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

MAPEAMENTO DINÂMICO DE APLICAÇÕES PARA MPSOCS HOMOGÊNEOS

MARCELO GRANDI MANDELLI

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

Orientador: Prof. Dr. Fernando Gehm Moraes
Co-orientador: Alexandre de Morais Amory

Porto Alegre
2011

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

M271m Mandelli, Marcelo Grandi
Mapeamento dinâmico de aplicações para MPSOCS homogêneos / Marcelo Grandi Mandelli. – Porto Alegre, 2011.
106 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Informática. 2. Mulitprocessadores. 3. Arquitetura de Computador. I. Moraes, Fernando Gehm. II. Título.

CDD 004.35

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

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

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Mapeamento Dinâmico de Aplicações Multitarefa para MPSoC Homogêneos**", apresentada por Marcelo Grandi Mandelli, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais , aprovada em 22/03/2011 pela Comissão Examinadora:

Prof. Dr. Fernando Gehm Moraes – Orientador — PPGCC/PUCRS

Dr. Alexandre de Morais Amory – Co-orientador — PPGCC/PUCRS

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

Profa. Dra. Fernanda Gusmão de Lima Kastensmidt – — UFRGS

Homologada em 10/05/11, conforme Ata No. 007 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

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

AGRADECIMENTOS

Até que enfim essa dissertação sai
Por isso agradeço minha mãe e o meu pai.
Na minha vida, eles são tudo
Me deram a oportunidade do estudo.
Aos meus irmãos, Alexandre e Maurício
Que me deram todo apoio desde o início.
Toda minha família agradeço, enfim
Sempre unida e um exemplo para mim.
Também agradeço ao Moraes,
Orientador bom não se faz mais!
Ao Tiago e o Daniel, lá de Caxias
Amigos que se pode contar todos dias.
Ao pessoal do Gaph também agradeço,
Por me aturarem desde o começo.
Wachter, Heck, Castilhos e resto do pessoal
Valeu por tudo, amizade sem igual.
Agradeço ao Ost por nos papers me ajudar
E que tu coloque meu nome sem pestanejar.
Ao amigo Nemetz, eu quase esqueci,
Mas valeu por tudo, qualquer coisa tô aí.
A qualquer outro que não menciono aqui,
Me perdoa que essa rima também é pra ti.

MAPEAMENTO DINÂMICO DE APLICAÇÕES PARA MPSOCS HOMOGÊNEOS

RESUMO

O avanço na tecnologia de fabricação de circuitos integrados permite obter transistores cada vez menores, tornando possível o desenvolvimento de sistemas completos em um único chip (System-on-Chip - SoC). Muitas aplicações requerem SoCs com vários processadores para poder suprir seus requisitos de desempenho. Um SoC que contém diversos elementos de processamento (Processing Element - PEs) é chamado de MPSoC. Um MPSoC pode ser classificado em homogêneo, quando todos seus PEs são iguais; ou heterogêneo, quando seus PEs são diferentes. Como infraestrutura de comunicação, o MPSoC pode utilizar NoCs como forma de interconectar os PEs. O uso de NoCs deve-se a suas vantagens em relação a barramentos, entre as quais maior escalabilidade e paralelismo na comunicação.

Um dos principais problemas relativos ao projeto de MPSoCs é a definição de qual dos PEs do sistema será responsável pela execução de cada tarefa de uma aplicação. Este problema é chamado de mapeamento de tarefas. O mapeamento pode ser classificado em estático, que ocorre em tempo de projeto, ou em dinâmico que ocorre em tempo de execução. A abordagem de mapeamento dinâmico requer primeiramente o mapeamento de tarefas iniciais de uma aplicação (que não dependem de nenhuma outra tarefa) das aplicações, sendo que as outras tarefas são mapeadas dinamicamente quando solicitadas. Também se pode classificar o mapeamento quanto ao número de tarefas que executam em um PE do sistema. O mapeamento é dito monotarefa, quando apenas uma tarefa é executada por PE, e multitarefa, quando múltiplas tarefas podem ser executadas em um mesmo PE.

Este trabalho propõe novas heurísticas de mapeamento dinâmico monotarefa e multitarefa, visando à redução de energia de comunicação. Resultados são avaliados através do MPSoC HeMPS, que executa códigos de aplicações geradas a partir de um ambiente de simulação baseado em modelos. Estas heurísticas são comparadas com heurísticas de mapeamento apresentadas na literatura, apresentando uma redução média de energia de comunicação nos cenários avaliados de até 9,8% na abordagem monotarefa e 18,6% na multitarefa. Este trabalho também avalia a inserção dinâmica de carga no sistema, utilizando para isto a implementação de uma heurística de mapeamento dinâmico de tarefas iniciais. Esta heurística é uma contribuição inovadora, visto que uma abordagem parecida não é encontrada em nenhum outro trabalho da literatura.

Palavras-Chave: Mapeamento dinâmico de tarefas, NoC, SoC, MPSoC.

DYNAMIC APPLICATION MAPPING FOR HOMOGENEOUS MPSOCS

ABSTRACT

The advance in manufacturing technology of integrated circuits enables smaller transistors, making possible the development of SoCs (System-on-Chip). Many applications require multi-processor SoCs in order to meet their performance requirements. A SoC containing several processing elements (PEs) is called MPSoC. An MPSoC can be classified as homogeneous, when all their PEs has the same architecture; or heterogeneous, when they have different architectures. . As communication infrastructure, the MPSoC can use NoCs as a way to interconnect the PEs. NoCs may be used to replace busses, due to their advantages of higher scalability and communication parallelism.

One of the main problems related to MPSoC projects is to define a PE of the system that will run each task. This problem is called task mapping. The mapping can be classified into static, which occurs at design time, and dynamic that occurs at runtime. The dynamic mapping approach requires firstly the mapping of the initial tasks of an application (which does not depend on any other task). The other tasks, in this approach, are mapped dynamically when requested. The mapping can be also classified by the number of tasks running in a PE. The mapping is classified as single task, when only one task is executed by a PE, and as multitask, when multiple tasks can be executed in a same PE.

This work proposes new single task and multitask dynamic task mapping heuristics, in order to reduce communication energy. Results are evaluated using the MPSoC HeMPS, which executes application code generated from a model-based simulation environment. These heuristics are compared with mapping heuristic presented in literature, obtaining, in the evaluated scenarios, an average communication energy reduction of 9.8%, for the single task approach, and 18.6%, for the multitask approach. This work also evaluates the inclusion of dynamic load on the system, which makes necessary the implementation of an initial tasks mapping heuristic. This heuristic is an innovative contribution, since a similar approach is not found in any other work in literature.

Key-Words: Dynamic Task Mapping, NoC, SoC, MPSoC.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BN .....Best Neighbor
CA .....Cluster Agent
CAFES .....Communication Analysis for Embedded Systems
CM .....Centralized Management
CPU .....Central Processing Unit
CWG .....Communication Weighted Graph
CWM .....Communication Weighted Model
DMA .....Direct Memory Access
DN - .....Dependences Neighborhood
DSM .....Dynamic Spiral Mapping
DSP .....Digital Signal Processor
ES .....Exhaustive Search
FDSM .....Full Dynamic Spiral Mapping
FF ..... First Free
FPGA.....Field Programmable Gate Array
GA ..... Global Agent
GI ..... Greedy Incremental
GPC ..... gerador de pacotes de controle
GPP ..... General Purpose Processor
IP ..... Intellectual Property
LCF ..... Largest Communication First
LEC-DN ..... Low Energy Consumption - Dependences Neighborhood
LEC-DN-MT ..... Low Energy Consumption - Dependences Neighborhood Multitask
LTC ..... Lista de Tarefas Comunicantes
MPEG ..... Moving Picture Experts Group
MPSoC ..... Multiprocessor System on Chip
MSA ..... Monitoring Service Access Point
MWD ..... Multi-Window Display
NH ..... Número de *Hops*
NI ..... Network Interface
nJ ..... nano Joule
NoC ..... Network on Chip
NN ..... Nearest Neighbor
PE ..... Processing Element
PL ..... Path Load
PREMAP-DN ..... PREMAP - Dependences Neighborhood
PSDM ..... Partial Dynamic Spiral Mapping
QoS ..... Quality of Service
RTL ..... Register Transfer Level
SA ..... Simulated Annealing
TS ..... Tabu Search
SO ..... Sistema Operacional
SoC ..... System on Chip
SSM ..... Static Spiral Mapping
TCB ..... Task Control Block
TLM ..... Transaction-level modeling
UML ..... Unified Modeling Language
us ..... microssegundo
VHDL ..... VHSIC Hardware Description Language
VHSIC ..... Very-High-Speed Integrated Circuit
VLSI ..... Very-Large-Scale Integration
VOPD ..... Video Object Plane Decoder
XY ..... algoritmo de roteamento XY

SUMÁRIO

1. INTRODUÇÃO

O avanço na tecnologia de fabricação de circuitos integrados permite obter transistores cada vez menores. Isto torna possível o desenvolvimento de sistemas completos em um único chip, chamados de *System-on-a-Chip* (SoC). Um SoC é um circuito integrado que implementa a maioria ou todas as funções de um sistema eletrônico completo [JER05]. Os componentes que são agregados em um SoC variam de acordo com sua aplicação. Em geral um SoC pode conter memória, processadores de diversos tipos, lógica especializada, interfaces de comunicação, e outras funções digitais e analógicas.

Muitas aplicações requerem SoCs com vários processadores para poder suprir seus requisitos de desempenho. Um SoC que contém diversos elementos de processamento (PEs, em inglês, *processing element*) é chamado de MPSoC. PE, no escopo deste trabalho, é sinônimo de processadores. Na prática, a maioria dos SoCs são MPSoCs pois é muito difícil desenvolver um sistema complexo em um chip sem utilizar múltiplas CPUs [JER05]. Os MPSoCs já estão presentes em várias implementações comercias, sendo amplamente utilizados na área de redes, telecomunicação, processamento de sinais, multimídia, entre outras [WOL08].

Uma aplicação, para que seja executada em paralelo nos diversos recursos (PEs) de um MPSoC, é decomposta em tarefas. Define-se tarefa como um conjunto de instruções e dados, com informações necessárias à sua correta execução em um dado elemento de processamento. Essas tarefas podem ser executadas de forma independente ou então ter dependência uma das outras, onde uma tarefa pode precisar de dados pré-processados por outra tarefa ou então realizar troca de dados durante as execuções das mesmas. Um dos principais problemas relativos à execução paralela das tarefas de uma aplicação é a definição de em qual dos PEs do sistema cada tarefa será executada. Este problema é chamado de *mapeamento de tarefas.*

1.1 MPSoCs

Sistemas multiprocessados em chips (MPSoC, do inglês *Multiprocessor System-on-Chip*) são arquiteturas personalizadas que fazem um balanço entre as restrições da tecnologia VLSI, com os requisitos da aplicação [JER05]. O MPSoC está se tornando um estilo de projeto cada vez mais utilizado por reduzir o tempo no qual os produtos chegam ao mercado, maximizar a reutilização de projetos, simplificar o processo de verificação (devido à existência de módulos replicados) e proporcionar flexibilidade e programabilidade para reuso de plataformas complexas depois de fabricadas [MAR06].

Quando os PEs de um MPSoC são iguais, diz-se que este é um MPSoC homogêneo. Já, quando utiliza diferentes elementos de processamento (GPPs, DSPs,

etc) diz-se que é um MPSoC heterogêneo. Um MPSoC homogêneo facilita o desenvolvimento de software e o mapeamento da aplicação na sua arquitetura, porém o uso de uma arquitetura heterogênea pode melhorar o desempenho da aplicação [JER05]. Esta melhora de desempenho em arquiteturas heterogêneas se deve principalmente ao fato de que nestas arquiteturas há a possibilidade de utilização de um módulo dedicado para cada uma das necessidades de uma aplicação.

Um tópico importante relativo aos MPSoCs é a infraestrutura de comunicação utilizada para interconectar os elementos de processamento. Entre os meios de interconexão em um MPSoC pode-se citar:

- **conexão ponto a ponto:** apresenta o melhor desempenho, por proporcionar um alto grau de paralelismo ao sistema, uma vez que os PEs são conectados por canais exclusivos. Porém, esta solução apresenta pouca escalabilidade, reusabilidade e flexibilidade por necessitar de um número excessivo de fios para a interligação.
- **barramento:** pode ser utilizado um único barramento ou então múltiplos barramentos, interligados por *bridges* ou organizados de forma hierárquica. A utilização de um barramento único apresenta maior escalabilidade e reusabilidade comparado à conexão ponto a ponto, porém, apresenta um baixo grau de paralelismo, pelo fato de que apenas uma transação pode ser realizada por instante de tempo. A utilização de múltiplos barramentos, tanto interligados através de *bridges* ou de forma hierárquica, apenas minimiza os problemas citados.
- **redes intra-chip:** chamadas de NoCs (do inglês, *Networks-on-Chip*), são compostas basicamente por um conjunto de roteadores e canais de comunicação que interconectam os núcleos do sistema. A comunicação entre tais núcleos ocorre através da troca de mensagens geralmente transmitidas na forma de pacotes ao longo da rede. O uso de NoCs provê uma maior escalabilidade e reusabilidade do que as abordagens anteriores, conseguindo também um bom grau de paralelismo na comunicação.

1.2 Mapeamento de Tarefas

O ato de escolher o melhor PE do MPSoC para alocar uma tarefa é denominado de mapeamento de tarefas. Em alguns trabalhos como [CHO08][SCH10] o mapeamento é chamado de *alocação de tarefas.* Ao escolher o melhor recurso para se mapear uma tarefa deve-se buscar atender os requisitos do sistema. Por exemplo, o consumo de energia torna-se importante em dispositivos portáteis. Por outro lado, o atendimento dos prazos (em inglês, *deadlines*) das tarefas é fundamental em sistemas com restrições de tempo real. Assim, decisões de mapeamento podem influenciar drasticamente o

desempenho do sistema.

Neste trabalho é proposta a classificação do mapeamento de acordo com quatro critérios: (1) o momento em que é executado, (2) o número de tarefas por PE, (3) a entidade que controla o mapeamento; e (4) a arquitetura alvo.

Considerando o momento em que o mapeamento é executado, as seguintes abordagens podem ser utilizadas:

- **em tempo de projeto**: é chamado também estático ou *off-line*. Neste tipo de mapeamento podem-se usar algoritmos mais complexos avaliando um número maior de alternativas de mapeamento, visto que o tempo de computação destes algoritmos não vão influenciar no tempo de execução das aplicações. No entanto, o mapeamento estático não é capaz de lidar com uma carga de trabalho dinâmico, ou seja, novas aplicações inseridas em tempo de execução.
- **em tempo de execução**: é chamado também dinâmico ou *on-line*. Este tipo de mapeamento exige uma heurística simples e rápida uma vez que esta pode interferir no tempo de execução das aplicações. Além disso, este algoritmo suporta uma carga dinâmica de trabalho, podendo lidar com novas tarefas ou aplicações inseridas no sistema em tempo de execução. Duas abordagens de mapeamento dinâmico são encontradas na literatura: com e sem reserva de recursos. No escopo deste trabalho, recurso é sinônimo de processador. Estas duas abordagens são descritas a seguir:
    - *com reserva de recursos*: a heurística de mapeamento verifica, antes da realização do mapeamento, se há recursos suficientes no MPSoC para mapear todas as tarefas de uma aplicação. A clara vantagem deste método é garantir que a aplicação será mapeada por completo. Por outro lado, a aplicação pode levar mais tempo para iniciar a sua execução se não houver disponibilidade de recursos suficientes quando ela for solicitada.
    - *sem reserva de recursos*: nesta abordagem uma aplicação não precisa ser mapeada por completo. As tarefas de uma aplicação são mapeadas quando solicitadas por outras tarefas. Para isso, primeiramente, é necessário o mapeamento de tarefas sem dependências de outras tarefas, chamadas tarefas iniciais. Estas tarefas iniciais são responsáveis por inicializar a execução da aplicação, solicitando o mapeamento das demais tarefas quando estas forem necessárias. Esta abordagem faz com que as aplicações iniciem mais rapidamente, pois não é preciso esperar que haja recursos disponíveis para o mapeamento de todas as suas tarefas. Porém,

algumas tarefas podem esperar recursos se tornarem disponíveis quando o uso de recursos for alto.

Considerando o número de tarefas mapeadas por PE, as seguintes abordagens podem ser utilizadas:

- **monotarefa**: nessa abordagem apenas uma tarefa é mapeada em um PE.
- **multitarefa**: nessa abordagem mais de uma tarefa podem ser mapeadas em um mesmo PE. Para isto, é necessária uma técnica de *agrupamento* (em inglês, *clustering*) para definir um grupo de tarefas a ser mapeado em um mesmo PE. Esta técnica é realizada de acordo com alguns critérios, como a comunicação entre as tarefas, tempo de execução, prazos (em inglês, *deadlines*), etc. Em PEs que executam sistemas operacionais multitarefa, este tipo de mapeamento torna-se obrigatório para melhor explorar os recursos do sistema.

O mapeamento dinâmico requer uma entidade responsável por mapear as tarefas em tempo de execução. Tal controle pode ser:

- **centralizado**: um único PE é responsável por receber as solicitações de mapeamento, ler o código-objeto da tarefa solicitada a partir de uma memória externa (chamada neste trabalho de repositório de tarefas), executar heurísticas de mapeamento, e enviar a tarefa para o PE escolhido. Esta abordagem não é escalável, podendo levar a regiões congestionadas (em inglês, *hot-spots*) na NoC e reduzir o desempenho global.
- **distribuído**: o MPSoC é dividido em regiões (*clusters*), e um PE em cada região é responsável por executar as heurísticas de mapeamento dentro dela. Apesar da maior escalabilidade, um gargalo persiste: o acesso ao repositório de tarefas, caso este seja global para o sistema.

Finalmente, o mapeamento pode ser classificado de acordo com a arquitetura do sistema:

- **homogêneo**: quando todos os PEs são idênticos. Isto torna o mapeamento e migração de tarefas uma tarefa mais fácil, porque não é necessário considerar o tipo de PE no momento do mapeamento.
- **heterogêneo**: quando PEs diferentes são utilizados no mesmo sistema, como por exemplo, processadores de propósito geral, DSPs, IPs dedicados, etc.. Antes do mapeamento, um processo de *binding* (do inglês, ligação) é executado, definindo em qual(is) PE(s) uma tarefa pode executar, limitando a escolha do mapeamento para esta tarefa.

Outro ponto importante relacionado ao mapeamento de tarefas é quanto à função

custo utilizada. Como dito anteriormente, na realização do mapeamento de tarefas busca-se uma otimização de desempenho a fim de atender requisitos do sistema. Para isto, uma função custo é utilizada para avaliar qual PE do sistema é o melhor para se mapear determinada tarefa. Esta função custo leva em conta métricas de desempenho como energia consumida, latência, vazão, ocupação de PEs, ocupação de canais de comunicação. Estas métricas podem ser combinadas, buscando otimizações multi-objetivo.

É também importante mencionar que existe certa confusão de conceitos entre mapeamento de tarefas e migração de tarefas na literatura [ZIP09]. A migração de tarefas é o ato de transferir uma tarefa já mapeada de um processador para outro. Já, o mapeamento de tarefas define o posicionamento inicial de uma dada tarefa. Uma vez que essa tarefa está sendo executada, seu desempenho pode se degradar devido, por exemplo, a sobrecarga de processamento do PE onde esta está executando. A migração de tarefas, assim, permite[MIL00]:

- distribuição de carga, migrando tarefas de nodos sobrecarregados para outros com menor carga;
- tolerância a falhas, migrando tarefas de nodos que podem ter tido falhas;
- administração do sistema facilitado, migrando tarefas de nodos que podem ser desligados ou indisponibilizados;
- localidade de acesso de dados, migrando tarefas para mais perto da fonte de dados.

Além disso, a migração de tarefas requer a definição de pontos de migração, salvamento de contexto, restauração do contexto, entre outras ações não incluídas no mapeamento tarefa.

1.3 Objetivos

Os objetivos do presente trabalho se dividem em estratégicos e específicos. Entre os objetivos estratégicos cita-se:

- Domínio da tecnologia de projeto de sistemas multiprocessados em chip (MPSoC);
- Domínio da tecnologia de redes intra-chip (NoC);
- Domínio de técnicas de mapeamento de tarefas.

Específicos:

- *Inserção de heurísticas de mapeamento na plataforma HeMPS.* Aplicar as heurísticas de mapeamento dinâmico de tarefas desenvolvido em [CAR10] na

plataforma HeMPS. Avaliar estas heurísticas no MPSoC HeMPS com o objetivo de otimizá-las.

- *Integração da infraestrutura de monitoramento com as heurísticas de mapeamento.* Integrar heurísticas de mapeamento com a infraestrutura de monitoramento proposta por [MAR10]. O monitoramento tem como função propiciar informações do sistema em tempo de execução para que estas heurísticas tomem decisões de mapeamento.
- *Geração de códigos de aplicações para plataforma HeMPS.* Geração de código de aplicações em linguagem C para a plataforma HeMPS. Estes códigos são utilizados na realização da experimentação das heurísticas de mapeamento implementadas.
- *Implementação de novas heurísticas de mapeamento.* Propor novas heurísticas de mapeamento priorizando o mapeamento multitarefa. Estas heurísticas têm como meta principal de otimização a redução de energia consumida na comunicação.
- *Comparação das políticas de mapeamento de tarefas.* Definir qual das políticas é mais eficiente, e quais as vantagens e limitações de cada uma.
- *Inserção dinâmica de aplicações na plataforma HeMPS.* Agregar suporte à inserção de aplicações durante o tempo de execução na plataforma HeMPS.

1.4 Estrutura do Documento

O restante deste documento é organizado como segue. No Capítulo 2 são apresentados trabalhos relacionados, primeiramente mostrando o estado da arte em mapeamento dinâmico de tarefas e após apresentando os trabalhos que serão utilizados como referência. No Capítulo 2.3, é apresentada a plataforma MPSoC de referência deste trabalho, abordando sua arquitetura e sistema operacional. No Capítulo 4 são apresentadas as novas heurísticas de mapeamento propostas por este trabalho. O Capítulo 5 avalia estas heurísticas, apresentando resultados experimentais, bem como uma discussão dos mesmos. O Capítulo 6 apresenta modificações na plataforma de referência deste trabalho com o propósito de agregar suporte à inserção de aplicações durante o tempo de execução. No Capítulo 7 são apresentados as conclusões e trabalhos futuros. E, por fim, no Capítulo 0 são apresentadas as referências bibliográficas deste trabalho.

2. TRABALHOS RELACIONADOS

Este Capítulo apresenta inicialmente as abordagens de mapeamento utilizadas como referência neste trabalho, as quais servirão como base de comparação em resultados experimentais. A abordagem proposta por [MAR08], apresentada na Seção 2.1, é utilizada como referência de mapeamento estático. Já, as heurísticas propostas por [CAR10], apresentada na Seção 2.2, são utilizadas como referência de mapeamento dinâmico.

Na sequência são apresentados, na Seção 2.3, trabalhos relacionados ao mapeamento dinâmico de tarefas. Estes trabalhos são classificados de acordo com a taxonomia proposta anteriormente, sendo comparados ao presente trabalho. A classificação das heurísticas de mapeamento proposta neste trabalho incluem:

- mapeamento dinâmico;
- sem reservas de recursos;
- arquitetura homogênea;
- mono e multitarefa;
- função-custo: energia consumida na comunicação.

2.1 Algoritmo de Mapeamento Estático de Referência

Marcon et al. [MAR07][MAR08] propõem uma comparação entre algoritmos de mapeamento estático de tarefas visando baixo consumo de energia. Os algoritmos utilizam um modelo de comunicação com pesos (CWM, em inglês *Communication-Weighted Model)* para produzir um conjunto experimentos representando padrões de comunicação de aplicações. A estrutura básica utilizada para representar uma aplicação é um grafo de comunicação com pesos (CWG, em inglês *Communication-Weighted Graph*). Este é um grafo dirigido, onde conjunto de vértices representa os núcleos da aplicação, e o conjunto de arestas representa as comunicações entre cada par de núcleos.

Uma ferramenta chamada CAFES (em inglês, *Communication Analysis For Embedded Systems*) [MAR05] foi desenvolvida para a avaliação das estratégias de mapeamento em diferentes topologias de infraestrutura de comunicação. Além disso, o CAFES permite a comparação e inclusão de modelos, gerar e simular aplicações para avaliar o seu comportamento e estimar o tempo e consumo de energia devido à comunicação.

Cinco algoritmos de mapeamento são avaliados: busca exaustiva (ES, em inglês *Exaustive Search*), dois algoritmos estocásticos (*simulated annealing* (SA) e *tabu search*

(TS)) e duas heurísticas g*reedy: Largest Communication First* (LCF, do inglês maior comunicação primeiro) e *Incremental* (GI). Neste trabalho o algoritmo S*imulated Annealing* é utilizado como referência com o objetivo de comparar as heurísticas de mapeamento aqui propostas com um algoritmo de mapeamento estático. A escolha deste algoritmo se deve ao fato de ser utilizado em vários trabalhos como [NGO06][CAR10][ORS07][LIN05]. Além disso, a disponibilidade da ferramenta gráfica CAFES para o mapeamento de tarefas contribuiu para a escolha do trabalho como base das comparações.

O SA é um algoritmo estocástico que tem como base dois laços aninhados. O laço mais externo gera um mapeamento inicial criado a partir de trocas aleatórias de vários módulos. O laço interno realiza um refinamento do mapeamento inicial, escolhendo aleatoriamente um par de módulos a serem trocados de posição, gerando um novo mapeamento. As trocas respeitam um parâmetro de controle, chamado *temperatura*. Este parâmetro permite trocas mesmo quando se obtém soluções piores dentro de uma faixa aceitável. A solução buscada quando do mapeamento dos módulos é uma distribuição que resulte em um menor consumo de energia. A *temperatura* é decrementada a cada iteração do laço interno restringindo, assim, cada vez mais as trocas do mapeamento. O melhor mapeamento encontrado a cada interação do laço interno é comparado a um mapeamento chamado global que armazena a melhor distribuição encontrada durante a execução do SA. Se encontrado um mapeamento com melhor resultado de consumo de energia quando comparado ao mapeamento global, este se torna o novo mapeamento global.

2.2 Heurísticas de Mapeamento Dinâmico de Referência

Nesta Seção são apresentadas as heurísticas de mapeamento dinâmico propostas por Carvalho et. al. [CAR10]. Estas heurísticas são utilizadas como referência de comparação nos resultados experimentais, principalmente devido ao acesso a uma descrição detalhada destas heurísticas. Assim, na Seção 2.2.1, é mostrada como é feita a modelagem das aplicações utilizada no trabalho. Depois, na Seção 2.2.2, é apresentado como é feito o mapeamento da tarefa inicial de uma aplicação. Por último, na Seção 2.2.3 são mostradas as heurísticas de mapeamento dinâmico.

2.2.1 Modelagem de Aplicações

A abordagem utilizada para a modelagem de aplicações no trabalho é representada por um *grafo dirigido*, em que os vértices representam tarefas e as arestas representam comunicações entre tarefas da aplicação. A Figura 1 mostra um exemplo de uma aplicação modelada de acordo com esta abordagem.

Figura 1 – Exemplo de aplicação modelada por um grafo de tarefas [CAR10].

Vértices representados por círculos com linhas duplas se referem a *tarefas iniciais da aplicação* e os demais vértices são *tarefas* de *hardware* e de *software.* Há apenas uma tarefa inicial por aplicação que é iniciada logo que a aplicação é disparada pelo usuário. As arestas do grafo possuem pesos referentes ao *volume* e as *taxas* de comunicação entre as tarefas, em ambos os sentidos. Uma aresta define, assim, um *par de tarefas mestre-escravo* em que a tarefa chamada *mestre* precisa solicitar o mapeamento da sua tarefa *escrava* antes da comunicação propriamente dita ser realizada. Cada aresta possui quatro pesos atribuídos {*Vms, Rms, Vsm, Rsm*}, definindo o volume V e a taxa R de transmissão de dados entre as tarefas no sentido mestre-escrava *ms* e escrava-mestre *sm*.

2.2.2 Mapeamento das Tarefas Iniciais da Aplicação

A abordagem de mapeamento de tarefas iniciais adotada no trabalho define posições fixas e distantes umas das outras para o mapeamento das tarefas iniciais de cada aplicação. Isto propicia que cada aplicação ocupe uma região diferente do MPSoC, reduzindo o número de canais compartilhados por comunicações de aplicações diferentes. Esta abordagem é denominada *clusterização*, pois simula a divisão do MPSoC em regiões (em inglês, *clusters*). Vale ressaltar que o termo *clusterização* utilizado no trabalho difere da denominação utilizada para designar o agrupamento de tarefas num mesmo PE utilizado no mapeamento multitarefa.

Um *cluster* tem limites virtuais, sendo que uma aplicação pode utilizar recursos que ultrapassem os limites de um *cluster* para mapear suas tarefas. Dessa forma, o limite para mapear as tarefas de uma aplicação é função apenas da demanda da aplicação e da disponibilidade de recursos no MPSoC. Entretanto, o número de aplicações simultâneas é limitado pelo número de recursos dedicados a receber tarefas iniciais da aplicação.

Na Figura 2, apresenta-se um exemplo MPSoC 6x6 particionado em 4 clusters. Os PEs em destaque são aqueles reservados para o mapeamento das tarefas iniciais das aplicações. Eles são posicionados preferencialmente no centro do *cluster* gerado pelo particionamento. Assim deve-se reduzir a sobreposição entre as tarefas de aplicações diferentes. No caso deste MPSoC, é permitida a execução simultânea de 4 aplicações.

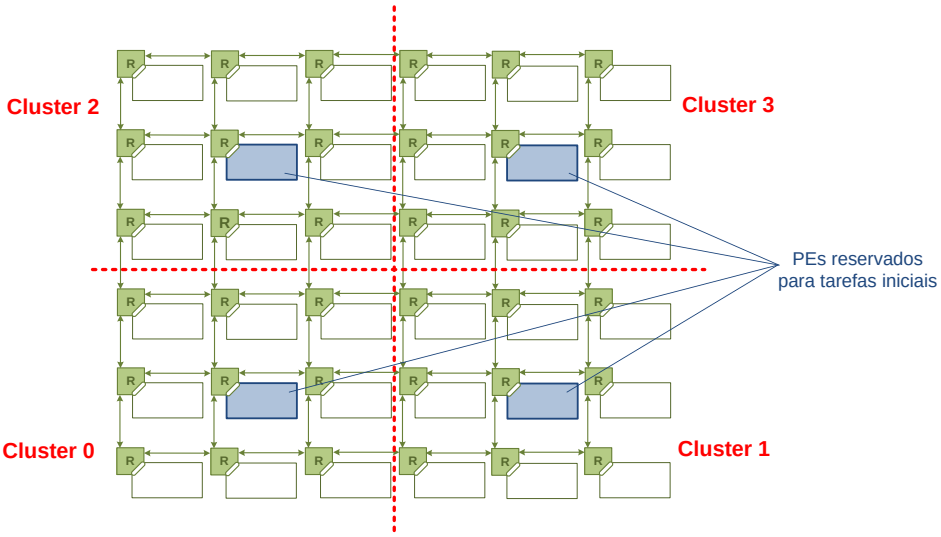


Figura 2 - Mapeamento de tarefas iniciais das aplicações [CAR10].

2.2.3 Heurísticas de Mapeamento Dinâmico

Nesta Seção são apresentadas as heurísticas de mapeamento proposta por Carvalho et al. [CAR10] que serão utilizadas como referência no presente trabalho. Vale ressaltar que antes de executar estas heurísticas é verificado se há recursos disponíveis no sistema para mapear a tarefa requisitada. Somente caso exista recursos disponíveis estas heurísticas são executadas. Caso contrário, a tarefa requisitada é escalonada para ser mapeada em outro momento. A forma com que isto é realizado não está no enfoque deste trabalho.

2.2.3.1 First Free (FF)

O algoritmo *First Free* (FF) seleciona o primeiro recurso livre para mapear uma nova tarefa. A escolha deste recurso é diretamente relacionada ao caminho de procura usado na heurística que, no trabalho, é apresentado na Figura 3. Este caminho se dá a partir da esquerda para direita, buscando coluna por coluna, de baixo para cima.

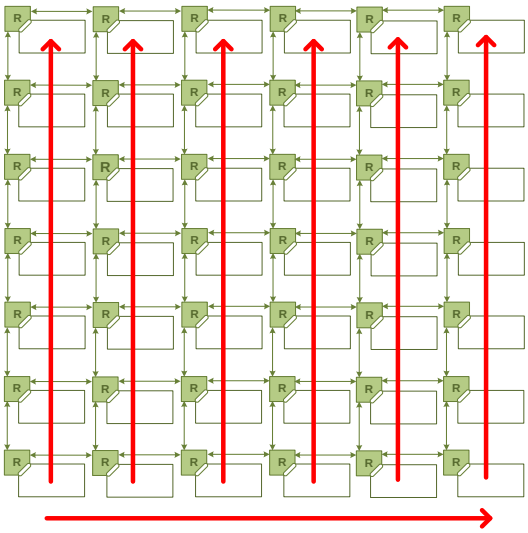


Figura 3 - Caminho de procura da heurística FF.

O pseudocódigo desta heurística é mostrado na Figura 4, mostrando a simplicidade de seu algoritmo. O algoritmo inicia analisando-se cada PE p_i do sistema na ordem do caminho apresentado na Figura 3. Para cada PE é verificado se ele está livre (linha 3). Em caso afirmativo, este PE é o escolhido para o mapeamento, sendo terminada a execução do algoritmo (linha 4). Caso contrário, prossegue-se a busca por um PE livre.

Entrada: Não há entrada para a heurística
Saída: O PE p_i onde será mapeada a tarefa requisitada

```
1.  PARA TODOS p_i
2.      // Testa a ocupação de p_i
3.      SE estado(p_i) = livre ENTÃO
4.          retorna p_i  //Termina a execução e retorna p_i
5.      FIM SE
6.  FIM PARA
```

Figura 4 – Pseudocódigo da Heurística FF.

Esta heurística gera soluções não otimizadas, pois seu critério de mapeamento é extremamente simples, demandando um tempo de execução pequeno em relação a outras heurísticas que serão apresentados a seguir.

2.2.3.2 Nearest Neighbor (NN)

A heurística *Nearest Neighbor* (NN) considera apenas a proximidade de um recurso disponível para mapear a tarefa solicitada. A procura por um recurso livre se dá a partir da posição da tarefa mestre (i. e. a tarefa que solicitou o mapeamento), conforme ilustrado na Figura 5. A partir desta posição é seguido um caminho circular, onde os vizinhos são testados de acordo com o número de *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. O caminho de procura da heurística NN também pode ser visto na Figura 5.

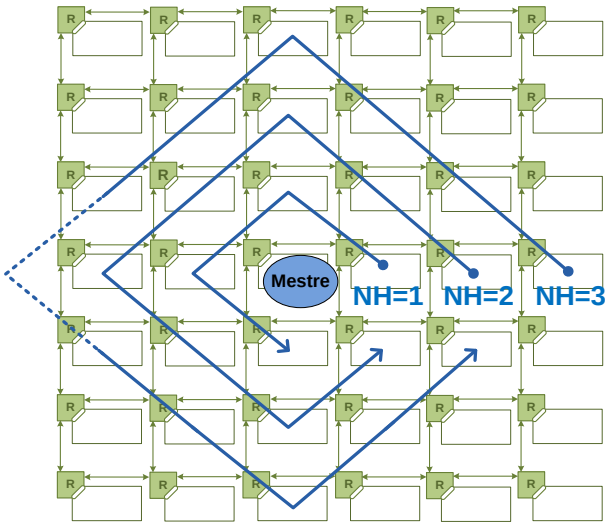


Figura 5 - Caminho de procura da heurística NN.

Na Figura 6 é mostrado o pseudocódigo da heurística NN. A heurística NN tem como entrada o PE p_m onde está mapeada a tarefa mestre. Para se escolher um PE para mapear a tarefa requisitada, os vizinhos mais próximos de p_m são analisados primeiro, atribuindo-se à distância *dist* o valor de 1 *hop* (linha 1). Estes vizinhos são colocados em uma lista (linha 3) que será percorrida verificando se cada PE p_i nesta lista está livre (linhas 6). O primeiro PE livre dentro deste grupo de vizinhos será retornado para ser mapeada a tarefa requisitada, terminando-se a execução do algoritmo (linha 7). Caso nenhum PE deste grupo de vizinhos estiver livre, a distância *dist* é incrementada (linha 10), buscando-se um novo grupo de vizinhos. A busca por um PE para mapear a tarefa requisitada se dará enquanto a distância dos vizinhos não ultrapassem os limites da NoC (linhas 2-11).

```
Entrada: O PE p_m onde está mapeada a tarefa mestre
Saída: O PE p_i onde será mapeada a tarefa requisitada
1.  dist ← 1      // Inicializa a distância em nº de hops em 1
2.  ENQUANTO dist<=tamanho_da_NoC FAÇA
3.      lista_de_vizinhos ← vizinhos(dist, p_m) // Obtêm todos vizinhos de p_m com distância dist
4.      PARA TODOS ELEMENTOS p_i NA lista_de_vizinhos
5.          // Testa a ocupação do PE
6.          SE estado(p_i) = livre ENTÃO
7.              retorna p_i  //Termina a execução e retorna p_i
8.          FIM SE
9.      FIM PARA
10.     dist ← dist + 1 // Incrementa dist para se procurar na próxima faixa de vizinhos
11. FIM ENQUANTO
```

Figura 6 - Pseudocódigo da heurística NN.

2.2.3.3 Best Neighbor (BN)

A heurística *Best Neighbor* (BN) combina estratégias da heurística PL (*Path Load)*, detalhada a seguir, e NN*.* A BN tem como principal objetivo a redução de congestionamentos na rede buscando, assim, reduzir distâncias entre tarefas comunicantes.

A heurística PL percorre todos os PEs disponíveis do sistema, calculando o somatório de ocupação dos canais do caminho de comunicação entre a tarefa mestre (que solicita o mapeamento) e cada possível PE que está sendo testado para mapear a tarefa solicitada. A nova tarefa é mapeada no primeiro PE que obtiver o menor custo de comunicação. O maior problema desta heurística é o fato de ser exaustiva, podendo comprometer o tempo de execução de sistemas maiores. Para resolver isso, foi proposta a heurística BN que utiliza a mesma função custo da PL (somatório da ocupação dos canais), porém com um diferente caminho de procura para encontrar um PE para mapear a tarefa requisitada.

O caminho de procura utilizado na BN é o mesmo da NN, porém não é selecionado o primeiro vizinho livre como em NN. Em vez disto, quando há mais de um nodo livre em um anel de verificação, ou seja, com o mesmo número de *hops* de distância da tarefa mestre, o BN avalia e seleciona o melhor deles de acordo com o cálculo da carga do

caminho utilizado no algoritmo PL.

Vale ressaltar que para a obtenção de dados para o cálculo da função custo utilizada nas heurísticas PL e BN, é necessária a utilização de uma infraestrutura de monitoramento do sistema. Neste caso, o monitoramento tem a função de recolher informações de volume de comunicação que está passando pelos canais da rede.

Na Figura 7 é mostrado um pseudocódigo descrevendo o comportamento do algoritmo da heurística BN. Este algoritmo é praticamente igual ao da heurística NN, porém é inserida a função custo do PL. Dessa forma, como entrada para o algoritmo tem-se também o PE p_m onde está mapeada a tarefa mestre. E como saída espera-se um PE p_i para se mapear a tarefa requisitada.

O algoritmo começa verificando os vizinhos mais próximos da tarefa mestre, atribuindo, assim, à variável *dist* o valor de 1 *hop* de distância. Se nenhum mapeamento viável for encontrado durante essa iteração, a distância é incrementada, testando um novo grupo de vizinhos (linha 20). Para isto, em cada iteração da estrutura ENQUANTO (linha 3 - 21), a função *vizinhos(dist, p_m)* retorna a lista de todos PEs que estão a *dist hops* de distância do PE p_m (PE da tarefa mestre). Cada PE nesta lista é avaliado, sendo verificado primeiramente se este PE está livre ou em uso (linha 7). Depois, caso estiver livre, é calculado o custo de mapeamento para este PE como proposto em PL.

Entrada: O PE p_m onde está mapeada a tarefa mestre
Saída: O PE *melhor_pe* onde será mapeada a tarefa requisitada

```
1.  dist ← 1      // Inicializa a distância em nº de hops em 1
2.  menor_custo ← ∞        // Inicializa menor_custo com valor mais alto
3.  ENQUANTO menor_custo = ∞ FAÇA
4.      lista_de_vizinhos ← vizinhos(dist, pm) // Obtêm todos vizinhos de pm com distância dist
5.      PARA TODOS ELEMENTOS pi NA lista_de_vizinhos
6.          // Testa a ocupação do PE
7.          SE estado(pi) = livre ENTÃO
8.              custo_caminho ← 0 // Inicializa o custo do caminho de comunicação entre pm a pi
9.              lista_canais ← caminho_roteamento(pi,pm ) // Obtêm todos canais ci do caminho de comunicação entre pm e pi
10.             PARA TODOS ci NA lista_canais
11.                 custo_caminho ← custo_caminho + peso(ci) // Obtém o somatório de ocupação dos canais
12.             FIM PARA
13.             // Define o menor custo e o PE alvo
14.             SE custo_caminho < menor_custo ENTÃO
15.                 menor_custo ← custo_caminho
16.                 melhor_pe ← pi
17.             FIM SE
18.         FIM SE
19.     FIM PARA
20.     dist ← dist + 1 // Incrementa dist para se procurar na próxima faixa de vizinhos
21. FIM ENQUANTO
22. retorna melhor_pe
```

Figura 7 – Pseudocódigo da heurística BN.

O cálculo do custo está compreendido entre as linhas 10 e 12 do algoritmo. Para isto, primeiro obtém-se a lista de canais que compõem a comunicação entre p_m e p_i (PE testado no momento), utilizando-se a função *caminho_roteamento(pi, p_m)* (linha 9). Esta função retorna uma lista de canais conforme o algoritmo de roteamento utilizado. A variável *custo_caminho* é o custo de caminho obtido através da soma dos pesos

(ocupação) de todos os canais da lista de canais. Se o custo de caminho obtido for menor que o obtido anteriormente, a variável *menor_custo* é atualizada, e é feito o mesmo para a variável *melhor_pe* (linhas 14 a 16). Finalmente, tendo-se obtido um PE, o algoritmo termina e é retornado o melhor PE para se mapear a tarefa requisitada.

2.2.3.4 Nearest Neighbor e Best Neighbor Multitarefa

As heurísticas NN e BN são estendidas para suportar mapeamento multitarefa em [SIN09a]. Para isto, o caminho de procura destas heurísticas é modificado, verificando primeiramente a possibilidade de mapear a tarefa solicitada juntamente no PE da tarefa que a solicitou. Na Figura 8, pode-se ver o pseudocódigo da heurística NN multitarefa. A única modificação realizada é que se inicia o caminho de procura com o número de *hops* igual a zero, ou seja, no PE que solicitou a tarefa a ser mapeada, como se pode ver na linha 1. O mapeamento multitarefa limita o número máximo de tarefas que podem ser mapeadas em um mesmo PE. Isto porque cada PE acessa uma memória de tamanho fixo, dividida em páginas. Em cada uma destas páginas, que também têm um tamanho fixo, pode-se alocar uma tarefa. Assim, a função estado(p_i) (linha 6) retorna que um PE está livre, se ele tem páginas livres de memória para poder se alocar uma nova tarefa.

Entrada: O PE p_m onde está mapeada a tarefa mestre
Saída: O PE p_i onde será mapeada a tarefa requisitada

```
1.  dist ←0      // Inicializa a distância em nº de hops em 1
2.  ENQUANTO dist<=tamanho_da_NoC  FAÇA
3.      lista_de_vizinhos ←vizinhos(dist, p_m) // Obtêm todos vizinhos de p_m  com distância dist
4.      PARA TODOS ELEMENTOS p_i NA lista_de_vizinhos
5.          // Testa a ocupação do PE
6.          SE estado(p_i) = livre ENTÃO
7.              retorna p_i  //Termina a execução e retorna p_i
8.          FIM SE
9.      FIM PARA
10.     dist ←dist + 1 // Incrementa dist para se procurar na próxima faixa de vizinhos
11. FIM ENQUANTO
```

Figura 8 - Pseudocódigo da heurística NN multitarefa.

No caso da heurística BN, são realizadas as mesmas modificações feitas para NN, visto que, como mencionado, BN utiliza o mesmo caminho de procura da heurística NN.

2.2.4 Resultados Obtidos por [CAR10]

As heurísticas de referência aqui apresentadas foram avaliadas em [CAR10], onde foram feitas comparações em relação à carga dos canais, ocupação da rede, latência de pacotes, tempo de execução, complexidade dos algoritmos e número de *hops*. Para avaliação das técnicas de mapeamento foi utilizado um ambiente de simulação que utiliza uma NoC descrita em VHDL RTL, enquanto os PEs são descritos em SystemC TLM. A NoC utilizada possui topologia malha 2D com o algoritmo de roteamento XY.

Primeiramente é feita uma comparação entre as heurísticas de mapeamento dinâmico. Para isto, é utilizado um MPSoC heterogêneo de tamanho 8x8, executando três cenários de teste que variam o tipo de aplicação (i.e. *pipeline*, árvore e genéricas), o

número de tarefas de uma aplicação (i.e. de 5 a 10 tarefas), a taxa de injeção de dados, e o tipo de recursos do sistema. A heurística FF foi utilizada como referência de pior caso.

Na Tabela 1, é mostrado um resumo dos resultados obtidos por Carvalho normalizados em relação à heurística FF. Observa-se que as heurísticas dinâmicas NN/PL/BN geram resultados médios similares. *Na presente Dissertação são utilizadas as heurísticas de mapeamento dinâmico NN e BN como referência, dado que o tempo de execução para a heurística PL é superior às demais (complexidade $O(x^3)$). Além disso, a NN é utilizada como referência por ser utilizada como heurística de comparação em diversos trabalhos* [FAR08][SIN09a][SIN09b][SIN10][WIL09].

Tabela 1 – Comparação entre as Heurísticas de Mapeamento Dinâmico, normalizadas em relação à heurística FF.

Métricas de Desempenho	Heurística			
	FF	**NN**	**PL**	**BN**
Complexidade de Mapeamento (x é a largura NoC)	$O(x^2)$	$O(x^2)$	$O(x^3)$	$O(x^2)$
Carga nos Canais (média)	1.00	0.70	**0.69**	0.70
Carga nos Canais (desv. pad.)	1.00	0.80	**0.78**	0.80
Latência de Pacotes (média)	1.00	0.85	**0.84**	0.85
Latência de Pacotes (desv. pad.)	1.00	**0.66**	0.98	1.19
Tempo de Execução (Vol)	1.00	**1.00**	1.09	1.03
Tempo de Execução (10xVol)	1.00	**0.98**	0.99	0.99

O Autor também avalia o custo do mapeamento dinâmico frente ao mapeamento estático. Para isto são comparadas as heurísticas dinâmicas PL e BN com os algoritmos estáticos SA e *Tabu Search* (TS). A comparação foi realizada através de uma aplicação executando em um MPSoC homogêneo de dimensões 5x4. Os resultados obtidos neste teste são ilustrados na Tabela 2, normalizados de acordo com os resultados do algoritmo SA, *utilizado como referência de mapeamento estático na presente Dissertação*. PL e BN apresentam respectivamente 4 e 3% de aumento do tempo de execução em relação ao algoritmo SA, mostrando o baixo impacto das heurísticas de mapeamento dinâmico no tempo total de execução. A energia consumida na comunicação, PL e BN apresentam 38% mais consumo comparado ao SA.

Tabela 2 - Comparação das Técnicas de Mapeamento Dinâmico e Estático com apenas uma aplicação mapeada no sistema, normalizados de acordo com o algoritmo SA.

Métricas de Desempenho	Dinâmicas		Estáticas	
	PL	**BN**	**SA**	**TS**
Ocupação dos Canais (média)	1.07	1.06	1.00	0.96
Ocupação dos Canais (desv. pad.)	0.97	0.92	1.00	0.94
Latência de Pacotes (média)	1.01	1.01	1.00	1.01
Latência de Pacotes (desv. pad.)	0.99	0.99	1.00	1.00
Número de *Hops*	1.14	1.14	1.00	1.26
Tempo Total de Execução	1.04	1.03	1.00	1.01
Energia de Comunicação	1.38	1.38	1.00	1.11

2.3 Mapeamento Dinâmico

Smit et al. [SMI05] propõem um método iterativo hierárquico para o mapeamento de aplicações com reserva de recursos em SoCs heterogêneos, baseados em NoC. O método visa reduzir o consumo de energia aliado à manutenção da Qualidade de Serviço (QoS do inglês, Q*uality of Service*) requerida pelo sistema. No método proposto, primeiramente cada tarefa é atribuída a um tipo de recurso do sistema (e. g. FPGA, DSP), de acordo com seus requisitos de desempenho. Depois, cada tarefa é mapeada a um dos recursos disponíveis daquele tipo, procurando-se minimizar a distância entre tarefas comunicantes. Ao final, o mapeamento resultante é verificado, e caso não atenda as necessidades da aplicação, uma nova iteração é necessária. Um único experimento que propõe o mapeamento de uma aplicação de 13 tarefas em um SoC de dimensão 4x4 é utilizado para a avaliação do método proposto, que é comparado com outros dois métodos de mapeamento. O primeiro método de comparação é o exaustivo, executado por 10 horas até encontrar a solução ótima. Já, o segundo é o algoritmo *minWeight* apresentado em [SMI04a][SMI04b]. Como resultado, o algoritmo *minWeight* apresentou uma solução 5% pior comparada ao método exaustivo, o que é considerado bom pelos Autores, porém segundo os mesmos este algoritmo apresenta baixa escalabilidade e flexibilidade. Já o método iterativo hierárquico atinge a mesma solução do método exaustivo (solução ótima) com apenas três iterações, ou seja, em muito menos tempo, tornando-se, segundo os Autores, um método promissor a ser explorado em demais experimentos.

Ngouanga et al. [NGO06] apresentam uma heurística de Força Direcionada para o mapeamento de tarefas em um MPSoC homogêneo, baseado em NoC. A heurística seleciona o posicionamento para uma nova tarefa de acordo com uma força atrativa proporcional ao volume de comunicação e a distância entre as tarefas. Os autores também avaliam o algoritmo *Simulated Annealing* (SA). Os resultados mostram que a heurística de Força Direcionada é mais rápida quando comparada ao *SA* (uma ordem de grandeza), porém o número de iterações utilizado pelo *SA* não é mencionado. A distância média total do caminho de comunicação entre tarefas é equivalente em ambos os algoritmos.

Hölzenspies et al. [HÖL07] investigam técnicas de mapeamento com requisitos de tempo real, considerando aplicações de *streaming* mapeadas em MPSoCs heterogêneos. Para isto, é proposto um método para estimar o desempenho do mapeamento destas aplicações. Segundo os Autores, o desempenho das aplicações executando em um MPSoC é influenciado, entre outros fatores, pela organização de memória dos PEs, as frequências de relógio dos PEs e a NoC. Por este motivo, métricas relativas à arquitetura de *hardware* utilizada (i.e. vazão, latência e energia) são integradas aos modelos de desempenho das aplicações em tempo de projeto. Os recursos do MPSoC são gerenciados por um sistema operacional (SO) que utiliza estas informações coletadas

sobre a aplicação e a arquitetura com o objetivo de satisfazer os requisitos de QoS, otimizar o uso de recursos e minimizar o consumo de energia. Durante a execução, o SO determina quando a ferramenta de mapeamento é chamada e, quando uma aplicação deve ser migrada para executar mais eficientemente. Esse trabalho concentra-se nos métodos de estimativa de desempenho de uma aplicação. Já em [HÖL08], o processo de mapeamento é mais bem detalhado, utilizando uma abordagem de mapeamento iterativo hierárquico como em [SMI05]. O processo de mapeamento utiliza os modelos de desempenho das aplicações propostos no trabalho anterior aqui relatado. No trabalho, o algoritmo de mapeamento é executado sobre um processador ARM que leva 4ms para apresentar uma solução de mapeamento adequada. Os Autores não fornecem comparações com outros métodos.

Chou e Marculescu [CHO07] apresentam uma estratégia incremental para mapeamento de tarefas em MPSoCs homogêneos baseados em NoC. Os PEs conectados à NoC têm vários níveis de tensão, enquanto que a própria rede (incluindo canais, roteadores, etc) tem seu domínio de frequência e tensão próprio. Um gerente global é responsável por encontrar uma área contígua para mapear uma aplicação, bem como para definir a posição das tarefas dentro desta área. Segundo os Autores, esta estratégia evita a fragmentação do sistema e tem como objetivo minimizar o consumo de energia de comunicação. Em [CHO08], os Autores estendem o trabalho para considerar também um modelo de comportamento do usuário no mapeamento das tarefas. O comportamento do usuário alimenta um perfil de operação da aplicação, que contém dados sobre sua periodicidade e volume de dados transferidos entre suas tarefas. Duas estratégias de mapeamento são investigadas. A primeira delas consiste no método adotado no trabalho anterior [CHO07]. A outra estratégia define um formato de região para uma dada aplicação, e realiza transformações geométricas (como rotações) se necessário, para em seguida mapear a aplicação no MPSoC. Para aplicações reais, considerando as informações do comportamento do usuário, os Autores obtiveram em torno de 60% de redução no consumo de energia em relação a um cenário de distribuição aleatória.

Al Faruque et al. [FAR08] propõem um esquema de mapeamento distribuído baseados em agentes. O esquema proposto divide o sistema em *clusters* virtuais. Um agente de *cluster* (CA, em inglês *cluster agent*) é responsável por todas as operações de mapeamento dentro de um *cluster*. Agentes globais (GA, em inglês *global agent*) armazenam informações sobre todos os *clusters* da NoC e usam uma política de negociação com os CAs, a fim de definir em qual *cluster* será mapeada uma dada aplicação. Um GA, primeiramente, tenta encontrar um *cluster* apropriado para mapear uma aplicação; se nenhum *cluster* apropriado é encontrado, o GA tenta usar a migração de tarefas para tornar um *cluster* apropriado para o mapeamento; enfim, se nenhum *cluster* apropriado e nenhum *cluster* candidato para migração de tarefas forem

encontrados, então o conceito de re-clusterização (os *clusters* do sistema são definidos novamente) é utilizado. O esquema de mapeamento distribuído proposto gera 10,7 vezes menor tráfego de monitoramento em comparação a um esquema centralizado para uma rede de dimensão 64x64. Além disso, atinge um esforço de mapeamento 7,1 vezes menor em relação à heurística *Nearest Neighbor*, proposta em [CAR10], em uma NoC de dimensão 64x32. A forma em que é modelada a arquitetura de referência utilizada para a avaliação do trabalho não é mencionada. Um melhor detalhamento da arquitetura se faz necessário, visto que dependendo da modelagem desta arquitetura a avaliação de desempenho pode ser imprecisa.

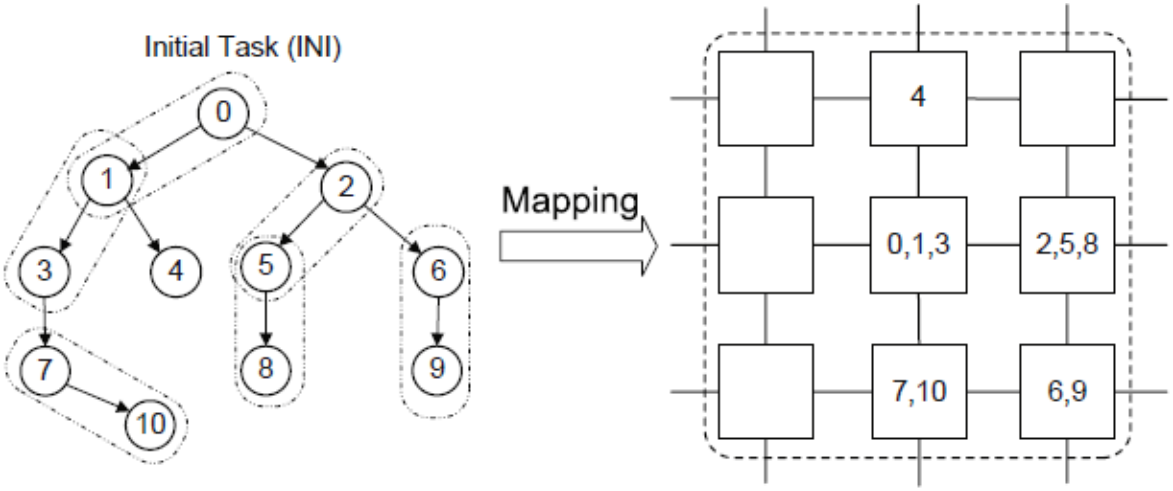
Mehran et al. [MEH08] apresentam um algoritmo de mapeamento dinâmico espiral (DSM, do inglês *Dynamic Spiral Mapping*), para mapear de forma otimizada os núcleos em uma rede malha 2D. O que se pretende no trabalho é otimizar o tempo de reconfiguração, ou seja, o tempo necessário para uma aplicação reorganizar a configuração atual de tarefas em uma nova. São apresentadas duas abordagens baseadas no DSM: o FDSM (do inglês, *Full Dynamic Spiral Mapping)*, mapeamento dinâmico espiral completo; e o PSDM (do inglês, *Partial Dynamic Spiral Mapping),* mapeamento dinâmico espiral parcial. O DSM é baseado em sua versão estática: o SSM (do inglês, *Static Spiral Mapping)*, que também é utilizado no processo de mapeamento proposto. No algoritmo de mapeamento em espiral as tarefas com grande transferência de dados entre si são dispostas o mais próximo possível umas das outras, sendo que as tarefas com maior prioridade são mapeadas espiralmente do centro para a borda da rede.

O processo de mapeamento começa com a utilização do SSM no pré-processamento. Depois, durante o tempo de execução, uma das duas abordagens baseadas no DSM é utilizada. No FDSM, a cada vez que o grafo de aplicação é modificado, é parada a execução da aplicação e executado novamente o SSM para reconfigurar todas as tarefas da aplicação. Já no caso do PDSM, a reconfiguração é parcial, onde apenas tarefas necessárias são reconfiguradas. Segundo os Autores, a vantagem do PDSM é poupar tempo de reconfiguração na rede, enquanto a desvantagem é que esta abordagem pode não gerar um mapeamento otimizado. Os resultados experimentais mostraram que o PDSM apresentou vantagens na redução do tempo de configuração em 82% dos casos em comparação ao FDSM, com redução entre 5% e 28% no tempo para mapear a aplicação.

Carvalho et al. [CAR10] apresentam heurísticas de mapeamento que visam a redução do congestionamento em MPSoCs baseadas em NoC. As heurísticas tentam reduzir o congestionamento da rede, aproximando tarefas comunicantes e reduzindo a carga do caminho de comunicação entre tarefas. Segundo os autores, o custo das heurísticas de mapeamento dinâmico, em relação ao mapeamento estático, é de 10% na ocupação de canal, de 8,5% em latência, 3,5% em tempo de execução total e 15,5% no consumo de energia de comunicação. Eles argumentam que é um *overhead* aceitável,

considerando as vantagens oferecidas pelo mapeamento dinâmico. Entre estas vantagens podem ser apontadas: *(i)* sistemas menores podem ser utilizados, já que apenas as tarefas a serem executadas são obrigadas a serem mapeadas no sistema, *(ii)* o número de tarefas pode ser superior aos recursos do sistema disponíveis, *(iii)* a inclusão de novas aplicações em tempo de execução estende a usabilidade do MPSoC. A Seção 2.2 apresenta este trabalho mais detalhadamente.

Singh et al. estendem em [SIN09a] as heurísticas de mapeamento dinâmico NN e BN propostas por Carvalho et al. [CAR10] para suportarem um mapeamento multitarefa. Em [SIN09b] é proposta uma técnica de agrupamento (*clustering)* de tarefas que tenta maximizar o número de pares de tarefas comunicantes em um mesmo PE. Todas as aplicações são modeladas como grafos em forma de árvore e possuem apenas uma tarefa inicial, como pode ser visto na Figura 9(a). Para a realização do mapeamento a NoC é dividia em regiões (em inglês, *clusters*). A tarefa inicial de uma aplicação é mapeada no centro de uma região. Os limites de uma região são virtuais, possibilitando o mapeamento de tarefas de uma aplicação, se for necessário, fora de uma região. Na Figura 9(b) é mostrada uma região de parte do sistema utilizada para o mapeamento da aplicação da Figura 9(a), onde é possível ver o mapeamento centralizado da tarefa inicial 0. A partir do mapeamento da tarefa inicial a técnica de agrupamento proposta mapeia as tarefas quando elas forem solicitas por suas tarefas comunicantes. Porém, para tentar maximizar o número de tarefas comunicantes em um mesmo PE, a técnica proposta é dividida em partes. Primeiramente é percorrido um dos galhos da árvore (por exemplo, de 0 → 10) e depois as tarefas de outro galho. Assim, quando é solicitado o mapeamento de tarefas pertencentes a outro galho que não é o que está sendo percorrido no momento, estas tarefas são escalonadas em uma fila para serem mapeadas quando seu galho for percorrido. Isto pode fazer com que algumas tarefas demorem para serem mapeadas, degradando o desempenho do sistema.



(a) grafo de aplicação em árvore

(b) Mapeamento da aplicação respeitando um limite de tarefas 3 tarefas por PE

Figura 9 – Mapeamento proposto por [SIN09a].

Outra técnica de agrupamento é proposta em [SIN10], também procurando maximizar o número de tarefas comunicantes num mesmo PE. Esta técnica verifica as tarefas anteriormente mapeadas em um dado PE para se tomar a decisão de mapeamento de uma tarefa solicitada neste PE: se a tarefa solicitada se comunica com alguma tarefa previamente mapeada neste PE, ela é mapeada, se não, então outro PE é verificado. Além disso, uma tarefa pode ser mapeada em um PE que não contenha nenhuma tarefa mapeada previamente. Os Autores citam que em alguns casos, pode ocorrer o fato de um PE receber apenas uma tarefa, subutilizando os recursos do sistema. Isto pode acontecer, por exemplo, no caso a seguir: é solicitado o mapeamento de uma tarefa t_1; somente a tarefa t_2 se comunica com $t_{1;}$ e t_2 está mapeada em um PE que não pode mais receber tarefas. Neste caso, segundo a técnica proposta, t_1 só pode ser mapeada em um PE p_i onde não esteja mapeada nenhuma outra tarefa. Desta forma, t_1 será única em p_i e, como não possui outras tarefas comunicantes, nenhuma outra tarefa poderá ser mapeada neste PE. Outro problema que pode ocorrer e que não está detalhado no trabalho, é a questão de que algumas tarefas podem demorar para serem mapeadas. Isto pode acontecer quando ocorrer o caso do exemplo anterior e não houver nenhum PE disponível que esteja completamente sem tarefas. Assim, a tarefa solicitada terá de esperar até que algum PE assuma esta condição (não ter nenhuma tarefa mapeada) ou então que seja liberada uma posição de mapeamento em um PE que conter sua tarefa comunicante. Esta abordagem de *clustering*, em comparação com uma abordagem não-*clustering*, melhora em média 15% a ocupação dos canais e consumo de energia, com algumas melhorias na latência de pacotes e tempo de execução.

Wildermann et al. [WIL09] avaliam os benefícios do uso de uma heurística de mapeamento que visa a redução de energia aliada a manutenção de desempenho. Esta heurística foi validada utilizando aplicações mestre/escravo executando em um MPSoC homogêneo baseado em NoC. O ambiente de simulação (baseado em SystemC) pode criar ou excluir dinamicamente tarefas para uma aplicação, durante a simulação, emulando uma carga de trabalho dinâmico. A heurística inclui uma métrica de proximidade de tarefas inspirada em regras conhecidas a partir de autômatos celulares, o que permite diminuir a sobrecarga de comunicação produzida por aplicações dinâmicas. Os experimentos realizados avaliam a heurística proposta quanto ao *overhead* de comunicação na rede, o tempo médio relativo de processamento das aplicações e o *deadline* de uma aplicação. O *overhead* de comunicação na rede é utilizado como indicador de energia consumida e o tempo relativo de processamento de uma aplicação é calculado dividindo-se a latência pelo *deadline* de uma aplicação. O primeiro experimento utiliza o mapeando aplicações sintéticas mapeadas em uma NoC de dimensões 9x9.Já o segundo experimento considera o mapeamento de uma aplicação de processamento de imagem adaptativo em uma NoC 7x7. Resultados mostram que a heurística NN, utilizada para comparação com a heurística proposta, apresenta o mais baixo *overhead* de

comunicação da rede em ambos os experimentos, porém apresenta o pior resultado em relação ao tempo médio relativo de processamento no primeiro experimento e não cumpre o *deadline* da aplicação do segundo experimento. Já a heurística proposta obtém o melhor compromisso entre as métricas avaliadas em ambos os experimentos.

Schranzhofer et al. [SCH10] propõem uma técnica de mapeamento dinâmico baseada em *templates* pré-computados de mapeamento (definidos em tempo de projeto). Um gerente monitora o sistema em tempo de execução e escolhe o *template* pré-computado apropriado. Em comparação com abordagens de mapeamento estático, os resultados obtidos pelos autores mostram que é possível alcançar uma redução média no consumo de energia entre 40 e 45%, sendo que o *overhead* introduzido para armazenar os *templates* de mapeamento é menor que 1 KB.

2.3.1 Considerações sobre Mapeamento Dinâmico

A Tabela 3 apresenta os trabalhos relacionados classificados de acordo com a taxonomia proposta de mapeamento para as heurísticas de mapeamento dinâmico, além do objetivo de otimização destes trabalhos. A Tabela revela duas características comuns na maioria das propostas: controle centralizado e mapeamento monotarefa.

Mesmo o controle centralizado não sendo escalável, esta é a estratégia adotada na maioria dos trabalhos. O único trabalho que apresenta um controle distribuído é a proposta de Al Faruque et al. [FAR08], usando uma NoC com dimensão de até 64x64. Este trabalho, como dito anteriormente, não define como é modelada a arquitetura de referência utilizada no trabalho. NoCs de grandes dimensões, como a utilizada neste trabalho, requerem um grande tempo de simulação no caso de uso de uma modelagem de baixo nível. No caso de uso de uma modelagem abstrata, a avaliação de desempenho pode tornar-se imprecisa. Além disso, a definição de quando um controle centralizado se torna um gargalo, requerendo um controle distribuído, é uma questão em aberto na literatura. Nos trabalhos revisados, as heurísticas de mapeamento são avaliadas em NoCs com topologia malha, com dimensões inferiores ou iguais a 8x8. No presente trabalho é utilizada uma arquitetura com controle centralizado. O uso de um controle distribuído é tema para trabalhos futuros.

A maioria dos trabalhos, excetuando-se Singh [SIN09a][SIN09b][SIN10], utiliza mapeamento monotarefa, atribuindo apenas uma tarefa para cada PE. Em um ambiente contendo processadores com sistema operacional multitarefa, a utilização de uma abordagem monotarefa subutiliza o uso do sistema. Além disso, a utilização de uma abordagem multitarefa, com múltiplas tarefas executando simultaneamente em um mesmo PE, reduz a comunicação no meio de interconexão do sistema, reduzindo a energia consumida na comunicação. Para comprovar isto, neste trabalho são feitas comparações entre as abordagens mono e multitarefa. O desafio para mapear várias

tarefas em tempo de execução em um mesmo processador é a forma de agrupá-las, uma vez que as abordagens de agrupamento (em inglês, *clustering)* exigem uma visão global da aplicação para um aumento no desempenho do mapeamento, como será mostrado neste trabalho.

Tabela 3 – Trabalhos relacionados classificados de acordo com a taxonomia proposta para mapeamento de tarefas.

Autor	Reserva de recursos	Multitarefa/ Monotarefa	Arquitetura	Gerenciamento de Controle	Objetivo de Otimização
Smit [SMI05] 2005	Sim	Monotarefa	Heterogêneo	Centralizado	Consumo de Energia e requisitos de QoS para as aplicações
Ngouanga [NGO06] 2006	Sim	Monotarefa	Homogêneo	Centralizado	Volume de comunicação Carga de computação
Hölzenspies [HÖL07][HÖL08] 2007/2008	Sim	Monotarefa	Heterogêneo	Centralizado	Consumo de Energia e requisitos de QoS para as aplicações
Chou et al. [CHO07][CHO08] 2007/2008	Sim	Monotarefa	Homogêneo	Centralizado	Consumo de Energia Contenção na rede
Al Faruque [FAR08] 2008	Não	Monotarefa	Heterogêneo	Distribuído	Tempo de execução e tempo de mapeamento
Mehran et al. [MEH08]	Sim	Monotarefa	Homogêneo	Centralizado	Tempo de mapeamento, consumo de energia e complexidade de mapeamento.
Wildermann [WIL09] 2009	Não	Monotarefa	Homogêneo	Centralizado	Latência de comunicação, consumo de energia e cumprimento de *deadlines* das aplicações
Carvalho [CAR10] 2010	Não	Monotarefa	Heterogêneo	Centralizado	Contenção na rede e volume de comunicação
Singh [SIN09a][SIN09b][SIN10] 2009, 2010	Não	Multitarefa	Heterogêneo	Centralizado	Contenção na rede, volume de comunicação e consumo de energia
Schranzhofer [SCH10] 2010	Sim	Monotarefa	Homogêneo	Centralizado	Consumo de Energia
Trabalho proposto	**Não**	**Monotarefa e Multitarefa**	**Homogêneo**	**Centralizado**	**Consumo de Energia**

Outra observação diz respeito à reserva de recursos. Alguns trabalhos descrevem a reserva de recursos conforme o número de tarefas, definindo, por exemplo, *templates* de mapeamento pré-computados para cada aplicação. Considerando que nem todas as tarefas sejam executadas simultaneamente, uma reserva de recursos para todas as tarefas da aplicação pode subutilizar o MPSoC, assim como exigir sistemas maiores.

Além disso, a reserva de recursos pode aumentar o tempo total de execução, visto que uma aplicação pode esperar até que haja recursos disponíveis para que todas suas tarefas sejam mapeadas. O mapeamento dinâmico sem reservas usa os recursos do sistema quando são efetivamente necessários. Por este motivo, o mapeamento sem reservas de recursos é utilizado neste trabalho.

A maioria dos trabalhos revisados utiliza modelos abstratos do sistema. Como dito anteriormente, modelos abstratos como TLM permitem simulações mais rápidas, mas não permitem uma avaliação precisa de desempenho (tempo de execução, latência, consumo de energia). Por outro lado, a modelagem RTL com precisão de ciclos de relógio fornece resultados mais precisos, porém com um longo tempo de simulação. Alguns trabalhos, como [CAR10][SIN10], adotam um modelo misto, com PEs descritos em SystemC TLM e a NoC em VHDL RTL. O presente trabalho adota esta modelagem mista, tendo por objetivo obter resultados rapidamente, com a precisão de um modelo totalmente descrito em RTL.

Por fim, o objetivo de otimização buscado por cada um dos trabalhos é variado, dependendo de restrições do sistema utilizado. Este trabalho tem como meta de otimização a redução de energia consumida na comunicação. Esta métrica foi escolhida devido ao fato que em dispositivos móveis o baixo consumo de energia é primordial.

Assim, este trabalho busca avançar o estado-da-arte em mapeamento dinâmico, e disponibiliza para a academia uma estrutura para avaliar os diferentes aspectos dos projetos MPSoCs. O principal avanço deste trabalho é em relação à implementação de heurísticas de mapeamento multitarefa, avaliadas em apenas um dos trabalhos apresentados. O presente trabalho também mostra as vantagens de uma abordagem multitarefa comparado a uma monotarefa, principalmente na questão do consumo de energia consumida na comunicação.

3. PLATAFORMA MPSOC DE REFERÊNCIA

Neste Capítulo é apresentado o MPSoC homogêneo HeMPS [WOZ07][CAR09], utilizado como plataforma de referência deste trabalho. Nesta plataforma é que são integradas as heurísticas de mapeamento dinâmico de tarefas desenvolvidas. Assim, na Seção 3.1 são abordados aspectos da arquitetura e uma ferramenta utilizada para geração da plataforma. Depois, na Seção 3.2, é abordada a infraestrutura de *software*, sendo explicado o *microkernel* que é executado nos processadores da plataforma. Na Seção 3.3 é apresentado como as aplicações são descritas para poderem ser executadas neste MPSoC. Este Capítulo, além de apresentar a plataforma de referência, também apresenta duas contribuições da presente Dissertação. A primeira compreende a integração das heurísticas propostas por Carvalho na plataforma de referência, o que exigiu a compreensão do hardware e do software da mesma (Seção 3.2.6). A segunda contribuição corresponde à geração de código sintético para aplicações a serem executadas na plataforma, a partir de diagramas de sequência UML (Seção 3.3.1).

3.1 Arquitetura e ferramenta de apoio à geração da plataforma

Os principais componentes do MPSoC HeMPS são os elementos de processamento, denominados Plasma-IP, que são interconectados pela NoC Hermes [MOR04], utilizada como infraestrutura de comunicação. Além disso, tem-se uma memória externa, chamada de repositório de tarefas. Na Figura 10 é ilustrada uma instância do MPSoC HeMPS utilizando a NoC Hermes de dimensão 2x3 interconectando os Plasmas-IP.

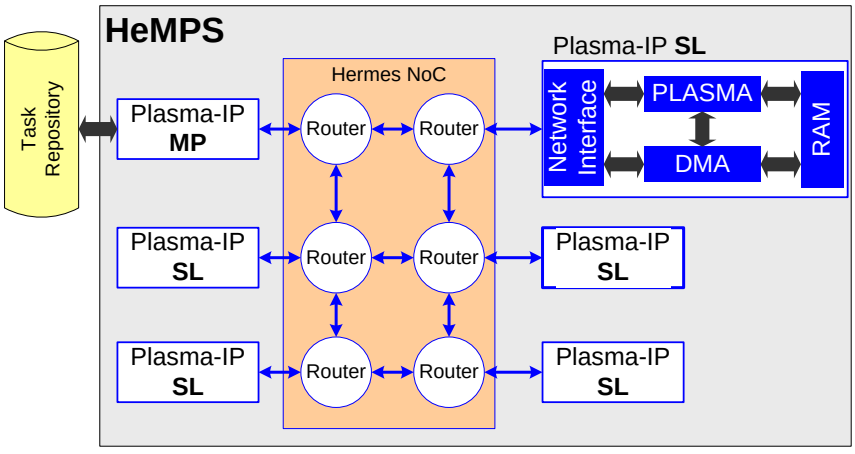


Figura 10 - Instância da HeMPS utilizando uma NoC 2 x3.

3.1.1 Plasma-IP

Os elementos de processamento do MPSoC HeMPS, chamados Plasma-IP, são subdivididos em: mestre, chamado de Plasma-IP MP, responsável pela gerência dos

recursos do sistema; e escravos, chamados de Plasma-IP SL. O MPSoC HeMPS tem uma gerência de recursos centralizada, contendo apenas um Plasma-IP MP. Os componentes do Plasma-IP compreendem:

- *um processador Plasma* [PLA10]: é um processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS. Diferentemente do MIPS original, o Plasma apresenta uma organização de memória Von Neumann. Além disso, o processador ainda oferece suporte a linguagem C e tratamento de interrupções. O Plasma do MPSoC HeMPS possui algumas modificações em relação ao Plasma original, como por exemplo, a criação do mecanismo de interrupção, exclusão de módulos e inclusão de novos registradores mapeados em memória.
- *uma memória privada*: contém o *microkernel* executado pelo processador Plasma. No caso dos Plasma-IP SL, a memória é dividida em páginas de tamanho fixo onde é feita a alocação de tarefas. É importante dizer que uma tarefa utiliza apenas uma página de memória.
- *uma interface de rede* (do inglês, *Network Interface – NI)* : realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.
- *um módulo de acesso direto à memória* (do inglês, *Direct Memory Access* - DMA): desenvolvido para auxiliar o Plasma na troca de mensagens com a NI, possibilitando ao processador continuar sua execução de tarefas sem controlar diretamente a troca de mensagens com a rede. O DMA tem como principal função transferir o código-objeto de tarefas que chegam na NI para a memória do processador.

3.1.2 NoC Hermes

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

Os roteadores da NoC possuem *buffers* de entrada, uma lógica de controle compartilhada por todas as portas do roteador, um *crossbar* interno e até cinco portas bi-direcionais. Estas portas são: *East*, *West*, *North*, *South* e *Local*. A porta Local estabelece a comunicação entre o roteador e seu núcleo local, sendo as demais portas utilizadas para ligar o roteador aos roteadores vizinhos. A arbitragem utilizada pelo roteador da NoC Hermes é *Round-Robin*. Essa política utiliza um esquema de prioridades dinâmicas, proporcionando um serviço mais justo que a prioridade estática.

Cada roteador possui um endereço único na rede. Este endereço é expresso nas coordenadas XY, onde X representa a posição horizontal e Y, a posição vertical do roteador na rede, sendo a posição 00 no canto inferior esquerdo. O algoritmo de roteamento utilizado é o XY, que envia os pacotes na rede primeiramente horizontalmente até chegar à posição de X do roteador destino, e depois percorre verticalmente, até encontrar o roteador destino.

3.1.3 Repositório de Tarefas

O MPSoC HeMPS assume que todas as aplicações são modeladas através de um grafo de tarefas, em que somente as tarefas iniciais são carregadas no sistema no momento da inicialização do mesmo. As demais tarefas são inseridas dinamicamente no sistema em função das requisições de comunicação e dos recursos disponíveis. O repositório de tarefas é uma memória externa que contém o código-objeto de todas as tarefas que executarão no sistema. O Plasma-IP MP é o único a ter acesso ao repositório de tarefas e é responsável por alocar as tarefas nos Plasma-IP SL. Todas as tarefas contidas no repositório são armazenadas em tempo de projeto, no momento da geração da plataforma.

Na Figura 11 pode ser visto o formato do repositório de tarefas, onde cada posição é uma palavra de 32 *bits*. Na primeira posição deste repositório é armazenado o número total de tarefas do sistema. Depois, o repositório se divide em duas seções: uma área contendo um cabeçalho para cada tarefa e uma área de dados contendo os códigos-objeto das tarefas. Tanto os cabeçalhos, como os códigos-objeto são ordenados do menor ao maior identificador único de uma tarefa. O cabeçalho de uma tarefa contém:

- o identificador único da tarefa;
- o tamanho do código-objeto da tarefa;
- o processador onde deve ser alocada a tarefa, caso esta seja mapeada em tempo de projeto; ou um valor que indica que a tarefa deve ser mapeada dinamicamente;
- o endereço inicial no repositório de onde está armazenado o código-objeto da tarefa.

O ordenamento das tarefas e o tamanho fixo do cabeçalho de uma tarefa permitem o fácil acesso às informações de uma tarefa pelo *microkernel* do Plasma-IP MP. Este acesso é realizado através da obtenção do endereço de memória onde está contido o cabeçalho de uma tarefa, a partir da Equação 1:

$$End\ \ Mem\ \ (ID) = 1 + (tam_{\ cabeçalho}\ \ * ID) \quad (1)$$

Nesta equação, deseja-se encontrar o endereço de memória de uma tarefa com

identificador ID. Para isso, soma-se 1 (referente à posição contendo o tamanho total de tarefas) com a multiplicação do ID pelo tamanho do cabeçalho de uma tarefa $tam_{cabeçalho}$. Por exemplo, o cabeçalho da tarefa 5 estará no endereço 21 (1 + (4 x 5)). Assim, acessando-se a posição 21 do repositório, pode-se obter as informações da tarefa, como o endereço inicial de memória onde está alocado seu código-objeto.

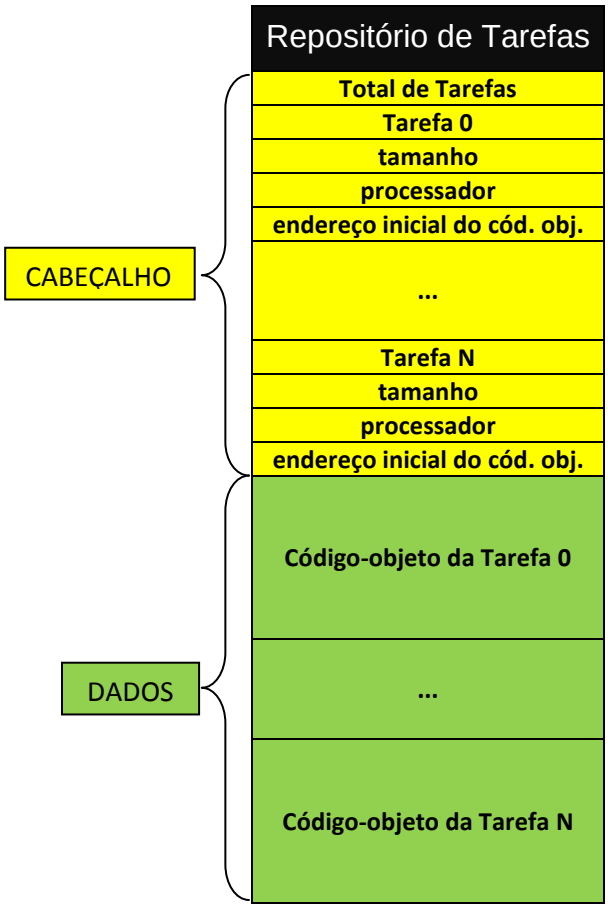


Figura 11 - Formato do repositório de tarefas.

3.1.4 Infraestrutura de Monitoramento da NoC Hermes

Este trabalho utiliza a infraestrutura de monitoramento proposta por Marczak [MAR10]. Esta infraestrutura serve como base para algumas heurísticas de mapeamento que foram implementadas (i.e. BN e BN multitarefa). Esta infraestrutura de monitoramento analisa o desempenho da NoC Hermes através de monitores inseridos nas portas de entrada dos roteadores da rede. Estes monitores têm por finalidade contar a quantidade de *flits* que passam por cada porta em uma determinada janela de tempo. Dessa forma, este monitoramento indica a vazão em cada porta de entrada e permite, assim, determinar a carga da rede.

As informações geradas por cada monitor são enviadas a um módulo gerador de pacotes de controle (GPC). Este módulo tem por finalidade enviar periodicamente pacotes de controle a um sistema de controle de monitoramento (MSA- Monitoring Service Access

Point), onde as informações de todos os monitores são coletadas e utilizadas para a tomada de ações de controle do sistema.

A estrutura do roteador da NoC Hermes contendo monitores nas portas dos roteadores e o módulo gerador de pacotes pode ser visto na Figura 12. Como há somente uma rede como infraestrutura de comunicação, pode-se ver também que são utilizados multiplexadores para que sejam transmitidos dados provenientes do IP (PE), ou seja, das aplicações, ou dados de monitoramento provindos do GPC.

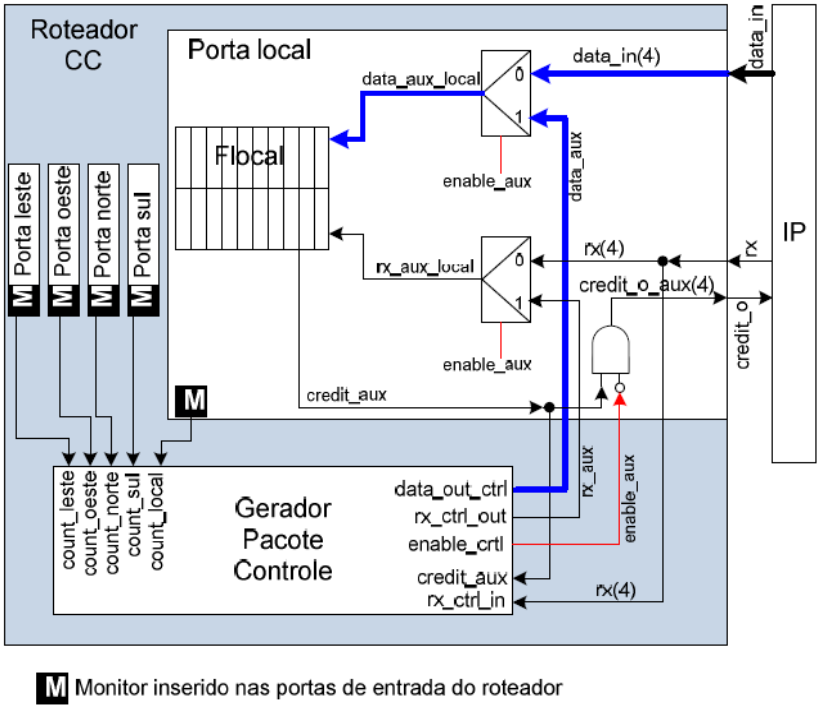


Figura 12 - Estrutura do roteador da NoC Hermes com monitoramento [MAR10], destacando-se a estrutura interna da porta local. As demais portas possuem a mesma estrutura.

O módulo gerador de pacotes recebe as informações sobre o tráfego provindo das cinco portas dos roteadores, encapsulando-as em um pacote como o visto na Figura 13. Este pacote tem um tamanho total de 10 *flits*, sendo quatro para o cabeçalho (*header*) e seis para o corpo de dados (*payload*). Para compor o cabeçalho do pacote tem-se o campo *target*, em que é determinado o destino do pacote; o campo *size*, que possui o tamanho total do corpo de dados do pacote e os campos *serviceH* e *serviceL* que possuem a parte alta e a parte baixa da palavra que identifica o serviço de monitoramento no *microkernel* do Plasma-IP MP. Este serviço é denominado MONITORING_PACKET, como será visto na Seção 3.2.3. Já para os seis campos que compõem o corpo de dados tem-se: *source*, que identifica qual é o roteador gerador do pacote de controle e *peast*, *pwest*, *pnorth*, *psouth* e *plocal*, que contém a informação da quantidade de *flits* que passaram pelas portas leste, oeste, norte, sul e local do roteador gerador do pacote de controle. Estes pacotes são enviados de maneira periódica por cada roteador ao Plasma-

IP MP, sendo este período uma janela de tempo que é definida pelo projetista da rede.

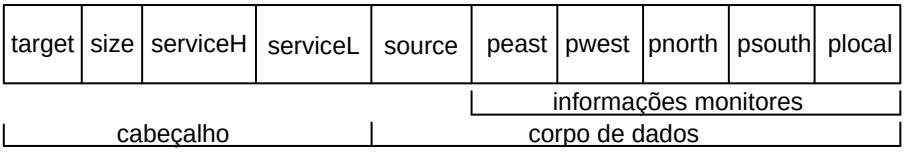


Figura 13 - Pacote de controle gerado pelo GPC.

A infraestrutura de monitoramento criada está inserida no *microkernel* do Plasma-IP MP do MPSoC HeMPS, que é o responsável pelo MSA. O *microkernel* trata o monitoramento como um serviço que recebe e identifica pacotes de controle, coletando seus dados e os armazenando em estruturas de dados. Estas estruturas são compostas por cinco matrizes, uma para cada porta dos roteadores da NoC. Cada posição das matrizes contém o volume de comunicação, em número de *flits*, que passa por aquela porta do roteador. Estas matrizes são utilizadas como suporte às heurísticas de mapeamento BN mono e multitarefa, fornecendo a carga nos canais da rede.

3.1.5 HeMPS Generator

Um ambiente automatizado, denominado HeMPS Generator [CAR09], é responsável pala a geração da plataforma HeMPS. A Figura 14 apresenta a interface gráfica principal do HeMPS Generator.

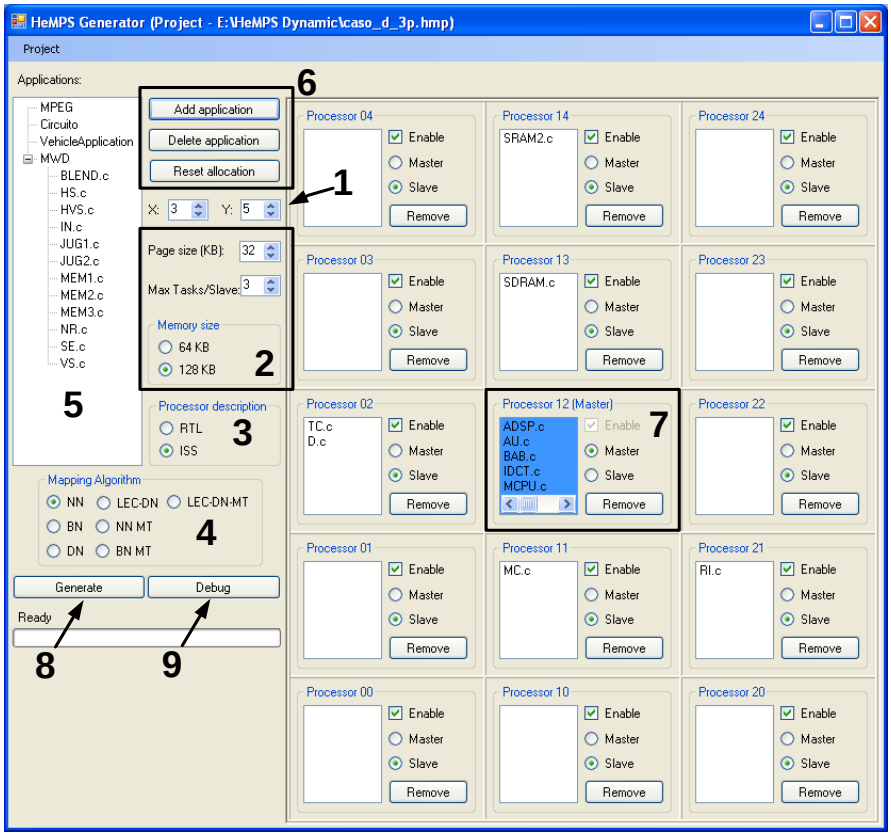


Figura 14 - Interface gráfica principal do HeMPS Generator

Na tela principal do ambiente pode-se personalizar a geração da plataforma. Primeiramente, o número de processadores da plataforma pode ser definido através dos parâmetros *X* e *Y* (1, na figura), os quais também representam cada uma das dimensões da rede malha utilizada. Depois, pode-se definir o tamanho de cada página de memória assim como o tamanho total de memória, sendo que através de uma função destes dois parâmetros é definido o número máximo de tarefas executadas pelos PEs escravos (2). Além disso, para permitir a exploração mais rápida do espaço de projeto, processadores e memórias locais podem ser modelados usando ISS (em inglês, *Instrution Set Simulator*) e modelos C/SystemC respectivamente (3). Também foi inserido no contexto deste trabalho, a possibilidade de escolher qual das heurísticas de mapeamento dinâmico será utilizada (4).

No lado esquerdo da tela principal da ferramenta (5) pode-se ver um painel contendo as aplicações adicionadas ao sistema, subdivididas pelas suas tarefas. Uma nova aplicação pode ser tanto adicionada ou removida utilizando os botões *Add application* e *Delete application* (6). O mapeamento estático das tarefas pode ser feito facilmente apenas arrastando cada tarefa ao processador escravo desejado. Já as tarefas contidas no processador mestre (7) corresponderão ao repositório de tarefas do sistema e serão mapeadas dinamicamente. A alocação das tarefas pode ser zerada através do botão *Reset Allocation* (6).

O botão *Generate* (8) realiza a integração *software-hardware* do sistema através da geração de códigos-objeto. Esta ação gera os *mickokernels* para os PEs escravos e mestre, e o repositório de tarefas contendo o cabeçalho e o código-objeto de todas as tarefas. Após essa geração, pode-se simular o sistema através de uma simulador RTL, como o *ModelSim*. Por fim, o botão *Debug* (9) abre uma janela onde podem ser vistas as mensagens de depuração de cada PE.

3.2 Microkernel

Cada um dos processadores do sistema executa um *microkernel*. Enquanto o *microkernel* do Plasma-IP SL tem como funções principais o suporte a serviços de execução multitarefa e comunicação entre tarefas, o Plasma-IP MP executa um *microkernel* que contém funções de gerência dos recursos do sistema, como por exemplo, o mapeamento de tarefas.

O *microkernel* do Plasma-IP MP é alocado inteiramente em sua memória privada. Já os Plasma-IP SL têm sua memória divida em páginas de tamanho fixo. Na primeira página é alocado o *microkernel* e nas demais são alocados código-objeto das tarefas que estão executando neste PE. O tamanho de cada página pode ser parametrizado de acordo com o tamanho da memória disponível.

O *microkernel* possui estruturas de dados para o controle de tarefas. Para o

gerenciamento de execução das tarefas há estruturas chamadas TCBs (*Task Control Block*) para cada uma das tarefas. Uma TCB contém, entre outros dados, os valores dos registradores do Plasma que são usados para salvamento e recuperação de contexto de execução das tarefas. Além disso, outras estruturas são utilizadas para guardar a localização das tarefas no sistema, para armazenar a ocupação dos PEs do sistema e requisições de serviços ainda não atendidas.

A estrutura do *microkernel* em níveis pode ser vista na Figura 15. No primeiro nível está o serviço de inicialização do sistema (*boot)* onde são inicializados os ponteiros para dados globais e para pilha, a seção de dados estáticos e as estruturas de dados para gerenciamento das tarefas. Além disso, após a inicialização, o serviço de *boot* aciona o escalonador de tarefas nos processadores escravos. Acima do *boot*, no Nível 2, encontram-se os *drivers* de comunicação. Por fim, o Nível 3 é composto pelos serviços de tratamento de interrupções, escalonamento, comunicação entre tarefas e chamadas de sistema.

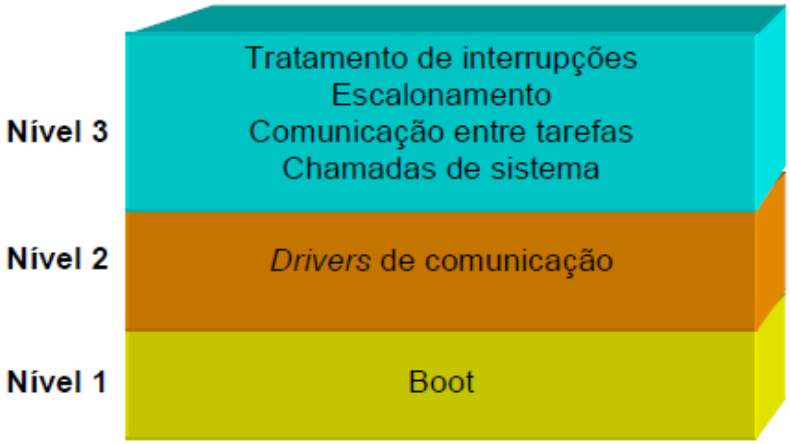


Figura 15 - Estrutura do *Microkernel.*

Neste trabalho se dará enfoque nos níveis 2 e 3 do *microkernel*, pois são diretamente relacionados com o fluxo de mapeamento de tarefas utilizado. Assim, na Seção 3.2.1 são apresentados os *drivers* comunicação. A seguir, na Seção 3.2.2 são apresentadas as chamadas de sistema. Depois, na Seção 3.2.3 é mostrado o tratamento de interrupções de *hardware,* seguido pelo escalonamento (Seção 3.2.4) e comunicação entre tarefas, descrita na Seção 3.2.5. Por fim é explicado o fluxo de mapeamento do *microkernel* (Seção 3.2.6), além de como é feita a inclusão de novas heurísticas de mapeamento.

3.2.1 *Drivers* de comunicação

Os *drivers* de comunicação são responsáveis pela comunicação entre os processadores do sistema e os módulos de hardware que compõe o Plasma-IP. As

principais rotinas desta camada são: *(i) NI_Write()* e *NI_Read()* responsáveis pela escrita e leitura dados da interface de rede e *(ii) DMA_Send()* responsável pela programação do DMA com a função de realizar leituras e escritas na memória. Além disso, através da *DMA_Send()* também é possível a programação do DMA para a realização de transmissão de blocos de memória através da NoC. Estes *drivers* de comunicação são acessados utilizando o espaço de endereçamento do *microkernel.*

3.2.2 Chamadas de Sistema

Chamadas de sistema (do inglês, *system calls*) são interrupções geradas pelo *software* que requisitam um dado serviço ao sistema operacional. No MPSoC HeMPS, as chamadas de sistema são utilizadas principalmente para prover comunicação entre tarefas e depuração, estando implementadas no *microkernel* do Plasma-IP SL. Para o tratamento da comunicação entre tarefas são utilizadas as rotinas das primitivas *WritePipe()* e *ReadPipe()*, que serão mais bem detalhadas na Seção 3.2.5. Já, para depuração, é utilizada uma rotina de *Echo()*, que envia mensagens para o Plasma-IP MP, responsável pela comunicação com o mundo externo. Além disso, para a depuração também há a rotina *GetTick()* que retorna o valor de um contador de ciclos de relógio, habilitando medidas de tempo de execução. Há ainda a rotina *Exit()* que indica que uma tarefa terminou sua execução e não deve mais ser escalonada para execução.

Quando, durante a execução, uma tarefa utiliza uma destas chamadas de sistema, o Plasma-IP SL executa um salto de posição de memória para o *microkernel*, onde está contido o tratamento da chamada. Antes disso, é feito salvamento do contexto da tarefa para posterior prosseguimento de sua execução.

3.2.3 Tratamento de Interrupções

O tratamento de interrupções só ocorre no Plasma-IP-SL, em que duas interrupções de *hardware* são implementadas. A primeira sinaliza a recepção de um pacote de rede proveniente da NI. Já a segunda, sinaliza o fim de um *timeslice* (tempo pré-determinado) de execução utilizado por uma tarefa no escalonamento.

Antes do tratamento destas interrupções é feito o salvamento e recuperação de contexto das tarefas que estão executando no momento. Para o tratamento da interrupção de *timeslice* é utilizada a rotina do escalonador, que define outra tarefa, se disponível, para executar no sistema. Este escalonador utiliza o algoritmo *Round Robin*. Já uma rotina chamada *Handler_NI()* é responsável pelo tratamento dos pacotes recebidos pela NI. O Plasma-IP MP também contém uma rotina para o tratamento destes pacotes, porém isto é feito através de *polling*.

Os pacotes que trafegam na NoC contém em seus cabeçalhos a definição dos serviços que serão tratados pelo *microkernel*. Assim, quando um pacote é recebido, a

rotina *Handler_NI()* analisa o pacote verificando seu serviço, desmontando-o, e utilizando suas informações para o tratamento daquele serviço. No tratamento dos serviços, pode ser necessária a montagem e envio de novos pacotes, assim como a leitura da memória externa. Para isto são utilizados os *drivers* de comunicação *NI_Write()* e *DMA_Send()*, explicados anteriormente.

Estes serviços são detalhados a seguir:

- MESSAGE_REQUEST: um pacote do serviço de MESSAGE_REQUEST é enviado por uma tarefa para realizar a requisição de uma mensagem a uma tarefa que está alocada em outro PE do sistema.
- MESSAGE_DELIVERY: um pacote do serviço de MESSAGE_DELIVERY contém a mensagem a ser entregue que foi requisitada por um pacote de MESSAGE_REQUEST.
- TASK_ALLOCATION: um pacote do serviço de TASK_ALLOCATION é utilizado para a alocação de uma tarefa solicitada em determinado PE da rede.
- TASK _ALLOCATED: é utilizado para informar que uma determinada tarefa foi alocada no sistema.
- TASK_REQUEST: é utilizado para solicitar o mapeamento de uma tarefa. Caso a tarefa solicitada já esteja mapeada, um pacote de resposta de LOCATION_REQUEST é enviado à tarefa solicitante contendo a localização da tarefa solicitada.
- TASK_TERMINATED: é utilizado para avisar que uma tarefa terminou sua execução.
- TASK_ DEALLOCATED: é utilizado para avisar que a tarefa terminou sua execução e pode ser liberada.
- LOCATION_REQUEST: é utilizado para requisitar e responder a localização de uma determinada tarefa.
- MONITORING_PACKET: é um pacote utilizado pela infraestrutura de monitoramento para que sejam enviadas informações sobre o estado do sistema para o Plasma-IP MP.
- DEBUG_MESSAGE: é um pacote utilizado pelos Plasma-IP SL para o envio de mensagens de depuração ao Plasma-IP MP, que a partir destas mensagens gera um arquivo externo contendo o *log* de simulação chamado de *output_master*.

Quando o *microkernel* do Plasma-IP MP recebe um pacote de serviço, e este não pode ser tratado no momento, a requisição do serviço é guardada em uma estrutura de

dados para ser tratada em um momento posterior. Um exemplo disto é uma requisição do tipo TASK_REQUEST, onde está se solicitando o mapeamento de uma tarefa. Caso não houver processadores disponíveis para o mapeamento no momento da solicitação, esta solicitação é armazenada em uma estrutura de dados e é tratada em um momento posterior.

É importante mencionar que pacotes que trafegam no sistema têm um tamanho máximo definido para o seu corpo de dados. Assim, quando uma mensagem maior que o tamanho definido precisa ser enviada, esta mensagem é dividida em vários pacotes.

3.2.4 Escalonamento

O escalonamento de tarefas implementado no *microkernel* do Plasma-IP SL é preemptivo e sem prioridades. A política de escalonamento utiliza o algoritmo *Round Robin.* Este algoritmo primeiramente define uma unidade de tempo chamada *timeslice* (do inglês, fatia de tempo). Todas as tarefas são colocadas em uma lista circular a qual é percorrida, alocando um *timeslice* de execução do processador para cada tarefa. Quando termina o *timeslice* de uma tarefa e esta não é finalizada, esta tarefa é colocada no fim da fila de escalonamento e a primeira da fila é escalonada.

Sempre que uma tarefa é escalonada, um contador implementado em *hardware* é inicializado para o controle do *timeslice*. Quando o *timeslice* é finalizado, é feito o salvamento de contexto da tarefa que está executando e uma interrupção é ativada para que o escalonador possa definir a nova tarefa a executar. Para a execução desta nova tarefa é feita a restauração do contexto da mesma.

3.2.5 Comunicação entre tarefas

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

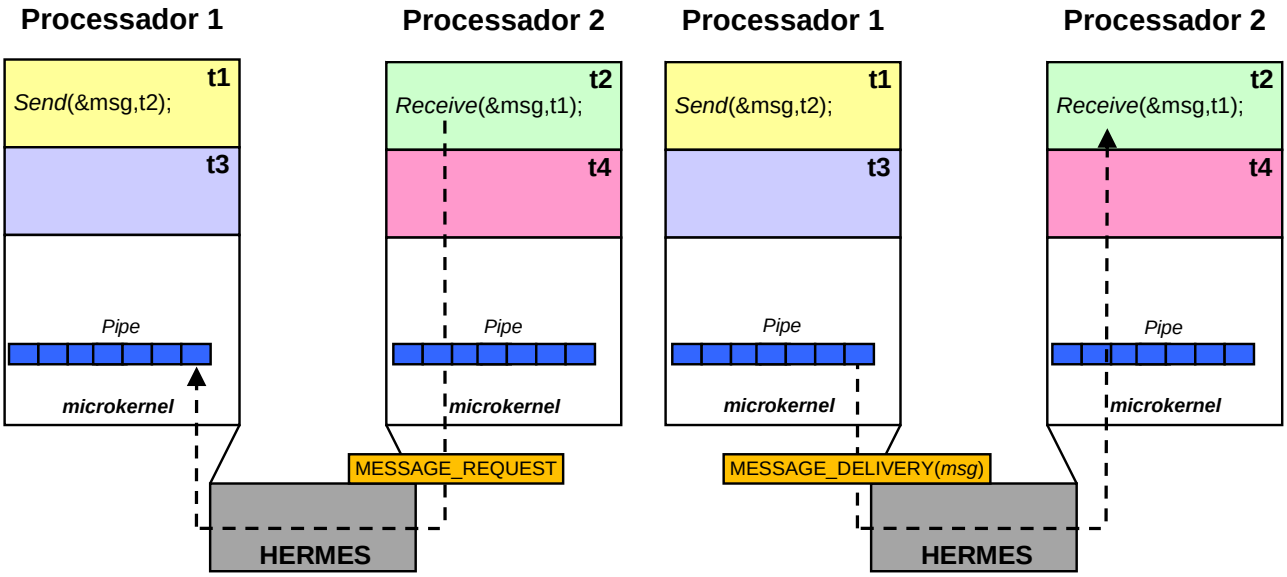
Duas chamadas de sistema são utilizadas para a troca de mensagens: *WritePipe*() e *Readpipe*(). No nível de aplicação estas chamadas de sistema são utilizadas através das primitivas *Send()* e *Receive()*, que respectivamente chamam as rotinas de *WritePipe*() e *Readpipe*() contidas no *microkernel*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

Considere a Figura 16 que ilustra o processo de troca de mensagens no MPSoC HeMPS. Quando a tarefa t_1 deseja enviar uma mensagem *msg* à tarefa t_2, é utilizada a primitiva *Send(&msg, t2)*. Esta primitiva chama a rotina de *WritePipe()* que escreve *msg* no *pipe* do Processador 1 onde está a tarefa t_1. Além disso, se a tarefa t_2 ainda não estiver mapeada, a rotina de *WritePipe()* monta um pacote de TASK_REQUEST endereçado ao

Plasma-IP MP, solicitando o mapeamento da tarefa t_2. Quando o Plasma-IP MP recebe este pacote, é realizado o fluxo de mapeamento mostrado na Figura 17, que será explicado na Seção a seguir.

Quando a tarefa t_2 deseja receber *msg* de t_1, é utilizada a primitiva *Receive(&msg, t1)* que chama a rotina de *ReadPipe()* no *microkernel*. Neste caso, podem ocorrer três situações diferentes:

- se t_1 e t_2 estiverem no mesmo PE: a rotina de *ReadPipe()* apenas lê *msg* do *pipe* do PE onde estão as duas tarefas e repassa *msg* para t_2.
- se t_1 e t_2 estiverem em PEs diferentes: a rotina de *ReadPipe()* monta um pacote de MESSAGE_REQUEST endereçado ao PE de t_1 (Processador 1) para requisitar *msg*, como pode ser visto na Figura 16(a). Quando o pacote de MESSAGE_REQUEST é recebido pelo Processador 1, o *microkernel* verifica se a mensagem solicitada está no seu *pipe.* Caso estiver, a mensagem é enviada através de um pacote de MESSAGE_DELIVERY, como pode ser visto na Figura 16(b). Em caso contrário, a solicitação da mensagem é escalonada para ser enviada quando a mensagem for alocada no *pipe*.
- se t_1 (a tarefa que envia a mensagem solicitada) ainda não estiver mapeada: a rotina de *ReadPipe()* monta um pacote de LOCATION_REQUEST para solicitar a localização da tarefa t_1.



(a) Envio de uma solicitação de mensagem através de um pacote de MESSAGE_REQUEST

(b) Entrega de uma mensagem através de um pacote de MESSAGE_DELIVERY

Figura 16 – Troca de mensagens no MPSoC HeMPS.

3.2.6 Fluxo de mapeamento no *microkernel* do Plasma-IP MP

Ao receber uma solicitação de mapeamento através de um pacote TASK_REQUEST, o *microkernel* do Plasma-IP MP executa o fluxo apresentado na Figura 17.

O primeiro passo do fluxo de mapeamento é o recebimento e identificação do pacote de TASK_REQUEST. Este pacote contém em seu *payload* informações referentes ao PE e ID da tarefa que solicitou o mapeamento, e o ID da tarefa a ser mapeada. A seguir é verificado se a tarefa solicitada já está mapeada. Caso estiver termina-se o fluxo, sendo enviado um pacote de TASK_ALLOCATED endereçado ao PE que requisitou o mapeamento da tarefa. Caso contrário, prossegue-se o fluxo.

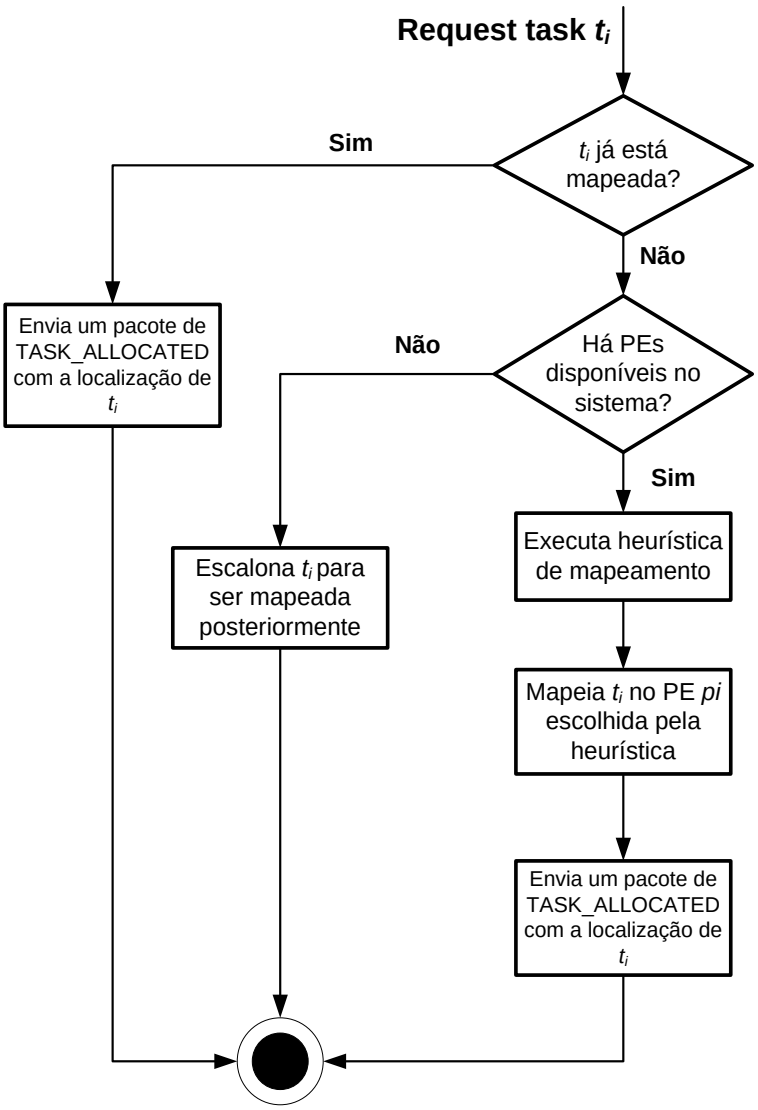


Figura 17 - Fluxo de mapeamento no *microkernel* do Plasma-IP MP.

O fluxo prossegue verificando se há PEs disponíveis no sistema para o mapeamento da nova tarefa. Em caso negativo, a tarefa é escalonada para ser mapeada em outro momento (este escalonamento está fora do contexto do presente trabalho). Em caso

afirmativo, é utilizada uma heurística de mapeamento para encontrar um PE para mapear a tarefa solicitada. O MPSoC HeMPS, antes da realização deste trabalho utilizava a heurística FF para o mapeamento de tarefas. Como já explicado, esta heurística é extremamente simples e não provê um mapeamento otimizado. A ausência de um método de mapeamento otimizado na HeMPS devia-se ao fato que o desenvolvimento da plataforma ocorreu em paralelo com o desenvolvimento das políticas propostas por Carvalho [CAR10]. *Tal limitação é explorada como motivação principal deste trabalho, que em um primeiro momento integrou as heurísticas propostas de Carvalho no microkernel, e depois propôs novas heurísticas provendo suporte a mapeamento multitarefa*.

A integração das heurísticas propostas por Carvalho no *microkernel* do Plasma-IP MP é a primeira contribuição do presente trabalho. Para isto, foi necessária a compreensão do fluxo de mapeamento mostrado nesta Seção. As heurísticas de mapeamento são, assim como o *microkernel*, descritas em linguagem C e inseridas no fluxo como uma rotina que é chamada passando-se por parâmetro a posição do PE que solicita o mapeamento da nova tarefa. Procurou-se descrever o código visando um menor tempo de computação possível, pois esta computação pode influenciar no tempo total de execução das aplicações. Estas heurísticas foram incluídas no *microkernel* conforme apresentado na Seção 2.2. Além disso, para a heurística BN, foi necessária a utilização da infraestrutura de monitoramento, como explicado anteriormente.

Após a definição do PE a ser mapeada a tarefa através da heurística de mapeamento, o Plasma-IP MP configura o seu módulo de DMA, que acessa o repositório de tarefas e transmite o código-objeto da tarefa para o Plasma-IP SL escolhido. Para isto é utilizado um pacote de TASK_ALLOCATION. Depois de mapeada a tarefa, um pacote de TASK_ALLOCATED é enviado pelo Plasma-IP MP para o PE que requisitou o mapeamento da tarefa e para o da tarefa mapeada, com o objetivo deles atualizarem a localização desta nova tarefa.

3.3 Aplicações do Usuário

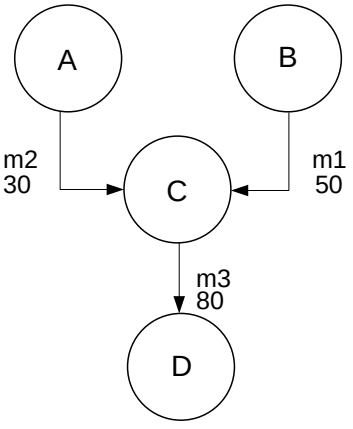
As aplicações de usuário utilizadas na HeMPS são descritas em código C e se baseiam principalmente nas primitivas de comunicação *Send()* e *Receive()*, que realizam, respectivamente, o envio e recebimento de mensagens entre tarefas, além das primitiva *Echo()* e *GetTick()*, utilizada para *debug* dos códigos. Estas primitivas, quando chamadas, utilizam uma chamada de sistema. No caso de *Send()* e *Receive()*, como explicado anteriormente, as chamadas de sistema de *WritePipe()* e *ReadPipe()* são utilizadas, respectivamente. Já as primitivas *Echo()* e *GetTick()*, utilizam as chamadas de sistema de mesmo nome.

Na Figura 18(a) pode-se ver o código em linguagem C da tarefa A da aplicação representada pelo grafo da Figura 18(b). Este grafo representa uma aplicação com 4

tarefas transmitindo mensagens entre si. Cada mensagem possui o volume de comunicação a ser transferido em número *flits*. O código possui uma estrutura de dados do tipo *Mensagem* (linha 4) que contém o tamanho (volume de comunicação) e os dados da mensagem m1 a ser enviada pela tarefa A. Na linha 14 é definido o tamanho desta mensagem com valor 30, de acordo com o grafo da aplicação. Depois, no bloco entre as linhas 15 e 17 os dados desta mensagem são preenchidos. Entre as linhas 19 e 22, esta mensagem é enviada por 10 iterações através da primitiva de comunicação *Send*(&m1,C). Esta primitiva tem como parâmetros a mensagem a ser enviada, ou seja, uma estrutura de dados do tipo Mensagem (&m1) e a tarefa destino desta mensagem (C). Além disso, nas linhas 11 e 25, são enviadas mensagens de depuração ao Plasma-IP MP, informando o início e o fim da tarefa A, através da primitiva *Echo()*. Nas linhas 12 e 24, uma primitiva *Echo()* é utilizada conjuntamente com a primitiva *GetTick()* para enviar uma mensagem de depuração contendo a contagem de ciclos de relógio no momento de início e fim da tarefa. Quando é usada a primitiva *Echo(msg)* com uma mensagem de depuração *msg*, a chamada de sistema de mesmo nome monta um pacote de DEBUG_MESSAGE com *msg* endereçado ao Plasma-IP MP.

```
1. #include "../../include/task.h"
2. #include "../../include/stdlib.h"
3.
4. Mensagem m1;
5.
6. int main()
7. {
8.
9.     int i, j;
10.
11.    Echo("Tarefa A iniciada.");
12.    Echo(itoa(GetTick()));
13.
14.    m1.tamanho = 30;
15.    for(i=0;i<30;i++){
16.           m1.dados[i] = i;
17.    }
18.
19.    for(i=0;i<10;i++){
20.           Send(&m1,C);
21.           for(j=0;j<20000;j++);
22.    }
23.
24.    Echo(itoa(GetTick()));
25.    Echo("Tarefa A terminada.");
26.    return 0;
27.}
```

(a) código em linguagem C da tarefa A



(b) grafo representando uma aplicação

Figura 18 - Geração de código a partir do grafo da aplicação.

3.3.1 Geração de código C para a Plataforma HeMPS

A segunda contribuição deste trabalho se refere à geração automática de códigos em linguagem C de aplicações para a plataforma HeMPS. Isto foi feito devido à falta de aplicações para serem utilizadas nos cenários de testes de avaliação das técnicas de

mapeamento. Assim, foi utilizado o modelo de aplicações utilizado no trabalho proposto por Ost em [OST10a]. O Autor propõe um fluxo de projeto semi-automático baseado em modelos que permitem uma rápida análise de métricas de desempenho e diferentes alternativas de projeto de MPSoCs homogêneos baseados em NoC. Para isto, são utilizados modelos de NoCs e aplicações, descritos através do ambiente Ptolemy II [LEE03]. A utilização desta abordagem para a geração de códigos se deve principalmente ao fato do Autor já possuir várias aplicações reais modeladas. Além disso, a utilização desta abordagem tem em vista uma futura integração das heurísticas de mapeamento do presente trabalho nos modelos abstratos de plataforma propostos pelo Autor, o que possibilitará uma avaliação destas heurísticas em MPSoCs maiores, o que torna-se inviável no presente trabalho devido ao grande tempo de simulação.

O Ptolemy II é um ambiente de código aberto, desenvolvido na universidade de Berkeley, que consiste de um conjunto de pacotes Java que permitem a modelagem, simulação e projeto de sistemas concorrentes. Este ambiente dispõe de uma interface gráfica, chamada Vergil (Figura 19), a qual permite a construção de modelos a partir de uma biblioteca de componentes característicos de uma modelagem orientada a atores.

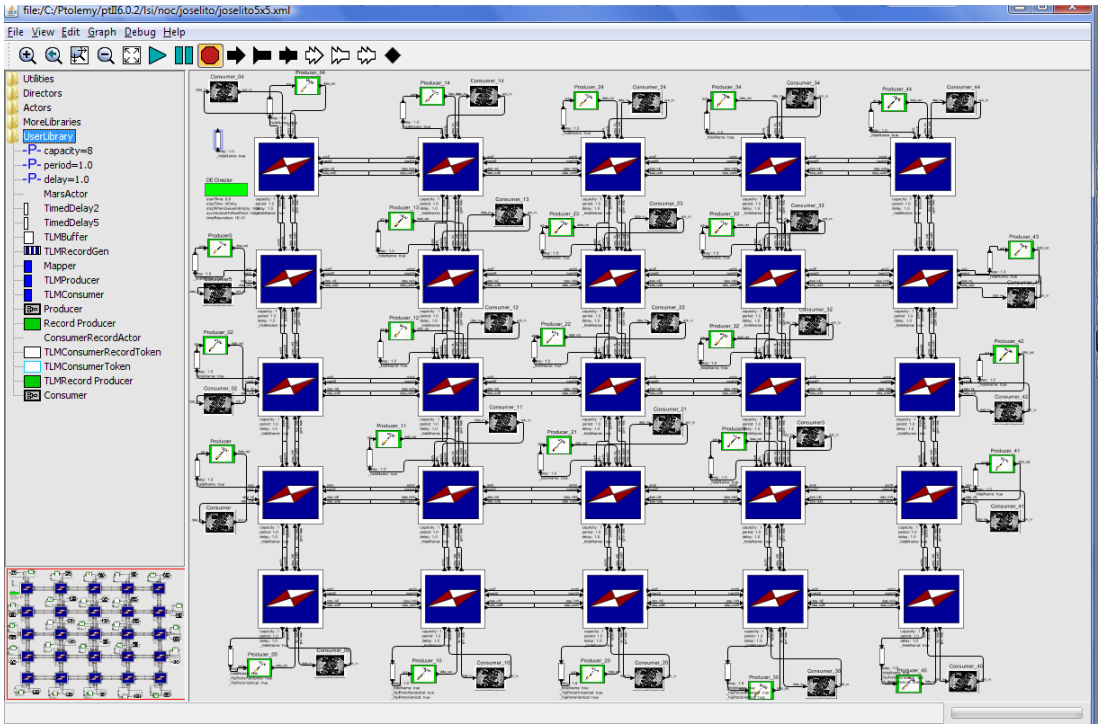


Figura 19 - Interface gráfica Vergil, ilustrando um modelo de uma NoC de dimensão 5x5.

A orientação a atores foi inicialmente, proposta como um modelo matemático de computação concorrente [HEW77] e, posteriormente, como um modelo de concorrência. Os principais componentes dessa abordagem são atores e diretores. Um diretor é responsável por comandar um conjunto de atores, definindo quando cada um deve agir e como são conectados entre si. Já atores, são elementos concorrentes que se comunicam entre si, processando dados disponíveis em suas portas de entrada e os disponibilizando

na porta de saída.

Usando idéias de orientação a atores e o ambiente Ptolemy II, modelos de aplicação executáveis foram criados, onde padrões de comunicação da aplicação são descritos a partir de diagramas de sequência UML. Um diagrama de sequência representa uma sucessão de eventos entre objetos de um sistema, procurando fornecer uma ordenação temporal a estes eventos. Para a representação de cada objeto utilizam-se linhas de vida (do inglês, *lifeline*), que são linhas verticais que representam tempo de vida de um objeto do sistema. Já os eventos, são representados por uma flecha que sai de uma linha de vida em direção a outra, indicando uma troca de mensagens entre objetos.

Neste contexto, uma aplicação é representada no diagrama de sequência de maneira que cada linha de vida representa uma tarefa da aplicação e cada evento pode ser visto como uma mensagem entre tarefas. As linhas de vida também denotam o sequenciamento de mensagens de uma aplicação, sendo que mensagens mais abaixo em uma linha de vida acontecem depois das mensagens mais acima. A Figura 20 apresenta um exemplo de um diagrama de uma aplicação, denominada *communication*, com quatro tarefas: A, B, C e D. As tarefas iniciais desta aplicação são A e B. Essas tarefas iniciam sua execução enviando mensagens para a tarefa C, que por sua vez, envia cada mensagem recebida à tarefa D.

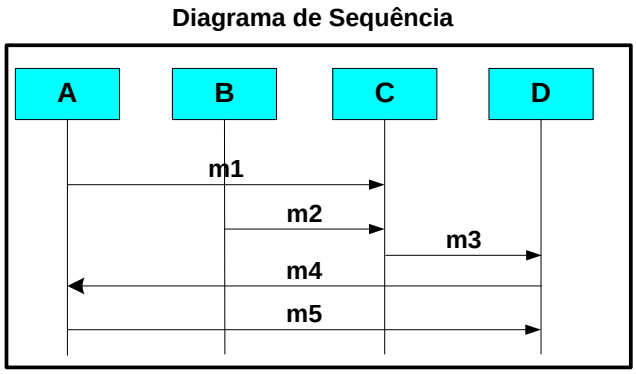


Figura 20 - Diagrama de Sequência de uma aplicação.

A abordagem adotada permite ao projetista caracterizar uma dada aplicação através de uma série de parâmetros, apresentados a seguir, os quais são utilizados para a geração de códigos (em linguagem C) de aplicações executáveis na plataforma HeMPS. Estes parâmetros são definidos através da interface gráfica do ambiente Ptolemy II. A obtenção destes parâmetros, assim como a geração do código para as aplicações são obtidos através de um arquivo Java que rege a execução dos modelos propostos durante a simulação. Dessa forma, quando é iniciada a simulação de uma aplicação sobre uma dada plataforma, automaticamente estes códigos são gerados.

Os parâmetros obtidos a partir do código Java são: *(i)* ordem das mensagens de

uma tarefa, *(ii)* volume de comunicação transmitido por cada mensagem (tamanho de uma mensagem em *flits*) e *(iii)* número de iterações que uma aplicação executará (e.g. número de *frames* a serem decodificados por uma aplicação do tipo *streaming*).

Utilizando como exemplo a aplicação modelada através do diagrama de sequência da Figura 20, a geração do código em linguagem C para a tarefa A procede da seguinte forma, considerando: (*i*) que o tamanho máximo do corpo de dados de um pacote a ser trafegado na rede tem valor 100 *flits*; (*ii*) a ordem das mensagens da tarefa A é m1, m4 e m5; (*iii*) o volume de comunicação de cada mensagem é 300 *flits* para m1, 200 *flits* para m4; e 80 *flits* para m5; (*iv*) a aplicação é executada por 10 iterações. O código sintético gerado para a tarefa A desta aplicação é apresentado na Figura 21.

```
1. #include "../../include/task.h"
2. #include "../../include/stdlib.h"
3.
4. Mensagem m;
5.
6. int main()
7. {
8.     int i, j;
9.
10.    Echo(strcat("b,A,",itoa(GetTick())));
11.
12.    for(i=0;i<10;i++){
13.
14.          for(j=0;j<100;j++) m.dados[j] = i;
15.
16.          //Comm C 300
17.          m.tamanho = 100;
18.          for(j=0;j<3;j++)Send(&m,C);
19.          Echo(strcat("s,Aplic_m1(300),",itoa(GetTick())));
20.          //Comm D 280
21.          m.tamanho = 100;
22.          for(j=0;j<2;j++)Receive(&m,D);
23.          Echo(strcat("r,Aplic_m4(200),",itoa(GetTick())));
24.          //Comm D 280
25.          m.tamanho = 80;
26.          Send(&m,D);
27.          Echo(strcat("s,Aplic_m5(80),",itoa(GetTick())));
28.          Echo(strcat(strcat(strcat("i,",itoa(i)),","),itoa(GetTick())));
29.
30.          for(j=0;j<20000;j++);
31.    }
32.
33.    Echo(strcat("e,A,",itoa(GetTick())));
34.    return 0;
35.}
```

Figura 21 - Código Sintético da Tarefa A, relativa à Figura 20.

A geração inicia através da definição das bibliotecas utilizadas no código e uma estrutura do tipo *Mensagem*, utilizada para o armazenamento das mensagens recebidas ou enviadas pela tarefa (linhas 1-5). Depois se define a rotina principal do código *main()* (linhas 6-35). Esta rotina, primeiramente contém declaração de variáveis auxiliares utilizadas (linha 8). Para a depuração do código são inseridas várias mensagens no código. Estas mensagens obedecem a um formato que é utilizado para a interpretação

das mesmas através de uma ferramenta que analisa o arquivo de *log* gerado na simulação. Na linha 10 e 33, podem-se ver mensagens que indicam, respectivamente, o início e o fim da tarefa. Nas linhas 19 e 27, pode ser vista uma mensagem que indica o envio de dados, já na linha 23, o recebimento. As mensagens de depuração de envio e recebimento contêm também informações de identificação de uma mensagem (i.e. m1, m4 e m5), além do volume de comunicação enviado. Por fim, na linha 28 é visto uma mensagem que indica o fim de uma iteração de execução da tarefa, contém o número da iteração que terminou. Além disso, todas as mensagens possuem informação da contagem de ciclos de relógio de simulação, obtidos através da primitiva *GetTick()*.

A descrição da comunicação da tarefa acontece dentro do laço *for* que indica o número de iterações que a aplicação será executada (linhas 12-31). O número de iterações é obtido através do modelo da aplicação e inserido dentro do laço *for*. Dentro deste laço, primeiramente é visto o preenchimento do corpo de dados das mensagens a com números repetidos do valor da iteração corrente (linha 14). Isto facilita a depuração por formas de onda, onde se pode verificar de qual iteração pertence determinada mensagem.

A seguir, tem-se o bloco principal da comunicação da tarefa A. Nas linhas 17 a 18, é descrito o envio da mensagem m1 para a tarefa C. Para isto, primeiro define-se o tamanho do corpo de dados da mensagem. Como o tamanho de m1 (300) excede o valor máximo suportado pelo pacote, define-se o tamanho com este valor máximo que é 100 (linha 17). Assim, para o envio de m1 é necessário o envio de 3 pacotes de 100 *flits*. Para isto, é utilizado um laço de repetição que envia através da primitiva *Send()* estes 3 pacotes, contendo mensagens de tamanho 100. Uma situação parecida acontece com o recebimento da mensagem m4 (linhas 21 a 23), que possui tamanho 200. É utilizado um laço que recebe dois pacotes de tamanho 100 (linha 22). Já no caso da mensagem m5 (linhas 25 a 27), o caso é diferente. Esta tarefa tem tamanho de 80 *flits* que é menor ao máximo permitido por pacote. Assim, primeiramente define-se o tamanho da mensagem m em 80 (linha 25) e esta é enviada através de uma única primitiva *Send()* para a tarefa D.

Para finalizar, nas linhas 16, 20 e 24 são inseridos comentários utilizados na extração das dependências das tarefas feita pela ferramenta HeMPS Generator, que será explicada na Seção 4.2.

Uma observação importante se refere ao tempo de computação de cada tarefa ainda não estar descrito no modelo, que é um dos trabalhos futuros do Autor. Pelo fato de não ser modelado ainda o tempo de computação, o código gerado descreve somente a comunicação de uma tarefa com as demais tarefas de uma aplicação, emulando um tempo arbitrário de computação (linha 30) que separa uma iteração de outra.

4. HEURÍSTICAS DE MAPEAMENTO DINÂMICO

Este Capítulo apresenta as novas heurísticas de mapeamento dinâmico, que se configuram nas principais contribuições do presente trabalho. Assim, a Seção 4.1, apresenta o mapeamento de tarefas iniciais de uma aplicação, passo inicial para a realização do mapeamento dinâmico de tarefas. Depois, a Seção 4.2, apresenta como é realizada a extração de dependência e volume de comunicação entre as tarefas, que são utilizados na implementação das novas heurísticas propostas. A seguir, a Seção 4.3, apresenta heurísticas de mapeamento monotarefa buscando otimizar as heurísticas propostas por Carvalho. E por fim, a Seção 4.4, propõe heurísticas visando o mapeamento multitarefa.

4.1 Mapeamento das Tarefas Iniciais da Aplicação

Poucas referências são encontradas na literatura, ao conhecimento do Autor, sobre o mapeamento de tarefas iniciais, as quais incluem Carvalho [CAR10] e Singh [SIN10], que utilizam a mesma abordagem. Esta abordagem leva em conta a divisão do MPSoC em *clusters,* também utilizada neste trabalho, porém se difere em dois pontos deste trabalho. Primeiro porque é considerado que uma aplicação pode possuir mais de uma tarefa inicial, ao contrário da abordagem de Singh e Carvalho, que considerava apenas uma tarefa inicial. Segundo, porque é considerado que as tarefas iniciais podem ter dependências entre si em um momento futuro, assim deve-se considerar essa possível dependência no momento do mapeamento. Levando em consideração estes dois pontos, o mapeamento de tarefas iniciais é feito estaticamente pelo projetista, de forma manual, através do conhecimento prévio do grafo das aplicações.

4.2 Extração de Dependência de Tarefas

As heurísticas desenvolvidas por Carvalho em [CAR10] não levam em conta todas as dependências entre tarefas para a realização do mapeamento de tarefas. Somente é observada a posição da tarefa a ser mapeada em relação à tarefa que requisitou o seu mapeamento, excluindo-se as outras que também interagem com a mesma. Assim, o mapeamento obtido não é otimizado devido a caminhos de comunicação não considerados durante o mapeamento. Para resolver este problema, este trabalho otimiza estas heurísticas, considerando dependências múltiplas no momento de realização do mapeamento.

A Figura 22 apresenta o diagrama de sequência da aplicação MPEG. Como dito anteriormente, este diagrama contém linhas horizontais chamadas linhas de vida, que representam cada uma das tarefas da aplicação denotando, de cima para baixo, o andamento do tempo de execução de cada tarefa. Estas linhas de vida são ligadas por

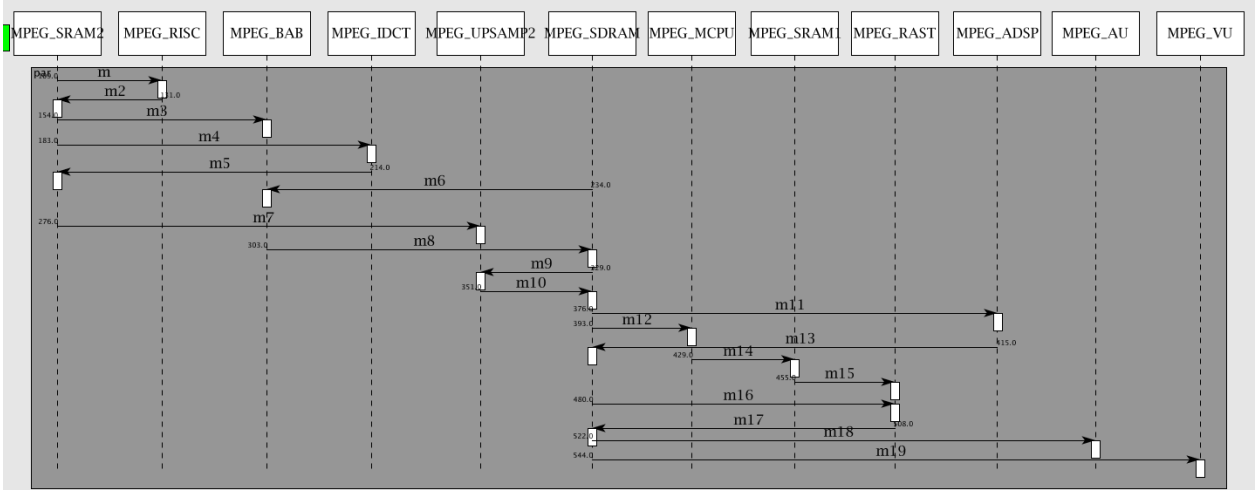


Figura 22 – Diagrama de Sequência da Aplicação MPEG

flechas que apontam o destino de uma comunicação entre duas tarefas. Assim, este diagrama mostra a ordem em que as comunicações entre tarefas acontecem, denotando assim as dependências de comunicações entre tarefas.

A aplicação MPEG tem SDRAM e SRAM2 como tarefas iniciais. Na figura pode-se observar que no total de doze tarefas, têm-se cinco que tem mais de uma dependência de comunicação. Um exemplo é a tarefa UPSAMP2 que é dependente de duas tarefas: SDRAM e SRAM2. No caso das heurísticas NN e BN, supondo que SDRAM, por exemplo, solicite o mapeamento de UPSAMP2, o mapeamento considerará somente a localização da primeira tarefa no momento do mapeamento, desconsiderando a posição de SRAM2 que também é uma tarefa dependente. Assim, as otimizações empregadas propõem que, no momento da escolha de mapeamento da tarefa UPSAMP2, busque aproximar esta tarefa de todas suas dependentes de comunicação, ou seja, as tarefas SDRAM e SRAM2. Vale ressaltar que pode ocorrer o fato de nem todas as tarefas serem mapeadas próximas as suas dependências, pois estas dependências podem não estar ainda mapeadas.

A aplicação MPEG representada pelo diagrama de sequência da Figura 22 também pode ser representada na forma de um grafo. apresentado na Figura 23, porém sem mostrar as dependências entre as tarefas. Neste grafo os vértices representam tarefas e arestas representam a comunicação entre as tarefas. Em cada aresta pode-se ver um valor que denota o volume de comunicação transferido entre as tarefas em número de *flits.* A direção desta comunicação é mostrada através de flechas que apontam para a tarefa que está recebendo o volume de dados.

Assim, outro ponto que pode ser explorado em otimizações das heurísticas desenvolvidas por Carvalho é quanto ao volume de comunicação. Como podemos ver na Figura 23, a tarefa UPSAMP2 se comunica com a tarefa SRAM2 recebendo um volume de dados de 1131 *flits*, e troca um total de 3000 *flits* com a tarefa SDRAM (recebe 1500 e envia 1500 *flits*). Neste caso, o mapeamento ideal considerando todas as dependências

de comunicação de UPSAMP2, deve levar em conta o volume de comunicação, ou seja, aproximar uma tarefa daquela(s) que ela mais se comunica. Neste exemplo, UPSAMP2 deve ser mapeada mais próximo a SDRAM.

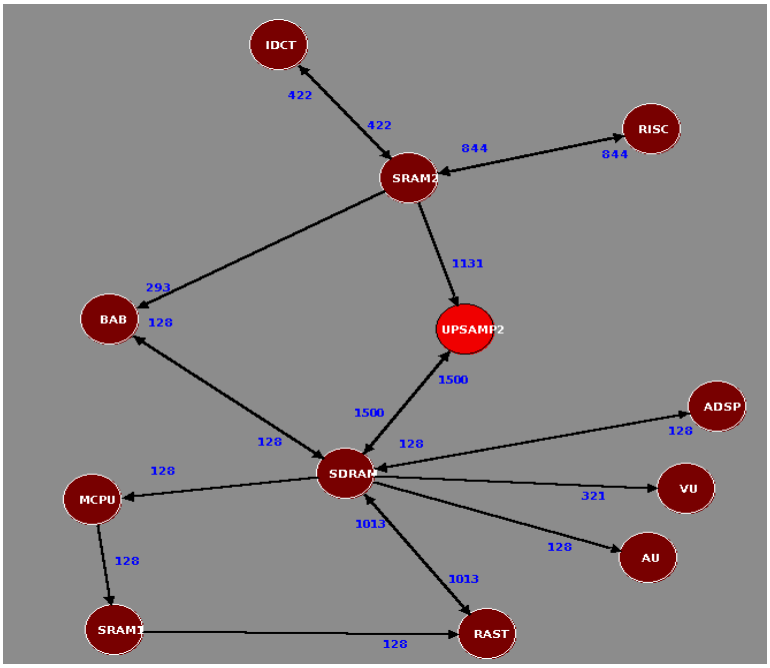


Figura 23 - Grafo de Comunicação da Aplicação MPEG.

As informações relativas às dependências de comunicação entre as tarefas, assim como o volume de comunicação, devem ser conhecidos no momento do mapeamento. Para obtenção destas informações, o presente trabalho utiliza a ferramenta HeMPS Generator. Esta ferramenta analisa o código em linguagem C que descreve cada tarefa das aplicações presentes no sistema. Este código possui um *tag* colocado como comentário, no momento da geração do código, da seguinte forma (a Figura 21, anteriormente apresentada, ilustra um código completo):

```
//Comm <tarefa> <volume_de_comunicação>
```

Assim, quando o código de uma tarefa t_i é analisado, ao se encontrar uma linha no que inicie com "`//Comm`" pode-se extrair a informação que t_i se comunica com uma tarefa "`<tarefa>`" com volume de comunicação "`<volume_de_comunicação>`". Esta informação e de todas as outras tarefas são coletadas e armazenadas em um arquivo de *header*, criado pela ferramenta HeMPS Generator no momento da geração da plataforma. Este arquivo possui uma função que é utilizada para o preenchimento de uma estrutura de dados presente no *microkernel*, em tempo de execução. Esta estrutura de dados é consultada no momento do mapeamento, para a escolha de um PE para mapear uma tarefa.

É importante mencionar que através do ambiente Ptolemy II, onde foram modeladas as aplicações (diagrama de sequência), também seria possível a extração das

dependências, assim como a geração do arquivo de *header.* Porém, a estratégia de inserir comentários no código C para a extração das dependências possibilita que o projetista gere códigos de aplicação de forma manual, sem depender do ambiente Ptolemy II.

4.3 Mapeamento Monotarefa com Dependências Múltiplas

Nesta Seção são apresentadas as heurísticas monotarefa que buscam otimizar o trabalho de Carvalho, utilizando o conceito de dependências múltiplas. As heurísticas propostas são *Dependences Neighborhood* (do inglês, vizinhança de dependências) que tem uma variante, a heurística *Low Energy Consumption – Dependences Neighborhood* (do inglês, baixo consumo de energia - vizinhança de dependências).

4.3.1 Dependences Neighborhood (DN)

A heurística *Dependences Neighborhood* (DN) utiliza apenas uma função de custo: a proximidade, em número de *hops.* Diferentemente das heurísticas NN e BN, que mapeiam a tarefa solicitada o mais próximo possível da tarefa que a solicitou, a DN considera o conceito de dependências múltiplas apresentado anteriormente, mapeando a tarefa solicitada o mais próximo possível das tarefas com que ela se comunica que já estão mapeadas.

Esta heurística utiliza dois caminhos de procura diferentes: um para quando se tem apenas uma tarefa dependente daquela que se quer mapear, e outro para quando se tem dependências múltiplas. No caso de existir apenas uma tarefa dependente utiliza-se o mesmo caminho de procura utilizado em NN, procurando a partir da tarefa que requisitou o mapeamento primeiramente nos vizinhos a um *hop* de distância, depois dois *hops*, e assim progressivamente. No caso de dependências, utiliza-se primeiramente a busca dentro de um quadrado envolvente entre as tarefas dependentes da tarefa solicitada. Caso não seja encontrado nenhum PE livre, procura-se nas primeiras linhas e colunas que circundam o quadrado envolvente e segue-se progressivamente até os limites da rede. Um exemplo deste caminho de procura pode ser visto na Figura 24, onde se deseja mapear uma tarefa que já possui duas tarefas dependentes mapeadas: A e B. Assim, primeiramente são avaliados os PEs que estão dentro do quadrado envolvente entre A e B. Não encontrando PEs livres, o caminho prossegue avaliando os PEs quadriculados que circundam o quadrado envolvente. Por último, caso ainda não sejam encontrados PEs livres, são avaliados os PEs pontilhados.

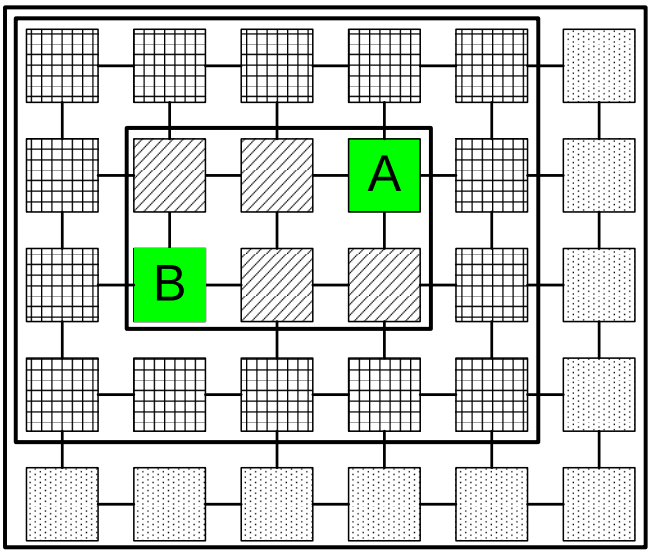
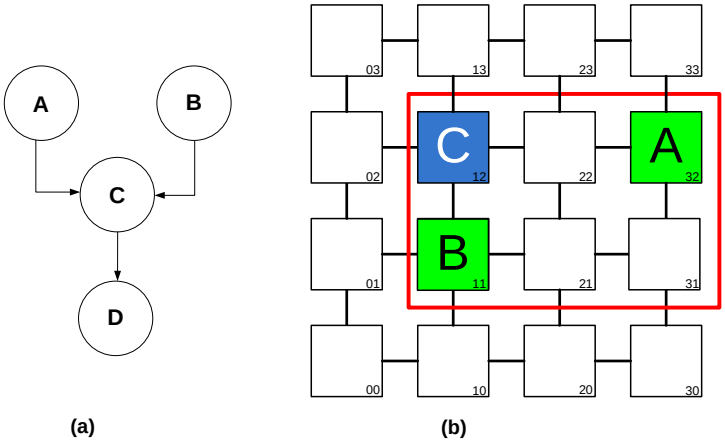


Figura 24 – Caminho de procura da heurística DN para tarefa com mais de uma dependência já mapeada.

Considere a aplicação da Figura 25(a), contendo quatro tarefas, onde A e B são as tarefas iniciais e se comunicam com a tarefa C. Em um dado momento o mapeamento da tarefa C é solicitado pela primeira tarefa que deseja se comunicar com ela, por exemplo, a tarefa A. Neste caso, como a tarefa C possui duas tarefas comunicantes já mapeadas, o espaço de busca por um PE para mapeá-la corresponde ao quadrado envolvente definido pela posição das tarefas A e B (Figura 25(b)). Com isto, buscar-se-á mapear a Tarefa C o mais próximo possível das Tarefas A e B.

Assim, avaliam-se os PEs livres dentro do quadrado envolvente calculando-se o somatório das distâncias em *hops* até A e B, utilizando o roteamento XY. Neste caso, todos estes PEs possuem um somatório de distâncias igual a 3: PEs 22 e 31 estão a um *hop* de distância de A e dois de B; e os PE 21 e 12 estão a um *hop* de B e dois de A. Dessa forma, o primeiro PE encontrado com este somatório é o escolhido, que no caso foi o PE 12. Note que a tarefa D ainda não está mapeada, uma vez que depende da tarefa C.



(a) grafo representando uma aplicação sintética

(b) espaço de busca para mapear a tarefa C, e um possível mapeamento para C

Figura 25 – Exemplo de mapeamento de uma tarefa na heurística DN.

A Figura 26 apresenta o pseudocódigo do algoritmo DN. A heurística tem como entrada uma tarefa t_i a ser mapeada. Cada tarefa t_i possui conjunto Ct_i contendo suas tarefas comunicantes. O algoritmo começa obtendo as tarefas que se comunicam com t_i que já estão mapeadas, colocando-as na *lista_de_dependentes* (linha 2). Esta lista é testada (linha 5 e 20) para ver qual dos caminhos de procura será utilizado no algoritmo: se esta lista contiver apenas uma dependência, o algoritmo age como a heurística NN (linhas 5-19); já, se contiver múltiplas dependências, será procurado primeiramente dentro de um quadrado envolvente um PE para mapear a tarefa t_i (linhas 20-45).

```
Input: Uma tarefa t_i a ser mapeada
Output: O PE melhor_pe para mapear a tarefa t_i
1.  melhor_pe ← -1
2.  // Obtêm todas as tarefas comunicantes de t_i que já estão mapeadas
3.  lista_de_dependentes ← tarefas_mapeadas (Ct_i)
4.  // Se há apenas uma tarefa dependente
5.  SE tamanho(lista_de_dependentes) = 1 ENTÃO
6.      dist ← 1
7.      // Procura enquanto a distância dist ser menor ou igual ao tamanho_da_NoC
8.      ENQUANTO dist<= tamanho_da_NoC FAÇA
9.          p_d ← obtém o PE p_d do primeiro elemento da lista_de_dependentes
10.         lista_de_vizinhos ← vizinhos(dist, p_d) // Obtêm todos vizinhos de p_d com distância dist
11.         PARA TODOS ELEMENTOS p_i NA lista_de_vizinhos
12.             // Verifica ocupação do PE
13.             SE estado(p_i) = livre ENTÃO
14.                 retorna p_i //Termina a execução e retorna p_i
15.             FIM SE
16.         FIM PARA
17.         dist ← dist + 1 // Incrementa dist para se procurar na próxima faixa de vizinhos
18.     FIM ENQUANTO
19. FIM SE
20. SE tamanho(lista_de_dependentes) > 1 ENTÃO
21.     distância_mínima ← ∞ // Inicializa distância_mínima com maior valor
22.     quadrado_envolvente ← área(lista_de_dependentes) // set a as coordenadas do quadrado_envolvente
23.     ENQUANTO quadrado_envolvente<= tamanho_da_NoC e melhor_pe=-1 FAÇA
24.         // Obtêm todos PEs dentro do quadrado_envolvente
25.         lista_de_PEs ← procura_PEs(quadrado_envolvente)
26.         PARA TODOS ELEMENTOS p_i NA lista_de_PEs
27.             // Verifica ocupação do PE
28.             SE estado(p_i) = livre ENTÃO
29.                 distância ← 0
30.                 PARA TODOS d_i NA lista_de_dependentes
31.                     distância ← distância + calcula_distância(d_i,p_i)
32.                 FIM PARA
33.                 // Define a distância mínima e o PE alvo
34.                 SE distância < distância_mínima ENTÃO
35.                     distância_mínima ← distância
36.                     melhor_pe ← p_i
37.                 FIM SE
38.             FIM SE
39.         FIM PARA
40.         SE melhor_pe = -1 ENTÃO
41.             aumenta(quadrado_envolvente)
42.         FIM SE
43.     FIM ENQUANTO
44. FIM SE
45. retorna melhor_pe
```

Figura 26 – Pseudocódigo da heurística DN.

No caso do algoritmo agir como a NN, obtém-se o PE da única tarefa dependente contida na *lista_de_dependentes* (linha 9) e a partir disso começa-se a busca nos vizinhos distantes a um *hop* deste PE e assim progressivamente. No caso de múltiplas dependências, primeiramente obtém-se a área do quadrado envolvente através da

lista_de_dependentes (linha 22). Depois, todos os PEs dentro desta área são armazenados na *lista_de_PEs* (linha 25). Para cada PE p_i nesta lista é calculado o somatório da distância em *hops* até cada uma das tarefas dependentes d_i (linhas 26-39). Entre os PEs livres analisados, é retornado para o mapeamento aquele que obtiver um menor somatório (linha 45). Caso nenhum PE livre seja encontrado neste quadrado envolvente, este é aumentado (linha 41), como explicado anteriormente, e assim progressivamente até se chegar aos limites da rede.

4.3.2 Low Energy Consumption – Dependences Neighborhood (LEC-DN)

A heurística *Low Energy Consumption – Dependences Neighborhood* (LEC-DN) é baseada na heurística DN e tem como principal objetivo a redução na energia de comunicação. Diferentemente da heurística DN, a heurística LEC-DN emprega uma função custo que considera dois critérios: *(i)* proximidade, em número de *hops*, *(ii)* e o volume de comunicação entre tarefas. O segundo critério é usado quando uma determinada tarefa se comunica com pelo menos duas tarefas mapeadas. Nesta situação, a nova tarefa é mapeada o mais próximo da tarefa com maior volume de comunicação.

Esta heurística, assim como a DN, utiliza dois caminhos de procura para selecionar um PE para mapear a tarefa selecionada. No caso de haver apenas uma tarefa dependente, a LEC-DN se comporta como a NN. Já no caso de haver múltiplas dependências, a procura dentro do quadrado envolvente é utilizada selecionando o PE que proporcione um menor consumo de energia dentre os demais. Para isto, é utilizado o modelo de energia proposto por Hu et al. [HU03]. Este modelo calcula a energia consumida na comunicação para transmitir um *bit* por uma distância de n_{hops}, utilizando a Equação 2.

$$E_{bit}^{hops} = n_{hops} * E_{Sbit} + (n_{hops} - 1) * E_{Lbit} \quad (2)$$

onde: E_{Sbit}, E_{Lbit} e n_{hops} representam o consumo de energia em um roteador, nos fios de interconexão e o número de roteadores por onde o *bit* passou. O valor de n_{hops} é obtido a partir do critério de proximidade em número de *hops* apresentado anteriormente. Já o volume de comunicação é utilizado para indicar o número de *bits* transmitidos. Por fim, os valores de E_{Sbit} e E_{Lbit} são obtidos a partir do modelo de energia proposto em [OST10b]. Vale ressaltar que é considerada uma NoC malha homogênea, sendo que a energia consumida para a transferência de um bit por cada roteador da rede é a mesma.

A Figura 27 mostra o pseudocódigo da heurística LEC-DN. Este pseudocódigo é praticamente igual ao da heurística DN, com a principal diferença que a métrica principal utilizada agora é o consumo de energia na comunicação, utilizada no caso da tarefa solicitada ter mais de uma dependência de comunicação já mapeada. Assim, são

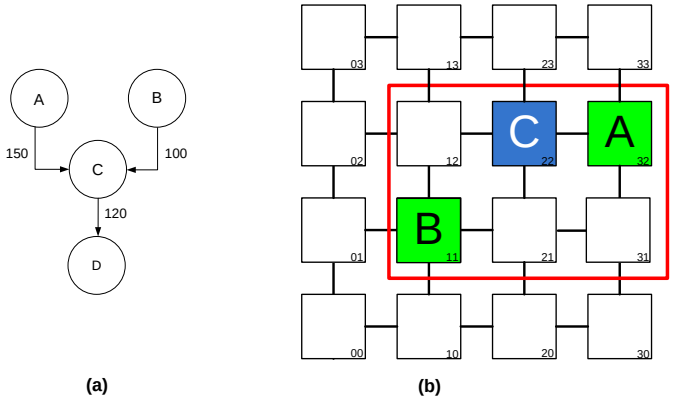
analisados todos os PEs dentro de um quadrado envolvente, como na heurística DN, porém para cada um deles é calculado o somatório de energia consumida na transferência de dados para cada uma das tarefas que se comunicam com a tarefa solicitada. Este somatório pode ser visto na linha 31 do código, onde é acumulada a energia consumida que é calculada simplesmente através da multiplicação da distância até uma tarefa comunicante e o volume de comunicação transferido.

```
Entrada: Uma tarefa t_i a ser mapeada
Saída: O PE melhor_pe para mapear a tarefa t_i
1.  melhor_pe ← -1
2.  // Obtêm todas as tarefas comunicantes de t_i que já estão mapeadas
3.  lista_de_dependentes ← tarefas_mapeadas (Ct_i)
4.  // Se há apenas uma tarefa dependente
5.  SE tamanho(lista_de_dependentes) = 1 ENTÃO
6.      dist ← 1
7.      // Procura enquanto a distância dist ser menor ou igual ao tamanho_da_NoC
8.      ENQUANTO dist<= tamanho_da_NoC FAÇA
9.          p_d ← obtém o PE p_d do primeiro elemento da lista_de_dependentes
10.         lista_de_vizinhos ← vizinhos(dist, p_d) // Obtêm todos vizinhos de p_d com distância dist
11.         PARA TODOS ELEMENTOS p_i NA lista_de_vizinhos
12.             // Verifica ocupação do PE
13.             SE estado(p_i) = livre ENTÃO
14.                 retorna p_i //Termina a execução e retorna p_i
15.             FIM SE
16.         FIM PARA
17.         dist ← dist + 1 // Incrementa dist para se procurar na próxima faixa de vizinhos
18.     FIM ENQUANTO
19. FIM SE
20. SE tamanho(lista_de_dependentes) > 1 ENTÃO
21.     energia_mínima ← ∞ // Inicializa energia_mínima com maior valor
22.     quadrado_envolvente ← área(lista_de_dependentes) // set a as coordenadas do quadrado_envolvente
23.     ENQUANTO quadrado_envolvente<= tamanho_da_NoC e melhor_pe=-1 FAÇA
24.         // Obtêm todos PEs dentro do quadrado_envolvente
25.         lista_de_PEs ← procura_PEs(quadrado_envolvente)
26.         PARA TODOS ELEMENTOS p_i NA lista_de_PEs
27.             // Verifica ocupação do PE
28.             SE estado(p_i) = livre ENTÃO
29.                 energia ← 0
30.                 PARA TODOS d_i NA lista_de_dependentes
31.                     energia ← energia +volume_de_cominicação(d_i,p_i) * calcula_distância(d_i,p_i)
32.                 FIM PARA
33.                 // Define a energia mínima e o PE alvo
34.                 SE energia < energia_mínima ENTÃO
35.                     energia_mínima ← energia
36.                     melhor_pe ← p_i
37.                 FIM SE
38.             FIM SE
39.         FIM PARA
40.         SE melhor_pe = -1 ENTÃO
41.             aumenta(quadrado_envolvente)
42.         FIM SE
43.     FIM ENQUANTO
44. FIM SE
45. retorna melhor_pe
```

Figura 27 – Pseudocódigo da heurística LEC-DN.

Para explicar melhor a heurística, será utilizada a mesma aplicação utilizada no exemplo heurística DN, vista no grafo da Figura 28(a). O grafo desta aplicação possui agora o volume de comunicação transferido entre as tarefas, denotado ao lado de cada aresta. Da mesma forma que o exemplo anterior, esta aplicação possui as tarefas iniciais A e B que se comunicam com a tarefa C. A tarefa C é solicitada, utilizando-se assim a

heurística LEC-DN para escolher um PE para mapeá-la. Como a tarefa C possui duas tarefas comunicantes já mapeadas (A e B), se utiliza como espaço de busca o quadrado envolvente entre estas tarefas. Assim, cada PE deste quadrado será analisado, calculando-se o somatório da energia consumida usada na comunicação até as tarefas A e B. O PE escolhido é o 22 (visto na Figura 28 (b)), pois foi o primeiro PE encontrado que apresenta uma menor energia de comunicação consumida: está a um *hop* de distância de A que envia 150 *flits* a C (1*150 = 150); está a dois *hops* de distância de B que envia 100 *flits* a C (2*100 = 200); totalizando numa energia consumida com peso 350 (150 + 200). O PE 31 também tem um peso de consumo de energia de 350 e poderia ser escolhido. Já os PEs 12, que foi o escolhido pela heurística DN, e 21 apresentam um peso de 400. Pode-se perceber que a tarefa C será mapeada mais próxima da tarefa A, uma vez que de acordo com o grafo de aplicação, o volume de comunicação em A → C é maior que em B → C.



(a) (a) grafo representando uma aplicação

(b) (b) espaço de busca para mapear a tarefa C, e um possível mapeamento para C

Figura 28 - Exemplo de mapeamento de uma tarefa na heurística LEC-DN.

4.4 Mapeamento Multitarefa com Dependências Múltiplas

Nesta Seção são apresentadas as heurísticas de mapeamento dinâmico multitarefa propostas neste trabalho. Primeiramente é apresentada a heurística LEC-DN-MT, uma extensão para multitarefa da heurística LEC-DN apresentada anteriormente. Depois, é apresentada a heurística PREMAP-DN, uma otimização da heurística LEC-DN com a utilização do método de agrupamento PREMAP.

4.4.1 LEC-DN-MT

A heurística LEC-DN foi estendida para multitarefa, sendo chamada de LEC-DN-MT (Low Energy Consumption – Dependeces Neighborhood - Multitask). Para suportar um mapeamento multitarefa, a heurística LEC-DN foi modificada da mesma forma que ocorreu nas versões de NN e BN multitarefa. Dessa forma, a procura por um PE para mapear uma tarefa solicitada agora inclui também a possibilidade do mapeamento desta tarefa no mesmo PE de suas comunicantes. Para isto, a única mudança no pseudocódigo

da heurística LEC-DN, da Figura 27 ocorre na linha 6 onde o valor de *dist* começa em zero. Além disso, o conceito de PE livre (linhas 13 e 28) se estende para considerar a abordagem multitarefa, onde livre é sinônimo de páginas livres para realizar o mapeamento de uma nova tarefa.

4.4.2 PREMAP-DN

A heurística PREMAP-DN (PREMAP - Dependeces Neighborhood) otimiza o mapeamento multitarefa, integrando a heurística LEC-DN-MT com o método de agrupamento PREMAP. Este método tenta suprir a maior restrição de uma abordagem de mapeamento dinâmico de tarefas sem reserva de recursos, que é a ausência de uma visão (conhecimento) global de aplicação. Nesta abordagem de mapeamento, as tarefas são mapeadas quando são solicitadas por uma de suas tarefas comunicantes. Dessa forma, esta abordagem de mapeamento só vem a ter conhecimento de uma tarefa quando é solicitado o seu mapeamento. Em outras palavras, esta abordagem tem como restrição a ordem de solicitação das tarefas para realizar o mapeamento, o que interfere diretamente na qualidade de mapeamento. O método de agrupamento busca otimizar o mapeamento, proporcionando uma visão mais ampla da aplicação. Uma melhor explicação sobre a interferência de uma visão global da aplicação no momento do mapeamento é apresentada na Seção 5.3.3, onde é comparado o algoritmo de mapeamento estático SA que tem visão global da aplicação, com as heurísticas de mapeamento dinâmico monotarefa apresentadas anteriormente.

Esta Seção primeiramente apresenta o método PREMAP que visa agrupar um conjunto de tarefas comunicantes em um mesmo PE. Depois, mostra a integração do PREMAP com a heurística LEC-DN-MT, assim como o novo fluxo de mapeamento utilizado.

4.4.2.1 Método de Agrupamento PREMAP

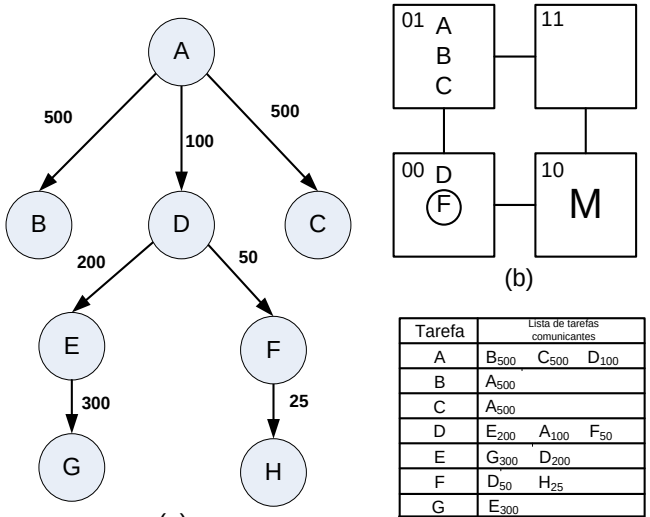
O objetivo do método PREMAP, como dito anteriormente, é agrupar um conjunto de tarefas comunicantes em um mesmo PE. Este método é executado sempre quando uma tarefa nova é mapeada em um novo PE (que não continha tarefas anteriormente). A idéia principal deste método é de reservar um lugar para o mapeamento de uma tarefa, possibilitando que ela esteja o mais próximo possível de suas tarefas comunicantes. Para isto, este método utiliza uma estrutura de dados que será apresentada a seguir, que possibilita uma visão mais ampla da aplicação.

É importante ressaltar que uma tarefa é dita *premapeada*, quando ela possui uma posição de mapeamento reservada dentro de um PE. Esta reserva não indica que este método necessita de uma reserva de recursos para o mapeamento de uma aplicação inteira, visto que esta reserva só é utilizada para priorizar que tarefas comunicantes sejam mapeadas em um mesmo PE.

Para melhor explicar o funcionamento deste método, considere a aplicação da Figura 29(a), com 8 tarefas, sendo t_A a tarefa inicial. Uma instância de MPSoC 2x2 (vista na Figura 29(b)) é utilizada, com o processador mestre localizado no PE 10 e três PEs escravos disponíveis para o mapeamento, cada um capaz de executar até 3 tarefas. A Figura 29(c) apresenta uma tabela contendo, para cada tarefa da aplicação, uma lista de tarefas de comunicantes (LTC), estrutura de dados usada por este método. Assim, cada tarefa t_i possui uma LTC contendo um conjunto $C(t_i) = \{t_1, t_2, ..., t_n\}$, correspondendo a todas as tarefas comunicantes de t_i, ordenadas de acordo com o seu volume de comunicação (maior volume primeiro).

Quando o sistema é iniciado, as tarefas iniciais são mapeadas, de acordo com sua posição definida em tempo de projeto. No exemplo, a tarefa inicial t_A é mapeada no PE 01. A primeira execução do PREMAP ocorre após o mapeamento de tarefas iniciais. Assim, este método analisa cada tarefa t_j do conjunto $C(t_A)$ verificando se t_j não se comunica com nenhuma tarefa com um volume de comunicação maior do que o volume transferido a t_A. No exemplo $C(t_A)=\{t_B,t_C,t_D\}$, $C(t_B)=\{t_A\}$, $C(t_C)=\{t_A\}$ e $C(t_D)=\{t_A,t_E,t_D\}$. Como t_B e t_C são aquelas que se comunicam com um maior volume com t_A, visto que só se comunicam com t_A, elas são *premapeadas* junto a tarefa A no PE 01.

Em um momento posterior, depois do mapeamento da tarefa inicial, t_B e t_C são solicitadas a serem mapeadas. Como elas já estavam *premapeadas*, elas são efetivamente mapeadas no PE 01 (através da transmissão de seus códigos-objeto). Caso não estivessem *premapeadas*, uma heurística de mapeamento seria executada para encontrar um PE para mapeá-las, no caso a heurística LEC-DN-MT.



Tarefa	Lista de tarefas comunicantes
A	B_{500} C_{500} D_{100}
B	A_{500}
C	A_{500}
D	E_{200} A_{100} F_{50}
E	G_{300} D_{200}
F	D_{50} H_{25}
G	E_{300}
H	F_{25}

(c)

(a) grafo de uma aplicação exemplo. (b) possível mapeamento das tarefas da aplicação em um MPSoC 2x2 (c) tabela das tarefas e suas listas de tarefas comunicantes

Figura 29 – Estrutura de dados para a lista de tarefas comunicantes.

Em seguida solicita-se o mapeamento de t_D. Como esta tarefa não foi *premapeada*, a heurística LEC-DN-MT é utilizada e escolhe o PE 00 para mapear t_D (também poderia ser escolhido o PE 11, neste caso). Como t_D "abriu" um novo PE (o PE não possuía tarefas anteriormente), o método PREMAP é executado avaliando-se o grupo $C(t_D)$ para encontrar tarefas a serem *premapeadas* no PE 00. Neste caso $C(t_D)=\{t_E,t_A,t_F\}$, $C(t_A)=\{t_B,t_C,t_D\}$, $C(t_E)=\{t_G,t_D\}$ e $C(t_F)=\{t_D,t_H\}$. Assim, a primeira tarefa avaliada para ser *premapeada* é t_E, por ser a primeira tarefa do conjunto $C(t_D)$. Para isto, é verificado em $C(t_E)$ se há uma tarefa que se comunique com t_E com um maior volume de comunicação do que t_D. Como o volume de comunicação em $t_E \rightarrow t_G$ é superior ao volume de comunicação $t_E \rightarrow t_D$, t_E não é *premapeada* no mesmo PE de t_D. Em seguida t_A deveria ser analisada, mas como já está mapeada, prossegue-se analisando t_F. Como o volume de comunicação de $t_F \rightarrow t_D$ é superior ao de $t_E \rightarrow t_F$, t_F é *premapeada* junto a t_D. Assim, termina-se a execução do PREMAP para o PE 00 com duas tarefas: t_D mapeada e t_F *premapeada*. Ainda sobra uma posição para o mapeamento de uma tarefa no PE 00, que pode ser utilizado pela heurística de mapeamento.

A Figura 30 mostra a implementação do método PREMAP, que busca tarefas comunicantes de t_i a serem *premapeadas* no PE p_i. O método começa atribuindo a $nC(t_i)$ o conjunto de tarefas não mapeadas que se comunicam com t_i, ordenadas do maior ao menor volume de comunicação (linha 2). O próximo passo avalia cada tarefa d_i de $nC(t_i)$, para escolher quais tarefas serão *premapeadas* em p_i. Esta avaliação (linhas 5-16) acontece enquanto p_i possuir um número de tarefas mapeadas/premapedas menor que *Tarefas_por_PE* (o número máximo de tarefas suportadas por um PE), ou se todas as tarefas possíveis em $nC(t_i)$ já foram avaliadas.

```
Entrada: Um PE p_i que foi pela primeira vez utilizado para mapear uma tarefa t_i
Saída: Um conjunto de tarefas comunicantes de t_i a serem premapeadas em p_i
1.  // nC(t_i) contêm todas as tarefas comunicantes de t_i , ordenadas pelo volume de comunicação, que não estão mapeadas
2.  nC(t_i) ← tarefas_não_mapeadas(C(t_i))
3.  // d_i contém a primeira tarefa de nC(t_i)
4.  d_i ← primeira(nC(t_i))
5.  ENQUANTO tarefas(p_i)<Tarefas_por_PE or !fim(nC(t_i)) FAÇA
6.      // Obtém a primeira tarefa h_i (com maior volume de comunicação) do conjunto C(d_i)
7.      h_i ← primeira(C(d_i))
8.      SE h_i= t_i ENTÃO
9.          // premapeia d_i em p_i
10.         premapeia(d_i, p_i)
11.         // aumenta o número de tarefas mapeadas/premapeadas de p_i
12.         tarefas(p_i)++
13.     FIM SE
14.     // Obtém a próxima tarefa no conjunto nC(t_i)
15.     d_i ← próxima(nC(t_i))
16. FIM ENQUANTO
```

Figura 30 – Pseudocódigo do método PREMAP.

Para cada tarefa d_i, a primeira tarefa h_i da sua LTC $C(d_i)$ é obtida (linha 7). Depois, é verificado se h_i é igual a t_i (linha 8) para verificar se d_i se comunica com o maior volume com t_i. Em caso afirmativo, d_i é *premapeada* em p_i, incrementando também o número de tarefas mapeadas ou *premapeadas* em p_i (linhas 9-12). Caso contrário, a próxima tarefa

da LTC $C(d_i)$, se disponível, é avaliada.

É importante mencionar que verificar se uma tarefa comunicante de t_i se comunica com t_i com o maior volume de comunicação é extremamente simples. Basta verificar se t_i é a igual à primeira tarefa da LTC de sua tarefa comunicante.

4.4.2.2 PREMAP-DN

A heurística PREMAP-DN é a combinação do método PREMAP e a heurística LEC-DN-MT através da integração de ambos no fluxo de mapeamento do *microkernel* do Plasma-IP MP, como pode ser visto na Figura 31. Quando uma tarefa t_i é solicitada a ser mapeada por um Plasma-IP SL, uma mensagem de REQUEST_TASK é enviada ao Plasma-IP MP, que ao recebê-la, começa a executar o fluxo de mapeamento. Assim, primeiramente, é verificado se existe algum recurso disponível no sistema. Se não houver recursos disponíveis, a tarefa é escalonada para ser mapeada mais tarde. Caso contrário, o fluxo de mapeamento prossegue para a próxima etapa.

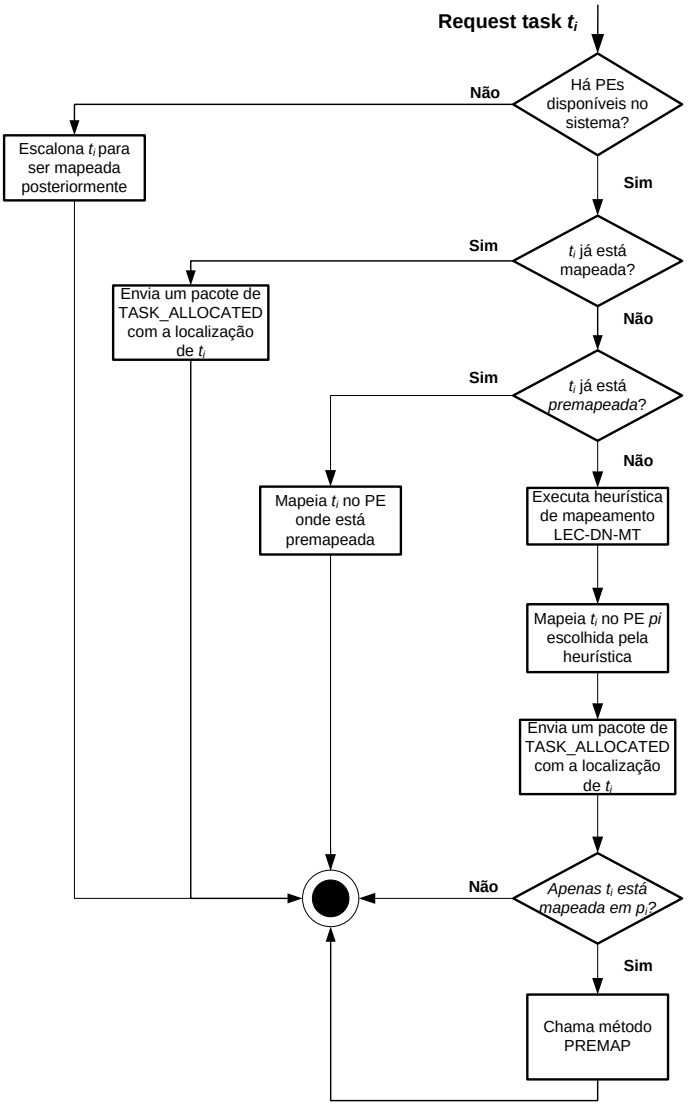


Figura 31 – Integração da heurística PREMAP-DN no fluxo de mapeamento.

O próximo passo verifica se t_i já está mapeada. Caso estiver, envia-se para o PE de quem solicitou t_i um pacote de TASK_ALLOCATED informando a localização de t_i. Caso a tarefa não esteja mapeada, verifica-se se t_i já está *premapeada*. Em caso afirmativo, a tarefa é efetivamente mapeada no PE onde estava *premapeada*, caso contrário, a heurística de mapeamento LEC-DN-MT é executada. Esta heurística retorna o PE p_i onde t_i é mapeada. Depois do mapeamento, mensagens de TASK_ALLOCATED para informar a localização de t_i são enviadas à p_i e ao PE da tarefa que requisitou t_i. Além disso, é verificado se p_i tem apenas uma tarefa mapeada, o que significa que contém apenas a tarefa t_i. Se for verdade, a heurística PREMAP é chamada para avaliar tarefas que se comuniquem com t_i a serem *premapeadas* em p_i e o fluxo é finalizado.

5. AVALIAÇÃO DAS HEURÍSTICAS DE MAPEAMENTO DINÂMICO

Este Capítulo avalia as heurísticas de mapeamento propostas frente a vários cenários de teste e diversas aplicações. As três métricas avaliadas incluem: *(i)* energia consumida na comunicação, *(ii)* somatório da distância em *hops* entre tarefas comunicantes e *(iii)* tempo total de execução das aplicações mapeadas. Estes cenários, bem como as aplicações utilizadas são descritas na Seção 5.1. Depois, a Seção 5.2 avalia a definição da janela de envio de pacotes usada na infraestrutura monitoramento, visto que esta infraestrutura é utilizada nas heurísticas BN monotarefa e multitarefa. Por fim, apresenta-se a avaliação das heurísticas que é dividida entre a avaliação das heurísticas monotarefa, apresentada na Seção 5.3, e das heurísticas multitarefa, apresentada na Seção 5.4.

5.1 Cenários de Teste

Para a realização dos cenários de teste foi utilizado como plataforma o MPSoC HeMPS configurado como segue: NoC malha 2D, algoritmo de roteamento XY, tamanho do *flit* de 16 bits, pacotes com tamanho 128 flits e controle de fluxo baseado em créditos. O MPSoC é dimensionado da seguinte forma: 7x6 (41 Plasma-IPs SL) para a realização de testes de mapeamento monotarefa e 3x5 (14 Plasma-IPs SL) para o mapeamento multitarefa. Considerando-se até 3 tarefas mapeadas por processador, o mapeamento multitarefa pode mapear até 42 tarefas (14 Plasma-IPs SL x 3 tarefas por PE), enquanto o monotarefa pode mapear até 41 tarefas. Dessa forma, as configurações de sistema escolhidas proporcionam praticamente um mesmo número de tarefas executando simultaneamente (41 contra 42 tarefas). Assim, pode-se realizar uma comparação justa entre as abordagens de mapeamento multitarefa e monotarefa.

O modelo de consumo de energia [OST10b] foi calibrado usando a tecnologia CMOS de 65 nm com tensão de alimentação de 1,0 V da ST/IBM, adotando estratégia de *clock-gating*, e uma freqüência de 100 MHz de relógio. Este modelo de energia foi utilizado para a obtenção de valores de energia consumida nos canais e roteadores da NoC. Estes valores foram inseridos na Equação 2, apresentada anteriormente, para o cálculo da energia de comunicação consumida. Além disso, a fim de avaliar a heurística BN, tanto em sua versão mono como multitarefa, a infraestrutura de monitoramento, apresentada na Seção 3.1.4, foi utilizada.

Aplicações reais e sintéticas modeladas em código C, contendo a descrição da comunicação entre as tarefas, são utilizadas. Os códigos destas aplicações foram gerados a partir dos diagramas de sequência e o ambiente Ptolemy II, como foi descrito anteriormente. As aplicações, assim como o seu comportamento de comunicação, são descritos a seguir:

- MPEG-4: é um padrão utilizado para compressão de dados digitais de áudio e vídeo, definido pelo *Moving Picture Experts Group* (MPEG). Neste trabalho, é utilizada uma aplicação que contém 12 tarefas simulando a iteração entre os módulos de *hardware* de um decodificador do padrão MPEG-4 [MIL09]. Esta aplicação contém, como característica relevante, um alto grau de comunicações: duas das 12 tarefas estão ligadas a até 7 outras tarefas. O grafo desta aplicação é ilustrado pela Figura 32. Observar que este grafo apenas ilustra a relação de comunicação entre as tarefas. O código C gerado para a aplicação é obtido a partir do diagrama de sequência, o qual contém a ordem na qual as comunicações ocorrem (ver Figura 22 e Figura 21)

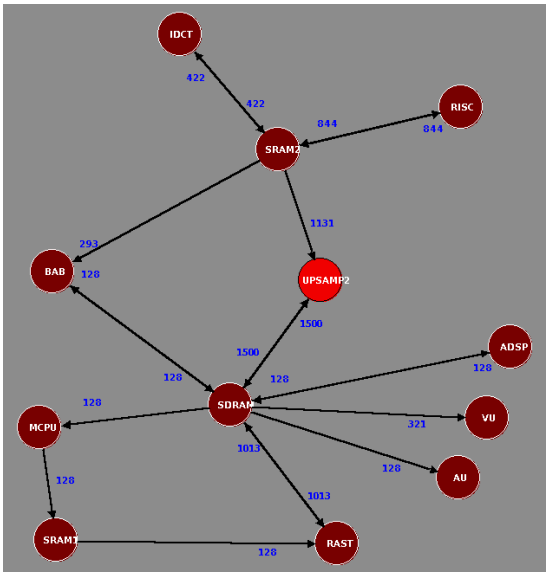


Figura 32 - Grafo da Aplicação MPEG-4.

- VOPD: é uma aplicação que simula a iteração entre os módulos de hardware de um decodificador VOP (do inglês, *Video Object Plane Decoder* ou VOPD) através de 12 tarefas e um formato de fluxo de dados. Esta aplicação tem duas tarefas iniciais: ARM e VLD, que iniciam suas execuções paralelamente. O comportamento desta aplicação pode ser mais bem avaliado através da Figura 33.

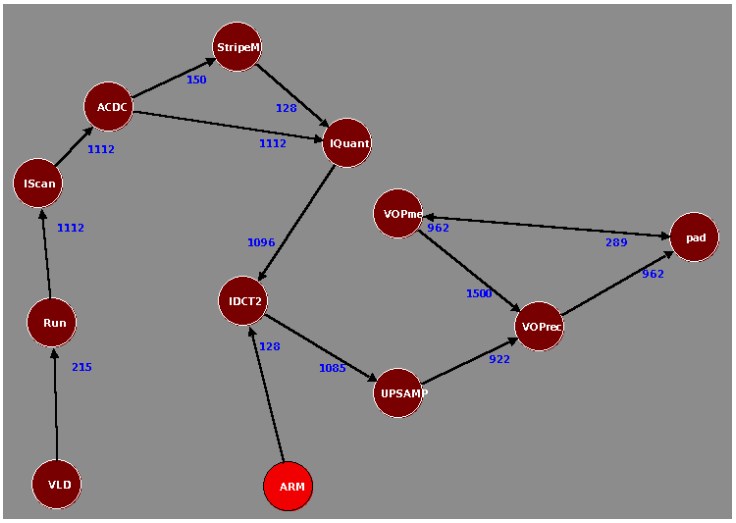


Figura 33 – Grafo da Aplicação VOPD.

- Aplicação Veicular: Aplicação de controle veicular modelada com 10 tarefas [MÄÄ08][MÄÄ10]. Esta tarefa contém um alto nível de dependência de comunicação entre tarefas, sendo também transferido um alto volume de dados entre as tarefas. Além disso, há várias comunicações paralelas. O grafo desta aplicação pode ser visto na Figura 34.

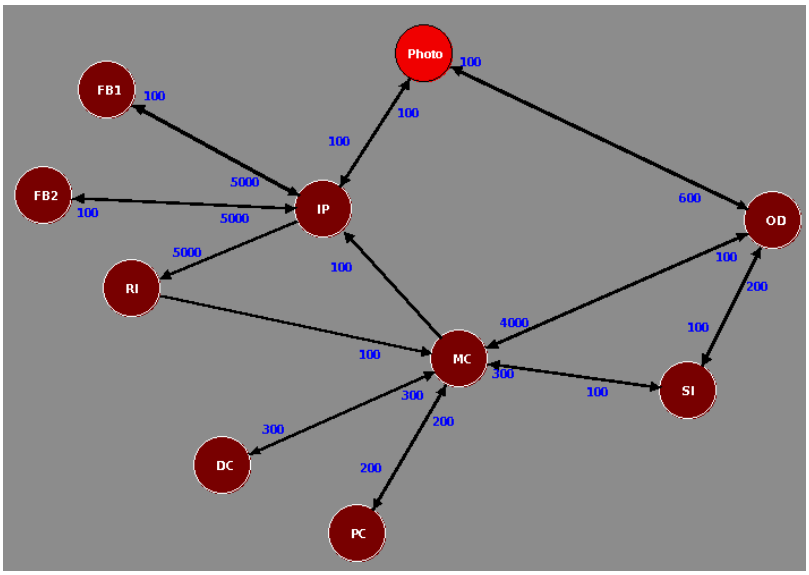


Figura 34 – Grafo da Aplicação Veicular.

- Circuito: Aplicação sintética de 4 tarefas, que apresenta comportamento de fluxo de dados. O grafo desta aplicação é mostrado na Figura 35.

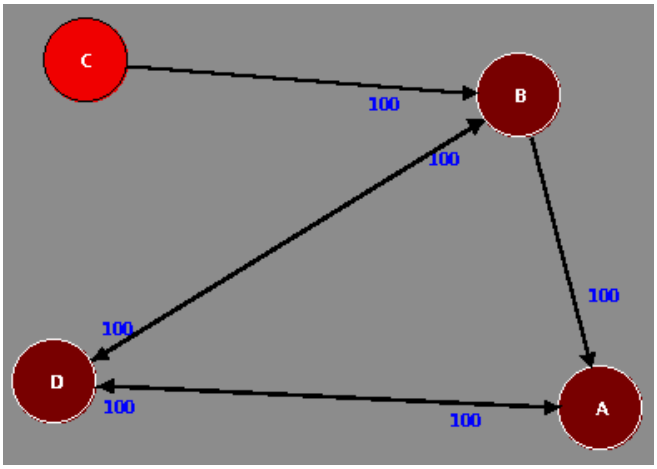


Figura 35 – Grafo da Aplicação Circuito.

- Segmentação de Imagem: Aplicação de segmentação de imagem contendo 6 tarefas. Contém comunicações paralelas. O grafo desta aplicação é ilustrado na Figura 36.

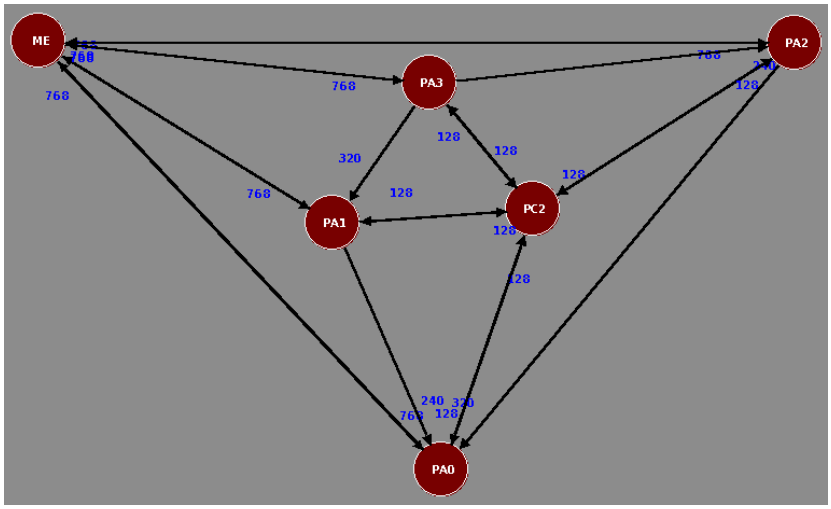


Figura 36 – Grafo da Aplicação de Segmentação de Imagens.

- Hipotética: Aplicação sintética contendo 6 tarefas. A Figura 37 apresenta o grafo desta aplicação.

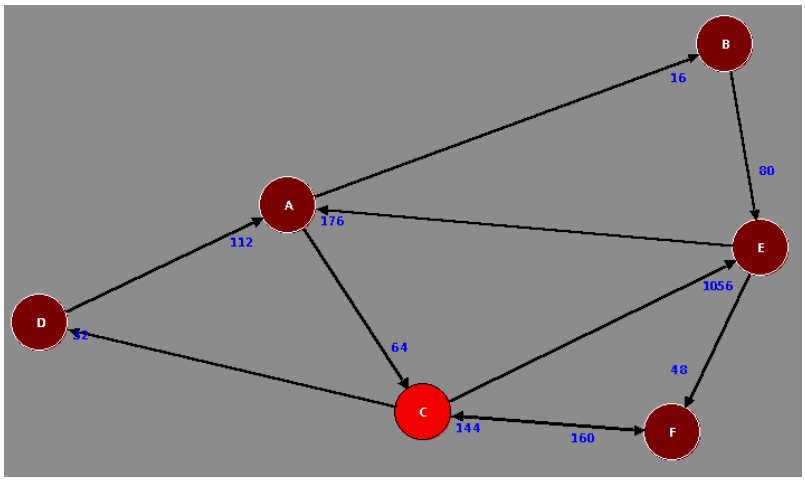


Figura 37 – Grafo da Aplicação Hipotética.

- MWD: aplicação denominada *Multi-Window Display* (ou MWD) possui 12 tarefas. A Figura 38 ilustra o grafo mostrando o comportamento desta aplicação.

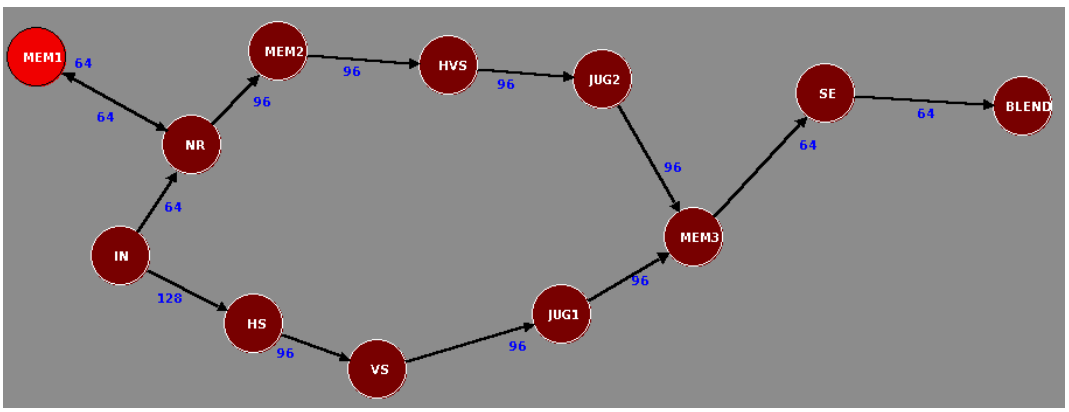


Figura 38 – Grafo da Aplicação MWD.

Os cenários de teste utilizados são mostrados a seguir:

A. MPEG-4, VOPD, Aplicação Veicular e Circuito;

B. MPEG-4, VOPD, Segmentação de Imagem é Hipotética;

C. MPEG-4 e VOPD;

D. MPEG-4, Aplicação Veicular e Circuito;

E. MPEG-4, MWD e VOPD.

Os cenários A, B e E, contêm 38, 36 e 36 tarefas, respectivamente. Estes cenários correspondem a uma ocupação do MPSoC igual a 93% (cenário A) e 86% (cenário B e E). Estes 2 cenários testam as heurísticas de mapeamento dinâmico em um pior caso, uma vez que o espaço de busca é drasticamente reduzido quando quase todas as tarefas já estão mapeadas. Os cenários C e D contêm 24 e 26 tarefas, respectivamente, permitindo avaliar as heurísticas de mapeamento quando o espaço de busca não é restrito.

Nestes cenários as aplicações executam por 10 iterações, sendo que entre cada iteração há um intervalo de aproximadamente 28 mil ciclos de relógio nas tarefas inicias da aplicação (tempo equivalente para o processamento da linha 30 na Figura 21.), simulando, por exemplo, o intervalo entre a decodificação de *frames* por uma aplicação. O objetivo da execução das aplicações por várias iterações é que haja iterações em que a execução da aplicação não sofra interferência dos pacotes de mapeamento que trafegam na rede, permitindo uma melhor avaliação da comunicação das aplicações. O valor de 10 iterações foi escolhido arbitrariamente.

As métricas de desempenho avaliadas são descritas a seguir:

- **Tempo total de execução**: é o tempo até que todas as tarefas das aplicações terminem de executar as 10 iterações definidas nos casos de teste. Como explicado anteriormente, o tempo de computação não está presente nos códigos da aplicação, assim, o tempo total de execução mostra o tempo necessário para que todas as comunicações entre tarefas sejam realizadas.
- **Somatório da distância em *hops* entre tarefas comunicantes**: um menor somatório de número de *hops* indica a utilização de caminhos curtos na comunicação entre tarefas, tornando este critério uma estimativa do congestionamento e ocupação da rede. Caminhos longos de comunicação aumentam a utilização dos canais da rede e a probabilidade de ocorrência de congestionamentos.
- **Energia consumida na comunicação**: energia consumida para a transmissão de dados pela NoC. Para isto, é levada em consideração a energia consumida para transmitir cada *bit* pelos roteadores e canais da rede.

5.2 Definição da Janela de Envio de Pacotes de Monitoramento

Como mencionado anteriormente, a infraestrutura de monitoramento apresentada na Seção 3.1.4 é utilizada nas heurísticas BN monotarefa e multitarefa para prover informações do sistema em tempo de execução que serão utilizadas na escolha do mapeamento. Uma das questões a serem definidas na utilização desta infraestrutura, é a definição da janela de tempo, isto é, o intervalo de tempo em que os pacotes de monitoramento serão enviados ao Plasma-IP MP. Diversos cenários de teste foram utilizados para o dimensionamento desta janela, porém neste trabalho para fins de simplificação, serão mostrados apenas os casos extremos: o pior caso obtido e o melhor. Estes dois casos são suficientes para mostrar que um mau dimensionamento da janela de tempo pode interferir diretamente no desempenho do mapeamento.

Assim, para a avaliação da janela de tempo, foi utilizado como base o cenário C, apresentado anteriormente. Este cenário é executado no MPSoC HeMPS com dimensão 3x5, utilizando a heurística de mapeamento BN multitarefa. A janela de tempo foi variada em dois valores: 10 mil ciclos de relógio e 70 mil ciclos de relógio. As três métricas de desempenho apresentadas anteriormente são utilizadas para avaliação dos dois casos.

A Tabela 4 apresenta os resultados obtidos para esses casos.

Tabela 4 – Resultados da avaliação do dimensionamento da janela de tempo de envio de pacotes de monitoramento

Janela de Tempo:	**10 mil ciclos de relógio**	**70 mil ciclos de relógio**
Tempo Total de Execução (em milhões de ciclos de relógio (100MHz))	16,227	1,950
Somatório da distância em *hops*	15	15
Energia Consumida na Comunicação (em nJ)	555	371

Observa-se pela tabela que a diferença é significativa quanto ao tempo total de execução. A utilização de uma janela de tempo de 10 mil ciclos de relógio apresenta um aumento de 732% em relação à utilização de uma janela de 70 mil ciclos de relógio. Isto deve-se ao fato de que a janela de tempo de 10 mil ciclos produz 7 (70mil/10mil) vezes mais tráfego de pacotes de monitoramento na rede, com maior carga de processamento no MSA. Com isso, pode-se deduzir que o aumento no tempo total de execução é proporcional à redução da janela de tempo, ou seja, como mostrado, diminuindo-se 7 vezes o tamanho da janela, observa-se um aumento de 7 vezes no tempo de execução. Outra explicação para este aumento, deve-se ao fato do envio destes pacotes ser realizado compartilhadamente com o envio de pacotes de dados, através de um multiplexador na porta local de cada roteador da NoC. Assim, os pacotes de monitoramento podem causar um atraso no envio de pacotes de dados, aumentando

ainda mais o tempo total de execução do sistema.

Além disso, o dimensionamento da janela de tempo em 10 mil ciclos de relógio apresentou um aumento de 49,6% de energia consumida na comunicação em relação à janela dimensionada em 70 mil ciclos. Isto pode ser explicado por um mapeamento diferente em ambos os casos, visto que o mapeamento é influenciado diretamente pelo tráfego de pacotes na rede. Isto se deve ao fato do tráfego poder mudar a ordem em que a solicitação de uma tarefa chega ao Plasma-IP MP, devido a canais congestionados, mudando o mapeamento como um todo. Assim, não se pode afirmar que o aumento da janela de tempo tem influência na redução de energia de comunicação para os pacotes de dados, visto que a ordem de solicitação das tarefas que resulta em um melhor consumo de energia pode acontecer em um dimensionamento de janela de tempo desconhecido. Por outro lado, no consumo de energia de comunicação avaliado, não foram levados em conta a energia consumida no envio de pacotes de monitoramento. Assim, pode se afirmar que quanto menor a janela de tempo, maior a energia consumida na transferência de pacotes de monitoramento, visto que há um aumento no volume destes pacotes, como foi mostrado anteriormente.

Desta forma, adota-se uma janela de tempo de 70 mil ciclos de relógio para a avaliação das heurísticas de mapeamento que necessitam de monitoramento da rede. A utilização desta janela propiciou às heurísticas BN (mono e multitarefa), a obtenção de um tempo de execução igual ou melhor que as outras heurísticas sem monitoramento, como será mostrado a seguir.

5.3 Avaliação das Heurísticas de Mapeamento Dinâmico Monotarefa

Esta Seção avalia as heurísticas de mapeamento monototarefa, considerando os 5 cenários apresentados. As heurísticas avaliadas nos cenários de teste são NN, BN, DN e LEC-DN. Além disso, com o objetivo de comparar a abordagem de mapeamento dinâmico com a estática, o algoritmo SA é utilizado. A avaliação dos cenários de teste é dividida por cada métrica de desempenho: tempo total de execução, na Seção 5.3.1; somatório de número de *hops,* na Seção 5.3.2; e energia consumida na comunicação, na Seção 5.3.3. Por fim, são feitas considerações sobre as heurísticas considerando todas as métricas avaliadas, na Seção 5.3.4.

5.3.1 Tempo Total de Execução

A Tabela 5 apresenta os resultados experimentais obtidos na avaliação dos cenários de teste em relação ao tempo total de execução das tarefas, em milhões de ciclos de relógio (100 MHz). O melhor resultado para cada cenário de teste está destacado na tabela.

Tabela 5 – Avaliação dos casos de teste em relação ao tempo total de execução, em milhões de ciclos de relógio (100MHz).

Cenário	Estático	Dinâmico			
	SA	NN	BN	DN	LEC-DN
A	4,981	4,886	4,735	4,623	4,623
B	1,899	2,042	2,104	2,241	2,350
C	1,658	1,671	1,724	1,696	1,700
D	5,049	4,533	4,598	4,565	4,591
E	1,869	1,954	1,694	1,942	1,866

Através da análise dos resultados mostrados na tabela, verifica-se que a heurística LEC-DN é a que obtém um maior tempo total de execução médio. Esta heurística obtém, em relação à SA, NN, BN e DN, um aumento no tempo total de execução médio para todos os casos de aproximadamente 2%, 1,6%, 3,6% e 0,3%, respectivamente. Este aumento no tempo de execução se deve ao maior tempo de computação utilizado pela heurística, que ao contrário das heurísticas NN e BN, considera mais de uma dependência de comunicação de uma tarefa. A heurística DN, que também considera mais de uma dependência de comunicação de uma tarefa no momento de mapeamento obtém resultado similar ao LEC-DN.

Já, a heurística BN é aquela que apresenta uma maior redução no tempo total de execução. Esta redução é de 1,7%, 1,8%, 2,8% e 3,1% comparado à SA, NN, DN e LEC-DN. Esta redução se deve principalmente ao fato desta heurística ter como objetivo principal a redução de congestionamento na rede. Dessa forma, ela procura reduzir a carga na rede e assim, proporciona um menor tempo de comunicação. Lembrando que o tempo total de execução avaliado considera apenas a comunicação entre tarefas.

O algoritmo SA apresenta uma maior variação do tempo total de execução entre todos os casos. Este algoritmo apresenta o menor tempo nos casos B e C, porém também apresenta o maior tempo total de execução em comparação às heurísticas avaliadas nos casos A e D. Como este algoritmo é aplicado em tempo de projeto, todas as aplicações são mapeadas conjuntamente no início da execução do sistema, causando uma sobrecarga no tráfego na rede devido aos pacotes de mapeamento. Este comportamento também seria observado em outros algoritmos realizados em tempo de projeto, como *Taboo Search*. Nos casos A e D soma-se a esta sobrecarga, o grande volume de dados transferidos, principalmente pela aplicação veicular, o que explica o maior tempo de execução nestes casos. Já nos casos B e C, a transferência de dados entre as tarefas é menor, obtendo-se assim uma menor sobrecarga na rede e por consequência um menor tempo de execução.

5.3.2 Somatório da Distância em *hops* entre Tarefas Comunicantes

A Tabela 6 mostra a avaliação dos casos teste em relação ao somatório da distância

em *hops* entre tarefas comunicantes, com destaque ao melhor resultado para cada caso. Pode-se notar por estes resultados destacados que a heurística DN é a que obtém os menores valores para a métrica avaliada. Isto pode ser explicado, tendo em vista que a redução da distância em *hops* entre tarefas comunicantes é o principal objetivo desta heurística. Esta redução é de 15,4%, 9%, 9,6% e 3,8 % respectivamente para SA, NN, BN e LEC-DN.

Tabela 6 - Avaliação dos casos de teste em relação ao somatório da distância em *hops* entre tarefas comunicantes.

Cenário	Estático	Dinâmico			
	SA	NN	BN	DN	LEC-DN
A	115	118	113	112	120
B	108	120	120	110	102
C	56	51	55	46	49
D	115	100	99	86	93
E	88	64	65	59	63

5.3.3 Energia Consumida na Comunicação

A Tabela 7 ilustra os resultados experimentais obtidos quanto à energia consumida na comunicação, para os cenários de teste avaliados. Em destaque estão os resultados com melhor desempenho para cada caso.

Dentre as heurísticas de mapeamento dinâmico, a heurística LEC-DN é que obtém os melhores resultados. Esta heurística obtém uma redução média de energia consumida na comunicação de 9,8%, 9,6% e 4,8% em relação à NN, BN e DN, respectivamente. Estes bons resultados obtidos pela heurística LEC-DN são consequência da consideração do volume de comunicação no momento do mapeamento. As demais heurísticas consideram apenas a proximidade em *hops* entre as tarefas comunicantes.

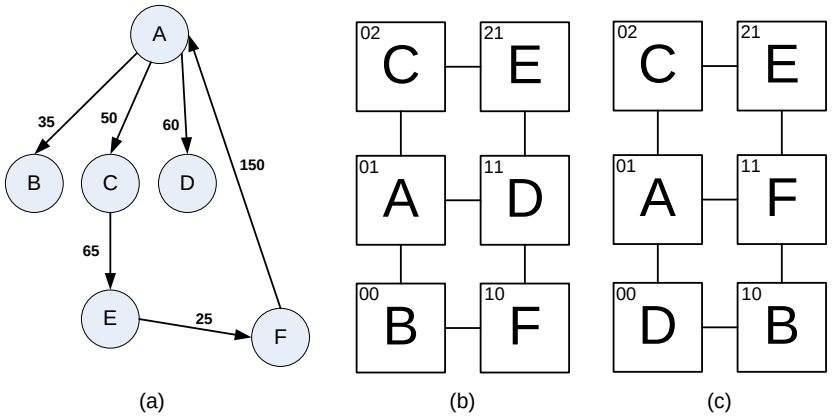
Tabela 7 - Avaliação dos casos de teste em relação à energia consumida na comunicação (em nJ).

Cenário	Estático	Dinâmico			
	SA	NN	BN	DN	LEC-DN
A	2480	3241	3271	3023	2829
B	1869	2061	2217	1999	1929
C	1079	1254	1345	1183	1189
D	2042	2651	2134	2311	2022
E	1143	1236	1288	1252	1228

O custo, em energia de comunicação, do mapeamento dinâmico comparado ao estático é de 19%, 18%, 12% e 6% para as heurísticas NN, BN, DN e LEC-DN, respectivamente. Mostra-se assim que houve uma significativa otimização no mapeamento dinâmico, reduzindo o custo deste frente à abordagem estática para apenas 6%.

O algoritmo SA apresenta o menor consumo de energia devido ao fato de avaliar um número maior de alternativas de mapeamento do que as heurísticas dinâmicas. O SA consegue isto, pois seu tempo de execução não compromete o tempo total de execução do sistema, visto que é executado em tempo de projeto. Além disso, o algoritmo SA realiza o mapeamento de todas as tarefas da aplicação ao mesmo tempo, enquanto as heurísticas dinâmicas realizam o mapeamento de acordo com solicitações que ocorrem a partir do mapeamento das tarefas iniciais de uma aplicação. Dessa forma, as heurísticas dinâmicas têm como restrição a ordem de solicitação das tarefas para realizar o mapeamento, o que interfere diretamente na qualidade de mapeamento. Já, o algoritmo SA não possui esta restrição.

Para melhor exemplificar a diferença entre o algoritmo SA e as heurísticas de mapeamento dinâmico considere o exemplo de mapeamento da Figura 39. No exemplo, supõe-se o mapeamento da aplicação do grafo da Figura 39(a) em um MPSoC de dimensões 2x3, como ilustrado na Figura 39(b). Esta aplicação possui 6 tarefas, sendo A a tarefa inicial mapeada no PE 01. A tarefa A possui 4 tarefas comunicantes: B, C, D e F.



(a) grafo de uma aplicação exemplo (b) possível mapeamento para a aplicação segundo uma das heurísticas de mapeamento dinâmico (c) possível mapeamento para a aplicação segundo o algoritmo estático SA.

Figura 39 – Diferença de mapeamento utilizando heurísticas estáticas e dinâmicas.

No caso de uma das heurísticas de mapeamento dinâmico, o mapeamento das demais tarefas será feito no momento em que forem solicitadas. Assim, supõe-se que primeiramente A solicita o mapeamento da tarefa B, que é mapeada através de uma heurística em um dos PEs que estão a 1 *hop* da tarefa A (PEs mais próximos de A), como por exemplo, no PE 00. Depois são solicitadas em sequência, as tarefas C e D, que são

mapeadas respectivamente nos PEs 02 e 11, ambos a 1 *hop* de distância de A. A tarefa F, também comunicante com A, só será mapeada depois do mapeamento da tarefa E, como pode ser visto no grafo da aplicação. Supõe-se o mapeamento final das tarefas como ilustrado na Figura 39(b). O que se pode notar é que o mapeamento não foi otimizado, pois a tarefa F está mais distante de A, porém esta é a tarefa que se comunica com A com uma maior transferência de dados. Isto ocorre, pois as heurísticas dinâmicas não têm o conhecimento da existência desta tarefa até que ela seja solicitada. Isto não ocorre no algoritmo SA, pois este tem uma visão geral de todas as tarefas a serem mapeadas, conseguindo aproximar da tarefa A, as tarefas com que ela se comunica com maior volume de dados. Assim, na Figura 39(c), é mostrado um possível mapeamento otimizado para a aplicação. Pode-se notar que as tarefas que mais se comunicam com A (F, D e C) estão mais próximas a ela.

5.3.4 Considerações sobre Mapeamento Dinâmico Monotarefa

Através da análise dos resultados apresentados, conclui-se que entre as heurísticas de mapeamento dinâmico a que apresenta um melhor compromisso entre as métricas de desempenho é a LEC-DN. Em relação às demais heurísticas, a LEC-DN obtém resultados expressivos na redução de energia de comunicação, conseguindo reduzir até 23,7% em um dos casos de teste (no caso D, em comparação à NN). Esta vantagem é obtida ao custo de um aumento do tempo total de execução médio de 3,6% comparado às outras heurísticas, o que é considerado aceitável frente ao ganho na redução de energia.

As heurísticas NN e BN, apesar de apresentarem um menor tempo total de execução, não obtêm uma redução tão efetiva de energia de comunicação. No caso da heurística BN, agrava-se o fato devido à utilização da infraestrutura de monitoramento para a coleta de informações para a realização do mapeamento. Além do consumo de energia utilizado para o funcionamento desta infraestrutura, também se deve constar o consumo na transmissão de pacotes de monitoramento pela rede. Pacotes estes que não são levados em conta nos resultados experimentais apresentados. Outro fator negativo no uso do monitoramento é o aumento de área da NoC.

Por fim, o algoritmo SA mesmo gerando os melhores resultados em relação à energia consumida de comunicação, não suporta cenários de carga dinâmica de trabalho, dado seu alto tempo de execução. O uso de um algoritmo SA em um cenário dinâmico, com poucas iterações (para reduzir seu tempo de execução), conduziria a resultados não otimizados.

Vale ressaltar que as heurísticas que apresentam um menor somatório da distância em *hops*, não apresentam obrigatoriamente um menor consumo de energia de comunicação, pois não consideram o volume de comunicação. Isto pode ser observado através dos resultados obtidos pela heurística DN que apresenta uma redução de

respectivamente 15,4% e 3,8% em relação à SA e LEC-DN, porém tem um aumento de energia na comunicação de 12,2% e 5,2% em relação a estas duas técnicas, respectivamente.

5.4 Avaliação das Heurísticas de Mapeamento Dinâmico Multitarefa

Esta Seção apresenta a avaliação das heurísticas de mapeamento multitarefa. Os cenários de teste utilizam um MPSoC com dimensão 3x5, permitindo a execução de até 42 tarefas simultâneas, considerando até 3 tarefas por PE. As heurísticas avaliadas são NN e BN multitarefa, LEC-DN-MT e PREMAP-DN. Além disso, para comparação das abordagens mono e multitarefa é usada também a heurística LEC-DN, para a qual são utilizados os resultados apresentados anteriormente, utilizando um MPSoC de dimensões 7x6. A heurística LEC-DN é utilizada, pois obteve os melhores resultados em redução de energia de comunicação entre as heurísticas monotarefa.

Assim como na avaliação entre as heurísticas monotarefa, esta Seção é dividida pela avaliação de cada métrica de desempenho: tempo total de execução, na Seção 5.4.1; somatório de número de *hops,* na Seção 5.4.2; e energia consumida na comunicação, na Seção 5.4.3. Ao final, na Seção 5.4.4, são feitas considerações sobre as heurísticas avaliadas.

5.4.1 Tempo Total de Execução

A Tabela 8 apresenta os resultados relativos ao tempo total de execução para cada cenário, em milhões de ciclos de relógio (100 MHz), destacando-se o melhor resultado para cada caso. Nestes cenários de teste o tempo total de execução para os cenários B / C / E é dominado pelo tempo de execução de aplicações MPEG e VOPD, enquanto os cenários A / D são dominadas pela aplicação veicular.

Primeiramente é feita uma comparação entre as abordagens mono e multitarefa. Através desta comparação nota-se que a sobrecarga média do tempo total de execução em relação à heurística monotarefa LEC-DN é de 14%, 13%, 16% e 19%, para as heurísticas NN, BN, LEC-DN-MT e PREMAP-DN, respectivamente. Esta sobrecarga no tempo de execução nas heurísticas multitarefa se deve principalmente ao compartilhamento do tempo de execução entre as tarefas executando em um mesmo processador, definido por fatias de tempo no algoritmo de escalonamento utilizado. Assim, durante uma fatia de tempo de execução do processador, uma tarefa pode ficar um longo período em estado de repouso sem executar esperando o recebimento de uma determinada mensagem. Isto acarreta em um desperdício de tempo de execução, pois quando esta tarefa se torna ociosa, uma nova tarefa poderia assumir a execução. Além disso, vale relembrar que as heurísticas multitarefa estão executando em um MPSoC de dimensão menor que a heurística monotarefa (dimensão 3x5 contra 7x6). Dessa forma,

com a utilização de um MPSoC de tamanho menor, o tempo de execução pode aumentar devido a maior probabilidade de existir congestionamentos na rede. Isto acontece, pois o mesmo número de tarefas da avaliação monotarefa agora compartilham um número reduzido de canais da rede neste MPSoC menor. Por estes motivos, pode-se dizer que a sobrecarga no tempo de execução é relativamente baixa comparado à abordagem monotarefa, considerando que a abordagem de mapeamento multitarefa permite com que diversas tarefas executem em um mesmo processador, possibilitando o uso de MPSoCs menores e o aumento da utilização dos processadores.

Tabela 8 – Avaliação dos casos de teste em relação ao tempo total de execução, em milhões de ciclos de relógio (100MHz).

Cenário	Monotarefa	Multitarefa – até 3 tarefas por PE			
	LEC-DN	NN	BN	LEC-DN-MT	PREMAP-DN
A	4,623	5,419	5,329	5,787	5,755
B	2,350	2,436	2,555	2,603	2,483
C	1,700	1,932	1,950	1,912	2,042
D	4,591	5,465	5,430	5,251	5,454
E	1,866	2,168	2,027	2,143	2,312

Agora, comparando somente as heurísticas multitarefa, pode-se perceber que a heurística que apresenta o menor tempo de execução varia de caso a caso. A heurística PREMAP-DN é a que tem o maior tempo de execução entre todos os casos, apresentando um aumento médio de aproximadamente 4%, 4,9% e 2,7% em relação à NN, BN e LEC-DN-MT, respectivamente. Isto pode ser explicado pelo fato do maior tempo de computação utilizado pela heurística PREMAP-DN, que combina a heurística LEC-DN-MT com o método PREMAP. Este aumento é baixo, considerando as vantagens desta heurística que serão apresentadas nas Seções a seguir. Já, a heurística BN é a que obtém o menor tempo total de execução entre as demais. Esta heurística obtém uma redução no tempo de execução em média de 0,6%, 1,9% e 4,3%, em comparação à NN, LEC-DN-MT e PREMAP-DN. Os motivos para esta redução foram explicados anteriormente, na avaliação do tempo total de execução para a abordagem de mapeamento monotarefa.

5.4.2 Somatório da Distância em *hops* entre Tarefas Comunicantes

A Tabela 9 apresenta os resultados obtidos em relação ao somatório da distância em *hops* entre tarefas comunicantes, sendo destacado o melhor resultado em cada caso.

Primeiramente, pode-se ver que a utilização de uma abordagem multitarefa reduz em média 45% o somatório da distância em *hops* em relação à abordagem monotarefa. Na abordagem multitarefa a distância entre tarefas comunicantes pode ser de 0 *hops*, nos casos em que estas tarefas estão alocadas em um mesmo PE. Dessa forma o tráfego de comunicação destas tarefas não passa pela rede, sendo internas aos PEs. Com isto, o

consumo de energia na comunicação é reduzido numa abordagem multitarefa, como será mostrado na Seção a seguir.

Tabela 9 – Avaliação dos casos de teste em relação ao somatório da distância em *hops* entre tarefas comunicantes.

Cenário	Monotarefa	Multitarefa – até 3 tarefas por processador			
	LEC-DN	NN	BN	LEC-DN-MT	PREMAP-DN
A	120	81	57	67	67
B	102	36	44	42	42
C	49	17	15	16	18
D	93	52	35	44	43
E	63	31	30	32	32

Comparando-se as heurísticas multitarefa, quem apresenta os melhores resultados é a BN, sendo melhor em 80% dos casos. Esta heurística consegue uma redução de 11%, 8,6% e 10,3% em relação à NN, LEC-DN-MT e PREMAP-DN, respectivamente. Isto se deve ao fato da heurística BN ter como objetivo principal de otimização a redução do congestionamento na rede, podendo avaliar em tempo de execução a carga nos canais da rede para realizar o mapeamento. Este não é o objetivo principal das heurísticas propostas neste trabalho.

5.4.3 Energia Consumida na Comunicação

A avaliação da energia consumida na comunicação entre tarefas do sistema é o principal objetivo de otimização das heurísticas propostas neste trabalho. Assim, a avaliação desta métrica de desempenho se torna fundamental para o cumprimento dos objetivos do trabalho proposto. Na Tabela 10 são mostrados os resultados obtidos em relação à energia consumida na comunicação para cada cenário de teste, onde os melhores resultados são destacados.

Tabela 10 - Avaliação dos casos de teste em relação à energia consumida na comunicação (em nJ).

Cenário	Monotarefa	Multitarefa – até 3 tarefas por processador			
	LEC-DN	NN	BN	LEC-DN-MT	PREMAP-DN
A	2829	2098	1332	1476	1260
B	1929	618	901	678	755
C	1189	449	371	462	370
D	2022	1007	888	1249	559
E	1228	599	596	617	521

A comparação entre a heurística monotarefa com as multitarefa demonstra uma clara redução de energia na comunicação na abordagem multitarefa. Na Figura 40 podem-se observar os resultados desta redução nas heurísticas multitarefa

comparativamente a heurística LEC-DN. Esta redução chega a uma média de 56% em todos os casos de teste. Isto se deve, principalmente pela redução de tráfego na rede causado pelo mapeamento de várias tarefas em um mesmo PE, como explicado anteriormente.

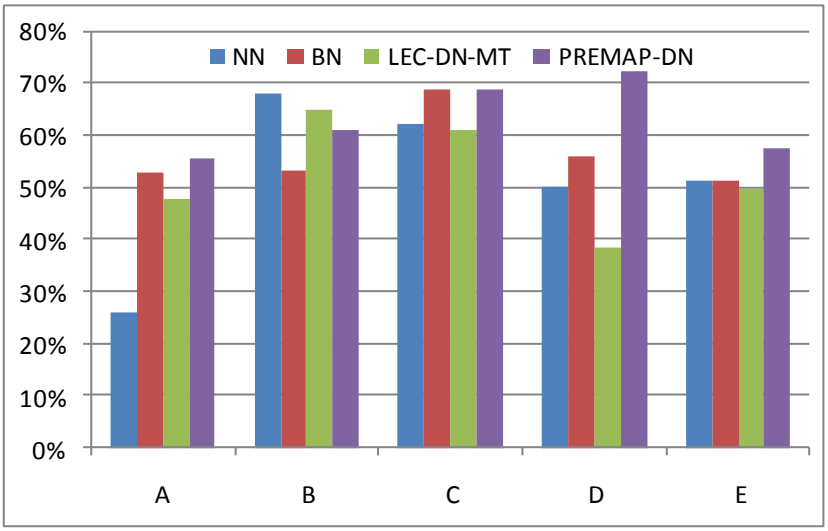


Figura 40 – Redução da energia de comunicação, normalizada pela heurística LEC-DN.

Entre as heurísticas multitarefa, a PREMAP-DN obtém uma redução de energia de comunicação em 80% dos casos. Comparativamente às outras heurísticas, esta redução é de respectivamente 18,6%, 14,3% e 18,8% para NN, BN e LEC-DN-MT. Esta redução é maior se desconsiderarmos o caso B, onde PREMAP-DN foi superada por NN e LEC-DN-MT. Desconsiderando o caso B, a redução atinge 28,8%, 13,8% e 26,3% comparado à NN, BN e LEC-DN-MT respectivamente. A explicação pelo resultado obtido pela heurística PREMAP-DN no caso B, deve-se principalmente ao mapeamento da aplicação VOPD que possui um baixo índice de dependências e um baixo volume de comunicação transferido entre tarefas. Esta aplicação também está presente nos casos A, C e E. Nos casos A e E, a redução de energia de comunicação nas demais aplicações mapeadas compensa um maior consumo obtido pela aplicação VOPD. Já no caso C, como há um espaço de busca menos restrito, o mapeamento da aplicação VOPD consegue um baixo consumo de energia comparando-se aos outros cenários. Assim, pode-se avaliar que a heurística PREMAP-DN apresenta um melhor comportamento na redução de energia para o caso do mapeamento de aplicações que possuem um alto grau de dependências e alto volume de comunicação transferido entre tarefas. Além disso, um espaço de busca maior, ou seja, uma rede com menor taxa de ocupação dos PEs também leva esta heurística à obtenção de resultados melhores. Isto pode ser comprovado em dois cenários de teste: no caso A, que contém a aplicação veicular que possui um volume de dados transferidos maior que as demais aplicações, obtendo-se uma redução de até 40%; e no caso C, que possui um espaço de busca maior, obtendo-se uma redução de até 55%.

Um ponto importante a se ressaltar se refere à heurística LEC-DN-MT, que não

obteve os resultados esperados. Isto se deve a dois fatos principais. O primeiro é devido ao fato das heurísticas NN e BN terem seu comportamento otimizado devido à abordagem de mapeamento multitarefa. Estas heurísticas, na abordagem monotarefa não conseguiam mapear as tarefas de um conjunto de tarefas comunicantes próximas as outras, devido ao seu comportamento de considerar somente o par de tarefas mestre/escravo na escolha do mapeamento. No caso, da abordagem multitarefa, como pode-se alocar mais de uma tarefa por PE e tem-se um tamanho de rede reduzido, as tarefas comunicantes acabam ficando mais próximas, otimizando essas heurísticas (NN e BN). O segundo fato se deve ao comportamento da heurística LEC-DN-MT, que tem tendência a mapear uma tarefa próxima à sua comunicante que transfira o maior volume de dados. Isto pode causar um espalhamento de tarefas nos nodos na rede, podendo muitas vezes ser escolhido um novo nodo ao invés do mapeamento de uma tarefa em um PE que contenha uma tarefa comunicante já mapeada, o que não ocorre nas heurísticas NN e BN. Além disso, a heurística LEC-DN-MT não leva em conta a possível existência de uma tarefa que se comunique com um maior volume de dados com àquela solicitada do que as atuais tarefas já mapeadas, o que é otimizado pela heurística PREMAP-DN.

5.4.4 Considerações sobre Mapeamento Dinâmico Multitarefa

Entre as heurísticas avaliadas, a que apresenta um melhor desempenho em relação a todas as métricas de desempenho é a PREMAP-DN. Esta heurística foi a que obteve um melhor compromisso entre os critérios avaliados, tendo destaque a redução de energia na comunicação que atinge até 55% em comparação com as demais heurísticas. O tempo total de execução é baixo, obtendo um aumento médio de 4,9% em relação à heurística BN, que obteve os melhores resultados. Já em relação ao somatório da distância em *hops*, esta heurística apresenta um aumento de 12,5% e 2%, respectivamente, em relação a BN e LEC-DN-MT. Porém consegue reduzir 3,8% em relação à NN.

A heurística BN, que é a melhor em duas das métricas de desempenho avaliadas (i.e. tempo total de execução e somatório da distância em *hops*) é desconsiderada por dois motivos principais. Primeiro, pois aumentou a energia consumida na comunicação média em aproximadamente 19,7%, comparado às outras heurísticas. O segundo, e mais importante, é o fato da necessidade da heurística BN de utilizar uma infraestrutura de monitoramento para a obtenção dos resultados de mapeamento. Esta infraestrutura de monitoramento, como dito anteriormente, acarreta em área adicional ao circuito integrado e um consumo maior de energia para seu funcionamento. Além disso, para a obtenção de dados do monitoramento, pacotes provenientes de todos os PEs do sistema devem ser transferidos pela rede a cada intervalo de tempo definido pelo projetista. Estes pacotes não são levados em conta nos resultados apresentados, o que pode acarretar em um consumo ainda maior de energia para a transferência destes pacotes.

6. SUPORTE A INSERÇÃO DINÂMICA DE CARGA

Durante o desenvolvimento das heurísticas de mapeamento, verificou-se uma lacuna no MPSoC HeMPS. Esta lacuna se refere à impossibilidade de inserir novas aplicações em tempo de execução neste MPSoC. Todas as aplicações a serem executadas neste MPSoC eram definidas em tempo de projeto. Para isto, a ferramenta *HeMPS Generator* era utilizada, compilando códigos-objeto de tarefas e, depois, gerando o repositório de tarefas. Este repositório possuía um tamanho fixo igual à soma do tamanho do cabeçalho de cada uma das tarefas e de seus respectivos códigos-objeto, impossibilitando a inserção de novas aplicações no sistema.

A inserção de novas aplicações no sistema é uma das grandes vantagens da utilização de uma abordagem de mapeamento dinâmico em relação a uma abordagem estática. O mapeamento dinâmico consegue lidar com a chegada de uma nova tarefa em tempo de execução, determinando um melhor PE para alocá-la. Porém, um ponto crucial no mapeamento de uma aplicação é o mapeamento de suas tarefas iniciais. É a partir da alocação das tarefas iniciais que as demais tarefas da aplicação serão mapeadas no sistema, assim, pode-se dizer que este mapeamento influencia diretamente no desempenho do mapeamento global da aplicação.

Este Capítulo apresenta outra contribuição deste trabalho que é a inserção de novas aplicações em tempo de execução, ou seja, inserção de carga dinâmica no sistema. Isto é realizado durante a simulação do sistema utilizado neste trabalho, possibilitando uma melhor avaliação das heurísticas de mapeamento propostas. Três pontos devem ser considerados para a simulação de inserção de carga dinâmica no sistema: *(i)* geração de novos repositórios de tarefas - Seção 6.1; *(ii)* a criação de um serviço de mapeamento de novas aplicações - Seção 6.2; *(iii)* uma heurística de mapeamento das tarefas iniciais, discutida na Seção 6.3. Por fim, na Seção 6.4, são apresentados experimentos mostrando as influências da inserção dinâmica de carga no mapeamento das tarefas.

6.1 Geração de Repositórios de Tarefas

Como explicado anteriormente, o repositório de tarefas presente no MPSoC HeMPS não propiciava a inserção de novas aplicações no sistema durante o tempo de execução. Assim, foi criado um novo formato para o repositório, prevendo suprir esta lacuna. Além disso, para a inserção de novas aplicações no sistema, foram criados repositórios parciais, contendo uma aplicação a ser inserida no sistema. Dessa forma, esta Seção se divide em duas partes: a criação deste novo formato para o repositório de tarefas, mostrado na Seção 6.1.1; e a geração destes repositórios parciais, vista na Seção 6.1.2

6.1.1 Novo Formato para o Repositório de Tarefas

A nova estrutura para o repositório de tarefas tem por objetivo permitir a inserção de novas aplicações em tempo de execução. Este novo repositório é apresentado na Figura 41 tendo como base o repositório existente anteriormente, que foi apresentado na Figura 11.

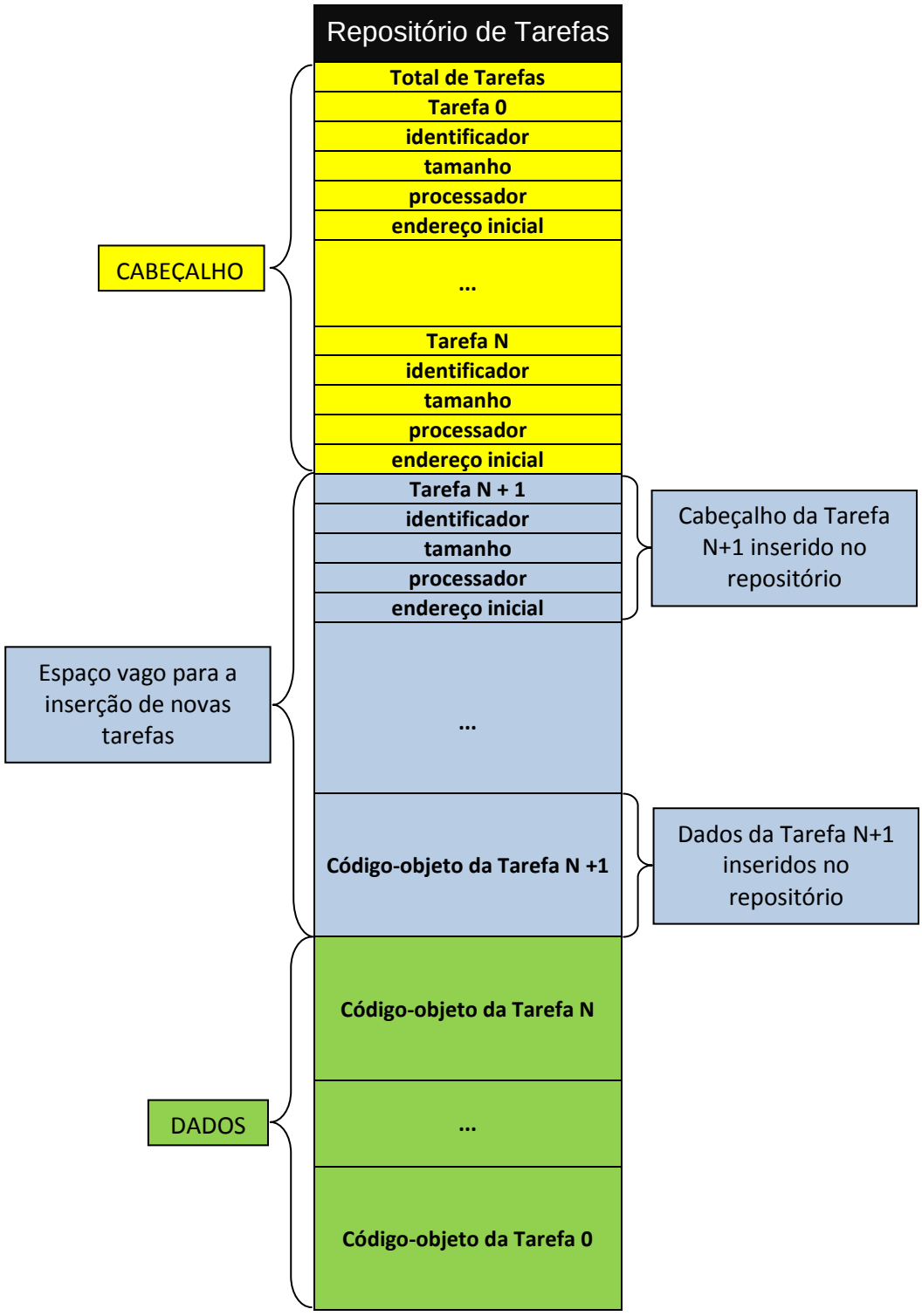


Figura 41 – Novo Formato do repositório de tarefas.

O formato do repositório é praticamente o mesmo do anterior, porém com a inserção de um espaço para que sejam armazenadas novas tarefas (em azul), posicionado entre os cabeçalhos (em amarelo) e os dados das tarefas (em verde). Para a inserção deste espaço, é definido em tempo de projeto, um tamanho máximo para o repositório. Este tamanho é a restrição para a inserção de novas tarefas, visto que posições ocupadas por uma tarefa não podem ser reutilizadas. O armazenamento dos códigos-objeto é feito dos endereços mais altos para os mais baixos, como em uma estrutura de pilha. Assim, quando uma nova tarefa é inserida no repositório, seu código-objeto é armazenado nos endereços anteriores a última tarefa inserida.

6.1.2 Geração de Repositórios Parciais

A inserção de uma nova aplicação no sistema, e por sua vez no repositório de tarefas apresentado anteriormente (Figura 41), requer a transmissão de repositórios parciais à plataforma HeMPS. Os repositórios parciais contêm somente as tarefas da nova aplicação. Este repositório possui o formato ilustrado na Figura 42. O formato utilizado é praticamente igual ao do repositório de tarefas anterior (da Figura 11), apenas com a exclusão no cabeçalho do campo indicativo ao mapeamento das tarefas.

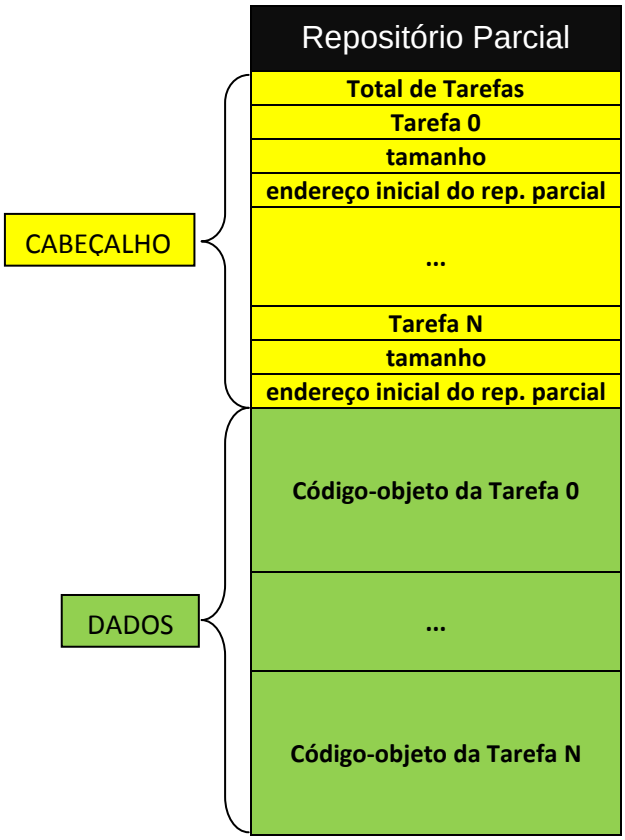


Figura 42 – Formato do repositório parcial.

A geração de um repositório parcial é realizada através de uma ferramenta, onde é definida para qual aplicação quer se gerar este repositório. Esta ferramenta primeiramente

compila os códigos C das tarefas da aplicação gerando seus códigos-objeto, que serão incluídos no repositório. Depois, são gerados os identificadores de cada tarefa. Estes identificadores iniciam sua contagem a partir do número total de tarefas já contidas no repositório de tarefas, no momento quando se deseja inserir a nova aplicação. Isto é realizado para se manter a sequencialidade destes identificadores. Por fim, a ferramenta monta o cabeçalho de cada tarefa, contendo também o endereço inicial do seu código-objeto no repositório parcial.

6.2 Serviço de Mapeamento Dinâmico de Novas Aplicações

A inserção de novas aplicações no sistema em tempo de execução requer primeiramente a inserção das tarefas da nova aplicação no repositório de tarefas. Para isto, o ambiente de simulação transmite ao repositório de tarefas o repositório parcial, contendo as tarefas desta nova aplicação. Esta inserção no repositório de tarefas é feita sem a interferência da HeMPS. Não há conflitos de acesso ao repositório de tarefas, uma vez que a HeMPS utiliza este repositório como uma memória ROM (apenas leitura). Esta inserção se dá em um tempo definido, por enquanto manualmente, pelo projetista no arquivo de *test bench* de simulação do sistema.

Uma vez realizada a inserção do repositório parcial no repositório de tarefas, um sinal é ativado no *test bench* indicando que repositório de tarefas foi alterado. A partir deste sinal é iniciado o processo de mapeamento da nova aplicação no sistema. Para isto, basta o mapeamento das tarefas iniciais desta nova aplicação, que também são definidas, por enquanto manualmente, no *test bench*. Este mapeamento é realizado através de um novo serviço implementado no *microkernel* do Plasma-IP MP.

Para a implementação do novo serviço de mapeamento dinâmico de novas aplicações, foram incluídos dois novos registradores mapeados em memória: REQ_TASK e ACK_TASK, indicando respectivamente a requisição e confirmação do mapeamento de uma nova tarefa (no caso uma tarefa inicial da nova aplicação). O registrador REQ_TASK utiliza o bit 31 para alertar que uma nova tarefa deve ser mapeada, enquanto os bits de 0 a 30 contêm o identificador desta tarefa. Já o registrador ACK_TASK seta todos os bits em ‘1’ quando mapeamento de uma tarefa é confirmado.

Assim, o *test bench,* ao ativar o sinal de alteração do repositório de tarefas, inicia um protocolo de mapeamento de cada uma das tarefas iniciais da nova aplicação. Supondo a inserção de uma nova aplicação no sistema que tem duas tarefas iniciais de identificadores 8 e 9, o protocolo de mapeamento com a utilização dos dois novos registradores é apresentado a seguir:

1. REQ_TASK e ACK_TASK iniciam em zero;
2. Para a requisição do mapeamento da primeira tarefa inicial de identificador 8,

REQ_TASK tem seu valor alterado para “0x8000008”.

3. O *microkernel* do Plasma-IP MP quando verifica que o *bit* 31 do registrador REQ_TASK está em “1”, lê os demais *bits* deste registrador para obter o identificador da tarefa a ser mapeada.
4. É verificado se há PEs disponíveis para mapear esta tarefa. Caso houver, prossegue-se para o próximo passo. Caso contrário, REQ_TASK permanece com o seu valor, requisitando esta tarefa até que um PE se torne livre. Quando isto acontecer, prossegue-se para o próximo passo.
5. É verificado se esta tarefa já foi mapeada. Caso não tenha sido, uma heurística de mapeamento de tarefas iniciais será utilizada para a definição de um PE para mapear esta tarefa. Esta heurística será explicada na Seção a seguir. Definido o PE para mapear a tarefa, são utilizados serviços do *microkernel* já existentes, como TASK_ALLOCATION para realizar o mapeamento. Depois disto, prossegue-se para o próximo passo. Caso esta tarefa já tenha sido mapeada, somente prossegue-se para o próximo passo.
6. O registrador ACK_TASK tem o seu valor alterado para “0xFFFFFFFF”, confirmando que a tarefa já está mapeada.
7. Através da confirmação de ACK_TASK, a requisição de REQ_TASK é finalizada e este registrador é zerado.
8. Ao notar que REQ_TASK está zerado, ACK_TASK também é zerado.
9. Para o mapeamento da segunda tarefa inicial, de identificador 9, REQ_TASK tem seu valor alterado para “0x8000009” e o protocolo dos passos 3 a 8 é novamente realizado.

6.3 Heurística de Mapeamento de Tarefas Iniciais

O mapeamento de uma tarefa inicial, como enfatizado neste trabalho, influencia diretamente no desempenho do mapeamento das demais tarefas de sua aplicação. Uma heurística para realizar automaticamente o mapeamento destas tarefas em tempo de execução é uma lacuna no estado-da-arte de mapeamento de dinâmico de tarefas, não sendo abordada em nenhum dos trabalhos revisados. Assim, este trabalho propõe uma heurística simples para este mapeamento.

A heurística proposta tem por objetivo principal encontrar uma área no sistema onde se tenha o maior número de PEs livres. Para isto, a heurística avalia todos os PEs livres no sistema, contando quantos outros PEs livres estão em seu entorno dentro de uma distância de 2 *hops.* Este valor de distância é parametrizável, podendo ser alterado dependendo do tamanho da rede utilizada.

6.4 Resultados Experimentais

Nesta seção é feita uma avaliação de inserção dinâmica de carga no sistema. Esta avaliação tem por objetivo apenas demonstrar que dependendo da inserção de uma nova aplicação no sistema, muda-se o desempenho obtido pelo mapeamento. Além disso, é mostrada a influência do mapeamento das tarefas iniciais no mapeamento global de sua aplicação.

A inserção de carga dinâmica no sistema, por enquanto, não possibilita a avaliação das heurísticas que consideram dependências múltiplas de tarefas. Isto se deve ao fato das informações relativas às dependências de comunicação e o volume transferido, ainda não estarem armazenadas nos repositórios parciais de uma aplicação. Isto será realizado em trabalhos futuros. Assim, para a avaliação da inserção de carga dinâmica no sistema, será utilizada a heurística NN multitarefa. Esta heurística será avaliada sobre o cenário C, utilizado anteriormente. Este cenário possui 2 aplicações: MPEG-4 (com 12 tarefas) e VOPD (com 12 tarefas). Neste cenário, que tem tempo médio de simulação de 22ms nos casos analisados, é inserida a aplicação MWD (com 12 tarefas) em 3 tempos diferentes da simulação: 500us de simulação, 8ms e 16ms. A inserção desta nova aplicação no cenário C resulta ao final do mapeamento da aplicação MWD no cenário E (MPEG-4, VOPD e MWD). No cenário E as tarefas iniciais das 3 aplicações são mapeadas em tempo de projeto, considerando a técnica de mapeamento de tarefas iniciais apresentada na Seção 4.1.

A Tabela 11 apresenta os resultados obtidos na avaliação da inserção da aplicação MWD no Cenário C nos 3 tempos de simulação previamente definidos, mais os resultados do cenário E. As métricas avaliadas são tempo total de execução em ciclos de relógio (100 MHz), somatório da distância em *hops* entre tarefas comunicantes e energia consumida na comunicação (em nJ). Lembrando que esta avaliação foi realizada utilizando o MPSoC HeMPS com dimensão 3x5.

Tabela 11 - Resultados relativos à inserção dinâmica de carga no sistema

	Cenário E	Cenário C		
Tempo de Inserção da Aplicação MWD	Inserida em tempo de projeto	500us	8ms	16ms
Tempo Total de Execução	2,168	2,184	2,155	2,247
Somatório da distância em *hops*	31	39	28	25
Energia Consumida na Comunicação	599	861	483	477

Através dos resultados apresentados na Tabela 11 nota-se a diferença no desempenho do sistema, dependendo do momento em que a nova aplicação é inserida no sistema. A diferença no tempo total de execução não é expressiva, sendo que a maior variação ocorre nos casos onde a inserção da aplicação é feita aos 8 e 16ms. O caso onde a inserção é realizada aos 16ms apresenta aproximadamente 4% de aumento no tempo de execução em relação ao cenário onde esta inserção é realizada aos 8ms. Quando a inserção é feita aos 16ms, a redução em número de *hops* é de 19,4%, 35,9% e 10,7% em comparação à inserção em tempo de projeto, em 500us e em 8ms, respectivamente. Para melhor explicar o motivo desta redução, considere a Figura 43, que mostra a disposição das tarefas de cada cenário apresentado, no momento de inserção da tarefa inicial IN da aplicação MWD.

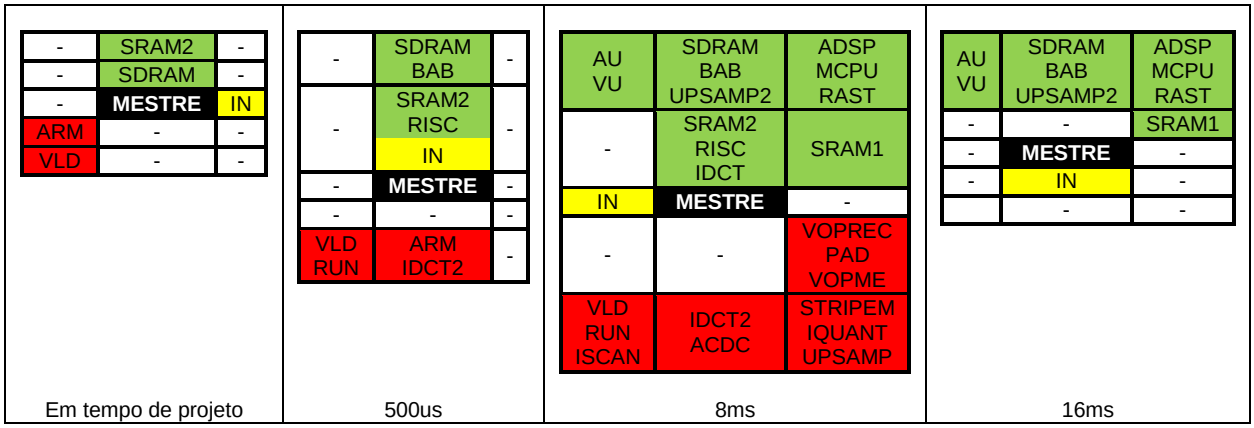


Figura 43 – Disposição das tarefas no momento de inserção da tarefa inicial IN, da aplicação MWD. Em verde, são mostradas as tarefas da aplicação MPEG-4; em vermelho, da VOPD; em amarelo, da MWD; e em preto o processador mestre.

Observa-se pela figura que quando a inserção é realizada aos 16ms, há um menor número de tarefas mapeadas no entorno de onde a tarefa IN foi mapeada (a aplicação VOPD inclusive já terminou sua execução). Isto possibilita maiores alternativas de mapeamento, facilitando a aproximação de tarefas comunicantes da aplicação. Isto também serve de explicação para o fato do cenário de mapeamento da aplicação em tempo de projeto apresentar um aumento do número de *hops* de 24% em comparação ao cenário onde a inserção da aplicação é feita aos 16ms. Outra explicação para isto é que no cenário em que o mapeamento é feito em tempo de projeto, todas as aplicações do sistema estão concorrendo para mapear suas tarefas ao mesmo tempo, o que não ocorre aos 16ms. Observar a inserção da aplicação MWD no tempo 8ms. Neste momento todas as tarefas das aplicações MPEG-4 e VOPD já estão mapeadas e em execução. Assim, o mapeamento da aplicação MWD não disputa o mapeamento com outras tarefas, gerando melhores resultados em todos os parâmetros avaliados.

A energia consumida na comunicação é outra métrica que apresenta grande variação entre os cenários de teste. Na avaliação deste critério, a inserção da aplicação

aos 16ms apresenta o menor consumo de energia, que pode ser explicado pelos mesmos motivos de sua redução no número de *hops.* A inserção da aplicação aos 16ms proporcionou uma redução na energia consumida na comunicação de 20,4%, 44,6% e 1,2% em relação à inserção em tempo de projeto, em 500us e em 8ms.

Agora, para a avaliação da heurística de mapeamento de tarefas iniciais, é feita a comparação do cenário onde a inserção é realizada aos 500us, que utiliza esta heurística, com o cenário onde o mapeamento da aplicação é realizado em tempo de projeto, utilizando o mapeamento apresentado na Seção 4.1 baseado na distribuição das tarefas iniciais em regiões do MPSoC. Uma comparação entre estes dois cenários foi escolhida, pois ambos apresentam condições de mapeamento semelhantes, onde todas as aplicações são inseridas no início da execução do sistema e concorrem para escolha de um PE para mapear suas tarefas. O somatório do número de *hops* e a energia refletem na principal diferença entre estes dois cenários. O mapeamento em tempo de projeto obteve uma redução de 20,5%, no número de *hops*, e 30,4% na energia consumida na comunicação, em relação à inserção aos 500us. Isto mostra a efetividade do mapeamento de tarefas iniciais baseado na distribuição das tarefas iniciais em regiões do MPSoC, comparado com a heurística de mapeamento de tarefas inicias proposta.

Este resultado negativo em 500us deve-se ao fato que as demais aplicações ainda estavam sendo mapeadas, havendo disputa por PEs. Entretanto, uma vez que o sistema já se encontra com suas aplicações mapeadas (em 8 ms) ou terminando a execução de determinadas aplicações (em 16 ms), a heurística de mapeamento de tarefas iniciais proposta é efetiva, resultando no mapeamento da nova aplicação em uma área contígua. Vale ressaltar que não há outra heurística com esta função apresentada na literatura. Apenas o trabalho de Wildermann et al. [WIL09] apresenta uma abordagem de inserção de carga dinâmica. Porém, esta abordagem utiliza a criação ou exclusão de tarefas de uma aplicação já existente no sistema para geração de uma carga dinâmica, não havendo a necessidade da definição do mapeamento de tarefas iniciais..

Um problema relativo à inserção de carga dinâmica no sistema é a fragmentação apresentada no mapeamento, onde tarefas de aplicações diferentes se misturam. Para mostrar este comportamento, foi realizado um teste através da inserção da aplicação VOPD no cenário D, apresentado no Capítulo anterior, aos 500us de execução. Um MPSoC de dimensão 3x5 foi utilizado, realizando-se o mapeamento através da heurística NN multitarefa. A distribuição final das tarefas no sistema pode ser visto através da Figura 44. Pode-se perceber que vários PEs compartilham tarefas de duas aplicações diferentes e que as aplicações não estão mapeadas em bloco. Isto prejudica no desempenho do mapeamento, afastando pares de tarefas comunicantes, podendo aumentar o tráfego no sistema, e consequentemente, a energia consumida na comunicação.

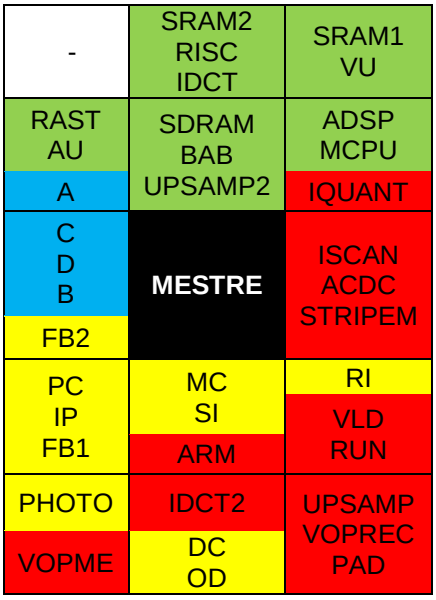


Figura 44 – Exemplo de fragmentação no sistema. Em azul, é representada a aplicação Circuito; em vermelho, a VOPD; em amarelo, a Aplicação Veicular; em verde, a MPEG-4; e em preto, o processador mestre do sistema.

Para resolver este problema, uma das possibilidades é a utilização da técnica de migração de tarefas. Como explicado anteriormente, esta técnica propicia a migração de uma tarefa de um PE para outro em tempo de execução, visando à otimização no sistema. No caso da fragmentação, a migração se faria útil para aproximar tarefas comunicantes, migrando uma tarefa que se comunica com outra de um PE distante para um mais próximo. Uma avaliação sobre a técnica de migração de tarefas será feita em trabalhos futuros, procurando também definir quando e se ela é realmente necessária.

7. CONCLUSÃO E TRABALHOS FUTUROS

A presente Dissertação abordou o tema de mapeamento dinâmico de tarefas para MPSoCs homogêneos. O presente trabalho propôs uma taxonomia para o mapeamento de tarefas, avaliando as lacunas presentes no estado-da-arte. Identificou-se que a minimização da energia consumida na comunicação e o mapeamento multitarefa eram temas inovadores, desenvolvendo-se assim contribuições nestas áreas. Desta forma, as contribuições desta Dissertação podem ser resumidas nos seguintes temas:

- **Geração Automática de Códigos de Aplicações para o MPSoC HeMPS.** Para a avaliação das heurísticas de mapeamento propostas neste trabalho, observou-se a necessidade do uso de aplicações a serem executadas no MPSoC HeMPS para a obtenção de resultados experimentais. Este MPSoC possuía poucas aplicações implementadas, que possuíam uma baixa complexidade e comportamentos de comunicação similares. Dessa forma, uma geração de códigos automáticos de aplicações foi desenvolvida neste trabalho. A geração destes códigos é realizada através do modelo de aplicações proposto por [OST10a], onde estas aplicações são modeladas em diagramas de sequência no ambiente de simulação Ptlolemy II. Assim, foram gerados códigos para um conjunto de 7 aplicações reais ou sintéticas com diferentes comportamentos de iteração entre suas tarefas.
- **Implementação de Heurísticas de Mapeamento de Referência no MPSoC HeMPS:** as heurísticas de mapeamento NN e BN, propostas por Carvalho [CAR10], foram inseridas no *microkernel* do Plasma-IP MP do MPSoC HeMPS. A seguir, a extensão das heurísticas NN e BN para multitarefa, propostas por Singh [SIN09a], foram inseridas. Estas heurísticas foram utilizadas como referência para a avaliação das heurísticas de mapeamento monotarefa e multitarefa propostas neste trabalho. Além disso, a inserção destas novas heurísticas permitiu a avaliação das mesmas em um MPSoC real, com NoC descrita em VHDL RTL e processadores descritos em SystemC com precisão de ciclos de relógio.
- **Integração e Avaliação do Monitoramento do Sistema para utilização no Mapeamento de Tarefas:** Para a implementação da heurística BN, tanto nas versões mono e multitarefa se fez necessária a utilização de uma infraestrutura de monitoramento. Esta infraestrutura propiciou a coleta de informações relativas à carga nos canais da rede, utilizada nesta heurística. Isto permitiu uma melhor avaliação de utilização desta infraestrutura.
- **Implementação de Novas Heurísticas de Mapeamento Dinâmico Monotarefa:** Duas novas heurísticas de mapeamento monotarefa foram propostas neste trabalho: DN e LEC-DN. Estas heurísticas têm como principal objetivo, a redução de energia consumida na comunicação entre tarefas. Para isto, estas heurísticas consideram no

momento do mapeamento todas as dependências de comunicação da tarefa a ser mapeada, além do volume de dados transferido. Com isto, as heurísticas NN e BN foram otimizadas, obtendo-se uma redução de energia consumida na comunicação Esta redução, obtida pela heurística proposta LEC-DN, é de aproximadamente 9,8% e 9,6% em relação à NN e BN, respectivamente.

- **Implementação de Novas Heurísticas de Mapeamento Dinâmico Multitarefa:** A proposição de novas heurísticas de mapeamento dinâmico multitarefa buscou avançar o estado-da-arte nesta abordagem de mapeamento, visto que apenas um Autor ([SIN10]) na literatura aborda tal tema. A heurística PREMAP-DN proposta utiliza uma nova abordagem para a realização do agrupamento (em inglês, *clustering*) de tarefas a serem mapeadas em um único PE. Esta nova abordagem, comparada a trabalhos anteriores na literatura, proporciona uma visão mais ampla do comportamento da aplicação, através da extração de informações de dependência de comunicação entre tarefas e seu volume transferido. Isto mostrou trazer benefícios quanto ao desempenho do mapeamento, principalmente em relação à redução de energia consumida na comunicação. A heurística PREMAP-DN obteve uma redução de energia consumida na comunicação de 18,6% e 14,3%, quando comparado às heurísticas NN e BN multitarefa, respectivamente.
- **Avaliação de Heurísticas de Mapeamento:** Uma avaliação das heurísticas de mapeamento foi realizada neste trabalho comportando três métricas de desempenho: *(i)* tempo total de execução, *(ii)* somatório da distância em *hops* entre tarefas comunicantes e *(iii)* energia consumida na comunicação. O mapeamento dinâmico resulta em um baixo custo de tempo de execução e energia quando comparado ao mapeamento estático, além de permitir a inserção de carga no sistema em tempo de execução. Mostrou-se também que a utilização de um mapeamento dinâmico multitarefa frente a um monotarefa possibilita uma redução no consumo de energia com a utilização de MPSoCs de menor dimensão e com menor tráfego de comunicação pela NoC. Resultados mostram uma redução de energia consumida na comunicação média de 56% entre os cenários de teste avaliados para uma abordagem multitarefa em relação a uma monotarefa.
- **Inserção Dinâmica de Carga no MPSoC HeMPS:** A inserção dinâmica de carga no MPSoC HeMPS proporciona avaliar uma das principais vantagens da abordagem de mapeamento dinâmico frente ao estático. Para isto, foi necessária a alteração do formato do repositório de tarefas do sistema e a criação de repositórios parciais, contendo as tarefas da nova aplicação a ser inserida. Além disso, foi necessária a criação de um serviço de mapeamento das tarefas iniciais das aplicações a serem inseridas, o que tornou necessária a implementação de uma heurística de mapeamento destas tarefas iniciais. Esta heurística é uma contribuição inovadora neste trabalho.

As contribuições apresentadas neste trabalho foram enviadas para conferências pelo meio de dois aprovados. Estes artigos compreendem:

- OST, Luciano; INDRUSIAK, Leandro Soares; MAATTA, Sanna; MANDELLI, Marcelo; NURMI, Jari; MORAES, Fernando Gehm. ***Model-based Design Flow for NoC-based MPSoCs***. In: ICECS, 2010, Atenas. IEEE International Conference on Electronics, Circuits, and Systems, 2010. (Qualis B1)
- MANDELLI, Marcelo; OST, Luciano; CARARA, Everton Alceu; GUINDANI, Guilherme Montez; ROSA, Thiago; MEDEIROS, Guilherme; MORAES, Fernando Gehm. ***Energy-Aware Dynamic Task Mapping for NoC-based MPSoCs***. In: ISCAS, 2011, Rio de Janeiro. IEEE International Symposium on Circuits and Systems, 2011. (Qualis A2)

Diversos trabalhos futuros podem dar continuidade a presente Dissertação. Estes trabalhos futuros incluem:

- *Implementação de um MPSoC com controle distribuído.* Este MPSoC seria dividido em regiões, que possuem um agente controlador que é responsável pelo mapeamento no interior daquela região. A partir desta implementação, desenvolver novas heurísticas para esta arquitetura e avaliá-las. Além disso, proporcionar a avaliação de quando é necessária a utilização de um sistema de controle distribuído frente a um sistema de controle centralizado.
- *Otimização da infraestrutura de monitoramento proposta por [MAR10],* visando a redução no tráfego de pacotes de monitoramento no sistema. Para isto, pode-se utilizar uma infraestrutura de monitoramento distribuído.
- *Integração das heurísticas de mapeamento propostas neste trabalho no ambiente de simulação baseado em modelos proposto por [OST10a]*. Esta integração proporcionará a avaliação das heurísticas em MPSoCs maiores, devido a menores tempos de execução, e com a possibilidade de avaliação de outras métricas de desempenho, como latência de pacotes, detecção de *hot-spots* de tráfego, etc..
- *Consideração do tempo de processamento utilizado por cada tarefa de uma aplicação nas heurísticas de mapeamento.* Isto tornará possível uma melhor distribuição de carga entre os PEs do sistema, evitando sobrecargas de processamento.
- *Proposição de técnicas de migração de tarefas visando uma otimização de desempenho do mapeamento.* Para atingir esta otimização propõem-se técnicas que visem uma melhor distribuição de carga entre os processadores do sistema e uma maior aproximação entre pares de tarefas comunicantes de uma aplicação. Com isto, também busca-se reduzir a fragmentação de tarefas no sistema.
- *Otimização do método de inserção de carga dinâmica no MPSoC HeMPS*, inserindo

informações relativas ao comportamento de uma aplicação no repositório de tarefas do sistema. Entre estas informações citam-se as dependências de comunicação entre tarefas, assim como, o volume de dados transferidos. Estas informações são utilizadas por, entre outras, as heurísticas LEC-DN e PREMAP-DN, propostas neste trabalho. Dessa forma, será possível a avaliação de todas as heurísticas apresentadas neste trabalho em relação à inserção de novas aplicações no sistema em tempo de execução.

REFERÊNCIAS

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

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

[CHO07] Chou, C-L.; Marculescu, R. “Incremental Run-time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels”. In: CODES+ISSS, 2007, pp.161-166.

[CHO08] Chou, C-L.; Marculescu, R. “User-Aware Dynamic Task Allocation in Networks-on-Chip”. In: DATE, 2008, pp. 1232-1237.

[FAR08] Faruque, M. A.; et al. “ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication”. In: DAC, 2008, pp. 760-765.

[HEW77] Hewitt, C. “Viewing control structures as patterns of passing messages”. *Journal of Artificial Intelligence*, vol. 8-3, Jun 1977, pp. 323-363.

[HÖL07] Hölzenspies, P. K. F.; Smit, G. J. M.; Kuper, J. “Mapping streaming applications on a reconfigurable MPSoC platform at run-time”. In: SoC, 2007, pp.1-4.

[HÖL08] Hölzenspies, P. K. F.; Hurink, J. L.; Kuper, J.; Smit, G. J. M. “Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)”. In: DATE, 2008, pp. 212-217.

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

[JAL03] Jantsch, A. “Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation.” San Francisco: Morgan Kaufmann Publishers Inc, 2003, 375p.

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

[LEE03] Lee, E. A.; Neuendorffer, S.; Wirthlin, M. J. “Actor-Oriented Design of Embedded Hardware and Software”. *Journal of Circuits, Systems, and Computers*, vol. 12-3, Jan 2003, pp. 231-260.

[LIN05] Lin, L.; Wang, C.; Huang, P.; Chou, C.; Jou, J. “Communication-driven task binding for multiprocessor with latency insensitive network-on-chip”. In: ASP-DAC, 2005, pp.39-44.

[MÄÄ08] Määttä, S.; Indrusiak, L.S.; Ost, L.; Möller, L.; Nurmi, J.; Glesner, M.; Moraes, F.. “Validation of Executable Application Models Mapped onto Network-on-Chip 96 Platforms”. In: SIES, 2008, pp. 118-125.

[MÄÄ10] Määttä, S.; Möller, L.; Indrusiak, L.; Ost, L.; Glesner, M.; Nurmi, J.; Moraes, F. “Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms*”. International Journal of Embedded and Real-Time Communication Systems*, vol. 1-1, Jan-Mar 2010, pp. 86-101.

[MAR05] Marcon, C.; Palma, J.; Susin, A.; Reis, R.; Calazans, N.; Moraes, F..“Modeling the Traffic Effect for the Application Cores Mapping Problem onto NoC” In: VLSI-SoC, 2005, pp. 391-396.

[MAR06] Martin, G. “Overview of the MPSoC Design Challenge”. In: DAC, 2006, pp. 274-279.

[MAR07] Marcon, C.; Moreno, E.; Calazans, N.; Moraes, F. “Evaluation of Algorithms for Low Energy Mapping onto NoCs”. In: ISCAS, 2007, pp.389-392.

[MAR08] Marcon, C.; Moreno, E.; Calazans, N.; Moraes, F. “Comparison of network-on-chip mapping algorithms targeting low energy consumption”. *IET Computers and Digital Techniques*, vol. 2-6, Nov 2008, pp. 471-482.

[MAR10] Marczak, S. S.; Moraes, F. G. “Implementação de uma infra-estrutura de monitoramento para avaliação de plataformas MPSoC baseada em NoC”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2010, 63p.

[MEH08] Mehran, A.; Khademzadeh, A.; Saeidi, S. “DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip”. *IEICE Electronics Express*, vol. 5-13, 2008, pp. 464-471.

[MIL00] Milojicic, D. S.; Douglis, F.; Paindaveine, Y, Wheeler, R.; Zhou, S. “Process Migration Survey”. *ACM Computing Surveys*, vol 32-3, Set 2000, pp. 241-299.

[MIL09] Milojevic, D.; Montperrus, L.; Verkest, D. “Power Dissipation of the Network-on-Chip in Multi-Processor System-on-Chip Dedicated for Video Coding Applications”. *Journal of Signal Processing Systems*, vol. 57-2, Nov 2009, pp. 139-153.

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

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

[ORS07] Orsila, H.; Kangas, T.; Salminen, E.; Hämäläinen, T.; Hännikäinen, M. “Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips”. *Journal of Systems Architecture*, vol. 53-11, Nov 2007, pp.795-815.

[OST10a] Ost, L. C. “Abstract Models of NoC-based MPSoCs for Design Space Exploration”. Tese de Doutorado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2010, 99p.

[OST10b] Ost, L.; Guindani, G.; Indrusiak, L.; Määttä, S; Moraes, F. “Using Abstract Power Estimation Models for Design Space Exploration in NoC-based MPSoC”. *IEEE Design & Test of Computers*, (Preprint), 2010.

[PLA10] Processador PLASMA. Capturado em: http://plasmacpu.no-ip.org:8080/, Dezembro 2010.

[SCH10] Schranzhofer, A.; Jian-Jia C.; Santinelli, L.; Thiele, L. “Dynamic and adaptive allocation of applications on MPSoC platforms”. In: ASP-DAC, 2010, pp. 885-890.

[SIN09a] Singh, A.K.; Wu Jigang; Prakash, A.; Srikanthan, T. “Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms”. In: Euromicro, 2009, pp. 133-140.

[SIN09b] Singh, A.K. et al. “Eficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms”. In: RSP, 2009, pp. 55-60.

[SIN10] Singh, A. K.; et al. "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms". *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 56-7, Jul 2010, pp. 242-255.

[SMI04a] Smit, L.T.; Smit, G.J.M.; Hurink, J.L.; Broersma, H.; Paulusma, D.; Wolkotte. P.T. “Run-time assignment of tasks to multiple heterogeneous processors”. In: PROGRESS workshop on embedded systems, 2004. pp.185-192.

[SMI04b] Smit, L.T.; Smit, G.J.M.; Hurink, J.L.; Broersma, H.; Paulusma, D.; Wolkotte. P.T. “Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture”. In: FPT, 2004. pp.421-424.

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

[WIL09] Wildermann, S.; et al. “Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures”. In: FPT, 2009, pp. 514 - 517.

[WOL04] Wolf, W. “The Future of Multiprocessors Systems-on-Chip”. In: DAC, 2004, pp. 681-685.

[WOL08] Wolf, W.; Jerraya, A. A.; Martin, G. “Multiprocessor System-on-Chip (MPSoC) Technology”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.27-10, Out 2008, pp. 1701-1713.

[WOZ07] Woszezenki, C. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2007, 121p.

[ZIP09] Zipf, P.; et al. “A Decentralised Task Mapping Approach for Homogeneous Multiprocessor Network-On-Chips,” *International Journal of Reconfigurable Computing*, vol. 2009, 2009, pp. 1-14.