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

Controle Adaptativo para Atendimento a Requisitos de Aplicações em MPSoCs

GUILHERME AFONSO MADALOZZO

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

Porto Alegre

2013

Dados Internacionais de Catalogação na Publicação (CIP)

M178c Madalozzo, Guilherme Afonso
Controle adaptativo para atendimento a requisitos de aplicações em MPSoCs / Guilherme Afonso Madalozzo. – Porto Alegre, 2013.
68 f.

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

1. Informática. 2. Mulitprocessadores. 3. Algoritmos. 4. Monitoramento de Tarefas. I. Moraes, Fernando Gehm. II. Título.

CDD 004.35

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

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

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Controle Adaptativo para Atendimento a Requisitos de Aplicações em MPSoCs" apresentada por Guilherme Afonso Madalozzo como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 11/03/2013 pela Comissão Examinadora:

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

Prof. Dr. Ney Laert Vilar Calazans – — PPGCC/PUCRS

Prof. Dr. Ricardo Augusto da Luz Reis – — UFRGS

Prof. Dr. Everton Alceu Carara– — UFSM

Homologada em 15/04/2013, conforme Ata No. 006 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
www.pucrs.br/facin/pos

AGRADECIMENTOS

O caminho percorrido para atingir o objetivo de finalizar esta Dissertação de Mestrado não foi simples. Neste caminho muitos desafios e obstáculos foram encontrados, e não teria sido possível superá-los sozinho. Por isso, dedico esta parte de minha Dissertação para expressar o quão agradecido sou por cada um que ao meu lado esteve, seja pessoalmente ou em pensamento, mandando vibrações positivas para que esse sonho fosse concretizado. A todos, o meu muito obrigado.

Em especial, à **Deus**, por iluminar este caminho nos momentos mais escuros. Por me dar inspiração, tranquilidade e força para superar os desafios. Falando em Deus, gostaria de dedicar esta conquista aos meus falecidos avós: Vó **Ires** e Vô **Dirceu**, Vó **Jandira** e Vô **Waldemar**. Foram exemplos de pessoas para minha formação pessoal. Em muitos momentos, a eles pedi força e iluminação, pois ninguém melhor do que meus queridos avós para acompanhar e abençoar meu caminho.

Aos meus pais, **Paulo** e **Janaína**, por sempre me apoiar e incentivar no objetivo de alcançar caminhos cada vez mais distantes. Por, mesmo de longe, estarem a meu lado nestes dois anos de mestrado. Por sempre acreditarem e apostarem em mim. Se cheguei até aqui, devo a vocês. Para mim, o conceito de *família* vai além de parentesco, é a união de pessoas que se amam e que se cuidam, e essa é a minha família. Dessa forma, agradeço meus tios(as) e primos(as) que torceram por mim e me apoiaram. Além dessas pessoas, devo fazer um agradecimento especial para minha irmã **Camile**, meu cunhado **Fabrício** e para minha sobrinha do coração, **Julia**. Por torcerem tanto por mim e, além disso, alegrar esse caminho que se estenderá por mais alguns anos. Minha família é a base para tudo em minha vida, e a eles dedico esta Dissertação.

Dentro dos agradecimentos para minha família deixei uma pessoa de lado, mas não por desconsidera-la parte desta família, muito pelo contrário, foi acolhida nesta família. Apenas gostaria de dedicar um parágrafo especial a ela, minha namorada **Silvana**. Esta conquista só foi possível porque você esteva ao meu lado, me dando força, tranquilidade e carinho. Juntos viemos de longe, juntos viemos viver o desconhecido, juntos chegamos até aqui e juntos iremos adiante. Agradeço a ti por toda a paciência e compreensão durante este período. Muito obrigado por todo amor que dedica a mim, serei eternamente grato, te amando e te respeitando. Agradeço, também, minha sogra **Inês** e meu sogro **Alcides** por rezar e torcer tanto por nós.

Ao Professor **Fernando Moraes**, por aceitar o desafio de orientar este desorientado. Muito obrigado por me ensinar desde conceitos simples até complexos funcionamentos de um sistema embarcado, que em minha formação não tive contato. Por ter compartilhado seu tempo e conhecimento para que a realização deste trabalho fosse possível. Sou grato por me aceitar como aluno de mestrado e agora doutorado.

Agradeço aos professores do PPGCC pela transmissão de seus conhecimentos. Agradeço aos funcionários da PPGCC pela dedicação em nos manter sempre informados de toda burocracia e prazos. Um agradecimento especial ao Professor Ney, que em suas avaliações enriqueceu esta dissertação. Aos colegas do GAPH, Wachter, Ruaro, Castilhos, Mandelli, Matheus, LHeck, Jeronimo, GHeck e a todos os demais. E ao CNPq por ter possibilitado e financiado esta pesquisa.

"Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível." Charles Chaplin.

CONTROLE ADAPTATIVO PARA ATENDIMENTO A REQUISITOS DE APLICAÇÕES EM MPSOCS

RESUMO

A capacidade de integração em sistemas embarcados acompanha a tendência da Lei de Moore, a qual prevê que a cada dezoito meses o número de transistores em circuitos integrados dobra, enquanto seu custo permanece constante. Outra observação importante em sistemas embarcados é que aplicações com mais de um processador estão cada vez mais presentes no mercado. Estes dispositivos com diversos elementos de processamento são denominados MPSoCs (do inglês, *Multiprocessor Sytem-on-Chip*). Os MPSoCs permitem o desenvolvimento de sistemas complexos, com alto desempenho. Para que um MPSoC atenda às restrições das aplicações nele executadas, técnicas de gerência e adaptabilidade de recursos devem ser pesquisadas e desenvolvidas.

O presente trabalho apresenta o desenvolvimento e avaliação de técnicas de controle adaptativo para atendimentos a requisitos de aplicações executando em MPSoCs. Para efetuar o controle do MPSoC utiliza-se o mecanismo de monitoramento das aplicações. A técnica de monitoramento analisa os requisitos das aplicações, em tempo de execução, verificando possíveis violações nestes requisitos, como vazão e latência. O monitoramento é o gatilho para a execução das técnicas adaptativas desenvolvidas no escopo deste trabalho: alteração dinâmica na prioridade de escalonamento de tarefas e migração de tarefas.

Para avaliar as técnicas propostas, foi utilizado a plataforma HeMPS com gerência de recursos centralizada e distribuída. Os resultados mostram que, independente da gerência de recursos que se utiliza, centralizada ou distribuída, as técnicas de adaptabilidade proveem redução de latência e jitter, sem comprometimento do tempo total de execução das aplicações. Com a execução das técnicas de adaptabilidade, o tempo total de execução da aplicação principal não é penalizado, nos casos de teste, melhorando-se em até 7%.

Palavras-Chave: MPSoC, NoC, gerenciamento de aplicações, controle adaptativo, técnicas de adaptabilidade, monitoramento de tarefas, migração de tarefas, escalonamento.

ADAPTIVE CONTROL TO MEET APPLICATIONS' CONSTRAINTS IN MPSOCS

ABSTRACT

The growing number of manufactured transistors in embedded systems follows the trend of Moore's Law, which states that every eighteen months the number of transistors on integrated circuits doubles, while its cost remains constant. Another important issue in embedded systems is that applications with more than one processor are increasingly present in market. These devices with several processing elements are named MPSoCs (Multiprocessor System-on-Chip). MPSoCs enables the development of complex systems, together with high performance. Applications executing in MPSoC have constraints to be respected. To meet these constraints, management techniques and resources adaptability should be researched and developed.

This work presents the development and evaluation of adaptive management techniques that enable applications executing in MPSoCs to meet their performance requirements. The MPSoC management uses monitoring techniques, which evaluate applications constraints, as throughput and latency. When violations are detected by the monitoring infrastructure, adaptive techniques are executed. In the scope of this work, two techniques were developed: dynamic change in the priority scheduling of tasks and task migration.

The evaluation of the proposed techniques is carried out using the HeMPS MPSoC, with centralized and distributed resource management. Results show that, regardless the resource management technique adopted, the proposed adaptive techniques decrease latency and jitter, without affecting the total execution time of applications. With performed adaptive techniques the total execution time wasn't penalized, in presented experiments increased 7%.

Keywords: MPSoC, NoC, application management, adaptive control, adaptive techniques, task monitoring, task migration, scheduling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
DMA	Direct Memory Access
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
GMP	Global Master Processor
HeMPS	HERMES Multiprocessor System
IP	Intellectual Property ou Internet Protocol
LMP	Local Master Processor
MP	Master Processor
MPSoC	Multiprocessor System-on-Chip
MSA	Monitoring Service Access Point
NI	Network Interface
NoC	Network-on-Chip
PE	Processing Element
RISC	Reduced Instruction Set Computer
RM	Resource Manager
SoC	System-on-Chip
SP	Slave Processor

SUMÁRIO

1 INTRODUÇÃO

Gordon Moore, em 1965, constatou que a cada 18 meses o número de transistores em circuitos integrados dobra, enquanto seu custo permanece constante. Tortato e Hexsel [TOR09] comentam que a capacidade de sistemas embarcados vem acompanhando a tendência da Lei de Moore e que as aplicações com mais de um processador estão cada vez mais presentes no mercado. Como consequência do aumento do número de transistores, tornou-se possível desenvolver sistemas completos em um único circuito integrado, denominados de sistema em chip (SoC – *System on Chip*) [JER05], ou MPSoC (*Multiprocessor System on Chip*) quando o SoC contém diversos elementos de processamento (PEs – *Processing Elements*). Segundo [AGU08], sistemas embarcados estão cada vez mais presentes em produtos de bens de consumo como televisores, micro-ondas, telefones celulares e *tablets*, e a principal característica nestes produtos é executar funcionalidades específicas.

Um MPSoC pode ser projetado para uma determinada aplicação ou conjunto de aplicações, ou ser de propósito genérico, tal qual os FPGAs (*Field-Programmable Gate Array*) são hoje para o projeto de circuitos digitais. No primeiro caso, as decisões arquiteturais são realizadas em tempo de projeto, podendo-se otimizar elementos como a NoC ou a hierarquia de memória para os requisitos das aplicações alvo. No segundo caso, MPSoCs de propósito genérico, estes devem ser capazes de executar diferentes aplicações e classes de aplicações. Para que isto seja possível, os MPSoCs devem prover técnicas de adaptabilidade para prover qualidade de serviço. Exemplos de MPSoCs de propósito genérico incluem os dispositivos TILERA [TIL09] e INTEL [VAN07].

O foco da presente Dissertação é no projeto de MPSoCs de propósito genérico e no desenvolvimento de técnicas de adaptabilidade para garantir o atendimento às restrições das aplicações. As técnicas de adaptabilidade incluem, por exemplo: (*i*) migração de tarefas para balanceamento de carga, mantendo as tarefas comunicantes da mesma aplicação sendo executas próximas uma das outras; (*ii*) monitoramento de tarefas, para verificar o atendimento às restrições das aplicações, como vazão e latência; (*iii*) controle distribuído, para efetuar gerenciamento de recursos; (*iv*) DVFS, alteração dinâmica de voltagem e frequência; (*v*) chaveamento por circuito, reserva totalmente o canal físico entre o par comunicante (processadores que executam tarefas comunicantes), possibilitando máxima vazão; (*vi*) alterações na prioridade de comunicação.

Para prover adaptabilidade em tempo de execução, Fattah et al. [FAT11] comentam que muitos trabalhos propõem estruturas de monitoramento que observam o estado dos sistemas em tempo de execução em termos de: transações, temperatura, sobrecarga na rede, falhas, defeitos, etc. Para que um sistema possa efetuar o monitoramento das aplicações, é necessário coletar as características das aplicações através de *profiling*

(perfil). Baseado nos dados desta coleta na etapa de monitoramento, o processador monitor é capaz de efetuar tomadas de decisão, tais como: alteração da prioridade de comunicação, migração de tarefas, escalonamento, DVFS (*Dynamic Voltage and Frequency Scaling*), e aumento da fatia de tempo alocada em cada processador para a execução das tarefas.

A adaptabilidade em um MPSoC pode ser vista como um sistema de controle em malha fechada. A Figura 1 ilustra os três passos do processo: monitoramento, diagnóstico e ação. O monitoramento é efetuado em nível de tarefa em um dado elemento de processamento (PE). Os dados monitorados são enviados para um dado PE do sistema, o qual é responsável pela decisão de qual ação deve ser tomada em caso de não atendimento a determinados requisitos de aplicação. A ação de diagnóstico pode ser centralizada ou distribuída. Finalmente temos a tomada de ação, que envolve alguma das técnicas mencionadas anteriormente. O processo é cíclico, executado de forma contínua ao longo da execução das aplicações.

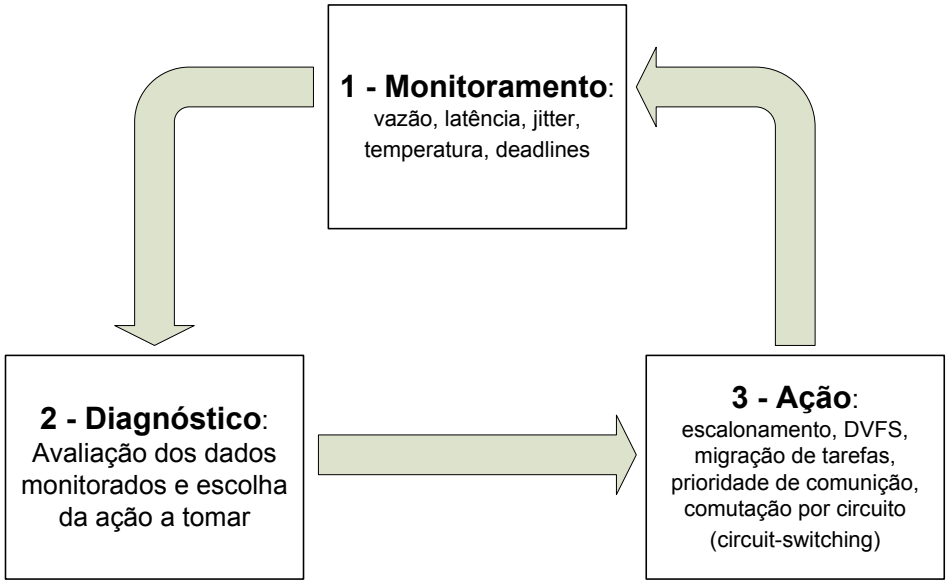

Figura 1 – Ações para prover adaptabilidade em MPSoCs.

Escalabilidade é outro ponto importante que se deve levar em consideração no projeto de MPSoCs. MPSoCs podem executar diversas aplicações em paralelo com carga de trabalho dinâmica (inserção de novas aplicações no MPSOC em tempo de execução). No nível arquitetural a utilização de redes intra-chip (NoCs) tem se mostrado uma alternativa promissora em MPSoCs, dado que estas permitem múltiplas comunicações simultâneas [PAS08]. No nível de gerência de recursos, o gerenciamento distribuído do MPSoC em regiões virtuais, denominadas *clusters*, é também apontada na literatura como uma técnica para prover escalabilidade em sistemas com elevado número de elementos de processamento [FAT11][SHA11].

1.1 Objetivos

A presente Dissertação possui os seguintes objetivos estratégicos:

- Domínio da tecnologia de projeto de MPSoCs que utilizam NoCs como meio de comunicação;
- Domínio de técnicas adaptativas para garantir o atendimento às restrições das aplicações;
- Domínio de técnicas de monitoramento de tarefas;
- Domínio de técnicas de gerenciamento de recursos em MPSoCs.

Os objetivos específicos da presente Dissertação incluem:

- Desenvolver uma técnica de migração de tarefas em MPSoCs;
- Desenvolver escalonamento baseado em prioridades para tarefas executando em MPSoCs;
- Desenvolver monitoramento para controle de *deadlines* de tarefas;
- Utilizar a migração de tarefas para a desfragmentação do sistema, técnica esta denominada de *reclustering*.

1.2 Contribuições da Dissertação

A presente Dissertação tem por contribuição o desenvolvimento de duas técnicas de adaptabilidade no projeto de MPSoCs: alteração dinâmica na prioridade de escalonamento de tarefas e migração de tarefas. Ambas as técnicas são controladas por monitores de desempenho. A migração de tarefas é empregada em dois cenários: gerência centralizada de recursos e gerência distribuída de recursos. No primeiro cenário, gerência centralizada, o objetivo é aumentar o desempenho das aplicações. No segundo cenário, gerência distribuída, acrescenta-se ao objetivo de aumento de desempenho a desfragmentação do MPSoC, através da aproximação de tarefas comunicantes.

1.3 Estrutura do Documento

A estrutura do documento está organizada como segue. No segundo Capítulo apresenta-se uma visão do estado da arte nos temas nos quais a presente Dissertação contribui: monitoramento de tarefas, escalonamento, controle distribuído e migração de tarefas em MPSoC. O Capítulo 3 apresenta de forma sucinta a arquitetura de referência adotada – MPSoC HeMPS [CAR09]. O Capítulo 4 detalha a técnica de monitoramento implementada. Os Capítulos 5 e 6 apresentam as técnicas de escalonamento por prioridade e a migração de tarefas, respectivamente. O Capítulo 7 apresenta os resultados obtidos e a respectiva discussão dos mesmos. O Capítulo 8 encerra esta Dissertação, com as conclusões e direções para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Neste Capítulo apresenta-se o estado da arte nos temas relacionados ao presente trabalho. Analisaremos as pesquisas realizadas para o monitoramento de tarefas, escalonamentos eficientes em MPSoC, controle distribuído e migração de tarefas.

A estrutura do Capítulo está dividida conforme os assuntos relacionados ao trabalho. Na primeira Seção apresenta-se o estado da arte para o monitoramento de tarefas. Na segunda Seção tem-se as pesquisas relacionadas a escalonamentos em MPSoC. Na terceira Seção apresenta-se o estado da arte para controle distribuído. Na quarta Seção apresentam-se os trabalhos relacionados à migração de tarefas. Por fim, a última Seção deste Capítulo, tem-se as considerações finais e uma tabela comparativa entre os trabalhos analisados.

2.1 Monitoramento de Tarefas

Nesta seção apresentam-se pesquisas referentes ao monitoramento de tarefas em MPSoCs. O objetivo de uma técnica de monitoramento de tarefas é para efetuar o controle das aplicações. [MAT10] definem que a adaptabilidade dos MPSoCs necessitam desse tipo de mecanismos.

2.1.1 Stan et al.

Stan et al. [STA11] apresentam um método para controle de *deadlines* melhorando a qualidade de serviço em aplicações multimídia. O método consiste na detecção de violações de tempo em um MPSoC. Os PEs do MPSoC contém uma estrutura de monitoramento que verifica a infraestrutura de comunicação. Todos os dados transferidos são armazenados e seus tempos são comparados a um comportamento de referência esperado. A estrutura de monitoramento é implementada em um dispositivo que é mapeado no espaço de endereçamento do processador, como podemos ver na Figura 2.

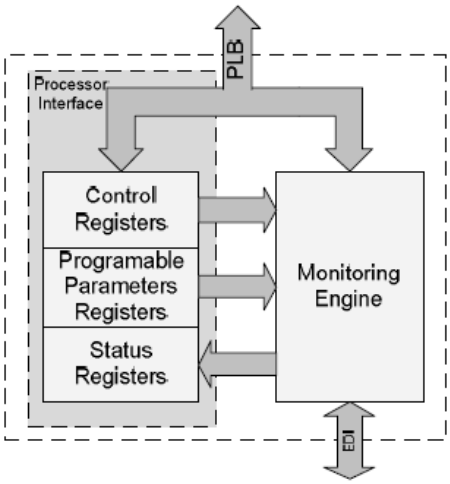

Figura 2 – Arquitetura da estrutura de monitoramento. Fonte: [STA11]

A estrutura de monitoramento implementa um mecanismo *watchdog* que conta a quantidade de eventos que são executados antes do tempo limite, ou entre dois limites de tempo ou depois de um tempo limite. Os limites de intervalo de tempos utilizados para verificar um determinado evento são armazenados em parâmetros programáveis, que são configurados nas aplicações; esses valores são computados em tempo de projeto. O tempo limite pode ser alterado pelas variações de frequências (causadas pela variação de temperatura no ambiente operacional) ou por algumas falhas de *hardware*. Nesta pesquisa, os autores não apresentam os resultados obtidos, apenas explicam o funcionamento de seu mecanismo de monitoramento de tarefas em MPSoC.

2.1.2 Ciordas et al.

Ciordas et al. [CIO06] apresentam diversas alternativas para monitoramento de NoCs e avaliam qual é o impacto dessas técnicas no projeto de NoCs. Essas alternativas variam do uso de interconexões físicas, separadas para dados de aplicações e dados de monitoramento. Para cada alternativa os autores avaliaram o custo da área das interconexões, o impacto de modificações no fluxo do projeto e a reusabilidade dos recursos de *debug* para tráfego de dados das aplicações.

O monitoramento da NoC é explorado em três alternativas: *(i)* interconexões físicas separadas; *(ii)* interconexões físicas comuns com recursos físicos separados; *(iii)* interconexões físicas comuns com recursos físicos compartilhados. Para a avaliação das alternativas de monitoramento, os autores apresentam duas características: *(i) probes*, componentes de *hardware* que monitoram o tráfego dos dados na rede; *(ii)* serviço de monitoramento de acesso (MSA – *Monitoring Service Access Point*), efetua a reconfiguração em tempo de execução baseado nos dados enviados pelos *probes*.

Na Figura 3 os autores apresentam as três alternativas de monitoramento. Na Figura 3(a) temos uma NoC sem monitoramento. Na Figura 3(b) é apresentada uma NoC com conexões físicas separadas para fazer o tráfego dos dados monitorados pela NoC, enviados do *probe* para o MSA e vice-versa. Na Figura 3(c) é apresentada uma NoC com conexões físicas comuns e recursos físicos separados para efetuar um monitoramento separado em subredes, sem a necessidade de adicionar roteadores para o monitoramento, porém são adicionados *links* e novas portas aos roteadores da NoC. Na Figura 3(d) é apresentada uma NoC com conexões físicas comuns e com acesso compartilhado aos recursos físicos para tráfego de dados e para o tráfego de monitoramento. Ambos utilizam todos os recursos compartilhados da NoC, mas o tráfego de dados e o tráfego de monitoramento são mantidos em separado, dessa forma uma NoC Virtual para monitoramento é criada.

As alternativas apresentadas na Figura 3(b) e na Figura 3(c) requerem um maior número de recursos de *hardware* e de conexões que a alternativa apresentada na Figura 3(d), porém separam os dados de monitoramento dos dados das aplicações. A escolha do

projetista deve ser baseada no compromisso entre custo de área e desempenho do monitoramento.

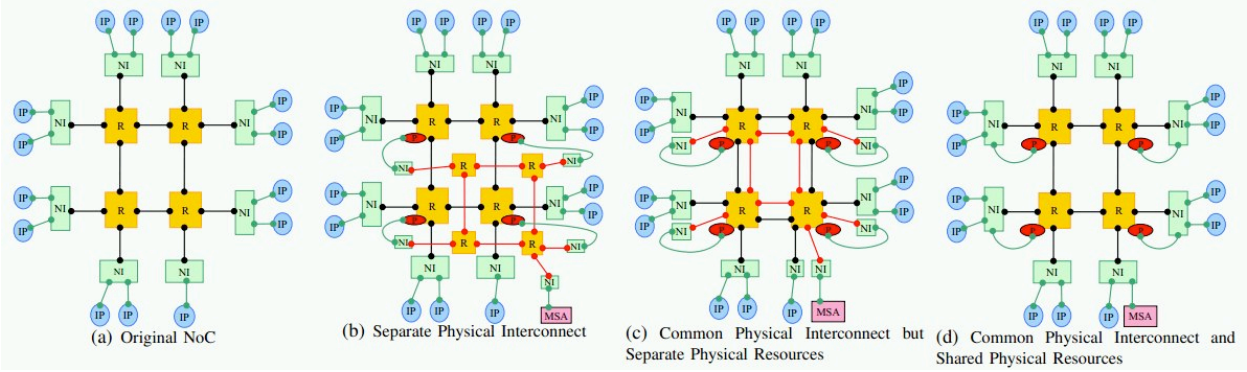

Figura 3 – Versões de monitoramento. Fonte: [CIO06]

2.1.3 Matos et al.

Matos et al. [MAT10] apresentam um mecanismo para controle de redimensionamento dinâmico de *buffer* para cada canal de entrada do roteador, que é feito através de um monitoramento de tráfego na rede, em tempo de execução. Além disso, a configuração dinâmica da profundidade do *buffer* é feita sem qualquer pausa ou interrupção do sistema.

O bloco de controle de profundidade do *buffer* foi implementado para cada canal de entrada do roteador, sendo a arquitetura com o bloco de controle apresentada na Figura 4. O bloco *Input Buffer* contém a FIFO usada no canal de entrada e é responsável por armazenar os *flits*. A FIFO é controlada pelo bloco *Input Channel Controller* que realiza o controle de fluxo, e o roteamento dos *flits* do canal de entrada. O *Buffer Depth Controller* é o bloco com o controle proposto pelo trabalho.

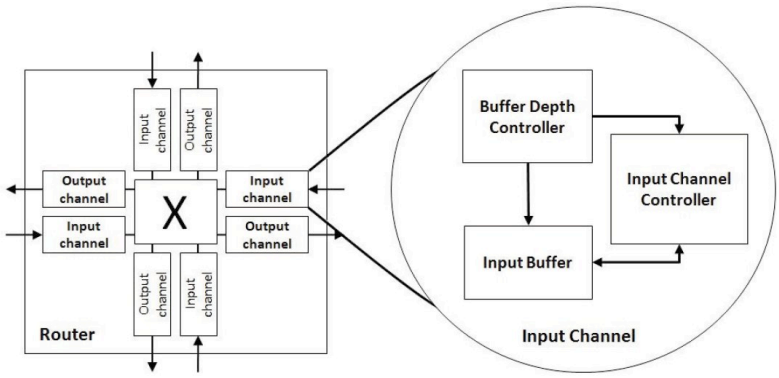

Figura 4 – Arquitetura do roteador com controle de profundidade do buffer. Fonte: [MAT10]

O bloco de controle de profundidade do *buffer* é composto por quatro outros blocos: *(i)* monitor, observa o tráfego de dados no canal; *(ii)* integrador, calcula a nova profundidade do *buffer* de acordo com o tráfego de dados monitorado e a aplicação; *(iii)* alocação de *slots* de *buffer*, implementa um protocolo para distribuir os *slots* do *buffer* para cada canal de acordo com a profundidade calculada pelo integrador; *(iv)* decisão de

redimensionamento, verifica quando cada canal irá permitir a troca da profundidade do *buffer*. Esse bloco de controle de profundidade do *buffer* é apresentado na Figura 5.

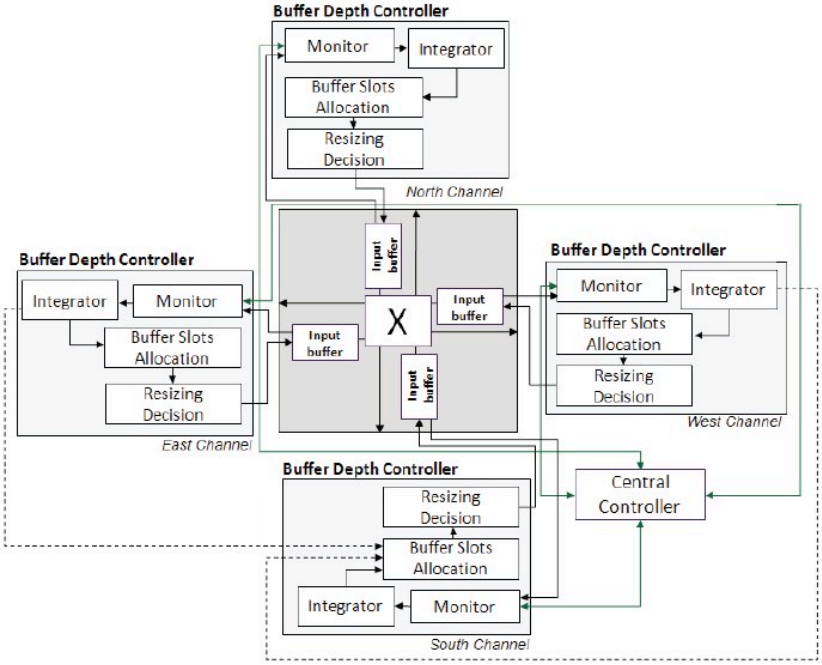

Figura 5 – Arquitetura do roteador com o fluxo de controle detalhado. Fonte: [MAT10]

Com a arquitetura proposta, os autores conseguiram diminuir a latência em 80% e dobrar a vazão, com a alteração dinâmica da profundidade do *buffer*.

2.2 Escalonamento

Nesta seção apresentam-se pesquisas referentes a algoritmos de escalonamento de tarefas em MPSoCs.

2.2.1 Coskun et al.

Coskun et al. [COS07] apresentam um algoritmo de escalonamento de tarefas com o objetivo de atingir uma distribuição térmica uniforme em MPSoCs. Os autores projetaram e avaliaram políticas de escalonamento no nível de sistema operacional. Também, apresentam duas técnicas que podem ser aplicadas para o controle da temperatura em MPSoCs. A primeira técnica é a migração dinâmica de *threads*. É um método de gerenciamento térmico em MPSoC que migra *threads* de um processador quente para um processador frio. Na implementação dessa técnica a limiar de temperatura para efetuar a migração foi configurada para 85°C, que é considerada uma temperatura crítica em muitos sistemas. A segunda técnica é a de gerenciamento térmico para dimensionamento dinâmico de voltagem e frequência quando atingida a temperatura limiar. Essa técnica diminui a temperatura do processador reduzindo o consumo de energia. Todas as aplicações são executadas com frequência máxima, a menos que a temperatura crítica (85°C) seja atingida. Se um processador atinge esta temperatura, o nível de voltagem do processador é reduzido para diminuir a frequência até que a aplicação atual termine.

Além dessas duas técnicas, os autores apresentam duas políticas de escalonamento usando como função custo a temperatura. O primeiro algoritmo envia tarefas para processadores mais frios. O segundo algoritmo efetua o envio de tarefas analisando o histórico de temperaturas do processador e pode ser implementada em sistemas reais.

Essa pesquisa demonstrou que técnicas que fazem uso de medições de temperatura para atingir um valor térmico ideal e para otimizar a temperatura não afetam o desempenho do sistema. Os autores avaliaram o desempenho do sistema utilizando as técnicas apresentadas, demonstrando que o desempenho do sistema não foi comprometido com o uso dessas técnicas.

2.2.2 Tafesse et al.

Tafesse et al. [TAF11] apresentam dois algoritmos de escalonamento, um para MPSoCs baseados em barramento e outro para MPSoCs baseados em NoC: *(i)* algoritmo de escalonamento objetivando maximizar o desempenho (para MPSoCs baseados em barramento); *(ii)* algoritmo de escalonamento baseado no volume de tráfego (para MPSoCs baseados em NoC).

O primeiro algoritmo apresentado é o escalonador para barramentos. Esse algoritmo utiliza um índice de desempenho, que quantifica o desempenho do sistema considerando o valor médio do tempo de execução por tarefa, utilização do processador, vazão, uso do buffer, e a energia, determinando o custo da tarefa por processador. O escalonador irá decidir por escalonar a tarefa no processador que tiver a melhor função custo de desempenho.

O segundo algoritmo apresentado pelos autores é o escalonamento que utiliza o volume de tráfego como função custo. A principal contribuição dessa técnica é que o mapeamento da aplicação e o processo de escalonamento são controlados juntos. Esse algoritmo calcula a latência da comunicação para escalonar e mapear as tarefas.

Os resultados obtidos na avaliação dos dois algoritmos de escalonamento mostraram que projetar técnicas de escalonamento, buscando resolver problemas como otimização de temperatura, balanceamento de cargas e gerenciamento térmico, resulta em melhores desempenhos para o sistema do que utilizando algoritmos de escalonamento clássicos.

2.2.3 Zhaoguo et al.

Zhaoguo et al. [ZHA09] apresentam um algoritmo de escalonamento para diminuir a temperatura e economizar energia em MPSoCs. O algoritmo atribui alta prioridade para os processadores com baixa temperatura.

O sistema opera em dois modos: modo ativo e modo ocioso. Durante o modo ativo as tarefas podem ser escalonadas e executadas. Durante o modo ocioso as tarefas são

preemptadas e o sistema passa a economizar energia. Inicialmente, os processadores operam em modo ocioso. Então, a decisão do escalonamento é baseada na temperatura de todos os processadores do MPSoC, e é escolhido o processador com menor temperatura.

Para apresentar os benefícios do algoritmo proposto, os autores efetuaram uma comparação da eficiência do controle de temperatura da técnica proposta com a técnica de alteração dinâmica de voltagem (DVS). Com isso, concluem que o algoritmo proposto atinge o valor de temperatura ideal (34°C) em todos processadores do MPSoC, mantendo-os frios. Já o DVS, que diminui a voltagem e a frequência do chip, irá perder energia durante o longo período de execução, o que faz a temperatura subir rapidamente.

O trabalho apresentado pelos autores mostra a importância de efetuar um controle da temperatura do chip considerando as perdas de energia. Com a proposta, os autores conseguiram efetuar uma economia de 70% de energia no sistema.

2.3 Controle Distribuído

Nesta Seção apresentam-se pesquisas referentes ao gerenciamento distribuído de tarefas em MPSoCs. Essas técnicas têm por objetivo efetuar o monitoramento de tarefas de forma distribuída, como apresentado em [FAT11][KOB11][SHA11]. Segundo [FAT11], o gerenciamento distribuído pode garantir ganhos de desempenho, tolerância a falhas e escalabilidade. [SHA11] e outros comentam que o gerenciamento distribuído é mais escalável e eficiente.

2.3.1 Fattah et al.

Fattah et al. [FAT11] apresentam uma pesquisa de monitoramento de sistemas adaptativos, para facilitar o gerenciamento do MPSoC em diferentes aspectos, tais como: consumo de energia, desempenho, tolerância a falhas e sistemas reconfiguráveis. Nesta pesquisa, os autores apresentam uma abordagem com a combinação de métodos centralizados e distribuídos. Ou seja, o gerenciamento do sistema é executado de forma que alguns dados monitorados e relatórios de estado sejam enviados para o gerenciador de alto nível que retorna comandos para serem executados.

Como se pode ver na Figura 6, em destaque, o roteador e seus componentes locais compõem um célula. De acordo com a demanda do sistema, uma célula pode conter diferentes capacidades de monitoramento, tais como: sensores de temperatura, monitores de consumo de energia e detectores de falhas. Cada célula tem seu próprio gerenciador, que em função dos seus mecanismos de monitoramento, relata as condições da célula para um gerenciador de alto nível (não detalhado no artigo), recebendo comandos do mesmo.

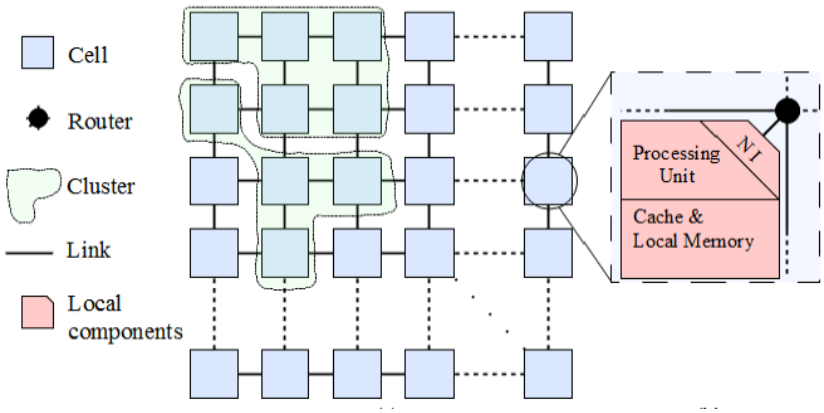

Figura 6 – NoC com agrupamento de tarefas por regiões. Fonte: [FAT11]

Um conjunto de células forma um grupo (*cluster*) e são gerenciadas por um gerenciador do grupo (nível intermediário de gerenciamento). Diferentes políticas de agrupamento podem ser aplicadas: uma aplicação pode conter múltiplos grupos, onde cada um executa um conjunto de tarefas ou cada grupo pode executar tarefas de várias aplicações.

Assim, este trabalho propõe um gerenciamento hierárquico de três níveis, conforme apresentado na Figura 7: (*i*) no nível de *célula* (local); (*ii*) no nível de grupo (intermediário); (*iii*) alto nível (global). O gerenciador de alto nível é responsável pelo controle global e pela coordenação dos gerenciadores de grupo. O gerenciador no nível de célula é executado no sistema operacional e é carregado na criação de uma nova tarefa, quando invocada. Os autores comentam que o gerenciamento hierárquico pode ser visto como uma solução de 'dividir e conquistar' para futuros sistemas *multicores*.

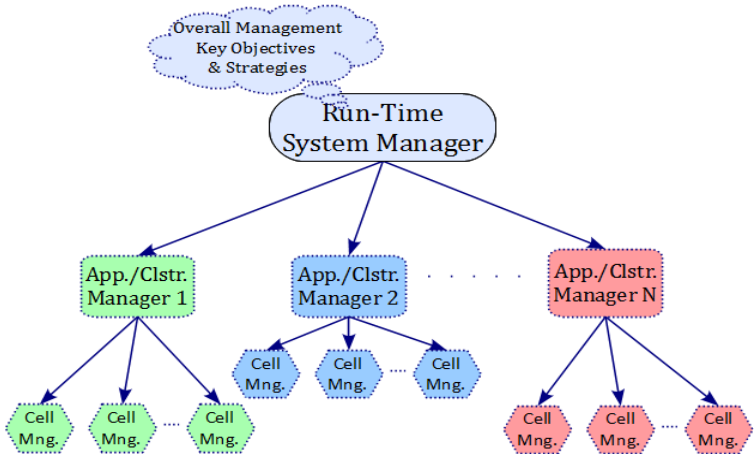

Figura 7 – Gerenciamento Hierárquico de MPSoC proposto por [FAT11].

2.3.2 Shabbir et al.

Shabbir et al. [SHA11] apresentam um comparativo entre duas versões de gerenciamento de recursos distribuídos: *Credit Based* e *Rate Based*. Os gerenciadores foram desenvolvidos para controlar um grande número de processadores executando aplicações concorrentes.

A primeira versão do gerenciador de recursos proposto pelos autores é o *Credit Based*, que permite o uso de aplicações que tenham restrições rígidas de desempenhos; onde os desempenhos não podem ter valores alterados, mesmo que os recursos disponíveis sejam capazes de melhorar seu desempenho. A segunda versão do gerenciados de recursos proposto pelos autores é o *Rate Based*, que é ideal para aplicações que podem ter mais desempenho do que um valor mínimo, caso os recursos estejam disponíveis.

A arquitetura modelada para a proposta consiste em processadores conectados em uma NoC e os gerentes de recursos consistem em um controlador de admissão. Os controladores de admissão são responsáveis pela avaliação de restrições de tempo de novas aplicações utilizando os recursos disponíveis. Caso os recursos disponíveis do MPSoC não atendam os requisitos da nova aplicação, então, o controlador de admissão rejeita o pedido e a aplicação pode solicitar serviços com um menor nível de qualidade.

Na Figura 8 os autores apresentam diagramas de gerenciamento de recursos centralizados e sua proposta de gerenciamento distribuído. O modelo de gerenciamento centralizado monitora a vazão de cada aplicação e compara com sua vazão desejada. O gerente centralizado (RM) tem que monitorar e controlar todas as aplicações e seus desempenhos, ocasionando problemas de escalabilidade devido ao tempo de monitoramento. Para resolver esse problema, os autores apresentam duas versões de gerenciamento distribuído. O gerenciamento distribuído procura minimizar o envolvimento do gerenciador central no processo e investe mais inteligência no processador local.

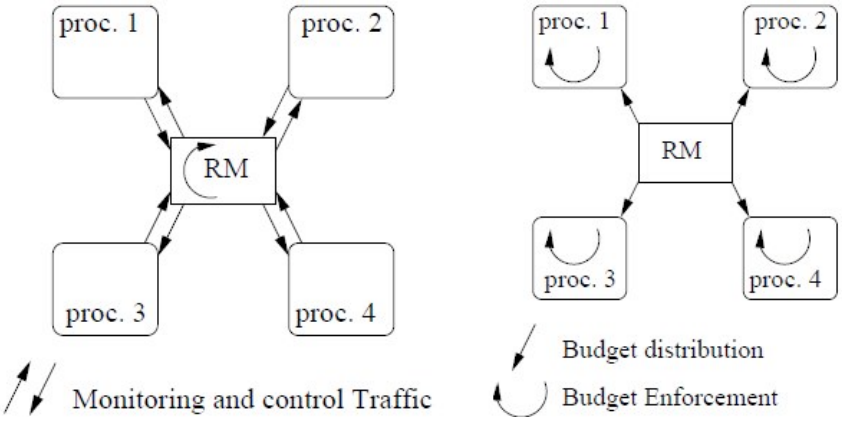

Figura 8 – Gerenciamento de recursos: centralizado x distribuído. Fonte: [SHA11]

Nas duas versões de gerenciamento de recursos distribuído há um controlador de admissão central conectados aos núcleos de processamento em uma NoC. O controlador de admissão central é uma interface que calcula os créditos, sendo estes créditos distribuídos entre os processadores do sistema. Os árbitros dos processadores locais aplicam os créditos para que as restrições de vazão das aplicações sejam satisfeitas.

No gerenciador de recursos por *Credit Based* o controlador de admissões envia os créditos dos processadores conforme o mapeamento das tarefas nos processadores.

Para fazer cumprir esses créditos, cada processador tem um *kernel* que armazena esses créditos em um contador. Depois que os créditos acabam os contadores são recarregados com seus valores recebidos pelo controlador central e o processo continua.

No gerenciador de recursos por *Rate Based* cada processador tem seu arbitro local. O controlador de admissão do *rate based* calcula os créditos da mesma forma que o *credit based*. O controlador de admissão envia os créditos para cada arbitro local de cada processador. Os árbitros locais recebem os créditos e executam as tarefas de modo que se existir recursos disponíveis no sistema a tarefa é executada em uma taxa maior do que a desejada, terminando mais rapidamente.

Os experimentos mostram que gerenciadores de recursos distribuídos são mais escaláveis, são mais eficientes com aplicações dinâmicas e requerem menos armazenamento. O gerenciamento Credit *Based* é mais eficaz para fazer cumprir as restrições de vazão, já o *Rate Based* é mais eficaz para aplicações que podem fazer uso de todos os recursos disponíveis do sistema.

2.3.3 Kobbe et al.

Kobbe et al. [KOB11] apresentam um esquema de gerenciamento de recursos distribuído em MPSoC, chamado DistRM. Essa proposta foi projetada para ser um sistema sem qualquer sincronização global ou comunicação global. Para alcançar escalabilidade, os autores desenvolveram os princípios de sistemas multi-agentes para efetuar o gerenciamento de recursos. Cada aplicação do sistema tem um agente dedicado para gerenciar os recursos.

A computação necessária de cada agente é desempenhada nos mesmos PEs das aplicações, mas a computação de todo o gerenciamento global de recursos é distribuída em todos os PEs do MPSoC. Cada agente visa aumentar a aceleração de suas aplicações procurando por outros PEs no sistema que possam ser usados. Portanto, ele usa a capacidade de aplicações maleáveis (o número de PEs atribuídos a um agente pode ser alterado durante a sua execução [FEI97]) para poder adaptar PEs adicionais. Quando existir mais de uma aplicação executando no sistema, os PEs disponíveis devem ser compartilhados dentre as diferentes aplicações.

Os recursos não são mais gerenciados em uma região central, mas em muitas regiões espalhadas em todo o chip. Adicionalmente, as comunicações necessárias para gerenciar os recursos ocorrem principalmente em áreas locais. Essas áreas são distribuídas no chip ao invés de estarem concentradas em um único ponto. Todas essas vantagens ajudam a criar um sistema multi-agentes escalável e menos intrusivo para aplicações executando no sistema.

Comparando com o gerenciamento centralizado, a solução proposta pelos autores apresenta um maior número de mensagens trafegando pela NoC, mas os autores avaliam

que essas mensagens são pacotes pequenos e na maioria das vezes requer poucos *hops* na NoC e são distribuídos em todo o MPSoC.

2.4 Migração de Tarefas

Nesta Seção apresentam-se pesquisas referentes à migração de tarefas em MPSoCs. Essa técnica surgiu com foco em desempenho e tolerância a falhas em sistemas distribuídos. Barcelos [BAR08] comenta que a área de sistemas embarcados está próxima a de sistemas distribuídos.

Segundo [BAR08], especificamente em MPSoCs, diversas condições podem requerer migração de tarefas:

1. Tarefas que se comunicam podem estar sendo executadas em processadores distantes um do outro, fazendo com que o tráfego de dados na NoC aumente;
2. Processadores com muitas tarefas esperando para serem executadas;
3. Processadores com alto consumo de energia, devido à quantidade de tarefas simultaneamente sendo executadas, enquanto outros processadores estão ociosos.

2.4.1 Acquaviva et al.

Acquaviva et al. [ACQ07] propõem uma nova técnica para reduzir a temperatura de um MPSoC, baseada em migração de tarefas entre processadores. O algoritmo proposto explora as informações de temperatura em tempo de execução para balancear a temperatura do MPSoC.

MPSoCs são caracterizados por possuírem uma grande área de silício, e experimentos demonstram uma distribuição desigual de densidade de calor. A técnica proposta é chamada *MiGra* e é composta por três algoritmos. O primeiro algoritmo é denominado *Total Swap*. Esse algoritmo consiste na seleção de um conjunto de processadores entre o processador fonte (onde está a tarefa a ser migrada) e processadores candidatos a receber a tarefa. O algoritmo utiliza três condições para realizar a migração:

1. Caso o processador fonte esteja com alta temperatura o processador destino deve estar com baixa temperatura.
2. Caso a frequência do processador fonte esteja acima da frequência média dos processadores do MPSoC, a frequência do processador destino deve estar abaixo da frequência média.
3. O consumo total dos dois processadores depois da migração tem que ser menor que o consumo total antes da migração.

Este algoritmo assume que todas as tarefas presentes no processador fonte são migradas. Devido a este fato, foi desenvolvido um novo algoritmo que desempenha uma busca de tarefas mais eficiente para migrar entre dois processadores, buscando diminuir o número de migrações e a quantidade de dados trafegando na NoC.

O segundo algoritmo do MiGra, denominado *Full Search*, não está sujeito às limitações da pesquisa exaustiva do *Total Swap*, pois permite que seja migrada apenas uma tarefa de cada vez. O problema é o número de comparações necessárias para calcular o custo da migração de todos os possíveis conjuntos de tarefas a migrar de um processador fonte para todos os possíveis conjuntos de tarefas para um processador destino.

Buscando evitar este problema, da necessidade de um grande número de comparações, foi desenvolvido o algoritmo *Bounded Search* que diminui as limitações do *Full Search* e mantém o custo computacional do *Total Swap*. Assim, o algoritmo foi modificado considerando que o efeito da migração da tarefa no balanceamento da temperatura diminui junto com a carga. Isto leva para o fato de que se pode limitar o número de tarefas para serem migradas, considerando apenas as tarefas com maior carga.

Esses três algoritmos exploram a uniformização de temperatura de um MPSoC em tempo de execução para determinar o conjunto de tarefas que será migrado de um processador com temperatura elevada para um processador com baixa temperatura. Porém, o problema deste trabalho é que os autores comentam os algoritmos utilizados, mas não detalham a forma de como a migração é efetuada. Apenas são explicados os motivos para fazer a migração, faltando informar como a mesma é realizada.

2.4.2 Pittau et al.

Pittau et al. [PIT07] apresentam uma camada *middleware* que implementa a migração de tarefas em um MPSoC. Os autores comentam que o mapeamento dinâmico de tarefas baseado em migração tem sido muito explorado para melhorar o desempenho e diminuir o consumo de energia em MPSoCs.

Para aplicações multimídia, migração de tarefas deve ser cuidadosamente avaliada para que não ocorram perdas de *deadline*. Foi caracterizado o desempenho de uma dada aplicação multimídia e o seu gasto de energia. Os experimentos mostram que a migração em nível de *middleware* ou sistema operacional é possível e pode levar a melhorias na economia de energia.

Os autores propõem dois tipos de mecanismos para migração de tarefas, os quais diferem na forma de gerenciar a memória. A primeira versão é a *Task Recreation*. Este mecanismo destrói o processo no processador fonte e recria no processador destino. Esta versão é baseada na execução de chamadas de sistema que cuidam da alocação de

espaço de memória necessário para as tarefas. A outra estratégia para migração é a *Task Replication*, onde existe uma réplica de cada tarefa em cada sistema operacional local. Somente um processador por vez pode executar uma réplica da tarefa. Enquanto a tarefa é executada em um processador, a réplica dela está em uma fila de tarefas em outro processador.

Nesta pesquisa os autores consideram cada tarefa um processo com seu espaço de endereçamento privado. Assim facilita-se o mapeamento de aplicações em uma plataforma com memória distribuída. O *framework* suporta migração de tarefas de modo que a política de gerenciamento de tarefas possa cuidar do mapeamento em tempo de execução, melhorando o desempenho, gerenciamento térmico e segurança.

2.4.3 Layouni et al.

Layouni et al. [LAY09] avaliam a técnica de migração de tarefas de *software* por replicação, em MPSoC. A migração de tarefas de *software* é realizada em tempo de execução e gerenciada pelo sistema operacional. A necessidade de migração pode ser para fins de: balanceamento de cargas, tolerância a falhas, economia de energia e gerenciamento térmico do chip. Os autores comentam que a migração de tarefas envolve a habilidade de interromper a execução de uma tarefa em um processador e dar sequência à mesma em outro.

Os autores adotam uma estratégia de replicação de tarefas, tanto para MPSoCs heterogêneos e homogêneos, juntamente com o uso de notificação de *checkpoints* (pontos de migração da tarefa). O usuário é responsável por configurar esses *checkpoints* no código da aplicação, o desempenho da aplicação depende desses *checkpoints*. Isso é feito via sistema operacional e é facilmente extensível, segundo os autores.

A estratégia de migração consiste em replicação dos códigos das tarefas em cada processador onde a migração será habilitada. Na Figura 9 os autores apresentam a funcionalidade da migração de tarefas, com a tarefa migrada e a réplica da mesma. A CPU_1 e a CPU_2 contém as réplicas das tarefas, contendo dois possíveis estados: *Dormant* e *Waiting*.

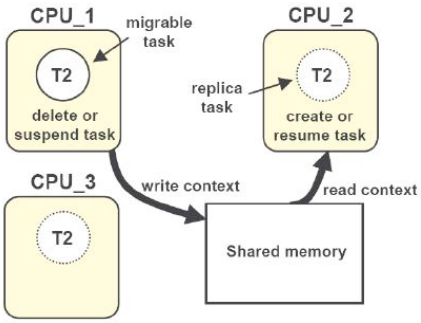

Figura 9 – Esquema geral de um sistema baseado em migração de tarefas por réplicas. Fonte: [LAY09].

A migração da tarefa só poderá ser executada quando a tarefa encontrar seu *checkpoint*. Quando se decide efetuar a migração de tarefas, primeiramente se salva o contexto desta tarefa em uma memória compartilhada. Após, a tarefa "migrada" é retirada do sistema ou suspensa. A réplica pode ser reiniciada ou criada em outro processador, o contexto salvo na memória compartilhada é restaurado. Por fim a tarefa pode ser reiniciada do *checkpoint* salvo na migração.

Os autores definem que o processo de migração de tarefas envolve a habilidade de parar uma tarefa em um processador e dar continuidade a ela em outro processador, melhorando o desempenho do sistema.

2.4.4 Ozturk et al.

O objetivo da pesquisa de Ozturk et al. [OZT06] é apresentar um método de migração. O algoritmo decide se será migrado o código ou os dados, buscando satisfazer os requisitos de comunicação. A escolha entre as duas opções de migração é realizada em tempo de execução baseado em estatísticas coletadas na etapa de *profiling*.

A migração é dividida em três etapas, como é apresentada na Figura 10: *profiling*: etapa para fazer o experimento de cada aplicação, sendo calculado o custo da energia de comunicação na rede e, também, calculado o custo da transferência dos dados das tarefas; anotação de código: etapa para especificar a migração de código ou de dados. Para a decisão da migração de código ou dados os autores calculam o custo da energia de comunicação entre as tarefas da aplicação. Essas duas primeiras etapas são desempenhadas em tempo de compilação. A terceira etapa é a execução do algoritmo de migração do código ou dos dados.

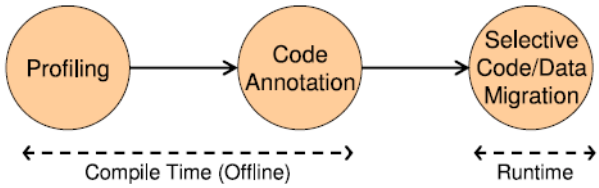

Figura 10 – Etapas para a decisão de migração de tarefas. Fonte: [OZT06]

Neste trabalho os autores focam apenas na descrição dos cálculos da decisão de migrar o código ou os dados das tarefas, não descrevendo como realmente é efetuada a migração. Outro importante detalhe não descrito pelos autores é apresentar de que forma segue a execução da aplicação e a comunicação entre as tarefas, levando em conta a migração de apenas o código ou os dados. Os autores descrevem que em suas experiências, com várias aplicações em MPSoCs, a migração de tarefas pode melhorar o desempenho e o poder computacional destes.

2.5 Considerações Finais

A Tabela 1 apresenta um comparativo entre os trabalhos analisados neste Capítulo. Como podemos notar, foram analisados trabalhos com objetivos diferentes, mostrando que as pesquisas estão cada vez mais dedicadas para MPSoCs adaptativos. Visando a adaptabilidade dos MPSoCs, as pesquisas analisadas no estado da arte apresentam técnicas de monitoramento de tarefas, escalonamento, controle distribuído e migração de tarefas para gerenciar a uniformização de temperatura, minimização de energia, balanceamento de carga, melhorias no desempenho, etc.

As pesquisas analisadas no estado da arte de monitoramento de tarefas apresentam diferentes objetivos de monitoramento. Em [STA11] temos a proposta de um método para controle de *deadlines* capaz de melhorar a qualidade de serviços para aplicações multimídia. Em [CIO06] os autores apresentam alternativas de monitoramento utilizando interconexões físicas separadas para dados de aplicações e dados de monitoramento, avaliando o impacto dessas técnicas nos projetos de NoCs. Em [MAT10] é proposto um mecanismo baseado em monitoramento de tarefas para efetuar um controle de redimensionamento dinâmico de *buffer*, em tempo de execução.

No estado da arte de escalonamento de tarefas, foram apresentadas pesquisas que apresentam objetivos de gerenciamento térmico e economia de energia. [COS07] [TAF11] [ZHA09] mostram que as pesquisas para escalonamento em MPSoCs focam na minimização da energia consumida.

As pesquisas referentes a controles distribuídos de MPSoCs tem por objetivo tornar os sistemas escaláveis, capazes de ter vários monitores gerenciando os recursos do sistema. Em [FAT11] temos uma proposta de controle distribuído capaz de efetuar o monitoramento do sistema em três níveis: local, *cluster* e global. Em [SHA11] os autores apresentam duas técnicas para efetuar melhorias no desempenho do sistema, com aplicações controladas por um gerente de recursos. Em [KOB11] os autores apresentam uma proposta de controle distribuído, em MPSoC, baseado em um algoritmo multi-agente, que gerencia os recursos do sistema.

Para a migração de tarefas, temos no estado da arte como principal objetivo o balanceamento de carga. Em alguns casos, o ponto de migração é predefinido na codificação da tarefa. Apenas [ACQ07] efetua a migração de tarefas conforme real necessidade, considerando o processador com alta temperatura ou até mesmo com alta frequência, sem a necessidade de forçar a migração (com pontos de migração configurados pelo usuário em tempo de projeto), como feito em [PIT07] e [LAY09]. Nesta questão, mostra-se que nenhuma das pesquisas analisadas apresenta um controle distribuído para o gerenciamento de tarefas a ponto de controlar a necessidade de efetuar a migração de uma determinada tarefa baseado em dados coletados por um monitoramento do sistema.

Outro ponto importante a se destacar, é que na pesquisa realizada por Ozturk et al. [OZT06] é utilizado um processo dividido por etapas (*profiling*, configuração e execução), assim como o presente trabalho.

Tabela 1 – Comparativo entre as pesquisas analisadas.

Referência	Monitoramento de Tarefas	Escalonamento	Controle Distribuído	Migração de Tarefas	Função Custo	Aplicação Foco
[STA11]	Sim	Não	Não	Não	Controle de *deadline*	Multimídia
[CIO06]	Sim	Não	Não	Não	Escalabilidade / Custo de área e projeto / Reusabilidade de recursos	Não abordada
[MAT10]	Sim	Não	Não	Não	Redimensionamento dinâmico de *buffer*	Multimídia
[COS07]	Não	Sim	Não	Não	Uso consciente de temperatura	Não abordada
[TAF11]	Não	Sim	Não	Não	Aumentar desempenho	Não abordada
[ZHA09]	Não	Sim	Não	Não	Otimização no consumo de energia	Não abordada
[FAT11]	Não	Não	Sim	Não	Gerenciar recursos	Não abordada
[SHA11]	Não	Não	Sim	Não	Escalabilidade	Multimídia
[KOB11]	Não	Não	Sim	Não	Escalabilidade	Aplicações reais
[ACQ07]	Não	Não	Não	Sim	Uniformizar a temperatura	Não abordada
[PIT07]	Não	Não	Não	Sim	Otimização no consumo de energia	Multimídia
[LAY09]	Não	Não	Não	Sim	Balanceamento de carga	Não abordada
[OZT06]	Não	Não	Não	Sim	Consumo de energia	Aplicações reais para manipulação de imagens
Dissertação	**Sim**	**Sim**	**Sim**	**Sim**	**Balanceamento de carga / Consumo de energia / Controle de *deadline***	**Aplicações com vazão garantida**

A originalidade da pesquisa desenvolvida na presente Dissertação é a integração e avaliação de diversas técnicas, as quais são avaliadas individualmente na literatura. As técnicas que serão detalhadas posteriormente envolvem o controle distribuído de recursos do MPSoC, monitoramento de vazão e latência, e as técnicas de escalonamento e migração de tarefas para prover adaptabilidade às aplicações.

3 ARQUITETURA HEMPS

Neste Capítulo é apresentado o MPSoC homogêneo HeMPS (HERMES Multiprocessor System) [WOS07][CAR09], utilizado como plataforma de referência para integração das técnicas de adaptabilidade deste trabalho. Essa plataforma é utilizada em duas versões: (*i*) arquitetura com controle de recursos centralizado, contendo um processador responsável pela gerencia de todo o sistema, apresentada na Seção 3.1; (*ii*) arquitetura com controle de recursos distribuído, contendo um MPSoC dividido em regiões, cada qual com um processador responsável pela gerencia da região, apresentada na Seção 3.2.

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 11). 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 11 mostra um exemplo de uma aplicação modelada de acordo com esta abordagem.

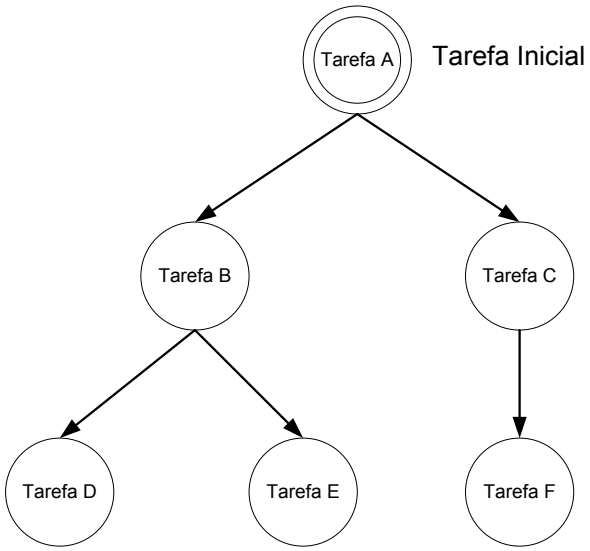

Figura 11 – Exemplo de aplicação modelada por um grafo de tarefas.

3.1 Arquitetura com Controle Centralizado de Recursos

A arquitetura HeMPS é um MPSoC homogêneo que emprega elementos de processamentos, denominados Plasma-IP [PLA01], interconectados pela NoC Hermes [MOR04] que é utilizada como infraestrutura de comunicação. Também, tem-se uma memória externa, denominada repositório de tarefas, aa qual contém o código-objeto de todas as tarefas que executarão no sistema. Na Figura 12 é ilustrada uma instância do

MPSoC HeMPS configurado como uma NoC de dimensão 2x3 interconectando os Plasmas-IP.

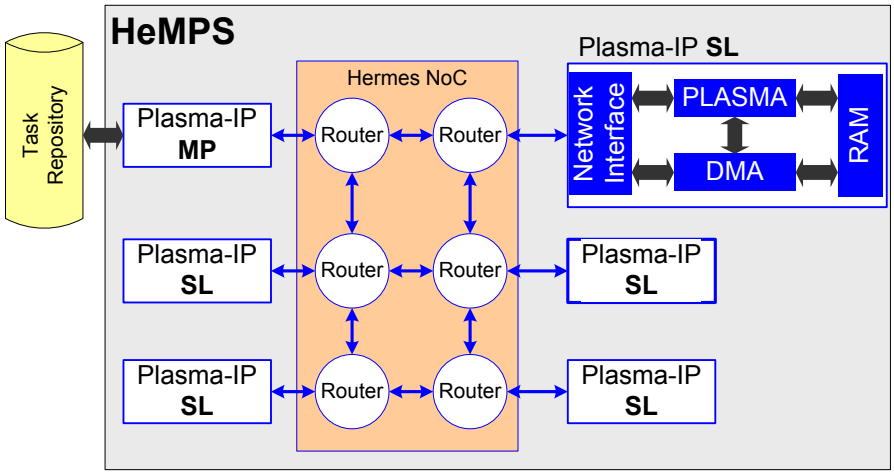

Figura 12 – Instância da HeMPS Centralizada.

Os elementos de processamento do MPSoC HeMPS são divididos em: mestre, Plasma-IP MP, o qual gerencia os recursos do sistema; escravos, Plasma-IP SL, efetuam a execução das tarefas. Cada Plasma-IP é composto por:

- Interface de rede (NI, do inglês, *Network Interface*): é a interface de ligação entre o elemento de processamento e a NoC. Sendo o módulo responsável pelo envio e recebimento de pacotes na rede.
- Processador Plasma: processador RISC de 32 bits com organização de memória Von Neumann. Oferece suporte a linguagem de programação C com tratamento de interrupções.
- Memória privada: onde está armazenado o *microkernel* (sistema operacional simples) que executa em cada processador Plasma. No Plasma-IP SL a memória é dividida em páginas com tamanho fixo para alocação das tarefas (cada tarefa utiliza uma página da memória).
- Módulo de acesso direto à memória (DMA, do inglês, *Direct Memory Access*): módulo que auxilia o Plasma no envio e recebimento de mensagens com a NI. Assim, o DMA permite que o Plasma continue sua execução de tarefas sem a necessidade de controlar diretamente a troca de mensagens com a rede. A principal função deste módulo é transferir o código-objeto das tarefas (que chegam pela NI) para a memória do Plasma.

A gerência de recursos do sistema é centralizada devido ao fato de conter apenas um processador mestre para gerenciar todo o sistema. O processador mestre é responsável por gerenciar os recursos do sistema, sendo o único processador que tem acesso ao repositório de tarefas. No momento em que a HeMPS inicia sua execução, o processador mestre aloca as tarefas iniciais nos processadores escravos. Durante a

execução, cada processador escravo pode solicitar ao mestre a alocação dinâmica de tarefas do repositório. Além disso, recursos podem ficar disponíveis quando determinada tarefa terminar a sua execução. Cada processador escravo executa um *microkernel*, que suporta execução multitarefa e comunicação entre tarefas.

O protocolo adotado para a comunicação entre tarefas é o *read request*. Segundo Carara [CAR11], este protocolo garante o controle de fluxo fim-a-fim e o ordenamento de mensagens. Quando uma determinada tarefa executa a primitiva `Send()`, a mensagem é armazenada no *pipe* local (área de memória do sistema operacional), e a computação continua, caracterizando uma escrita não-bloqueante. Ao executar a primitiva `Receive()`, a tarefa destino envia um comando *message_request* para o processador origem, e havendo dados no *pipe*, estes são enviados através da NoC. A leitura é desta forma bloqueante. Este protocolo tem a vantagem de reduzir o tráfego na rede, pois uma mensagem só é injetada na rede se a mesma tiver sido solicitada.

3.2 Arquitetura com Controle Distribuído de Recursos

Da mesma forma que a arquitetura com gerência de recursos centralizada, a arquitetura distribuída da HeMPS emprega processadores Plasma-IP [PLA01] interconectados pela NoC Hermes [MOR04]. Na Figura 13 é ilustrada uma instância do MPSoC HeMPS distribuído configurado como uma NoC de dimensão 6x6 divido em 4 *clusters* 3x3.

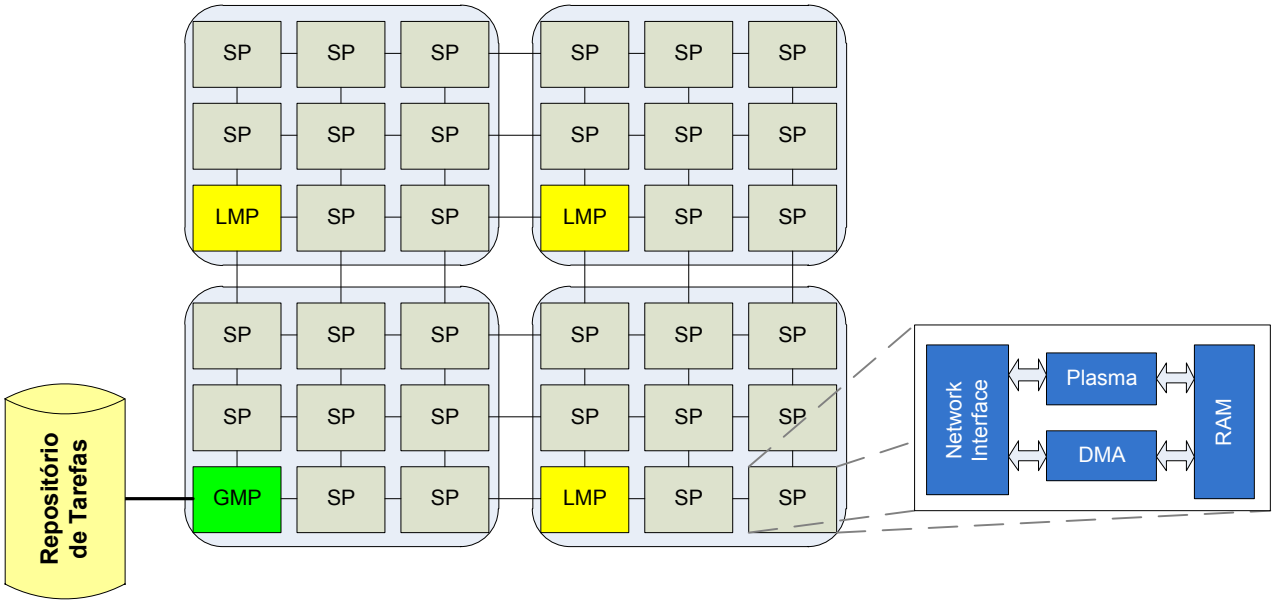

Figura 13 – Instância da HeMPS Distribuída.

Na arquitetura centralizada temos um processador mestre que gerencia todo o sistema. A característica da arquitetura distribuída é aumentar a quantidade de mestres dividindo a gerência por regiões. Com isso, um processador não fica responsável por gerenciar todo o sistema, mas sim a sua região (que é formada por *n* processadores,

formando um *cluster*). Assim, a arquitetura distribuída contém três níveis de processamento: (*i*) mestre global; (*ii*) mestres locais; (*iii*) processadores escravos.

O SP (processador escravo) é responsável pela execução das tarefas. Todos os SPs executam um sistema operacional simples que permite comunicação entre PEs e execução multitarefa, com suporte a função de monitoramento de tarefas. O LMP (processador mestre local) é responsável pela gerência do *cluster*, executando funções de adaptabilidade, mapeamento de tarefas e efetua a comunicação com os outros LMPs e o GMP. O LMP é responsável pela gerência das aplicações. Em alguns casos, o *cluster* não tem espaço para mapear todas as tarefas de uma aplicação, então, o LMP pode mapear tarefas fora de seu *cluster*. O GMP (processador mestre global) efetua ações de gerência global, como acesso ao repositório de tarefas, seleção de *cluster* que irá receber determinada aplicação, envio de tarefas aos *clusters*, além do controle do próprio c*luster.*

A vantagem da gerência distribuída sobre a centralizada reside no fato que as tarefas que requerem mais computação, como mapeamento e migração de tarefas, são delegadas a vários PEs (LMPs). Também, o monitoramento é efetuado em nível de *cluster*, diminuindo o tráfego na rede.

4 MONITORAMENTO DE TAREFAS

Este Capítulo apresenta a primeira contribuição da Dissertação: a implementação do mecanismo de monitoramento de tarefas. O monitoramento de tarefas foi projetado buscando minimizar problemas que afetam diretamente o desempenho das aplicações sendo executadas em MPSoCs, tais como: (*i*) perda de *deadlines* das aplicações; (*ii*) vazão ou latência não respeitadas; (*iii*) tarefas sendo executadas distantes de suas comunicantes, aumentando o consumo de energia na NoC. A solução para esses problemas é essencial para que um MPSoC tenha características de um sistema adaptativo, provendo ganhos de desempenho e economia de energia.

O mecanismo de monitoramento é a primeira técnica do processo de adaptabilidade. Este é um processo cíclico, executando de forma contínua ao longo da execução das aplicações no sistema. A adaptabilidade faz uso das três técnicas descritas nesta Dissertação. A Figura 14 ilustra as três etapas do processo. A primeira etapa é efetuar a gerência de determinada aplicação, coletando dados relativos aos parâmetros de desempenho monitorados (como *deadlines*, vazão, latência). Após efetuar o monitoramento, temos a técnica de gerência (como apresentada no Capítulo 3, podendo ser centralizada ou distribuída), que é responsável pela tomada de decisões, baseado nos dados coletados pelo monitor. Com isso, a última etapa do processo de adaptabilidade é a execução de uma das técnicas adaptativas, que podem ser: (*i*) alteração das características de escalonamento, detalhada no Capítulo 5; (*ii*) efetuar a migração de determinada tarefa da aplicação monitorada, detalhada no Capítulo 6.

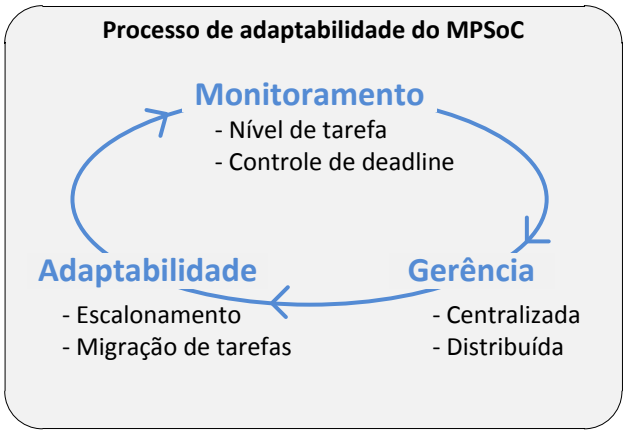

Figura 14 – Processo de Adaptabilidade do MPSoC.

A técnica de monitoramento foi desenvolvida para as plataformas com gerencia de recursos centralizada e distribuída, sendo o mesmo protocolo para ambas as plataformas. Na Seção 4.1, apresenta-se a estrutura do monitoramento de tarefas, descrevendo a configuração necessária para sua execução. Na Seção 4.2, apresenta-se o protocolo de monitoramento, descrevendo o funcionamento desta técnica e de que forma pode afetar o desempenho das aplicações do sistema.

4.1 Estrutura do Monitoramento de Tarefas

O monitoramento é configurado baseado nos dados coletados na etapa de *profiling* (avaliação das características da aplicação). Na etapa de *profiling* deve-se verificar qual o *deadline*[1] aceitável da aplicação. Para isso, a aplicação a ser monitorada é executada sozinha no MPSoC e é analisada a latência média de comunicação entre as tarefas. Com base nessas latências, é definido o *deadline* aceitável da aplicação.

Para configurar o *deadline* de uma aplicação, foi adicionada uma nova função na API de comunicação: `SetDeadline(int deadline)`. Na implementação desta função foi criada uma chamada de sistema, denominada `SETDEADLINE`, que atribui o valor de `deadline` no campo `deadline` da TCB[2] da tarefa. O campo `deadline` foi adicionado na TCB, pois originalmente este não existia. Durante a análise de dados na etapa de *profiling*, também, deve-se informar a quantidade aceitável de perdas de *deadlines* da aplicação. Para armazenar essa quantidade adicionou-se na TCB o campo `max_miss`, que representa quantas violações a aplicação pode sofrer antes do PE monitor solicitar ao PE mestre a alteração nas características do escalonador de tarefas (descritas no Capítulo 5). Uma violação de *deadline* ocorre quando a latência média de comunicação entre duas tarefas excede o valor configurado no campo `deadline`, na etapa de *profiling.* Também, adicionou-se na TCB o campo `max_miss_migra` que representa o valor aceitável de perdas de *deadlin*e antes de solicitar de migração. Logo, na etapa de *profiling* devem ser configurados três parâmetros: `deadline`, `max_miss`, e `max_miss_migra`.

No Figura 15, ilustra-se o processo para configuração do monitoramento da aplicação. Inicialmente tem-se determinada aplicação descrita por algumas tarefas. Cada tarefa, `TaskC.c` por exemplo, faz uso da API de comunicação, implementada no `task.h`. Como explicado anteriormente, usa-se a função `SetDeadline()` para informar os valores de configuração da tarefa. Esta função realiza uma chamada de sistema implementada no *microkernel* do processador escravo (`kernel_slave.c`). Nesta chamada de sistema, atribui-se os valores parametrizáveis, na função, aos respectivos campos da TCB, que estão declarados no `kernel_slave.h`.

O PE que estiver executando a tarefa com configuração de *deadline* é o responsável pelo monitoramento da aplicação. Com o *deadline* configurado, o monitor inicia sua execução analisando a latência de comunicação da tarefa. Quando a quantidade de perdas de *deadlines* atingir o valor configurado em `max_miss`, este PE solicita a alteração das características de escalonamento ao PE mestre, e quando a quantidade de perdas de

[1] na presente Dissertação *deadlines* referem-se a valores de latência que devem ser respeitados

[2] *task control block,* estrutura do sistema operacional que tem por função armazenar parâmetros de controle das tarefas.

deadlines atingir o valor configurado em `max_miss_migra`, o PE solicita migração de alguma tarefa da aplicação ao PE mestre.

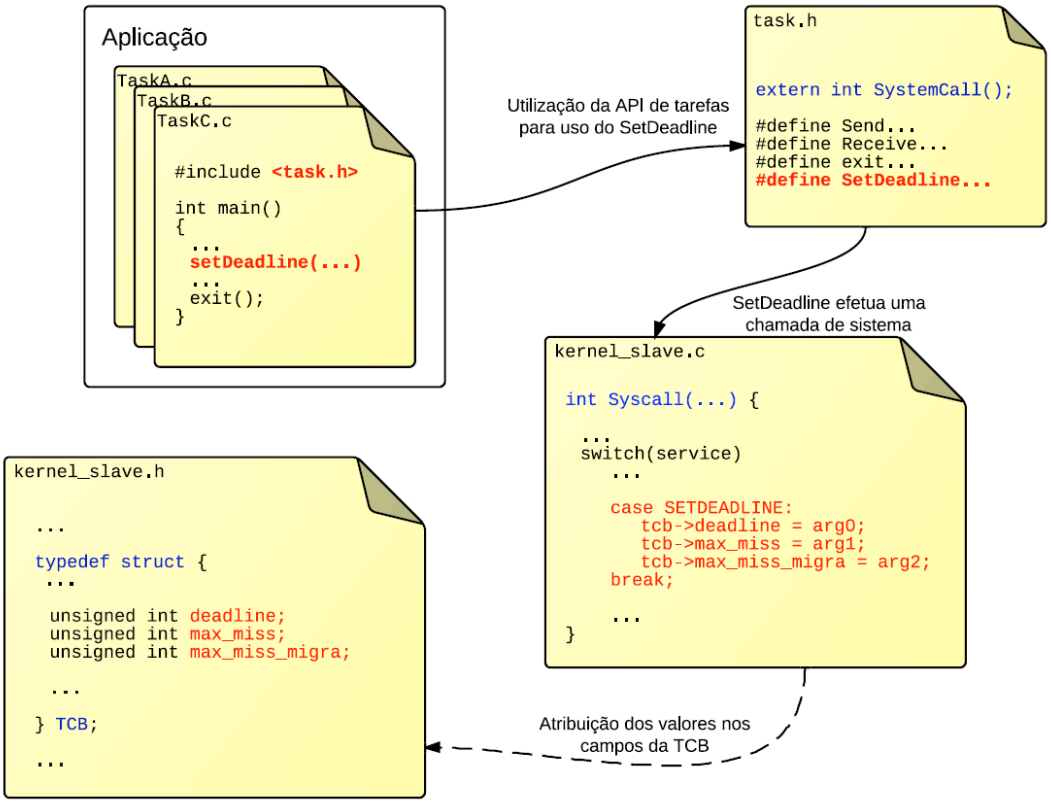

Figura 15 – Fluxo de configuração de tarefas para monitoramento.

4.2 Protocolo do Monitoramento de Tarefas

Nesta Seção apresenta-se o protocolo de monitoramento de tarefas. Define-se como “PE monitor” um PE que está monitorando uma ou mais tarefas nele executando. O PE monitor verifica violações de *deadline* e a necessidade de aplicar alguma técnica de adaptabilidade, repassando a informação ao PE mestre, responsável pela gerência de recursos. Como apresentado na Seção 3.1, a arquitetura com gerência centralizada de recursos contém apenas um processador mestre. Neste caso, a tomada de decisões será totalmente gerenciada por tal processador. Já na arquitetura com gerência distribuída de recursos, Seção 3.2, existe um processador mestre por região. Assim, o PE mestre responsável pela tomada de decisões será o mestre responsável pela aplicação monitorada.

Na Figura 16 apresenta-se um cenário de monitoramento em um MPSoC com gerência centralizada de recursos. O cenário contêm uma aplicação formada por seis tarefas (A-F), executando no MPSoC, juntamente com uma aplicação que gera tráfego de dados que compete por recursos na rede (aplicação denominada *distrubing*). A aplicação *disturbing* gera atrasos na comunicação, podendo gerar a ocorrência de perdas de *deadlines* na aplicação principal.

A tarefa F é a tarefa monitorada, pois é a tarefa que gera os dados da aplicação (última tarefa do grafo). O processador que está executando essa tarefa é o processador responsável pelo monitoramento da aplicação. A Figura 16(a), apresenta o mapeamento inicial das tarefas, com o monitoramento analisando o cumprimento dos *deadlines* da tarefa F. Na Figura 16(b), o PE monitor verifica que a quantidade de violações de *deadlines* atingiu a quantidade configurada no campo `max_miss` da TCB da tarefa F. Com isso, o PE monitor envia um pacote ao PE mestre solicitando que sejam alteradas as características de escalonamento de todas as tarefas da aplicação. Essas características serão detalhadas na Seção 5.3. Na Figura 16(c), o PE mestre recebe a solicitação do PE monitor, para executar a técnica de adaptabilidade. Após isso, o PE mestre envia pacotes para todos os processadores, que executam as tarefas da aplicação monitorada, solicitando aumento da prioridade e *time-slice* das mesmas. Ao receber o pacote de solicitação, do PE mestre, os PEs escravos alteram as características do escalonamento das tarefas da aplicação monitorada.

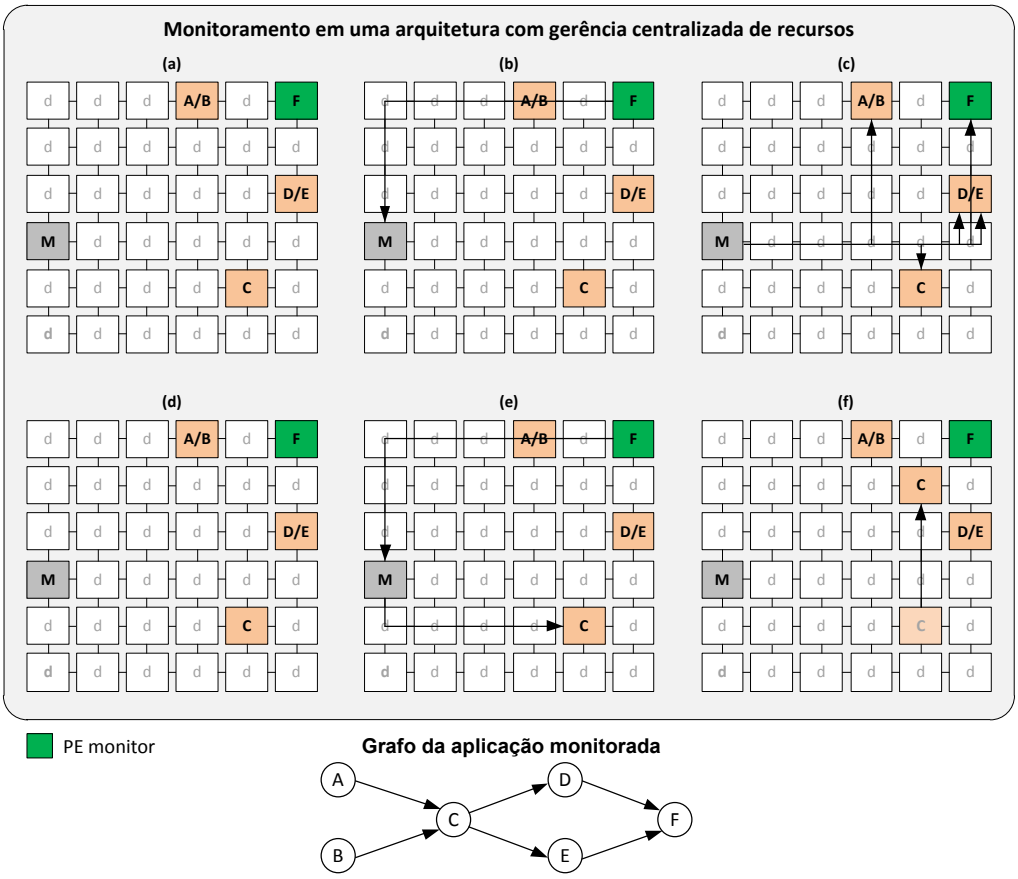

Figura 16 – Protocolo do mecanismo de monitoramento em uma arquitetura com gerência centralizada de recursos.

Na sequencia, o monitoramento volta a sua execução normal, na Figura 16(d), verificando a existência de novas violações de *deadline*. Na Figura 16(e) o PE monitor identifica que novas violações de *deadline* aconteceram, atingindo o valor configurado no campo `max_miss_migra` da TCB da tarefa monitorada. Com isso, o PE monitor envia

um pacote ao PE mestre informando tais violações e solicitando a migração de alguma tarefa da aplicação monitorada. Ao receber o pacote, o PE mestre identifica qual tarefa deve ser migrada (essa decisão é detalhada no Capítulo 6). Neste caso, o PE mestre decide que a tarefa a ser migrada é a tarefa C. Sabendo qual tarefa migrar, o PE mestre calcula uma nova posição para a tarefa (esse cálculo é, também, detalhado no Capítulo 6) e envia um pacote de migração ao processador que está executando a tarefa C. Na Figura 16(f) o PE escravo recebe o pacote de migração e migra a tarefa C para sua nova posição, mais próxima de suas comunicantes. Com isso, as tarefas comunicantes ficam em execução próximas umas das outras, melhorando-se o desempenho da aplicação, e reduz-se a energia consumida na NoC.

Na Figura 17 apresenta-se um cenário de monitoramento em um MPSoC com gerência distribuída de recursos. Da mesma forma que o cenário apresentado na Figura 16, este cenário contêm uma aplicação formada por seis tarefas (A-F), executando no MPSoC, juntamente com uma aplicação *disturbing* para gerar atrasos na comunicação, podendo gerar a ocorrência de perdas de *deadlines* na aplicação principal.

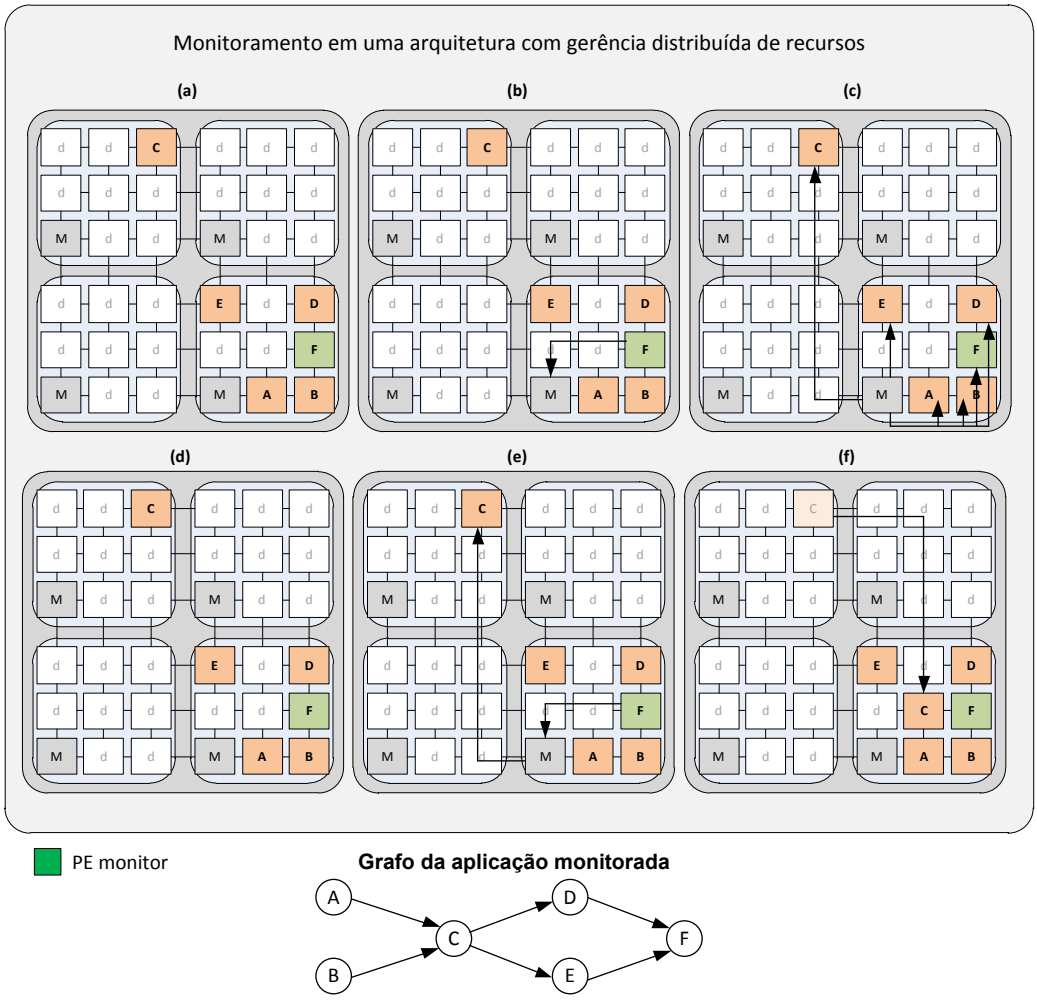

Figura 17 – Protocolo do mecanismo de monitoramento em uma arquitetura com gerência distribuída de recursos.

O protocolo de monitoramento em uma arquitetura com gerência distribuída de recursos é semelhante ao protocolo com gerência centralizada de recursos. A principal diferença está no processador que gerencia as ações de adaptabilidade. Na gerência centralizada o processador mestre é o PE responsável pelas ações, já na gerência distribuída, o PE responsável pela execução das ações de adaptabilidade são os processadores locais de cada *cluster*. Quando um PE monitor identificar a necessidade de executar alguma técnica de adaptabilidade, este PE deve enviar um pacote ao PE mestre local, responsável pela gerência de recursos do *cluster,* informando tal necessidade.

Assim como no cenário apresentado na Figura 16, a tarefa F é a tarefa configurada para conter o controle de perdas de *deadline*, e o processador que está executando essa tarefa é o processador responsável pelo monitoramento da aplicação. A Figura 1Figura 17(a), apresenta o mapeamento inicial das tarefas, com o monitoramento analisando o *deadline* da tarefa F. Na Figura 17Figura 16(b), o PE monitor verifica que a quantidade de violações de *deadlines* atingiu a quantidade configurada no campo `max_miss` da TCB da tarefa monitorada. Com isso, o PE monitor envia um pacote ao PE mestre local, solicitando que sejam alteradas as características de escalonamento de todas as tarefas da aplicação. Na Figura 17Figura 16(c), o PE mestre local recebe a solicitação do PE monitor, para executar a técnica de adaptabilidade. Após isso, o PE mestre local envia pacotes para todos os processadores, que executam as tarefas da aplicação monitorada, solicitando aumento da prioridade e *time-slice* das mesmas.

Ao receber o pacote de solicitação, do PE mestre local, os PEs escravos alteram as características do escalonamento das tarefas. Na sequencia, o monitoramento volta a sua execução normal, na Figura 17 (d), verificando a existência de novas violações de *deadline*. Na Figura 17 (e) o PE monitor identifica que novas violações de *deadline* aconteceram, atingindo o valor configurado no campo `max_miss_migra` da TCB da tarefa F. Com isso, o PE monitor envia um pacote ao PE mestre local informando tais violações e solicitando a migração de alguma tarefa da aplicação monitorada. Ao receber o pacote, o PE mestre local verifica se existem tarefas sendo executadas fora do seu *cluster* (essa verificação é detalhada na Seção 6.3). Neste caso, o PE mestre local verifica que a tarefa C deve ser migrada. Sabendo qual tarefa migrar, o PE mestre local calcula uma nova posição para a tarefa (esse cálculo é, também, detalhado na Seção 6.3) e envia um pacote de migração ao processador que está executando a tarefa C. Na Figura 17(f) o PE escravo recebe o pacote de migração e migra a tarefa C para sua nova posição, mais próxima de suas comunicantes. Com isso, o sistema é desfragmentado, melhorando-se o desempenho da aplicação, e reduz-se a energia consumida na NoC.

5 ESCALONAMENTO

Neste Capítulo apresentam-se três algoritmos de escalonamento. O primeiro é o *Round-Robin*, descrito na Seção 5.1. O segundo algoritmo apresentado é o *Round-Robin* com adição de prioridades de tarefas, detalhado na Seção 5.2. Ambos os algoritmos serviram como base para o desenvolvimento do algoritmo de escalonamento preemptivo baseado em prioridades com uso de *time-slice*, apresentado na Seção 5.3. Este é o algoritmo de escalonamento utilizado no MPSoC HeMPS para avaliação das técnicas de adaptabilidade da presente Dissertação. Ele foi desenvolvido por [CAR11] e adaptado para o presente trabalho, nas plataformas centralizada e distribuída.

5.1 Algoritmo de Escalonamento *Round-Robin*

Nesta Seção apresenta-se o algoritmo de escalonamento *Round-Robin*. Este escalonamento garante que todas as tarefas do processador sejam executadas com o mesmo período de tempo e uma por vez. As tarefas são escalonadas em uma lista circular percorrida regularmente.

Em [LI03], os autores apresentam o algoritmo *Round-Robin*. Esse algoritmo garante que cada tarefa é alocada no processador por um determinado intervalo de tempo fixo, denominado *time-slice*. Para cada ciclo de relógio, um contador de *time-slice* é incrementado. Quando uma tarefa completa seu *time-slice*, o seu contador é zerado e a próxima tarefa da fila é escalonada, passando a ser executada com o mesmo período de tempo.

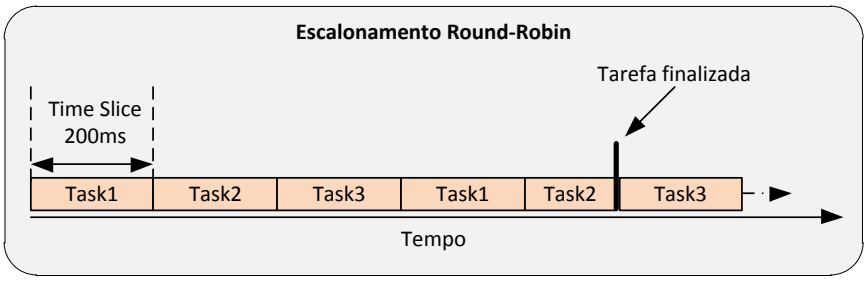

Figura 18 – Escalonamento Round-Robin.

Na Figura 18, apresenta-se uma fila circular com três tarefas a serem executadas no processador. Inicialmente, é alocada a primeira tarefa da fila, Task1. Como podemos ver na figura, o tempo de execução é de 200 ms. Quando seu *time-slice* acabar, a tarefa é preemptada para que a Task2 seja escalonada. O ciclo é contínuo, alocando uma tarefa por vez e com o mesmo período de tempo. Quando uma tarefa é finalizada, antes de seu *time-slice* encerrar (como podemos ver no caso da Task2), a próxima tarefa da fila é alocada.

Segundo [LI03], o escalonamento por *Round-Robin* não satisfaz os requisitos de sistemas tempo real, pois esses sistemas contém tarefas que executam com diferentes requisitos de desempenho.

5.2 Algoritmo de Escalonamento *Round-Robin* com Prioridade de Tarefas

Nesta Seção apresenta-se um complemento ao algoritmo de escalonamento *Round-Robin*, apresentado na Seção anterior. Em [LI03], os autores apresentam o algoritmo *Round-Robin* baseado em prioridade. De modo geral, os *kernels* suportam até 256 níveis de prioridades, onde o valor 0 representa maior prioridade e o valor 255 representa menor prioridade.

Com este algoritmo, a alocação das tarefas é feita de forma circular e cada tarefa tem sua prioridade. Sendo que algumas tarefas podem ter prioridade maior do que as outras. As tarefas com maiores prioridades executam primeiro. Se uma tarefa com maior prioridade do que a tarefa atual em execução estiver pronta para executar, o *microkernel* salva o contexto da tarefa atual, em sua TCB, e passa a executar a tarefa com a prioridade maior.

A Figura 19 apresenta uma fila circular de tarefas para escalonar. A *Task1* é inicialmente escalonada. Quando um evento externo, como dados para uma tarefa de maior prioridade, como a *Task2*, é recebido pelo processador, a *Task1* é preemptada para que a tarefa com maior importância seja executada. Em seguida o mesmo ocorre com a *Task2*, que é preemptada pela *Task3* que contém maior prioridade. A *Task3* ficará em execução até ser finalizada, ou entrar em estado de espera ou até que entre, na fila de tarefas, uma tarefa com maior prioridade.

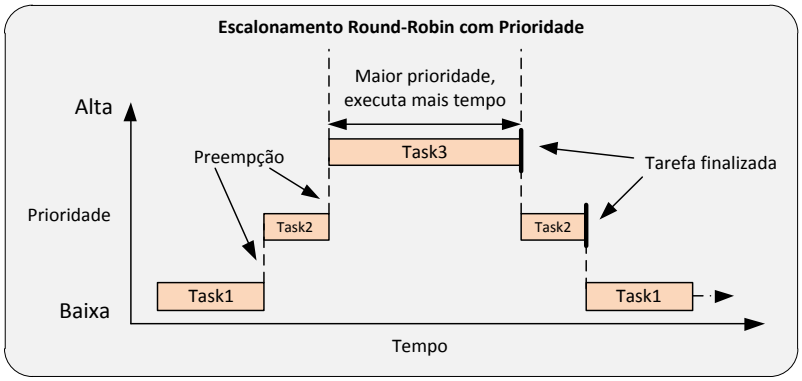

Figura 19 – Escalonamento *Round-Robin* com prioridade de tarefas. Fonte: [LI03]

Da mesma forma que o algoritmo *Round-Robin,* apresentado na Seção anterior, [LI03] comentam que o algoritmo ilustrado na Figura 19 não atende os requisitos de sistemas em tempo real. Como alternativa, na próxima Seção apresenta-se um algoritmo baseado no *Round-Robin* com prioridade fazendo uso do *time-slice* dinâmico e não fixo, descrito em [LI03].

5.3 Algoritmo de Escalonamento Preemptivo Baseado em Prioridade / *Time-Slice*

Nesta Seção apresenta-se o algoritmo de escalonamento adaptado para o presente trabalho. Este algoritmo tem por base o escalonamento *Round-Robin* com prioridade de tarefas, apresentado na Seção 5.2. Da mesma forma que o *Round-Robin,* esse escalonador contem uma fila circular de tarefas. Porém, o diferencial deste algoritmo é o uso de duas características fundamentais para a escolha de qual tarefa escalonar: (*i*) prioridade, responsável por definir qual tarefa deve ser escalonada antes; (*ii*) *time-slice*, neste algoritmo o tempo não é fixo, podendo ser alterado no decorrer da execução do sistema.

Na Figura 20, apresenta-se um cenário com três tarefas para serem escalonadas. Inicialmente a *Task1* é escalonada com *time-slice* de 100ms e com a mesma prioridade das demais tarefas, *Task2* e *Task3*. Assim como o algoritmo apresentado na Seção 5.2, este algoritmo escalona as tarefas em uma fila circular regular, executando a *Task2* e na sequencia a *Task3*, com o mesmo período de tempo. Quando *Task1* retorna sua execução, a *Task3* tem sua prioridade e seu *time-slice* alterados em tempo de execução. Dessa forma, a *Task1* tem seu contador de tempo salvo e é preemptada, para que a *Task3* seja escalonada e executada com seu novo período de tempo (170ms). Após a execução da *Task3*, o contador de tempo da *Task1* é restaurado e ela retorna sua execução a partir do ponto em que foi preemptada.

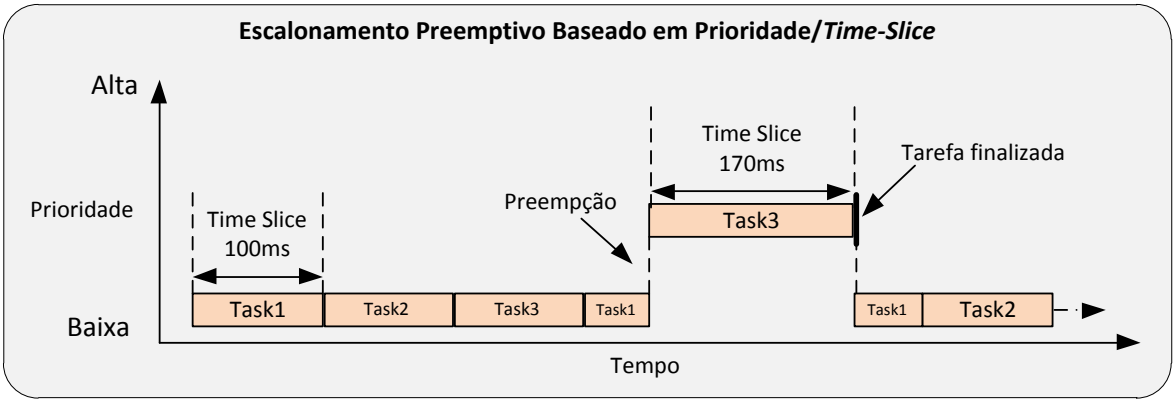

Figura 20 – Escalonamento preemptivo baseado em prioridade/*time-slice*. Adaptado: [LI03]

Como descrito no Capítulo 4, o monitoramento de tarefas, para controle de violações de *deadlines,* afeta diretamente a decisão do algoritmo de escalonamento. Como o monitoramento envia pacotes ao PE mestre solicitando a execução de técnicas de adaptabilidade, o PE mestre efetua alterações nas prioridades e *time-slice* de todas as tarefas da aplicação em monitoramento. Dessa forma, quando uma aplicação começar a violar o *deadline*, então, as tarefas passarão a ter maiores prioridades e maior tempo de execução em seus processadores. As prioridades das tarefas são aumentadas do nível baixo para o alto, já o *time-slice* é aumenta em 60%. Inicialmente as tarefas são escalonadas com 10 kciclos de relógio. Ao efetuar a troca das características o *time-slice* passa para 16 kciclos de relógio.

6 MIGRAÇÃO DE TAREFAS

O presente Capítulo apresenta a técnica de migração de tarefas desenvolvida no escopo da presente Dissertação. Esta técnica tem por objetivos realizar o balanceamento de carga no MPSoC e desfragmentar o sistema. Para isso, as tarefas comunicantes devem ser executas próximas uma das outras. Com essa técnica podemos reduzir a energia consumida na comunicação e otimizar o desempenho das aplicações do sistema.

Na Seção 6.1 apresenta-se o protocolo de migração de tarefas, descrevendo como o código objeto, dados e contexto da tarefa são enviados para um novo processador. Na Seção 6.2 apresenta-se a heurística de migração de tarefas em uma arquitetura com gerência centralizada de recursos. A heurística de migração de tarefas em um MPSoC com gerência distribuída de recursos apresenta-se na Seção 6.3. Na Seção 6.4 descreve-se a forma que o monitoramento de tarefas age diretamente no momento que a migração deve ser efetuada.

O protocolo e a heurística de migração, aqui detalhados, foram publicados em [MOR12a].

6.1 Protocolo de Migração de Tarefas

Nesta Seção apresenta-se o protocolo de migração de tarefas, definindo a forma como o código objeto, dados e contexto são enviados para outro processador. Este protocolo é utilizado, da mesma maneira, na arquitetura com gerência de recursos centralizada e na distribuída.

A técnica de migração de tarefas foi desenvolvida utilizando o MPSoC HeMPS como plataforma de referência, contendo as seguintes características:

1. Bloqueante, com migração de código, dados e contexto;
2. Sem pontos de controle de migração (*checkpoints*);
3. A migração é efetuada a partir de uma análise nos dados de comunicação de tarefas, pela técnica de monitoramento;
4. Uma tarefa pode ser migrada quantas vezes forem necessárias;
5. As mensagens a serem enviadas a outras tarefas, armazenadas na estrutura *pipe* do *microkernel*, não são migradas.

Na Figura 21, apresenta-se o protocolo de migração de tarefas em um diagrama de sequência. A seguir, serão descritos os passos presentes na figura.

1. O PE monitor verifica violações de *deadline,* resultando na necessidade de executar uma técnica de adaptabilidade, como descrito no Capítulo 4. Com isso, ele envia um pacote de solicitação de migração de tarefa ao PE mestre.

2. O PE mestre recebe o pacote de adaptabilidade e verifica a solicitação de efetuar a migração de determinada tarefa da aplicação monitorada. Ao receber o pacote, o PE mestre executa a heurística para saber qual tarefa deve ser migrada e computa qual será a nova posição dessa tarefa. Note que a tarefa que será migrada continua sua execução, em paralelo. A heurística de decisão será apresentada nas Seções 6.2 e 6.3.
3. O PE escravo, que está executando a tarefa escolhida para ser migrada (T1), recebe um pacote com a definição da nova posição da tarefa. A tarefa pode ser migrada se e somente se ela está em execução (não pode estar em estado bloqueado, esperando dados de outro PE).
4. Se a tarefa pode ser migrada, o *microkernel* do PE executando a tarefa envia ao PE alvo (T1*) um pacote com todo o conteúdo da página (com código e dados) da tarefa e sua TCB. A tarefa migrada é escalonada em sua nova posição, uma vez que o código objeto, os dados e a TCB foram completamente recebidos.
5. Ao final do processo de migração **não** há notificação para outros PEs que a tarefa migrou para uma nova posição. Para que a comunicação ocorra de forma correta, foi efetuada uma alteração no protocolo de comunicação `read_request`, para que seja enviada a nova posição da tarefa. As solicitações de mensagens, `message_requests`, são enviadas para a posição original da tarefa. Havendo dados no *pipe*, eles são enviados para a tarefa que solicitou os dados, sem a transmissão da nova posição. Assim, a tarefa receptora continua usando a posição original da tarefa que foi migrada.
6. Quando o *pipe* não contiver mais dados da tarefa migrada, a `message_request` é encaminhada para a nova posição da tarefa.
7. A tarefa migrada envia a mensagem solicitada, com sua nova posição, e o PE receptor atualiza sua tabela de tarefas.

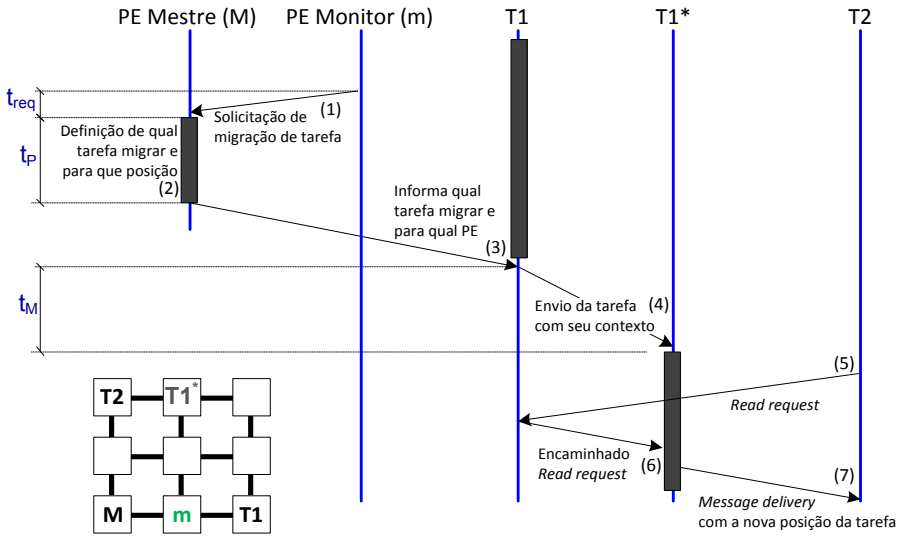

Figura 21 – Protocolo de migração de tarefas com atividades por processador.

O desempenho do protocolo de migração de tarefas é a função t_{req} e t_P, tempo para solicitar a migração e computar a nova posição da tarefa; e t_M, tempo para transmitir a tarefa e escalona-la em um novo PE. Durante t_{req} e t_P a tarefa a ser migrada continua sua execução. Durante t_{mig} a tarefa é preemptada e só retorna sua execução na nova posição

Esse protocolo foi desenvolvido em conjunto com outro membro do grupo de pesquisa do Autor, Guilherme Castilhos, na plataforma HeMPS com gerência de recursos centralizada e distribuído.

6.2 Heurística de Migração de Tarefas com Gerência Centralizada de Recursos

Esta Seção apresentada a heurística de migração de tarefas em uma arquitetura com gerência centralizada de recursos. Esta heurística define qual tarefa, da aplicação monitorada, deve ser migrada e para qual processador ela deve ser alocada. Como dito no início deste Capítulo, a heurística de migração de tarefas é realizada de forma diferente entre a gerência centralizada e a distribuída. Em relação ao protocolo apresentado na Figura 21, a heurística de migração de tarefas corresponde à segunda etapa do protocolo de migração de tarefas.

O processo de migração de tarefas foi desenvolvido visando reduzir o tráfego de dados na rede. Para reduzir o congestionamento na rede, precisamos manter as tarefas sendo executadas em processadores vizinhos, visando reduzir a carga de comunicação na NoC. A Figura 22 apresenta o pseudocódigo da heurística de migração de tarefas. Este processo está dividido nas seguintes etapas:

1. O *microkernel* do processador detecta que uma determinada tarefa está perdendo *deadlines*, solicitando o serviço de migração de tarefas ao PE mestre. Baseado na etapa de *profiling*, o PE mestre verifica qual tarefa (t_M) da aplicação deve ser migrada. A tarefa t_M continua a ser normalmente escalonada, passando para o estado *migrating* apenas quando vier a confirmação do mestre com o endereço do PE destino.
2. O PE mestre, ao receber a solicitação de migração, inicia a execução da heurística apresentada na Figura 22, responsável por definir qual PE receberá a tarefa t_M.
3. O PE mestre define o conjunto N de todos os processadores que possuem tarefas comunicantes com a tarefa t_M. A partir de {N}, o nodo mestre define um segundo conjunto P de processadores que podem receber t_M (linha 4). Caso exista apenas um elemento em {N}, n0, {P} conterá n0 e os PEs distantes 1 hop de n0. Caso |N|>1, P conterá os processadores contidos no quadrado envolvente que contém todos os processadores de {N}.
4. É localizado o processador central (p_c) em {P} (linha 6). Este processador corresponde à primeira opção de endereçamento para recepção de t_M. Caso |N|=1, p_c será o PE n0 (assume-se que os processadores podem executar várias tarefas

simultaneamente). Caso |N|>1, as coordenadas de p_c correspondem à posição central do quadrado envolvente, caso as dimensões do quadrado envolvente forem pares. Logo, p_c será o PE mais próximo do PE onde está a tarefa comunicante.

5. Caso p_c tenha condições de receber t_M e o custo de energia na comunicação (que é a distância em *hops* entre as tarefas comunicantes) de t_M mapeando em p_c com todos os elementos de {N}, segundo a equação 1, for reduzido atribui-se p_c como candidato a receber t_M (linhas 8-11). O consumo entre as tarefas comunicantes é o parâmetro para a tomada de decisões.

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

Onde: E_{sbit}: consumo de energia de cada roteador (esse valor é igual para todos os roteadores); n_{hops}:distância entre os roteadores comunicantes.

6. Na sequência da heurística inicia-se um laço que busca o processador que possa receber t_M. O laço entre as linhas 14-18 percorre o conjunto P buscando o processador que contém o menor custo de comunicação.

Input: t_M: tarefa a ser migrada; p_M: PE executando t_M; N: conjunto que contem tarefas comunicantes com t_M
Output: p_T: PE que receberá a tarefa t_M

```
1.  migration_cost = cost(p_M,t_M,N)        // custo inicial da migração, com o custo de t_M na posição original
2.  p_T ← NULL
3.  // define o conjunto de PEs que podem receber t_M
4.  P ← processor_box(N)
5.  // define o PE central do conjunto P
6.  p_c ← center(P)
7.  // se o PE central pode receber t_M, o custo inicial da migração é calculado, e p_T recebe p_c
8.  IF migrate(p_c,t_M )=TRUE and cost(p_c,t_M,N) < migration_cost THEN
9.     migration_cost = cost(p_c,t_M,N)
10.    p_T ← p_c
11. ENDIF
12. DO
13.    // verifica os outros PEs em P
14.    FOR ALL ELEMENTS p_i IN P
15.       IF cost(p_i,t_M,N)<migration_cost THEN
16.          migration_cos t= cost(p_i,t_M,N)
17.          p_T ← p_i
18.    ENDIF
19.    // verifica se é possível efetuar a migração
20.    IF p_T ≠ NULL THEN
21.       return p_T
22.    ELSE
23.       P´ ← extend_search_space_1_hop(P)
24.       IF P´ = P THEN
25.          return NULL          // não é possível efetuar a migração
26.       ENDIF
27.       P ← P´
28    ENDIF
29. ENDDO
```

Figura 22 – Pseudo-código da heurítica de migração com gerênica centralizada de recursos.

7. Havendo processador candidato, retorna-se sua coordenada (linhas 20-21). Caso

contrário, o espaço de busca é estendido (linha 23), incrementa-se 1 hop e os novos elementos são acrescentados em {P}.

8. Caso não seja possível agregar novos elementos em {P}, a migração não é executada (linha 25).

6.3 Heurística de Migração de Tarefas com Gerência Distribuída de Recursos

Esta Seção apresenta a heurística de migração de tarefas em uma arquitetura com gerência distribuída de recursos. A diferença desta heurística para a heurística apresentada na Seção 6.2, está na forma como o processador decide quem deve ser migrado e para onde migrar. A Figura 23 apresenta o pseudocódigo da heurística de migração de tarefas desenvolvido. Este processo está dividido nas seguintes etapas:

1. Da mesma forma que na gerência centralizada de recursos, o *microkernel* detecta que uma determinada tarefa está perdendo *deadlines*, solicitando o serviço de migração de tarefas ao PE mestre local. Na gerência distribuída de recursos, o PE mestre local verifica quais são as tarefas que estão sendo executadas fora do *cluster* de sua aplicação. Todas as tarefas fora do cluster (tM_i) são candidatas a serem migradas para dentro do *cluster.* As tarefas tM_i continuam sua execução normal, passando para o estado *migrating* apenas quando o PE mestre enviar a solicitação de migração das tarefas com o endereço destino.
2. O PE mestre local da aplicação, ao receber a solicitação de migração, inicia a execução da heurística apresentada na Figura 23, responsável por definir qual PE receberá cada tarefa tM_i.
3. O PE mestre local define o conjunto N com todos os processadores que possuem tarefas comunicantes com a tarefa tM_i. A partir de {N}, o mestre local define um segundo conjunto P dos processadores no cluster da aplicação que podem receber tM_i (linha 4). Caso exista apenas um elemento em {N}, n0, o {P} conterá n0 e os PEs distantes 1 hop de n0. Caso |N|>1, P conterá os processadores contidos no quadrado envolvente que contém todos os processadores de N. Para esta heurística, só serão adicionados ao conjunto P aqueles processadores que pertencerem ao *cluster* da aplicação monitorada.
4. Esta etapa é idêntica a etapa 4 da heurística apresentada na Seção 6.2.
5. Esta etapa é idêntica a etapa 5 da heurística apresentada na Seção 6.2.
6. Esta etapa é idêntica a etapa 6 da heurística apresentada na Seção 6.2.
7. Havendo processador candidato, retorna-se sua coordenada (linhas 20-21). Caso contrário, o espaço de busca é estendido (linha 23), incrementa-se 1 hop e os novos elementos (do *cluster*) são acrescentados em P.
8. Esta etapa é idêntica a etapa 8 da heurística apresentada na Seção 6.2.

Input: t_M: tarefa a ser migrada; p_M: PE executando t_M; N: conjunto que contem tarefas comunicantes com t_M
Output: p_T: PE que receberá a tarefa t_M

```
1.  migration_cost = cost(p_M,t_M,N)          // custo inicial da migração, com o custo de t_M na posição original
2.  p_T ← NULL
3.  // define o conjunto de PEs, do cluster, que pode receber t_M
4.  P←processor_box_in_cluster(N)
5.  // define o PE central do conjunto P
6.  p_c←center(P)
7.  // se o PE central pode receber t_M, o custo inicial da migração é calculado, e p_T recebe p_c
8.  IF migrate(p_c,t_M )=TRUE and cost(p_c,t_M,N) < migration_cost THEN
9.     migration_cost = cost(p_c,t_M,N)
10.    p_T←p_c
11. ENDIF
12. DO
13.    // verifica todos os PEs em P menos o Pc que já foi verificado
14.    FOR ALL ELEMENTS p_i IN P
15.       IF cost(p_i,t_M,N)<migration_cost THEN
16.          migration_cos t= cost(p_i,t_M,N)
17.          p_T←p_i
18.    ENDIF
19.    // verifica se é possível efetuar a migração
20.    IF p_T≠ NULL THEN
21.       return p_T
22.    ELSE
23.       P´← extend_search_space_1_hop_in_cluter(P)
24.       IF P´ = P THEN
25.          return NULL          // migração não é possível
26.       ENDIF
27.       P←P´
28    ENDIF
29. ENDDO
```

Figura 23 – Pseudo-código da heurítica de migração com gerênica distribuída de recursos.

6.4 Interação do Monitoramento com Migração de Tarefas

Como apresentado no Capítulo 4, o monitoramento de tarefas é o mecanismo responsável pela solicitação da alteração de prioridades e *time-slice* das tarefas de uma determinada aplicação, afetando diretamente o escalonamento (Capítulo 5). Em alguns casos, apenas efetuar a troca das características de escalonamento é suficiente para não haver mais violações de *deadlines* pela aplicação. Porém, em outros casos efetuar essa troca não basta e a aplicação continua violando *deadlines.* Nesses casos o monitoramento de tarefas deve solicitar a execução da técnica de migração de tarefas.

Na Figura 24, apresenta-se a interação do monitoramento com a migração de tarefas. Para ilustrar a migração de tarefas foi utilizada arquitetura distribuída de recursos. A etapa 1 da Figura ilustra a configuração inicial da aplicação. O LMP_0 é responsável por gerenciar a aplicação principal. Há quatro tarefas mapeadas no *cluster* da aplicação (A, B, C, F) e duas tarefas mapeadas em *clusters* vizinhos (D, E). Na etapa 2 o PE monitor verifica perdas de *deadlines* e solicita a alteração das características de escalonamento. A etapa 3 apresenta-se os pacotes de solicitação para aumento das prioridades da

aplicação principal. Até este ponto, tem-se a execução do protocolo da técnica de adaptabilidade responsável pelas trocas das características das tarefas, apresentadas no capítulo 5, afetando diretamente o escalonamento.

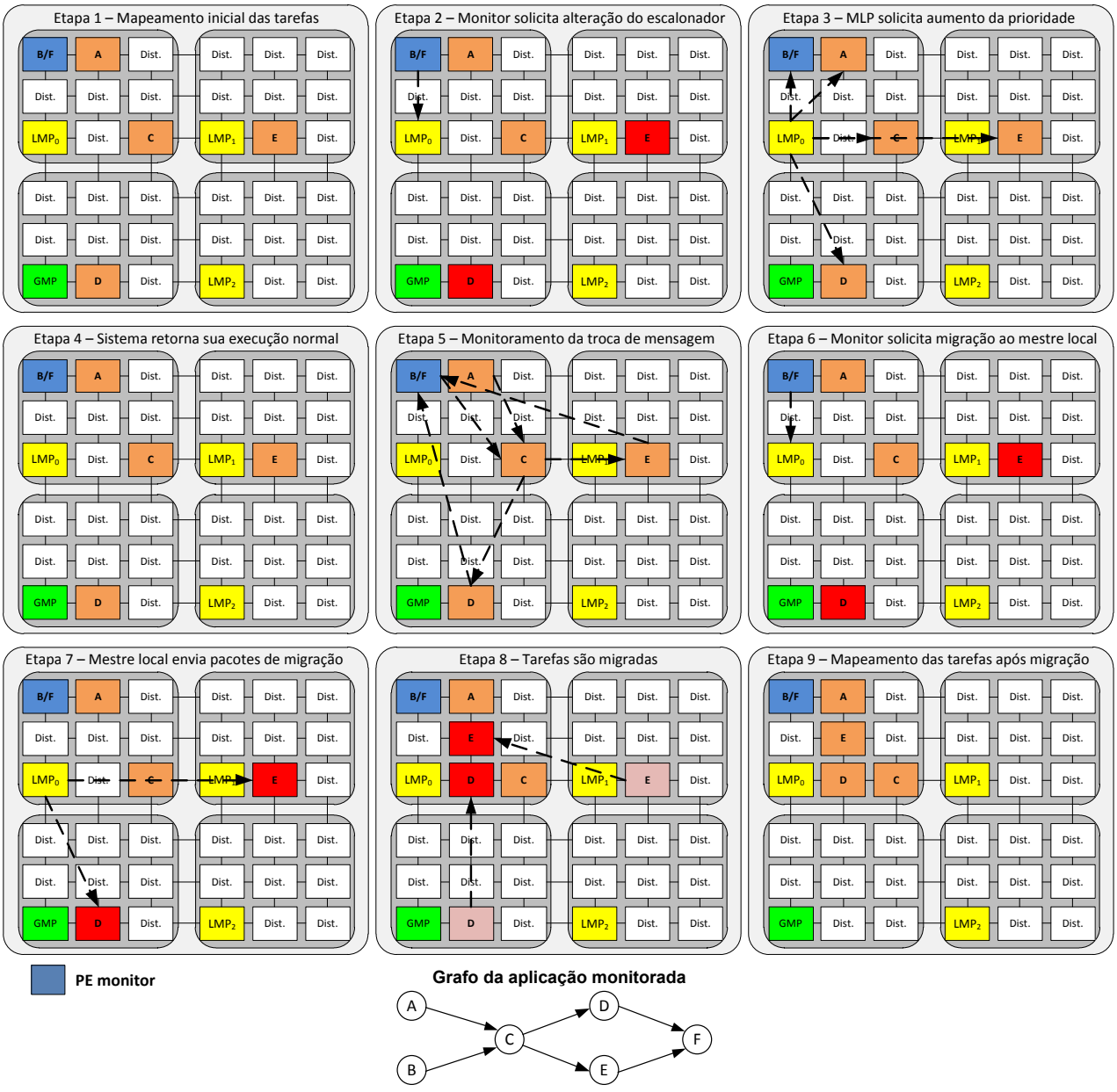

Figura 24 – Mecanismo de monitoramento com migração de tarefas.

Despois que o mestre envia um pacote solicitando para que todas as tarefas da aplicação sejam alteradas (Etapas 1, 2 e 3), o PE monitor continua efetuando o monitorando da aplicação, para verificar se as violações de *deadlines* continuam ocorrendo, etapa 4. Na Etapa 5, o PE monitor verifica as mensagens de comunicação das tarefas. Na Etapa 6, o PE monitor identifica que a quantidade de perdas de *deadline* configurada no `max_miss_migra`, da TCB da tarefa F, atingiu as violações analisadas. Com isso, solicita ao PE mestre local a execução da técnica de adaptabilidade. Como explicado anteriormente, o PE mestre local define que todas as tarefas fora do *cluster* de

sua aplicação devem ser migradas, nesse caso as tarefas D e E. Sabendo quais tarefas migrar (t_m) o PE mestre local, na Etapa 7, define qual será a nova posição de t_m e envia uma solicitação de migração para os PEs escravos que estão executando as tarefa D e E. Na Etapa 5 ocorre a migração das tarefas e logo em seguida, na Etapa 6, as tarefas migradas são escalonadas em suas novas posições.

Assim como descrito na Seção 4.2, com a execução das técnicas de adaptabilidade, o sistema é desfragmentado e não há mais perturbações entre as tarefas comunicantes, melhorando-se o desempenho da aplicação, e reduz-se a energia consumida na NoC.

7 AVALIAÇÃO

Este Capítulo apresenta os resultados da implementação da arquitetura HeMPS com técnicas de adaptabilidade apresentadas nos Capítulos 4, 5 e 6. São analisados dois cenários de execução: o primeiro onde se apresenta uma execução com gerência centralizada de recursos, e outro com a execução em um MPSoC com gerência distribuída de recursos.

Para efetuar a avaliação do trabalho realizado no decorrer do curso de Mestrado, obteve-se resultados experimentais a partir de uma NoC 6x6 instanciada no MPSoC HeMPS [CAR09], apresentado no Capítulo 3. Na Seção 7.1, apresenta-se os resultados obtidos a partir de um MPSoC com gerência centralizada de recursos. Na Seção 7.2, apresente-se as avaliações aplicadas a um MPSoC com gerência distribuída de recursos.

A seguir apresentam-se algumas características relevantes para a avaliação das técnicas de adaptabilidade:

1. Tamanho da página: 16 Kbytes (4.096, código e dados).
2. *Time-slice* das tarefas: 10 Kciclos de relógio, podendo ser aumentado para 16 Kciclos de relógio, conforme decisão do monitoramento de tarefas.
3. Prioridade das tarefas: baixa (255), podendo ser alterada conforme decisão do monitoramento para mais ou para menos.
4. *Deadline* da tarefa monitorada: 4,5 Kciclos de relógio para gerência centralizada e 9 Kciclos de relógio para gerência distribuída.
5. Máxima perda de *deadline* aceitável para solicitar alteração das características de escalonamento: 10. Como apresentado na Seção 4.1 esse é o `max_miss` da TCB. Quando a perda de *deadline* atingir esse valor a prioridade e o *time-slice* das tarefas da aplicação serão alterados.
6. Máxima perda de deadline aceitável para solicitar migração da tarefa: 20. Esse é o `max_miss_migra` da TCB, é o ponto em que a técnica de migração de tarefas deverá ser solicitada ao PE mestre.
7. Escalonador: preemptivo baseado em prioridades, apresentado na Seção 5.3.
8. Roteador da NoC: roteamento XY, arbitragem centralizada *round-robin*, buffer de entrada (profundidade do buffer igual a 16 *flits*).
9. Mapeamento de tarefas: dinâmico, desenvolvido por [MAN11].

7.1 Avaliação de um MPSoC com Gerência Centralizada de Recursos

Nesta Seção apresentam-se os primeiros resultados obtidos do presente trabalho. É avaliado o comportamento de um sistema com gerência centralizada de recursos, fazendo

uso das técnicas de adaptabilidade. Na Seção 7.1.1 tem-se a descrição do cenário de teste. Na Seção 7.1.2 apresenta-se os resultados obtidos deste cenário descrito.

7.1.1 Aplicação Experimental

Nesta Seção apresenta-se o cenário experimental configurado para a obtenção dos dados apresentados na Seção 7.1.2.

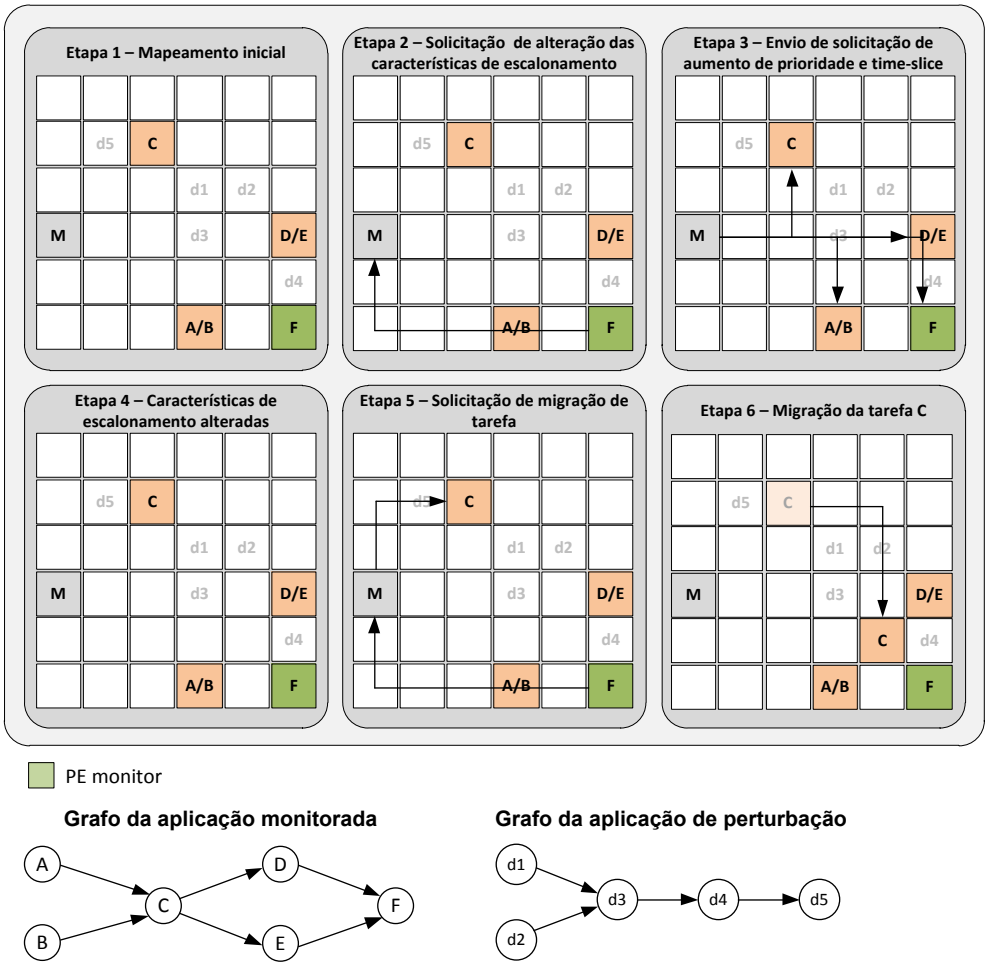

Figura 25 – Cenário para avaliação das técnicas de adaptabilidade em MPSoC com gerência centralizada de recursos.

Na Figura 25 apresenta-se o cenário utilizado para a obtenção dos dados desta Seção. A Etapa 1 ilustra o mapeamento inicial das tarefas. A aplicação monitorada está em destaque, composta por 6 tarefas (A, B, C, D, E, F). A tarefa F contém a configuração necessária para que o processador, que a está executando, faça o monitoramento de sua aplicação. Em um cenário dinâmico, onde aplicações são inseridas/removidas em tempo de execução, o MPSoC pode estar com muitos de seus recursos em uso. Em um dado momento, novas aplicações podem ser carregadas no MPSoC, com comunicações competindo com a comunicação de aplicação principal. Desta forma, o desempenho da aplicação principal é penalizado e o *microkernel* do PE responsável pelo monitoramento solicita ao PE mestre (M) o aumento da prioridade e do *time-slice* das tarefas da aplicação monitorada, Etapa 2. Ao receber a solicitação, na Etapa 3, o PE mestre envia pacotes de

solicitação de alteração das características do escalonamento aos PEs escravos, que executam as tarefas da aplicação monitorada. A Etapa 4 ilustra o PE monitor continuando seu monitoramento, agora com as alterações no escalonamento. Nota-se, na Etapa 5, que as alterações de escalonamento não foram suficientes para melhorar o desempenho da aplicação. Assim, o PE monitor envia um pacote ao PE mestre solicitando migração de alguma tarefa da aplicação. Baseado nos dados coletados na etapa de *profiling,* o PE mestre decide qual tarefa deve ser migrada, neste caso, tarefa C. Sabendo qual tarefa deve migrar, o PE mestre escolhe uma nova posição e, ainda na Etapa 5, envia um pacote solicitando a migração desta tarefa. Na Etapa 6 a tarefa é migrada para a posição mais próxima entre suas tarefas comunicantes, restaurando o desempenho da aplicação e economizando energia na comunicação.

7.1.2 Resultados Obtidos

Nesta Seção apresenta-se a avaliação da aplicação experimental descrita na Seção anterior, apresentando e analisando gráficos dos tempos de iteração da aplicação principal para o cenário de teste. Estes resultados foram publicados em [MOR12b].

O desempenho do protocolo de monitoramento com a execução das técnicas de adaptabilidade foi avaliado de acordo com os quatro cenários descritos a seguir, baseado no mapeamento das tarefas apresentados na Figura 25:

1. Aplicação principal executando sozinha no MPSoC;
2. Aplicação principal executando juntamente com outra aplicação, causando perturbação na comunicação entre as tarefas da aplicação principal;
3. Mesmo cenário que o ‘2’, mas com o monitoramento de tarefas solicitando alteração nas características do escalonamento das tarefas da aplicação principal;
4. Mesmo cenário que o ‘3’, mas com o monitoramento de tarefas verificando a necessidade de efetuar a migração de alguma tarefa da aplicação principal, no caso, tarefa C.

A aplicação principal executa 150 interações para todos os cenários. As Figura 26, Figura 27, Figura 28 e Figura 29 apresentam os gráficos com o tempo das iterações da Tarefa C (a tarefa que deve ser migrada) da aplicação principal executando, respectivamente, nos cenários ‘1’, ‘2’, ‘3’, ‘4’. As tarefas iniciais da aplicação (‘A’ e ‘B’) são as primeiras a serem mapeadas, produzindo dados para as tarefas que ainda não estão mapeadas. Quando as tarefas consumidoras são mapeadas, elas podem consumir mais depressa, o que explica as pequenas variações de tempo no início do processo.

A Figura 26 (cenário ‘1’) ilustra que, após o mapeamento das tarefas consumidoras, o tempo das iterações passa a ser praticamente constante, oscilando entre 3800 e 4500 ciclos de relógio. Este é considerado o cenário de referência para os demais dados desta Seção.

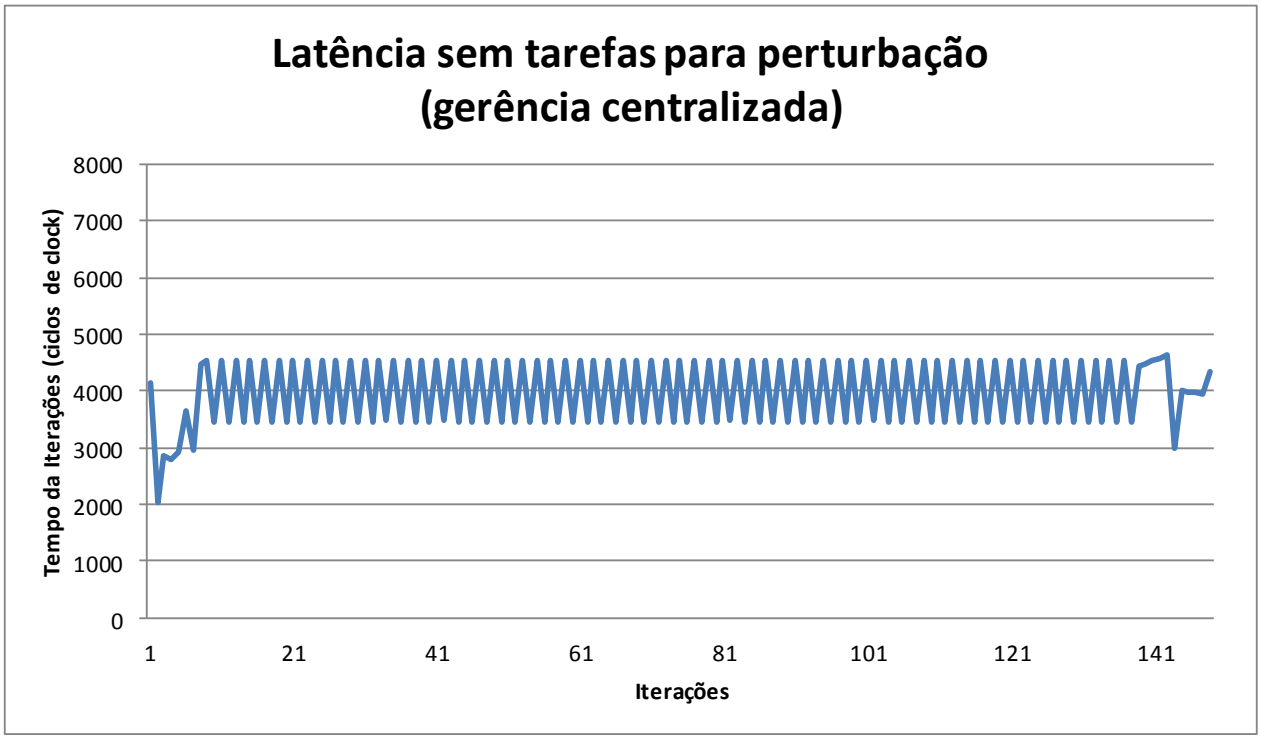

Figura 26 – Gráfico de analise de tempo das iterações da tarefa F, aplicação sozinha no sistema (gerência centralizada).

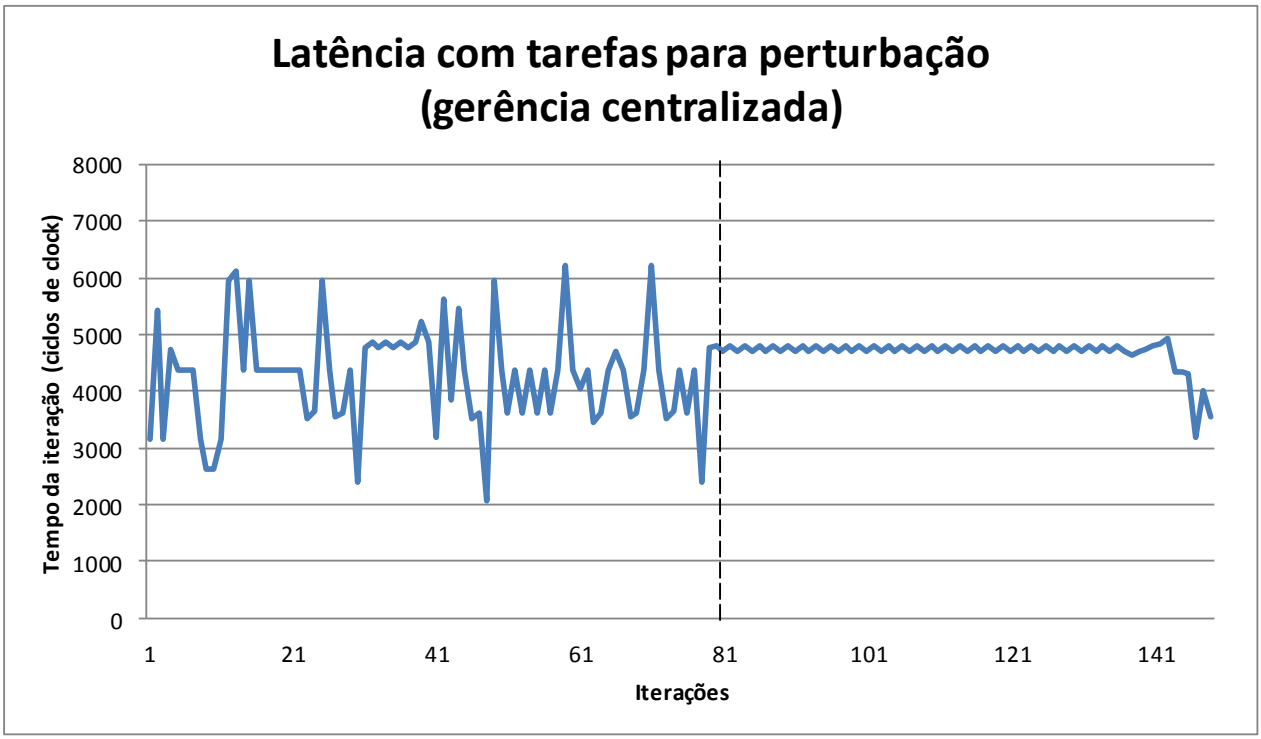

Figura 27 – Gráfico de analise de tempo das iterações da tarefa F, com perturbação na comunicação sem uso de monitoramento (gerência centralizada).

A Figura 27 (cenário ‘2’) mostra que o tempo das iterações, mesmo depois de mapeadas as tarefas consumidoras, continua com alta variação. Esta variação ocorre por causa das aplicações de perturbação mapeadas no sistema. Essas aplicações atrasam a

comunicação da aplicação inicial, o que explica tal variação, com altos picos. O tempo se torna constante após a iteração 80, pois as tarefas de perturbação são finalizadas, como em destaque na figura. A oscilação de tempo, após o término das tarefas de perturbação, fica alta pelo fato de que as tarefas de perturbação se comunicam com o PE mestre para finalizar a tarefa.

A Figura 28 (cenário ‘3’) apresenta-se o tempo de iteração com o monitoramento analisando os dados da tarefa F. Podemos notar poucas variações de tempo, pois após algumas perdas de *deadline* (`max_miss`) o PE escravo, responsável pelo monitoramento da aplicação, solicita ao PE mestre a troca da prioridade e o aumento do *time-slice* de todas as tarefas da aplicação principal. Dessa forma notamos que em alguns períodos o tempo fica praticamente constante em 5000 ciclos de relógio, como podemos ver entre as linhas tracejadas.

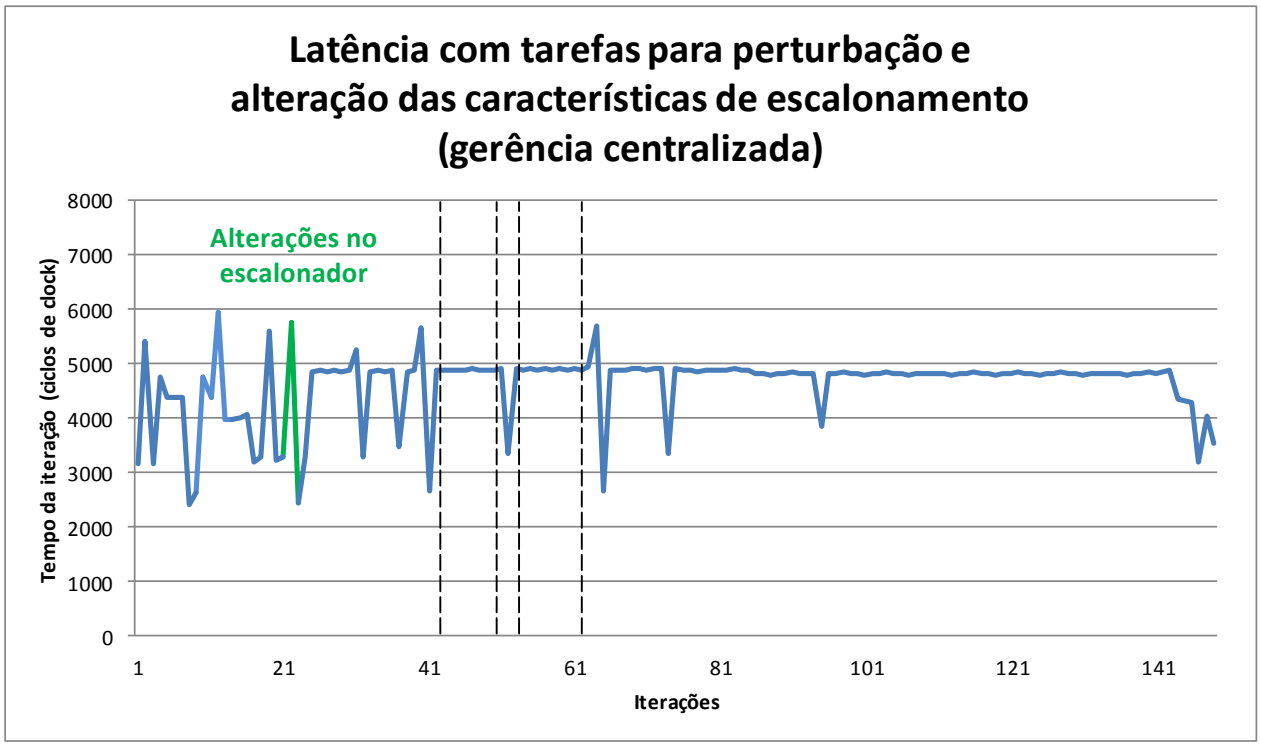

Figura 28 – Gráfico de analise de tempo das iterações da tarefa F, com uso de monitoramento solicitando alteração de características de escalonamento (gerência centralizada).

Na Figura 29 (cenário ‘4’) tem-se a execução da técnica de migração de tarefas. Nesse caso, a tarefa F (tarefa que está sendo monitorada) continua com perdas de *deadline,* mesmo após as alterações no escalonamento na iteração 20. Podemos notar que na iteração 40 a tarefa C é migrada, havendo uma elevação no pico de tempo. Esse pico ocorre devido ao tempo de execução do protocolo de migração. Após a migração, notamos que o tempo das iterações passa a ser praticamente constante, oscilando entre 3500 e 4100 ciclos de relógio. Dessa forma, este cenário demonstra que a migração da tarefa C atingiu o objetivo desejado, melhorando o desempenho do sistema, em

comparação com a Figura 26, cenário com a aplicação sozinha no sistema. E comparado ao cenário com aplicações de perturbação, Figura 27, o ganho é maior, passando a latência de, aproximadamente, 5000 ciclos de relógio para 3500 ciclos de relógio.

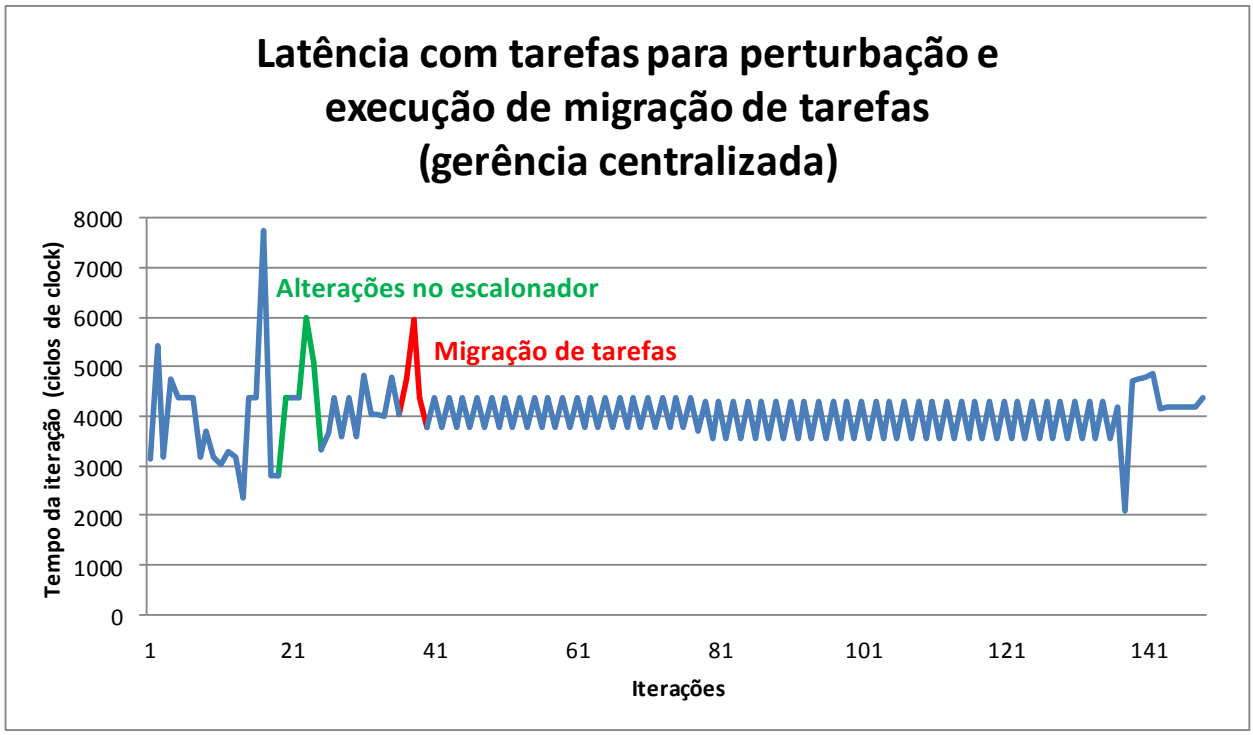

Figura 29 – Gráfico de analise de tempo das iterações da tarefa F, com uso de monitoramento solicitando migração de determinada tarefa (gerência centralizada).

Como pode-se verificar nos resultados apresentados, as técnicas de adaptabilidade atingiram seus objetivos em diminuir as violações de *deadline* da aplicação monitorada. Na Tabela 2, podemos analisar que com a execução dessas técnicas o desempenho do sistema não foi penalizado, baseando-se no tempo total de execução do cenário com tarefas para perturbação.

Tabela 2 – Tempo total de execução da aplicação principal com gerência centralizada.

Tempo total de execução da aplicação principal		
Cenário	**Tempo total de execução**	
	Tempo	**Diferença (%)**
Com tarefas para perturbação	1705706	Referência
Alteração de prioridade	1632863	-4,3%
Migração de tarefas	1577019	-7,5%

Com a execução das técnicas de adaptabilidade, o tempo total de execução da aplicação principal não é afetado, nesse caso, melhorando-se em até 7,5%. Isso

demonstra que as técnicas de adaptabilidade proveem redução de latência e jitter, sem afetar o tempo de execução total do sistema.

7.2 Avaliação de um MPSoC com Gerência Distribuída de Recursos

Após apresentar os resultados obtidos com a utilização de um MPSoC com gerência centralizada de recursos, nesta Seção, apresentam-se os resultados obtidos com o uso de um MPSoC com gerência distribuída de recursos. É avaliado o comportamento do sistema, fazendo uso das técnicas de adaptabilidade. Na Seção 7.2.1 tem-se a descrição do cenário de teste. Na Seção 7.2.2 apresenta-se os resultados obtidos através deste cenário descrito.

7.2.1 Aplicação Experimental

Nesta Seção apresenta-se o cenário experimental configurado para a obtenção dos dados apresentados na Seção 7.2.1.

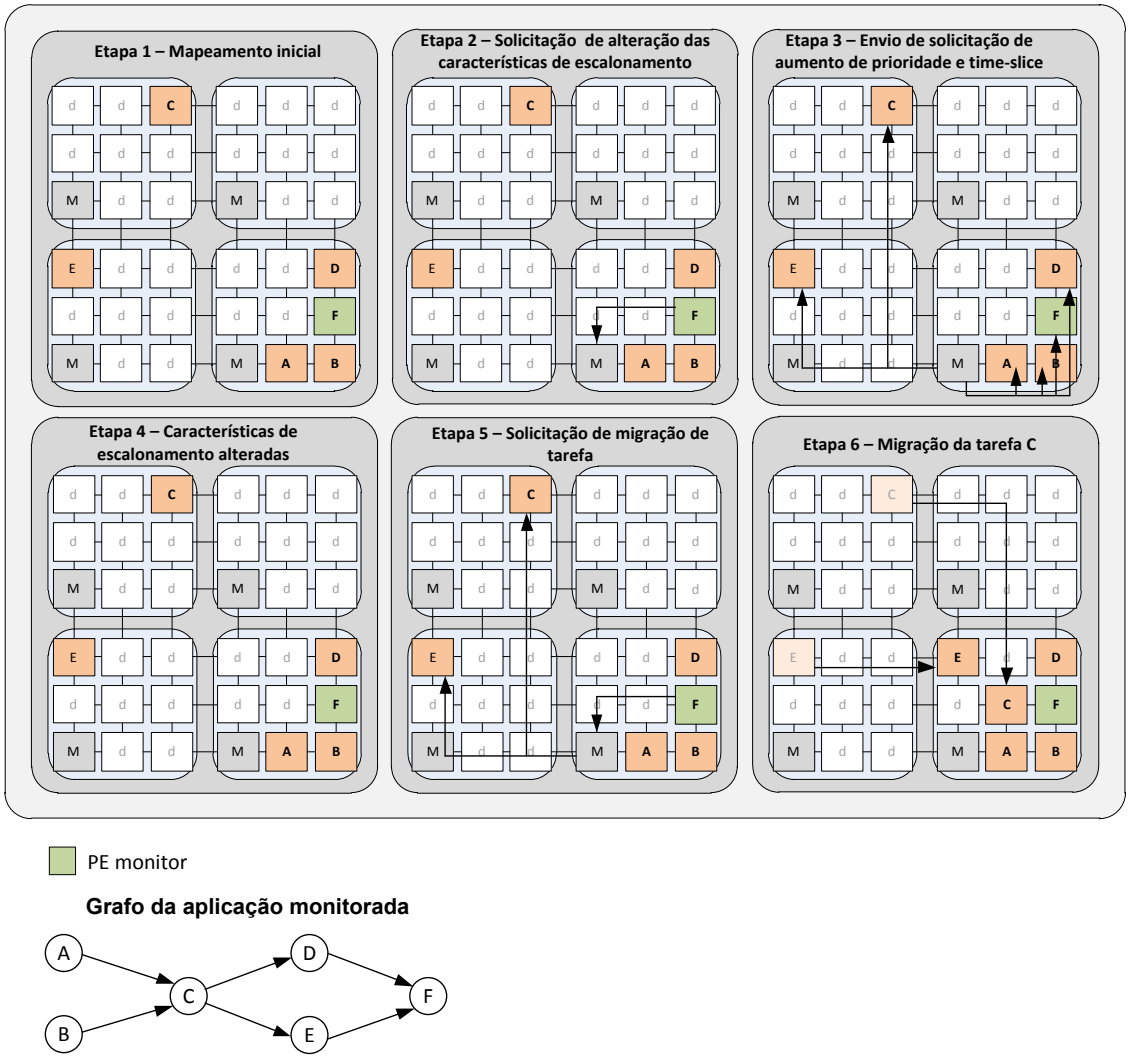

Figura 30 – Cenário para avaliação das técnicas de adaptabilidade em MPSoC com gerência distribuída de recursos.

Na Figura 30 apresenta-se o cenário utilizado para a obtenção dos dados analisados na próxima Seção. Da mesma forma que na Figura 25, a aplicação monitorada está em destaque, composta por 6 tarefas (A, B, C, D, E, F). A tarefa F contém a configuração necessária para que o processador, que está executando ela, faça o monitoramento de sua aplicação. O mapeamento, no MPSoC com gerência distribuída, é dinâmico e tenta mapear as tarefas próximas umas das outras. Por isso, para este cenário, usam-se diversas aplicações para perturbação, fazendo com que a aplicação principal (que é a última a ser mapeada) fique com algumas tarefas fora do *cluster* de sua aplicação, deixando o sistema fragmentado devido ao fato de não conter espaços para todas as tarefas no mesmo *cluster*. As Etapas de 1 a 4, deste cenário, são as mesmas apresentadas na Seção 7.1.1. A diferença entre o cenário distribuído para o centralizado é no protocolo de migração. Na gerência centralizada deve-se migrar uma tarefa da aplicação. Na gerência distribuída devem-se migrar todas as tarefas que estiverem fora do *cluster* de seu mestre, se possível. Desta forma, na Etapa 5 o PE mestre envia uma solicitação de migração para as tarefas C e E que executam fora do *cluster.* Na Etapa 6 nota-se a desfragmentação do sistema, fazendo com que a aplicação principal seja executada com suas tarefas próximas umas das outras. Assim, é possível economizar energia na comunicação e aumentar o desempenho do sistema.

7.2.2 Resultados Obtidos

Nesta Seção apresenta-se a avaliação da aplicação experimental descrita na Seção anterior, apresentando e analisando gráficos dos tempos de iteração da aplicação principal para o cenário de teste.

O desempenho do protocolo de monitoramento com a execução das técnicas de adaptabilidade, em um sistema com controle de recursos distribuído, foi avaliado de acordo com os três cenários descritos a seguir, baseado no mapeamento das tarefas apresentados na Figura 30:

1. Aplicação principal executando juntamente com outras aplicações, causando perturbação na comunicação entre as tarefas da aplicação monitorada;
2. Mesmo cenário que o ‘1’, mas com o monitoramento de tarefas solicitando alteração nas características do escalonamento das tarefas da aplicação principal;
3. Mesmo cenário que o ‘2’, mas com o monitoramento de tarefas verificando a necessidade de efetuar a migração de algumas tarefas da aplicação principal, no caso, as tarefas C e E que estão fora de seu *cluster*.

A aplicação principal executa 150 interações para todos os cenários. A Figura 31, Figura 32 e Figura 33 apresentam os gráficos com o tempo das iterações da Tarefa F (tarefa monitorada) da aplicação principal executando, respectivamente, nos cenários ‘1’, ‘2’ e ‘3’. As tarefas iniciais da aplicação (‘A’ e ‘B’) são as primeiras a serem mapeadas,

produzindo dados para as tarefas que ainda não estão mapeadas. Quando as tarefas consumidoras são mapeadas, elas podem consumir mais depressa, o que explica as variações de tempo no início do processo.

A Figura 31 (cenário ‘1’) mostra que o tempo das iterações, mesmo depois de mapeadas as tarefas consumidoras, continua com alta variação. Esta variação ocorre por causa das aplicações de perturbação mapeadas no sistema. Após o mapeamento das tarefas consumidoras, o tempo das iterações passa a ser praticamente constante, atingindo 10 kciclos de relógio por iteração. Este é considerado o cenário de referência para os próximos cenários analisados.

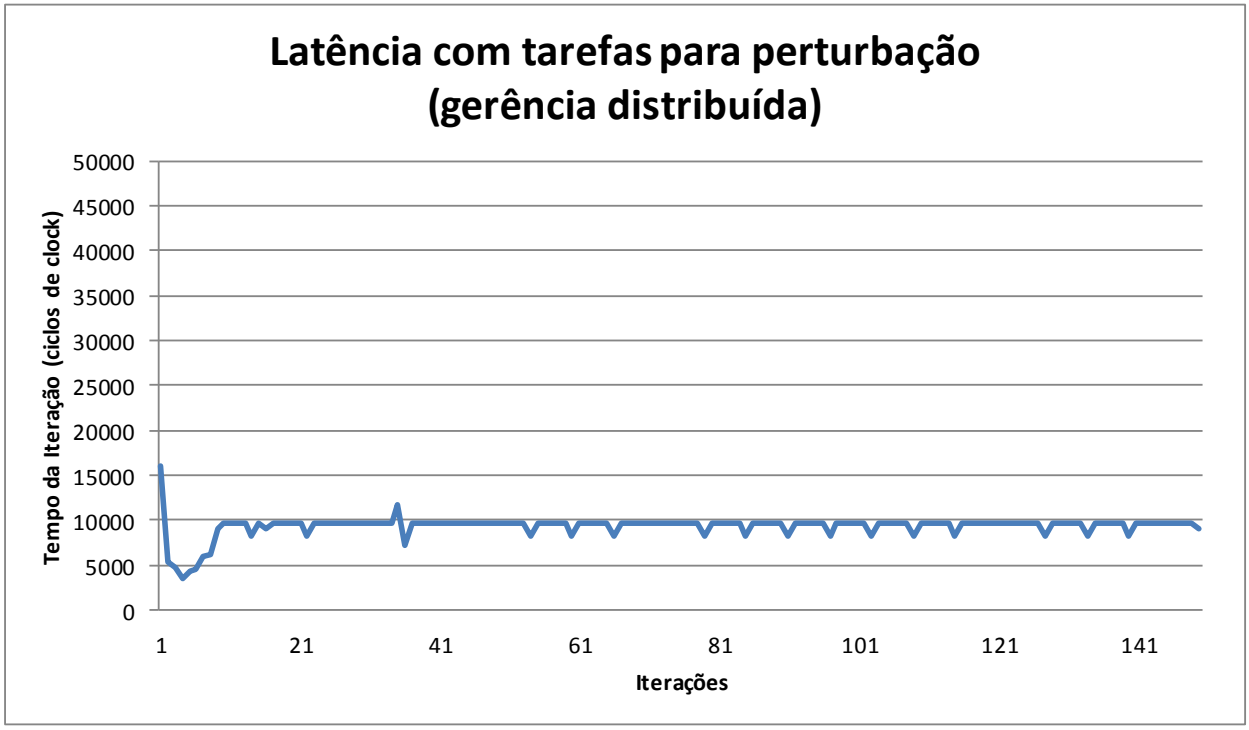

Figura 31 – Gráfico de analise de tempo das iterações da tarefa F, sem uso de monitoramento (gerência distribuída).

A Figura 32 (cenário ‘2’) apresenta o tempo de iterações com o monitoramento da aplicação principal. Podemos notar que temos a variações de tempo oscila entre 8,5 e 10 kciclos de relógio, pois após algumas perdas de *deadline* (`max_miss`) o PE monitor solicita ao PE mestre a troca da prioridade e o aumento do *time-slice* de todas as tarefas da aplicação principal. Dessa forma, notamos que após a iteração 41, aonde ocorrem as trocas das características de escalonamento, o sistema apresenta alguns ganhos relativos ao cenário de referência. Porém, a aplicação segue com violações de *deadline*. Assim, na Figura 33, tem-se a executação da técnica de migração de tarefas.

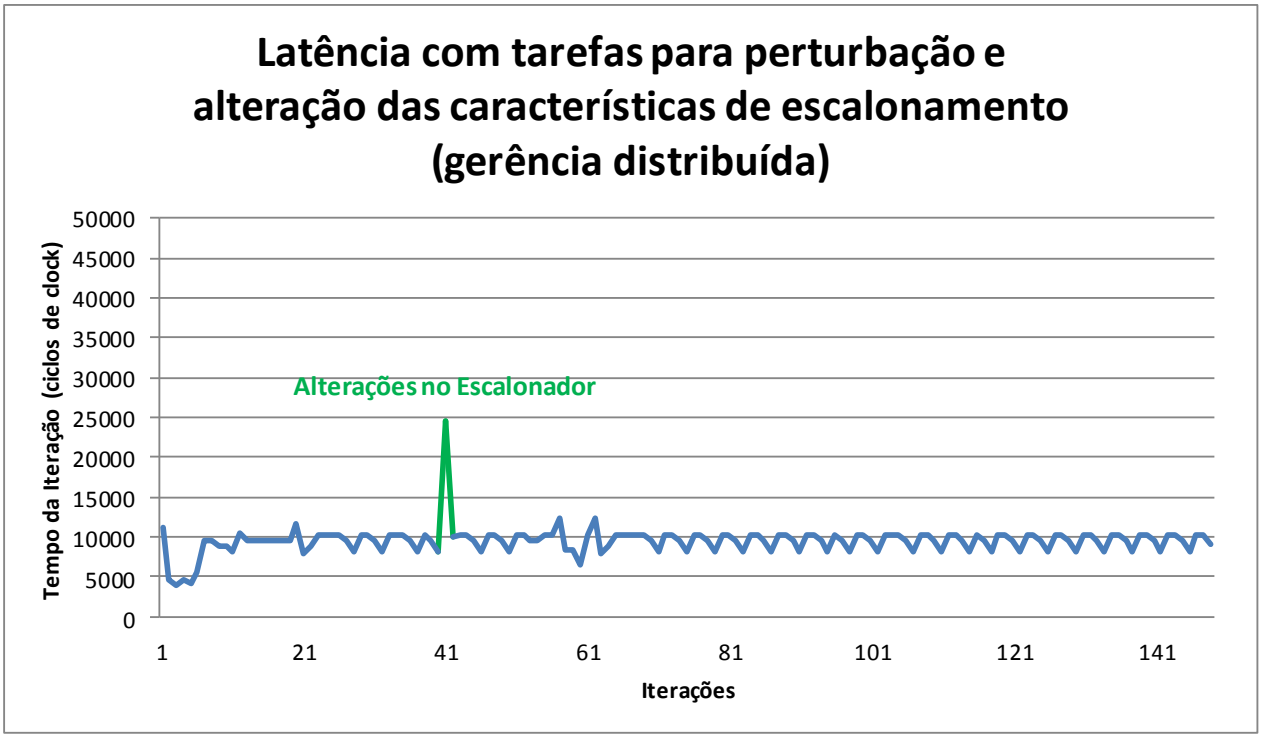

Figura 32 – Gráfico de analise de tempo das iterações da tarefa F, com uso de monitoramento solicitando alteração de características de escalonamento (gerência distribuída).

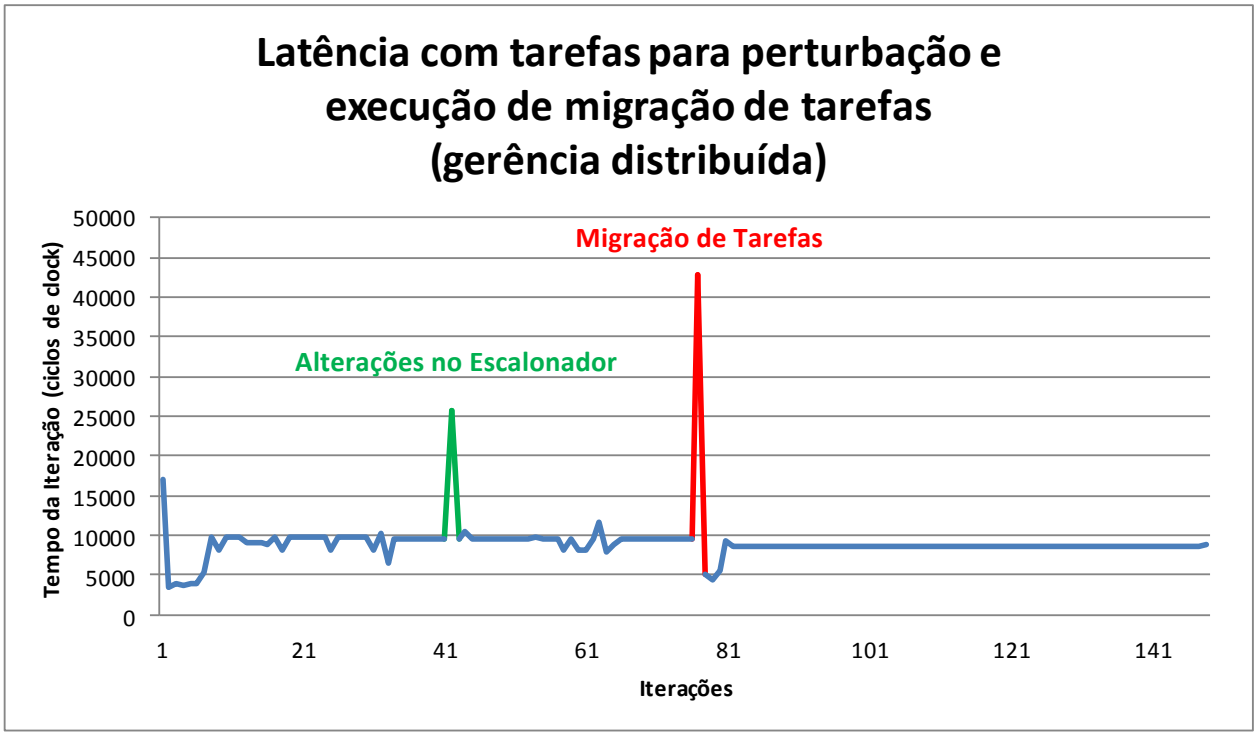

Figura 33 – Gráfico de analise de tempo das iterações da tarefa F, com uso de monitoramento solicitando migração das tarefas fora do *cluster* (gerência distribuída).

Na Figura 33 (cenário ‘3’) temos a execução da técnica de migração de tarefas. Nesse caso, a tarefa F (tarefa que está sendo monitorada) continua com perdas de *deadline* após alterações no escalonador. Podemos notar que, aproximadamente, na iteração 80 as tarefas C e E são migradas, havendo um alto pico de tempo. Esse pico

ocorre devido ao tempo de execução do protocolo de migração descrito no Capítulo 6. Após as migrações, notamos que o tempo das iterações passa a ser constante em 8,5 kciclos de relógio, abaixo do tempo apresentado no cenário de referência (Figura 31) que se mostra entre 9,3 e 10 kciclos de relógio. Desta forma, a tarefa parou de perder *deadline*, que inicialmente estava configurado em 9 kciclos de relógio.

Assim como nos cenários de gerência centralizada de recursos, apresentados na Seção 7.1, as técnicas de adaptabilidade atingiram seus objetivos, também na gerência distribuída de recursos, reduzindo a quantidade de violações de *deadline* da aplicação monitorada. Na Tabela 3, analisa-se o tempo total de execução, em ciclos de relógio, da aplicação principal.

Tabela 3 – Tempo total de execução da aplicação principal com gerência distribuída de recursos.

Tempo total de execução da aplicação principal		
Cenário	**Tempo total de execução**	
	Tempo	**Diferença (%)**
Com tarefas para perturbação	1687091	Referência
Alteração de prioridade	1698619	+0,7%
Migração de tarefas	1637316	-3,0%

Com a execução das técnicas de adaptabilidade, o tempo total de execução da aplicação principal não é penalizado, nesse caso, melhorando-se em até 3%. A diferença de tempo entre o cenário de referência e o cenário com alterações nas características de escalonamento é o tempo gasto pelo protocolo adaptativo, que solicita as alterações de prioridade e *time-slice* das tarefas da aplicação monitorada. Da mesma forma que na gerência centralizada, demonstra-se que as técnicas de adaptabilidade proveem redução de latência e jitter, sem afetar o tempo de execução total do sistema.

8 CONCLUSÃO E TRABALHOS FUTUROS

Esta Dissertação teve como principal objetivo apresentar técnicas de controle adaptativo para atendimento a requisitos de aplicações em MPSoCs. Descreveu-se uma técnica de análise de dados em tempo de execução, que é o monitoramento de tarefas, apresentado no Capítulo 4. Também, foram apresentadas duas técnicas para prover a adaptabilidade: (*i*) alteração das características do escalonamento das tarefas, apresentado no Capítulo 5; (*ii*) migração de tarefas, aproximando tarefas comunicantes e desfragmentando o sistema, apresentado no Capítulo 6.

Como ponto de partida desta Dissertação, temos a plataforma HeMPS centralizada e distribuída, como descrito no Capítulo 3. Esta plataforma foi modificada a fim de serem adicionadas técnicas de adaptabilidade para MPSoCs. Essas técnicas foram estudadas e analisadas dentro de um contexto de estado-da-arte, apresentado no Capítulo 2. Dentre as modificações efetuadas na plataforma, destacam-se o desenvolvimento do monitoramento de tarefas, heurística de migração de tarefas e adaptação do algoritmo de escalonamento preemptivo baseado em prioridades.

A implementação das técnicas de adaptabilidade envolveu modificações de software (porte do kernel). A primeira contribuição deste trabalho foi discutida durante o Capítulo 4, onde apresenta-se a forma de como uma aplicação deve ser monitorada para aplicar técnicas a fim de evitar violações de *deadline*. A segunda contribuição deste trabalho foi detalhada no Capítulo 5, demonstrando a operação do escalonador adaptado neste trabalho, apresentando sua sequência de criação divida em três etapas: (i) algoritmo base (*Round-Robin*); (*ii*) aprimoramento do algoritmo base, adicionando prioridade às tarefas; (*iii*) algoritmo final, com alteração da prioridade e *time-slice* em tempo de execução. A terceira contribuição deste trabalho foi descrita no Capítulo 6, onde apresentou-se a técnica de migração de tarefas, protocolo e heurística, possibilitando a desfragmentação do sistema, economia de energia e melhoras no desempenho das aplicações. Com isso, os resultados analisados mostram que, independente da gerência de recursos que se utiliza, centralizada ou distribuída, as técnicas de adaptabilidade proveem redução de latência e jitter, sem comprometimento do tempo total de execução das aplicações.

A contribuição mais relevante deste trabalho é que as técnicas de adaptabilidades adicionadas na plataforma MPSoC, centralizada e distribuída, estão validadas e operacionais. Ou seja, as duas arquiteturas da plataforma executam um *microkernel* com suporte a controle adaptativo do sistema.

8.1 Publicações Relacionadas ao Tema da Dissertação

As contribuições apresentadas neste trabalho foram publicadas na forma de dois artigos em Conferências:

- 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.
- MORAES, F.G ; CARARA, E. ; RUARO, M. ; MADALOZZO, G.. "***Evaluation of Adaptive Management Techniques in NoC-Based MPSoCs***". In: ICECS 2012, pp. 548-551.

8.2 Trabalhos Futuros

Como trabalhos futuros enumeram-se as seguintes atividades:

- Adicionar as técnicas de adaptabilidade, apresentadas neste trabalho, a uma plataforma com suporte à tolerância a falhas.
- Incluir outras técnicas de adaptabilidade: prioridade de comunicação, DVFS, comutação por circuito (*circuit-switching*).
- Generalizar os monitores para avaliarem outros requisitos de aplicação, como tempo de execução ou vazão.
- Avaliar e implementar a utilização de uma rede dedicada para o fluxo de dados do gerenciamento.

REFERÊNCIAS

[ACQ07] Acquaviva, A; Carta, S; Mereu, F; Micheli, G. “MiGra: a task migration algorithm for reducing temperature gradient in multiprocessor systems on chip”. In: SoC, 2007, 6p.

[AGU08] Aguiar, A; Filho, J. “Architectural Support for Task Migration Concerning MPSoC”. In: WSO, 2008, pp.169-178.

[BAR08] Barcelos, D. “Modelo de Migração de Tarefas para MPSoCs baseados em Redes-em-chip”. Dissertação de Mestrado, UFRGS, 2008, 92p.

[CAR09] Carara, E; Oliveira, R; Calazans, N; Moraes, F. "HeMPS - a Framework for NoC-based MPSoC Generation". In: ISCAS, 2009, pp. 1345-1348.

[CAR11] Carara, E. “Serviços de Comunicação Diferenciados em Sistemas Multiprocessados em Chip Baseados em Redes Intra-Chip”. Tese de Doutorado, PUC-RS, 2011, 107p.

[CIO06] Ciordas, C; Goossens, K; Basten, T. “NoC Monitoring: Impact on the Design Flow”. In: ISCAS, 2006, pp1981-1984.

[COS07] Coskun, A; Rosing, T; Whisnant, K. “Temperature Aware Task Scheduling in MPSoCs”. In: DATE, 2007, 6p.

[FAT11] Fattah, M; Daneshtalab, P; Liljeberg, P; Plosila J. “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”. In: Job Scheduling Strategies for Parallel Processing, 1997, pp. 1-34.

[JER05] Jerraya, A. A.; Wolf, W. *“Multiprocessor Systems-on-Chips”*. Morgan Kaufmann, 2005, 602p.

[KOB11] Kobbe, S; Bauer, L; Lohmann, D; Schröder-Preikschat, W; Henkel, J. “DistRM: Distributed Resource Management for On-Chip Many-Core Systems”. In: ISSS, 2011, pp. 119-128.

[LAY09] Layouni, S; Benkhelifa, M; Verdier, F; Chauvet, S. “Multiprocessor task migration implementation in a reconfigurable platform”. In: ReConFig, 2009, pp.362–367.

[LI03] Li, J; Yao, C. “Real-Time Concepts for Embedded Systems”. CPM Books, 2003, 294p.

[MAN11] Mandelli, M. “Mapeamento Dinâmica de Aplicações para MPSoCs Homogêneos”. Dissertação de Mestrado, PUC-RS, 2011, 106p.

[MAT10] Matos, D; Concatto, C; Kologeski, A; Carro, L; Kastensmidt, F; Susin, A. “Monitor-Adapter Coupling for NoC Perfomance Tuning”. In: ICECS, 2010, pp. 193-199

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

[MOR12a] Moraes, F; Madalozzo, G; Castilhos, G.; Carara, E. “Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs”. In: ISCAS, 2012, pp. 644-647.

[MOR12b] Moraes, F; Madalozzo, G; Ruaro, M.; Carara, E. “Evaluation of Adaptive Management Techniques in NoC-Based MPSoCs”. In: ICECS, 2012.

[OZT06] Ozturk, O; Kandemir, M; Son, S; Karaköy, M. “Selective code/data migration for reducing communication energy in embedded MpSoC architectures”. In: GLSVLSI, 2006, pp.386-391.

[PAS08] Pasricha, S.; Dutt, N. "On-Chip Communication Architectures: System On Chip Interconnect". Morgan Kaufmann. 2008. 522p.

[PIT07] Pittau, M; Alimonda, A; Carta, S; Acquaviva, A. “A. Impact of Task Migration on Streaming Multimedia for Embedded Multiprocessors: A Quantitative Evaluation”. In: ESTImedia, 2007, pp. 59-64.

[PLA01] PLASMA MIPS - Definição original do processador Plasma. Capturado em: http://www.opencores.org/projects.cgi/web/mips/overview, acessado em: Janeiro 2013.

[SHA11] Shabbir, A; Kumar, A; Mesman, B; Corporaal, H. “Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms”. In: SAMOS, 2011, pp.132-139.

[STA11] Stan, A; Valachi, A; Bêrleanu, A. “The design of a run-time monitoring structure for a MPSoC”. In: ICSTCC, 2011, 4p.

[TAF11] Tafesse, B; Raina, J; Muthukumar. “Efficient Scheduling Algorithms for MPSoC Systems”. In: ITNG, 2011, pp. 683-688.

[TIL09] Tilera Corporation. “The TILE-GX™ Processor”. Capturado em: http://www.tilera.com/products/processors/TILE-Gx_Family, acessado em: Janeiro 2013.

[TOR09] Tortato J; Hexsel R. “MPSoC minimalista com caches coerentes implementado num FPGA.” In: WSCAD-SSC, 2009, pp.1-8.

[VAN07] Vangal, S.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Iyer, P.; Singh, A.; Jacob, T.; Jain, S.; Venkataraman, S.; Hoskote, Y.; Borkar, N. “An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS”. In: ISSCC, 2007, pp. 5-7.

[WOS07] Woszezenki, C. *“Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”*. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 121p.

[ZHA09] Zhaoguo, F; Chaoshan, S; Zuying L. “A Task Scheduling Algorithm of Real-Time Leakage Power and Temperature Optimization for MPSoC”. In: CAD/Graphics, 09, pp. 478-483.