Proposta de Controle Adaptativo para Atendimento a Requisitos de Aplicações em MPSoCs

Guilherme A. Madalozzo, Fernando G. Moraes

PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brasil
guilherme.madalozzo@acad.pucrs.br; fernando.moraes@pucrs.br

Resumo: O presente trabalho propõe técnicas de controle adaptativo para atendimento a requisitos de aplicações em MPSoCs. Técnicas como monitoramento de aplicações, escalonamento, gerencia distribuída e migração de tarefas apresentam-se como alternativa para MPSoCs auto-adaptativos. Este trabalho apresenta as pesquisas referentes às técnicas citadas e propõe novas alternativas para esses mecanismos. Por fim apresentam-se os resultados obtidos com uso das técnicas aqui propostas.

I. INTRODUÇÃO

Gordon Moore, em 1965, constatou que a cada 18 meses o número de transistores em circuitos integrados dobram 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. Entretanto, sistemas embarcados devem ser auto adaptativos, capazes de prover alto desempenho através da utilização de técnicas, tais como: migração de tarefas para balanceamento de carga, mantendo as tarefas comunicantes da mesma aplicação sendo executas próximas uma das outras; monitoramento de tarefas, para fazer uso consciente de energia e controle distribuído, para efetuar gerenciamento de recursos.

Para prover adaptabilidade em tempo de execução, Fattah et al. [FAT11] comentam que muitos trabalhos vem propondo estruturas de monitoramento tentando observar o estado de sistemas em tempo de execução em termos de: transações, temperatura, sobrecarga na rede, falhas, defeitos, etc.

Pensando em sistemas auto adaptativos constatamos a necessidade de coletar requisitos das aplicações através de *profiling* (perfil). Baseado nos dados desta coleta é feito um monitoramento constante para efetuar tomadas de decisão, tais como: alteração da prioridade de comunicação, migração de tarefas, escalonamento, DVFS (*Dynamic Voltage and Frequency Scaling*), DFS (*Dynamic Frequency Scaling*), aumento do *time-slice*, etc.

A. Objetivos

A presente Dissertação tem por objetivo estratégico propor e desenvolver técnicas de adaptabilidade para atendimento de requisitos de aplicações em MPSoCs.

Para atingir o objetivo estratégico estabeleceu-se um conjunto de objetivos específicos:

1. Propor e desenvolver técnica de migração de tarefas em MPSoCs.
2. Propor e desenvolver escalonamento baseado em prioridades para tarefas executando em MPSoC.
3. Propor e desenvolver monitoramento para controle de deadline de tarefas.
4. Adaptar a arquitetura de controle distribuído (desenvolvida por outros alunos do grupo) visando a inclusão das técnicas de monitoramento e migração de tarefas.

B. Estrutura do Documento

A estrutura do documento está organizada como segue. No segundo Capítulo apresenta-se uma visão do estado da arte para monitoramento de tarefas, escalonamento, controle distribuído e migração de tarefas em MPSoC. No terceiro Capítulo tem-se o detalhamento do escalonamento por prioridade, desenvolvido no terceiro semestre do curso de mestrado. No quarto Capítulo tem-se o detalhamento da migração de tarefas desenvolvida no primeiro ano do curso de mestrado. No quinto Capítulo tem-se a interação do monitoramento com a migração de tarefas, desenvolvido no terceiro semestre do curso de mestrado. No sexto Capítulo têm-se os resultados obtidos com o uso das técnicas de monitoramento com migração de tarefas. No sétimo Capítulo apresenta-se o detalhamento das atividades e cronogramas.

II. ESTADO-DA-ARTE

Neste Capítulo apresenta-se o estado da arte para os assuntos 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.

A. 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 auto adaptabilidade dos MPSoCs necessitam desse tipo de mecanismos.

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 guardados 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 1.

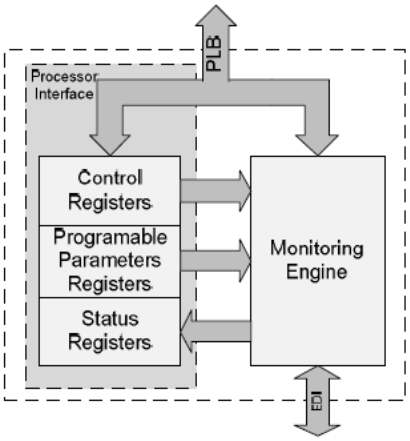

Figura 1 – Arquitetura da estrutura de monitoramento. Fonte: [STA11]

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.

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 2 os autores apresentam as três alternativas de monitoramento. Na Figura 2(a) temos uma NoC sem monitoramento. Na Figura 2(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 2(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 2(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 2(b) e na Figura 2(c) requerem um maior número de recursos de *hardware* e de conexões que a alternativa apresentada na figura Figura 2(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 moniotoramento.

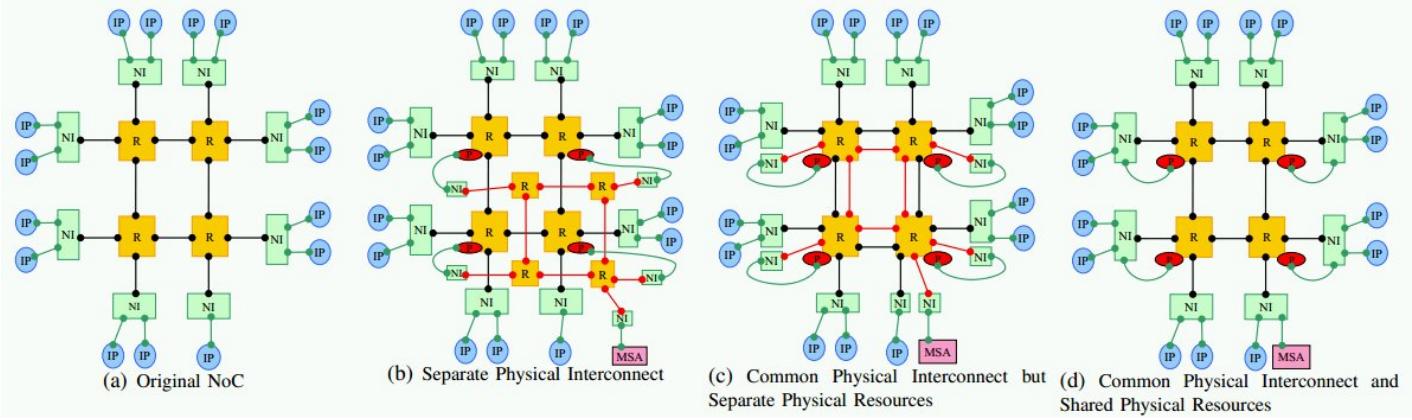

Figura 2 - Versões de monitoramento. Fonte: [CIO06]

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 3. O bloco *Input Buffer* contém a FIFO usada no canal de entrada e é responsável por armazenar os *flits*. A FIFO é controlado 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.

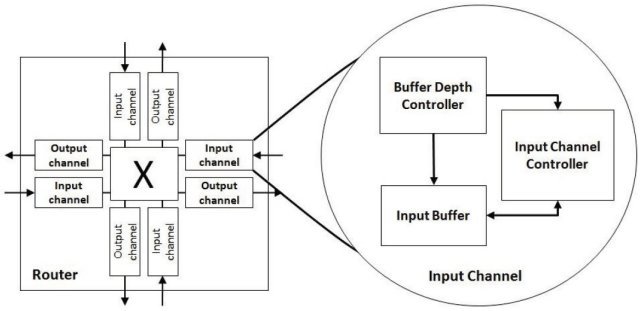

Figura 3 - Arquitetura do roteador com controle de profundidade do buffer. Fonte: [MAT10]

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 4.

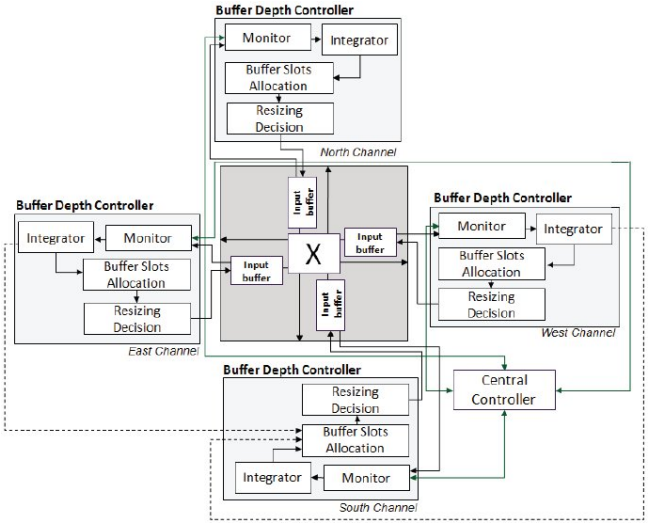

Figura 4 - Arquitetura do roteador com o fluxo de controle detalhado. Fonte: [MAT10]

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

B. Escalonamento

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

Coskun et al.

Coskun et al. [COS07] apresentam um algoritmo de escalonamento de tarefas com o objetivo de atingir uma distribuição térmica uniforme em MPSoC. 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 executados com frequência máxima, a menos que a temperatura crítica (85°C) seja atingida. Se um processador atinge o 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 algoritmos 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.

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 guiado por 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 guiado por desempenho. 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.

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 redimensionamento dinâmico 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 baixa 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.

C. Controle Distribuído

Nesta Seção apresentam-se pesquisas referentes ao gerenciamento distribuído de tarefas em MPSoCs. Essas técnicas têm como 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.

Fattah et al.

Fattah et al. [FAT11] apresentam uma pesquisa de monitoramento de sistemas auto-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 pode-se ver na Figura 5, 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.

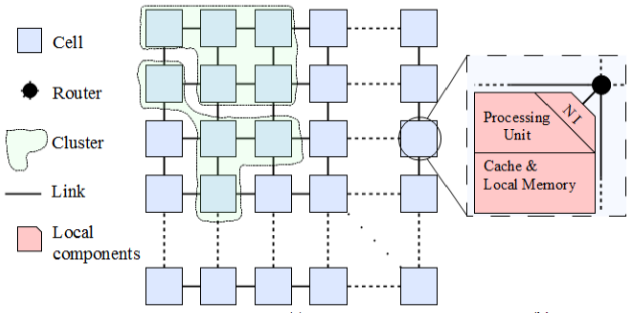

Figura 5 – NoC com agrupamento de tarefas por regiões. Fonte: [FAT11]

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: (*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 *multi-core*.

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 fixos, 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 6 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 as duas versões de gerenciamento distribuído. O gerenciamento distribuído procura minimizar o envolvimento do gerenciador central no processo e investe mais inteligência no processador local.

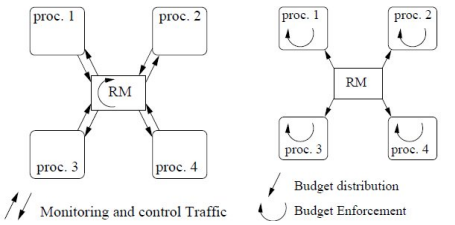

Figura 6 - Gerenciamento de recursos: centralizado x distribuído. Fonte: [SHA11]

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, esses créditos são 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.

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 alta 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 inteiro ao invés de estarem concentradas em um único ponto. Todas essas vantagens ajudam a criar um sistema multi agentes escaláveis 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.

D. 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.

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.

Pittau et al.

Pittau et al. [PIT07] apresentam uma camada *middleware* que implementa 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.

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 7 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*.

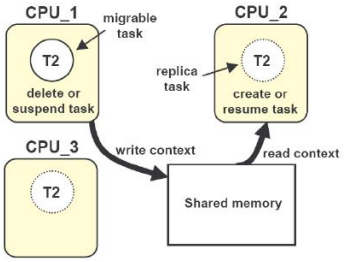

Figura 7 – Esquema geral de um sistema baseado em migração de tarefas por réplicas. Fonte: [LAY09].

A migração da tarefa só poderá ser executada quando a tarefa encontrar seu *checkpoint*. Quando se decide efetuar a migração de tarefas, primeiramente salva-se o contexto desta 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.

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 8: *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.

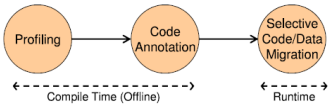

Figura 8 - Etapas para a decisão de migração de tarefas. Fonte: [OZT06]

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.

E. 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 auto-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, 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 consimida.

As pesquisas referentes a controles distribuídos de MPSoCs tem por objetivo tonar 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 PE com alta temperatura ou até mesmo com alta frequência, sem a necessidade de forçar a migração (com pontos de migração configurados pelo usuário), 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
Proposta	**Sim**	**Sim**	**Sim**	**Sim**	**Balanceamento de carga / Consumo de energia / Controle de *deadline***	**Sintéticas e reais**

III. ESCALONAMENTO

Neste Capítulo apresentam-se algoritmos clássicos de escalonamento, tais como: FIFO, EDF, SJF, BFF e BBF. Após isso, será descrito o algoritmo de escalonamento preemptivo baseado em prioridade desenvolvido inicialmente por [CAR11] e adaptado para o presente trabalho.

A. *Algoritmos clássicos de Escalonamento*

Encontramos na literatura diversos algoritmos para escalonamento de tarefas. A seguir, são apresentados algoritmos descritos por Tanenbaum et al. [TAN08]:

1. *First In First Out* (FIFO): algoritmo não preemptivo onde as tarefas são escalonadas na mesma ordem que chegam ao processador. É um algoritmo simples e de fácil implementação.
2. *Earliest Deadline First* (EDF): algoritmo preemptivo que fornece prioridade para as tarefas que estão com seus deadlines expirando.
3. *Shortest Job First* (SJF): algoritmo não preemptivo que, baseado nos dados da etapa de profiling, conhece o tempo de execução das tarefas e escalona as tarefas com o tempo de execução mais curto.
4. *Backfill First Fit* (BFF): algoritmo não preemptivo onde as tarefas são escalonadas na mesma ordem que chegam ao processador (idem ao FIFO). Porém, quando não houver recursos para a tarefa executar, o algoritmo verifica a fila de tarefas e executa a primeira tarefa que possa ser executada com os recursos disponíveis.

Muitos dos algoritmos clássicos necessitam de informações específicas sobre as tarefas a serem executadas, tais como: tempo de execução, *deadline*, tipos de tarefas (tempo real, interativas, em lote), etc.

B. *Algoritmo de escalonamento preemptivo baseado em prioridade*

Nesta Seção apresenta-se o algoritmo de escalonamento adaptado para o presente trabalho. O algoritmo toma suas decisões com base nos dados coletados pelo monitoramento de tarefas para controle de *deadline*, que pode efetuar a troca de prioridade e *time-slice* das tarefas.

Em [LI03] os autores apresentam um algoritmo preemptivo baseado em prioridade. Geralmente os *kernels* suportam até 256 níveis de prioridades, onde o valor 0 representa maior prioridade e o valor 255 representa menor prioridade.

Com o algoritmo preemptivo baseado em prioridade, cada tarefa tem sua prioridade. 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 *kernel* salva o contexto da tarefa atual em sua TCB e passa a executar a tarefa com a prioridade maior do que a da tarefa atual.

Na Figura 9(a) a *Task1* é preemptada pela *Task2* que tem maior prioridade, e logo em seguida é preemptada pela *Task3*. Assim que a *Task3* é finalizada, a *Task2* reassume a execução; acabando a *Task2*, então, a *Task1* reassume sua execução. Embora as prioridades das tarefas sejam atribuídas quando são criadas, elas podem ser alteradas dinamicamente utilizando chamadas de sistemas.

Além do algoritmo preemptivo baseado em prioridade, [LI03] apresentam o algoritmo *Round-Robin*. Esse algoritmo fornece para cada tarefa o mesmo tempo de execução. Escalonamento por *Round-Robin* não satisfaz os requisitos de sistemas em tempo real, pois esses sistemas contém tarefas que executam em diferentes períodos de tempos. Como alternativa, [LI03] apresentam um algoritmo de escalonamento utilizando o *Round-Robin* com o algoritmo baseado em prioridades, fazendo uso do *time-slice* para realizar uma igual alocação de tarefas com mesma prioridade, como podemos ver na Figura 9(b).

O escalonamento *round-robin* sem prioridades executa as tarefas de forma cíclica, sendo cada tarefa executada por um período de tempo fixo, pré-determinado, denominado *time-slice.* Para cada ciclo de relógio o contador de *time slice* é incrementado, e quando uma tarefa completa seu *time slice*, o seu contador é zerado e a próxima tarefa é escalonada.

Caso uma tarefa seja preemptada no decorrer do ciclo do *round-robin,* devido à existência de uma tarefa de maior prioridade, seu contador de *time slice* é salvo e será restaurado apenas quando a tarefa interrompida for novamente escalonada. Esse cenário pode ser analisado na Figura 9(b), onde a *Task1* é preemptada por uma tarefa de maior prioridade, *Task4*, mas é restaurada quando o *time slice* da *Task4* for completado.

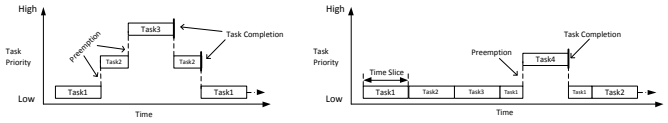

Figura 9 - (a) Escalonamento preemptivo baseado em prioridades. (b) Escalonamento preemptivo baseado em prioridade com round-robin. Fonte: [LI03]

O monitoramento de tarefas, para controle de *deadline,* afeta diretamente a decisão do algoritmo de escalonamento. Onde o monitoramento é capaz de efetuar trocas de prioridade e *time-slice*, duas informações utilizadas pelo escalonador para tomar a decisão de que tarefa deve ser escalonada.

IV. MIGRAÇÃO DE TAREFAS

O presente Capítulo apresenta o trabalho realizado pelo Autor relacionado à migração de tarefas.

A. *Arquitetura HeMPS*

No presente trabalho, é utilizado o MPSoC HeMPS. A arquitetura HeMPS é um MPSoC homogêneo que emprega processadores Plasma-IP [PLA01] interconectados pela NoC Hermes [MOR04]. Também, tem-se uma memória externa, denominada repositório de tarefas. Na Figura 10 é ilustrada uma instância do MPSoC HeMPS configurado como uma NoC de dimensão 2x3 interconectando os Plasmas-IP.

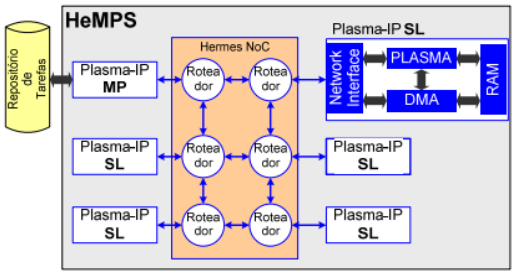

Figura 10 – Instância da HeMPS utilizando uma NoC com dimensões 2x3 para interconectar seis processadores Plasma-IP

O sistema possui um processador mestre 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 aos 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 bloqueante, e este protocolo tem a vantagem de reduzir o tráfego na rede, pois uma mensagem só é enviada se a mesma tiver sido solicitada.

B. Protocolo de Migração de Tarefas

Utilizando o MPSoC HeMPS como plataforma de referência, propôs-se um algoritmo de migração de tarefas com as seguintes características:

1. bloqueante, com migração de código, dados e contexto;
2. sem pontos de controle de migração (*checkpoints*);
3. a migração é executada conforme a necessidade definida pelo monitoramento de tarefas do sistema;
4. as mensagens a serem enviadas a outras tarefas, armazenas no microkernel na estrutura pipe, não são migradas, mas não são perdidas e sim consumidas no processador origem.

O protocolo de migração de tarefas é ilustrado na Figura 11 e é resumido a seguir:

Detecção da necessidade de migrar uma tarefa, resultando em um pedido de migração de tarefa para o PE Mestre. A detecção foi desenvolvida no presente trabalho através do monitoramento de perdas de deadline e será apresentada no Capítulo 0. Na Figura 11 corresponde ao evento 1, onde T1, tarefa a ser migrada, faz a solicitação da nova posição ao PE mestre.

1. O PE Mestre executa a heurística para computar qual será a nova posição da tarefa. Note que a tarefa que será migrada continua sua execução em paralelo.
2. O PE *Slave*, que solicita a migração da tarefa, recebe a nova posição da tarefa.
3. A tarefa somente pode ser migrada se e somente se ela está em execução (não está em estado bloqueado, esperando dados de outro processador). O escalonador do *microkernel* verifica esta condição. Se a tarefa pode ser migrada, o *microkernel*, envia para o PE alvo um pacote com todo o conteúdo da página (que contém o código e os dados) da tarefa e sua TCB (bloco de controle de tarefas).
4. A tarefa migrada é escalonada uma vez que o código e a TCB foram completamente recebidos.

Esse protocolo foi desenvolvido por outro aluno do grupo. 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_{mig}, tempo para transmitir a tarefa e reescalona-la em um novo PE. Durante t_{req} e t_p a tarefa a ser migrada continua sua execução. Durante t_{mig} a tarefa para sua execução.

Depois de a tarefa ter sido migrada, todas as tarefas que se comunicam com ela de alguma forma irão receber sua nova posição. Notar que ao final do processo de migração não há notificação para nenhum processador que a tarefa migrou para um novo processador. Para que a comunicação ocorra de forma correta, o protocolo de comunicação *read request* foi alterado para enviar 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 dados, sem a transmissão da nova posição. Assim, a tarefa receptora continua usando a posição original da tarefa que foi migrada. Quando o *pipe* não contiver mais dados da tarefa que migrou, a *message request* é encaminhada para a nova posição da tarefa (ver na Figura 11 a seta *read request forwarding*). A tarefa migrada envia a mensagem solicitada, com sua nova posição, e o PE receptor atualiza sua tabela de tarefas (ver na Figura 11 a seta *message delivery with new task position*).

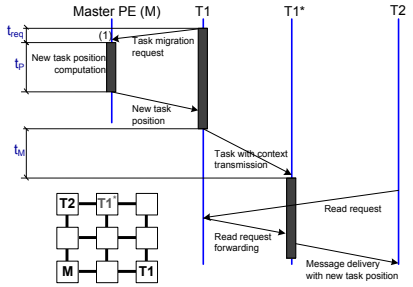

Figura 11 – Protocolo de Migração de Tarefas com atividades por PE.

C. Heurística de Migração de Tarefas

No contexto do protocolo apresentado na Figura 11, o Autor deste trabalho desenvolveu a heurística de migração de tarefas, correspondendo à etapa 2 do protocolo.

O processo de migração de tarefas foi projetado 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 12 apresenta o pseudo-código da heurística de migração de tarefas proposto. Este processo está dividido nas seguintes etapas:

1. O *microkernel* do processador que está monitorando o sistema detecta que uma determinada tarefa (executando nele mesmo) 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 destino.
2. O PE mestre ao receber a solicitação inicia a execução da heurística apresentada na Figura 12, responsável por definir o PE que 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 dos processadores que podem receber t_M (linha 4). Caso exista apenas um elemento no conjunto N, n_0, o conjunto P conterá n_0 e os PEs distantes 1 *hop* de n_0. 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 n_0 (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, então, o 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 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).

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

Onde:

- Esbit = consumo de energia de cada roteador (esse valor é igual para todos os roteadores);

- nhops = distância entre os roteadores comunicantes.

6. Na sequência da heurística inicia-se um laço que busca 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.

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.

Input: t_M: task to be migrated; p_M: PE executing t_M; *N*:set containing the tasks t_M communicates with
Output: p_T: PE to receive the task t_M

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

Figura 12 – Pseudo-código da migração de tarefas proposto

V. INTERAÇÃO DO MONITORAMENTO COM A MIGRAÇÃO

Neste Capítulo apresenta-se a interação do monitoramento de tarefas com a migração de tarefas. Como no Capítulo 0 apresentou-se a migração de tarefas, este Capítulo descreve o monitoramento de tarefas proposto.

A. Monitoramento de Tarefas

O monitoramento de tarefas foi projetado buscando solucionar dois problemas: *(i)* perda de *deadline* das aplicações; *(ii)* ponto ideal para a migração de tarefas. A solução desses problemas é essencial para que um MPSoC tenha características de um sistema auto adaptativo.

O mecanismo de monitoramento é configurado baseado nos dados coletados na etapa de *profiling*. A etapa de *profiling* é responsável por verificar qual deve ser o *deadline* 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* ideal da aplicação. Então, para configurar o *deadline* da aplicação foi criada uma nova chamada de sistema, *setdeadline*. Essa chamada de sistema é gerada com a execução de uma nova função adicionada a API de comunicação, *setDeadline(int ticks)*. O *deadline* é informado apenas na tarefa final da aplicação, pois esta corresponde para o usuário ao ponto onde é feita a medida. Na etapa de *profiling*, também, deve ser informado qual é o valor aceitável de perdas de *deadline*, para isso foi adicionado na estrutura da TCB o campo *max_miss*.

O processador que estiver executando a tarefa configurada será 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 *deadline* atingir o valor configurado no *max_miss*, então, o *microkernel* do PE escravo envia uma solicitação ao PE mestre para que esse altere a prioridade e o *time-slice* de todas as tarefas da aplicação. A política de solicitação de troca de prioridade e *time-slice* é a mesma tanto para solicitar que as tarefas tenham prioridades altas e um longo *time-slice* quanto para solicitar que a prioridade e o *time-slice* sejam diminuídos.

Na Figura 13 apresenta-se um cenário com uma aplicação, com seis tarefas (A-F), executando no MPSoC juntamente com uma aplicação para *disturbing*. O processador que está executando a tarefa F é o processador responsável pelo monitoramento da aplicação. Na Figura 13(a) temos a aplicação sendo executada com o monitoramento em execução. Na Figura 13(b) notamos que o PE monitor encontrou a quantidade configurada (*max_miss*) de perdas de *deadline* e enviou uma solicitação ao mestre para efetuar a troca de prioridade e *time-slice*. Na Figura 13(c) o PE mestre recebe a solicitação e envia um pacote, para todos os processadores que contém as tarefas da aplicação, solicitando que a prioridade e o *time-slice* das tarefas sejam alterados.

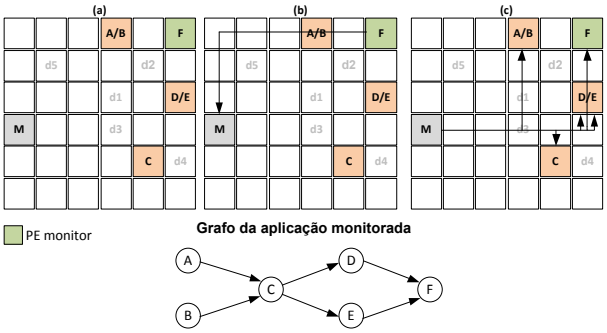

Figura 13 - Mecanismo de monitoramento com solicitação de troca de prioridade e *time-slice.*

A próxima Seção descreve em que ponto o monitoramento de tarefas é utilizado para a de migração de tarefas.

B. Monitoramento de Tarefas com Migração de Tarefas

Como vimos na seção anterior, o monitoramento de tarefas é o mecanismo responsável pela troca de prioridades e *time-slice* das tarefas de uma determinada aplicação. Em alguns casos, apenas efetuar a troca da prioridade e do *time-slice* já se faz necessário para que a aplicação não perca *deadlines*. Mas, temos casos em que efetuar essa troca não basta e a aplicação continua perdendo *deadlines.* É nesse caso que o monitoramento de tarefas age juntamente com a migração de tarefas.

Da mesma maneira que o campo *max_miss* foi adicionado na estrutura da TCB, também foi adicionado o campo *max_miss_migra*. Então, na etapa de *profiling* deve ser configurado o valor aceitável de perdas de *deadline* para efetuar a migração de tarefas.

Como descrito na seção anterior, assim que o PE monitor encontrar a quantidade de perdas de *deadline* configurada no *max_miss*, esse PE envia uma solicitação para o PE mestre alterar as prioridades e *time-slice* das tarefas. Caso a tarefa monitorada continuar com perdas de *deadline* e essa perda atingir o valor configurado no *max_miss_migra*, mesmo depois de alteradas prioridades e *time_slices*, então, o PE monitor solicitará ao PE mestre a migração de alguma tarefa da aplicação. A escolha de qual tarefa deve ser migrada é configurada na etapa de *profiling*. Quando a tarefa estiver sozinha executando no MPSoC é verificada qual tarefa tem o maior tempo médio de comunicação, sendo essa a tarefa a ser migrada.

Na Figura 14 apresenta-se a interação do monitoramento de tarefas com a migração de tarefas. Os cenários apresentados nas Figura 14(a)(b)(c) são os mesmos que os apresentados na Seção anterior; mecanismo de monitoramento da aplicação com o PE monitor solicitando a troca de prioridade e do *time-slice* das tarefas ao PE mestre. Despois que o mestre envia um pacote solicitando para que todas as tarefas da aplicação sejam alteradas, o PE monitor continua monitorando a aplicação para verificar se continua ocorrendo perdas de *deadline*. Na Figura 14(e) o PE monitor encontra a quantidade de perdas de *deadline* configurada no *max_miss_migra* e, com isso, solicita ao PE mestre a migração de uma determinada tarefa da aplicação. Como explicado anteriormente, o PE mestre decide qual tarefa deve ser migrada na etapa de *profiling*, nesse caso a tarefa a ser migrada é a C. Sabendo qual tarefa migrar (t_m) o PE mestre, na Figura 14(e), define qual será a nova posição de t_m e envia uma solicitação de migração para o PE que está executando a tarefa C. Por fim, pede-se notar que na Figura 14(f) ocorre a migração da tarefa.

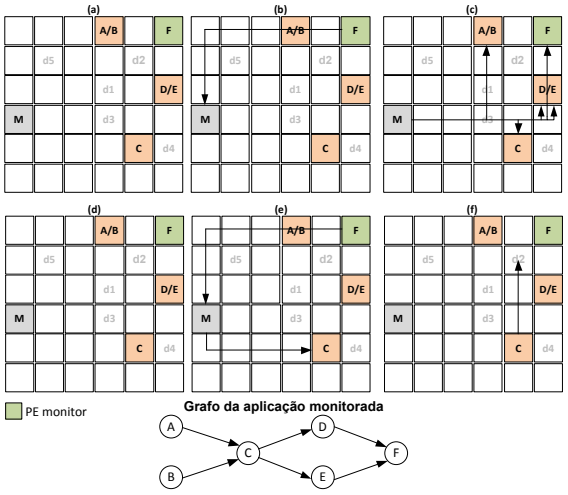

Figura 14 - Mecanismo de monitoramento com migração de tarefas.

VI. RESULTADOS EXPERIMENTAIS

Para a avaliação do trabalho realizado no decorrer dos três semestres de mestrado, efetuou-se resultados experimentais a partir de uma NoC 6x6 instanciada no MPSoC HeMPS [CAR09].

A seguir apresentam-se algumas características relevantes para o monitoramento com migração de tarefas:

1. Tamanho da página: 16 Kbytes (4.096, código e dados).
2. *Tim- slice* da tarefa: 10.000 ciclos de *clock*, podendo ser alterado conforme decisão do monitoramento.
3. Prioridade da tarefa: baixa, podendo ser alterada conforme decisão do monitoramento.
4. Deadline da tarefa monitorada: 5 Kciclos de *clock*.
5. Máxima perda de *deadline* aceitável: 3. 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 migração da tarefa: 7. Esse é o *max_miss_migra* da TCB, é o ponto em que o mecanismo de migração de tarefas deve ser executado.
7. Escalonador: *round-robin* preemptivo com prioridades.
8. Roteador da NoC: roteamento XY, arbitragem centralizada *round-robin*, *buffer* de entrada (profundidade do *buffer* igual a 16 *flits*).

A. Aplicação Sintética

Na Figura 15(a) apresenta-se o mapeamento de uma determinada aplicação, composta por 6 tarefas (A, B, C, D, E, F). 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 (Figura 15(b)), 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, que nesse caso é o PE que está executando a tarefa F, solicita ao PE mestre (M) a migração de alguma tarefa da aplicação. Baseado nos dados coletados na etapa de *profiling,* o PE mestre decide qual tarefa será migrada. Sabendo qual tarefa

deve migrar, o PE mestre escolhe uma nova posição e a tarefa é migrada para a posição mais próxima entre as tarefas comunicantes (Figura 15(c)), restaurando e/ou melhorando o desempenho da aplicação.

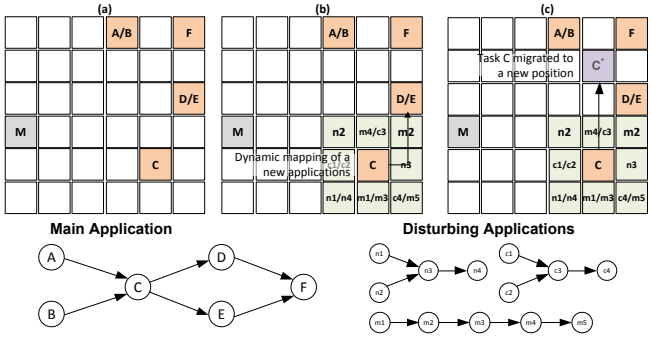

Figura 15 – Aplicação experimental para migração de tarefas

Na Seção seguinte apresenta-se a avaliação dessa aplicação experimental, apresentando e analisando gráficos do tempos de iteração da aplicação para os cenários de teste.

B. Avaliação com a Aplicação Sintética

O desempenho do protocolo de monitoramento com migração de tarefa foi avaliado de acordo com os quatro cenários descritos a seguir, baseado no mapeamento das tarefas apresentados na Figura 15:

1. Aplicação principal executando sozinha no MPSoC;
2. Aplicação principal executando com outras três aplicações causando perturbação na comunicação entre as tarefas da aplicação principal;
3. Mesmo cenário que o 'II' mas com o monitoramento de tarefas alterando as prioridades de escalonamento e time-slice das tarefas da aplicação principal;
4. Mesmo cenário que o 'III', mas com o monitoramento de tarefas verificando a necessidade de efetuar a migração de alguma tarefa da aplicação principal, tarefa C.

A aplicação principal executa 150 interações para todos os cenários. A Figura 16, Figura 17, Figura 18 e Figura 19 apresentam os gráficos com o tempo das iterações da Tarefa C da aplicação principal executando, respectivamente, nos cenários 'I', 'II', 'III', 'IV'. As tarefas iniciais da aplicação ('A' e 'B') são mapeadas primeiro, 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 16 (cenário 'I') mostra que, após o mapeamento das tarefas consumidoras, o tempo das iterações passa a ser praticamente constante, oscilando entre 3700 e 4500 ciclos de relógio. Este é considerado o cenário de referência.

A Figura 17 (cenário 'II') 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. O tempo se torna constante após 1.000 Kciclos de *clock*, 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 18 (cenário 'III') apresenta o tempo de iteração com o monitoramento da aplicação principal. Podemos notar que temos 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.

Na Figura 19 (cenário 'IV') temos a execução da técnica de migração de tarefas. Nesse caso, a tarefa F (tarefa que está sendo monitorada) continua com perdas de *deadline*. Podemos notar que com aproximadamente 500 Kciclos de *clock* a tarefa C é migrada, havendo um alto 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, ficando próxima ao cenário de referência Figura 16.

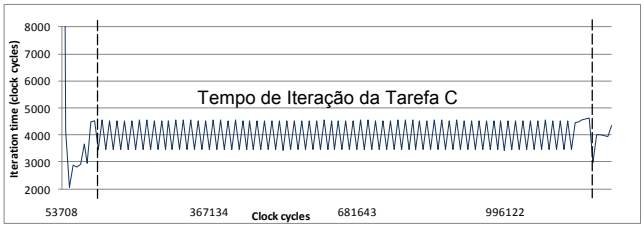

Figura 16 - Gráfico de tempo de iteração da tarefa C da aplicação principal sem perturbação.

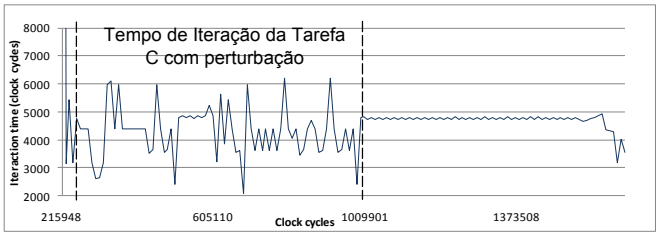

Figura 17 - Gráfico de tempo de iteração da tarefa C da aplicação principal com perturbação.

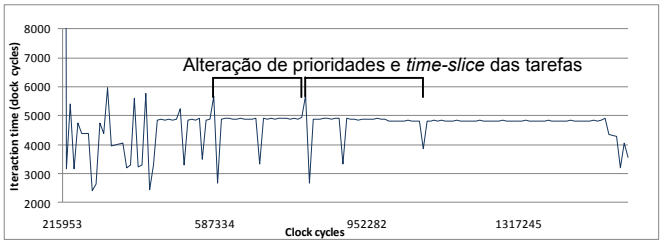

Figura 18 - Gráfico de tempo de iteração da tarefa C principal com perturbação e monitoramento com alteração de prioridade *time-slice*.

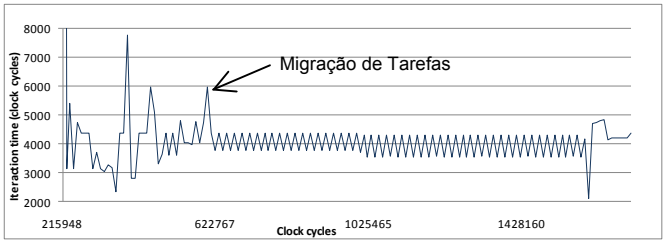

Figura 19 - Gráfico de tempo de iteração da tarefa C da aplicação principal com perturbação, monitoramento com alteração de prioridade e *time-slice* e com migração de tarefas.

VII. CRONOGRAMA DE ATIVIDADES

O protocolo de Migração de Tarefas desenvolvido tem como limitação mais restritiva o controle centralizado, o qual pode

comprometer o desempenho de todo o sistema, devido à sobrecarga de processamento no processador mestre. Devido a esta limitação, o *foco* da presente Dissertação é a definição de uma estratégia distribuída de migração de tarefas baseado nos dados de monitoramento, adicionando mestres locais por regiões. Os mestres locais, na migração de tarefas, passarão a atuar da mesma maneira que o mestre global atua na migração existente. Esses mestres locais decidirão qual o PE destino de uma determinada tarefa a ser migrada. Passarão a controlar sua sub-rede, sendo capazes de decidir o local da migração de uma tarefa com perda de *deadline*.

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

As atividades 1-3 foram realizadas, assim como a atividade 7. Resumidamente 3 grandes macro-atividades foram desenvolvidas no primeiro semestre de 2012: inclusão do escalonamento com prioridades e *time-slice* variável, monitoramento, e integração da migração com estas técnicas. Destaca-se o artigo apresentado no ISCAS 2012 [MOR12a] e submissão ao ICECS [MOR12b].

As atividades relacionadas à gerência distribuída foram postergadas para o segundo semestre de 2012, pois a infra-estrutura necessária (*microkernels* locais) está em desenvolvimento por outros alunos do grupo.

Assim, o cronograma de atividades para o segundo semestre á apresentado na Tabela 2.

Tabela 2 – Cronograma de Atividades atualizado, para o segundo semestre de 2012.

Ativ	Objetivo	Ago		Set		Out		Nov		Dez	
1	Revisão do Estado da Arte										
2	Estudar a atual implementação de gerência distribuída para a arquitetura HEMPS e integrar com a migração de tarefas										
3	Desenvolver protocolo para monitoramento baseado em regiões										
4	Controle de comunicação entre regiões – detectar quando tarefas de uma determinada aplicação se encontram em clusters vizinhos e solicitar a migração da mesma para o cluster da aplicação, se possível										
5	Avaliar outras técnicas de escalonamento, além da atual baseado em *round-robin* com prioridades										
6	Integração e avaliação de todas as técnicas implementadas										
7	Escrita da dissertação										

Tabela 3 – Cronograma de Atividades apresentado no PEP. Atividades em destaque são consideradas críticas no desenvolvimento do trabalho.

Ativ	Objetivo	Jan		Fev		Mar		Abr		Mai		Jun		Jul		Ago		Set		Out		Nov		Dez	
1	Revisão do Estado da Arte																								
2	Desenvolver protocolo para controle de *deadline* de tarefas																								
3	Adaptar algoritmo de escalonamento com prioridade																								
4	Alterar arquitetura da HeMPS, adicionando mestres locais							**postergado**																	
5	Alterar heurística de mapeamento																								
6	Controle de comunicação entre regiões																								
7	Alterar a solicitação de migração																								
8	Desenvolver protocolo para monitoramento da região																								
9	Modificar o algoritmo de migração de tarefas																								
10	Seminário de andamento																								
11	Submissão de artigos																								
12	Executar e avaliar a migração de tarefas																								
13	Escrita da dissertação																								

VIII. REFERÊNCIAS BIBLIOGRÁFICAS

[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.

[BER06] Bertozzi, S; Acquaviva, A; Bertozzi, D; Poggiali, A. "Supporting task migration in multi-processor systems-on-chip: a feasibility study". In: DATE, 2006, pp. 15-20.

[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, 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: Date07, 2007, 6p.

[FAD11] Fadhel, A; Smiri, K; Ammari, A; Jemai, A. “Automatic Generation of Software-Hardware Migration in MPSoC Systems”. In: CCCA, 11, pp. 1-7.

[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; 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.

[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. Outubro 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”. Submitted To: 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.

[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. Disponível em: http://www.opencores.org/projects.cgi/web/mips/overview.

[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.

[TAN08] Tanenbaum, A; Woodhull, A. “Sistemas Operacionais, Projeto e Implementação”. Tradução João Tortello, 3 ed. Porto Alegre: Bookman, 2008, 992p.

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

[WAC10] Wachter, E. “Integração de novos processadores em arquitetura MPSoC: Um estudo de caso”. Dissertação de Mestrado – PUCRS, 2010, 86p.

[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.