PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Distribuição de Serviços de Gerência em MPSoCs – Mapeamento e Migração de Tarefas

Guilherme Machado de Castilhos

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre

2012

SUMÁRIO

1. INTRODUÇÃO 6

1.1. OBJETIVOS 7
1.2. RESULTADOS ESPERADOS 7
1.3. ESTRUTURA DO DOCUMENTO 7

2. ESTADO-DA-ARTE 8

2.1. CONTROLE DISTRIBUÍDO 8
2.1.1. DISTRM: DISTRIBUTED RESOURCE MANAGEMENT FOR ON-CHIP MANY-CORE SYSTEMS [KOB11] ... 8
2.1.2. DISTRIBUTED RESOURCE MANAGEMENT FOR CONCURRENT EXECUTION OF MULTIMEDIA APPLICATIONS ON MPSOC PLATFORMS [SHA11] 9
2.1.3. EXPLORATION OF MPSOC MONITORING AND MANAGEMENT SYSTEMS [FAT11] 11
2.2. MAPEAMENTO DISTRIBUÍDO 13
2.3. MIGRAÇÃO DE TAREFAS 15
2.3.1. TASK MIGRATION IN MESH NOCS OVER VIRTUAL POINT-TO-POINT CONNECTIONS [GOO11] 15
2.3.2. EVALUATING THE IMPACT OF TASK MIGRATION IN MULTI-PROCESSOR SYSTEM-ON-CHIP [ALM10] 17
2.4. CONSIDERAÇÕES FINAIS 18

3. PLATAFORMA DE REFERÊNCIA – MPSOC HEMPS 20

3.1. ARQUITETURA DA HEMPS 20
3.1.1. PLASMA-IP 20
3.1.2. NOC HERMES 21
3.1.3. REPOSITÓRIO DE TAREFAS 21
3.2. COMUNICAÇÃO ENTRE TAREFAS 21

4. ARQUITETURA DESENVOLVIDA 24

4.1. ARQUITETURA CENTRALIZADA X ARQUITETURA DISTRIBUÍDA 24
4.2. ARQUITETURA DESENVOLVIDA 25
4.3. MAPEAMENTO DISTRIBUIDO 26
4.3.1. INSERÇÃO DE UMA NOVA APLICAÇÃO 26
4.3.2. PROCESSO DE MAPEAMENTO DE TAREFAS 28
4.3.3. MAPEAMENTO CENTRALIZADO VS DISTRIBUÍDO 29

5. RESULTADOS PRELIMINARES 30

5.1. MIGRAÇÃO DE TAREFAS 30
5.1.1. PROTOCOLO DA MIGRAÇÃO DE TAREFAS 30
5.1.2. HEURÍSTICA DE MIGRAÇÃO DE TAREFAS 32
5.1.3. RESULTADOS 32
5.2. MPSOC COM CONTROLE DISTRIBUÍDO 34

6. CRONOGRAMA E ATIVIDADES 38

7. CONCLUSÃO 40

8. REFERÊNCIAS 41

LISTA DE FIGURAS

Figura 1 – *Speedup* médio das aplicações para diversos tamanhos de sistemas. 9
Figura 2 – Diagrama funcional do gerente de recursos centralizado e distribuído. 10
Figura 3 – Gráficos das tarefas e seus mapeamentos no sistema. 11
Figura 4 – Exemplo de inserção de aplicação em tempo de execução. 11
Figura 5 – Hierarquia de gerenciamento do MPSoC. 12
Figura 6 – MPSoC baseado em NoC (a), e possível conteúdo da célula. 12
Figura 7 – Fluxo do método ADAM. 13
Figura 8 – Cenário de teste com apenas um cluster comparando o método proposto por ALF08 com os demais analisados. 14
Figura 9 – Cenário de teste com diversos *clusters* comparando o método proposto por ALF08 com os demais analisados. 15
Figura 10 – Fases do processo de migração proposta por [GOO11]. 16
Figura 11 – Resultados para migração de tarefas proposta por [GOO11] em uma NoC 16x16. 17
Figura 12 – Visão estrutural da plataforma usada em [ALM10]. 17
Figura 13 – Visão geral do protocolo de migração. 18
Figura 14 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09]. 20
Figura 15 – Topologia de uma rede 2x3 usada para exemplificar a troca de mensagens. 22
Figura 16 – Diagrama de sequência de troca de mensagens entre processadores. 23
Figura 17 – Arquitetura com controle (a) centralizado e (b) distribuído 24
Figura 18 – Arquitetura com controle distribuído desenvolvido. 26
Figura 19 – Protocolo para inserção de uma nova aplicação no sistema. 27
Figura 20 – Exemplo de descrição de uma tarefa inicial, com o comando *send*. 28
Figura 21 – Protocolo para mapear uma tarefa 28
Figura 22 – Mapeamento Centralizado (a) vs Mapeamento Distribuído (b). 29
Figura 23 – Exemplo motivacional mostrando benefícios da migração de tarefas. 30
Figura 24 – Protocolo de migração de tarefas. 31
Figura 25 – Pseudocódigo da migração de tarefas. 32
Figura 26 – Vazão da tarefa F, em Mbps. (a) corresponde a aplicação principal executando sozinha no MPSoC; (b) aplicação principal com tráfego perturbador; (c) o mesmo que o 'b' com a tarefa C sendo migrada 34
Figura 27 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark MPEG com três tamanhos de NoC, cenários B, E e H. 36
Figura 28 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark Multispect com três tamanhos de NoC, cenários C, F e I. 37
Figura 29 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark Synthetic com três tamanhos de NoC, cenários A, D e G. 37

LISTA DE TABELAS

Tabela 1 – Características dos cenários avaliados 35
Tabela 2 – Redução do tempo total de execução, adotando o mapeamento centralizado como referência. 35
Tabela 3 – Cronograma de Atividades apresentado no PEP..... 38
Tabela 4 – Cronograma de Atividades para o segundo semestre de 2012. 39

1. INTRODUÇÃO

Com o avanço da tecnologia de fabricação de circuitos integrados, é possível criar transistores cada vez menores, possibilitando o desenvolvimento de sistemas completos em um único chip, denominados Systems-on-Chip (SoCs). Um SoC é um circuito integrado que implementa a maioria ou todas as funções de um sistema eletrônico completo [JER05]. Em geral um SoC pode conter memória, processadores de diversos tipos, lógica especializada, interfaces de comunicação entre outras funções, tanto digitais quanto 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 *Multiprocessor System-on-Chip* (MPSoC). A utilização de MPSoCs é uma tendência no projeto de sistemas embarcados [TAN06]. Um MPSoC consiste de uma arquitetura composta por recursos heterogêneos, que podem incluir múltiplos processadores, módulos de hardware dedicados, memórias e um meio de interconexão [WOL04].

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

Os MPSoCs podem receber uma carga dinâmica de trabalho [CAR07], ou seja, aplicações podem ser carregadas em tempo de execução. Por exemplo, um dado MPSoC pode estar executando o tratamento de um fluxo de dados para processamento de telefonia 4G (LTE) [JAL10], e durante o processamento desta aplicação o usuário inicia o processamento de vídeo em alta-resolução.

Apesar da escalabilidade oferecida por NoCs e processamento distribuído permitindo esta execução de carga dinâmica de trabalho, os recursos dos MPSoCs devem ser gerenciados para proporcionar o desempenho esperado. Tarefas de gerência incluem o acesso a dispositivos de entrada/saída, mapeamento tarefas [MAN11], migração de tarefas [BER06], monitoramento de desempenho, DVFS [PUS09], dentre outras. Um único elemento de processamento (PE), responsável pela gerência dos recursos pode se tornar um gargalo, já que este PE vai servir a todos os PEs do sistema, aumentando sua carga de trabalho e criando regiões com congestionamento de tráfego (hot-spot). Uma alternativa para garantir a escalabilidade é descentralizar ou distribuir as funções de gerenciamento do sistema.

As atuais tendências apontam para projetos de MPSoCs de centenas de PEs. O Intel SCC [HOW10] e a família de processadores Tilera TILE-Gx [TIL11] são exemplos oriundos da indústria. Ambas as arquiteturas possuem um grande conjunto de núcleos que estão ligados através de uma rede intra-chip (NoC). O Intel SCC possui 48 núcleos e o Tilera atualmente possui até 100 núcleos por chip.

Mil núcleos por chip é uma perspectiva tecnológica que deverá torna-se realidade dentro de uma década [BOR07]. Assim, o problema de escalabilidade é real e de significativa relevância. Se novos paradigmas de projeto não forem desenvolvidos, sistemas com múltiplos núcleos iram sofrer de baixa eficiência, uma vez que esses sistemas tendem a usar grande parte de sua comunicação e capacidade de computação com o gerenciamento de seus próprios recursos, ao invés que usar essa capacidade de computação para executar mais eficientemente as aplicações.

1.1. Objetivos

Os objetivos deste trabalho compreendem objetivos estratégicos e específicos.

Objetivos estratégicos:

- Domínio da tecnologia de projeto de sistemas multiprocessados em chip (MPSoC);
- Domínio das técnicas de gerência distribuída em MPSoC.

Objetivos específicos:

- Domínio do *microkernel* do MPSoC HeMPS, permitindo desenvolver técnicas para a gerência distribuída do recursos do MPSoC;
- Avaliação de desempenho do MPSoC desenvolvido. Comparação de desempenho entre MPSoCs com gerência centralizado (HeMPS) e gerência distribuída, visando definir as vantagens e desvantagens de cada arquitetura.

1.2. Resultados Esperados

O presente trabalho de Dissertação terá como contribuição a avaliação de duas técnicas de gerência de recursos em MPSoCs:

- Controle centralizado de recursos;
- Controle distribuído de recursos por região, com utilização da NoC para tráfego de dados e controle;

Especificamente, as atividades de gerência a serem implementadas no escopo deste trabalho incluem mapeamento e migração de tarefas. Estas duas arquiteturas de gerência serão avaliadas para MPSoCs de diferentes dimensões, quanto ao custo de tempo de processamento, potência consumida, área de silício. Como resultado, poder-se-á escolher qual a melhor arquitetura em função das aplicações a serem executadas e da dimensão do MPSoC.

1.3. Estrutura do Documento

Este documento está organizado da seguinte forma. O Capítulo 2 revisa o estado da arte em gerência distribuída em MPSoCs, migração de tarefas e mapeamento distribuído. O Capítulo 3 apresenta a plataforma HeMPS, usada como base desse trabalho. O Capítulo 4 apresenta a arquitetura desenvolvida. O Capítulo 5 apresenta resultados preliminares da arquitetura desenvolvida. O Capítulo 6 apresenta o cronograma de atividades a ser desenvolvidas durante o quarto semestre do mestrado, sendo seguida pelas referências bibliográficas.

2. ESTADO-DA-ARTE

Neste Capítulo é feita uma revisão no estado-da-arte em gerência distribuída em MPSoCs, que será o foco principal do trabalho proposto, seguido por uma revisão sobre mapeamento distribuído e migração de tarefas, que também serão técnicas utilizada no presente trabalho.

2.1. Controle Distribuído

2.1.1. DistRM: Distributed Resource Management for On-Chip Many-Core Systems [KOB11]

Em [KOB11] foi apresentado um esquema de gerente de recursos distribuído em tempo de execução para MPSoCs, chamado DistRM. Ele foi projetado para ser completamente distribuído, sem qualquer sincronização ou comunicação global, tornando o sistema escalável. Foi empregado o principio de multiagentes para gerenciar os recursos, no qual, cada agente gerencia uma aplicação.

Cada agente procura, automaticamente, aumentar a aceleração de suas aplicações procurando outros PEs do sistema que possam ser usados. Com isso, ele usa a ideia de aplicações *maleáveis* [FEI97], para que as aplicações se adaptem aos PEs adicionais. Essas aplicações maleáveis são capazes de adaptar seu grau de paralelismo ao numero de núcleos atribuídos dinamicamente.

Quando uma nova aplicação está prestes entrar no MPSoC, seu agente é iniciado em um PE qualquer do sistema. No entanto, o agente acabará migrando posteriormente para núcleos próximos à aplicação. O PE inicial escolhido, aleatoriamente, atua como uma semente para a busca de recursos.

Como o agente não conhece o estado global do sistema, ele tenta alocar aleatoriamente a aplicação em um recurso disponível do sistema. Para reduzir a distância média de comunicação, escolhem-se regiões próximas do agente para se alocar aleatoriamente. Caso não exista tal região, o agente vai aumentando o seu espectro de busca. Depois que um agente descobriu o conjunto inicial de PEs para a sua aplicação, ele não para de tentar aumentar o desempenho de sua aplicação, através de uma busca constante de novos núcleos para alocar a aplicação.

Para ser capaz de avaliar como o sistema multiagente executa, foi criado um ambiente de simulação completo em nível de sistema que é capaz de simular configurações arbitrárias de um sistema multiprocessado. A métrica usada para comparar o método proposto a um método centralizado foi à carga total de trabalho de todas as aplicações divido pela soma dos tempos de todas as aplicações em cada simulação, denominada *speedup*.

Os resultados da Figura 1 mostram que o método distribuído tem um desempenho melhor comparado ao método centralizado usando o *speedup* como métrica.

A desvantagem desse método, é que ele leva um tempo elevado para estabilizar, devido ao seu mapeamento aleatório, e demanda uma quantidade maior volume de comunicação em comparação aos métodos centralizados.

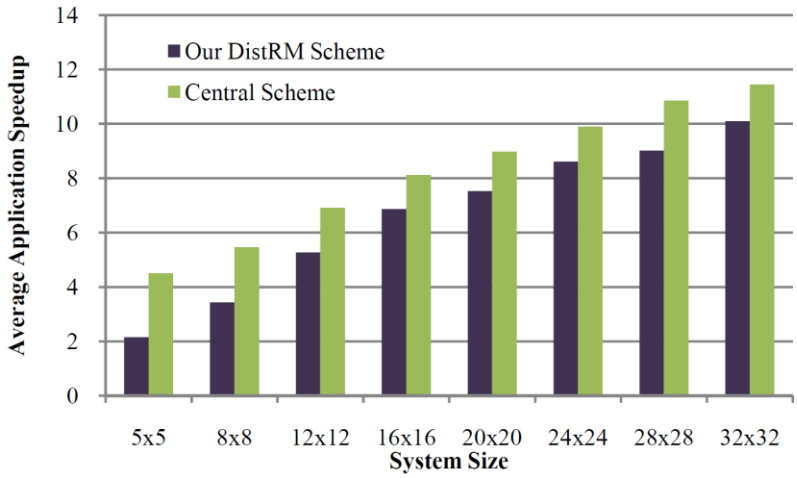

Figura 1 – *Speedup* médio das aplicações para diversos tamanhos de sistemas.

2.1.2. Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms [SHA11]

No trabalho de [SHA11] foram propostas duas versões de gerenciadores de recursos distribuídos em um MPSoC, as quais são escaláveis tanto em número de aplicações quanto em número de processadores, e comparados com um gerenciador de recurso centralizado.

O primeiro tipo de gerenciador de recursos proposto se chama *Credit Based*. Ele pode ser usado para aplicações que tenham restrições rígidas de desempenho, ou seja, seu desempenho não pode ser maior que um determinado valor, mesmo se os recursos estão disponíveis para ter um melhor desempenho.

O segundo tipo de gerenciador de recursos proposto se chama *Rate Based*. Ele é adequado para aplicações que podem ter mais desempenho do que um nível mínimo se os recursos de computação estiverem disponíveis. Por exemplo, codificadores de streaming são uma aplicação para esse tipo de gerenciador, de modo que se houver recursos disponíveis no MPSoC, eles podem codificar a uma taxa maior e terminar o seu trabalho mais rapidamente.

Neste trabalho, os gerenciadores de recursos distribuídos foram avaliados com base em sua capacidade de satisfazer as restrições de transferências de múltiplas aplicações. Foram realizados também experimentos, adicionando aplicações em tempo de execução e analisando seu comportamento. As métricas de avaliação utilizadas neste trabalho incluem perdas de *deadlines*, necessidade de *buffers* e *jitter* máximo.

As aplicações usadas foram especificadas como *Synchronous Data Flow (SDFGs)* [LEE87], onde os vértices indicam tarefas separadas (também chamados de atores) de uma aplicação, e as arestas denotam as dependências entre eles. *SDF* é amplamente utilizado, e é muito adequado para expressar a simultaneidade em aplicações, portanto, útil para analisar sistemas multiprocessados.

O arquitetura do modelo é composta de processadores conectados por uma NoC. Os gerentes de recursos consistem em um controlador de admissão. O papel do controlador de admissão é avaliar as restrições de tempo de uma nova aplicação contra os recursos disponíveis no MPSoC.

O controlador de admissão é uma interface com o usuário que calcula os créditos que serão distribuídos para os processadores. Os árbitros desses processadores aplicam localmente esses créditos, tal que as restrições de vazão da aplicação sejam satisfeita.

As seguintes informações são requeridas pelo controlador de admissão:

- modelo *SDF* de cada aplicação;
- estimativa de pior caso de tempo de execução para cada aplicação (em ciclos de relógio);
- desempenho desejado de cada aplicação, por exemplo, frames/seg;
- mapeamento de tarefas na plataforma (migração de tarefas não é suportada);
- previsão de desempenho de cada aplicação isoladamente com o mapeamento dado.

A Figura 2 mostra o diagrama funcional do gerente de recursos centralizado e do gerente de recursos distribuído proposto. O gerente centralizado monitora o a vazão de cada aplicação e compara com sua vazão desejada. Já o gerente distribuído procura minimizar o seu envolvimento no processo de monitoramento e investe em mais inteligência nos árbitros dos processadores locais, como mostra a Figura 2(b). A restrições de cada aplicação são calculados centralmente, mas eles são aplicados em todos os processadores localmente. O gerente distribuído não monitora cada aplicação, então o problema de escalabilidade devido ao período de monitoramento é eliminado.

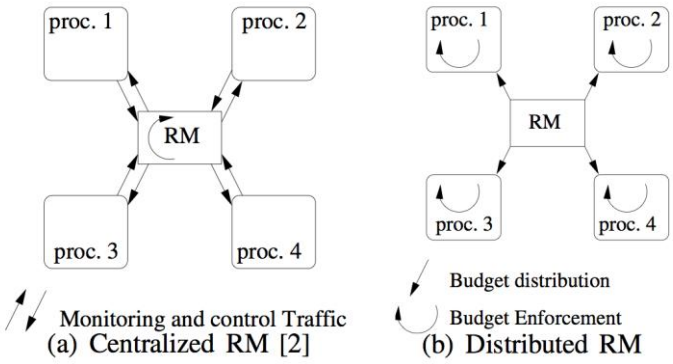

Figura 2 – Diagrama funcional do gerente de recursos centralizado e distribuído.

Os modelos dos gerentes de recursos distribuídos usados para a simulação foram desenvolvidos usando a linguagem POOSL [FEI97]. Foram usados dois modelos de aplicações para as simulações: decodificador JPEG (6 atores) e decodificador H.263 (4 atores), mostrados na Figura 3. A plataforma de computação é composto por 6 processadores, e a Figura 3 também mostra o mapeamento dos atores nos processadores.

Como cenário de teste, foi inserido no MPSoC uma aplicação H.263 com restrição de vazão de 20 frames/seg, no tempo de 700 milhões de ciclos de relógio. Na plataforma já estão executando previamente uma aplicação decodificador JPEG e outra aplicação decodificador H.263. A Figura 4(a) mostra o comportamento do gerenciador de recursos centralizado, no qual as aplicações que já estavam sendo executadas não sofrem qualquer impacto em seus desempenhos com a inserção da nova aplicação. O alto *jitter* na execução da aplicação é devido ao fato que as aplicações ficam ativas/desativas durante o período de monitoramento.

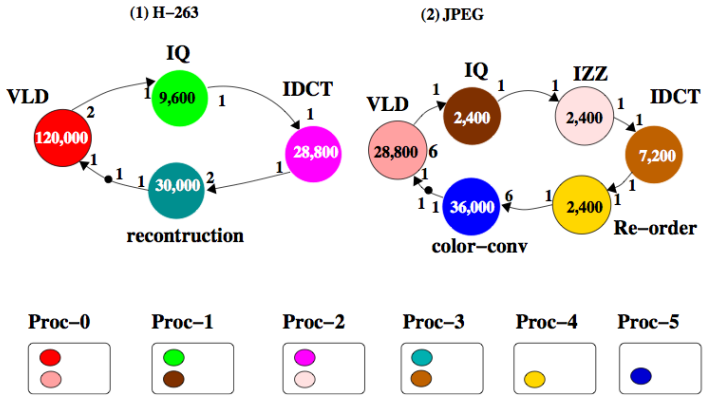

Figura 3 – Gráficos das tarefas e seus mapeamentos no sistema.

A Figura 4(b) mostra o comportamento do gerenciador de recursos *Credit Based*. Nesse cenário, as aplicações que já estão em execução na plataforma não sofrem qualquer efeito no seu desempenho. Além disso, o jitter na execução da aplicação é muito pequeno em comparação com o gerenciador centralizado. A Figura 4(c) mostra a resposta do gerenciador *Rate Based*, o desempenho dos decodificadores JPEG e H.263 diminuem imediatamente, logo que a segunda instancia do decodificador H.263 entra no MPSoC. Isto é devido ao fato de que quando a segunda instancia do H.263 entra, ele tem o mais baixo nível de desempenho a ser alcançado, o que faz ganhar a preferência e rapidamente consegue o nível de desempenho exigido. Com o tempo, as outras aplicações também obtêm os recursos de computação e o sistema converge rapidamente para o novo estado estacionário.

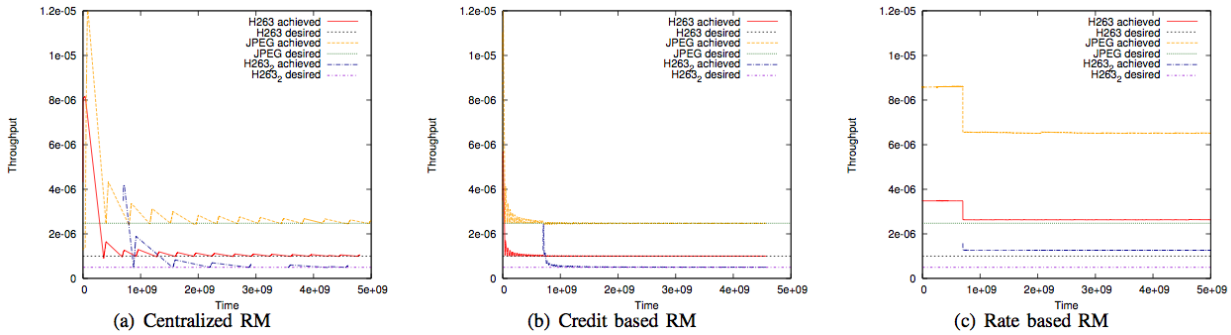

Figura 4 – Exemplo de inserção de aplicação em tempo de execução.

Os experimentos mostram que o Credit Based é muito eficaz para fazer cumprir as restrições de vazão, e o Rate Based é eficaz para a obtenção de uma alta utilização de recursos no contexto de aplicações que podem se beneficiar com a disponibilidade de recursos. Ambos os gerenciadores distribuídos podem lidar com um número maior de processadores, bem como um grande número de aplicações executando simultaneamente em comparação com o gerenciamento centralizado.

Os problemas dessa proposta, é que ela limitou-se a realizar experimentos com um numero reduzido de processadores (6 processadores), implementação alto nível da plataforma, além de não suportar mapeamento dinâmico e migração de tarefas.

2.1.3. Exploration of MPSoC Monitoring and Management Systems [FAT11]

Em [FAT11] é apresentada uma abordagem para combinar métodos centralizados e distribuídos de gerência de um MPSoC e construir um sistema de gerenciamento hierárquico, a fim de beneficiar os

esquemas distribuídos e centralizados, como mostrado na Figura 5. Isto é, a gerência do sistema é de granularidade fina em alguns aspectos, embora alguns dados monitorados e relatórios de status são enviados para o gerente de nível superior que envia de volta comandos a serem executados.

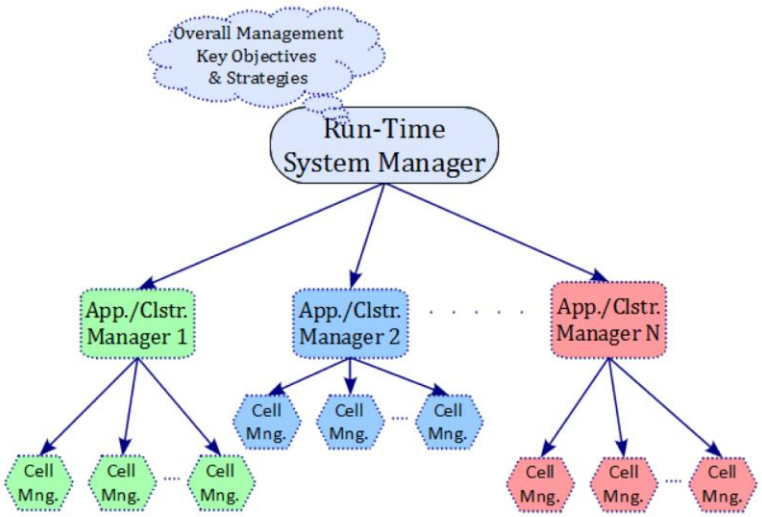

Figura 5 – Hierarquia de gerenciamento do MPSoC.

O roteador e os componentes locais ligados a ele são chamados de célula, como é mostrado na Figura 6, sendo a célula o menor componente de um sistema hierárquico. De acordo com a demanda do sistema, cada célula pode conter diferentes tipos de monitores, por exemplo, sensores de temperatura, monitores de energia, detectores de falhas, etc, juntamente com atuadores, tais como reguladores de tensão e frequência. Cada célula tem um próprio gerente, no qual reporta sua condição e obedece comandos.

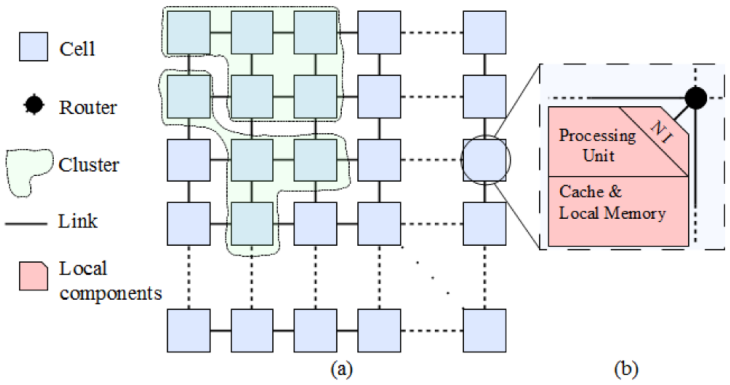

Figura 6 – MPSoC baseado em NoC (a), e possível conteúdo da célula.

Um grupo de células é agrupado para formar um *cluster*, e gerenciado por um gerente de nível médio. Diferentes políticas de agrupamento podem ser empregadas, mas a utilizada no trabalho foi que cada *cluster* executa tarefas de uma determinada aplicação. Esse gerente é chamado de gerente de aplicação. Cada gerente de aplicação é responsável pelas demandas de sua aplicação, isto é, ele gerencia os gerentes das células a fim de atender aos requisitos da aplicação.

O gerente de alto nível, denominado de gerente global, é responsável pelo controle geral do sistema, e coordena as ações dos gerentes de níveis inferiores. O gerente global é executado em nível de sistema operacional e está encarregado de criar novos gerentes no caso de novas alocações de aplicações.

Vantagens de sistemas de gestão centralizados e distribuídos são obtidas através da exploração de gestores inteligentes de células, e adicionando um gerente de nível médio para minimizar os dados monitorados que chegam aos gestores de nível superior.

Segundo autores, tal sistema tem como penalidade a ineficiência dos agentes de software para sistemas com menos de 100 núcleos.

Não foram apresentados resultados de implementação do sistema de gerenciamento hierárquico, apenas uma possível implementação como trabalhos futuros tendo como base a plataforma HeMPS.

2.2. Mapeamento Distribuído

[ALF08] propõe um método para um mapeamento de aplicações em tempo de execução de forma distribuída usando agentes, para MPSoC heterogêneos baseados em NoC. O método criado se chama ADAM (*Run-time Agent-based Distributed Application Mapping for on-chip Communication*), e uma visão geral sua é apresentado na Figura 7.

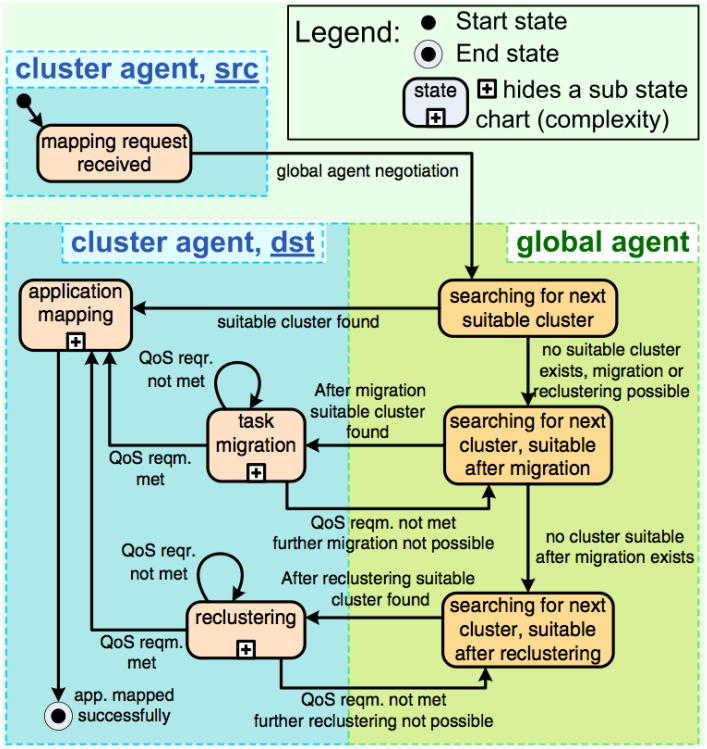

Figura 7 – Fluxo do método ADAM.

O mapeamento em tempo de execução é conseguido através de uma política de negociação entre agentes do Cluster (CA) e agentes Globais (GA). Na Figura 7, uma solicitação de mapeamento de uma aplicação é enviada para CA do *cluster* origem que recebe todos os pedidos de mapeamento e negocia com o GA. Pode haver várias instâncias de GA que são sincronizadas ao longo do tempo. O GA tem informações globais sobre todos os clusters da NoC, a fim de tomar decisões sobre para qual *cluster* a aplicação deve ser mapeada. Possíveis respostas a este pedido de mapeamento:

- Quando existe um *cluster* adequado para a aplicação, o GA informa ao CA que requisitou o mapeamento, a existência desse cluster, que pede ao CA do *cluster* destino o mapeamento real da aplicação.

- Quando não há clusters adequados, o GA avisa o *cluster* mais promissor, onde é possível mapear a aplicação após a migração de tarefas.
- Quando não há nenhum cluster adequado e nem um candidato para migração de tarefas, então o conceito de reagrupamento é usado. Ele tenta encontrar PEs livres de vizinhos para aumentar o seu *cluster*. Se os requisitos forem satisfeitos após o reagrupamento, a aplicação pode ser mapeada no cluster.

Nos experimentos realizados, foi comparado o método ADAM com outros mapeamentos centralizados.

Em um cenário com apenas um *cluster*, mostrado na **Erro! Fonte de referência não encontrada.**, o método ADAM só se torna mais eficiente em redes com mais de 4096 PEs, com menos processadores a heurística NN se faz mais eficiente.

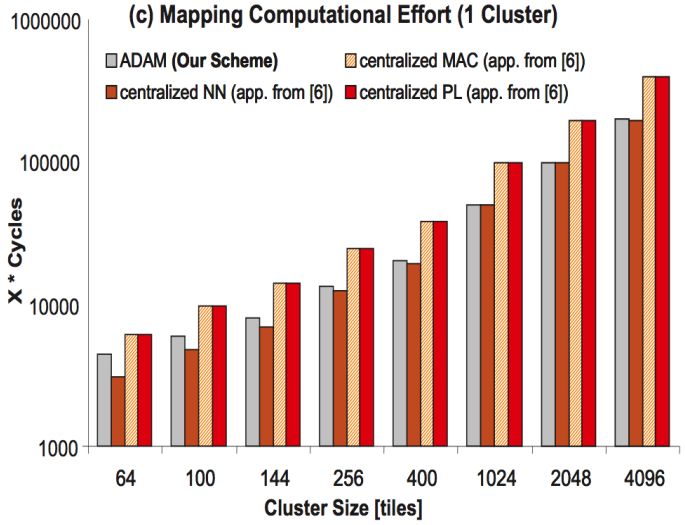

Figura 8 – Cenário de teste com apenas um cluster comparando o método proposto por ALF08 com os demais analisados.

Já em um cenário com diversos *clusters*, Figura 9, o método ADAM foi mais eficiente em redes com mais de 144 processadores, superando os demais métodos de mapeamento centralizado.

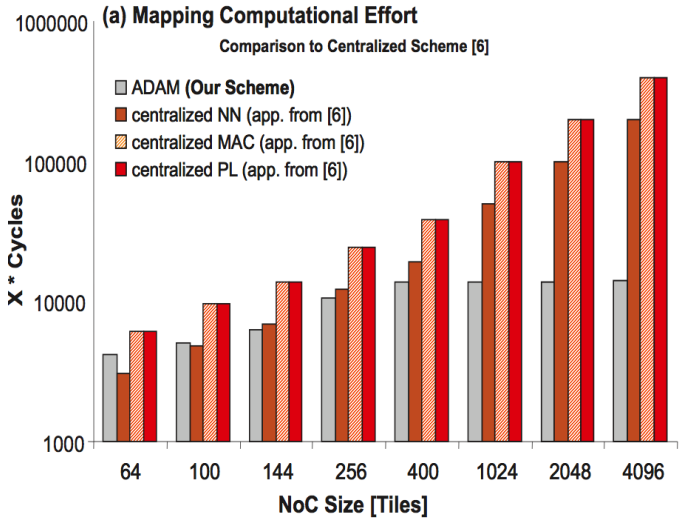

Figura 9 – Cenário de teste com diversos *clusters* comparando o método proposto por ALF08 com os demais analisados.

O que torna esse método relevante para a pesquisa realizada é a divisão das funções de mapeamento entre o agente global e um agente do *cluster,* não deixando só um agente a cargo dessa função. Porém, o ponto negativo, é que seu desempenho em redes com menos de algumas centenas de processadores é muito semelhante a alguns mapeamentos centralizados, até inferior em alguns casos, o que o tornaria útil somente para redes com centenas ou milhares de processadores. O trabalho não detalha como um sistema com 4096 PEs foi modelado.

2.3. Migração de Tarefas

2.3.1. Task Migration in Mesh NoCs over Virtual Point-to-Point Connections [GOO11]

O trabalho de [GOO11] propõe um método para migração de tarefas baseado em canais virtuais ponto-a-ponto (VIP) para criar conexões que proporcionem baixa latência e baixo consumo de energia para o fluxo de comunicação criado pelo mecanismo de migração de tarefas.

Em trabalhos anteriores sobre migração de tarefas, normalmente se considera cada par de PEs, origem-destino, e avalia-se o custo de inicialização e retomada do processo de migração. No trabalho do [GOO11], usa-se o conceito de submalhas, que é definido como um subconjunto da rede que contenha mais de um PE.

Por a migração de tarefas impor um atraso às mensagens de dados de outros PEs (a migração gera muito tráfego na rede), foi tentado reduzir o número de caminhos envolvidos na migração. Para isso foi usado o algoritmo Gathering-Routing-Scattering [CHE00], onde as tarefas que estão migrando são coletadas em um número limitado de PEs (nesse trabalho esses PEs são diagonais adjacentes) e transmitidas para a diagonal adjacente correspondente no destino. Como mensagens que transportam tarefas são maiores que mensagens normais, foram configuradas rotas para formar conexões dedicadas, responsáveis pela comunicação das mensagens de migração. Quando as mensagens de migração são recebidas nos núcleos da diagonal adjacente da submalha de destino, as tarefas são disseminadas para os destinos finais através de mensagens normais.

A Figura 10 mostra o processo de migração proposto pelo trabalho. A fase de coleta das tarefas é representada em vermelho, os *VIP* (caminhos) estão em azul e a fase de disseminação das tarefas é representada em verde.

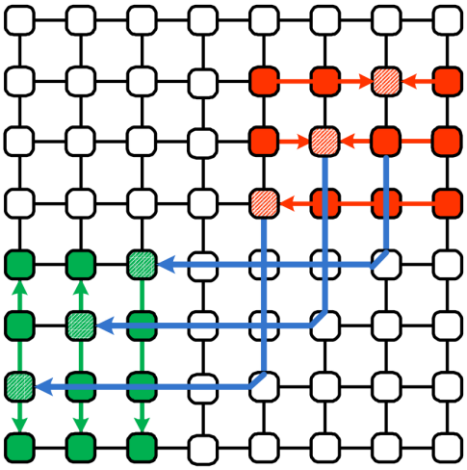

Figura 10 – Fases do processo de migração proposta por [GOO11].

Durante a migração, os PEs da submalha de origem param de se comunicar com outros PEs, limitando-se ao envio das mensagens de migração para a submalha de destino. Os demais PEs são informados sobre o processo de migração para evitar o envio de mensagens para os PEs da submalha de origem durante a migração. Entretanto, os PEs da submalha de destino podem continuar suas comunicações com outros PEs.

Experimentos foram realizados a fim de comparar o processo de migração proposto (com VIP) com outros dois processos: Gathering-Routing-Scattering e Diagonal [CHE00]. Todas as estratégias consideradas de migração de tarefas foram implementadas em uma arquitetura NoC simulada pelo Xmulator [NAY07]. Os experimentos foram realizados para uma plataforma de 128 bits, com uma NoC de tamanho 16x16, 32 flits de comprimento de mensagem, com frequência de 250 MHz.

A Figura 11 mostra os resultados da migração de tarefa de uma submalha 5x5 para outra submalha 5x5, sendo respectivamente ((2,2), (6,6)) e ((9,9), (13,13)) suas coordenadas na rede, em uma NoC 16x16. O eixo horizontal da figura representa a taxa de geração de tráfego, e o eixo vertical mostra a latência média de mensagem (em ciclos), a latência média da migração (em ciclos) e o total de energia consumida pela rede (em nJ/ciclos). Os resultados mostram que a proposta de migração de tarefas baseada em VIP reduziu a latência média de mensagem em 13%, a latência média da migração em 14% e o total de energia consumida pela rede em 10% comparada ao método Gathering-Routing-Scattering.

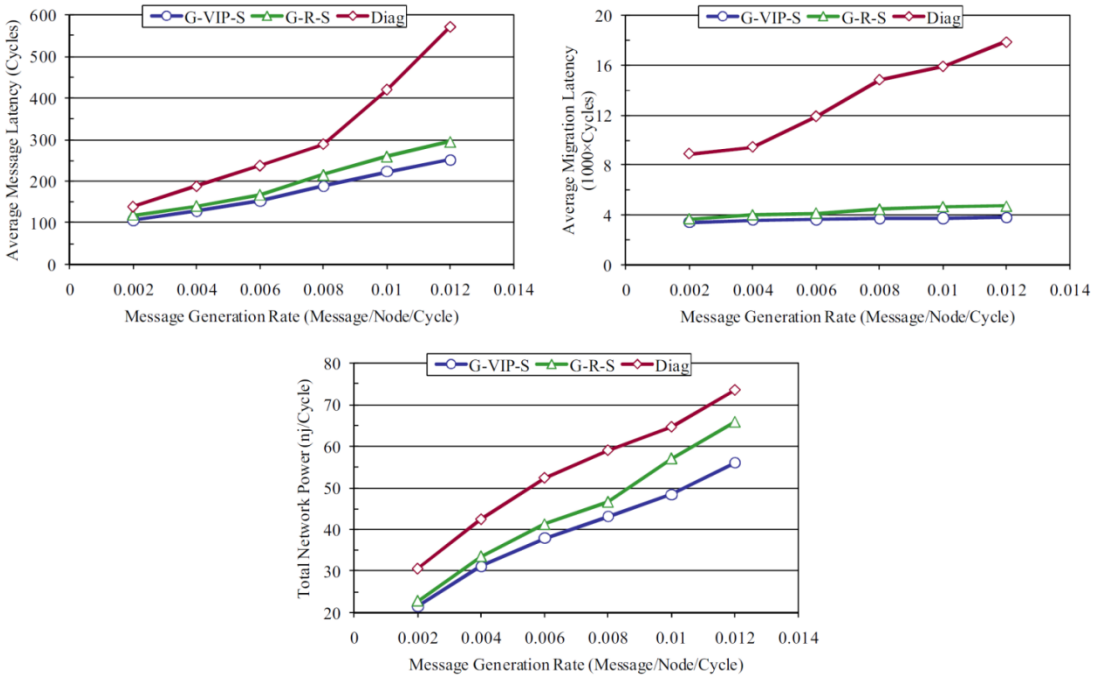

Figura 11 – Resultados para migração de tarefas proposta por [GOO11] em uma NoC 16x16.

2.3.2. Evaluating the Impact of Task Migration in Multi-Processor System-on-Chip [ALM10]

Este trabalho [ALM10] apresenta uma plataforma MPSoC capaz de migrar tarefas em tempo de execução. O sistema é distribuído, no sentido de que cada processador é capaz de tomar decisões locais, sem depender de uma unidade central. Toda a gestão é assegurada por um RTOS em cada processador, que é responsável principalmente pela execução e distribuição de tarefas entre os PEs. O objetivo dessa estratégia é melhorar o desempenho do sistema, assegurando a escalabilidade do projeto.

A arquitetura da plataforma é composta de uma matriz homogênea de elementos de processamento, interconectados pela NoC Hermes. A Figura 12 mostra uma visão estrutural da plataforma.

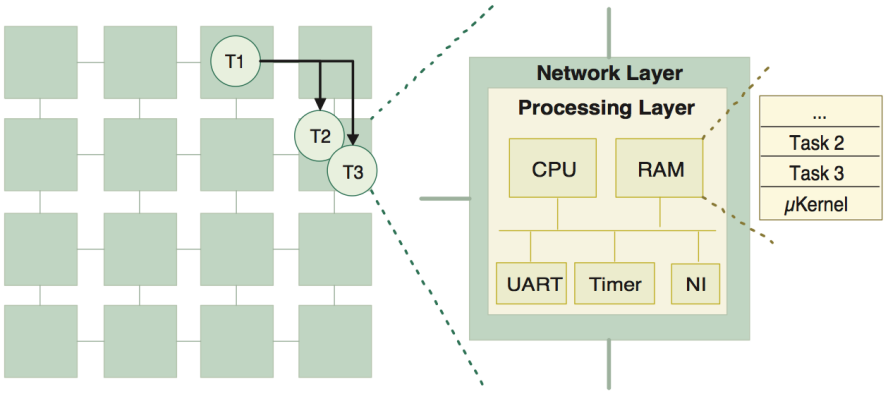

Figura 12 – Visão estrutural da plataforma usada em [ALM10].

O protocolo de migração de tarefas, ilustrado na Figura 13, foi implementado como segue. Considerando uma aplicação dividida em três tarefas, executando em uma arquitetura que possui uma NoC 2x2. No início do processamento, as tarefas estão executando em diferentes NPU (0...3). Num dado momento, NPU1 decide migrar T2 para NPU0 (passo 1). O NUP1 envia um pacote de controle para o mestre

da rede (NUP0) pedindo autorização para realizar a migração da tarefa (passo 2). O mestre verifica em sua tabela de roteamento se há uma ou mais tarefas enviando dados para a tarefa que deve ser migrada. Nesse caso T1 e T3. Em seguida, o mestre envia um pacote de controle para estas tarefas pedindo que elas não enviem mais pacotes para T2, para que a migração possa ser iniciada (passo 3). Vale ressaltar que o mestre apenas contém uma tabela global de roteamento, mas cada PE toma suas próprias decisões.

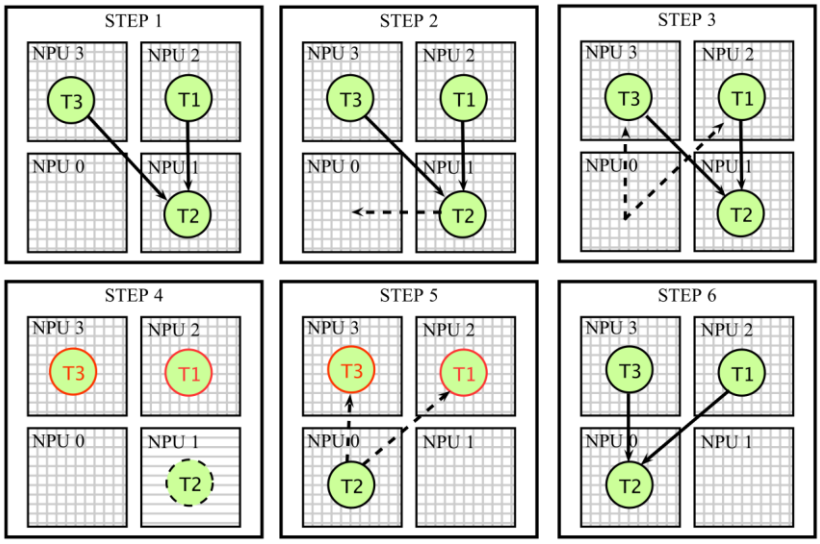

Figura 13 – Visão geral do protocolo de migração.

Imediatamente após a recepção do pacote de controle do Mestre, T1 e T3 param de enviar dados para T2 (passo 4). Então T2 é migrado para NUP0 (passo 5). Durante o processo de migração, somente o código objeto da tarefa é enviada através da NoC. A migração do contexto da tarefa não é suportada nesse sistema, portanto, tarefas que precisam manter seus parâmetros (como filtros adaptativos) não podem ser migradas. O próximo passo é registrar a tarefa na memória e inseri-la na lista de escalonamento. Esse processo é feito logo que a tarefa chega ao destino.

Finalmente, o mestre envia um pacote de controle para T1 e T3, informando que T2 foi migrado e a comunicação pode ser retomada. Este pacote de controle carrega a mensagem para a retomada da comunicação e a nova localização de T2.

Para avaliar o desempenho, a arquitetura do sistema foi descrita nas linguagens SystemC e VHDL, e foi usado como aplicação um decodificador MJPEG. Os resultados mostraram que a sobrecarga provocada pelo mecanismo de migração de tarefas é amortizada pelo ganho em termos de desempenho, que é cerca de 25% maior em comparação ao sistema sem migração.

2.4. Considerações Finais

Foram apresentados trabalhos com objetivos distintos, sendo três ([KOB11], [SHA11] e [FAT11]) com foco em desenvolver um controle distribuído a fim de tornar o MPSoC escalável, e compará-lo com arquiteturas centralizas, outro trabalho ([ALF08]) explora a idéia de mapeamento distribuído de tarefas em tempo de execução, visando evitar que um único agente tenha a função de mapear tarefas, e os dois últimos trabalho ([GOO11] e [ALM10]) são de migração de tarefas, com foco em proporcionar uma menor latência para o fluxo de comunicação criada pelo mecanismo de migração de tarefa [GOO11] e apresentar uma estratégia parcialmente distribuída de migração de tarefas [ALM10].

Entre todos os trabalhos revisados sobre controle distribuído, nenhum detalhou a arquitetura utilizada, e foram usadas linguagens alto nível para descrever o controle distribuído. No trabalho sobre mapeamento distribuído, a método proposto mostrou-se pouco eficaz em redes com menos de algumas centenas de processadores, já nos trabalhos revisados de migração de tarefas, nenhum detalhou o processo de migração de tarefas, assumindo uma modelagem abstrata para o mesmo.

3. PLATAFORMA DE REFERÊNCIA – MPSOC HEMPS

Neste Capítulo é apresentada a arquitetura e a estrutura de comunicação entre tarefas do MPSoC homogêneo HeMPS [CAR09], utilizado como plataforma de referência deste trabalho (Seção 3.1). Esta plataforma será modificada a fim de passar de uma plataforma com controle centralizado, executado por um mestre global, para uma plataforma com controle distribuído, controlado por mestres locais, posicionadas em regiões do MPSoC, regiões estas denominadas *clusters*. A Seção 3.2 apresenta o mecanismo de comunicação entre as tarefas.

3.1. Arquitetura da HeMPS

Os principais componentes da HeMPS são os elementos de processamento, denominados Plasma-IP, que são interconectados pela NoC Hermes [MOR04], utilizada como infraestrutura de comunicação. Além disso, tem-se uma memória externa, denominada de repositório de tarefas. Na Figura 14 é ilustrada uma instância da HeMPS utilizando a NoC Hermes de dimensão 2x3 interconectando os Plasmas-IP.

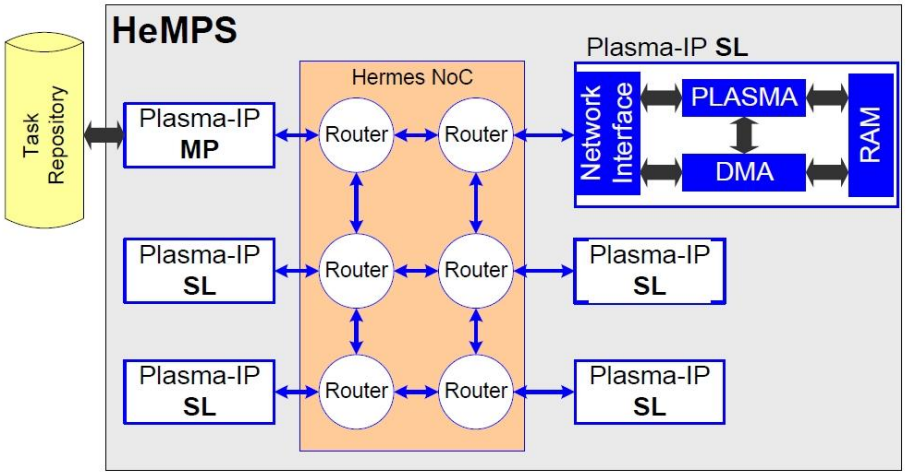

Figura 14 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09].

3.1.1. Plasma-IP

O Plasma-IP, elemento de processamento do MPSoC HeMPS, é subdividido em: mestre, chamado de Plasma-IP MP, responsável pela gerência dos recursos do sistema; e escravo, chamado de Plasma-IP SL. O MPSoC HeMPS contém apenas um Plasma-IP MP, pois possui uma gerência de recursos centralizada. O Plasma-IP contém os seguintes componentes:

- processador Plasma [PLA11]: é um processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS. O Plasma do MPSoC HeMPS possui algumas modificações em relação ao Plasma original, como por exemplo, a criação do mecanismo de interrupção, exclusão de módulos (UART) e inclusão de novos registradores mapeados em memória.
- memória privada: 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.

- interface de rede: realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.
- módulo de acesso direto à memória: possibilita o processador continuar sua execução de tarefas sem controlar diretamente a troca de mensagens com a rede. O DMA (do inglês, *Direct Memory Access*) tem como principal função transferir o código-objeto de tarefas que chegam à interface de rede para a memória do processador, e enviar para o processador mestre mensagens de depuração.

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 o modo *wormhole*, no qual um pacote é transmitido entre os roteadores em flits.

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

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.

3.2. 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 *microkernel*. 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.

Também existe a rotina chamada *Handler_NI()*, executada pelo *microkernel*, que trata a interrupção de *hardware* responsável pelo tratamento dos pacotes recebidos pela Interface de Rede. Assim, quando um pacote é recebido, a rotina *Handler_NI()* analisa o pacote verificando seu serviço, desmontando-o, e utilizando suas informações para tratamento do serviço. Estes serviços incluem:

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

Para exemplificar a funcionamento das trocas de mensagens, foi utilizada uma rede de tamanho 2x3, com a Tarefa A e a Tarefa B alocadas estaticamente nos processadores 02 e 12, respectivamente. A topologia dessa rede é mostrada na Figura 15. A Tarefa A executa somente a primitiva Receive() para a Tarefa B, e a Tarefa B executa a primitiva Send() para a Tarefa A.

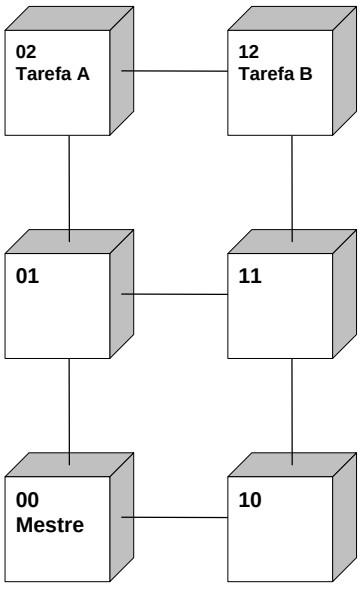

Figura 15 – Topologia de uma rede 2x3 usada para exemplificar a troca de mensagens.

No diagrama de sequência da Figura 16, pode-se ver a troca de mensagens entre os processadores.

No primeiro momento, o processador Mestre, envia para os processadores 02 e 12 o código objeto das tarefas A e B para serem alocadas. O envio dessas tarefas é realizado através da mensagem de TASK_ALOCATION. Após as tarefas serem alocadas, a Tarefa A faz um Receive() para a Tarefa B. Como ainda não houve comunicação entre as tarefas, o *microkernel* do processador executando a Tarefa A ainda não contém o endereço da Tarefa B. Com isso, a Tarefa A envia um LOCATION_REQUEST para o processador Mestre, requisitando a localização da Tarefa B. Então o processador Mestre envia para o processador 02 a localização da Tarefa B, através de outro LOCATION_REQUEST.

Após isso, a Tarefa B faz um Send() para a Tarefa A. Como no caso anterior, ele não sabe onde a Tarefa A se encontra. Por convenção, no Send(), ou WritePipe(), ao invés de se enviar um LOCATION_REQUEST, se envia um TASK_REQUEST para o processador mestre. Como resposta, o Mestre envia para o processador 02 e para o processador 12 uma mensagem de TASK_ALLOCATED. Então a Tarefa A envia um MESSAGE_REQUEST para a Tarefa B, requisitando dados. Enquanto a Tarefa A não receber a resposta da mensagem, esta tarefa fica bloqueada no estado Waiting. Se a Tarefa B já fez um Send() para Tarefa A, ele envia a reposta para Tarefa A através da mensagem de MESSAGE_DELIVERY. Quando as tarefas terminam, estas enviam para o Mestre uma mensagem TASK_TERMINATED, como resposta o Mestre envia para todos os processadores uma mensagem de TASK_DEALLOCATED, informando que a tarefa terminou.

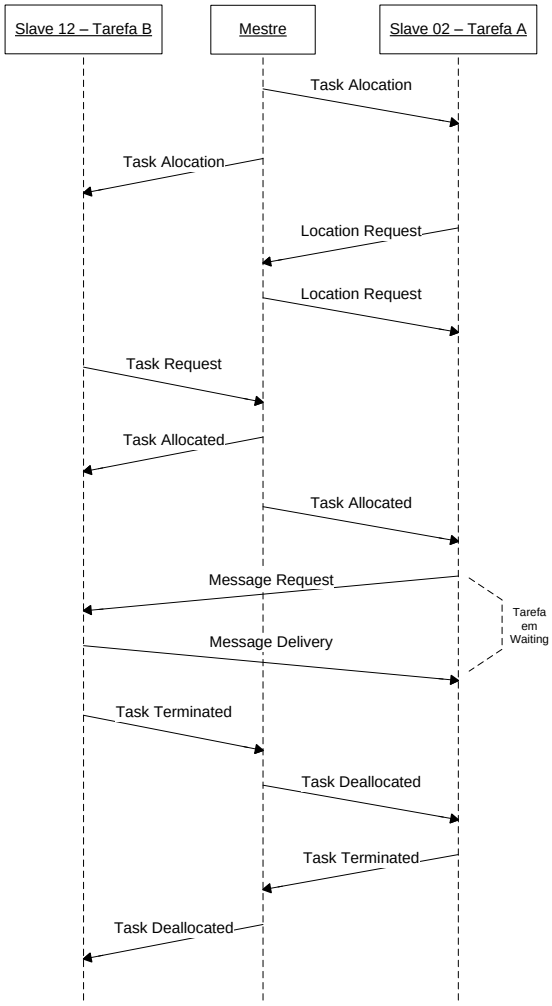

Figura 16 – Diagrama de sequência de troca de mensagens entre processadores.

4. ARQUITETURA DESENVOLVIDA

Neste Capítulo é apresentada a arquitetura com controle distribuído que foi desenvolvido, juntamente com a arquitetura com controle centralizado utilizado pela plataforma de referência HeMPS.

4.1. Arquitetura Centralizada X Arquitetura Distribuída

Utilizada pelo MPSoC HeMPS, a arquitetura centralizada, mostrada na Figura 17(a), consiste em reunir em um único elemento de processamento (Mestre Global), as funções de mapeamento de tarefas, migração de tarefas, interface com os sistemas externos (e.g. repositório de tarefas) e gerenciamento de recursos (controle de ocupação dos processadores).

Em MPSoCs com um grande número de PEs, a abordagem centralizada pode se tornar um gargalo, visto que somente o Mestre Global ficará encarregado de gerenciar toda a rede, responder as solicitações de serviços de todos os PEs e manter um mapa de localização de tarefas, o que pode se tornar inviável devido à quantidade de elementos de processamento. Por exemplo, na Dissertação de Marcsak [MAR10], o mestre era responsável por receber pacotes de monitoramento e manter uma estrutura de dados com a utilização dos enlaces da rede. Mesmo com uma quantidade pequena de pacotes de controle contendo informação de monitoramento, o mestre rapidamente tornava-se sobrecarregado.

Figura 17 – Arquitetura com controle (a) centralizado e (b) distribuído.

Já na arquitetura com controle distribuído, o MPSoC é subdividido em regiões (*clusters* virtuais), no qual um PE de cada região será o mestre local, e em uma região, existirá um mestre global, como mostrado na Figura 17 (b).

A diferença da arquitetura distribuída para uma arquitetura centralizada, é que a maior parte das funções atribuídas ao processador gerente da arquitetura centralizada passa a ser compartilhada pelos diversos mestres locais.

Os mestres locais (ML) passam a ter a função de:

- mapear as tarefas dentro do seu *cluster*;
- migrar tarefas, de aplicações gerenciadas por ele, para dentro do seu *cluster*, quando houver recurso livre;

- requisitar recursos a *clusters* vizinhos no caso de indisponibilidade de recursos para mapeamento ou migração;
- gerenciar aplicações;
- armazenar mensagens de depuração, e enviá-las em rajada para o mestre global;
- intermediar a comunicação entre os PEs do seu *cluster* com o mestre global;
- manter o mestre global atualizado sobre a utilização dos recursos de cada *cluster*.

Além das funções do ML, o mestre global do sistema também será responsável por:

- escolher em qual *cluster* uma aplicação será mapeada;
- saber quantos recursos estão disponíveis em cada *cluster*;
- realizar a interface com o mundo externo;
- receber mensagens de depuração e controle dos mestres locais.

4.2. Arquitetura Desenvolvida

A arquitetura com controle distribuído desenvolvida consiste em subdividir o MPSoC em *n* regiões (*clusters* virtuais) de mesmo tamanho, definidas em tempo de projeto. Em tempo de execução, se uma aplicação não couber em um cluster, esse deve pedir recursos para seus clusters vizinhos, processo no qual se dá o nome de reclusterização, que ainda não foi implementado nesse trabalho.

O MPSoC passa a ter 3 tipos de PEs: Mestre Global (GMP); Mestre Local (LMP); Escravo (SP).

O LMP é responsável por:

- controle do Cluster;
- executar os mapeamentos das tarefas;
- migração de tarefas;
- monitoramento;
- verificação de deadlines;
- comunicação com os outros LMPs e o GMP;

O GMP contém todas as funções do LMP, além de ser responsáveis por:

- escolher em qual cluster uma determinada aplicação será mapeada;
- controlar os recursos disponíveis em cada cluster;
- receber mensagens de depuração e controle provenientes de LMPs;
- acessar o repositório de aplicações (memória externa contendo todos os códigos objetos das tarefas);
- receber pedidos de novas aplicações a partir da interface externa;

Já os SPs são responsáveis pela execução das tarefas.

Quando o sistema inicia, o GMP é responsável pela inicialização dos clusters, informando os LMPs quais regiões eles iram gerenciar. Assim que o LMP conhecer a região que controlará, ele informa a todos os SPs de sua região, que será seu gerente. Esse mecanismo de inicialização dos clusters e dos SPs provê uma maior adaptabilidade ao sistema, permitindo a alteração do tamanho dos clusters em tempo de execução.

A Figura 18 mostra um MPSoC 9x9, contendo 9 clusters de tamanho 3x3. O GMP é somente um PE com acesso ao repositório de aplicações. Nesse exemplo, 9 PEs são reservados para as funções de gerentes, representando 11% dos PEs que não executam aplicações do usuário.

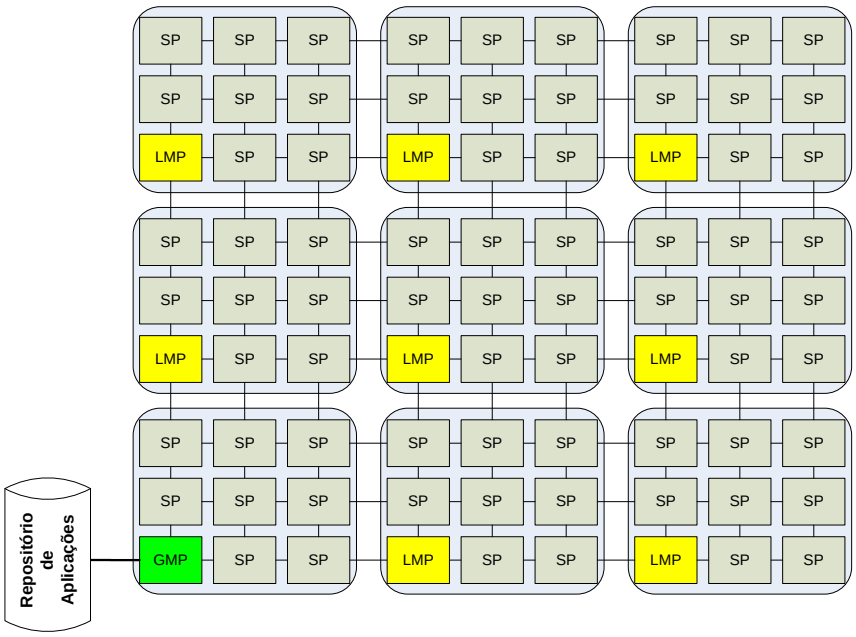

Figura 18 – Arquitetura com controle distribuído desenvolvido.

Como o mapeamento é a primeira ação executada pelo gerente quando uma aplicação entra no sistema, a próxima seção detalhará o processo de mapeamento distribuído.

4.3. Mapeamento Distribuido

Essa seção apresenta o processo de mapeamento distribuído. A seção 4.3.1 detalha o protocolo relacionado à seleção de clusters quando uma nova aplicação entra no sistema, detalhando a interação entre GMP→LMP. A seção 4.3.2 detalha o processo de mapeamento intra-clusters, detalhando a interação entre LMP→SP. A seção 4.3.3 compara qualitativamente o processo de mapeamento distribuído contra o centralizado.

4.3.1. Inserção de uma nova Aplicação

As aplicações são modeladas como grafos de tarefas. É suposto que pelo menos uma tarefa não possui nenhuma dependência por outras tarefas, sendo assim denominada(s) *tarefa(s) inicial(is)*. As aplicações são armazenadas no repositório de aplicações. De acordo com as solicitações dos usuários, uma nova aplicação pode ser solicitada para ser executada no sistema. Na Figura 19 essa ação corresponde à seta “nova aplicação”, entrando no GMP.

O GMP executa a heurística de “seleção de cluster” para escolher qual cluster irá receber a nova aplicação. A heurística escolhe o cluster que melhor se ajuste à aplicação em termos de disponibilidade de recursos. O numero de recursos de um dado cluster é definido de acordo com a equação 1.

$$Recursos\ do\ cluster = (PE\text{-}1) * páginas \qquad (1)$$

onde: PE é o numero de elementos de processamento do cluster (um PE é dedicado ao LMP), e as páginas são o numero de tarefas que um cada PE pode executar simultaneamente.

Uma vez o cluster selecionado, o GMP lê do repositório de aplicações à *descrição da aplicação* e a transmite ao LMP do cluster. A *descrição da aplicação* contém:

- Identificador de aplicação, juntamente com sua lista de tarefas.
- Identificação de cada tarefa que pertence a aplicação

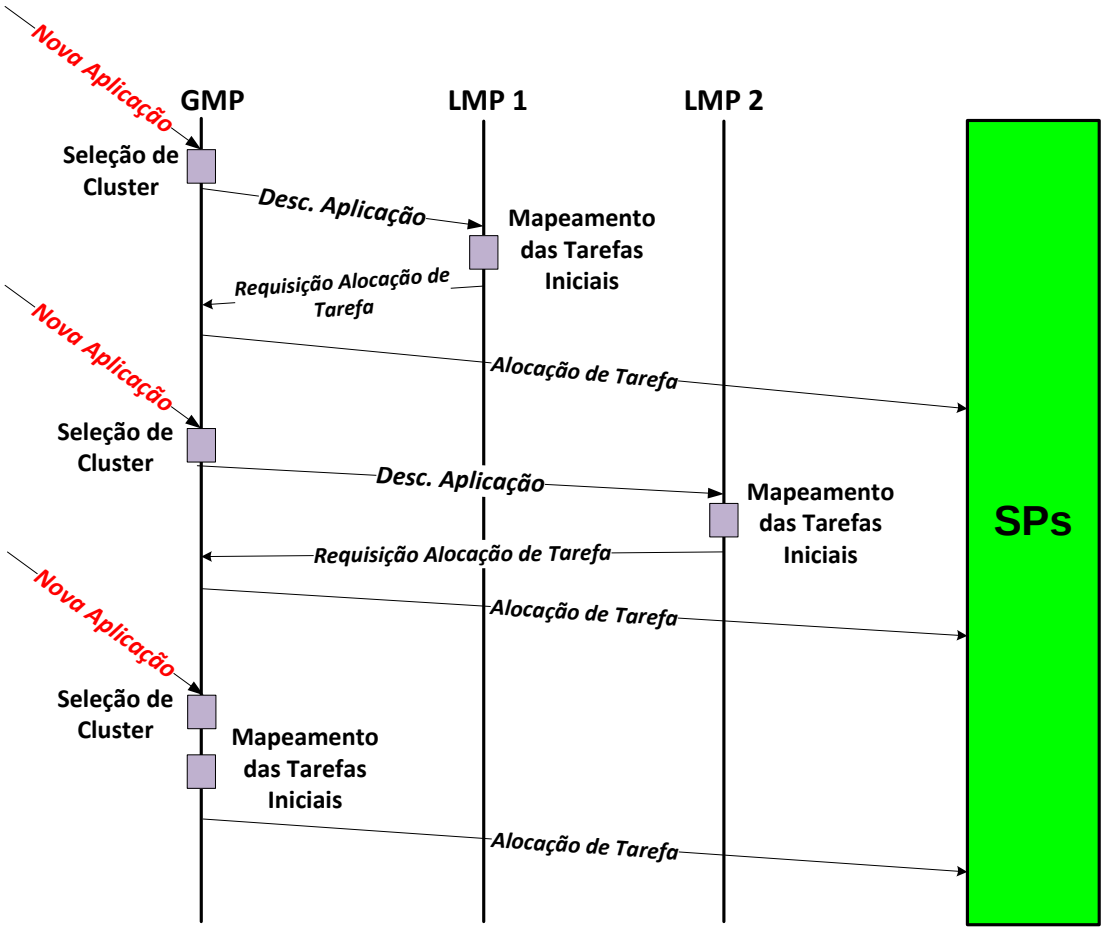

Figura 19 – Protocolo para inserção de uma nova aplicação no sistema.

O LMP selecionado armazena a *descrição da aplicação*, que inicia o processo de mapeamento das tarefas. Duas situações de mapeamento podem surgir: mapeamento das tarefas iniciais e mapeamento das tarefas remanescentes. O mapeamento das tarefas iniciais busca mapear as tarefas no SP com o maior número de recursos disponíveis ao seu redor. Isso aumenta a probabilidade de que as tarefas restantes da aplicação sejam mapeadas perto umas das outras, reduzindo a distância de comunicação entre tarefas, e por consequência, a energia de comunicação. O mapeamento das tarefas restantes é explicado a seguir.

Depois de selecionar o SP que receberá a tarefa inicial, o LMP envia um pacote para o GMP com o serviço *task allocation*. O GMP programa o modulo de DMA para ler a aplicação do repositório e transmitir

para o PE destino. O SP irá escalonar a nova tarefa no final da recepção do pacote de *task allocation*. Além disso, o LMP mantém uma tabela com todos os endereços das tarefas mapeadas.

Considerando a inserção da terceira aplicação na Figura 19, essa situação ilustra um cenário em que o cluster selecionado é aquele com o GMP. Neste caso, o GMP também executa o procedimento de mapeamento de tarefas.

4.3.2. Processo de Mapeamento de Tarefas

Cada aplicação é iniciada após o mapeamento da tarefa inicial. Quando uma tarefa inicial t_1 é iniciada, ela executa algumas funções, e em seguida, ele se comunica com uma determinada tarefa (send na Figura 20). O SP de origem da t_1 verifica se a tarefa alvo (t_2 no exemplo) está na tabela local de tarefas. Se estiver, a mensagem é transmitida para a tarefa alvo. Se não estiver, um pacote com o serviço de *task request* é transmitido para o LMP responsável pela t_1 ('1' na Figura 21).

```
Message msg1;
int main(){
   msg1.length = 128;
   ...
   send(&msg1,task_2);    //communicate with task_2
   ...
   exit();
}
```

Figura 20 – Exemplo de descrição de uma tarefa inicial, com o comando *send*.

Com o pacote de *task request* recebido, o LMP executa a heurística de mapeamento PREMAP-DN para selecionar o SP que devera receber t_2. Esta heurística de mapeamento minimiza a energia consumida na NoC, através da aproximação das tarefas com maior volume de comunicação. Distribuindo a heurística de mapeamento entre vários LMPs irá melhorar o desempenho geral do sistema, uma vez que o cálculo de mapeamento é um processo demorado.

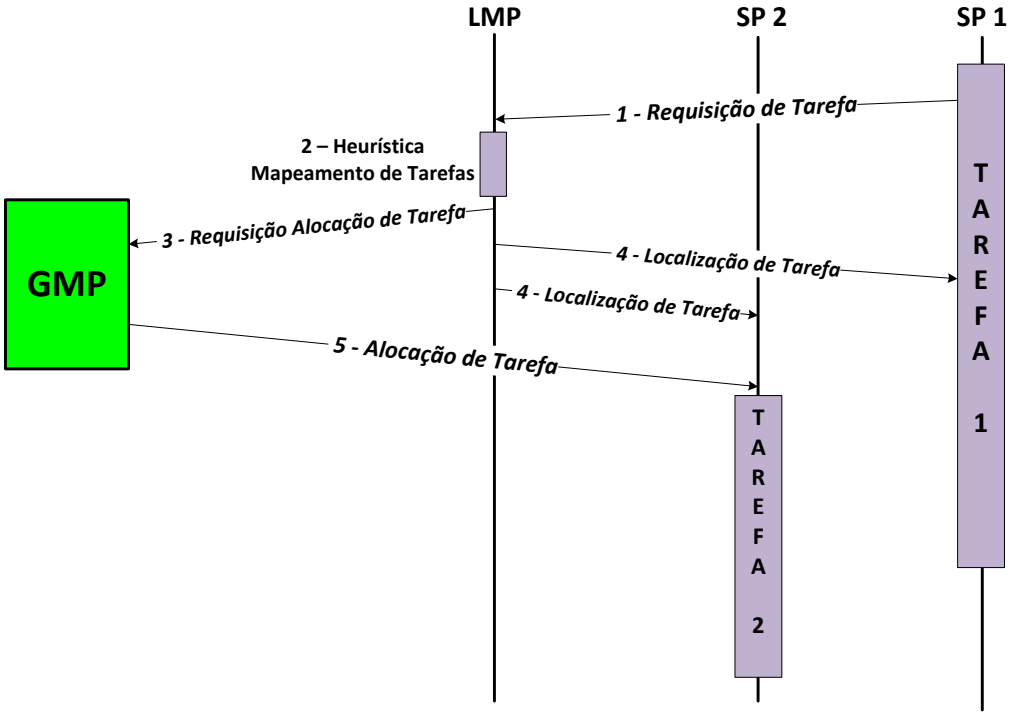

Figura 21 – Protocolo para mapear uma tarefa.

Supondo que a heurística de mapeamento selecionou o SP 2 para receber a t_2, assim o LMP envia um pacote para o GMP com o serviço de *task allocation request*, tal como explicado no mapeamento inicial de tarefas (‘3’ na Figura 21). O LMP também envia para o SP 1 a localização da t_2 e para o SP 2 a localização da t_1 (‘4’ na Figura 21). Essas localizações são armazenadas nas tabelas locais de tarefas. O último evento no processo de mapeamento de tarefa é a transmissão do código da tarefa pelo GMP para o SP 2 (‘5’ na Figura 21).

4.3.3. Mapeamento Centralizado VS Distribuído

Na abordagem de mapeamento centraliza, existe apenas um GMP, no qual é responsável pelo mapeamento de todas as tarefas. Todos os pedidos de mapeamento de tarefas são serializados (Figura 22 (a)), reduzindo o desempenho do sistema e aumentando o trafego na NoC na região do GMP. Usando o método de mapeamento distribuído de tarefas (Figura 22 (b)), o processamento do mapeamento é distribuído em vários LMPs, reduzindo a carga de comunicação gerada pelos pedidos de mapeamento.

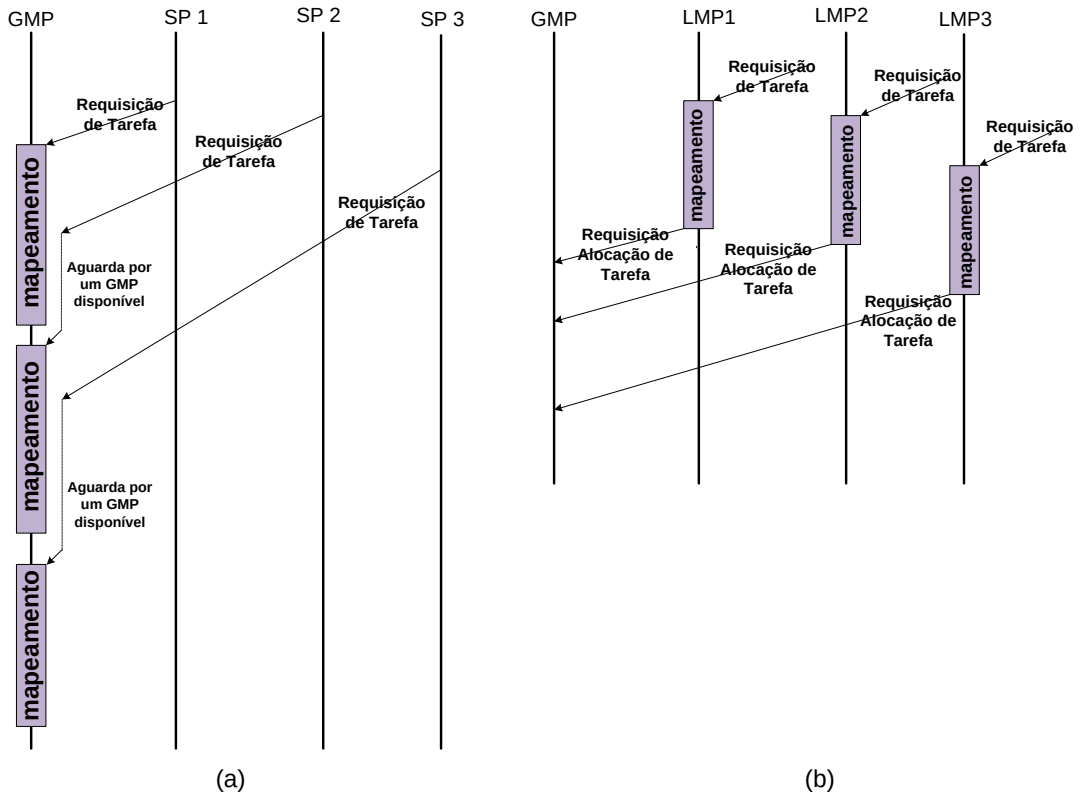

Figura 22 – Mapeamento Centralizado (a) vs Mapeamento Distribuído (b).

É importante mencionar duas limitações da abordagem distribuída. Mesmo se o mapeamento é distribuído, o acesso ao mundo externo (repositório de tarefas) não é. O impacto deste problema é minimizado pela transmissão dos dados das tarefas por meio de rajadas, usando a abordagem de DMA. A segunda limitação, é não incluir o procedimento de reclusterização (pedir recursos disponíveis para clusters vizinhos). Portanto, todas as aplicações devem caber em um cluster. O procedimento de reclusterização é um trabalho em andamento.

5. RESULTADOS PRELIMINARES

A partir dos trabalhos realizados no primeiro ano e no primeiro semestre do segundo ano de mestrado, apresenta-se os resultados preliminares do serviço de migração de tarefas, integrado à plataforma HeMPS e dos resultados do MPSoC com controle distribuído desenvolvido. Tais resultados de Migração foram submetidos e aceitos no IEEE International Symposium on Circuits and Systems 2012 (ISCAS 2012), e os resultados referentes ao MPSoC com controle distribuído foram submetidos ao IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012).

5.1. Migração de Tarefas

Migração de tarefas pode ser usada para balanceamento de cargas, tolerância a falhas e para restaurar o desempenho de uma determinada aplicação devido, por exemplo, a inserção de uma nova aplicação no sistema.

A Figura 23 mostra um exemplo motivador para aplicar a técnica de migração de tarefas. A Figura 23(a) apresenta o mapeamento de uma determinada aplicação composta de 6 tarefas (A-F). Em um cenário dinâmico, onde as aplicações são inseridas/removidas em tempo de execução, o MPSoC pode ter a maioria de seus recursos usados, restringindo a procura por recursos livres pela heurística de mapeamento.

Em um dado momento, uma nova aplicação pode ser carregada para o MPSoC (aplicação perturbadora na Figura 23(b)), com suas comunicações competindo com a comunicação da aplicação principal. Desta forma, o desempenho da aplicação principal é penalizado, e o *microkernel* do PE que executa a tarefa C pede ao PE mestre (M) para ser migrado. O PE mestre escolhe a nova posição e a tarefa é migrada para uma posição próxima as suas tarefas comunicantes (Figura 23(c)), restaurando/melhorando o desempenho da aplicação.

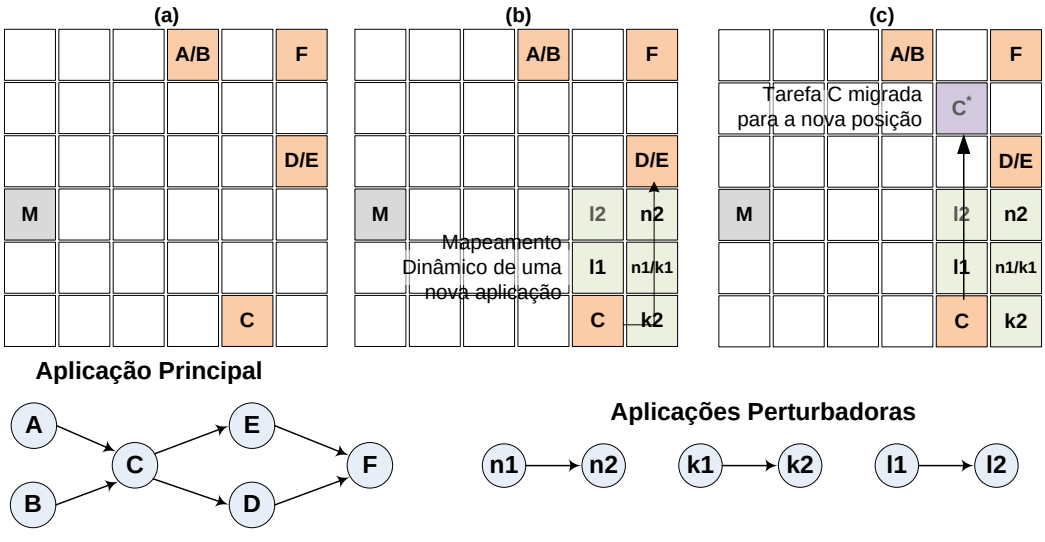

Figura 23 – Exemplo motivacional mostrando benefícios da migração de tarefas.

5.1.1. Protocolo da Migração de Tarefas

O protocolo de migração de tarefas é ilustrado na Figura 24. Ele foi desenvolvido e integrado a plataforma HeMPS ao longo do primeiro ano do mestrado, e pode ser resumido da seguinte forma:

1) Detecção da necessidade de migração, resultando em um pedido de migração de tarefas para o PE mestre. Essa detecção está fora do contexto deste trabalho, sendo alvo de estudo do mestrando Guilherme Madallozo.

2) O PE mestre executa a heurística para calcular a nova posição da tarefa. Nota-se que a tarefa a ser migrada (T1) continua a sua execução em paralelo.

3) O PE escravo, que solicitou a migração de tarefas, recebe a nova posição da tarefa.

4) A tarefa só pode ser migrada se ela está no estado de execução. O escalonador do *microkernel* verifica essa condição. Se a tarefa pode ser migrada, o *microkernel* envia para o PE destino, um pacote com o conteúdo completo da pagina da tarefa e seus descritores de tarefas (TCB – Task Control Block).

5) A tarefa migrada (T1*) é escalonada uma vez que o código e o TCB forem completamente recebidos.

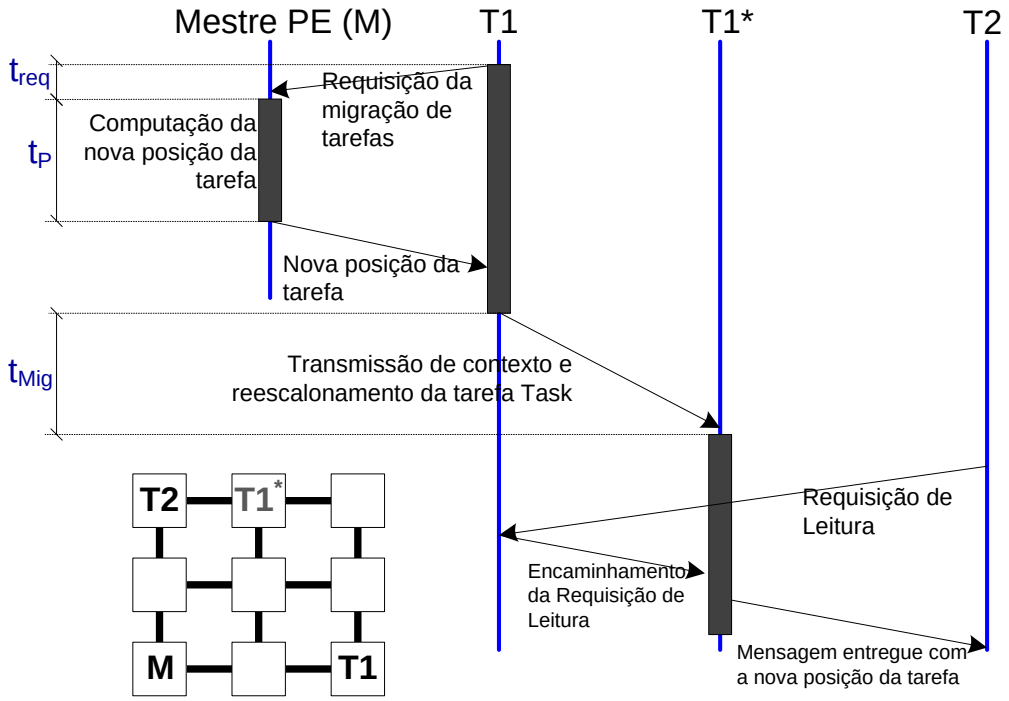

Figura 24 – Protocolo de migração de tarefas.

O desempenho do protocolo de migração é uma função do t_{req} e t_P, tempo para solicitação de migração e tempo para calcular a nova posição da tarefa, respectivamente, e também por t_{Mig}, que é o tempo para transmitir a tarefa e escaloná-la no novo PE. Durante t_{req} e t_P, a tarefa a ser migrada continua sua execução, já durante t_{Mig}, a tarefa suspende sua execução.

Este protocolo de migração envolve o desenvolvimento do controle da comunicação entre os PEs, implementado no *microkernel* dos PEs escravos, e o desenvolvimento da heurística de migração. O protocolo de migração foi desenvolvido por dois mestrandos. Toda a parte relativa ao controle da comunicação entre os PEs foi desenvolvido pelo proponente deste trabalho, resultando no domínio do *microkernel* da plataforma HeMPS, além de tornar a migração de tarefas operacional. A heurística de migração foi desenvolvida por Guilherme Madalozzo, apresentada na próxima subseção.

5.1.2. Heurística de Migração de Tarefas

A heurística de migração é executada no PE mestre após receber uma solicitação de migração de tarefas. A heurística proposta apresentada na Figura 25 é semelhante a um processo de mapeamento dinâmico [MAN11]. A linha 1 calcula a energia de comunicação inicial com a tarefa a ser migrado na sua posição original. Assumindo como exemplo o gráfico de tarefas da aplicação principal ilustrada na Figura 23, a tarefa a ser migrada (t_M) é a tarefa C, que se comunica com as tarefas A, B, D e E. Desta forma, o conjunto de tarefas comunicantes é N = {A, B, D, E}. A heurística, na linha 4, define o conjunto de PEs dentro do quadrado envolvente definidos por N – função *processor_box* - P. As linhas 6-11 verificam se o PE no centro do quadrado envolvente (pc) pode receber t_M, calculando a energia de comunicação entre t_M, mapeado no pc, e todos os elementos em N.

Em seguida, nas linhas 14-18, os outros elementos em P são avaliados para receber t_M. Se um dado PE em P pode receber t_M, o PE escolhido é o que minimiza a energia de comunicação (linha 22). Caso contrário, o espaço de busca é ampliado em um hop (linha 23), e se os novos PEs forem adicionados a P, a heurística continua a procura por uma posição para t_M. Se não for possível adicionar elementos ao P, isso significa que todos os recursos do MPSoC estão sendo usados, e a migração não é possível.

```
Input: t_M: task to be migrated; p_M:PE executing t_M; N:set containing the tasks t_M communicates with
Output: p_T: PE to receive the task t_M
1.    migration_cost = cost(p_M,t_M,N)        // initial migration cost
2.    p_T ← NULL
3.    // define the PE set that may receive t_M
4.    P←processor_box(N)
5.    // define the PE address in the center of the set P
6.    p_c ←center(P)
7.    // if the central PE may receive t_M, the initial migration cost is computed, and p_T receives p_c
8.    IF migrate(p_c,t_M )=TRUE and cost(p_c,t_M,N)<cost(p_M,t_M,N) THEN
9.        migration_cost= cost(p_c,t_M,N)
10.       p_T←p_c
11.   ENDIF
12.   DO
13.       // verifies all other PEs in P
14.       FOR ALL ELEMENTS p_i IN P
15.           IF cost(p_i,t_M,N)<migration_cost THEN
16.               migration_cos t= cost(p_i,t_M,N)
17.               p_T←p_i
18.       ENDIF
19.       // verifies if migration is possible in the set P
20.       IF p_T≠ NULL THEN
21.           return p_T
22.       ELSE
23.           P´← extend_search_space_1_hop(P)
24.           IF P´ = P THEN
25.               return NULL          // migration is not possible
26.           ENDIF
27.           P←P´
28        ENDIF
29.   ENDDO
```

Figura 25 – Pseudocódigo da migração de tarefas.

A heurística apresentada somente migra uma determinada tarefa se a migração minimizar a energia de comunicação, caso contrário, a tarefa permanece na sua posição original. A complexidade da heurística é O(NPE), onde NPE é o número de PEs no MPSoC. A carga do processador ainda não é considerada na heurística.

5.1.3. Resultados

Os resultados foram obtidos em uma instância de tamanho 6x6 do MPSoC HeMPS. Parâmetros relevantes da arquitetura para a migração de tarefas incluem:

- Tamanho da palavra do PE e do flit: 32 e 16 bits respectivamente.

- Tamanho da página: 16 Kbytes (4,096 palavras, código e dados).
- *Time slice* da tarefa: 16,384 ciclos de relógio.
- Escalonador: round-robin, consumindo em média 260 ciclos de relógio.

Custo da Migração de Tarefas

O primeiro conjunto de parâmetros avaliados compreende t_{req} e t_P (Figura 24). O pacote de requisição de migração de tarefa e o pacote contendo a nova posição da tarefa são pequenos, 8 flits, sendo sua latência proporcional ao numero de saltos no caminho entre origem e destino. Considerando que o roteamento e a arbitragem levam 5 ciclos de relógio, a distância de 7 saltos até o mestre (C-M, Figura 23) resulta em t_{req}=35 ciclos de relógio, se não houver outro trafico competindo pelo mesmo recurso. O tempo para computar a nova posição da tarefa, t_P, é em média 4,100 ciclos. Portanto, a latência da NoC tem um impacto muito pequeno nessa etapa do protocolo. Também, o tempo para computar uma nova posição para uma tarefa pode ser considerado pequeno, uma vez que é inferior a um *time slice*.

Para o cenário apresentado na Figura 23, a migração da tarefa C levou 11,895 ciclos de relógio (o tamanho do pacote contendo a tarefa, o TCB e algumas variáveis de controle é de 8,590 flits). O tempo total gasto entre a última execução da tarefa C em sua posição original, e o tempo que ela foi escalonada em sua nova posição (t_{Mig}) foi de 12,326 ciclos de relógio. Note que o t_{Mig} está relacionado com o tamanho da tarefa migrada.

Este resultado revela que a migração de tarefas tem um impacto pequeno no desempenho do sistema, como é demonstrado na próxima subseção.

Avaliação utilizando uma aplicação sintética

O desempenho do protocolo de migração de tarefas foi avaliado de acordo com os três cenários apresentados na Figura 23:

a) aplicação principal executando sozinha no MPSoC;

b) aplicação principal executando com 3 aplicações perturbadoras (produtor-consumidor);

c) o mesmo que o cenário ‘b’, com a tarefa C migrada depois de 2.1 ms.

A aplicação principal executa 150 iterações em todos os cenários. A Figura 26 apresenta a vazão da tarefa F, em Mbps, para as primeiras 40 iterações. As tarefas iniciais da aplicação (‘A’ e ‘B’) são mapeados primeiro, produzindo dados para as tarefas ainda não mapeadas. Quando as tarefas consumidoras são mapeadas, elas podem consumir rapidamente, o que explica a alta vazão observada nas primeiras iterações. O cenário ‘a’ estabiliza a sua vazão na sétima iteração. Nos cenários ‘b’ e ‘c’, estabilizam depois, devido ao alto numero de tarefas a serem mapeadas (aplicação principal e aplicações perturbadoras). É perceptível a redução da vazão no cenário ‘c’ (iteração 21), durante o período de migração da tarefa, e então a vazão atinge aos mesmos valores do cenário ‘a’, cenário sem aplicações perturbadoras.

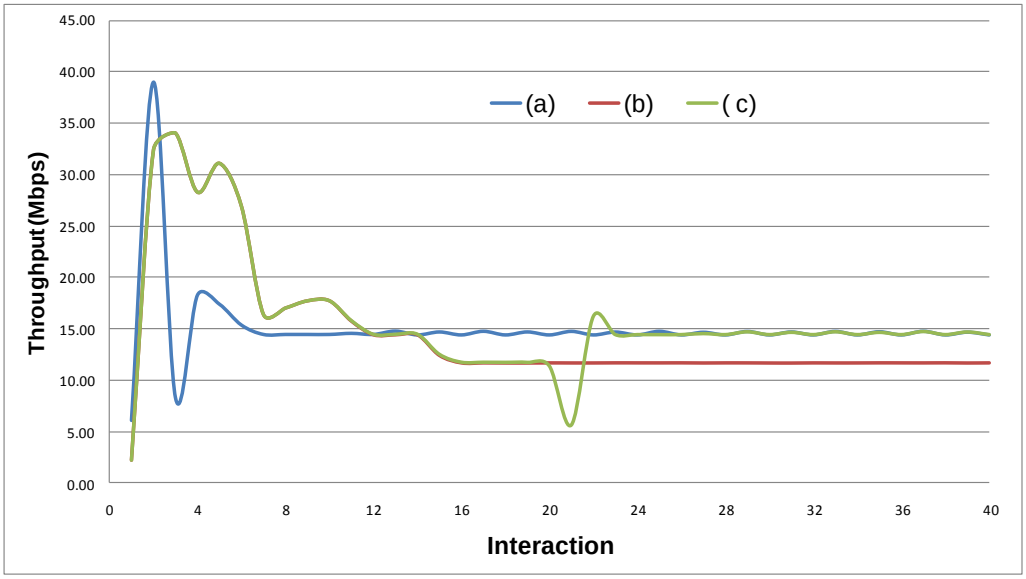

Figura 26 – Vazão da tarefa F, em Mbps. (a) corresponde a aplicação principal executando sozinha no MPSoC; (b) aplicação principal com tráfego perturbador; (c) o mesmo que o 'b' com a tarefa C sendo migrada.

A vazão da aplicação principal, sem tarefas perturbadoras é em média (medido depois da estabilização da vazão – trigésima iteração) 14.62 Mbps, caindo para 11.97 Mbps (-18.1%) quando as aplicações perturbadoras são inseridas. Tal resultado demonstra o impacto do tráfico da NoC no desempenho geral do sistema. Migrando a tarefa C perto de suas tarefas comunicantes, a vazão média torna-se 14.63 Mbps, ligeiramente superior ao original, devido a proximidade da tarefa C com duas tarefas comunicantes depois da migração.

O tempo de execução da aplicação principal, em ciclos de relógio, foi 1,337,172 (valor de referência), 1,626,812 (+21%) e 1,365,272 (+1.02%), para os cenários 'a', 'b' e 'c' respectivamente. Como mencionado na subseção anterior, o sobrecarga esperada causada pela migração de tarefas em tempo de execução seria pequeno. No experimento apresentou um acréscimo ao tempo total de execução de somente 1.02%.

5.2. MPSoC com Controle Distribuído

Os experimentos foram executados no MPSoC HeMPS, modelado em SystemC. Cada cluster contém 8 SPs (clusters 3x3) e cada SP pode executar até 2 tarefas simultâneas. Portanto, cada cluster pode executar 16 tarefas (ver equação 1, página 27). Três *benchmarks* foram avaliados: MPEG - executa um decodificador parcial de MPEG; análise de imagem MultiSpec - avaliar a similaridade entre 2 imagens usando freqüências diferentes [TAN08]; Synthetic - uma aplicação sintética [WAC11].

A Tabela 1 apresenta nove cenários avaliados (A, B, ..., I). A segunda coluna da Tabela 1 contém o tamanho do MPSoCs, o número de clusters e o número de SPs para as abordagens de gerência distribuído e centralizada. Nota-se que a abordagem centralizada tem mais SPs que a abordagem distribuída, uma vez que não usa LMPs. A terceira coluna apresenta o número de tarefas para cada benchmark, enquanto a coluna seguinte mostra o número de instâncias das aplicações (App_{CL}) que podem executar no cluster. A quinta coluna contém o número total de tarefas que devem ser mapeadas no MPSoC. As duas últimas colunas apresentam a utilização do sistema para as abordagens distribuídas e centralizadas, isto é, a percentagem de recursos utilizados do sistema (número de tarefas / (numero de SPs * 2)).

Tabela 1 – Características dos cenários avaliados

	MPSoC Size	Benchmark - Nb. of Tasks	App_{CL}	Total number of tasks	SU_{dist}	SU_{centr}
A	**6x6** - 4 clusters - 32 SPs (distributed) - 35 SPs (centralized)	Syntetic - 6	2	48	75%	69%
B		MPEG - 5	3	60	94%	86%
C		Multispec -14	1	56	88%	80%
D	**9x9** - 9 clusters - 72 SPs (distributed) - 80 SPs (centralized)	Syntetic - 6	2	108	75%	68%
E		MPEG - 5	3	135	94%	84%
F		Multispec -14	1	126	88%	79%
G	**12x12** - 16 clusters - 128 SPs (distributed) - 143 SPs (centralized)	Syntetic - 6	2	192	75%	67%
H		MPEG - 5	3	240	94%	84%
I		Multispec -14	1	224	88%	78%

Todas as instâncias das aplicações são inseridas no sistema ao mesmo tempo (1 ms) para maximizar a utilização de SPs. A Tabela 2 apresenta a redução do tempo de execução, adotando o mapeamento centralizado como referência. Tais resultados são uma clara demonstração do problema de escalabilidade relacionado com o gerenciamento centralizado de recursos em MPSoCs. Um cenário não reduz o tempo total de execução utilizando o mapeamento distribuído devido à baixa utilização do sistema. Como a abordagem centralizada tem mais recursos livres, alguns SPs podem receber uma tarefa, em vez de duas, o que reduz o tempo de execução da aplicação, já que não há compartilhamento de tempo entre as tarefas. Nota-se que este comportamento também se repete nos cenários D e G, onde o Benchmark Synthetc apresenta ganhos menores do que os cenários E/F e H/I, respectivamente.

O tempo de mapeamento na abordagem distribuída é menor, pois o espaço de busca é menor. Por exemplo, em um MPSoC 12x12, o mapeamento centralizado tem que avaliar o estado de 143 SPs, enquanto no mapeamento distribuído o espaço de busca é sempre o mesmo, proporcional ao tamanho do cluster. Portanto, a execução da heurística de mapeamento é mais rápida na versão distribuída.

Tabela 2 – Redução do tempo total de execução, adotando o mapeamento centralizado como referência.

Scenario	**MPSoC Size**	**Benchmark**	**Execution time reduction (w.r.t centralized mapping)**
A	6x6	Synthetic	-15% (increase of time)
B		MPEG	28%
C		Multispect	34%
D	9x9	Synthetic	50%
E		MPEG	63%
F		Multispect	54%
G	12x12	Synthetic	79%
H		MPEG	86%
I		Multispect	85%

A Figura 27 apresenta os tempos, em ciclos de relógio, de quando cada instância de aplicação começa e termina, para o cenário B, E e H.

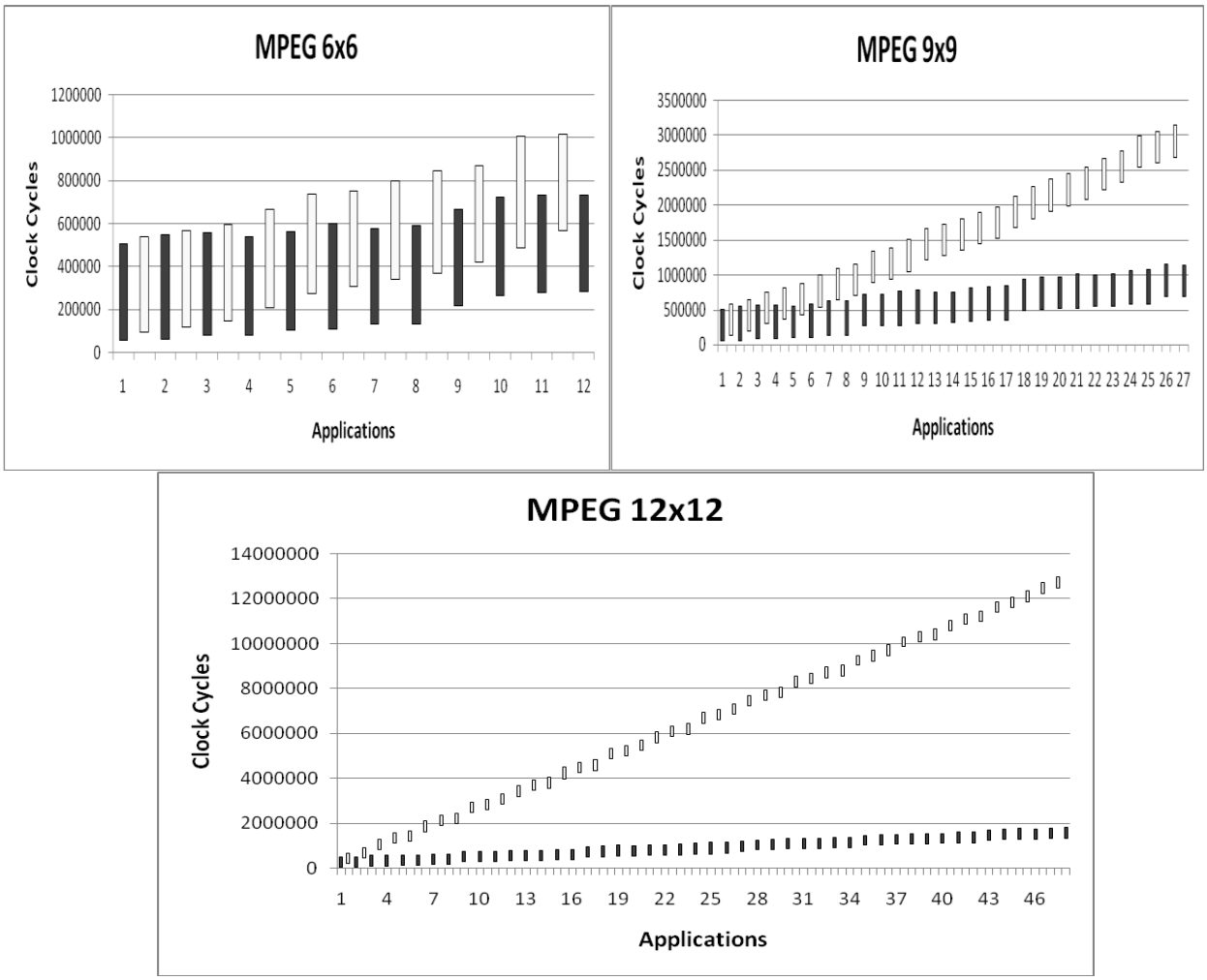

Figura 27 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark MPEG com três tamanhos de NoC, cenários B, E e H.

É importante observar na Figura 27 que em todos os casos, as aplicações tem aproximadamente o mesmo tempo de execução. Note que no mapeamento distribuído (barras cinzas) um conjunto de aplicações começam simultaneamente, devido a execução distribuída da heurística de mapeamento, como ilustrada na Figura 22 (b). Por outro lado, o mapeamento centralizado (barras brancas) tem que mapear as tarefas da aplicação (5 tarefas no Benchmark MPEG) e tratar o pedido de novas aplicações. Esta serialização do processo de mapeamento é claramente observado em todos os resultados da Figura 27, o que também explica os resultados observados na Tabela 2. À medida que o tamanho do MPSoC aumenta, o tempo de execução para o mapeamento centralizado cresce dramaticamente.

Outro beneficio do mapeamento distribuído é a redução do trafego ao redor do GMP, pois muitas mensagens de controle são tratadas dentro do cluster, e somente a mensagem de controle *task request* é enviada para o GMP.

A Figura 28 e a Figura 29 apresentam, respectivamente, os tempos dos cenários C, F, I e dos cenários A, D e G.

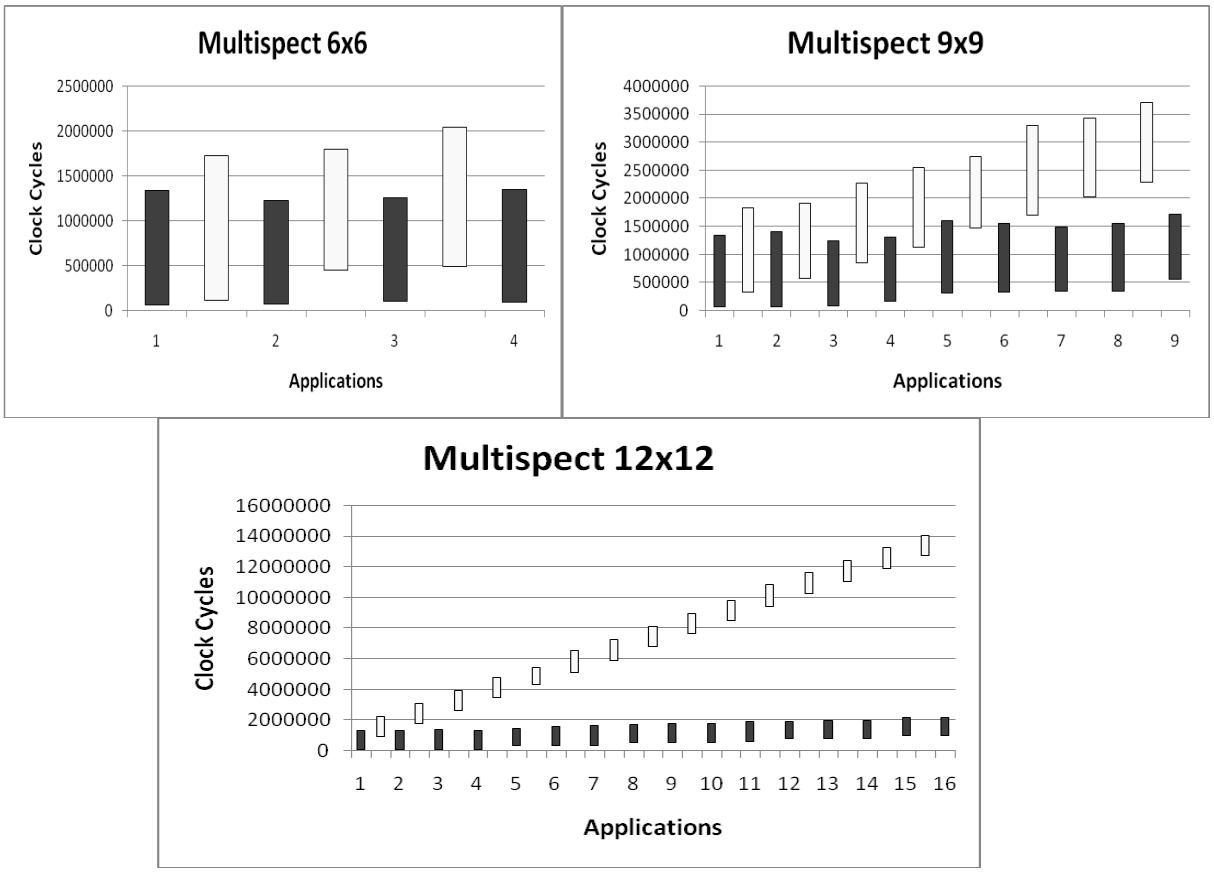

Figura 28 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark Multispect com três tamanhos de NoC, cenários C, F e I.

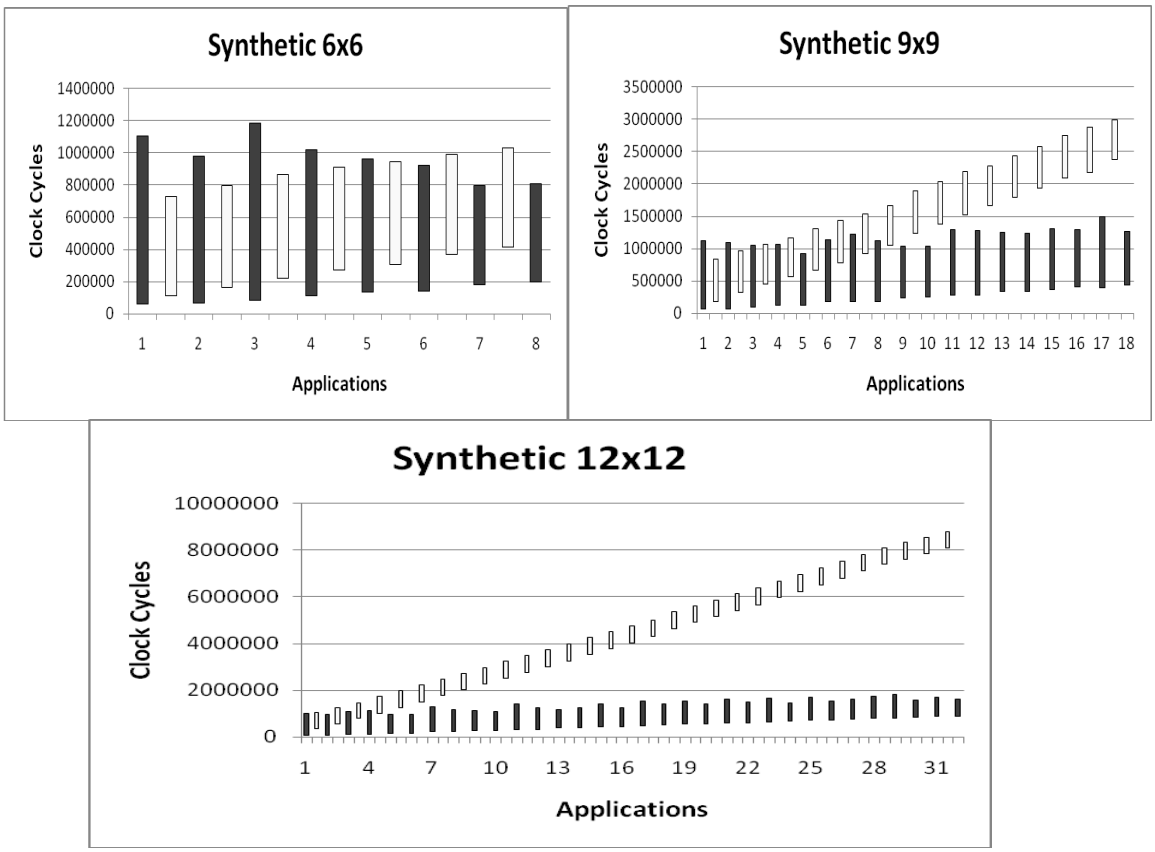

Figura 29 – Mapeamento Distribuído (barras cinzas) *versus* Centralizado (barras brancas), para o Benchmark Synthetic com três tamanhos de NoC, cenários A, D e G.

6. CRONOGRAMA E ATIVIDADES

A arquitetura de controle distribuído desenvolvido tem como limitação a execução de cada aplicação em apenas um cluster, o qual pode comprometer o desempenho de todo o sistema, uma vez que se não houver um cluster com a quantidade de recursos livres demandados pela aplicação, a mesma fica sem ser alocada, até os recursos necessários sejam liberados.

Uma vez desenvolvido o compartilhamento de recursos entre clusters, essa limitação não existirá mais, mas o sistema poderá ficar fragmentado. Esta limitação será resolvida com a integração da migração de tarefas, em desenvolvimento pelo mestrando Guilherme Madalozzo.

A Tabela 3 apresenta o cronograma original de tarefas propostas para o segundo ano de mestrado.

Tabela 3 – Cronograma de Atividades apresentado no PEP.

Atv.	Objetivo	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
1	Revisão do estado da arte												
2	Avaliação de arquitetura com controle centralizado	X	X										
3	Criação do kernel mestre local (ML)			X	X	X							
4	Heurística de mapeamento para o cluster				X	X	X						
5	Alterar kernel mestre global (MG)			X									
6	Integrar e avaliar o mapeamento distribuído						X	X	X				
7	Seminário de Andamento								X				
8	Migração distribuída												
9	Alterar roteador dos mestres												
10	Criar MPSoC com rede de controle distribuído												
11	Integrar e avaliar com rede dedicada												
12	Escrita de Artigo				X								
13	Escrita da dissertação												

As atividades 1-6 foram realizadas. Resumidamente 3 grandes macro-atividades foram desenvolvidas no primeiro semestre de 2012: avaliação da arquitetura com controle centralizado; criação dos kernels do mestre local e global; integração e avaliação da arquitetura com controle distribuído.

As atividades relacionadas à criação de uma rede dedicada de controle **não** serão realizadas (atividades 8/9/10), pois optou-se em aperfeiçoar a arquitetura desenvolvida, implementando o compartilhamento de recursos entre clusters e migração de tarefas. Outra razão para esta alteração é a complexidade do processo de reclusterização.

Assim, o cronograma de atividades para o segundo semestre á apresentado na Tabela 4.

Tabela 4 – Cronograma de Atividades para o segundo semestre de 2012.

Atv.	Objetivo	Ago	Set	Out	Nov	Dez
1	Revisão do estado da arte					
2	Compartilhar recursos entre Clusters - Se uma aplicação não couber em um cluster, pedir recursos para seus cluster vizinhos.					
3	Migração distribuída – se existir um recurso disponível no cluster e existir uma tarefa fora do cluster pertencente a uma aplicação gerenciada pelo respectivo mestre local.					
4	Avaliar o desempenho do MPSoC com compartilhamento de recursos e migração de tarefas.					
5	Escrita de Artigo					
6	Escrita da dissertação					

7. CONCLUSÃO

Este trabalho propôs um esquema de gerenciamento de recursos distribuídos com o objetivo de proporcionar escalabilidade para os MPSoCs. A abordagem proposta divide o sistema em clusters de tamanho fixo, controlados por um mestre local, reduzindo a carga computacional e de tráfego em comparação com uma gerencia centralizada. A proposta foi avaliada por meio de simulações com precisão de clico, e os resultados demonstraram a eficácia na gerencia distribuída utilizando o mapeamento de tarefas como estudo de caso. O aspecto negativo da proposta é que alguns processadores são reservados a gerencia dos recursos.

A continuidade desse trabalho será a inclusão da reclusterização, que permite alterar o tamanho do cluster em tempo de execução, e da migração de tarefas ao MPSoC desenvolvido.

8. REFERÊNCIAS

[AGU08] Aguiar, A.; Filho, S.J.; Santos, T.G.; Marcon, C.; Hessel, F. "*Architectural support for task migration concerning MPSoC*". In: WSO Workshop, SBC, 2008, pp.169-178.

[ALF08] Al Faruque, Mohammad Abdullah; Krist, Rudolf; Henkel, Jorg, Henkel. "ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication". In: DAC, 2008.

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

[BER06] Bertozzi, S.; Acquaviva, A.; Bertozzi, D.; Poggiali, A. "*Supporting Task Migration in Multi-Processor Systems-on-Chip: A Feasibility Study*". In: DATE06, 2006, 6p.

[BOR07] Borkar, S. "*Thousand core chips: a technology perspective*". In: DAC, 2007, pp. 746-749

[CAR07] *Carvalho, E.; Calazans, N.; Moraes, F. "Congestion-aware task mapping in NoC-based MPSoCs with dynamic workload*". In: ISVLSI, 2007, pp. 459-460.

[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 & Test of Computers, v. 27(5), pp. 26-35, 2010.

[CHE00] Chen, T.S. "*Task migration in 2D wormhole-routed mesh multicomputers*". Information Processing Letter, 2000, pp.103-110.

[CID11] Cidon, I. ; Manevich, R. ; Kolodny, A. ; Walter, I. ; Wimer, S. "*A Cost Effective Centralized Adaptive Routing for Networks-on-Chip*". In: Euromicro, 2011, pp. 39-46.

[FAT11] Fattah, Mohammad; Daneshtalab, Masoud; Liljeberg, Pasi; Plosila Juha. "Exploration of MPSoC Monitoring and Management Systems". In: ReCoSoC, 2011.

[FEI97] Feitelson D.; Rudolph L.; Schwiegelshohn U., "*Theory and practice in parallel job scheduling*". In: *Job Scheduling Strategies for Parallel Processing*, 1997, vol. 1291, pp. 1–34.

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

[HOW10] Howard, J; Dighe, S.; Hoskote, Y. "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS". IEEE International Solid-State Circuits Conference Digest of Technical Papers, 2010, pp.108-109.

[JAH11] Jahn, J.; Faruque, M. A. A. "*CARAT : Context-Aware Runtime Adaptive Task Migration for Multi Core Architectures*". In: DATE11, 2011., 6p.

[JAL10] Jalier, C.; Lattard, D.; Jerraya, A.A.; Sassatelli, G.; Benoit, P.; Torres, L. "*Heterogeneous vs Homogeneous MPSoC Approaches for a Mobile LTE Modem*". In: DATE10, 2010, 6p.

[JER05] Jerraya, A. A.; Wolf, W. "*Multiprocessor Systems-on-Chips*". In: Morgan Kaufmann Publishers Inc, 2005, 602p.

[KOB11] Kobbe, Sebastian; Bauer, Lars; Lohmann, Daniel; Schroder-Preikschat, Wolfgang; Henkel, Jorg. "*DistRM: Distributed Resorce Management for On-Chip Many-Core Systems*". In: ISSS11, 2011.

[LEE87] Lee, E.; Messerschmitt, D. G. "*Static scheduling of synchronous dataflow programs for digital signal processing*". IEEE Transactions on Computers, vol. 36, 1987, pp. 24-35

[MAN11] Mandelli, M.; Ost, L.; Carara, E.; Guindani, G.; Gouvea, T.; Medeiros, G.; Moraes, F. "*Energy-aware dynamic task mapping for NoC-based MPSoCs*". In: ISCAS, 2011, pp. 1676-1679.

[MAR10] Marczak, S. S.; Moraes, F. G. "*Monitoramento de Desempenho em Plataformas MPSoC Baseadas em NoC*". Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2010, 62p.

[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, 2004, vol. 38(1), pp. 69-93.

[NAY07] Nayebi, A.; Meraji, S.; Shamaei, A.; Sarbazi-Azad, H. "*XMulator: A Listener-Based Integrated Simulation Platform for Interconnection Networks*". In: AMS07, 2007.

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

[PUS09] Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. "*Adaptive energy-aware latency-constrained DVFS policy for MPSoC*". In: SOCC, 2009.

[RIC97] Richmond, M.; Hitchens, M. "*A New Process Migration Algorithm*". Operating Systems Review, vol. 31(1), 1997, pp. 31-42.

[SHA11] Shabbir, Ahsan; Kumar, Akash; Mesman, Bart; Corporaal, Henk. "*Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms*". In: SAMOS, 2011, pp. 132-139.

[TAN06] Tanurhan, Y. "*Processors and FPGAs Quo Vadis?*". Computer, 2006, v. 39(11), pp. 108-110.

[TAN08] Tan, J.; Zhang, L.; Fresse, V.; Legrand, A.; Houzet, D. "A predictive and parametrized architecture for image analysis algorithm implementations on FPGA adapted to

multispectral iaging". In: Int. Workshop on Image Processing Theory, Tools and Applications, 2008, pp 1-8.

[TIL11] Tilera Corporation, "Tile-GX Processor Family", 2011.

[VOE97] Voeten J. P. M.; van der Putten P. H. A., "*Specication of reactive hardware/software systems*". Ph.D. dissertation, Eindhoven University of Technology, 1997

[WOL04] Wolf, W. "*The Futere of Multiprocessors System-on-Chips*". 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, 2008, vol.27(10), pp. 1701-1713.

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