



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

# **Gerência Distribuída de Recursos em MPSoCs – Mapeamento e Migração de Tarefas**

Guilherme Machado de Castilhos

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: Prof. Dr. César Augusto Missio Marcon

Porto Alegre, Brasil

2013



C352g      Castilhos, Guilherme Machado de  
                 Gerência distribuída de recursos em MPSoCs – mapeamento e  
                 migração de tarefas / Guilherme Machado de Castilhos. – Porto Alegre,  
                 2013.  
                 66 f.

                 Diss. (Mestrado) – Fac. de Informática, PUCRS.  
                 Orientador: Prof. Dr. Fernando Gehm Moraes.  
                 Co-orientador: Prof. Dr. César Augusto Missio Marcon.

                 1. Informática. 2. Multiprocessadores. 3. Arquitetura de  
                 Computador. I. Moraes, Fernando Gehm. II. Marcon, César Augusto  
                 Missio. III. Título.

                 CDD 004.35

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

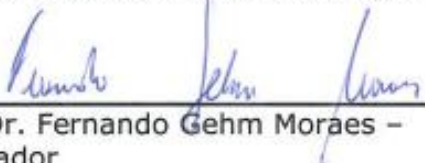





Pontifícia Universidade Católica do Rio Grande do Sul  
FACULDADE DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

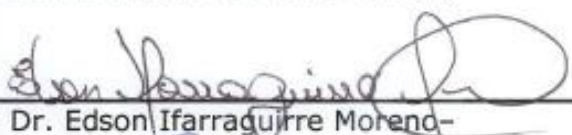
### TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Gerência Distribuída de Recursos em MPSoCs – Mapeamento e Migração de Tarefas" apresentada por Guilherme Machado de Castilhos como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 25/01/2013 pela Comissão Examinadora:

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

  
Prof. Dr. César Augusto Missio Marcon – PPGCC/PUCRS  
Coorientador

  
Prof. Dr. Alexandre de Moraes Amory – PPGCC/PUCRS

  
Prof. Dr. Edson Ifarraquirre Moreno – FACIN/PUCRS

  
Prof. Dr. Altamiro Amadeu Susin – UFRGS

Homologada em 07/03/2013, conforme Ata No. 003 pela Comissão Coordenadora.

  
Prof. Dr. Paulo Henrique Lemelle Fernandes  
Coordenador.

**PUCRS**

**Campus Central**

Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900

Fone: (51) 3320-3611 – Fax (51) 3320-3621

E-mail: [ppgcc@pucrs.br](mailto:ppgcc@pucrs.br)

[www.pucrs.br/facin/pos](http://www.pucrs.br/facin/pos)



## **AGRADECIMENTOS**

Gostaria de começar agradecendo minha família, ela é o centro da minha vida e me ajudou em todas as etapas que eu tive que passar até chegar aqui. Então, obrigado Pai, Mãe, Rapha e Gabi, amo muito vocês.

Outra pessoa importante na minha vida, que esteve ao meu lado durante todo o mestrado, que além de me ajudar e me cuidar, sobretudo me aturou durante todos os meus momentos ruins. Obrigado Karine, por ser minha companheira e por me ajudar tanto.

Tenho que agradecer a todas as pessoas do GAPH, mas principalmente ao Eduardo Wachter, Guilherme Madalozzo, Leandro Heck e Marcelo Mandelli, que me ajudaram muito no desenvolvimento do trabalho e também foram bons amigos.

Finalmente, gostaria de agradecer ao meu orientador por tudo que ele me ensinou e me ajudou, e pelo ótimo exemplo de pesquisador que ele me passou, que irei com certeza levar para toda a minha vida. Obrigado Moraes.

Gostaria de agradecer também, ao professor Marcon, Edson, Amory e Susin que aceitaram avaliar este trabalho.

# Gerência Distribuída de Recursos em MPSoCs – Mapeamento e Migração de Tarefas

## RESUMO

O projeto de MPSoCs é uma clara tendência na indústria de semicondutores. Os MPSoCs são capazes de executar várias aplicações em paralelo, suportando carga dinâmica de trabalho, ou seja, aplicações podem iniciar a qualquer momento. Outra característica importante em MPSoCs é QoS (qualidade de serviço), pois aplicações multimídia e de telecomunicações possuem requisitos estritos de desempenho, os quais devem ser respeitados pelo sistema. O crescimento constante do número de núcleos em MPSoCs implica em uma questão importante: escalabilidade.

Apesar da escalabilidade oferecida por NoCs, e o processamento distribuído permitindo a execução de carga dinâmica de trabalho, os recursos dos MPSoCs devem ser gerenciados para proporcionar o desempenho esperado. Tarefas de gerenciamento incluem acesso de entrada/saída a dispositivos externos ao MPSoC, mapeamento de tarefas, migração de tarefas, monitoramento, DVFS, dentre outras. Um único elemento de processamento (PE), responsável pela gerência dos recursos pode se tornar um gargalo no desempenho do sistema, 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-spots*). Uma alternativa para garantir escalabilidade é descentralizar ou distribuir as funções de gerenciamento do sistema.

Duas abordagens principais de gerência são discutidas na literatura: um gerente por aplicação, ou um gerente por região do MPSoC. A segunda abordagem é preferível, já que o número de recursos utilizados no gerenciamento permanece constante, independentemente do número de aplicações em execução na MPSoC. As regiões são definidas como *clusters*. Todas as tarefas das aplicações são executadas em um *cluster*, se possível. Em relação ao tamanho do *cluster*, ele pode ter seu tamanho modificável em tempo de execução para permitir o mapeamento de aplicações com um número de tarefas maior do que seus recursos disponíveis.

Este trabalho propõe uma gerência distribuída de recursos em MPSoCs, utilizando um método de clusterização, permitindo que o tamanho do *cluster* seja modificado dinamicamente. Esse sistema inicializa cada *cluster* com um tamanho fixo, e durante a execução das aplicações, os *clusters* podem requerer recursos a seus *clusters* vizinhos para mapear tarefas. Os testes foram executados utilizando a plataforma HeMPS, e foram comparados o desempenho do método de gerência centralizado contra o método de gerência distribuído. Os resultados mostram uma importante redução no tempo total de execução das aplicações e no número de *hops* entre as tarefas (menor energia de comunicação) utilizando o método de gerência distribuída. Os resultados também avaliam o método de reclusterização, utilizando monitoração de desempenho e migração de tarefas.

**Palavras-chave:** MPSoC, NoC, Gerência Distribuída, Mapeamento, Migração de Tarefas.



# Distributed Resource Management in MPSoCs – Task Mapping and Task Migration

## ABSTRACT

The design of MPSoCs is a clear trend in the semiconductor industry. Such MPSoCs are able to execute several applications in parallel, with support to dynamic workload, i.e., applications may start at any moment. Another important feature is QoS (quality of service), because multimedia and telecom applications have tight performance requirements that must be respected by the system. The constantly growth in the number of cores in MPSoCs implies in an important issue: scalability.

Despite the scalability offered by NoCs and distributed processing, the MPSoC resources must be managed to deliver the expected performance. Management tasks include access to input/output devices, task mapping, task migration, monitoring, DVFS. One processing element (PE) responsible for resource management may become a bottleneck, since this PE will serve all other PEs of the system, increasing its computation load and creating a communication hot-spot region. An alternative to ensure scalability is to decentralize or distribute the management functions of the system.

Two main approaches are discussed in the literature: one manager per application, and one manager per MPSoC region. The second approach is preferable, since the number of management resources remains constant, regardless the number of applications executing in the MPSoC. The regions are defined as clusters. All application tasks are executed in the cluster, if possible. Regarding the size of the cluster, it may have its size modifiable at runtime to allow the mapping of applications with a greater number of tasks than their available resources.

This work proposes a distributed resource management in NoC-based MPSoCs, using a clustering method, enabling the modification of the cluster size at runtime. At system start-up each cluster has a fixed size, and at runtime clusters may borrow resources from neighbor clusters to map applications. Results are evaluated using the HeMPS MPSoC, comparing the performance of the centralized versus distributed management approaches. Results show an important reduction in the total execution time of applications, and a reduced number of hops between tasks (smaller communication energy). Results also evaluate the reclustering method, through monitoring and task migration.

**Keywords:** MPSoC, NoC, Distributed Management, Mapping, Task Migration.

## LISTA DE FIGURAS

Figura 1 – <i>Speedup</i> médio das aplicações para diversos tamanhos de sistemas. ....	18
Figura 2 – Diagrama funcional do gerente de recursos centralizado e distribuído.....	19
Figura 3 – Grafos das tarefas e seus mapeamentos no sistema.....	20
Figura 4 – Exemplo de inserção de aplicação em tempo de execução. ....	20
Figura 5 – Hierarquia de gerenciamento do MPSoC.....	21
Figura 6 – MPSoC baseado em NoC (a), e conteúdo da célula (PE).....	21
Figura 7 – Fluxo do método ADAM.....	22
Figura 8 – Cenário de teste com apenas um <i>cluster</i> comparando o método proposto por [ALF08] com os demais analisados.....	23
Figura 9 – Cenário de teste com diversos <i>clusters</i> comparando o método proposto por [ALF08] com os demais analisados.....	24
Figura 10 – Fluxo do mapeamento descentralizado proposto por [CUI12]. ....	25
Figura 11 – Os três passos da reclusterização. Uma aplicação que contém 10 tarefas é mapeada em um <i>cluster</i> com apenas 9 recursos disponíveis [CUI12]. ....	25
Figura 12 – Comparação do volume de comunicação [CUI12].....	26
Figura 13 – Método proposto por [ANA12]. ....	27
Figura 14 – Comparação entre tempos de mapeamento de [ANA12] e outros métodos. ....	27
Figura 15 – Exemplo do método de mapeamento descentralizado proposto por [WEI11]. Topologia da aplicação (a) e Passos do mapeamento (b). ....	28
Figura 16 – Comparação do método de mapeamento proposto por [WEI11] com diferentes tipos de monitoramento.....	28
Figura 17 – Fases do processo de migração proposta por [GOO11]. ....	29
Figura 18 – Resultados para migração de tarefas proposta por [GOO11] em uma NoC 16x16.....	30
Figura 19 – Visão estrutural da plataforma usada em [ALM10].....	31
Figura 20 – Visão geral do protocolo de migração. ....	31
Figura 21 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09]. ....	34
Figura 22 – Exemplo de aplicação modelada por um grafo de tarefas. ....	35
Figura 23 – Protocolo de comunicação adotado no MPSoC HeMPS.....	36
Figura 24 – Grafo da aplicação Produtor-Consumidor (a) e mapeamento desta em uma rede 2x3 (b). ....	37
Figura 25 – Diagrama de sequência de troca de mensagens entre processadores. ....	38
Figura 26 – Abordagem clusterizada para uma gerência distribuída de recursos. ....	40
Figura 27 – Protocolo para inserir uma nova aplicação no sistema. ....	42
Figura 28 – Exemplo de descrição de uma tarefa inicial, com o comando <i>send</i> . ....	43
Figura 29 – Protocolo de mapeamento de tarefa.....	43
Figura 30 – Mapeamento Centralizado (a) vs Mapeamento Distribuído (b).....	44
Figura 31 – Protocolo para mapeamento de tarefas em um cluster vizinho. SPs Brancos são PEs com recursos disponíveis. ....	46
Figura 32 – Exemplo de troca de mensagens durante a reclusterização. ....	47
Figura 33 – Exemplo de reclusterização usando migração de tarefas. ....	48

Figura 34 – Protocolo de migração de tarefas.....	49
Figura 35 – Mapeamento distribuído (barras cinzas) versus centralizado (barras brancas), para o benchmark Multispec com três tamanhos de NoC.....	55
Figura 36 – Mapeamento Distribuído (barras cinzas) versus Centralizado (barras brancas), para o Benchmark MPEG para dois tamanhos de MPSoC.....	55
Figura 37 – Grafo da aplicação e o número de hops entre cada para comunicante (a), e a aplicação Synthetic mapeada no MPSoC (b).....	56
Figura 38 – Mapeamento de uma instância da aplicação MPEG em um MPSoC 6x6.....	58
Figura 39 – Exemplo de um cenário com reclusterização.....	58
Figura 40 – Tempo de iteração da Tarefa F, sem migração de tarefa.....	59
Figura 41 – Tempo de iteração da Tarefa F, com migração de tarefa.....	60

## LISTA DE TABELAS

Tabela 1 – Estado-da-Arte comparado com o trabalho proposto.....	32
Tabela 2 – Tempo de simulação (em segundos), considerando vários níveis de abstração do MPSoC HeMPS.....	51
Tabela 3 – Tempos de execução normalizados em relação a um MPSoC 12x12 com gerência centralizada e uma carga igual a 75%.....	52
Tabela 4 – Tempos de execução normalizados em relação a um MPSoC 6x6 com gerência centralizada e uma carga igual a 75%.....	53
Tabela 5 – Comparação do tempo de execução e do número de hops entre MPSoC de tamanho 6x6, 8x8 e 12x12, com carga de 75%. ....	53
Tabela 6 – Tempo total de execução (em ciclos de relógio) para um MPSoC 12x12, com carga de 1% e 75% e overhead no tempo total de execução em relação ao aumento da carga. ....	54
Tabela 7 – Número de hops entre tarefas em um MPSoC 12x12 com carga igual a 75%.....	57
Tabela 8 – Número médio de hops entre tarefas em MPSoCs com carga igual a 25%.....	57

## LISTA DE SIGLAS

SoC .....	System-on-Chip
PE .....	Processing Element
MPSoC .....	Multiprocessor System-on-Chip
DVFS .....	Distributed Voltage and Frequency Scaling
NoC .....	Network-on-Chip
SDF .....	Synchronous Data Flow
POOSL .....	Parallel Object-Oriented Specification Language
JPEG .....	Joint Photographic Experts Group
TGFF .....	Task graphs for free
VIP .....	Virtual Point-to-Point
RTOS .....	Real-time Operating System
NPU .....	Network Processing Unit
VHDL .....	VHSIC Hardware Description Language
VHSIC .....	Very-High-Speed Integrated Circuit
MJPEG .....	Motion JPEG
RISC .....	Reduced Instruction Set Computing
MIPS .....	Microprocessor without Interlocked Pipeline Stages
UART .....	Universal Asynchronous Receiver/Transmitter
RAM .....	Random Access Memory
NI .....	Network Interface
DMA .....	Direct memory access
LEC-DN .....	Low Energy Consumption - Dependences Neighborhood
GMP .....	Global Master Processor
LMP .....	Local Master Processor
SP .....	Slave Processor
ID <sub>i</sub> .....	Identificador de tarefa
ADD <sub>i</sub> .....	Endereço do código objeto da tarefa no repositório
SIZE <sub>i</sub> .....	Tamanho do código objeto da tarefa
RTL .....	Register-transfer level
ISS .....	Instruction Set Simulators
QoS .....	Quality of Service

# SUMÁRIO

<b>1. Introdução.....</b>	<b>15</b>
1.1. Objetivos .....	16
1.2. Contribuições.....	16
1.3. Estrutura do Documento .....	16
<b>2. Estado-da-Arte .....</b>	<b>17</b>
2.1. Controle Distribuído.....	17
2.2. Mapeamento Distribuído .....	22
2.3. Migração de Tarefas.....	29
2.4. Considerações Finais .....	32
<b>3. Plataforma de Referência – MPSoC HeMPS .....</b>	<b>33</b>
3.1. Arquitetura HeMPS (Hermes Multiprocessor System) .....	33
3.2. Comunicação entre Tarefas .....	35
<b>4. Gerência Distribuída de Recursos .....</b>	<b>39</b>
4.1. Arquitetura com Controle Distribuído de Recursos.....	39
4.2. Mapeamento Distribuído .....	41
<b>5. Gerência com Regiões Dinamicamente Adaptadas às Aplicações.....</b>	<b>45</b>
5.1. Aumento Dinâmico do Tamanho dos Clusters .....	45
5.2. Migração .....	47
<b>6. Resultados.....</b>	<b>51</b>
6.1. Tempo Total de Execução .....	52
6.2. Mapeamento Distribuído de Tarefas.....	54
6.3. Número de Hops .....	56
6.4. Reclusterização .....	58
<b>7. Conclusões e Trabalhos Futuros .....</b>	<b>61</b>
7.1. Publicações Relacionadas ao Tema da Dissertação .....	61
7.2. Trabalhos Futuros .....	62
<b>Referências .....</b>	<b>63</b>

## 1. INTRODUÇÃO

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

Muitas aplicações requerem SoCs com vários processadores 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 estão presentes em várias aplicações comerciais, 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 PE.

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 o processamento distribuído, permitindo a 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 [MAN11a], 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 no desempenho do sistema, já que este PE vai servir todos PEs do sistema, aumentando sua carga de trabalho e criando regiões com congestionamento de tráfego (*hot-spot*). Uma alternativa para garantir à escalabilidade é descentralizar ou distribuir as funções de gerenciamento do sistema.

As atuais tendências apontam para projetos de MPSoCs com dezenas de PEs. O Intel SCC [HOW10] e a família de processadores Tiler TILE-Gx [TIL11] são exemplos oriundos da indústria. Ambas as arquiteturas possuem um grande conjunto de núcleos que estão ligados através de uma rede intrachip (NoC). O Intel SCC possui 48 núcleos e o Tiler 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 irão 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, e são definidos a seguir.

Objetivos estratégicos:

- Domínio da tecnologia de projeto de sistemas multiprocessados em chip (MPSoC);
- Domínio dos mecanismos de comunicação e controle em sistemas operacionais (*microkernel*) em MPSoCs;
- Domínio das técnicas de gerência distribuída de recursos em MPSoC;
- Domínio de técnicas de mapeamento distribuído de tarefas e de migração de tarefas em MPSoCs.

Objetivos específicos:

- Definir qual das técnicas de gerência distribuída de recursos é mais eficiente e quais as vantagens e limitações de cada uma;
- Alterar o MPSoC HeMPS para que ele passe de uma gerência centralizada de recursos para uma gerência distribuída;
- Definir uma política de desfragmentação das aplicações no MPSoC;
- Avaliação de desempenho do MPSoC desenvolvido. Comparação de desempenho entre MPSoCs com gerência centralizado (HeMPS) e gerência distribuída.

## 1.2. Contribuições

O presente trabalho de Dissertação tem por contribuição principal a definição e implementação de um MPSoC com gerência distribuída de recursos. Esta arquitetura permite estudar quando a opção por gerenciamento centralizado ou distribuído é recomendada e, no caso de gerenciamento distribuído, qual a melhor estratégia a ser adotada.

Outra importante contribuição é a definição do processo de teste das arquiteturas através de testes de regressão. Estes testes permitem avaliar de forma automatizada dezenas de casos de testes, permitindo a verificação da corretude das implementações assim como a obtenção automatizada de resultados.

## 1.3. Estrutura do Documento

O Capítulo 2 apresenta o estado da arte em gerência distribuída de recursos em MPSoCs, migração de tarefas e mapeamento distribuído, apresentando em seu final uma tabela comparativa entre os trabalhos revisados e o trabalhado proposto por esta Dissertação. O Capítulo 3 descreve a plataforma HeMPS, utilizada como MPSoC de referência. O Capítulo 4 apresenta o desenvolvimento do mecanismo de gerência distribuída de recursos. O Capítulo 5 apresenta o mecanismo de reclusterização proposto por este trabalho. O Capítulo 6 apresenta resultados experimentais, fazendo uma comparação entre gerência centralizada e distribuída. O Capítulo 7 apresenta as conclusões desta Dissertação, publicações e trabalhos futuros.



## 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 são técnicas utilizadas na presente Dissertação.

### 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 princípio de multiagentes para gerenciar os recursos, no qual cada agente gerencia uma aplicação.

Cada agente tem por objetivo 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 número de núcleos atribuídos dinamicamente.

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

Como o agente não conhece o estado global do sistema, ele tenta alocar de forma aleatória 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 espaço 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 PEs 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, capaz de simular configurações arbitrárias de um sistema multiprocessado. A métrica usada para comparar o método proposto a um método centralizado foi a carga total de trabalho de todas as aplicações, dividido 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 inferior comparado ao método centralizado usando o *speedup* como métrica. De acordo com os Autores, à medida que o número de PEs aumenta, a diferença entre o mapeamento distribuído usando a técnica aleatória aproxima-se ao mapeamento centralizado utilizando uma heurística com visão completa do sistema. Esta desvantagem deve-se à escolha aleatória dos PEs a receberem tarefas e ao elevado número de migração de tarefas. Embora os resultados favoreçam o método centralizado, a proposta do método DistRM é capaz de gerenciar recursos em um sistema com dezenas de PEs, com *speedup* próximo ao do método centralizado. A diferença tende a reduzir-se,

segundo os Autores, com heurísticas melhores de mapeamento e restrições no número de migrações de tarefas.

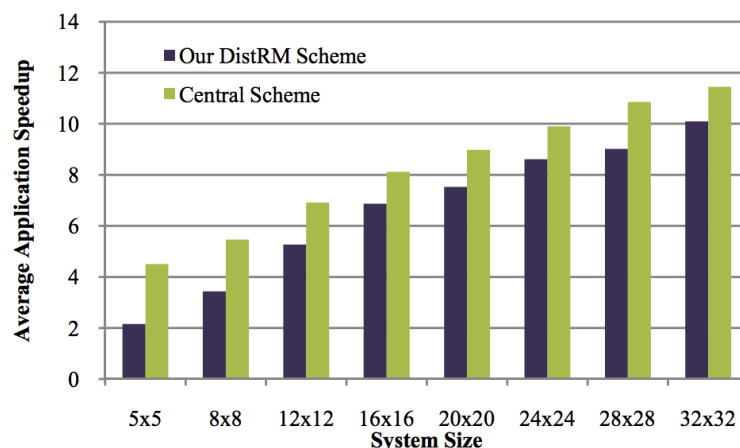


Figura 1 – *Speedup* médio das aplicações para diversos tamanhos de sistemas.

### 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, os quais são escaláveis tanto em número de aplicações quanto em número de processadores e comparados com um gerenciador de recursos centralizado.

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

O segundo gerenciador de recursos proposto chama-se *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 das 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 utilizadas foram especificadas como *Synchronous Data Flow (SDFs)* [LEE87], onde os vértices correspondem a tarefas (também chamados de atores) de uma aplicação e as arestas são as dependências entre as mesmas. *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 por processadores interconectados 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 satisfeitas.

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 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). As restrições de cada aplicação são calculadas centralmente, mas eles são aplicados em todos os processadores localmente. O gerente distribuído não monitora cada aplicação, eliminando o problema de escalabilidade devido ao monitoramento centralizado.

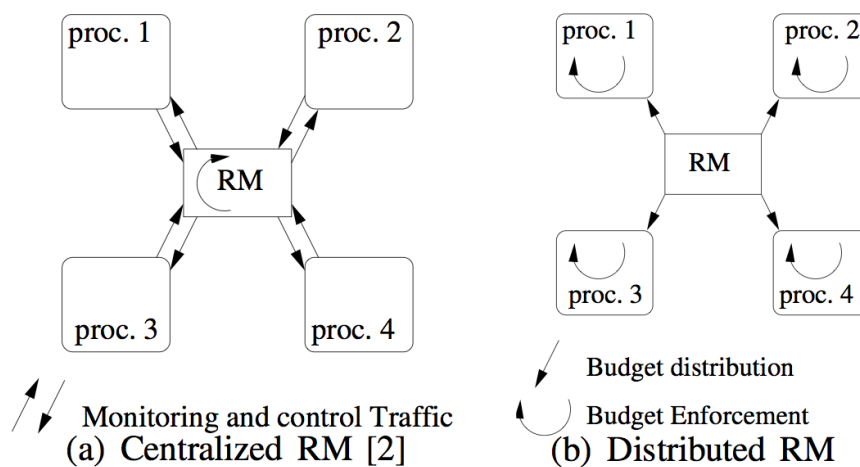


Figura 2 – Diagrama funcional do gerente de recursos centralizado e distribuído.

Os modelos dos gerentes de recursos distribuídos usados para a simulação foram desenvolvidos usando a linguagem POOSL [VOE97]. Foram usados dois modelos de aplicações para as simulações: decodificador JPEG (6 atores) e decodificador H.263 [COT98] (4 atores), mostrados na Figura 3. A plataforma de computação é composta 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 ocorre porque durante o período de monitoramento as aplicações podem estar inativas.

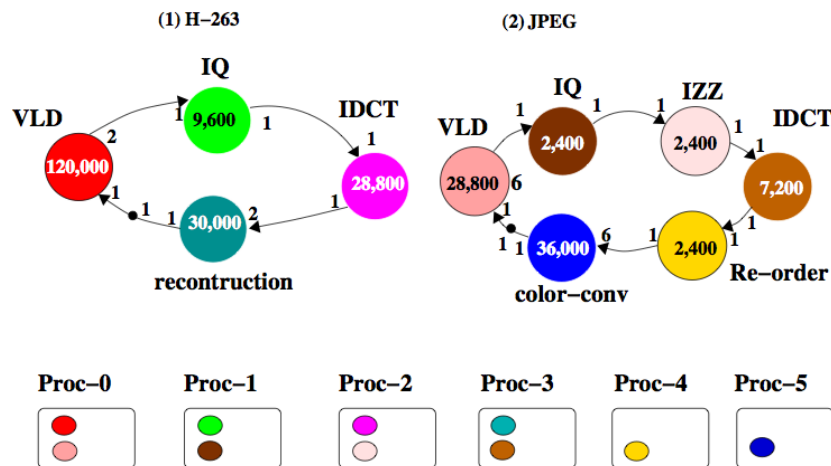


Figura 3 – Grafos das tarefas e seus mapeamentos no sistema.

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 instância do decodificador H.263 inicia sua execução. Isto é devido ao fato de que quando a segunda instância do H.263 inicia, 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.

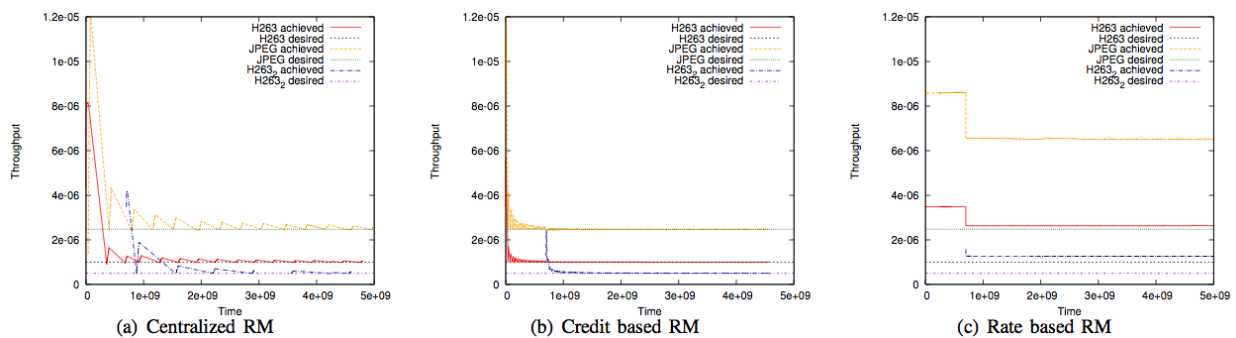


Figura 4 – Exemplo de inserção de aplicação em tempo de execução.

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 número reduzido de processadores (6 processadores), implementação em 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 com gerenciamento hierárquico, a fim de beneficiar os esquemas distribuídos e centralizados, como mostrado na Figura 5. A gerência do sistema é de granularidade fina em alguns aspectos, embora alguns dados monitorados e relatórios de desempenho sejam enviados para um gerente de nível superior que envia de volta comandos a serem executados.

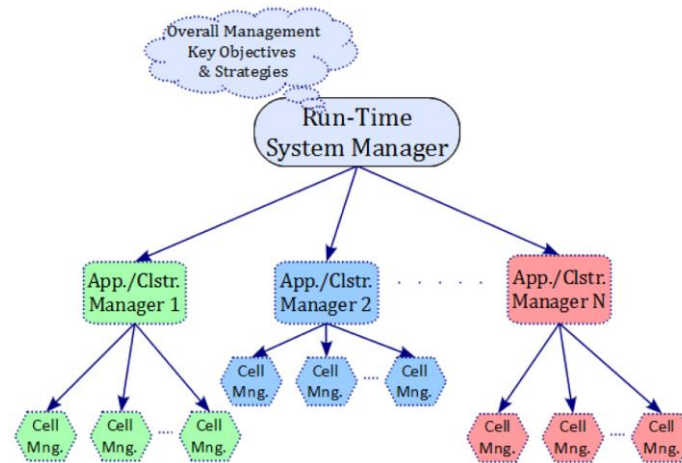


Figura 5 – Hierarquia de gerenciamento do MPSoC.

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, juntamente com atuadores, tais como reguladores de tensão e frequência. Cada célula tem seu próprio gerente, no qual reporta sua condição e executa comandos recebidos de um gerenciador de mais alto nível hierárquico.

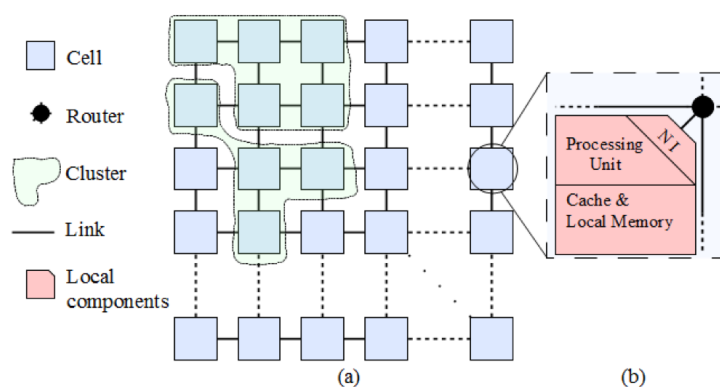


Figura 6 – MPSoC baseado em NoC (a), e conteúdo da célula (PE).

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.

As vantagens de sistemas de gestão centralizada e distribuída são obtidas através da exploração de gestores inteligentes de células, através de um gerente de nível médio para minimizar os dados monitorados que chegam aos gestores de nível superior.

Segundo os 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, desenvolvida no grupo de pesquisa GAPH e utilizada na presente Dissertação.

## 2.2. Mapeamento Distribuído

### 2.2.1. ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication [ALF08]

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

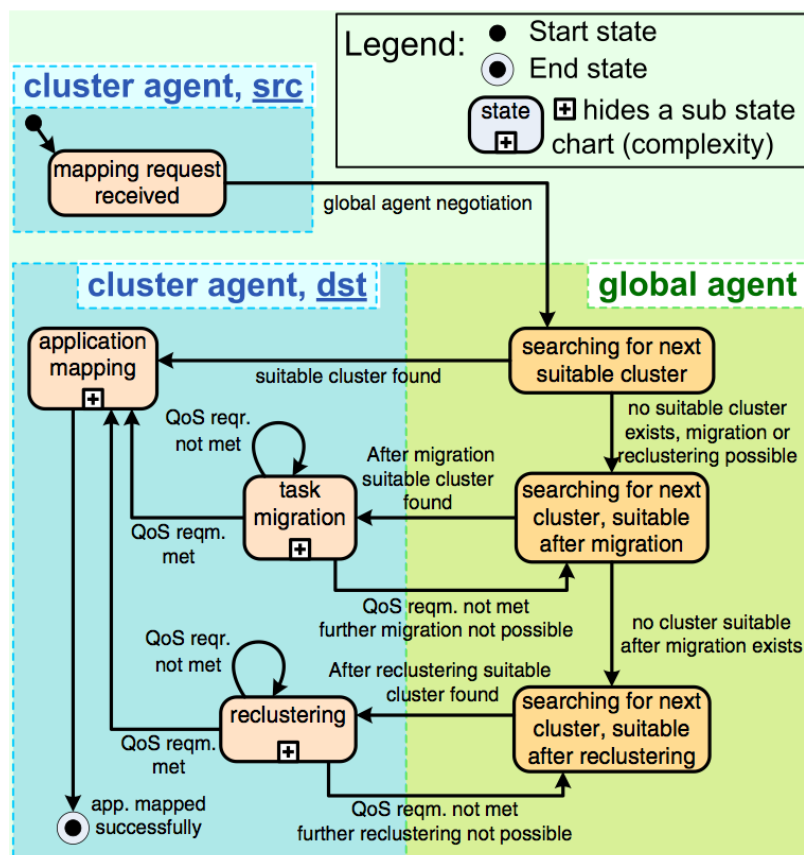


Figura 7 – Fluxo do método ADAM.

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 o CA do *cluster* origem que recebe todos os pedidos de mapeamento e negocia com o GA. Podem haver várias instâncias de GA que são sincronizadas ao longo do tempo. O GA tem informações globais sobre todos os clusters do MPSoC, 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 nenhum 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 Figura 8, o método ADAM só se torna mais eficiente em redes com mais de 4096 PEs, com menos processadores a heurística NN [CAR10] se faz mais eficiente.

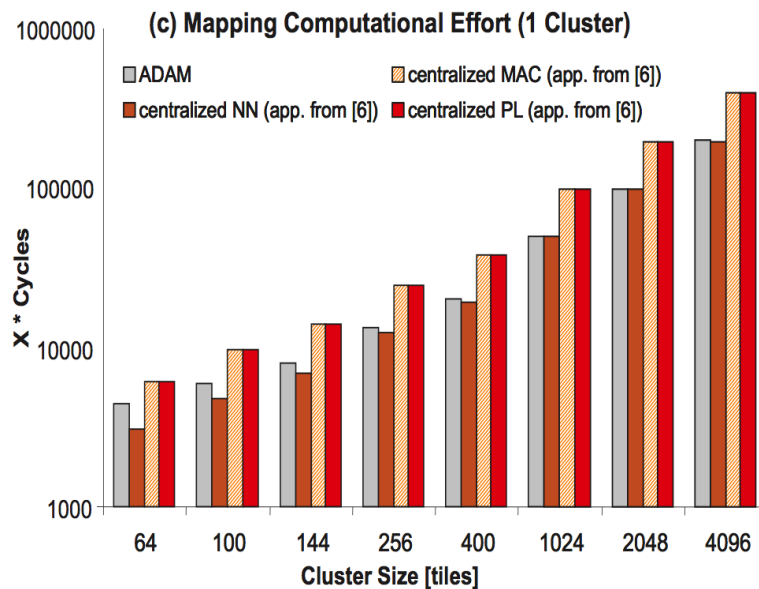


Figura 8 – Cenário de teste com apenas um *cluster* comparando o método proposto por [ALF08] com os demais analisados.

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.

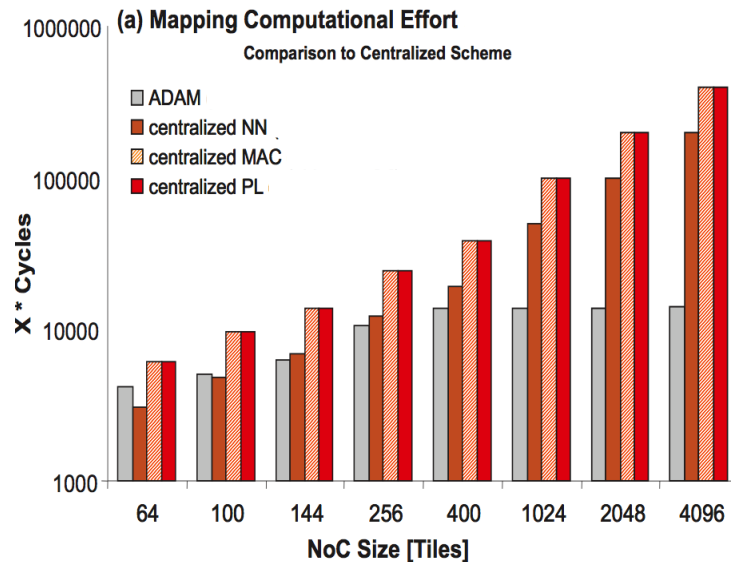


Figura 9 – Cenário de teste com diversos *clusters* comparando o método proposto por [ALF08] com os demais analisados.

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. É importante também destacar que o trabalho não detalha como um sistema com 4096 PEs foi modelado.

### 2.2.2. Decentralized Agent Based Re-clustering for Task Mapping of Tera-scale Network-on-Chip System [CUI12]

O trabalho apresentado por [CUI12] é uma evolução do método proposto por [ALF08]. É apresentado um método de mapeamento de tarefas descentralizado com reclusterização. O fluxo da proposta de mapeamento é mostrado na Figura 10. O mapeamento de tarefa é realizado pelo agente global e pelo agente local. Entretanto, diferentemente do método proposto por [ALF08], todas as informações dos recursos são armazenados nos *clusters*. O método de reclusterização é controlado localmente pelos *clusters*. O agente global não precisa armazenar as informações sobre recursos disponíveis como em [ALF08].

Após o agente global selecionar o cluster para uma nova aplicação, o grafo de comunicação das tarefas da aplicação é enviado para o agente local do *cluster* para realizar o mapeamento das tarefas. Se o agente local detectar que o número de recursos livres dentro do *cluster* não é o suficiente para alocar a aplicação, o agente local executa automaticamente o processo de reclusterização.



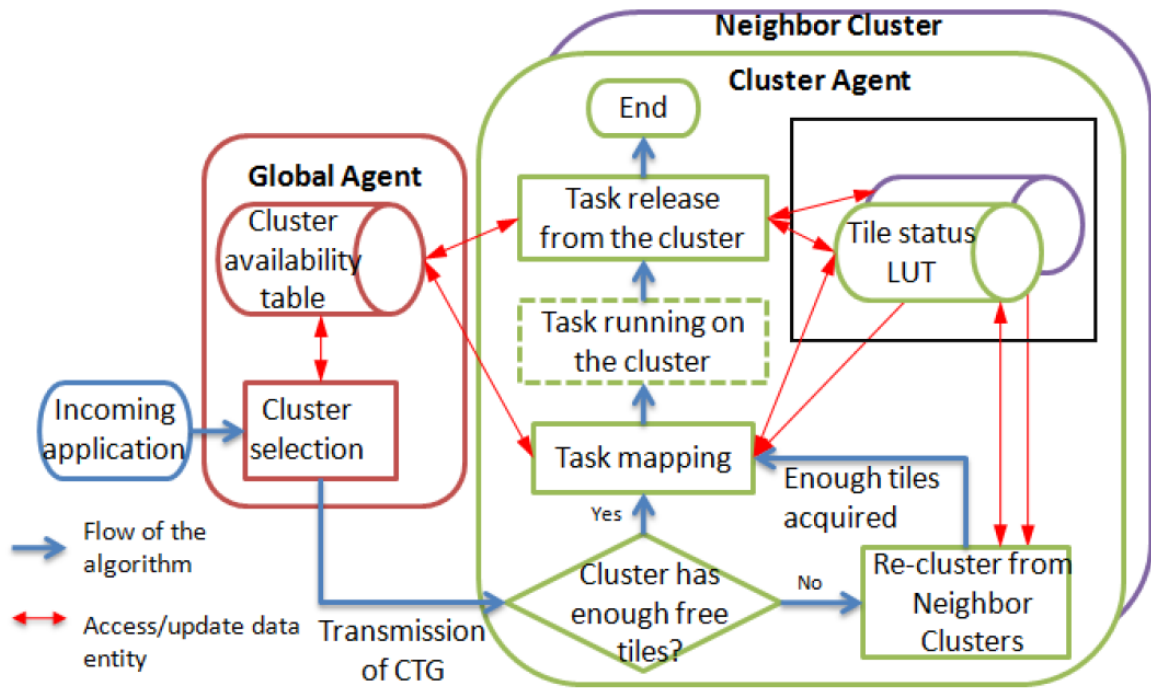


Figura 10 – Fluxo do mapeamento descentralizado proposto por [CUI12].

Os três principais passos da reclusterização são mostrados na Figura 11 e detalhados a seguir:

- Primeiro passo: *Requisição de recursos livres* – o cluster que requer mais recursos para executar o mapeamento de tarefas, envia mensagens de requisição para os agentes locais dos clusters vizinhos.
- Segundo passo: *Mapear tarefas* – as tarefas são mapeadas tanto nos recursos originais do cluster, quanto nos recursos obtidos na reclusterização.
- Terceiro passo: *Retornar ao tamanho original* – depois que a aplicação terminar sua execução, os recursos emprestados voltam ao cluster original.

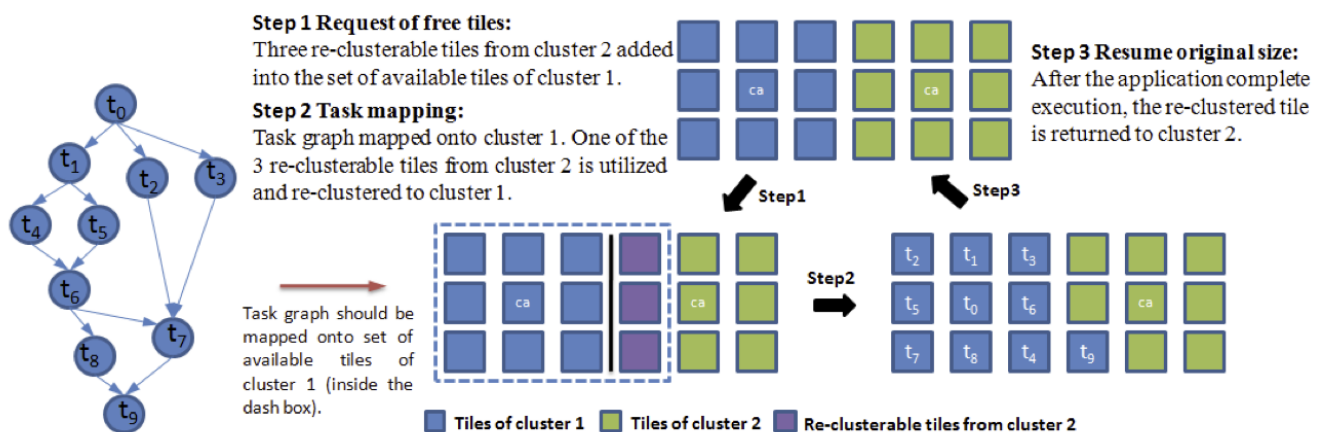


Figura 11 – Os três passos da reclusterização. Uma aplicação que contém 10 tarefas é mapeada em um cluster com apenas 9 recursos disponíveis [CUI12].

Para demonstrar as vantagens da proposta apresentada por [CUI12], foram executados aplicações cujos grafos de tarefas foram gerados randomicamente (TGFF [DIC98]). Os resultados de tráfego de monitoramento, volume de comunicação e consumo de energia são comparados com mapeamento centralizado e o método proposto por [ALF08].

Na Figura 12 foi comparado o volume de comunicação do mapeamento das aplicações em diferentes tamanhos de NoC. O volume de comunicação foi reduzido em 20,9% para uma NoC de tamanho 64x64.

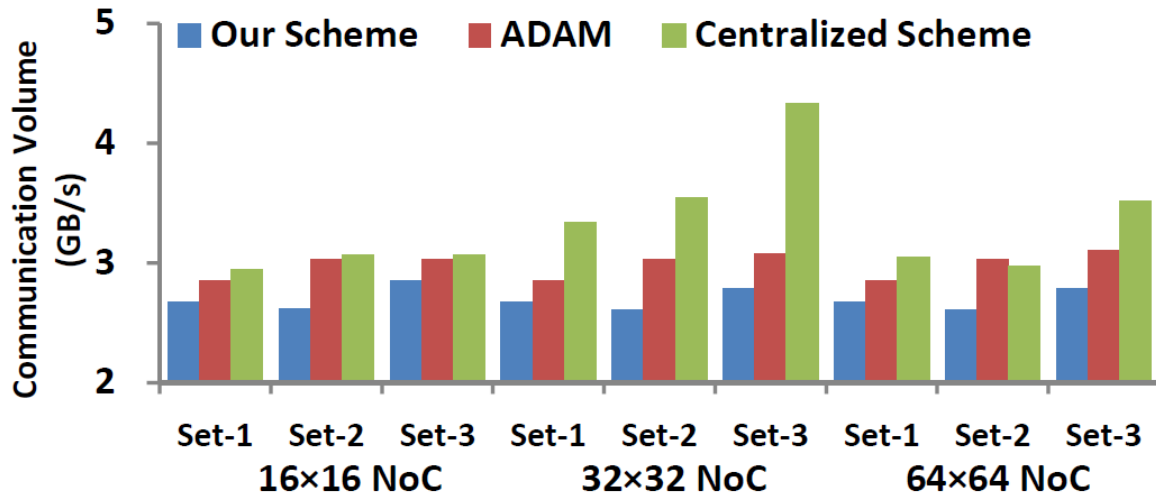


Figura 12 – Comparação do volume de comunicação [CUI12].

O trabalho, assim como [ALF08], não detalha como o sistema foi modelado e nem o seu nível de abstração, além de não demonstrar os resultados de tempo de execução das aplicações em comparação aos outros métodos.

### 2.2.3. A Divide and Conquer based Distributed Run-time Mapping Methodology for Many-Core platforms [ANA12]

Em [ANA12], foi proposto um método de dividir para conquistar para a execução de mapeamento distribuído em tempo de execução.

O método proposto pelos Autores é mostrado na Figura 13. Quando uma nova aplicação é requerida para ser executada no sistema, o método proposto divide a rede em *clusters*, usando como critério o tamanho da aplicação. A região que a aplicação melhor se adaptar é escolhida. Depois que a região do *cluster* é definida, um PE dentro do *cluster* é transformado em gerente do *cluster* para executar o algoritmo de mapeamento de tarefas.

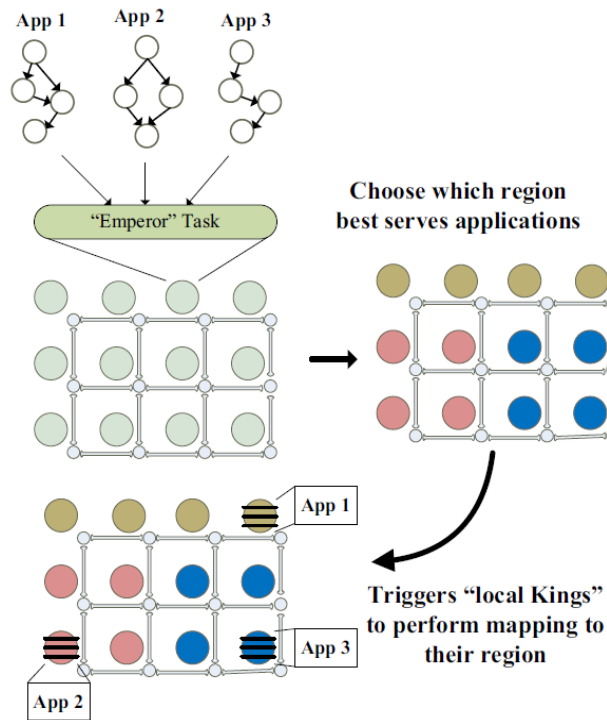


Figura 13 – Método proposto por [ANA12].

Foram executados diversos cenários de aplicações no método proposto por [ANA12], no método [ALF08] e em um sistema com mapeamento em tempo de projeto, sendo assim comparados os seus resultados de tempo de mapeamento em ciclos de relógio, tal comparação é mostrada na Figura 14. O método proposto por [ANA12] teve um ganho significativo no tempo de mapeamento em relação ao mapeamento em tempo de projeto, mas teve um tempo maior em quase todos os cenários comparando com o método proposto por [ALF08]. Segundo os Autores, extensões futuras do trabalho incluem heurística de mapeamento multi-objetivo e controle de mapeamento hierárquico.

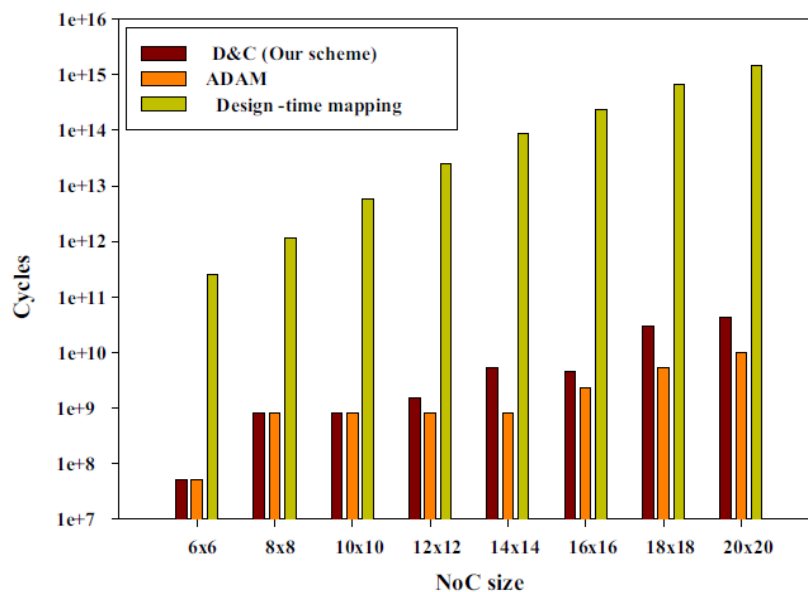


Figura 14 – Comparação entre tempos de mapeamento de [ANA12] e outros métodos.

### 2.2.4. Dynamic Decentralized Mapping of Tree-Structured Applications on NoC Architectures [WEI11]

[WEI11] propôs um método de mapeamento descentralizado visando reduzir o congestionamento da NoC. O método de mapeamento proposto considera apenas uma visão local dos nós adjacentes da NoC para executar o mapeamento, onde cada tarefa contém uma lista de suas tarefas seguintes a serem mapeadas. O método é limitado a aplicações com uma topologia em árvore. A raiz (tarefa inicial) é mapeada inicialmente e depois cada tarefa executa a heurística de mapeamento para mapear as tarefas que lhe sucederam. Esse método é mostrado na Figura 15.

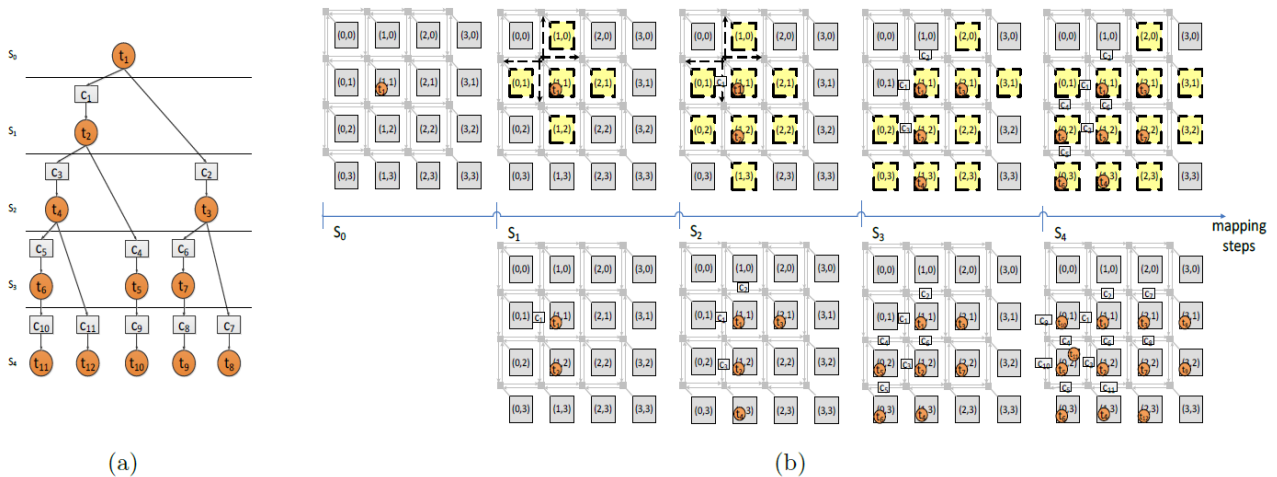


Figura 15 – Exemplo do método de mapeamento descentralizado proposto por [WEI11]. Topologia da aplicação (a) e Passos do mapeamento (b).

Para avaliar o algoritmo proposto, foi implementado um simulador de NoC em *Java*, e usadas aplicações com três tipos de classes de árvores: árvore binária, *quadtrees* e cadeias. Na Figura 16, é mostrada a quantidade de mensagens de monitoramento e a média de carga da rede com o algoritmo utilizando somente os seus vizinhos ou toda a rede como espaço de busca para o mapeamento das tarefas. Como se pode observar, quando se usa toda a rede como espaço de busca, há um grande aumento no número de mensagens de monitoramento em relação a um espaço de busca menor, mas a média de carga da rede produzida é similar.

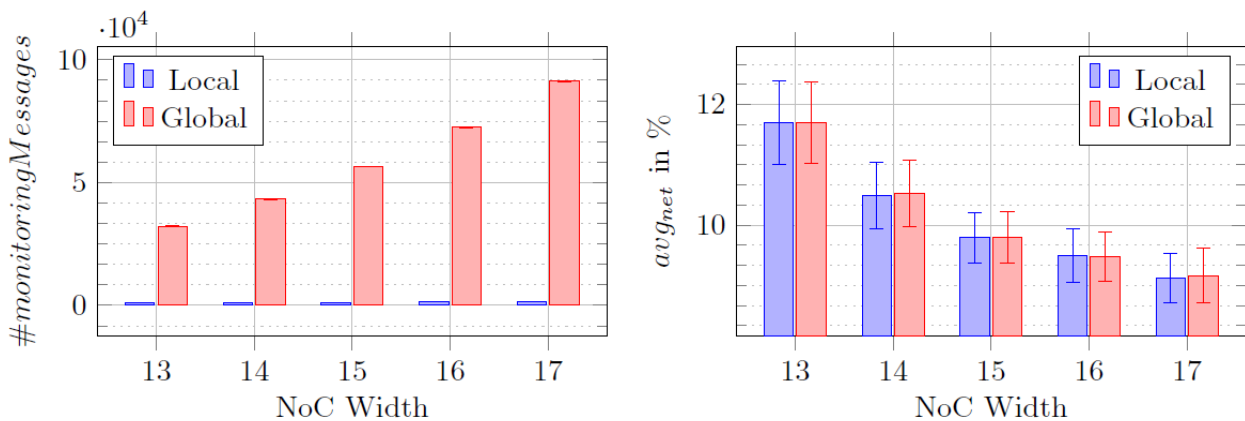


Figura 16 – Comparação do método de mapeamento proposto por [WEI11] com diferentes tipos de monitoramento.

Os resultados mostrados em [WEI11], se limitaram a comparações de variações do método proposto, não fazendo comparações com outros métodos, como por exemplo, o mapeamento centralizado de tarefas.

## 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.

A Figura 17 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.

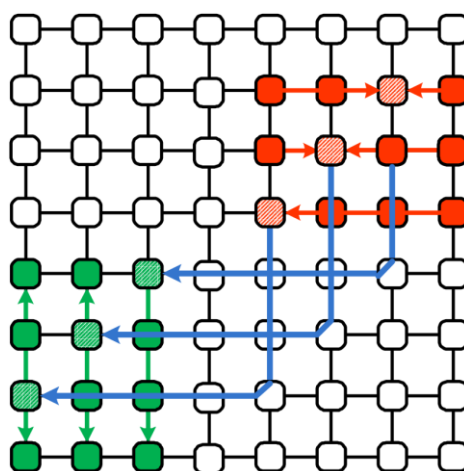


Figura 17 – Fases do processo de migração proposta por [GOO11].

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 18 mostra os resultados da migração de tarefas 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*.

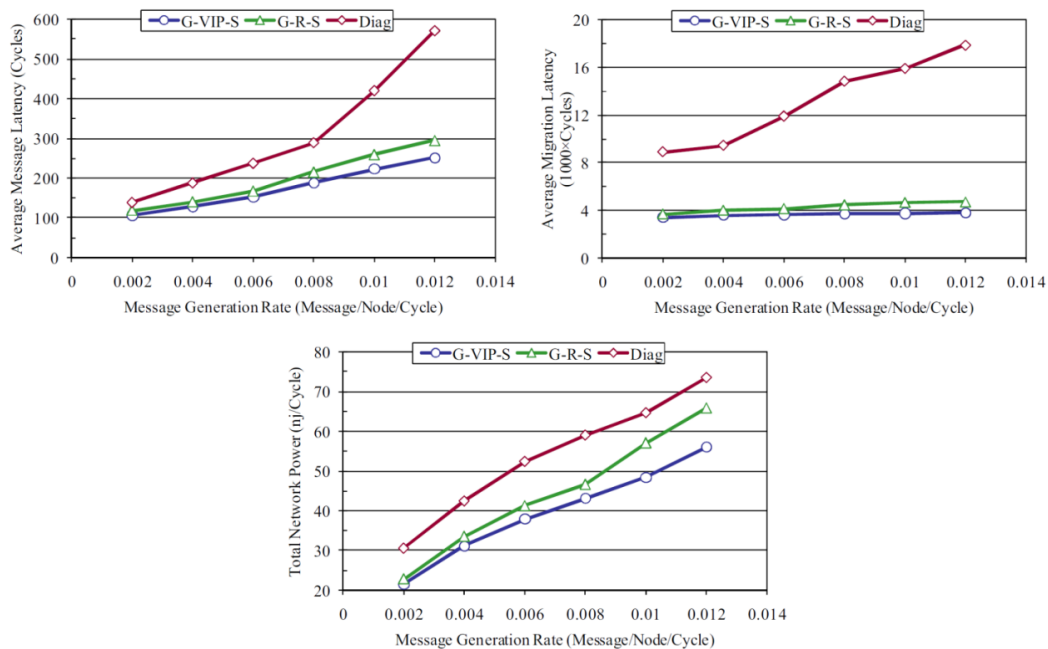


Figura 18 – Resultados para migração de tarefas proposta por [GOO11] em uma NoC 16x16.

### 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 [MOR04]. Esta plataforma é muito semelhante

à plataforma HeMPS, por utilizar a NoC Hermes e o processador *Plasma*. A Figura 19 mostra uma visão estrutural desta plataforma.

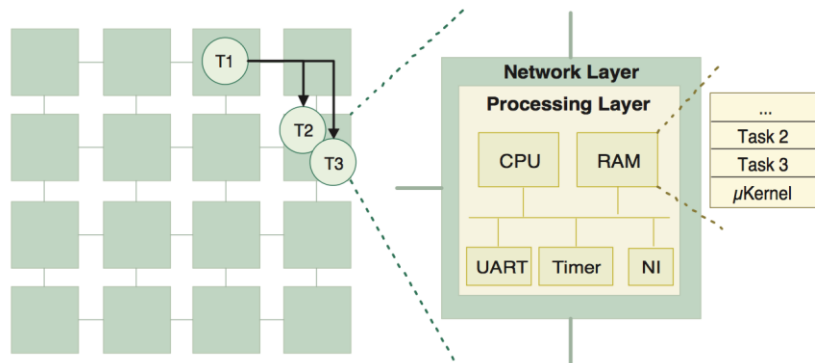


Figura 19 – Visão estrutural da plataforma usada em [ALM10].

O protocolo de migração de tarefas, ilustrado na Figura 20, 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.

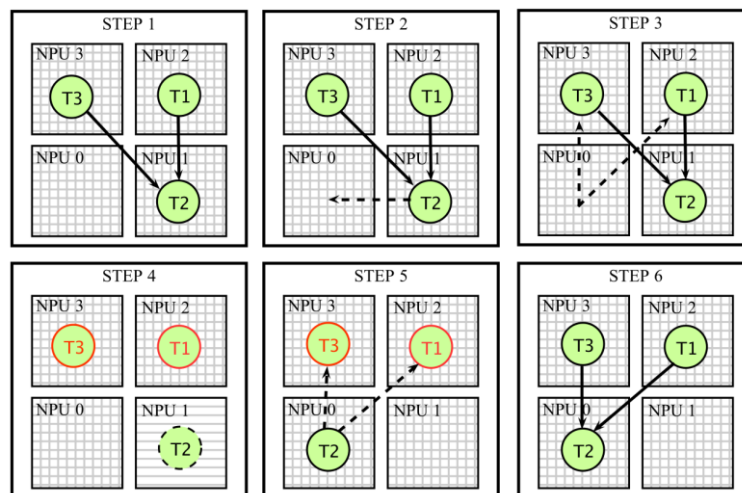


Figura 20 – Visão geral do protocolo de migração.

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 centralizadas, outros quatro trabalhos ([ALF08], [CUI12], [ANA12] e [WEI11]) exploram mapeamento distribuído de tarefas em tempo de execução, visando evitar que um único agente tenha a função de mapear tarefas, e os dois últimos trabalhos ([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].

As lacunas identificadas nos trabalhos revisados em gerência e mapeamento distribuído incluem: (1) ausência de informações relacionadas à arquitetura utilizada; (2) modelagem abstrata do sistema não permitindo avaliação detalhada do desempenho, ou mesmo ausência de informações relacionadas à modelagem. Já nos trabalhos revisados de migração de tarefas, nenhum detalhou o processo de migração de tarefas, assumindo uma modelagem abstrata para o mesmo, além de nenhum método possuir uma migração de tarefa que faça a migração de contexto.

A Tabela 1 posiciona o trabalho da presente Dissertação em relação ao estado da arte. A gerência de recursos será distribuída, havendo um gerente por *cluster* (terceira coluna). O capítulo 4 discute em mais detalhes as opções de gerência: por aplicação, por *cluster*, por PE. O sistema proposto é subdividido em regiões (*clusters*) onde seus tamanhos são dinamicamente adaptados em função das aplicações (capítulo 5). Há também um monitoramento de tarefas visando a desfragmentação do sistema, utilizando como mecanismo a migração de tarefas (seção 5.2). O sistema foi implementado usando uma modelagem precisa no nível de ciclo de relógio.

Tabela 1 – Estado-da-Arte comparado com o trabalho proposto.

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



### 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 foi modificada a fim de passar de uma plataforma com controle centralizado, executado por um único PE (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.

As características relevantes do MPSoC HeMPS incluem:

- *Processamento homogêneo*. Todos os PEs possuem a mesma arquitetura. Justifica-se o emprego de um MPSoC homogêneo para simplificar a geração dos códigos objetos e a migração de tarefas. Um MPSoC heterogêneo implica em diferentes códigos objetos para a mesma aplicação, ou tradução dinâmica de código [OST12]. Dados os objetivos apresentados na Seção 1.1, o emprego de MPSoCs heterogêneos não pertence ao escopo da presente Dissertação. Estudo relativo a MPSoCs heterogêneos foi o tema abordado da Dissertação “*Integração de Novos Processadores em Arquiteturas MPSoC: Um Estudo de Caso*” [WAC11].
- *Utilização de rede intrachip (NoC)*. Justifica-se o emprego desta arquitetura de comunicação dada a escalabilidade da mesma e a possibilidade de múltiplas comunicações entre PEs ocorrerem simultaneamente [PAS08].
- *Memória distribuída*. Cada processador possui uma memória privada, responsável por armazenar instruções e dados. Optou-se por esta arquitetura, pois a mesma reduz o tráfego de dados no meio de comunicação, e não requer memórias cache externas [WOL12].
- *Comunicação por troca de mensagens*. Justifica-se o emprego de troca de mensagens, pois este é o mecanismo natural de comunicação em um sistema com memórias distribuídas. A utilização de memória compartilhada é mais adequada a sistemas que utilizam barramento, com poucos PEs [PAT11].
- *Organização de memória paginada*. Este mecanismo possui as seguintes desvantagens, comparado à organização de memória segmentada: restrição do tamanho máximo das tarefas ao tamanho da página, desperdício de memória caso as tarefas possuam tamanho menor que a página (fragmentação interna). Porém esta escolha simplifica tanto o processo de mapeamento quanto o de migração de tarefas, pois uma página de memória é vista como um recurso de processamento de tarefas, podendo a tarefa ser mapeada em qualquer página livre. Não são necessários mecanismos complexos de gerência de memória.

#### 3.1. Arquitetura HeMPS (Hermes Multiprocessor System)

Os principais componentes da arquitetura HeMPS são os elementos de processamento, denominados Plasma-IP, que são interconectados pela NoC Hermes [MOR04], utilizada como infraestrutura de comunicação. Além disso, tem-se uma memória externa, denominada de repositório de tarefas. Na Figura 21 é ilustrada uma instância da arquitetura HeMPS, utilizando

uma NoC Hermes de dimensão 2x3 interconectando os Plasmas-IP.

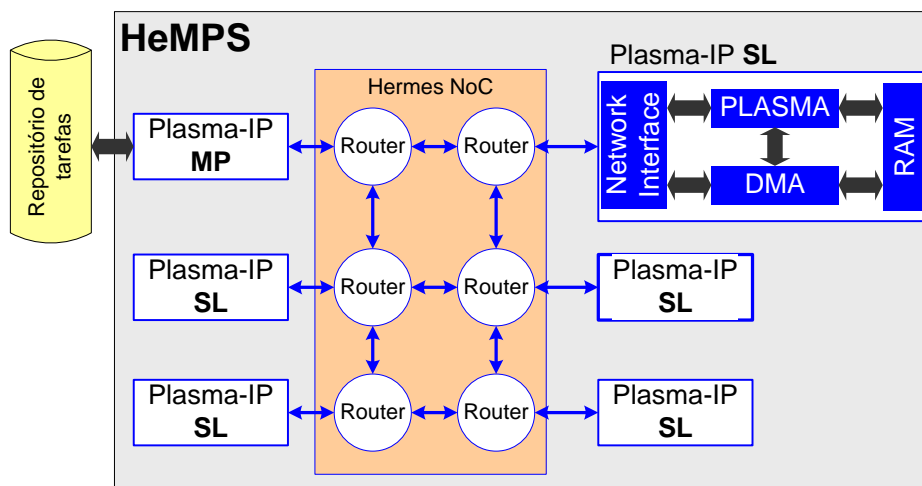


Figura 21 – Instância da HeMPS utilizando uma NoC 2x3 [CAR09].

### 3.1.1. Plasma-IP

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

- processador Plasma [PLA11]: é um processador RISC de 32 bits com subconjunto de instruções da arquitetura MIPS. O processador Plasma do MPSoC HeMPS possui algumas modificações em relação ao processador Plasma original, como por exemplo, a criação do mecanismo de interrupção, exclusão de módulos (UART) e inclusão de novos registradores mapeados em memória.
- memória privada(RAM): contém o sistema operacional, denominado *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 (NI): realiza a interface entre o Plasma e a NoC Hermes. É responsável pelo envio e recebimento de pacotes na rede.
- módulo de acesso direto à memória (DMA): possibilita o processador continuar a execução de tarefas sem controlar diretamente a troca de mensagens com a rede. O DMA (do inglês, *Direct Memory Access*) tem como principal função transferir o código-objeto de tarefas que chegam à interface de rede para a memória do processador e enviar para o processador 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. O algoritmo de roteamento utilizado é o XY, que envia os pacotes na rede primeiramente horizontalmente até chegar à coordenada X do roteador destino, e depois percorre verticalmente, até encontrar o roteador destino.

### 3.1.3. Repositório de Tarefas

O MPSoC HeMPS assume que todas as aplicações são modeladas através de um grafo de tarefas, onde os vértices representam tarefas e as arestas representam a comunicações entre as tarefas da aplicação. As tarefas sem dependências de recepção de dados são denominadas *iniciais* (tarefa A na Figura 22). No momento de inicialização de uma dada aplicação, a heurística de mapeamento carrega no MPSoC somente as tarefas iniciais. As demais tarefas são inseridas dinamicamente no sistema em função das requisições de comunicação e dos recursos disponíveis. A Figura 22 mostra um exemplo de uma aplicação modelada de acordo com esta abordagem.

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

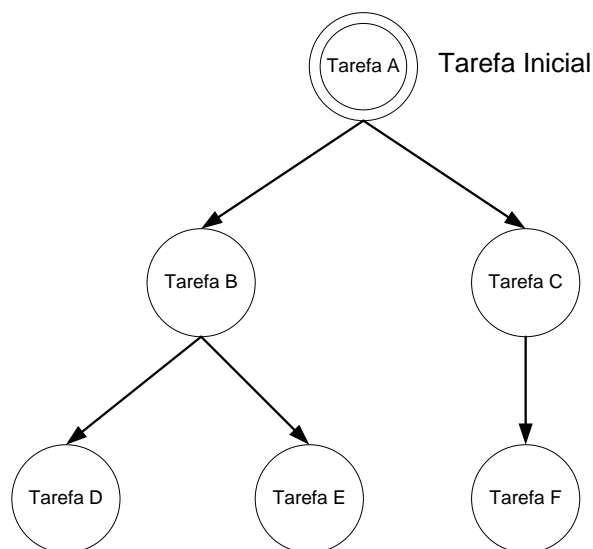


Figura 22 – Exemplo de aplicação modelada por um grafo de tarefas.

## 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,

alocado no *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 na mesma ordem.

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

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

A Figura 23 ilustra a comunicação entre duas tarefas, A e B, mapeadas em diferentes PEs. Quando a tarefa A executa um *Send()* (1), a mensagem é armazenada no *pipe* e a execução da tarefa A continua (2). Isso caracteriza uma escrita assíncrona não-bloqueante. Quando a tarefa B executa um *Receive()* (3), duas situações podem ocorrer. Se a tarefa destino está mapeada no mesmo processador, a tarefa executa uma leitura no *pipe* local. Se a tarefa está mapeada em outro PE, como nesse exemplo, o *microkernel* envia uma requisição de mensagem através da NoC (4) e a tarefa entra em estado de espera (estado *waiting*) (5), caracterizando uma leitura síncrona bloqueante. A tarefa A recebe a requisição de mensagem (6) e envia a mensagem através da NoC, liberando espaço no *pipe* (6). Quando a mensagem chega no PE da tarefa B (7), o *microkernel* armazena a mensagem no espaço de memória da tarefa B, habilitando a execução da tarefa B e mudando o seu estado (estado *ready*) (8).

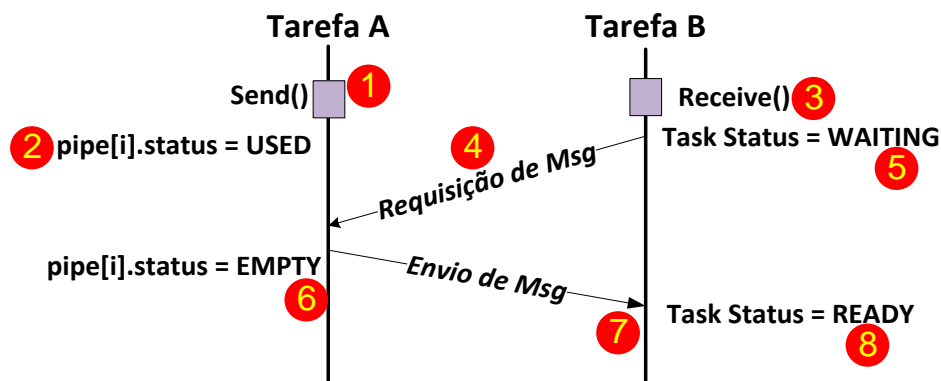


Figura 23 – Protocolo de comunicação adotado no MPSoC HeMPS.

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.

Os serviços que o *microkernel* pode tratar incluem:

- MESSAGE\_REQUEST: é enviado para realizar a requisição de uma mensagem a uma tarefa que está alocada em outro PE do sistema (direção Escravo → Escravo).

- MESSAGE\_DELIVERY: contém a mensagem a ser entregue 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 o funcionamento de trocas de mensagens e os respectivos serviços, foi utilizada uma rede de tamanho 2x3, com uma aplicação produtor-consumidor (Figura 24(a)). Ambas as tarefas são alocadas estaticamente, nos processadores 02 e 12. A topologia dessa rede é mostrada na Figura 24. A Tarefa A executa somente a primitiva *Receive()* para a Tarefa B, e a Tarefa B executa a primitiva *Send()* para a Tarefa A.

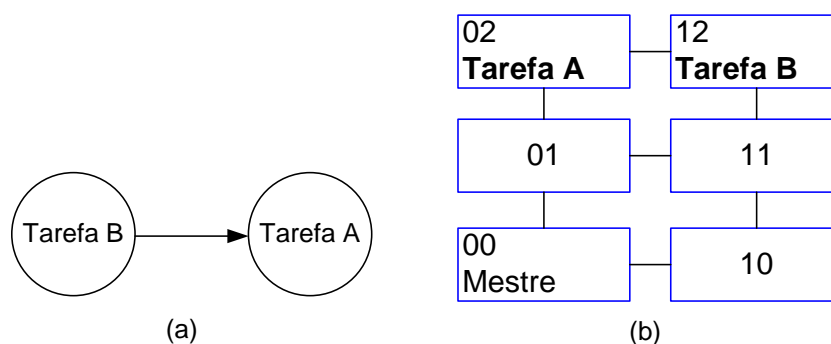


Figura 24 – Grafo da aplicação Produtor-Consumidor (a) e mapeamento desta em uma rede 2x3 (b).

No diagrama de sequência da Figura 25, pode-se ver a troca de mensagens entre os processadores. Neste diagrama, 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 contendo o serviço "TASK\_ALLOCATION". Após as tarefas serem alocadas, assume-se que a Tarefa A executa um *Receive()* para a Tarefa B. Como ainda não houve comunicação entre as tarefas, o *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".

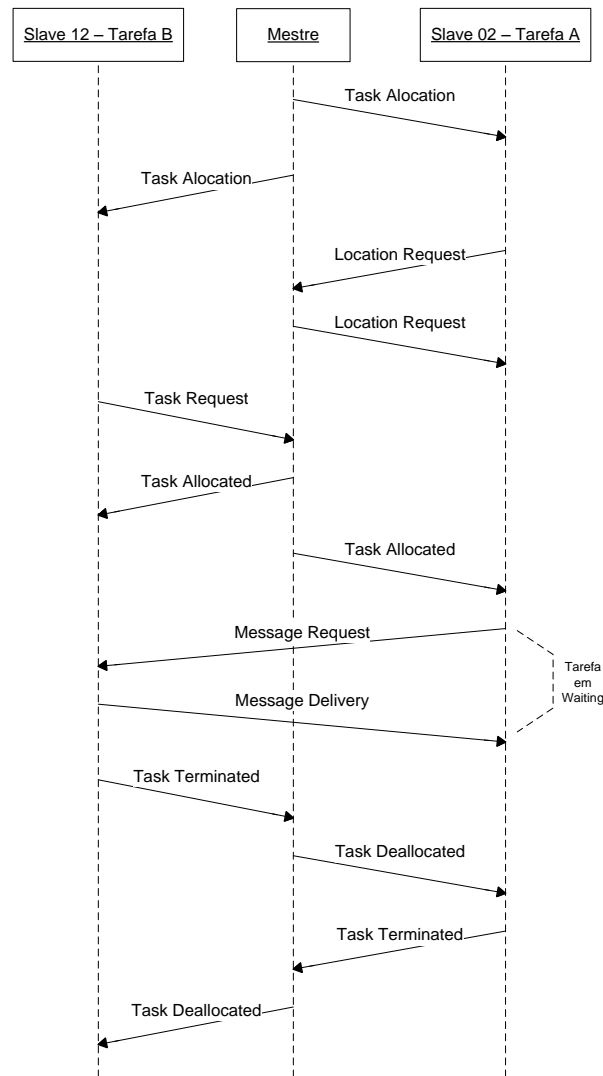


Figura 25 – Diagrama de sequência de troca de mensagens entre processadores.

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

Na sequência, a Tarefa A envia um "MESSAGE\_REQUEST" para a Tarefa B, requisitando dados. Enquanto a Tarefa A não receber a resposta da mensagem, esta tarefa fica bloqueada no estado *Waiting*. Se a Tarefa B já fez um *Send()* para Tarefa A, ele envia a resposta 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.

## 4. GERÊNCIA DISTRIBUÍDA DE RECURSOS

Este Capítulo apresenta a primeira contribuição da Dissertação: o mecanismo de gerência distribuída de recursos, publicada em [MAN12].

A revisão do estado-da-arte (Capítulo 2) identificou 4 abordagens distintas para realizar a gerência distribuída de recursos:

- I. Sistema dividido em *clusters*, onde cada *cluster* é controlado por um gerente. Os *clusters* podem ser responsáveis por mais que uma aplicação e eles também podem ser reorganizados, modificando seus tamanhos em tempo de execução através de um método de reclusterização [ALF08] [CUI12].
- II. O sistema é dividido em *clusters* baseados no tamanho das aplicações. Um *cluster* contém somente uma aplicação [ANA12].
- III. Um gerente de aplicação é usado como na segunda abordagem, mas esse método não usa *clusters* com tamanho predefinidos, permitindo assim o espalhamento de aplicações em PEs não contínuos [KOB11] [SHA11].
- IV. Uma função global do sistema é implementada em cada PE do sistema, tal como a heurística de mapeamento de tarefas [WEI11].

O presente Capítulo detalha o processo de mapeamento distribuído, utilizando *clusters* de tamanho fixo. O foco de Capítulo é o detalhamento do processo no nível de transações para tornar o mapeamento distribuído. A heurística de mapeamento utilizada, LEC-DN, não será detalhada, pois a mesma foi tema da Dissertação "*Mapeamento Dinâmico de Aplicações para MPSoCs Homogêneos*" [MAN11b].

### 4.1. Arquitetura com Controle Distribuído de Recursos

A arquitetura com controle distribuído proposta dividiu o MPSoC em  $n$  *clusters* de tamanhos iguais, definidos em tempo de projeto. Em tempo de execução, se uma dada aplicação não couber no *cluster* por falta de recursos disponíveis, o *cluster* pode pedir recursos emprestados aos seus *clusters* adjacentes, sendo o processo denominado reclusterização. Tal abordagem é preferível às demais abordagens pois:

- I. O número de PEs dedicados à função de gerência é limitado ao número de *clusters*. Uma abordagem com um gerente por aplicação pode implicar em uma maior sobrecarga, já que o número de aplicações que irão executar no sistema é desconhecido em tempo de execução.
- II. A abordagem clusterizada reduz o número de *hops* entre tarefas pertencentes a uma mesma aplicação, reduzindo o tráfego global da NoC.
- III. Não é necessário criar/destruir agentes toda a vez que uma nova aplicação entra/deixa o sistema, aumentando desse modo o desempenho global do sistema.

A Figura 26 mostra um MPSoC 9x9 contendo nove *clusters* de tamanho 3x3. O MPSoC contém três tipos de PEs: o Mestre Global (GMP), Mestre Local (LMP) e Escravo (SP). O LMP é responsável pelo controle do *cluster*, executando funções como o mapeamento de tarefas, migração de tarefas, monitoramento, verificação de *deadlines* e comunicação entre outros LMPs e

GMP. O GMP contém todas as funções do LMP, e as funções relacionadas com a gerência global do sistema. Um exemplo dessas funções inclui a escolha de qual *cluster* uma dada aplicação irá ser mapeada, o controle de recursos disponíveis em cada *cluster*, o recebimento de mensagens de controle e de *debug* dos LMPs, acesso ao repositório de aplicações (memória externa contendo os códigos objetos de todas as tarefas) e recebimento de requisições de novas aplicações de uma interface externa. Os SPs são responsáveis pela execução das tarefas.

Observar que a memória externa, anteriormente denominada “*repositório de tarefas*”, denomina-se na arquitetura com controle distribuído de recursos de “*repositório de aplicações*”. Esta alteração deve-se ao fato que a granularidade do mapeamento mudou de tarefa para aplicação. Anteriormente, no controle centralizado, o processador mestre recebia requisições de mapeamento de tarefas. Na arquitetura com controle distribuído de recursos, aplicações são inseridas no sistema, e o GMP envia o descritor das novas aplicações a um determinado LMP, o qual inicia o mapeamento das tarefas.

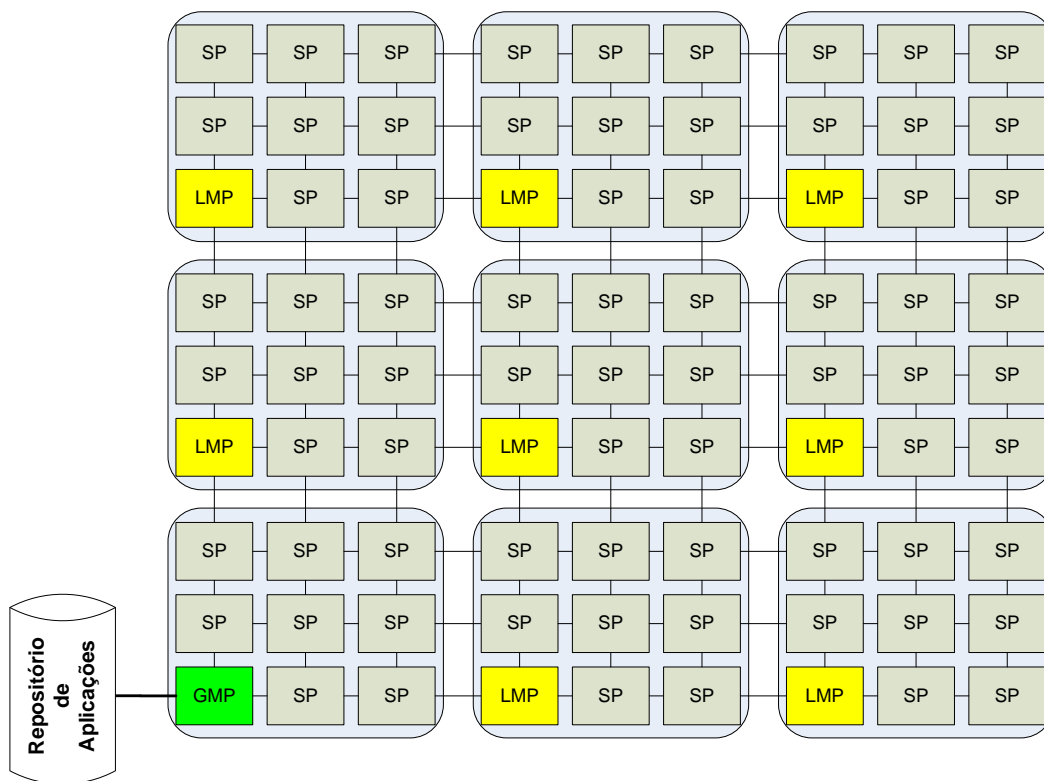


Figura 26 – Abordagem clusterizada para uma gerência distribuída de recursos.

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

O GMP é o único PE com acesso a dispositivos externos, como o repositório de aplicações. Na Figura 26, nove PEs são reservados para as funções de gerência, representando 11,1% dos PEs sem executar aplicações de usuário. Usando *clusters* de tamanho 4x4 em um MPSoC com tamanho de 16x16, esse número cai para 6,25%, o qual é um custo aceitável, considerando os benefícios obtidos, como será demonstrado na seção de resultados.



Dado que o mapeamento de aplicações é a primeira ação a ser executada pelo gerente do MPSoC quando uma dada aplicação entra no sistema, a próxima seção detalha a abordagem do mapeamento distribuído.

## 4.2. Mapeamento Distribuído

Essa seção apresentação o processo de mapeamento distribuído de tarefas. A seção 4.2.1 detalha o protocolo relacionado à seleção de *clusters* quando uma nova aplicação entra no sistema, detalhando a interação entre GMP → LMP. A seção 4.2.2 detalha o processo de mapeamento *intraclusters*, detalhando a interação entre LMP → SP. A seção 4.2.3 compara o processo de mapeamento distribuído contra o centralizado.

### 4.2.1. Inserção de uma nova aplicação

Como já mencionado, as aplicações são modeladas como grafos de tarefas. Isso supõe que pelo menos uma tarefa não possui dependência com outras tarefas, sendo essas chamadas de *tarefas iniciais*. As aplicações são armazenadas no *repositório de aplicações*. De acordo com as requisições do usuário, uma nova aplicação pode ser requisitada para ser executada no sistema. No exemplo apresentado na Figura 27, essa ação é representada como a seta “Nova Aplicação”, entrando no GMP.

O GMP lê do repositório de aplicações a *descrição da aplicação* e executa a heurística de “*Seleção de Cluster*” para escolher qual *cluster* irá receber a nova aplicação. A heurística escolhe o *cluster* em que a aplicação melhor se encaixa em termos de recursos disponíveis. O número de recursos de um dado *cluster* é definido de acordo com a equação 1.

$$\text{recursos do cluster} = (PE-1) * \text{páginas} \quad (1)$$

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

O *cluster* escolhido é aquele com o número de recursos livres igual ou superior ao número de tarefas da aplicação. Não havendo um *cluster* que atenda a esta condição, é escolhido o *cluster* com o maior número de recursos livres, desde que haja um número igual ou superior de recursos livres no MPSoC correspondente ao número de tarefas da aplicação. Essa condição garante que todas as aplicações que entrarem no sistema, tenham recursos livres para que suas tarefas sejam mapeadas. Caso uma aplicação tenha um número de tarefas superior ao número total de recursos livres do MPSoC, essa aplicação só poderá ser inserida no sistema quando o número total de recursos livres do MPSoC for igual ou superior ao seu número de tarefas.

Após a execução da heurística de “*Seleção de Cluster*”, o GMP transmite a descrição da aplicação para o LMP do *cluster* selecionado. A *descrição da aplicação* contém: (i) identificador de aplicação,  $ID_{app}$ , juntamente com sua lista de tarefas; (ii) identificação de cada tarefa que pertence a aplicação (equação 2).

$$\text{tarefa}_i = \{\text{inicial}_i, ID_i, ADD_i, SIZE_i, \{(t_1, ct_1); (t_2, ct_2); (t_n, ct_n)\}\} \quad (2)$$

onde:  $\text{inicial}_i$ , define se a tarefa é inicial ou não;  $ID_i$ , identificador de tarefa;  $ADD_i$ , endereço do código objeto da tarefa no *repositório de aplicações*;  $SIZE_i$ , tamanho do código objeto da tarefa;  $\{(t_1, ct_1); (t_2, ct_2); (t_n, ct_n)\}$ , lista de tarefas que se comunicam com a tarefa<sub>i</sub> e seus respectivos volumes

de comunicação.

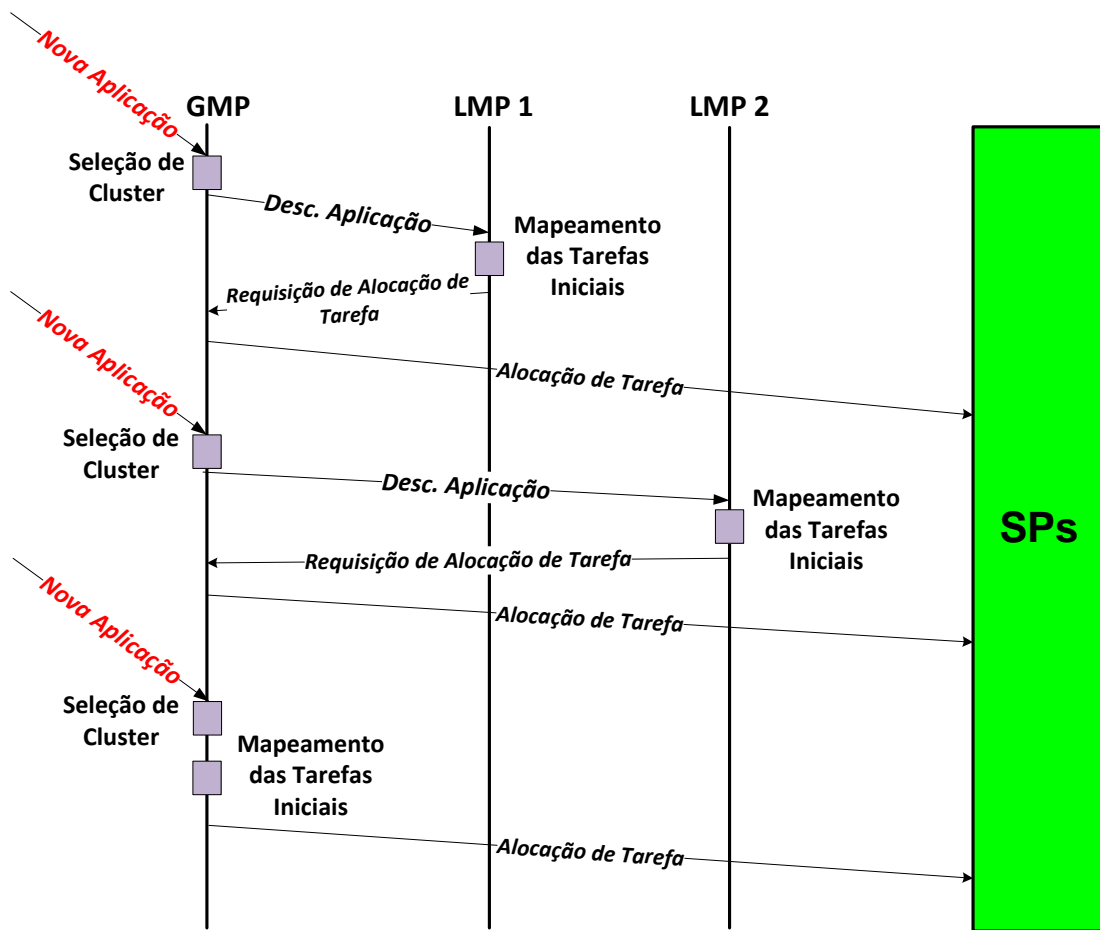


Figura 27 – Protocolo para inserir uma nova aplicação no sistema.

O LMP selecionado armazena a *descrição da aplicação*, que inicia o processo de mapeamento das tarefas (Figura 27 seta “*requisição de alocação de tarefa*”). 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 na seção 4.2.2.

Depois da seleção do SP que irá receber a *tarefa inicial*, o LMP envia um pacote para o GMP com o serviço “*requisição de alocação de tarefa*” contendo  $\{PE_{posição}, ID_i, ADD_i, SIZE_i\}$ . O GMP programa o módulo DMA para ler um dado número de palavras ( $SIZE_i$ ) do *repositório de aplicações* do endereço  $ADD_i$ , transmitindo essas palavras para o  $PE_{posição}$ . O SP irá escalonar a nova tarefa no final da recepção do pacote de *alocação de tarefa*. Além disso, o LMP mantém uma tabela com todos os endereços de rede das tarefas mapeadas.

Considerando a inserção da terceira aplicação na Figura 27, 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.

#### 4.2.2. Processo de Mapeamento de Tarefas

Cada aplicação é iniciada após o mapeamento da(s) tarefa(s) inicial(is). Quando uma tarefa inicial  $t_1$  é iniciada, ela executa algumas funções de processamento, e em seguida, se comunica com uma determinada tarefa (*send* na Figura 28). O SP de origem da  $t_1$  verifica se a tarefa alvo ( $t_2$  no exemplo) está na tabela local de endereços de rede das tarefas. Se estiver, a mensagem é transmitida para a tarefa alvo (de fato é armazenada no *pipe*). Se não estiver, um pacote com o serviço de “*requisição de tarefa*” é transmitido para o LMP responsável pela  $t_1$  (‘1’ na Figura 29).

```

Message msg1;
int main(){
    msg1.length = 128;
    ...
    send(&msg1,t2); //comunicação com a tarefa 2
    ...
    exit();
}

```

Figura 28 – Exemplo de descrição de uma tarefa inicial, com o comando *send*.

Com o pacote de “*requisição de tarefa*” recebido, o LMP executa a heurística de mapeamento PREMAP-DN [MAN11b] para selecionar o SP que deverá receber  $t_2$  (‘2’ na Figura 29), esse processo pode falhar caso o *cluster* não possua recursos disponíveis, o que acarretará na execução do protocolo de reclusterização, explicado na próximo Capítulo. 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 o desempenho geral do sistema melhora, uma vez que o cálculo de mapeamento é um processo computacionalmente intensivo [MAN11b].

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 “*requisição de alocação de tarefa*”, tal como explicado no mapeamento inicial de tarefas (‘3’ na Figura 29). 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 29). 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 29).

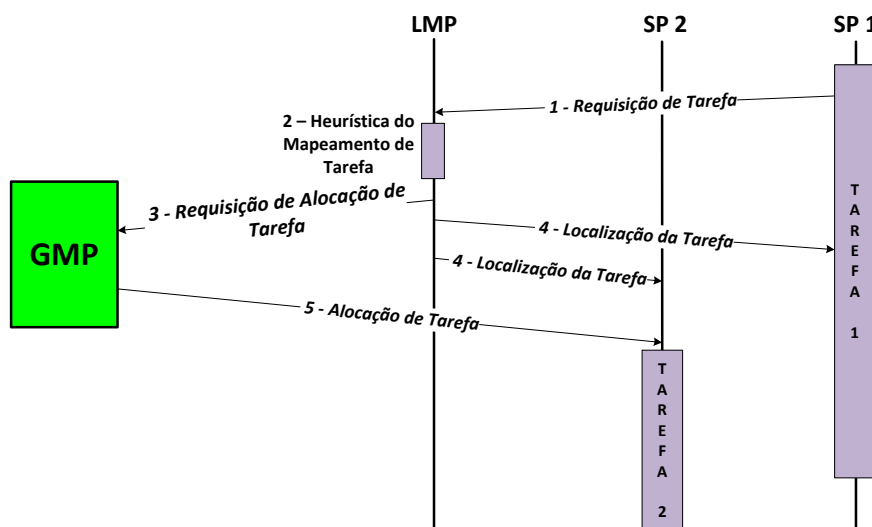


Figura 29 – Protocolo de mapeamento de tarefa.

### 4.2.3. Mapeamento Centralizado VS Distribuído

Na abordagem de mapeamento centralizado, existe apenas um GMP, o qual é responsável pelo mapeamento de todas as tarefas. Todos os pedidos de mapeamento de tarefas são serializados (Figura 30 (a)), reduzindo o desempenho do sistema e aumentando o tráfego na NoC na região do GMP. Usando o método de mapeamento distribuído de tarefas (Figura 30 (b)), o processamento do mapeamento é distribuído em vários LMPs, reduzindo a carga de computação gerada pelos pedidos de mapeamento.

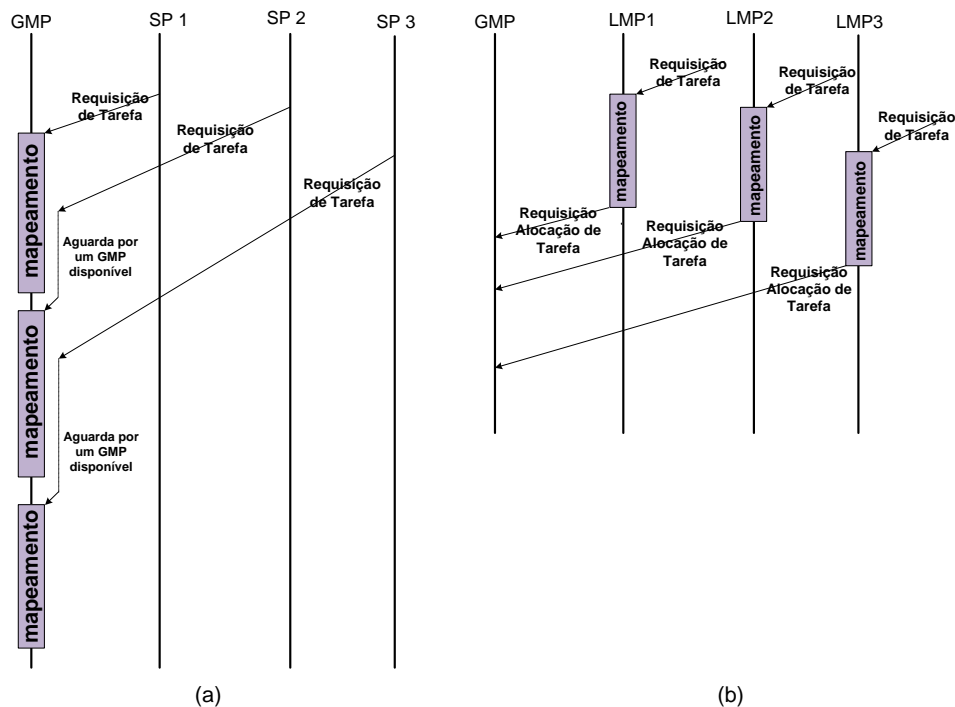


Figura 30 – Mapeamento Centralizado (a) vs Mapeamento Distribuído (b).

O mapeamento centralizado causará um maior atraso para o início da execução das tarefas, dada a serialização do processo de mapeamento. Por consequência, as aplicações terão seu tempo total de execução penalizado e durante o processo de mapeamento as restrições das aplicações, como vazão, dificilmente serão atendidas. Por outro lado, em aplicações periódicas, que executam por um longo período de tempo, após o sistema estabilizar, o mapeamento tanto centralizado quanto distribuído tendem a ter o mesmo desempenho. Esta é uma avaliação informal, sendo a avaliação quantitativa apresentada na Seção 6.2.

## 5. GERÊNCIA COM REGIÕES DINAMICAMENTE ADAPTADAS ÀS APLICAÇÕES

Este Capítulo apresenta a segunda contribuição da Dissertação: o mecanismo de reclusterização. O processo de reclusterização implica no ajuste em tempo de execução do tamanho de um determinado *cluster*. Em um primeiro momento o processo de reclusterização implica no aumento do tamanho do *cluster*, por indisponibilidade de recursos. Este processo de aumento dinâmico no tamanho dos *clusters* é detalhado na seção 5.1. Em um segundo momento o processo de reclusterização pode migrar tarefas que foram mapeadas em *clusters* vizinhos (*clusters* diferentes do qual a aplicação é gerenciada), com o objetivo de melhorar o desempenho das aplicações. Este segundo processo é condicionado não apenas à existência de recursos livres, mas também ao ganho de desempenho pela redução do número de *hops* entre as tarefas comunicantes. O processo de migração de tarefas é detalhado na seção 5.2.

### 5.1. Aumento Dinâmico do Tamanho dos Clusters

Considerar a Figura 31 como exemplo para ilustrar o processo de reclusterização. Nesta figura, o *cluster* posicionado na parte superior esquerda não possui recursos livres, enquanto os demais *clusters* possuem 1 SP disponível.

O *cluster* sem recursos disponíveis recebe em um dado momento uma solicitação de mapeamento de tarefa oriundo de um SP do seu *cluster*. Dada a ausência de recursos livres, o LMP desse *cluster* envia a mensagem “*Requisição de Empréstimo*” requisitando recursos a todos os seus *clusters* vizinhos (passo 1 da Figura 31). A mensagem de pedido de recursos, “*Requisição de Empréstimo*”, contém também o endereço do SP que requisitou o mapeamento de uma nova tarefa. Este processo de requisição é enviado em um primeiro momento para até 8 *clusters* vizinhos (quadrado envolvente do *cluster*). Caso não haja sucesso na primeira exploração, o quadrado envolvente cresce e a exploração pode continuar até todos os *clusters* serem requisitados a responderem se possuem recursos livres. Como já mencionado anteriormente, sempre haverá recurso disponível, pois a aplicação só é inserida no MPSoC se esta condição for verdadeira.

Os LMPs vizinhos ao receberem uma mensagem “*Requisição de Empréstimo*” buscam por recursos disponíveis em seus *clusters*. Se houver apenas um recurso disponível, este recurso é reservado para ser emprestado, caso contrário, se existir mais do que um recurso disponível, o LMP irá reservar o recurso mais próximo possível, em número de *hops*, entre a tarefa a ser mapeada e a tarefa que solicitou o mapeamento. Após a reserva, todos os LMPs vizinhos, notificam a posição do recurso (passo 2 da Figura 31, SPs azuis são reservados), caso esse existir.

O LMP que requisitou recurso externo ao *cluster* escolhe o SP mais próximo de quem requisitou a tarefa, enviando uma mensagem de “*Liberar Recursos*” para liberar os recursos dos LMPs que não foram selecionados (passo 3 da Figura 31). Em seguida, o LMP envia uma mensagem de “*Requisição de alocação de tarefa*” para o GMP requisitando o mapeamento da tarefa no recurso emprestado (passo 4 da Figura 31). Portanto o tamanho do *cluster* aumenta em tempo de execução, pois o recurso emprestado faz parte agora desse *cluster*. Esse processo otimiza a gerência do sistema, uma vez que as aplicações podem ser mapeadas, mesmo se o *cluster* não possuir recursos disponíveis.

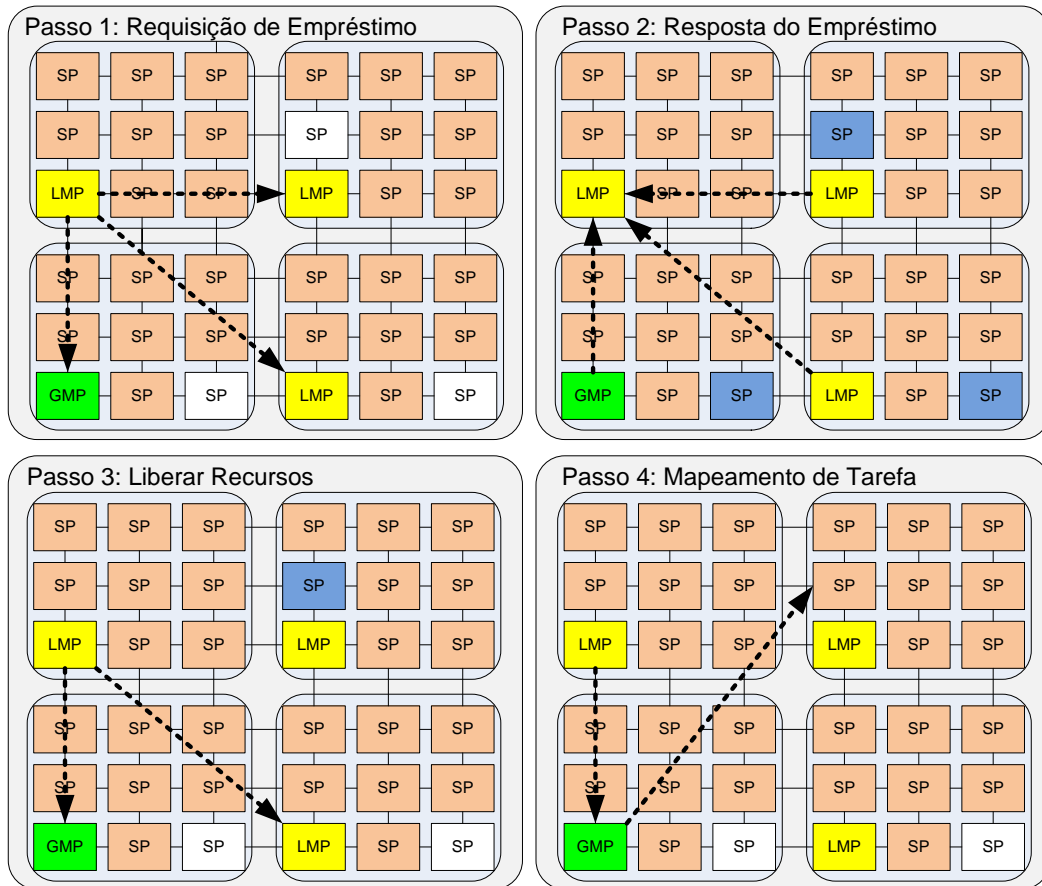


Figura 31 – Protocolo para mapeamento de tarefas em um cluster vizinho. SPs Brancos são PEs com recursos disponíveis.

A Figura 32 mostra um exemplo de troca de mensagens durante o processo de reclusterização. Nesse exemplo o LMP1 não possui recursos disponíveis para mapear uma tarefa, o que o faz iniciar o processo de empréstimo de recursos. Ele faz uma requisição de empréstimo para os seus *clusters* vizinhos. O LMP de cada *cluster* vizinho busca por um PE livre que minimize a distância *Manhattan* até a tarefa requisitante (ação “mapeamento do empréstimo”). Os *clusters* vizinhos ao terminarem o “mapeamento do empréstimo” respondem com a localização do recurso disponível, se houver. Nesse exemplo, o LMP1 escolheu o recurso do *cluster* gerenciado pelo LMP4, liberando os recursos dos demais *clusters*. Após essa escolha, o LMP1 requisita a alocação da tarefa para o GMP que faz a alocação no SP escolhido. Lembrando que esta alocação de tarefa corresponde apenas à transmissão do código objeto, através do DMA do GMP.

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

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

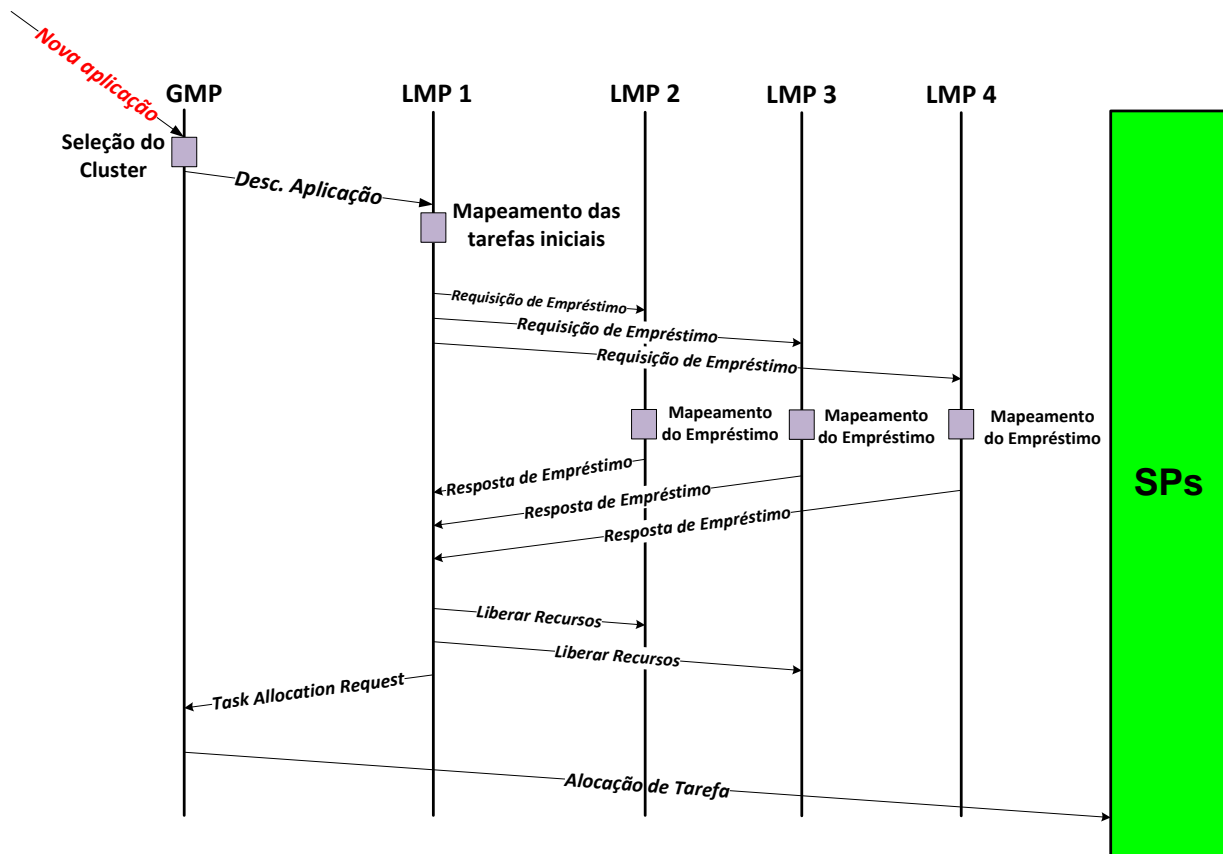


Figura 32 – Exemplo de troca de mensagens durante a reclusterização.

## 5.2. Migração

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

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

A Figura 33 mostra um exemplo do uso da migração de tarefas no protocolo de reclusterização. A Figura 33(a) apresenta o mapeamento de uma determinada aplicação composta de 6 tarefas (A-F). Em um cenário onde as aplicações são inseridas/removidas em tempo de execução. O MPSoC pode ter a maioria de seus recursos ocupados no momento da inserção da aplicação (PEs com a cor laranja), restringindo a procura por recursos livres pela heurística de mapeamento, o que ocasionou no mapeamento de algumas tarefas em *clusters* vizinhos ao *cluster* que gerencia a aplicação (cluster 2).

Em um dado momento, uma tarefa dentro do cluster termina a sua execução, informando ao LMP o seu término e liberando um recurso do cluster (Figura 33(b)). Após o recebimento da mensagem de término de tarefa, caso haja alguma tarefa que o LMP gerencia fora do seu cluster, ele executa uma função que calcula se é viável a migração. Essa função compara o número de *hops* da localização atual da tarefa até suas comunicantes com o número de *hops* da localização,

caso ela seja migrada até suas comunicantes. Se o número de *hops* se tornar menor após a realização da migração, essa migração é viável.

Com isso, o LMP envia uma mensagem para a tarefa que será migrada, solicitando sua migração para o PE com o recurso livre. A tarefa então migra para o PE destino e informa ao LMP do *cluster* que está mapeada, que está migrando e que desocupará o recuso do *cluster* (Figura 33(c)).



Figura 33 – Exemplo de reclusterização usando migração de tarefas.

O detalhamento do protocolo de migração de tarefas é mostrado na próxima seção.

### 5.2.1. Protocolo da Migração de Tarefas

O protocolo de migração de tarefas é ilustrado na Figura 34 com um exemplo e pode ser resumido da seguinte forma:

- 1) Uma tarefa alocada no SP1 termina sua execução e avisa o LMP do seu *cluster* o seu término.
- 2) O LMP verifica se há alguma tarefa que ele gerencia fora do *cluster*. Caso existe, ele verifica se a migração dessa tarefa é viável, ou seja, há uma redução no número de *hops* entre as tarefas comunicantes.



- 3) Caso a migração seja viável, o LMP envia uma mensagem de requisição de migração para essa tarefa, nesse exemplo, localizada no SP2.
- 4) A tarefa só pode ser migrada se está no estado de execução. O escalonador do *microkernel* verifica essa condição. Se a tarefa pode ser migrada, o *microkernel* envia para o PE destino, um pacote com o conteúdo completo da página da tarefa e seus descritores de tarefas (TCB – Task Control Block). O detalhamento do processo de migração de tarefas é detalhado em [MOR12].
- 5) A tarefa migrada é escalonada uma vez que o código e o TCB forem completamente recebidos.
- 6) No exemplo, uma tarefa comunicante envia uma requisição de leitura para a tarefa que foi migrada, mas para o SP que ela não está mais alocada.
- 7) Quando essa requisição de leitura chega, ela é repassada para o SP onde se encontra a tarefa.
- 8) O SP1 então recebe a requisição de leitura e responde a requisição, já com a sua nova posição.

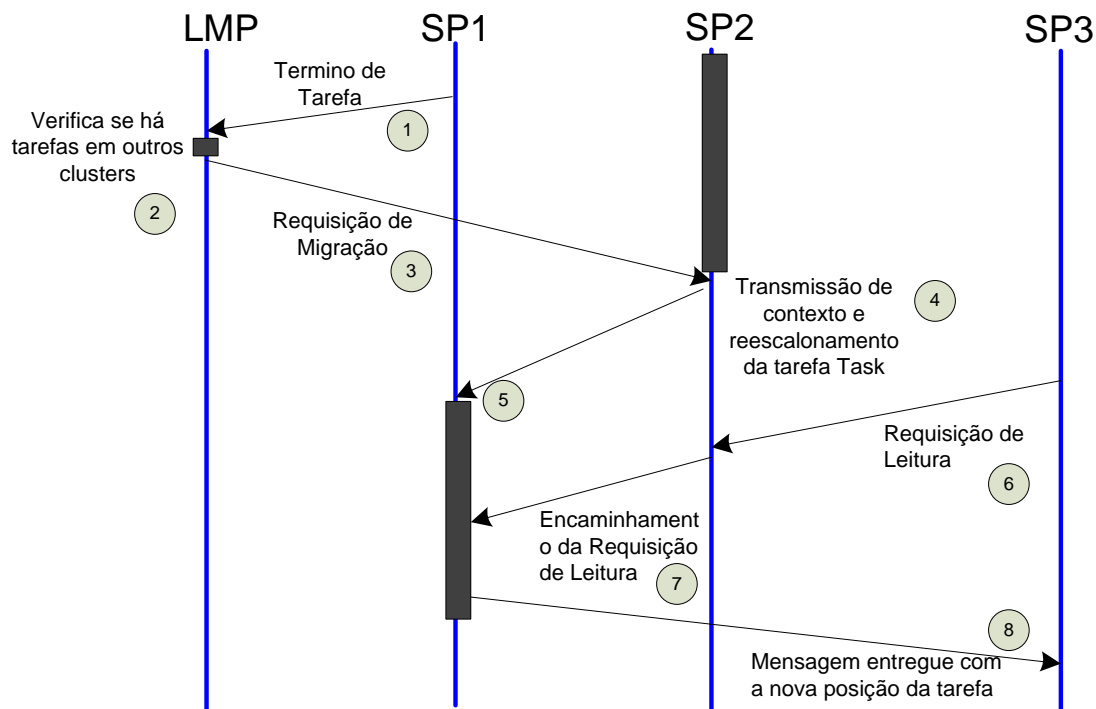


Figura 34 – Protocolo de migração de tarefas.

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

Ainda neste contexto, há o controle de entrega de mensagens. O processo que garante que as mensagens serão consumidas na ordem correta é o armazenamento local das tarefas produzidas. Quando a tarefa é migrada, as mensagens produzidas pelos *send()* permanecem armazenadas no *pipe* do PE original. Portanto, as tarefas que se comunicarem com a tarefa

migrada ainda utilizam o endereço da tarefa antes da mesma migrar. Uma vez que todas as mensagens sejam consumidas, as requisições de comunicação são encaminhadas para a nova posição da tarefa, que responde essas requisições com o novo endereço. A partir deste momento, as tarefas que se comunicam com a tarefa migrada utilizam o novo endereço.

## 6. RESULTADOS

Foi utilizado para os experimentos desse trabalho o MPSoC HeMPS [CAR09], descrito em um nível RTL preciso no nível de ciclo de relógio (SystemC). O MPSoC HeMPS é modelado em outros dois níveis de abstração: VHDL RTL, onde todos os módulos são descritos em VHDL sintetizável; ISS + VHDL, processadores são modelados usando um simulador de conjuntos de instruções preciso em nível de ciclo (ISSs), o restante dos módulos são descritos em VHDL.

A Tabela 2 mostra a comparação dos tempos de simulação entre os três diferentes níveis de abstração, onde cada linha da tabela é identificada pelo tamanho do MPSoC e um sufixo. O detalhamento destes experimentos foi publicado em [PET12]. Quando esse sufixo é *\_1*, isso significa que somente uma instância da aplicação foi inserida no MPSoC durante a simulação. Quando sufixo é *\_25*, *\_50*, *\_70* ou *\_100*, isso corresponde a um número de instâncias de aplicações que de tal modo chegue perto de 25%, 50%, 70% e 100% de páginas ocupadas do MPSoC.

Foi escolhido o modelo do MPSoC descrito em SystemC RTL, pois como mostra a Tabela 2, ele chega a ser 175 vezes mais rápido do que o modelo descrito em VHDL e 8 vezes mais rápido em comparação ao modelo ISS + VHDL.

Tabela 2 – Tempo de simulação (em segundos), considerando vários níveis de abstração do MPSoC HeMPS.

Cenário	VHDL	ISS + VHDL	SystemC (Cycle-accurate)
4x4_1	3852,34	217,35	34,99
4x4_25	4219,99	220,17	35,26
4x4_50	4701,47	231,67	37,97
4x4_75	5162,13	245,78	40,56
4x4_100	7288,23	291,73	46,86
5x5_1	7742,04	336,25	53,22
5x5_25	8134,28	359,32	60,83
5x5_50	9087,88	423,85	65,76
5x5_75	12205,26	429,10	82,84
5x5_100	12460,31	466,63	85,45
7x7_1	19765,50	682,96	114,18
7x7_25	21797,14	724,61	129,06
7x7_50	32153,21	1049,32	179,42
7x7_75	41211,06	1272,44	226,46
7x7_100	50557,62	1538,20	274,07
10x10_1	50862,91	2344,64	289,52

Essa seção apresenta os resultados usando três *benchmarks*: *MPEG*, executa uma parte de um decodificador MPEG; *multispec image analysis*, avalia a semelhança entre duas imagens usando frequências diferentes [TAN08]; *synthetic*, aplicação sintética.

Os dados obtidos na Tabela 2, e no presente Capítulo, foram obtidos com uma ferramenta desenvolvida pelo Autor, que permite a execução de dezenas de simulações simultaneamente, sobre diferentes configurações de *benchmarks*. Esta ferramenta permite a execução testes de regressão. Estes testes de regressão permitem validar cada nova técnica agregada ao MPSoC, e obter de forma automatizada os resultados.

Características relevantes do MPSoC incluem: palavra do PE de 32-bit e 16-bit de flit; tamanho da página com 16 Kbytes (4,096 palavras); time-slice: 16,384 ciclos de relógio (quantidade de tempo que uma tarefa é escalonada), 2 páginas por PE para execução de tarefas; roteador da NoC: *wormhole packet switching*, roteamento XY, arbitragem *round-robin*.

### 6.1. Tempo Total de Execução

O primeiro conjunto de resultados avalia o tempo total de execução das aplicações, considerando: (i) tamanho do MPSoC; (ii) tamanho do *cluster*; (iii) carga do MPSoC, ou seja, percentual de PEs escravos executando tarefas da aplicação.

A Tabela 3 apresenta os tempos de execução normalizados em relação a um MPSoC 12x12 com gerência centralizada (1 *cluster*) e uma carga igual a 75%. Como pode ser observado, a gerência distribuída conduz a uma redução total no tempo de execução. A menor redução observada no benchmark MPEG é devido à sua característica de periodicidade. A redução no tempo total de execução deve-se a:

1. LMPs executam o mapeamento de tarefas em paralelo.
2. Cada LMP trata de um menor número de pacotes de controle (requisição de mapeamento, fim de tarefas, monitoramento) comparado ao método centralizado. Isso reduz a carga do gerente e o tráfego na NoC.

Uma questão relevante a avaliar é o tamanho do *cluster*. Existe um tamanho ótimo de *cluster*? Mesmo com poucos *clusters*, como dois, o tempo total de execução é reduzido, devido o paralelismo das tarefas de gerenciamento. Um grande número de clusters requer um grande número de recursos do MPSoC, reduzindo o número de aplicações que se pode executar simultaneamente. A partir da Tabela 3, um tamanho de *cluster* com 18 (6x3) ou 16 (4x4) PEs representa um bom custo-benefício entre a redução no tempo de execução e os recursos reservados ao gerenciamento.

Tabela 3 – Tempos de execução normalizados em relação a um MPSoC 12x12 com gerência centralizada e uma carga igual a 75%.

Tamanho do Cluster	Nº de Clusters	Benchmark		
		MPEG	Synthetic	Multispec
12x12	1	1,00	1,00	1,00
12x6	2	0,94	0,78	0,77
6x6	4	0,90	0,67	0,63
6x4	6	0,88	0,58	0,71
<b>6x3</b>	<b>8</b>	<b>0,86</b>	<b>0,57</b>	<b>0,56</b>
<b>4x4</b>	<b>9</b>	<b>0,88</b>	<b>0,58</b>	<b>0,52</b>
3x3	16	0,87	0,54	0,49

A Tabela 4 apresenta o tempo total de execução em uma instância 6x6 do MPSoC. Nesse MPSoC menor, o número de tarefas simultâneas é menor comparada a um MPSoC de grande dimensão. Portanto, o gerenciamento de carga é menor. Nesse caso, o particionamento de um MPSoC 6x6 é menos eficiente do que em um MPSoC maior.

Tabela 4 – Tempos de execução normalizados em relação a um MPSoC 6x6 com gerência centralizada e uma carga igual a 75%.

Tamanho do Cluster	Nº de Clusters	Benchmark		
		MPEG	Synthetic	Multispec
6x6	1	1,00	1,00	1,00
6x3	2	1,00	0,95	0,96
3x3	4	0,99	1,03	0,91
3x2	6	0,99	0,92	0,87

Uma segunda questão se impõe: quando o gerenciamento distribuído deve começar a ser utilizado? A partir dos resultados apresentados Tabela 5, MPSoCs começando em 64 PEs, com 4 *clusters* de tamanho 4x4, apresentam significativas reduções no tempo de execução. Este tamanho de *cluster* também prove redução no número médio de hops entre tarefas (discussão aprofundada na Seção 6.3), e um baixo comprometimento de recursos para gerência (6,25%). Logo, *clusters* cujo tamanho implique na utilização de 6% a 8% dos PEs para atividades de gerência correspondem a um bom custo-benefício entre ganho de desempenho e perda de recursos.

Tabela 5 – Comparação do tempo de execução e do número de hops entre MPSoC de tamanho 6x6, 8x8 e 12x12, com carga de 75%.

Tamanho do MPSoC	Tamanho do cluster	Overhead de recursos	Benchmark			
			MPEG		Synthetic	
			Tempo de execução	Hops	Tempo de execução	Hops
12X12	12X12	0,69%	1	2,57	1	6,8
12X12	6x6	2,78%	0,9	2,26	0,67	4,9
12X12	4X4	6,25%	0,88	2,22	0,58	4,24
12x12	3X3	11,11%	0,87	2,18	0,54	4
8x8	8x8	1,56%	1	2,96	1	5,33
8x8	4x4	6,25%	0,93	2,58	0,86	4,26
8x8	2x2	25%	0,94	3,36	0,77	4
6x6	6x6	2,78%	1	3	1	4
6x6	6x3	5,56%	0,99	2,27	0,94	4,22
6x6	3x3	11,11%	0,99	2,22	0,92	5,3

A Tabela 6 avalia o impacto do tamanho do *cluster* contra a carga do sistema. Isso é importante para diferenciar o tempo de execução de uma aplicação e o tempo total de execução. O tempo de execução de uma aplicação é praticamente constante, uma vez que a aplicação inicia sua execução. O tempo total de execução considera a quantidade total de tempo necessário para executar todas as aplicações inseridas no MPSoC.

Dois cenários devem ser considerados. O primeiro está relacionado a aplicações periódicas, com tempo de execução longo. Em tais casos, as ações do gerenciamento não devem influenciar fortemente com o aumento da carga. Como pode ser observado no *benchmark* MPEG, o tempo

total de execução no método centralizado aumentou 48% comparando um sistema com uma aplicação (carga=1%) com outro sistema com 75% de carga. O gerenciamento distribuído, com tamanho de cluster 6x3, apresentou um aumento menor no tempo de execução, 28%. Apesar do fato da aplicação ser periódica, esse resultado mostra os benefícios de executar em paralelo o gerenciamento do sistema.

O segundo cenário diz respeito a aplicações com tempo de execução delimitado. Nesse cenário, o gerente desempenha um papel importante. No gerenciamento centralizado, todos os mapeamentos de tarefas são serializados, com aplicações sendo iniciadas sequencialmente. No gerenciamento distribuído, os mapeamentos são realizados em paralelo, com várias aplicações iniciando ao mesmo tempo. Isso explica as diferenças observadas na Tabela 6: de 264%/230% no gerenciamento centralizado para 117%/77% com tamanho de cluster 6x3 no gerenciamento distribuído.

Tabela 6 – Tempo total de execução (em ciclos de relógio) para um MPSoC 12x12, com carga de 1% e 75% e *overhead* no tempo total de execução em relação ao aumento da carga.

Tamanho do Cluster	Nº de Clusters	MPEG			Synthetic			Multispec		
		1%	75%	sobrecarga	1%	75%	sobrecarga	1%	75%	sobrecarga
12x12	1	4.868.651	7.184.706	<b>48%</b>	891.550	3242574	<b>264%</b>	1.230.154	4064464	<b>230%</b>
12x6	2	4.863.129	6.721.754	38%	861.654	2533323	194%	1.290.601	3141847	143%
6x6	4	4.859.902	6.454.773	33%	859.436	2160130	151%	1.274.700	2555688	100%
6x4	6	4.859.600	6.313.663	30%	856.859	1890110	121%	1.274.518	2890950	127%
6x3	8	4.860.342	6.213.974	<b>28%</b>	855.673	1857775	<b>117%</b>	1.275.361	2261704	<b>77%</b>
4x4	9	4.860.016	6.310.882	30%	855.787	1866830	118%	1.276.130	2121722	66%
3x3	16	4.861.843	6.281.683	29%	856.845	1763911	106%	1.277.223	1994494	56%

Resumindo a avaliação do tempo total de execução: (i) gerência distribuída é eficaz para MPSoCs grandes, ou seja, que contenham mais do que 64 PEs; (ii) um tamanho de *cluster* cujo tamanho implique na utilização de 6% a 8% dos PEs para atividades de gerência correspondem a um bom custo-benefício entre ganho de desempenho e perda de recursos; (iii) a gerência distribuída reduz o aumento do tempo total de execução quando se aumenta a carga do sistema.

## 6.2. Mapeamento Distribuído de Tarefas

Para avaliar a proposta de mapeamento distribuído de tarefas, foi realizada uma comparação entre o método de mapeamento proposto e um mapeamento centralizado. A primeira avaliação apresenta três instâncias da aplicação *Multispect* executando em três tamanhos de MPSoCs: 6x6, 9x9 e 12x12. A Figura 35 apresenta os tempos, em ciclos de relógio, de quando cada instância de aplicação começa e termina. Nota-se que no mapeamento distribuído (barras cinzas) um conjunto de aplicações começam simultaneamente, devido a execução distribuída da heurística de mapeamento. Por outro lado, o mapeamento centralizado (barras brancas) tem que mapear as tarefas da aplicação e então tratar o pedido de novas aplicações. Esta serialização do processo de mapeamento é claramente observada em todos os resultados da Figura 35. Entretanto, uma vez mapeadas as tarefas da aplicação, o tempo que ela leva para terminar sua execução é praticamente igual nos dois métodos.

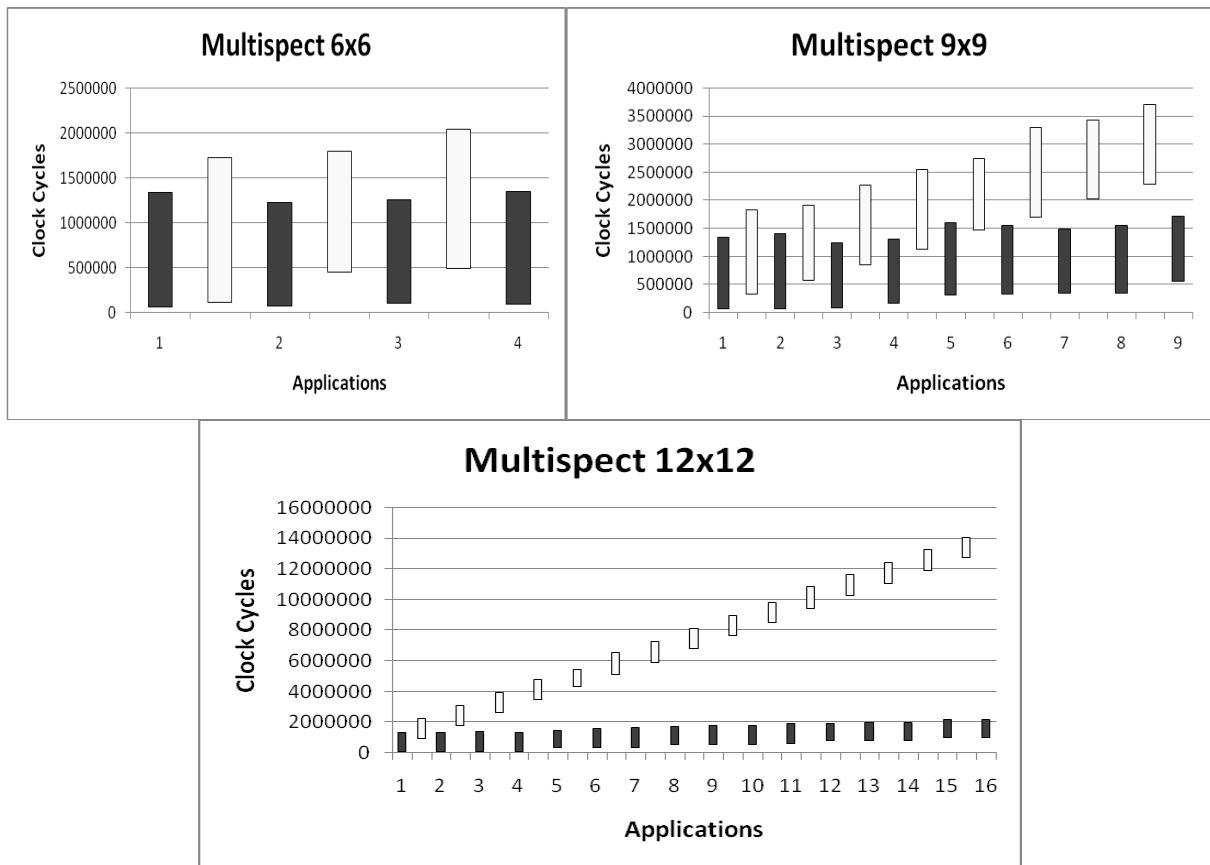


Figura 35 – Mapeamento distribuído (barras cinzas) versus centralizado (barras brancas), para o *benchmark* Multispec com três tamanhos de NoC.

A Figura 36 apresenta os tempos, em ciclos de relógio, de quando cada instância de aplicação começa e termina, para a aplicação MPEG, para dois tamanhos de MPSoC

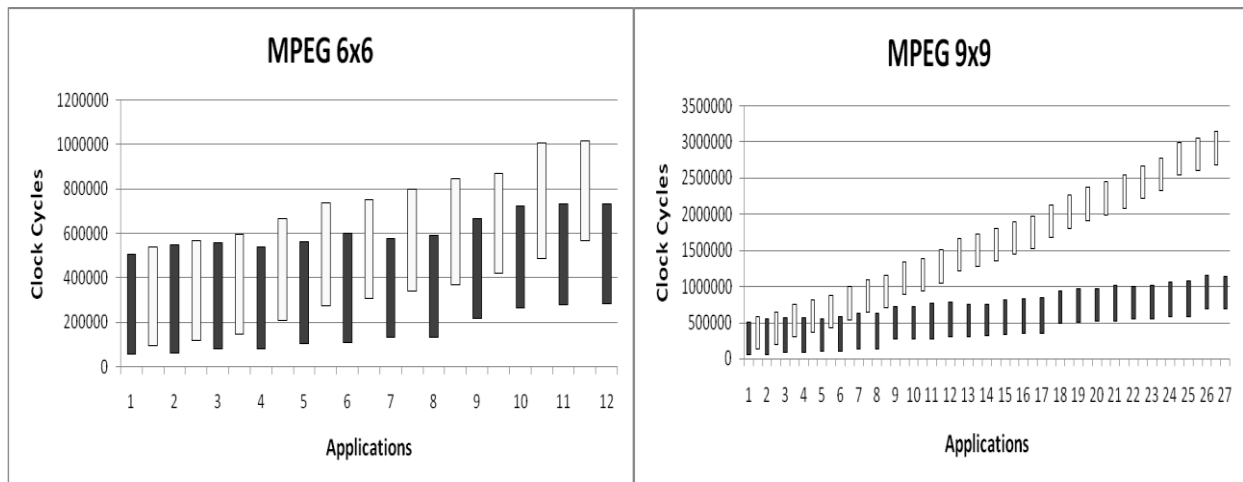


Figura 36 – Mapeamento Distribuído (barras cinzas) versus Centralizado (barras brancas), para o *Benchmark* MPEG para dois tamanhos de MPSoC.

Observa-se na Figura 36 um comportamento similar à Figura 35: serialização do mapeamento na gerência centralizada e tarefas com mesmo tempo de execução. Estes gráficos ilustram o porquê da redução do tempo de execução observado nas tabelas apresentadas na Seção anterior. À medida que o tamanho do MPSoC aumenta, o tempo de execução para o mapeamento centralizado cresce dramaticamente, impactando fortemente no tempo total de execução das aplicações (Tabela 3, apresentada anteriormente).

### 6.3. Número de Hops

A avaliação do número médio de *hops* é um parâmetro importante para avaliar a qualidade do mapeamento de tarefas. Um número pequeno de *hops* entre as tarefas reduz a energia de comunicação, uma vez que as tarefas comunicantes são mapeadas perto umas das outras. Foi utilizado como base o modelo de energia proposto por Hu et al. [HU03]. Esse modelo calcula a energia consumida na comunicação para transmitir um *bit* por uma distância de  $n_{hops}$ , utilizando a Equação 3.

$$E_{hops} = n_{hops} * E_{Sbit} + ( n_{hops} - 1 ) * E_{Lbit} \quad (3)$$

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 um *bit* passou. 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 constante.

Por outro lado, um número alto de *hops* entre as tarefas além de aumentar a energia consumida na comunicação prejudica o desempenho das aplicações, uma vez que o tráfego das demais aplicações pode interferir na comunicação (disputa por caminhos comuns na NoC).

O cálculo do número de *hops* de uma aplicação foi feito somando-se o número de *hops* que cada tarefa da aplicação tem de distância para suas tarefas comunicantes. Um exemplo desse cálculo é mostrado na Figura 37. Na Figura 37(a) é mostrado o grafo de uma aplicação, onde o número nas setas corresponde ao número de hops que a tarefa tem de distância entre sua tarefa comunicante. Esses números de hops foram obtidos usando o mapeamento da Figura 37(b). Uma distância em *hops* igual a zero (como entre as tarefas 'C' e 'D') significa que ambas foram mapeadas no mesmo PE.

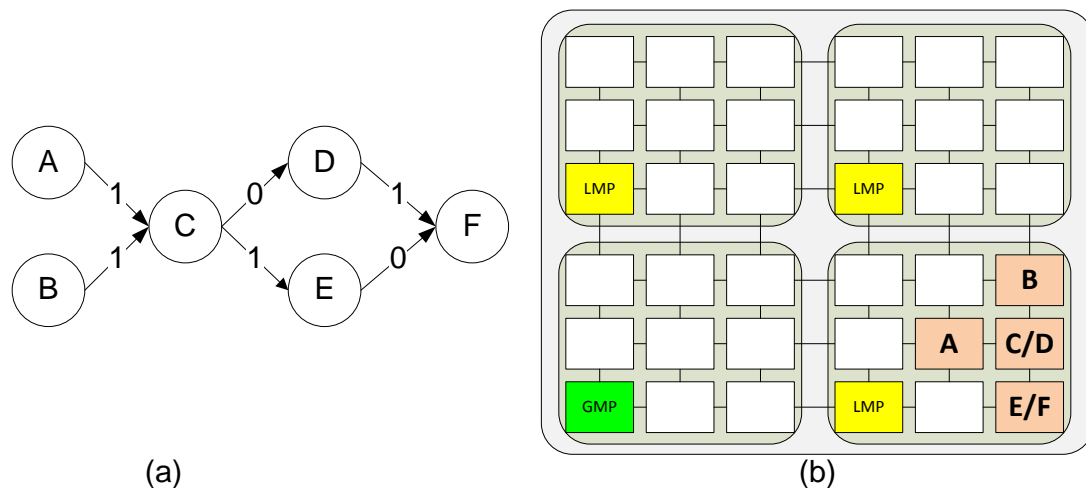


Figura 37 – Grafo da aplicação e o número de *hops* entre cada para comunicante (a), e a aplicação *Synthetic* mapeada no MPSoC (b).

A Tabela 7 apresenta os dados estatísticos relativos ao número de *hops* entre tarefas, em um MPSoC 12x12, com 75% de carga, para dois *benchmarks* e três tamanhos de cluster. Em ambos os *benchmarks*, a gerência centralizada (cluster com tamanho 12x12) apresentou um número médio de *hops* maior do que no gerenciamento distribuído. As colunas “min” da Tabela 7, representam o melhor mapeamento, em número de *hops*, das aplicações executadas.



Tabela 7 – Número de *hops* entre tarefas em um MPSoC 12x12 com carga igual a 75%.

Tamanho do Cluster	Nº de Clusters	Synthetic (6 tarefas)				MPEG (5 tarefas)			
		min	méd	max	Desv. Padrão	min	méd	max	Desv. Padrão
12x12	1	4	6,8	20	3,98	2	2,57	7	1,09
6x3	8	4	4,05	6	0,34	2	2,02	3	0,15
4x4	9	4	4,24	6	0,66	2	2,22	4	0,57

O mapeamento no gerenciamento centralizado poderia potencialmente gerar melhores resultados no que na versão com gerência distribuída, porque o mapeador tem uma visão global do MPSoC. Isso é verdade quando a carga aplicada no MPSoC é pequena. Em tais casos, ambas as técnicas de gerência apresentam resultados similares em termos do número médio de *hops*, como mostra a Tabela 8.

Tabela 8 – Número médio de *hops* entre tarefas em MPSoCs com carga igual a 25%.

Tamanho do MPSoC	Tamanho do cluster	Nº de Clusters	Benchmark	
			MPEG	Synthetic
12X12	12X12	1	2,43	4,85
12X12	6X3	8	2	4
12X12	4X4	9	2	4
6x6	6x6	1	2	4
6x6	3X3	9	2	4

Nos cenários com alta carga (Tabela 7), o número de regiões contínuas usando mapeamento centralizado reduz, aumentando desde modo o número de *hops*. Por outro lado, o uso de *clusters* favorece a utilização dos recursos do MPSoC, mantendo regiões contínuas mesmo sob cargas maiores aplicadas no MPSoC. O resultado apresentando na Tabela 7 está também relacionado ao tempo total de execução (seção anterior). Quanto maior o número médio de *hops*, maior o tempo total de execução.

O desvio padrão observado no MPSoC com gerência centralizada é alto (3,98 e 1,09), significando que algumas aplicações têm suas tarefas mapeadas longe umas das outras. O MPSoC com gerência distribuída apresentou um desvio padrão menor, de 0,15 até 0,66. Isso significa que a maioria das aplicações foram mapeadas em regiões contínuas, dentro do *cluster*, favorecendo QoS. Quando o *cluster* está quase cheio, a gerência distribuída “toma emprestado” recursos dos seus clusters vizinhos. Esse fato pode ser observado nas colunas “max” da terceira e quarta linha da Tabela 7, onde se pode observar um número máximo de *hops* bem superior à média obtida, devido ao fato de algumas tarefas terem sido mapeadas em outros *clusters* devido a falta de recursos. Para reduzir o número de *hops* em tempo de execução, é adotado a migração de tarefas, como explicado anteriormente.

Para ilustrar a diferença observada no número de *hops*, a Figura 38 mostra uma instância da aplicação MPEG mapeada com ambas as técnicas de gerenciamento. Na gerência centralizada múltiplas instâncias da aplicação são mapeadas em regiões contínuas, porém as últimas instâncias a serem mapeadas tendem a ficar fragmentadas, dada a escassez de recursos, como mostrado na

Figura 38(a). Por outro lado, o espaço de busca inicial do mapeamento com a gerência distribuída é sempre restrito ao cluster, favorecendo um número menor de *hops*. Esse exemplo simples mostra como é importante a preservação da localidade espacial, empregando o método de clusterização.

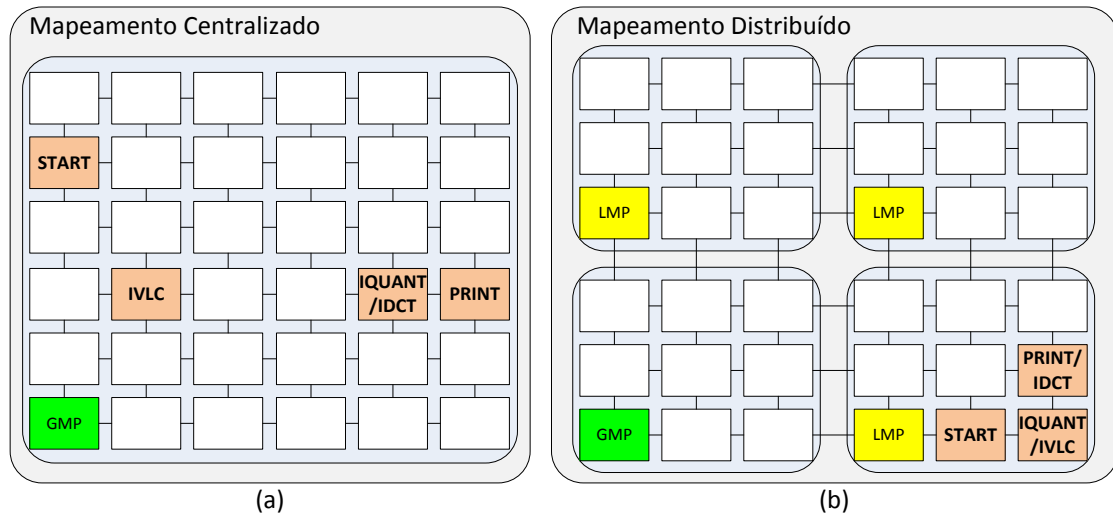


Figura 38 – Mapeamento de uma instância da aplicação MPEG em um MPSoC 6x6.

#### 6.4. Reclusterização

Essa seção avalia o processo de reclusterização usando migração de tarefas. Os resultados foram obtidos em um MPSoC com tamanho 6x6 com ambas as técnicas de gerenciamento. Na gerência distribuída, foi utilizado tamanho de *cluster* igual a 3x3. Dois cenários foram avaliados: (i) *main application* (aplicação principal) -  $MA_{pp}$  (aplicação avaliada) com aplicações “perturbadoras” (aplicações simples, que geram tráfego para competir com recursos da NoC utilizados por  $MA_{pp}$ ), sem migração de tarefas; (ii)  $MA_{pp}$  com aplicações “perturbadoras” e duas migrações de tarefas (tarefas E e D). Estes cenários são apresentados na Figura 39.

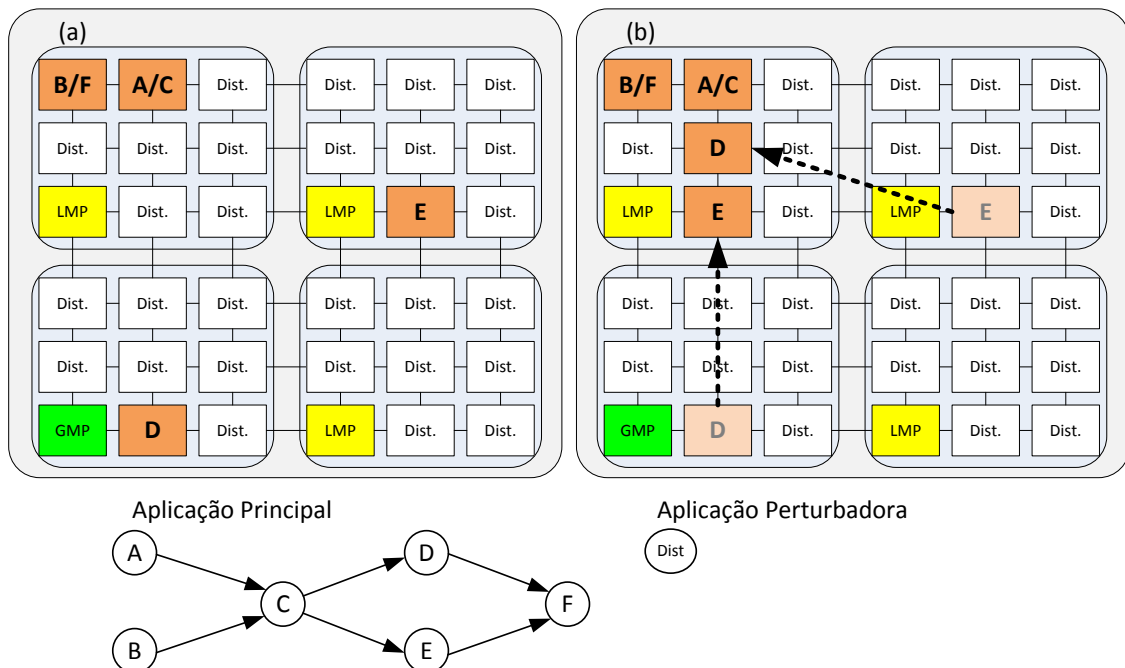


Figura 39 – Exemplo de um cenário com reclusterização.

A Figura 39(a) mostra o mapeamento inicial das tarefas. As tarefas D e E da  $MA_{pp}$  foram mapeadas em *clusters* vizinhos, e não no *cluster* que contém o PE que gerencia a  $MA_{pp}$  (*cluster* no topo à esquerda). Essas duas tarefas foram mapeadas em *clusters* vizinhos devido à ausência de recursos *cluster* que gerencia a aplicação  $MA_{pp}$ . Quando as tarefas pertencentes às aplicações “*perturbadoras*” terminam suas execuções, o gerente do *cluster* verifica se existem tarefas que ele gerencia, executando em outros *clusters*. Nesse caso, o LMP envia a requisição de migração para os SPs executando essas tarefas. Na Figura 39(b), as tarefas D e E foram migradas para o *cluster* que gerencia a aplicação  $MA_{pp}$ . Isso resultou em um menor número de *hops* entre as tarefas da aplicação  $MA_{pp}$ , com uma melhora no desempenho de execução.

A aplicação  $MA_{pp}$  é periódica, com um comportamento *pipeline*, repetindo cada tarefa por um número parametrizável de iterações. As Figura 40 e Figura 41 mostram o tempo de execução de cada iteração da tarefa F. Os resultados obtidos no começo da simulação correspondem ao mapeamento das tarefas, sendo considerado como estado transitório. De acordo com a Figura 40, o tempo por iteração é estabilizado em aproximadamente 10.000 ciclos de relógio (o tempo por iteração varia em função do tempo de chegada dos pacotes).

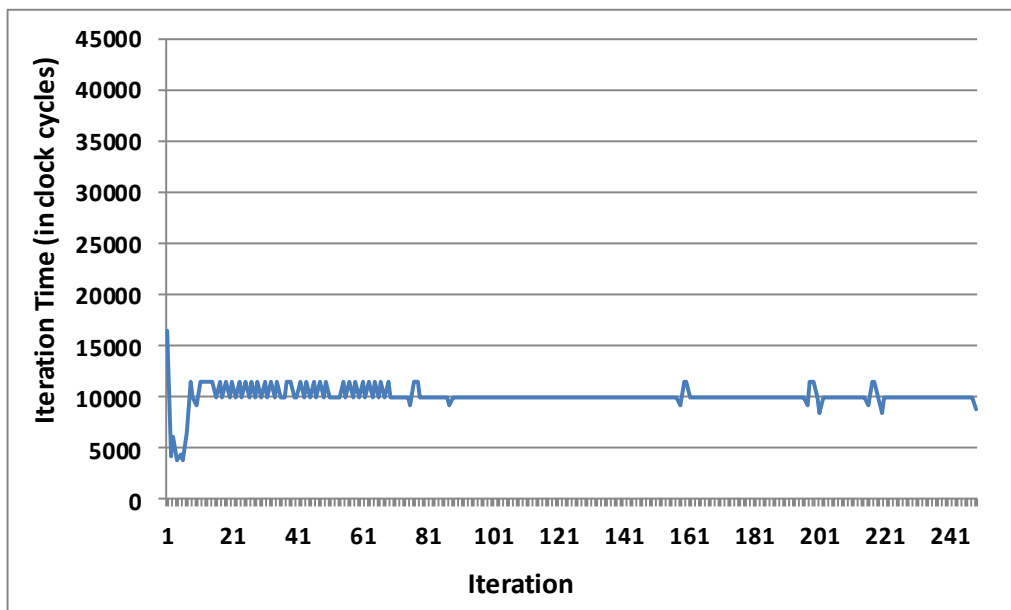


Figura 40 – Tempo de iteração da Tarefa F, sem migração de tarefa.

A Figura 41 avalia o cenário com migração de tarefas. Tal como esperado, durante a migração de tarefas o tempo da iteração aumenta, porque a aplicação é suspensa durante a migração, pois os dados não estão sendo consumidos pelas tarefas D e E, que estão sendo migradas. Depois da migração, o tempo por iteração da tarefa F estabiliza em 8.600 ciclos de relógio, reduzindo também a oscilação observada na Figura 40. Tal melhoria é decorrente da redução do número de *hops* entre as tarefas da aplicação  $MA_{pp}$ .

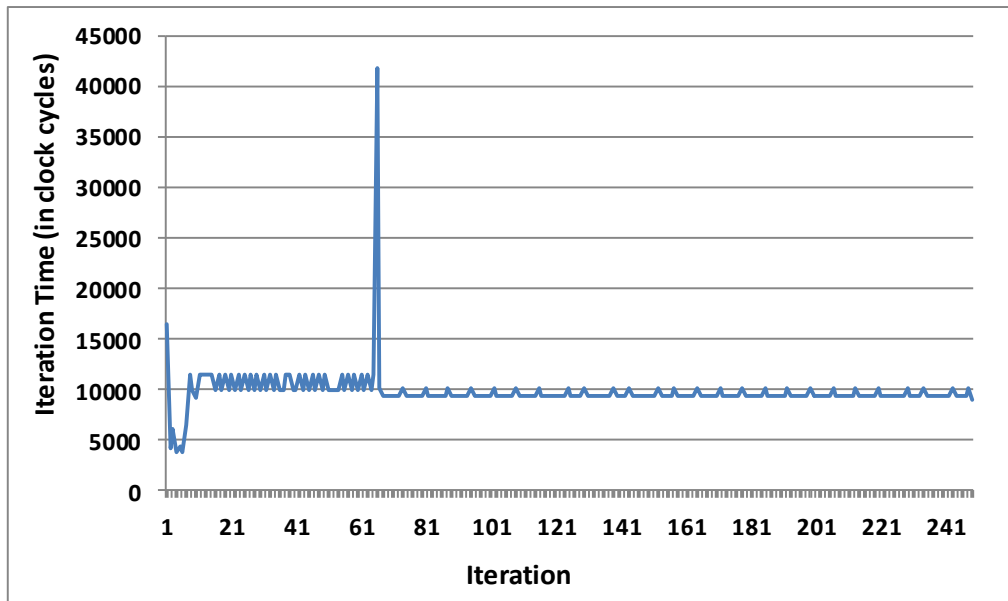


Figura 41 – Tempo de iteração da Tarefa F, com migração de tarefa.

O tempo total de execução para ambos os cenários foi de 2.772.692 (sem migração) e 2.698.812 (com migração), em ciclos de relógio. Isso corresponde a uma redução de 2,67% no tempo total de execução, considerando duas migrações de tarefas. Portanto, mesmo a migração de tarefa aumentando momentaneamente o tempo de execução, no resultado final há uma melhora no desempenho global.

## 7. CONCLUSÕES E TRABALHOS FUTUROS

A presente Dissertação contribuiu com o estudo da gerência distribuída de recursos em MPSoCs. A opção escolhida foi a gerência em *clusters*, com tamanho fixo em tempo de projeto, e dimensionamento modificável dinamicamente em tempo de execução.

As avaliações das técnicas desenvolvidas buscaram responder a questões como: (1) para qual tamanho de MPSoC a gerência distribuída é vantajosa em relação à gerência centralizada?; (2) qual o melhor compromisso entre o comprometimento de recursos *versus* ganho de desempenho, i.e., tamanho do *cluster*?; (3) o processo de reclusterização impacta no desempenho global, i.e., custo da migração de tarefas?

Para responder a estas questões, além da implementação das técnicas de gerência, foi desenvolvida uma ferramenta que permite a execução de dezenas de simulações simultaneamente sobre diferentes configurações de *benchmarks*. Este processo é denominado de “*teste de regressão*”. Este teste de regressão permite validar cada nova técnica agregada ao MPSoC, e obter de forma automatizada os resultados apresentados no Capítulo 6.

De posse dos resultados, pode-se responder às questões acima. MPSoCs a partir de 64 PEs, com um comprometimento entre 6-8% dos PEs para gerência respondem as questões (1) e (2). A migração de tarefas impacta apenas momentaneamente no desempenho das aplicações, mas no final o desempenho é melhorado devido à aproximação das tarefas comunicantes.

Finalmente, é importante destacar que este estudo foi implementado sobre um MPSoC particular, HeMPS. Entretanto, o Autor acredita que as características arquiteturais adotadas e justificadas no início do Capítulo 3 são genéricas o suficiente para que os resultados obtidos possam ser considerados genéricos para MPSoCs semelhantes, como os que estão sendo hoje adotados pela indústria e academia.

### 7.1. Publicações Relacionadas ao Tema da Dissertação

As contribuições apresentadas neste trabalho foram enviadas para conferências pelo meio de três artigos aprovados e um submetido. Estes artigos compreendem:

Referente ao Capítulo 4:

- CASTILHOS, G. M.; MANDELLI, M.; MORAES, F. G. “**Enhancing Performance of MPSoCs through Distributed Resource Management**”. In: ICECS 2012, Seville, Spain. IEEE International Conference on Electronics, Circuits and Systems, 2012. (QUALIS B1) [MAN12].

Referentes ao Capítulo 5:

- MORAES, F. G.; MADALOZZO, G; CASTILHOS, G; CARARA, E. “**Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs**”. In: ISCAS 2012, Seoul, Korea. IEEE International Symposium on Circuits and Systems, 2012. v. 1. p. 644-647. (QUALIS A1) [MOR12].
- CASTILHOS, G. M.; MADALOZZO, G; MANDELLI, M; MORAES, F. G. “**Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes**”. In: ISVLSI 2013 (QUALIS B1, submetido).

Referente ao Capítulo 6:

- PETRY, C. A.; WÄCHTER, E.W.; CASTILHOS, G. M.; MORAES, F. G.; CALAZANS, N. L. V. A *“Spectrum of MPSoC Models for Design Space Exploration and Its Use”*. In: RSP 2012, Tampere, Finland. (International Workshop on Rapid System Prototyping), 2012. (QUALIS B2) [PET12].

## 7.2. Trabalhos Futuros

Como trabalhos futuros enumeram-se as seguintes atividades:

- Incluir métricas de desempenho, como latência e energia de comunicação. Mesmo eles sendo proporcionais ao número de *hops*, sua avaliação é importante para corroborar a abordagem de gerenciamento distribuído.
- Adicionar parâmetro de QoS no monitoramento do sistema, como vazão, para adaptar dinamicamente as aplicações de acordo com a carga do sistema.
- Adicionar em um MPSoC com gerência distribuída uma rede dedicada para o fluxo de dados do gerenciamento.

## REFERÊNCIAS

- [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, pp. 760-765.
- [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.
- [ANA12] Anagnostopoulos, I; Bartzas, A.; Kathareios, G.; Soudris, D. “*A divide and conquer based distributed run-time mapping methodology for many-core platforms*”. In: DATE, 2012, pp. 111-116.
- [BER06] Bertozzi, S.; Acquaviva, A.; Bertozzi, D.; Poggiali, A. “*Supporting Task Migration in Multi-Processor Systems-on-Chip: A Feasibility Study*”. In: DATE, 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), 2010, pp. 26-35.
- [CHE00] Chen, T.S. “*Task migration in 2D wormhole-routed mesh multicomputers*”. Information Processing Letter, v. 73(3) 2000, pp.103-110.
- [COT98] Cote, G.; et al. “*H.263+: Video Coding at Low Bit Rates*”, IEEE Transactions on Circuits and Systems for Video Technology, v. 8(7), 1998, pp. 849-866.
- [CUI12] Cui, Y; Zhang, W; Yu, H. “*Decentralized Agent Based Re-Clustering for Task Mapping of Tera-Scale Network-on-Chip System*”. In: ISCAS, 2012, pp. 2437-2440.
- [DIC98] Dick, R; Rhodes, D; Wolf, W. “*TGFF: Task graphs for free*” In: CODES/CASHE, 1998, pp. 97–101.
- [FAT11] Fattah, Mohammad; Daneshtalab, Masoud; Liljeberg, Pasi; Plosila Juha. “*Exploration of MPSoC Monitoring and Management Systems*”. In: ReCoSoC, 2011, 3p.
- [FEI97] Feitelson D.; Rudolph L.; Schwiegelshohn U. “*Theory and practice in parallel job scheduling*”. Proceedings 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". In: ISSCC, 2010, pp. 108-109.
- [HU03] Hu, J.; Marculescu, R. "Energy-aware mapping for tile-based NoC architectures under performance constraints". In: ASP-DAC, 2003, pp. 233-239.
- [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: DATE, 2010, pp. 184-189.
- [JER05] Jerraya, A. A.; Wolf, W. "Multiprocessor Systems-on-Chips". Morgan Kaufmann Publishers Inc, 2005, 602p.
- [KOB11] Kobbe, Sebastian; Bauer, Lars; Lohmann, Daniel; Schroder-Preikschat, Wolfgang; Henkel, Jorg. "DistRM: Distributed Resource Management for On-Chip Many-Core Systems". In: CODES+ISSS, 2011, pp. 119-128.
- [LEE87] Lee, E.; Messerschmitt, D. G. "Static scheduling of synchronous dataflow programs for digital signal processing". IEEE Transactions on Computers, vol. 36(1), 1987, pp. 24-35
- [MAN11a] 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.
- [MAN11b] Mandelli, M.; Moraes, F. G. "Mapeamento Dinâmico de Aplicações para MPSoCs Homogêneos". Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 106p.
- [MAN12] Mandelli, M; Castilhos, G. M.; Moraes, F. G. "Enhancing Performance of MPSoCs through Distributed Resource Management". In: ICECS, 2012, 6p.
- [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.
- [MOR12] Moraes, F., G.; Madalozzo, G.; Castilhos, G; Carara, E. "Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs". In: ISCAS, 2012, pp. 644-647.
- [NAY07] Nayebi, A.; Meraji, S.; Shamaei, A.; Sarbazi-Azad, H. "XMulator: A Listener-Based Integrated Simulation Platform for Interconnection Networks". In: AMS, 2007, pp. 128-132.



- [OST12] Ost, L.; Varyani, S.; Mandelli, M.; Almeida, G.; Indrusiak, L.; Wachter, E.; Moraes, F.; Sassatelli, G. *“Enabling Adaptive Techniques in Heterogeneous MPSoCs based on Virtualization”*. ACM Transactions on Reconfigurable Technology and Systems, 2012, vol. 5(3), pp. 17:1-17:11.
- [PAS08] S. Pasricha, and N. Dutt. *“On-Chip Communication Architectures”*, Morgan Kaufman, 2008, 544p.
- [PAT11] David A. Patterson and John L. Hennessy. *“Computer Architecture, Fifth Edition: A Quantitative Approach”*. New York: Morgan Kaufmann Publishers, 2011, 856p.
- [PET12] Petry, C. A.; Wachter, E. W.; Castilhos, G. M.; Moraes, F. G.; Calazans, N. L. V. *“A Spectrum of MPSoC Models for Design Space Exploration and Its Use”*. In: RSP, 2012, 6p.
- [PLA11] Processador PLASMA. Capturado em: <http://plasmacpu.no-ip.org:8080/>, 2011.
- [PUS09] Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. *“Adaptive energy-aware latency-constrained DVFS policy for MPSoC”*. In: SOCC, 2009, pp. 89-92.
- [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] Tiler Corporation, *“Tile-GX Processor Family”*. Capturado em: [http://www.tiler.com/products/processors/TILE-Gx\\_Family](http://www.tiler.com/products/processors/TILE-Gx_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, 459p.
- [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.
- [WEI11] Weichslgartner, A.; Wildermann, S.; Teich, J. *“Dynamic decentralized mapping of tree-structured applications on NoC architectures”*. In: NOCs, 2011, pp. 201-208.
- [WOL04] Wolf, W. *“The Futere of Multiprocessors System-on-Chips”*. In: DAC, 2004, pp. 681-685

[WOL12] M. Wolf. "*Computers as components: principles of embedded computing system design*". New York: Morgan Kaufmann Publishers, 2012, 528p.