE), com exceção de Ost et al. (2013), que foca apenas no consumo de energia. Além disso, o tipo de arquitetura homogênea prevalece, já que dentre os trabalhos investigados apenas os trabalhos de Quan e Pimentel (2015) e Carvalho, Calazans e Moraes (2010) fazem uso de uma arquitetura heterogênea. Com relação ao tipo de controle de mapeamento, o controle centralizado é a estratégia adotada na maioria dos trabalhos, mesmo não sendo escalável. O único trabalho que apresenta um controle distribuído é a proposta de Mendis, Indrusiak e Audsley (2015). Os trabalhos apresentam diferentes abordagens. Na abordagem de Remapeamento, após o mapeamento inicial existe a migração de tarefas para outros PEs com o objetivo de melhorar o desempenho do MPSoC. Na Híbrida, se faz uso de mapeamentos em tempo de projeto, que servirão como parâmetro para o mapeamento dinâmico. Na abordagem de Desfragmentação, os PEs livres e dispersos (devido a constante alocação e desalocação) são reunidos e redimensionados. Por fim, na abordagem chamada Evitar Congestionamento, o foco está em diminuir o fluxo de dados nos canais da NoC, a fim de evitar que esses canais de comunicação fiquem saturados.

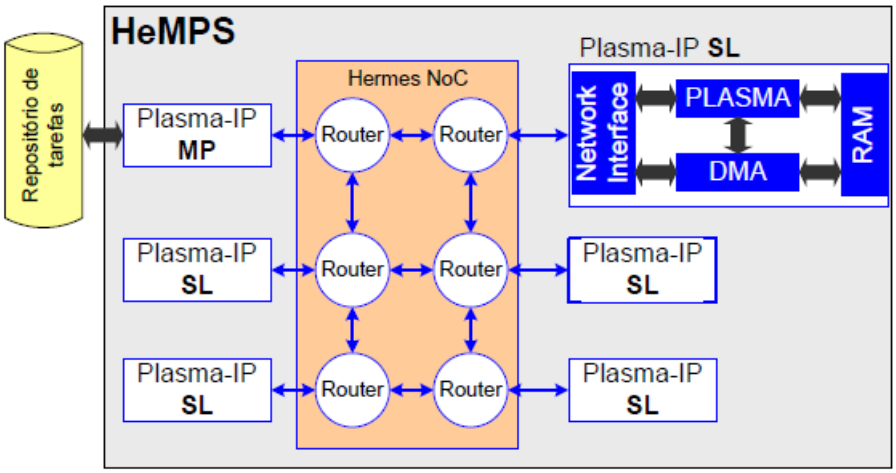
Tabela 3 – Trabalhos Investigados sobre Mapeamento de Tarefas.

Ref.	**Otimização**	**Arquitetura**	**Controle**	**Abordagem**
Carvalho et al.(2010)	TE CE	Heterogênea	Centralizado	Evitar Congest.
Ost et al.(2013)	CE	Homogênea	Centralizado	Evitar Congest.
Fattah et al.(2014)	TE CE	Homogênea	Centralizado	Evitar Congest.
Mendis et al.(2015)	TE CE	Homogênea	Distribuído	Remapeamento
Quan e Pimentel(2015)	TE CE	Heterogênea	Centralizado	Híbrida
Ng et al.(2015)	TE CE	Homogênea	Centralizado	Desfragmentação
Huang et al.(2015)	TE CE	Homogênea	Centralizado	Evitar Congest.
Singh et al.(2016)	TE CE	Homogênea	Centralizado	Híbrida
Seidipiri et al.(2016)	TE CE	Homogênea	Centralizado	Evitar Congest.

3 MPSOC HEMPS

Este Capítulo aborda com mais detalhes a HeMPS (*Hermes Multiprocessor System*) (CARARA et al., 2009), plataforma de MPSoC homogêneo, baseado no processador Plasma e na NoC Hermes que é utilizada neste trabalho para investigar diferentes heurísticas de mapeamento. A partir da interconexão de Plasma-IPs (PEs) através da NoC Hermes são criadas instâncias deste MPSoC. A HeMPS também possui uma memória externa que armazena as tarefas das aplicações a serem executadas, denominada repositório de tarefas (*Task Repository*). Uma instância 2x3 do MPSoC com seus principais componentes de hardware é ilustrada na Figura 11.

Figura 11 – Instância do MPSoC HeMPS utilizando uma NoC 2x3.



Fonte – Carara et al. (2009).

O MPSoC HeMPS possui processamento homogêneo, pois todos os PEs possuem a mesma arquitetura. Isto simplifica a geração dos códigos objetos, pois um sistema heterogêneo implica em diferentes códigos objetos para a mesma aplicação ou tradução dinâmica de código. A utilização de uma NoC justifica-se dada a escalabilidade da mesma e a possibilidade de múltiplas comunicações entre PEs ocorrerem simultaneamente. Cada processador possui uma memória privada que armazena instruções e dados. De acordo com os autores, isto reduz o tráfego de dados no meio de comunicação e não se faz preciso ter memórias cache externas. A comunicação é realizada via troca de mensagens, sendo este o mecanismo natural de comunicação em um sistema que possui memórias distribuídas. A organização de memória é paginada, o que simplifica o processo de mapeamento pois uma página de memória é vista como um recurso de processamento de tarefas e a tarefa pode ser mapeada em qualquer página livre. As subseções seguintes trazem uma descrição dos principais componentes de hardware deste MPSoC.

3.1 Plasma-IP

O elemento de processamento Plasma-IP possui duas instâncias diferentes. O Plasma-IP MP é a instância mestre responsável pela gerência dos recursos do sistema e o Plasma-IP SL é a instância escravo. Pela característica de possuir uma gerência de recursos centralizada, o MPSoC HeMPS contém apenas um Plasma-IP MP.

O processador *Plasma* (PLASMA, 2011) é um processador RISC de código aberto de 32 bits com subconjunto de instruções da arquitetura MIPS. O processador Plasma do HeMPS possui modificações frente ao Plasma original, como a criação de um mecanismo de interrupção, a exclusão de módulos (UART) e a inclusão de novos registradores mapeados em memória. Na *memória privada* (RAM), é onde se encontra o sistema operacional *microkernel* que é executado pelo processador Plasma. Nos Plasma-IP SL a memória é paginada, sendo dividida em páginas de tamanho fixo, onde é realizada a alocação das tarefas.

A *interface de rede* (NI) realiza a interface entre o Plasma e NoC Hermes, sendo responsável pelo envio e recebimento de pacotes na rede. O *módulo de acesso direto a memória* (DMA), dá a possibilidade do processador somente continuar a executar as tarefas sem controlar de forma direta a troca de mensagens com a rede. A principal função do DMA consiste em transferir o código-objeto de tarefas que chegam a interface de rede para a memória do processador e também enviar mensagens de depuração para o processador mestre.

3.2 NoC Hermes

A NoC Hermes faz a interconexão dos elementos de processamento no MPSoC HeMPS, sendo ela parametrizável e com topologia malha 2D. O chaveamento de pacotes utilizado como mecanismo de comunicação emprega a estratégia *wormhole*, e transmite pacotes entre os roteadores na forma de *flits*(32 bits). Os roteadores presentes na NoC possuem *buffers* de entrada, uma lógica de controle compartilhada por todas as portas do roteador, um *crossbar* interno e até cinco portas bidirecionais. As portas são *East*, *West*, *North*, *South* e *Local*, sendo a porta *Local* responsável por estabelecer a conexão entre o roteador e seu núcleo local e as demais portas ficando a cargo de conectar o roteador local ao roteador vizinho. O roteador da NoC Hermes utiliza a arbitragem *round-robin*, com uma política de prioridades dinâmicas, o que proporciona um serviço mais justo que um esquema com prioridade estática. O algoritmo de roteamento dos pacotes na rede é o XY, onde primeiramente os envios a rede são feitos horizontalmente até chegar na coordenada X do roteador de destino, e depois percorre a rede verticalmente até finalmente chegar no roteador destino.

3.3 Repositório de Tarefas

Todas as tarefas são modeladas como um grafo no MPSoC HeMPS, onde os vértices representam tarefas e as arestas representam as comunicações entre os tarefas das aplicações. Este segue o mesmo modelo apresentado na Figura 4 da introdução. As tarefas que não possuem nenhuma dependência de recepção de dados são tidas como tarefas iniciais. Quando uma aplicação inicia, a heurística de mapeamento utilizada carrega no MPSoC somente as tarefas iniciais. As demais tarefas são inseridas em tempo de execução no sistema, em função das requisições de comunicação e dos recursos disponibilizados. O *repositório de tarefas* é um espaço de memória que fica externo ao MPSoC e que contém o código-objeto de todas as tarefas que serão executadas no sistema . Em suas primeiras posições, o repositório de tarefas contém os descritores de cada tarefa, com informações como identificador único, posição inicial no repositório, tamanho da tarefa e outras informações.

3.4 Interface HeMPS

A HeMPS dispõe de uma completa ferramenta de depuração que permite avaliar as simulações, verificando o comportamento das tarefas, serviços e comunicação. Entre suas principais funções encontram-se: (i) acompanhamento da execução em tempo de simulação, (ii) visualização do mapeamento das tarefas, (iii) verificação dos percentuais de utilização de cada link entre um par de roteadores, (iv) visão geral da distribuição de serviços executados por cada PE, dentre outras.

A HeMPS gera uma visualização gráfica do MPSoC, como apresentado na Figura 12. O PE mestre está representado em laranja, e em verde os escravos. As setas que compõem a representação gráfica indicam um enlace entre dois roteadores (setas maiores) ou entre um roteador e o PE (setas menores nos cantos superior esquerdo). Os valores representados com o símbolo de % em cada enlace, representam a porcentagem de largura de banda utilizada para aquele enlace. A HeMPS também permite visualizar o mapeamento e o estado das tarefas no MPSoC. A Figura 13 apresenta uma tela onde o mapeamento de aplicações pode ser visualizado. Pode-se perceber uma dada aplicação mapeada em um MPSoC 3x3, sendo que as tarefas da aplicação estão distribuídas pelos seus recursos.

A razão que motivou a escolha da HeMPS como plataforma a ser utilizada neste trabalho, está no fato de ser totalmente aberta para utilização, estando seu código fonte disponível para download na rede. A HeMPS também permite que sua estratégia de mapeamento de tarefas seja configurada, um dos objetivos deste trabalho. As outras plataformas de MPSoC investigadas não dispõem destes recursos.

Dada a plataforma HeMPS, modificações no código fonte terão de ser realizadas para o andamento do trabalho. Estas serão discutidas nos capítulos a seguir.

Figura 12 – Visualização gráfica do MPSoC na HeMPS.

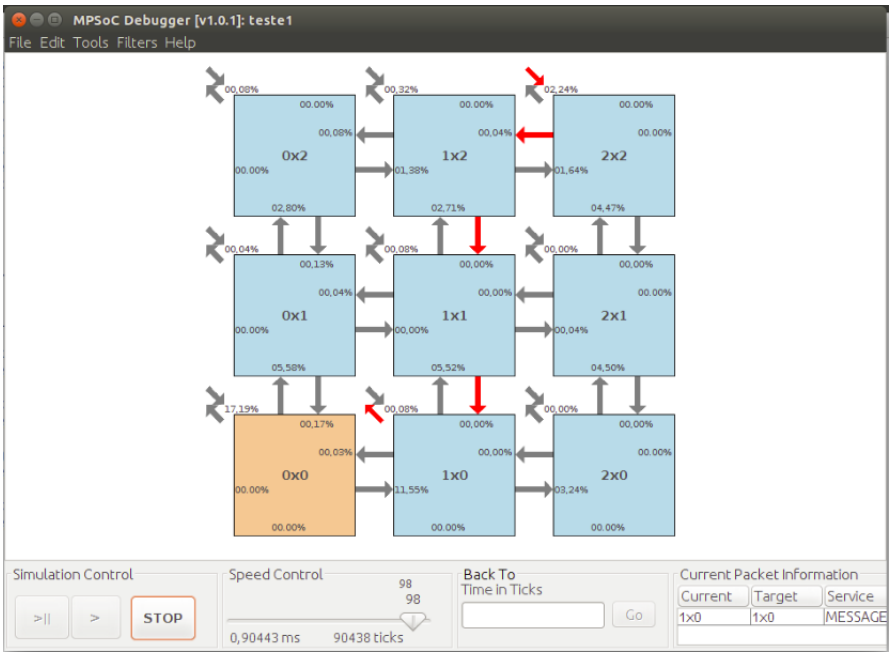
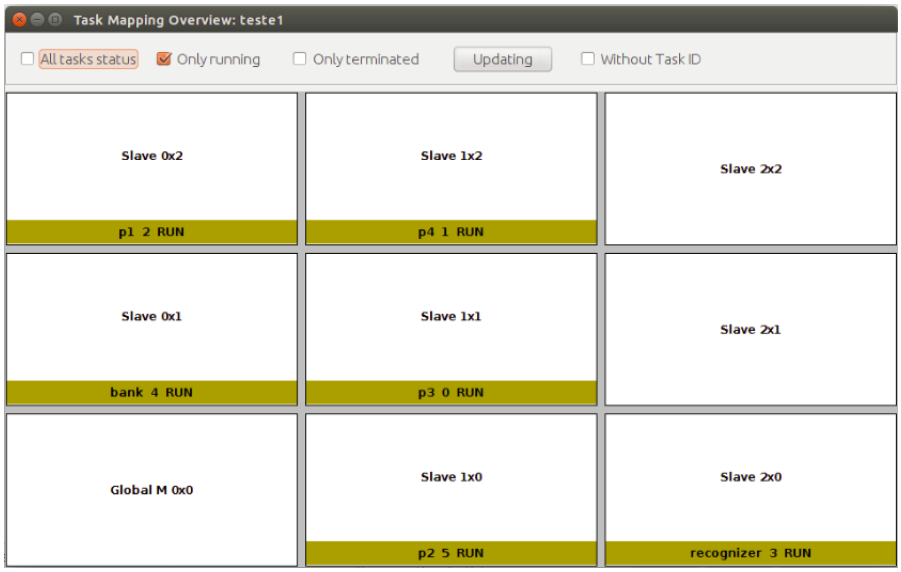


Figura 13 – Visualização gráfica do mapeamento na HeMPS.



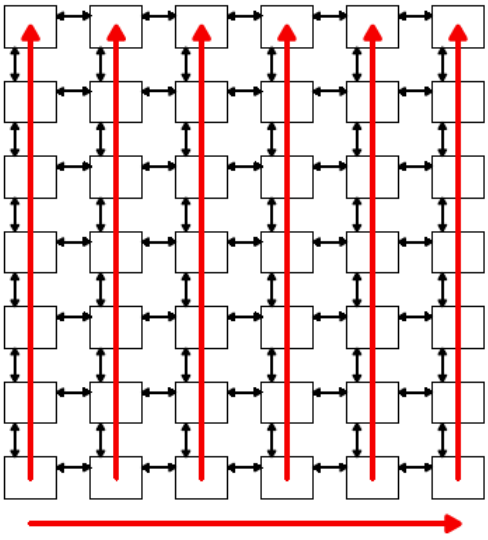
4 HEURÍSTICAS DE MAPEAMENTO

Neste Capítulo, as heurísticas de mapeamento propostas são discutidas. Para cada heurística, um pseudocódigo da sua implementação é apresentado, acompanhado da explicação de seu funcionamento e de sua complexidade. Na seção 4.1 é apresentada a heurística *First Free* (FF) e a seção 4.2 aborda a *Nearest Neighbor* (NN). A seção 4.3 detalha a implementação de estruturas adicionais na plataforma HeMPS. A seção 4.4 apresenta a *Path Load* (PL) e por fim, a seção 4.5 aborda a heurística *Best Neighbor* (BN).

4.1 First Free (FF)

O algoritmo *First Free* (FF) (CARVALHO, 2009) seleciona o primeiro recurso livre para mapear uma nova tarefa, de acordo com os tipos de recursos e tarefas, sem considerar métricas de desempenho. A escolha deste recurso é diretamente relacionada ao caminho de procura empregado na heurística. Este caminho se dá a partir da esquerda para direita, buscando coluna por coluna, de baixo para cima. Este algoritmo é bastante simples. Ele inicia analisando cada PE do sistema na ordem do caminho apresentado na Figura 14. Para cada PE, é verificado se ele está livre. Em caso afirmativo, este PE é o escolhido para o mapeamento, sendo terminada a execução do algoritmo. Caso contrário, prossegue-se a busca por um PE livre. Como o FF não possui nenhuma função custo, ele tende a resultar em mapeamentos não otimizados, apresentando resultados ruins com relação a ocupação dos canais.

Figura 14 – Caminho da procura por recurso realizado por *First Free*.



Fonte – Carvalho (2009).

Um pseudocódigo da implementação desta heurística é apresentado na Figura 15. Pode-se notar no pseudocódigo que o laço PARA da linha 2 é, no pior caso, executado para todos os PEs do MPSoC. Generalizando para um MPSoC com dimensões iguais, quando x é igual a y, então x^2 iterações serão realizadas. Isso faz com que sua complexidade seja $O(x^2)$.

Figura 15 – Pseudocódigo da heurística First Free.

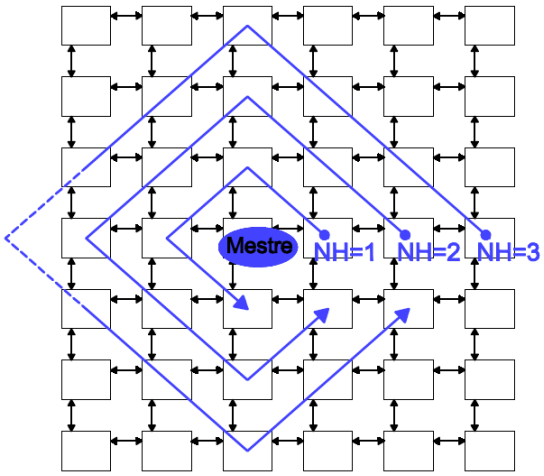
```
1  //percorre todos os PEs
2  PARA cada PE no MPSoC
3
4     //seleciona o primeiro PE livre
5     SE ( PE.estado == LIVRE ) ENTÃO
6        Retorna(PE)
7     FIM SE
8
9  FIM PARA
```

4.2 Nearest Neighbor (NN)

Assim como FF, o algoritmo *Nearest Neighbor* (NN) (CARVALHO, 2009) também não considera os congestionamentos quando decide o mapeamento de uma dada tarefa. Contudo, a definição do seu caminho de procura privilegia a proximidade entre as tarefas comunicantes. Conforme ilustrado na Figura 16, a procura por um recurso livre se dá a partir da posição da tarefa mestre (a tarefa que solicitou o mapeamento). A partir desta posição, é seguido um caminho circular, onde os vizinhos são testados de acordo com o número de saltos (ou *hops* NH) necessários para a comunicação. Assim, primeiramente são verificados os vizinhos localizados a um *hop* de distância da tarefa mestre. Neste caso, quando um primeiro recurso livre é encontrado, é nele que será mapeada a nova tarefa. Caso não sejam encontrados recursos livres a um *hop* de distância, são testados os vizinhos localizados a dois *hops* de distância, depois com três *hops* e assim por diante, até chegarmos aos limites da NoC. Mesmo também não possuindo uma função custo, a NN deve gerar resultados melhores que aqueles obtidos com FF, já que aplica um caminhamento mais eficiente.

Como pode-se observar na Figura 17, na linha 8 do código é criada uma lista com todos os vizinhos da tarefa mestre que estão em um mesmo número de saltos de distância (numSaltos na linha 3). A lista gerada possui os IDs de cada PE, de modo que é arranjada para que a procura seja de forma circular à tarefa mestre. No bloco PARA (na linha 11), todos os elementos de processamento que estão na lista criada são procurados e de forma idêntica ao teste em FF, o PE é retornado se estiver livre. A cada iteração do bloco ENQUANTO da linha 5, a distância é incrementada em um salto (na linha 18).

Figura 16 – Caminho da procura por recurso realizado por *Nearest Neighbor*.



Fonte – Carvalho (2009).

O bloco ENQUANTO (na linha 5) pode ser executado até o número máximo de saltos, e o bloco PARA (na linha 11) pode ser executado até o número máximo de vizinhos possíveis. Para encontrar o número máximo de iterações do método, o produto entre o número máximo de saltos e o número máximo de vizinhos é realizado. Generalizando para o caso de um MPSoC com dimensões iguais, quando x é igual a y, NN possui complexidade $O(x^2)$.

Figura 17 – Pseudocódigo da heurística Nearest Neighbor.

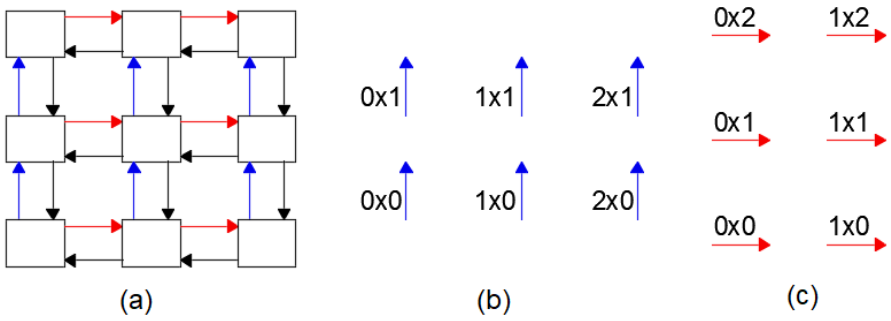
```
1   PEmestre = PE da tarefa mestre
2   PE = PEmestre
3   numSaltos = 1  // inicia com um salto de distância
4
5   ENQUANTO (1) FAÇA
6
7      //cria lista de vizinhos que estão a numSaltos
8      criaListaVizinhos ( PEmestre, numSaltos )
9
10     //percorre a lista de vizinhos
11     PARA cada PE em listaVizinhos
12
13        SE ( PE.estado == LIVRE ) ENTÃO
14           Retorna(PE)
15        FIM SE
16
17     FIM PARA
18
19     numSaltos = numSaltos + 1  //incrementa a distância
20
21  FIM ENQUANTO
```

4.3 Função Custo

As duas primeiras heurísticas apresentadas, FF e NN, não utilizam nenhuma função custo ao decidir o mapeamento das tarefas. Diferentemente, as heurísticas que se seguem consideram um fator custo quando decidem o mapeamento. Com uma investigação mais apurada da plataforma HeMPS, observou-se que a mesma não dispunha de uma implementação de canais de comunicação em um nível mais alto de abstração. Como o fator custo baseia-se no custo de comunicação nos canais, algumas estruturas tiveram de ser adicionadas a plataforma.

A implementação consistiu de estruturas de dados tipo matrizes. A Figura 18(a) traz um exemplo de visualização dos canais de comunicação para um MPSoC com dimensão 3x3. Pode-se notar que existem quatro tipos de canais, os canais na direção "para cima", "para baixo", "direita" e "esquerda". Para cada direção de comunicação foi implementada uma matriz. A Figura 18(b) apresenta a matriz para os canais com direção "para cima". Para os canais com direção "para baixo" a implementação segue a mesma lógica, apenas com direção oposta. A Figura 18(c) traz um exemplo de matriz para os canais com direção "direita", sendo que a matriz de direção "esquerda" segue a mesma lógica mas com direção oposta.

Figura 18 – Representação dos Canais de Comunicação.



Após esta implementação inicial, foram desenvolvidas funções para manipulação dos canais de comunicação. A função *adicionarPeso( )* adiciona os pesos de comunicação nos canais, a função *removerPeso( )* remove os pesos assim que a aplicação finaliza a sua execução. A função *calcularPeso( )* realiza o cálculo dos pesos dos canais de uma dado PE a outro e, será utilizada como função custo neste trabalho. As três funções possuem uma estrutura idêntica, apenas diferem em suas ações sobre os canais.

Ao observar o pseudocódigo da função *calcularPeso( )* na Figura 19, pode-se notar que nas linhas 9, 15, 23, 33, 39, 47, 60 e 67, se dá a diferenciação das três funções. Nesta função em questão, o peso de cada canal no caminho de comunicação das tarefas é somado em uma variável, calculando-se assim o peso total do caminho.

Figura 19 – Pseudocódigo da função *calcularPeso( )*.

```
1   xMestre, yMestre = coordenadas x,y do PE mestre
2   xAlvo,  yAlvo = coordenadas x,y do PE alvo
3   pesoCaminho = 0  //peso inicial
4
5   //se mestre na esquerda do escravo
6   SE ( xMestre < xAlvo ) ENTÃO
7
8         PARA cada canal de comunicação em x
9             pesoCaminho = pesoCaminho + pesoCanal
10
11            // se mestre na esquerda e acima do escravo
12            SE ( yMestre > yAlvo  e  x == xAlvo ) ENTÃO
13
14                PARA cada canal de comunicação  em y
15                    pesoCaminho = pesoCaminho + pesoCanal
16                FIM PARA
17
18            FIM SE
19            //se mestre na esquerda e abaixo do escravo
20            SE ( yMestre < yAlvo e  x == xAlvo ) ENTÃO
21
22                PARA cada canal de comunicação em y
23                    pesoCaminho = pesoCaminho + pesoCanal
24                FIM PARA
25
26            FIM SE
27        FIM PARA
28  FIM SE
29  //mestre na direita do escravo
30  SE ( xMestre > xAlvo ) ENTÃO
31
32        PARA cada canal de comunicação em x
33             pesoCaminho = pesoCaminho + pesoCanal
34
35            //se mestre na direita e acima do escravo
36            SE ( yMestre > yAlvo  e  x == xAlvo ) ENTÃO
37
38                PARA cada canal de comunicação  em y
39                    pesoCaminho = pesoCaminho + pesoCanal
40                FIM PARA
41
42            FIM SE
43            //se mestre na direita e abaixo do escravo
44            SE ( yMestre < yAlvo e  x == xAlvo ) ENTÃO
45
46                PARA cada canal de comunicação em y
47                    pesoCaminho = pesoCaminho + pesoCanal
48                FIM PARA
49
50            FIM SE
51        FIM PARA
52  FIM SE
53  //se mestre na mesma coluna do escravo
54  SE ( xMestre == xAlvo ) ENTÃO
55
56     //se mestre acima do escravo
57     SE ( yMestre > yAlvo ) ENTÃO
58
59        PARA cada canal de comunicação em y
60           pesoCaminho = pesoCaminho + pesoCanal
61        FIM PARA
62     FIM SE
63     //se mestre abaixo do escravo
64     SE ( yMestre < yAlvo ) ENTÃO
65
66        PARA cada canal de comunicação em y
67           pesoCaminho = pesoCaminho + pesoCanal
68        FIM PARA
69     FIM SE
70  FIM SE
```

Para a função *adicionarPeso( )*, estas linhas servirão para fazer um cálculo de adição de peso em cada canal de comunicação no caminho comunicante, observando-se o peso atual do canal. As mesma linhas aplicam-se à função *removerPeso( )*, que remove os pesos com um cálculo de subtração.

Nas linhas 1 e 2 o algoritmo tem como entrada as coordenadas x,y do PE mestre e do PE alvo, assim chamados o PE onde a tarefa solicitante está e o PE candidato a receber a tarefa comunicante. A partir disso, segue uma série de testes para detectar a posição de um PE em relação ao outro. Três testes principais são realizados para descobrir onde o PE mestre está localizado, se está mais à esquerda, mais à direita ou na mesma coluna do PE alvo em relação ao eixo x. O teste da linha 6 verifica se o PE mestre está mais à esquerda. Se a cláusula de teste for verdadeira, o peso em relação ao eixo x é computado. O teste aninhado na linha 12 verifica se o PE está em uma posição mais acima do PE alvo, e o teste aninhado da linha 20 verifica se está mais abaixo, observando-se o eixo y. O algoritmo só faz o cálculo dos pesos em relação ao eixo y ao alinhar os PEs na mesma coluna. Na linha 30, o teste verifica se a posição do PE mestre está mais à direita e os testes aninhados seguem a mesma lógica anterior. Por fim, o teste da linha 54 verifica se os PEs mestre e PE alvo estão inicialmente na mesma coluna. Para uma análise de complexidade, pode-se observar que o algoritmo é executado no pior caso para o maior caminho de comunicação possível. De acordo com o algoritmo de roteamento xy, a distância máxima entre duas tarefas em um MPSoC malha é igual a maior dimensão, que logo será um valor de tamanho x. Portanto a complexidade deste algoritmo é $O(x)$.

4.4 Path Load (PL)

Path Load (PL) (CARVALHO, 2009) possui um caminho de procura idêntico à *First Free*, porém o que difere as duas heurísticas é o fato que a PL possui uma função custo. Esta heurística verifica o peso de comunicação que o mapeamento em determinado recurso provocaria nos canais de comunicação. A PL calcula o custo para cada recurso livre, realizando o somatório dos pesos dos canais do PE da tarefa mestre (ou seja, aquela que solicitou o mapeamento), até um PE alvo para a tarefa escrava (aquela que foi solicitada). O PE selecionado é aquele que tem o custo mínimo de somatório dos canais.

Como pode ser observado na Figura 20, no bloco PARA (da linha 6) os canais que compõem o caminho de comunicação entre o PE da tarefa mestre e o PE alvo são considerados. Para cada canal no caminho de comunicação, o cálculo do peso dos canais no caminho de comunicação é realizado na linha 12 do algoritmo. O mapeamento selecionado será aquele que apresentar o menor somatório da ocupação dos canais, conforme o bloco SE (na linha 14). No pior caso, o bloco PARA da linha 6 pode ser executado para todos os PEs (x^2). Como já discutido, a função *calcularPeso( )* tem complexidade $O(x)$. Sendo assim, o número máximo de possíveis iterações é o produto das iterações do bloco PARA

da linha 6 com as x iterações da função *calcularPeso( )*. Isto confere complexidade $O(x^3)$ a este método.

Figura 20 – Pseudocódigo da heurística Path Load.

```
1  PEmestre = PE da tarefa mestre
2  peso = 1000000 //peso inicial alto
3  pesoTmp = 0
4
5  //percorre todos os PEs
6  PARA cada PE no MPSoC
7
8     //seleciona o primeiro PE livre
9     SE ( PE.estado == LIVRE ) ENTÃO
10
11       //calcula o peso dos canais de comunicação
12       pesoTmp = calcularPeso( PEmestre, PE )
13
14       SE ( pesoTmp < peso ) ENTÃO
15          peso = pesoTmp
16          PEalvo = PE
17       FIM SE
18
19    FIM SE
20
21 FIM PARA
22 Retorna(PEalvo)
```

4.5 Best Neighbor (BN)

A heurística *Best Neighbor* (BN) (CARVALHO, 2009) combina a estratégia de busca em espiral de NN com a abordagem de computação de peso dos canais em PL. De modo semelhante à NN, uma lista de PEs alvo vizinhos ao PE da onde se encontra a tarefa mestre é criada. Isso evita o cálculo de todas as soluções de mapeamento possíveis na malha do MPSoC como faz a heurística PL, fazendo com que o tempo de execução da heurística seja reduzido. A BN seleciona o melhor vizinho de acordo com os cálculos de canais em PL, em vez do primeiro vizinho livre como em NN.

De acordo com a Figura 21, podemos verificar nas linhas 11 e 16 que o somatório da ocupação dos canais no caminho de comunicação é computado apenas para uma dada lista de vizinhos, toda vez que for preciso haver uma nova iteração. Como pode ser visto, na NN é a lista de vizinhos que permite o andamento circular na procura de uma posição para a tarefa. O bloco ENQUANTO (na linha 5) pode ser executado no pior caso para todas as distâncias possíveis, $O(x)$. Na linha 11, o bloco PARA é executado para a lista de vizinhos cujo valor máximo é o mesmo que em NN, $O(x)$. A função *calcularPeso( )* na linha 16 é executada apenas uma única vez para cada PE na lista de vizinhos, e por isso é desconsiderado no cálculo do número de iterações máximo a serem realizadas por

BN. Sendo assim, BN apresenta a mesma complexidade que NN, $O(x^2)$. Pode-se esperar porém, que o método de cálculo de PL cause um certo impacto no desempenho de BN.

Figura 21 – Pseudocódigo da heurística Best Neighbor.

```
1   PEmestre = PE da tarefa mestre
2   numSaltos = 1  // inicia com um salto de distância
3   peso = 1000000 //peso inicial alto
4
5   ENQUANTO (1) FAÇA
6
7      //cria lista de vizinhos que estão a numSaltos
8      criaListaVizinhos ( PEmestre, numSaltos )
9
10     //percorre a lista de vizinhos
11     PARA cada PE em listaVizinhos
12
13        SE ( PE.estado == LIVRE ) ENTÃO
14
15           //calcula o peso dos canais de comunicação
16           pesoTmp = calcularPeso( PEmestre, PE )
17
18           SE ( pesoTmp < peso ) ENTÃO
19              peso = pesoTmp
20              PEalvo = PE
21           FIM SE
22
23        FIM SE
24
25     FIM PARA
26
27    //se chegou a calcular então achou o melhor PE
28     SE ( peso < 1000000 ) ENTÃO
29        Retorna(PEalvo)
30     FIM SE
31
32     numSaltos = numSaltos + 1  //incrementa a distância
33  FIM ENQUANTO
34  Retorna(PEalvo)
```

A implementação das heurísticas de mapeamento investigadas estão disponíveis em: <http://bit.ly/codigoTCCEzequiel>.

5 RESULTADOS

Este Capítulo apresenta os resultados da investigação do comportamento de diferentes heurísticas de mapeamento sobre a plataforma HeMPS. Na seção 5.1 são apresentadas as aplicações utilizadas neste trabalho. A seção 5.2 apresenta a configuração do MPSoC utilizado e detalha os casos de teste criados para comparar as heurísticas. As métricas para avaliações das diferentes simulações são descritas na seção 5.3. Por fim, a análise dos resultados obtidos é apresentada na seção 5.4.

5.1 Aplicações Alvo

Na plataforma HeMPS, é papel do programador particionar uma dada aplicação paralela em diferentes tarefas descritas na linguagem de programação C, onde cada tarefa tem sua própria função. Como já descrito anteriormente, as aplicações podem ser representadas a partir de grafos dirigidos, onde os vértices representam as tarefas e as arestas representam o fluxo de dados entre elas. Em termos de software, o fluxo de dados representados pelas arestas é implementado a partir das primitivas *Send( )* e *Receive( )*. Neste trabalho são utilizadas três aplicações disponibilizadas na plataforma HeMPS: *Dynamic Time Warping* (DTW), *Moving Picture Experts Group* (MPEG) e SYNTHETIC. O grafo de cada uma dessas aplicações pode ser visualizado na Figura 22.

A Figura 22(a) apresenta a aplicação DTW que é utilizada para encontrar o alinhamento não-linear ótimo entre duas sequências de valores numéricos, podendo ser utilizado para alinhar qualquer tipo de dado que obedeça uma ordem temporal. A Figura 22(b) traz a MPEG que é uma aplicação de áudio e vídeo. A Figura 22(c) apresenta a SYNTHETIC, uma aplicação sintética que simula o envio e recebimento de um conjunto de dados. A DTW representa o conjunto de aplicações que possuem alta taxa de comunicação entre as tarefas. Já a aplicação MPEG representa as aplicações com carga elevada de processamento. Por fim, a aplicação SYNTHETIC não possui carga de processamento mas permite manipular a taxa de comunicação entre as tarefas.

5.2 Configuração do MPSoC e Casos de Teste

Para todos os casos de teste a seguir, a configuração do MPSoC utilizado teve as seguintes características: um cluster com dimensão 5x5, totalizando vinte e cinco PEs, sendo que o PE 0x0 não recebe nenhuma tarefa pois tem como função ser o gerenciador de toda comunicação no MPSoC.

Foram realizados cinco casos de teste. Conforme apresentado na Tabela 4, para os casos de teste 1, 2 e 3, o MPSoC recebeu apenas uma aplicação por vez. Já nos casos de teste 4 e 5, respectivamente duas e três aplicações foram simuladas executando concorrentemente no MPSoC.

Figura 22 – Aplicações DTW, MPEG e SYNTHETIC.

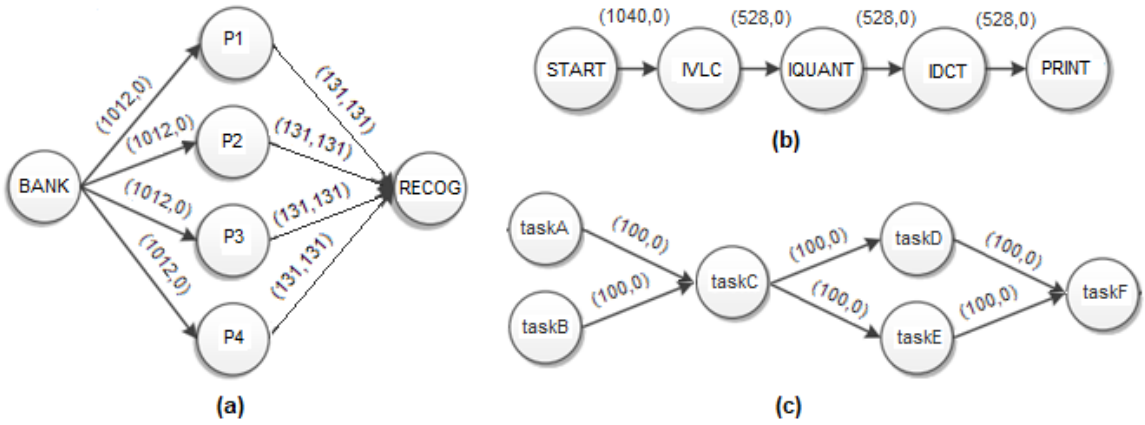


Tabela 4 – Casos de Teste.

ID	**Aplicação**
1	DTW
2	MPEG
3	SYNTHETIC
4	DTW e MPEG
5	DTW, MPEG e SYNTHETIC

Cada caso de teste foi simulado para cada uma das heurísticas: FF, NN, PL e BN. É importante ressaltar que mesmo havendo a mudança da heurística, sempre para a primeira tarefa de cada aplicação, a procura por um recurso livre utilizou a heurística FF e as demais tarefas seguiram com a heurística corrente.

Foram realizadas vinte simulações, ou seja, uma simulação por caso de teste x heurística. O uso do resultado de apenas uma simulação se justifica pelo fato do tempo de simulação fornecido pela plataforma HeMPS utilizar como referência o número de ciclos de relógio. Assim, se não há alteração no ciclo de relógio, o tempo de simulação em diferentes simulações não muda.

5.3 Métricas de Avaliação

Como em geral mais de um mapeamento é possível no sistema, faz-se necessário avaliar a qualidade de um dado mapeamento de acordo com um dado critério. Só assim, é possível escolher dentre as possibilidades de mapeamento, aquela que atende melhor aos requisitos do sistema. Além disso, o impacto que o mapeamento gera no sistema também deve ser considerado, visto que heurísticas lentas irão degradar o desempenho do

sistema como um todo. Ou seja, deve ser observado um compromisso entre a qualidade do mapeamento gerado e o tempo gasto para computá-lo. Neste sentido, faz-se necessário o emprego de um conjunto de métricas para comparação dos resultados obtidos. A seguir, as métricas utilizadas neste trabalho são discutidas. São elas:

- **Tempo de Mapeamento**: é afetado pela complexidade do algoritmo de mapeamento. Mesmo que o algoritmo gere um bom resultado, ele pode demorar para tomar uma decisão. Caso isso ocorra, o tempo de mapeamento de uma aplicação aumenta. Uma boa estratégia de mapeamento deve minimizar o tempo de mapeamento. Neste trabalho, esta métrica é apresentada sempre em milissegundos (ms);
- **Tempo de Simulação** : é fornecido pela plataforma HeMPS e consiste no tempo total de simulação de um caso de teste, englobando o tempo de mapeamento das tarefas. Neste trabalho, esta métrica também é apresentada sempre em milissegundos (ms);
- **Ocupação dos canais da NoC**: representa o volume de comunicação em cada um dos canais. Como visto na Figura 5 da seção 1.2, quando as tarefas de uma aplicação se comunicam e são mapeadas próximas umas das outras, a tendência é que os canais de comunicação da NoC sejam menos exigidos. Em contraste, os canais de comunicação tendem a ficar com cargas maiores de comunicação quando um mapeamento aleatório é utilizado. Além disso, em um mapeamento com mais qualidade, um número menor de canais de comunicação é exigido. Neste trabalho, esta métrica é apresentada sempre em número de *flits* (32 bits), que são unidades em que um pacote de comunicação é transmitido entre os roteadores da NoC.

5.4 Análise

Nesta seção são apresentados os dados obtidos com as simulações de todos os casos de teste da Tabela 4, bem como uma análise dos resultados. Na subseção 5.4.1 temos os resultados para os casos de teste 1, 2 e 3; e na subseção 5.4.2 temos os resultados para os casos de teste 4 e 5.

5.4.1 Casos de teste 1, 2 e 3

Nesta subseção é realizada uma análise em conjunto dos casos de teste 1, 2 e 3 que simulam o mapeamento e a execução de somente uma aplicação.

De acordo com a Figura 23 em (a), (b) e (c), podemos observar o desempenho das diferentes heurísticas para cada caso de teste. Analisando primeiramente o tempo de simulação dos casos de teste, observa-se a variação desta métrica para cada aplicação. Nota-se na Figura 23 que a aplicação MPEG possui a maior média de tempo (9,35947 ms), seguido de DTW (7,61622 ms) e da SYNTHETIC (1,41336 ms). A aplicação MPEG

possui um tempo de simulação maior comparada as demais por causa de sua carga de processamento maior, mesmo com DTW possuindo o maior número de tarefas das três aplicações. Já o tempo de mapeamento apresentou praticamente a mesma média para todas as aplicações, pois não é afetado por carga de processamento e sim pelas heurísticas de mapeamento.

Comparando as heurísticas de mapeamento, a PL teve o pior desempenho que as demais em relação ao tempo de mapeamento, alcançando uma média de 0,88953 ms. A heurística FF teve um desempenho 29,92% melhor que PL, seguido de NN com 29,83% e BN com 24,21%. O pior desempenho de PL em relação as demais heurísticas deve-se a complexidade de seu algoritmo, $O(x^3)$, já explicado anteriormente na seção 4.4. A FF obteve o melhor tempo pois possui um algoritmo muito mais simples que as demais heurísticas. A heurística BN não alcançou o melhor resultado nesta métrica porque utiliza a mesma função custo que PL, o que afeta o seu desempenho.

Como o tempo de simulação é afetado pelo tempo de mapeamento, é possível notar o mesmo padrão de desempenho para as duas métricas. A heurística PL teve o pior desempenho com média de tempo de simulação de 6,32565 ms, seguido por NN com um tempo 4,58% melhor que PL, FF com 4,54% e BN com 3,26%. A Tabela 5 e a Tabela 6 apresentam as médias de desempenho para os casos de teste 1, 2 e 3.

Figura 23 – Desempenho dos casos de teste 1, 2 e 3.

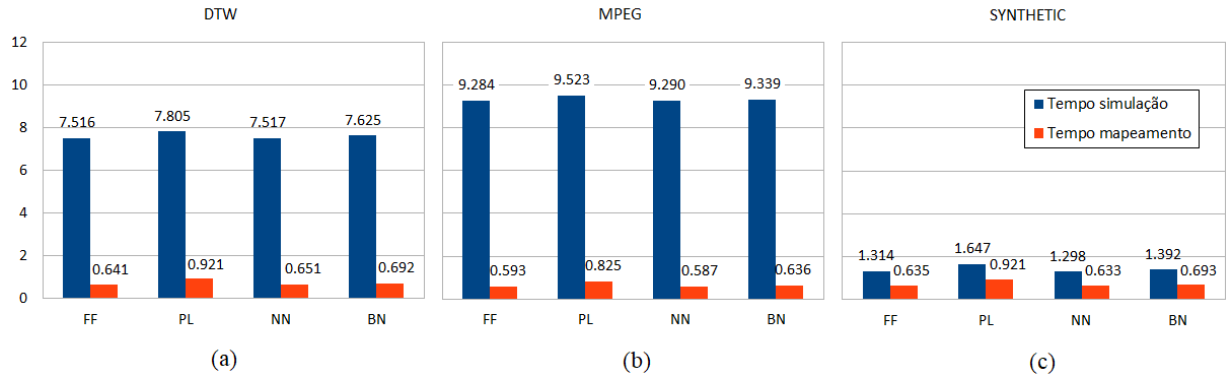


Tabela 5 – Média de desempenho das aplicações para os casos de teste 1, 2 e 3.

Aplicação	**Tempo Simulação(ms)**	**Tempo Mapeamento(ms)**
DTW	7,61622	0,72665
MPEG	9,35947	0,66072
SYNTHETIC	1,41336	0,72101

A Figura 24 apresenta os resultados para os pesos de comunicação nos canais da NoC para as três aplicações alvo. Podemos observar que a aplicação DTW possui as

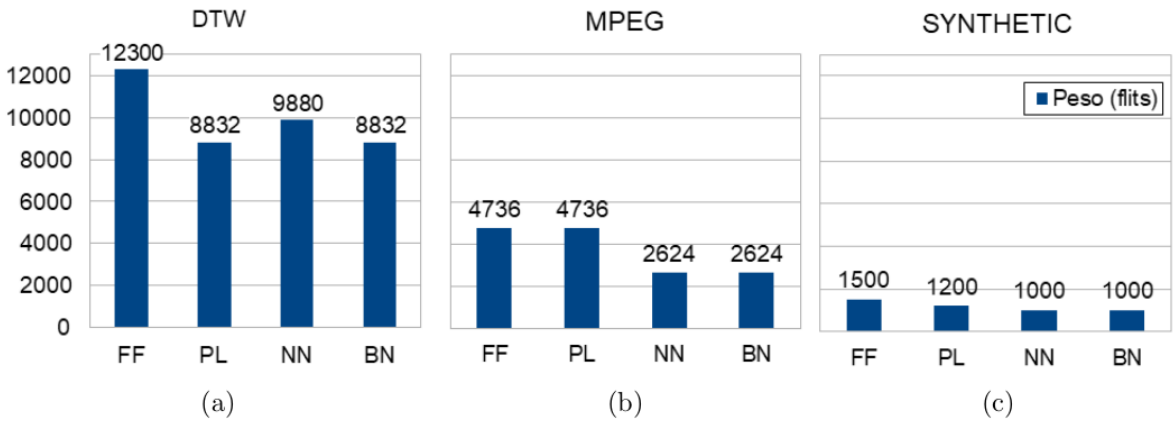
Tabela 6 – Média de desempenho das heurísticas para os casos de teste 1, 2 e 3.

Heurística	Tempo Simulação(ms)	Tempo Mapeamento(ms)
FF	6,03831	0,62336
PL	6,32565	0,88943
NN	6,03551	0,62414
BN	6,11927	0,67415

maiores taxas de comunicação em seus canais, com uma média de 9.961 flits, seguido de MPEG com 3.680 flits e SYNTHETIC com 1.175 flits. Como pode ser observado no grafo das aplicações (Figura 22), a DTW possui maior fluxo de comunicação do que as demais aplicações e um número maior de tarefas. A aplicação SYNTHETIC, apesar de possuir mais tarefas que MPEG, possui um fluxo de comunicação muito menor no grafo, por isso é a aplicação que apresentou os menores pesos em seus canais.

Com relação as heurísticas de mapeamento, foram obtidos resultados com maior variação entre si, em comparação com as métricas de desempenho anteriormente analisadas. A heurística BN apresentou os melhores resultados, com uma média de 4.152 flits, seguido de NN com 4.501, PL com 4.923 e FF com 6.179 flits. Analisando o ganho em porcentagem, como FF obteve o pior resultado, vemos que BN foi 32,80% melhor que FF, seguido de NN com 27,14% e PL com 20,32%. A Tabela 7 e a Tabela 8 apresentam as médias de pesos de comunicação para os casos de teste 1, 2 e 3.

Figura 24 – Pesos de Comunicação dos casos de teste 1, 2 e 3.



5.4.2 Casos de teste 4 e 5

Nesta subseção é realizada uma análise para os casos de teste 4 e 5 em conjunto. O objetivo é avaliar as simulações com mais de uma aplicação mapeada no MPSoC. Podemos observar na Figura 25 o desempenho das diferentes heurísticas para cada caso de teste.

Tabela 7 – Média de pesos de comunicação para as aplicações nos casos de teste 1, 2 e 3.

Aplicação	**Pesos de comunicação(flits)**
DTW	9.961
MPEG	3.686
SYNTHETIC	1.175

Tabela 8 – Média de pesos de comunicação para as heurísticas nos casos de teste 1, 2 e 3.

Heurística	**Pesos de comunicação(flits)**
FF	6.179
PL	4.923
NN	4.501
BN	4.152

Analisando primeiramente o tempo de simulação dos dois casos de teste, observa-se que os resultados foram muito parecidos. O caso de teste 4 (DTW + MPEG) obteve uma média de 10,12310 ms e o caso de teste 5 (DTW + MPEG + SYNTHETIC) registrou a média de 10,24278 ms. Considerando que: (*i*) após mapeadas, as tarefas executam em paralelo no MPSoC; (*ii*) a aplicação MPEG é a mais lenta das aplicações alvo; e (*iii*) faz parte dos casos de teste 4 e 5; é natural que ambos tenham aproximadamente o mesmo tempo de simulação.

Em relação ao tempo de mapeamento, é possível observar uma diferença nas médias para os casos de teste 4 (1,41053 ms) e 5 (2,19669 ms), justamente pelo fato de mais aplicações terem que ser mapeadas no caso de teste 5.

Comparando agora as heurísticas de mapeamento, é possível notar o mesmo padrão dos casos de teste 1, 2 e 3. Todas as justificativas de resultados de desempenho para estes casos de teste anteriores também se aplicam para estes. Em relação ao tempo de mapeamento, a heurística PL obteve a pior média de tempo, com 2,24904 ms. As melhores médias ficaram com NN com 29,29 % melhor que PL, seguido de FF com 27,37 % e BN com 22,55 %. Para a análise do tempo de simulação, a heurística PL obteve um valor médio de 10,606045 ms, com a heurística NN obtendo 6,24 % de melhoria, seguido de FF com 5,33 % e BN com 4,37 % em média. A Tabela 9 e a Tabela 10 apresentam as médias de desempenho para os casos de teste 4 e 5.

A seguir são apresentados os resultados para os pesos de comunicação nos canais da NoC para os casos de teste 4 e 5. De acordo com a Figura 26(b), o caso de teste 5 obteve uma maior média de ocupação dos canais pelo fato de ter uma aplicação a mais que o caso de teste 4 (Figura 26(a)). É possível notar também que, para casos de teste com múltiplas aplicações, os resultados evidenciam mais ainda como cada heurística de

Figura 25 – Desempenho dos casos de teste 4 e 5.

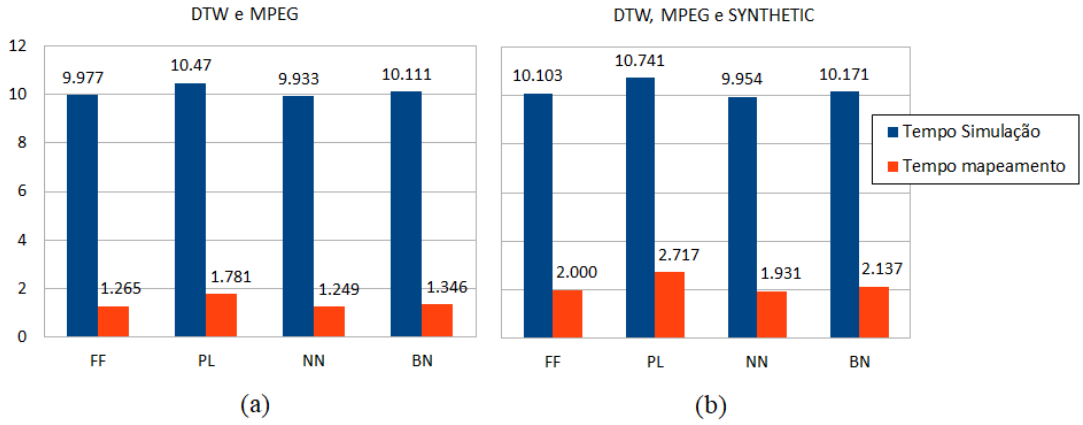


Tabela 9 – Média de desempenho das aplicações para os casos de teste 4 e 5.

Aplicação	Tempo Simulação(ms)	Tempo Mapeamento(ms)
DTW+MPEG	10,12310	1,41053
DTW+MPEG+ SYNTHETIC	10,24278	2,19669

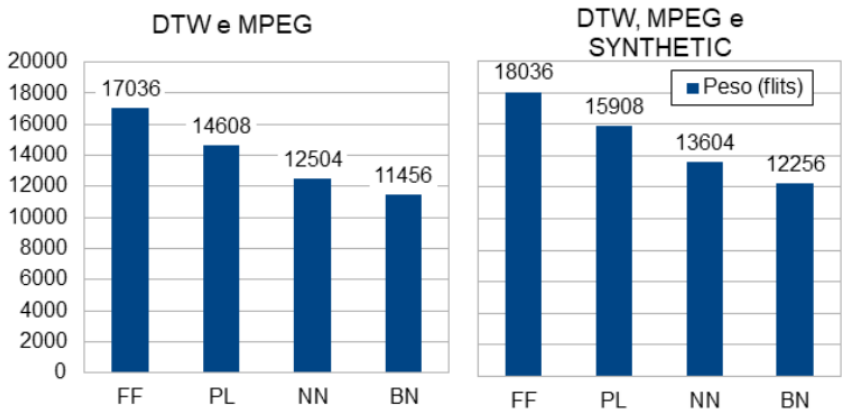
Tabela 10 – Média de desempenho das heurísticas para os casos de teste 4 e 5.

Heurística	Tempo Simulação(ms)	Tempo Mapeamento(ms)
FF	10,04057	1,63332
PL	10,60604	2,24904
NN	9,94365	1,59024
BN	10,1415	1,74183

mapeamento influência na ocupação da NoC. É trivial observar isso ao comparar com a Figura 24 na subseção 5.4.1. A heurística BN obteve a melhor média de ocupação dos canais para os dois casos de teste, com 11.856 flits, seguido de NN com 13.054, PL com 15.258 e FF com 17.536. Analisando o ganho em porcentagem, como FF obteve o pior resultado, vemos que BN foi 32,39 % melhor que FF, seguido de NN com 25,55 % e PL com 19,99 %. A Tabela 11 e a Tabela 12 apresentam as médias de pesos de comunicação para os casos de teste 4 e 5.

Os resultados descreveram o que se esperava das heurísticas. No que diz respeito à desempenho, apesar de ter ficado claro que esta métrica é afetada pela complexidade dos algoritmos, ainda assim os resultados obtidos ficaram muito próximos uns dos outros. Para a métrica de ocupação dos canais de comunicação, verificou-se que o intervalo entre os dados foi bem maior. As heurísticas PL e BN têm como proposta justamente aliviar o

Figura 26 – Pesos de Comunicação dos casos de teste 4 e 5.



(a) (b)

Tabela 11 – Média de pesos de comunicação para as aplicações nos casos de teste 4 e 5.

Aplicação	**Pesos de comunicação(flits)**
DTW+MPEG	13.901
DTW+MPEG+ SYNTHETIC	14.951

Tabela 12 – Média de pesos de comunicação para as heurísticas nos casos de teste 4 e 5.

Heurística	**Pesos de comunicação(flits)**
FF	17.536
PL	15.258
NN	13.054
BN	11.856

fluxo de dados nos canais da NoC, por isso são chamadas de Heurísticas *Congestion Aware*, ou conforme nomenclatura deste trabalho, Heurísticas que Evitam Congestionamento.

Comparando os resultados obtidos neste trabalho com os resultados de Carvalho, Calazans e Moraes (2010), que investigaram as mesmas heurísticas, temos o mesmo comportamento em relação ao desempenho, ou seja, as heurísticas FF e NN possuem praticamente o mesmo tempo de simulação, alcançando as melhores médias de tempo de mapeamento, seguido de BN e PL com a pior média. Entretanto, em relação à ocupação dos canais, o trabalho de Carvalho, Calazans e Moraes (2010) indica que a heurística PL possui resultado um pouco melhor do que BN. Já neste trabalho, a heurística BN foi a melhor considerando a média das heurísticas para todas as aplicações. Uma possível justificativa é uso de um conjunto de aplicações diferentes. Por exemplo, a Figura 24 mostra

que as heurísticas PL e BN possuem o mesmo resultado para a aplicação DTW,

6 CONSIDERAÇÕES FINAIS

A complexidade das aplicações atuais exigem uma grande capacidade de processamento, é nesse contexto que está o MPSoC. Estes sistemas são compostos por vários núcleos, o que permite o processamento de aplicações em paralelo. Porém, como observado na pesquisa realizada neste trabalho, o desempenho e o consumo de energia nesses sistemas não dependem somente das características de hardware, mas também das interações de software, como por exemplo o Mapeamento de Tarefas. De acordo com os objetivos deste trabalho, a literatura sobre o mapeamento foi investigada, sendo possível classificar as diferentes estratégias e identificar que resultados são alcançados dependendo de como as tarefas de uma dada aplicação estão dispostas sobre os recursos do MPSoC. As heurísticas *First Free*, *Nearest Neighbor*, *Path Load* e *Best Neighbor* foram selecionadas e implementadas. Este trabalho realizou também uma pesquisa sobre os MPSoCs que existem atualmente. Diante do problema de se obter tais sistemas compostos por centenas de processadores a fim de serem realizados testes sobre eles, foi realizada uma pesquisa sobre as plataformas de software que simulam tais sistemas. A plataforma HeMPS foi a única plataforma que se mostrou acessível e por este motivo foi selecionada. As heurísticas alvo foram então integradas à plataforma para simulação.

Foram criados cinco casos de teste e as simulações foram realizadas utilizando três aplicações que já vem integradas na plataforma HeMPS. Todos os casos de teste simularam uma plataforma com dimensão 5x5. Os casos de teste 1, 2 e 3 utilizaram uma aplicação diferente para cada caso de teste. Já os casos de teste 4 e 5, utilizaram duas e três aplicações respectivamente. Após as simulações, foram coletados dados referentes ao desempenho e a ocupação dos canais de comunicação do MPSoC. Os resultados das simulações evidenciaram que dentre as heurísticas investigadas, a *Best Neighbor* controla melhor os congestionamentos nos canais de comunicação, já que possui um algoritmo mais otimizado. Em relação ao desempenho das heurísticas foi possível identificar que estratégias mais elaboradas possuem resultados de tempo de mapeamento e de simulação mais altos que as estratégias menos elaboradas, apesar dos resultados serem muito próximos uns dos outros. Esse maior custo de desempenho ocorre porque as estratégias mais elaboradas possuem maior complexidade.

6.1 Desafios Encontrados e Limitações

No início da fase de implementação das heurísticas, foi possível notar que embora existisse um arquivo na HeMPS que implementava o grafo de uma dada aplicação, ele não era utilizado pela plataforma. A HeMPS simplesmente entrava no diretório da aplicação, lia as tarefas descritas em vários arquivos fonte em linguagem C, e as mapeava no MPSoC seguindo a ordem de leitura. Isso consistia em um desafio para a conclusão dos objetivos do trabalho, pois as heurísticas que seriam utilizadas eram guiadas pelo grafo de aplicação, mapeando em primeiro lugar as tarefas iniciais da aplicação, e em seguida as tarefas que

possuíam dependências com estas. Foi preciso então realizar modificações no sistema operacional da HeMPS que fizessem com que a plataforma se guiasse pelos grafos de aplicação, o que demandou um tempo a mais inicialmente não considerado.

Uma limitação encontrada foi que em alguns casos de teste, a plataforma travava a sua simulação. Em casos de teste com duas aplicações, por exemplo, as duas aplicações eram mapeadas no MPSoC, a primeira executava normalmente e a segunda aplicação simplesmente não terminava a execução, ficando a plataforma sem qualquer ação. Isso limitou em muito o escopo de casos de teste a serem simulados. Não foi possível simular por exemplo, casos de teste com quatro aplicações simultâneas. Em sistemas como estes, é sempre um desafio encontrar erros com diferentes níveis de software: componentes de hardware (e.g. processador, memória e NoC, todos descritos em software), sistema operacional e aplicações.

6.2 Sugestões para Trabalhos Futuros

A seguir, encontram-se listadas algumas sugestões para trabalhos futuros:

- Investigar as limitações apontadas na HeMPS com o objetivo de eliminá-las;
- Foi possível verificar que o tempo de simulação de uma aplicação é impactado pelo tempo de mapeamento, pois mesmo sendo simples, os algoritmos de mapeamento acabam inserindo um certo *overhead* devido à sua complexidade. Por exemplo, se um algoritmo levasse um segundo para mapear uma tarefa e esta executasse por dez segundos, teríamos um *overhead* inaceitável, mas se a tarefa executasse por vários minutos o *overhead* seria insignificante. Nesse sentido, seria interessante multiplicar as taxas de comunicação nas aplicações, para investigar se as heurísticas melhoram seu padrão de desempenho. Não foi possível realizar esta investigação devido a falta de tempo hábil;
- Investigar outras heurísticas de mapeamento na plataforma HeMPS, como por exemplo a heurística de Ng et al. (2015). Esta heurística implementa a migração de tarefas, onde uma dada tarefa já mapeada migra para um novo recurso de processamento, se adequando em tempo de simulação para melhorar desempenho e consumo de energia. Com esta estratégia, espera-se melhores resultados do que os encontrados com as heurísticas já investigadas; e
- Propor uma nova heurística de mapeamento com base nas descobertas deste trabalho.

REFERÊNCIAS

ABBAS, A. et al. A survey on energy-efficient methodologies and architectures of network-on-chip. **Computers & Electrical Engineering**, Elsevier, v. 40, n. 8, p. 333–347, 2014. Citado na página 27.

BENINI, L. et al. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In: EDA CONSORTIUM. **Proceedings of the Conference on Design, Automation and Test in Europe**. Dresden, Germany, 2012. p. 983–987. Citado na página 28.

BESERRA, G. S.; NIAKI, S. H. A.; SANDER, I. Integrating virtual platforms into a heterogeneous moc-based modeling framework. In: IEEE. **Specification and Design Languages (FDL), 2012 Forum on**. Vienna, Austria, 2012. p. 143–150. Citado na página 28.

BOHNENSTIEHL, B. et al. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In: IEEE. **VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on**. Honolulu, HI, USA, 2016. p. 1–2. Citado 3 vezes nas páginas 30, 34 e 36.

BORKAR, S. Thousand core chips: a technology perspective. In: ACM. **Proceedings of the 44th annual Design Automation Conference**. San Diego, California, 2007. p. 746–749. Citado na página 37.

CALISKANELLI, I. et al. Bioinspired load balancing in large-scale wsns using pheromone signalling. **International Journal of Distributed Sensor Networks**, SAGE Publications Sage UK: London, England, v. 9, n. 7, p. 172012, 2013. Citado na página 43.

CARARA, E. A. et al. Hemps-a framework for noc-based mpsoc generation. In: **Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on**. Taipei, Taiwan: IEEE, 2009. p. 1345–1348. Citado 4 vezes nas páginas 28, 39, 40 e 49.

CARVALHO, E. L. d. S. Mapeamento dinâmico de tarefas em mpsocs heterogêneos baseados em noc. Pontifícia Universidade Católica do Rio Grande do Sul, 2009. Citado 7 vezes nas páginas 24, 29, 53, 54, 55, 58 e 59.

CARVALHO, E. L. de S.; CALAZANS, N. L. V.; MORAES, F. G. Dynamic task mapping for mpsocs. **IEEE Design & Test of Computers**, IEEE, v. 27, n. 5, p. 26–35, 2010. Citado 5 vezes nas páginas 29, 30, 41, 46 e 68.

CASTILHOS, G. et al. Distributed resource management in noc-based mpsocs with dynamic cluster sizes. In: IEEE. **VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on**. Natal, Brasil, 2013. p. 153–158. Citado na página 43.

CASTRILLON, J. et al. Communication-aware mapping of kpn applications onto heterogeneous mpsocs. In: IEEE. **Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE**. San Francisco, CA, USA, 2012. p. 1262–1267. Citado na página 29.

CHEN, L.; MARCONI, T.; MITRA, T. Online scheduling for multi-core shared reconfigurable fabric. In: EDA CONSORTIUM. **Proceedings of the Conference on**

Design, Automation and Test in Europe. Dresden, Germany, 2012. p. 582–585. Citado 2 vezes nas páginas 29 e 30.

CHOI, J. et al. Executing synchronous dataflow graphs on a spm-based multicore architecture. In: ACM. **Proceedings of the 49th Annual Design Automation Conference**. San Francisco, CA, USA, 2012. p. 664–671. Citado na página 29.

DINECHIN, B. D. D. et al. Time-critical computing on a single-chip massively parallel processor. In: IEEE. **Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014**. Dresden, Germany, 2014. p. 1–6. Citado na página 30.

DING, J.-H. et al. An efficient and comprehensive scheduler on asymmetric multicore architecture systems. **Journal of Systems Architecture**, Elsevier, v. 60, n. 3, p. 305–314, 2014. Citado na página 30.

FATTAH, M. et al. Adjustable contiguity of run-time task allocation in networked many-core systems. In: IEEE. **Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific**. Singapore, 2014. p. 349–354. Citado 2 vezes nas páginas 29 e 42.

GAREY, M. R.; JOHNSON, D. S. A guide to the theory of np-completeness. **WH Freemann, New York**, v. 70, 1979. Citado na página 30.

GRüTTNER, K. et al. Enabling timing and power aware virtual prototyping of hw/sw systems. **DATE**, 2011. Citado na página 27.

HAIYUN, G. Survey of dynamically reconfigurable network-on-chip. In: IEEE. **Future Computer Sciences and Application (ICFCSA), 2011 International Conference on**. Hong Kong, China, 2011. p. 200–203. Citado na página 27.

HESHAM, S. et al. Survey on real-time networks-on-chip. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 28, n. 5, p. 1500–1517, 2017. Citado na página 26.

HUANG, L.-T. et al. Wena: Deterministic run-time task mapping for performance improvement in many-core embedded systems. **IEEE Embedded Systems Letters**, IEEE, v. 7, n. 4, p. 93–96, 2015. Citado na página 44.

JERRAYA, A.; WOLF, W. **Multiprocessor systems-on-chips**. [S.l.]: Morgan Kaufmann Publishers Inc, 2005. Citado na página 25.

KALREY. **MPPA-256 Processor**. 2017. Disponível em: <http://www.kalray.eu/kalray/products/#processors>. Citado 2 vezes nas páginas 34 e 36.

KAUSHIK, S. et al. Run-time computation and communication aware mapping heuristic for noc-based heterogeneous mpsoc platforms. In: IEEE. **Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on**. Tianjin, China, 2011. p. 203–207. Citado na página 29.

MADALOZZO, G. A. Adequação de modelos arquiteturais para aplicações tempo-real em sistemas many-core. Pontifícia Universidade Católica do Rio Grande do Sul, 2017. Citado na página 27.

MANDELLI, M. G. **Mapeamento dinâmico de aplicações para MPSOCS homogêneos**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2011. Citado 2 vezes nas páginas 25 e 28.

MARTIN, G. Overview of the mpsoc design challenge. In: IEEE. **Design Automation Conference, 2006 43rd ACM/IEEE**. San Francisco, CA, USA, 2006. p. 274–279. Citado na página 25.

MELLANOX, T. **"TILE-Gx72 Processor". Product Brief Description**. [S.l.: s.n.], 2015. 2 p. Citado 3 vezes nas páginas 30, 33 e 34.

MENDIS, H. R.; INDRUSIAK, L. S.; AUDSLEY, N. C. Bio-inspired distributed task remapping for multiple video stream decoding on homogeneous nocs. In: IEEE. **Embedded Systems For Real-time Multimedia (ESTIMedia), 2015 13th IEEE Symposium on**. Amsterdam, Netherlands, 2015. p. 1–10. Citado 2 vezes nas páginas 43 e 46.

MEYER, B. H.; HARTMAN, A. S.; THOMAS, D. E. Cost-effective slack allocation for lifetime improvement in noc-based mpsocs. In: IEEE. **Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010**. Dresden, Germany, 2010. p. 1596–1601. Citado na página 29.

MOHANTY, S. et al. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. **ACM SIGPLAN Notices**, ACM, v. 37, n. 7, p. 18–27, 2002. Citado na página 27.

MUNK, P. et al. Position paper: Real-time task migration on many-core processors. In: VDE. **Architecture of Computing Systems. Proceedings, ARCS 2015-The 28th International Conference on**. Porto, Portugal, 2015. p. 1–4. Citado na página 31.

NAVA, M. D. et al. An open platform for developing multiprocessor socs. **Computer**, IEEE, v. 38, n. 7, p. 60–67, 2005. Citado na página 25.

NG, J. et al. Defrag: Defragmentation for efficient runtime resource allocation in noc-based many-core systems. In: IEEE. **Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on**. Hong Kong, China, 2015. p. 345–352. Citado 4 vezes nas páginas 29, 30, 44 e 72.

NOLLET, V. et al. Run-time management of a mpsoc containing fpga fabric tiles. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, IEEE, v. 16, n. 1, p. 24–33, 2008. Citado na página 29.

OLOFSSON, A. Epiphany-v: a 1024 processor 64-bit risc system-on-chip. **arXiv preprint arXiv:1610.01832**, 2016. Citado 2 vezes nas páginas 35 e 37.

OST, L. et al. Power-aware dynamic mapping heuristics for noc-based mpsocs using a unified model-based approach. **ACM Transactions on Embedded Computing Systems (TECS)**, ACM, v. 12, n. 3, p. 75, 2013. Citado 2 vezes nas páginas 42 e 46.

OVP, I. **"Open Virtual Platform (OVP)"**. [S.l.: s.n.], 2016. Citado na página 28.

PAULIN, P. G. et al. Application of a multi-processor soc platform to high-speed packet forwarding. In: IEEE COMPUTER SOCIETY. **Proceedings of the conference on Design, automation and test in Europe-Volume 3**. Paris, France, 2004. p. 30058. Citado na página 38.

PAULIN, P. G. et al. Parallel programming models for a multiprocessor soc platform applied to networking and multimedia. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, IEEE, v. 14, n. 7, p. 667–680, 2006. Citado 3 vezes nas páginas 28, 37 e 40.

PLASMA. **Plasma CPU**. 2011. Disponível em: <http://plasmacpu.no-ip.org/>. Citado na página 50.

QUAN, W.; PIMENTEL, A. D. A hybrid task mapping algorithm for heterogeneous mpsocs. **ACM Transactions on Embedded Computing Systems (TECS)**, ACM, v. 14, n. 1, p. 14, 2015. Citado 2 vezes nas páginas 43 e 46.

REKIK, W. et al. Virtual prototyping of multiprocessor architectures using the open virtual platform. In: IEEE. **Computer Applications Technology (ICCAT), 2013 International Conference on**. [S.l.], 2013. p. 1–6. Citado na página 28.

RIGO, S. et al. Archc: A systemc-based architecture description language. In: IEEE. **Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on**. Foz do Iguacu, PR, Brasil, 2004. p. 66–73. Citado na página 28.

ROTH, C. et al. Modular framework for multi-level multi-device mpsoc simulation. In: IEEE. **Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on**. Shanghai, China, 2011. p. 136–142. Citado 2 vezes nas páginas 39 e 40.

RUARO, M.; CARARA, E. A.; MORAES, F. G. Runtime adaptive circuit switching and flow priority in noc-based mpsocs. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, IEEE, v. 23, n. 6, p. 1077–1088, 2015. Citado na página 27.

SAYUTI, M. N. S. M.; INDRUSIAK, L. S. Real-time low-power task mapping in networks-on-chip. In: IEEE. **VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on**. Natal, Brasil, 2013. p. 14–19. Citado na página 27.

SCHRANZHOFER, A.; CHEN, J.-J.; THIELE, L. Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms. In: IEEE. **Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE**. San Francisco, CA, USA, 2009. p. 151–160. Citado na página 29.

SEIDIPIRI, R. et al. Rasmap: An efficient heuristic application mapping algorithm for network-on-chips. In: IEEE. **Information and Knowledge Technology (IKT), 2016 Eighth International Conference on**. Hamedan, Iran, 2016. p. 149–155. Citado na página 45.

SIA. International technology roadmap for semiconductors (itrs). **Semiconductor Industry Association**, 1999. Citado na página 23.

SINGH, A. K. et al. Mapping on multi/many-core systems: survey of current and emerging trends. In: ACM. **Proceedings of the 50th Annual Design Automation Conference**. Austin, Texas, 2013. p. 1. Citado na página 24.

SINGH, A. K. et al. Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 35, n. 1, p. 72–85, 2016. Citado na página 45.

THIELE, L. et al. Thermal-aware system analysis and software synthesis for embedded multi-processors. In: IEEE. **Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE**. New York, NY, USA, 2011. p. 268–273. Citado na página 29.

VANGAL, S. et al. An 80-tile 1.28 tflops network-on-chip in 65nm cmos. In: IEEE. **Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International**. San Fransisco, CA, USA, 2007. p. 98–589. Citado 3 vezes nas páginas 30, 33 e 35.

VANGAL, S. R. et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. **IEEE Journal of Solid-State Circuits**, IEEE, v. 43, n. 1, p. 29–41, 2008. Citado na página 33.

WANG, F. et al. Variation-aware task and communication mapping for mpsoc architecture. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 30, n. 2, p. 295–307, 2011. Citado na página 29.

ZEFERINO, C. A. Redes-em-chip: arquiteturas e modelos para avaliação de área e desempenho. 2003. Citado na página 26.

ZHANG, D. et al. Mcvp-noc: Many-core virtual platform with networks-on-chip support. In: IEEE. **ASIC (ASICON), 2013 IEEE 10th International Conference on**. Shenzhen, China, 2013. p. 1–4. Citado 3 vezes nas páginas 28, 38 e 40.