PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MECANISMO DE CONTROLE DE QOS ATRAVÉS DE DFS EM MPSOCS

GUILHERME MONTEZ GUINDANI

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação.

ORIENTADOR: PROF. DR. FERNANDO GEHM MORAES

Porto Alegre
2014

Dados Internacionais de Catalogação na Publicação (CIP)

G964m Guindani, Guilherme Montez

Mecanismo de controle dos QoS através de DFS em MPSoCs / Guilherme Montez Guindani. - Porto Alegre, 2014.
175 p.

Tese (Doutorado) – Fac. de Informática, PUCRS.
Orientador: Prof. Fernando Gehm Moraes.

1. Informática. 2. Arquitetura de Computador.
3. Multiprocessadores. 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 TESE DE DOUTORADO

Tese intitulada "Mecanismo de Controle de QoS Através de DFS em MPSoCs", apresentada por Guilherme Montez Guindani, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, aprovada em 14/07/2014 pela Comissão Examinadora:

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

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

Prof. Dr. Everton Alceu Carara — UFSM

Prof. Dr. Eduardo Augusto Bezerra — UFSC

Homologada em...../...../....., conforme Ata No. ..... pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Agradecimentos

Gostaria de agradecer a diversas pessoas que contribuíram para a realização desta Tese de Doutorado, e não poderia deixar de manifestar meu profundo agradecimento a todas elas.

Agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo financiamento do início deste trabalho, e à Hewlett-Packard pelo financiamento do restante deste trabalho. Agradeço à Marinha do Brasil, em especial aos meus comandantes CMG (Ref-EN) Luiz Antônio Abdalla de Moura e CMG (Ref-EN) Leopoldo Jorge de Souza, por me conceder as licenças necessárias à conclusão deste trabalho. Aos funcionários, alunos e professores do PPGCC pela convivência, amizade e experiências compartilhadas ao longo do curso de doutorado. Aos colegas e amigos do grupo de apoio ao projeto de hardware (GAPH) pelo ótimo ambiente de trabalho e ótima convivência, tanto os que já não se encontram no grupo quanto os que ainda estão trabalhando comigo. Agradeço aos amigos, Eduardo Wächter, Marcelo Mandelli, Guilherme Castilhos e Matheus Trevisan pelos conselhos, discussões/sugestões técnicas e parceria nas publicações.

Aos Prof. Dr. Eduardo Augusto Bezerra, professor da Universidade Federal de Santa Catarina e Prof. Dr. Everton Alceu Carara, professor da Universidade Federal de Santa Maria, que gentilmente aceitaram participar e colaborar com este trabalho fazendo parte da banca de avaliação. Ao Prof. Dr. Ney Laert Vilar Calazans por seu apoio contínuo aos meus trabalhos, com comentários, críticas e sugestões que me ajudaram, desde meus primeiros artigos publicados como bolsista até esta Tese de Doutorado. Agradeço especialmente ao meu orientador e amigo, Prof. Dr. Fernando Gehm Moraes pelo constante incentivo, pela dedicação e paciência, por estar sempre disponível a esclarecer dúvidas e frustrações geradas durante meu trabalho, e por acreditar em mim quando passei no concurso da Marinha do Brasil e decidi continuar o meu doutorado à distância. Agradeço principalmente, por depositar sua confiança em mim e no meu trabalho, e espero que não o tenha decepcionado.

Agradeço, especialmente, à minha esposa Larissa Richa Fournier Guindani, por me ajudar a continuar o meus estudos, me incentivando a cada dia que se passou. Por abdicar de vários sábados e domingos para que eu pudesse fazer testes, simulações e escrita deste texto. E por aguentar uma pequena solidão nas vezes em que eu precisei ir a Porto Alegre. Amo-te do fundo da minha alma, e uma grande parte deste trabalho se deve a ti.

Agradeço a toda a minha família que sempre me acreditou em mim, e sempre esteve ao meu lado. Ao meu irmão Gustavo Montez Guindani, que tudo o que faço tento passar-lhe como exemplo. Aos meus pais Sergio Dantas Guindani e Angelina de Melo Montez Guindani pela confiança, amor incondicional e apoio total em tudo o que proponho fazer. Amo todos vocês.

Sobre tudo, agradeço a Deus por sempre estar ao meu lado nos melhores e piores momentos da minha vida.

Muito Obrigado.

MECANISMO DE CONTROLE DE QoS ATRAVÉS DE DFS EM MPSoCS

RESUMO

O controle dos requisitos de qualidade de serviço (QoS) em MPSoCs baseados em NoC, com dezenas de tarefas sendo executadas simultaneamente ainda é um desafio na área de projeto de circuitos integrados. Técnicas de adaptabilidade que adotam diferentes métricas de QoS são utilizadas tanto em tempo de projeto quanto em tempo de execução. Os projetistas incluem em seus sistemas mecanismos de monitoramento que guiam os controladores embarcados na adaptação dos recursos do MPSoC para atender os requisitos de QoS impostos a aplicações. Em outras palavras, MPSoCs são capazes de se auto-adaptarem, enquanto executam um dado conjunto de aplicações.

A capacidade de auto-adaptação é uma característica fundamental para atender requisitos de QoS nos sistemas que apresentam carga dinâmica de trabalho. O ajuste dinâmico de tensão e frequência (DVFS) é a técnica de adaptação mais utilizada para reduzir o consumo de energia geral de um MPSoC, porém esta técnica não leva em consideração outros requisitos de QoS, como vazão ou latência. Outro exemplo de técnica de adaptação frequentemente utilizada é a migração de tarefas, cujo foco é o balanceamento de carga de uma aplicação.

O mecanismo de controle de QoS em MPSoCs proposto no escopo desta Tese de Doutorado utiliza a técnica de adaptação de gerência dinâmica de frequência (DFS) para controlar os requisitos de QoS e aplicações de um MPSoC, mantendo um baixo perfil de consumo de energia. Cada processador possui um sistema de monitoramento, um sistema de avaliação de QoS e um módulo de adaptabilidade, que são utilizados para controlar os parâmetros de QoS das aplicações. Em um momento inicial, cada processador que executa uma tarefa de uma dada aplicação utiliza uma política de DFS, onde a comunicação com seus vizinhos é otimizada. Após atingir um estado de estabilidade de frequência, o desempenho da aplicação é monitorado e controlado, ajustando-se a frequência dos processadores da aplicação de acordo com os requisitos de QoS impostos em tempo de projeto.

O mecanismo proposto de controle de QoS em MPSoCs foi avaliado utilizando duas aplicações sintéticas e uma real, executadas sobre a plataforma HeMPS, e com a vazão e latência como requisitos de QoS controlados. Os resultados mostram que o mecanismo proposto de controle de QoS em MPSoCs consegue atender aos requisitos de QoS impostos a uma aplicação, através da utilização da técnica de DFS e manter um baixo consumo de energia.

Palavras chave: MPSoC, NoC, QoS, Consumo de Energia, DFS.

QoS CONTROL MECHANISMS IN MPSoCs USING DFS

ABSTRACT

The quality of service (QoS) management in NoC-based MPSoCs, with dozens of applications executing simultaneously, is an open research challenge in the integrated circuit design area. Adaptability techniques, which use different QoS metrics, have been used at design time to guarantee the QoS of the applications. Designers include in their systems monitoring schema that guides embedded controllers in managing the resources of the MPSoC to satisfy the QoS requirements imposed to the applications. In order words, MPSoCs are able to self-adapt while running a set of applications.

The self-adaptation capability is a fundamental characteristic to satisfy the QoS requirements on the systems with dynamic workload. The dynamic voltage and frequency scaling (DVFS) is the most used adaptation method for reducing the overall energy consumption of an MPSoC. However, this method does not take into account other QoS requirements such as throughput or latency. Another example of adaptation technique is task migration, whose main goal is to balance the workload of the MPSoC.

The QoS control mechanism proposed in the scope of this Thesis uses the dynamic frequency scaling (DFS) technique to control the QoS parameters of the application, keeping energy consumption low profile. Each processor has a monitoring system, a QoS evaluation system and an adaptation module, which are used to control the QoS parameters to satisfy the QoS requirements imposed to the applications. At the system startup, each processor uses a DFS policy that tries to optimize the communication with its neighbor's processors. The processors use this policy up to the moment when they reach a steady frequency state. After reaching the steady frequency state the QoS monitoring starts, evaluating if they the requirements imposed at design time are respected.

The proposed QoS control mechanism was evaluated using two synthetic and one real application, using the HeMPS MPSoC, with the throughput and latency parameters as the QoS parameters to be controlled. The presented results show that the proposed QoS control mechanism can satisfy the imposed QoS requirements using the DFS technique while maintaining low energy consumption on the HeMPS MPSoC.

Keywords: MPSoC, NoC, QoS, Energy Consumption, DFS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
BE	Best Effort
CF	Custom Feedback
CRC	Cyclic Redundancy Check
DFS	Dynamic Frequency Scaling
DMA	Direct Memory Access
DPM	Dynamic Power Management
DTS	Digital Thermal Sensor
DVFS	Dynamic Voltage and Frequency Scaling
FC	Fully-Centralized
FD	Fully-Decentralized
FRT	Firm Real-Time
GALS	Globally Asynchronous Locally Synchronous
GPU	Graphics Processing Unit
HEMPS	Hermes Multiprocessor System
IP	Intellectual Property
ISS	Instruction Set Simulator
ITRS	International Technology Roadmap for Semiconductors
JTAG	Joint Test Action Group
KPN	Kahn Process Networks
LTE	Long Term Evolution
MPEG	Moving Picture Experts Group
MPEG2	Motion Picture Experts Group 2
MPSoC	Multi-Processor System-on-a-Chip
NI	Network Interface
NoC	Network-on-a-Chip
NRT	Non Real-Time
NTBI	Negative Bias Temperature Instability
PE	Processing Element
PM	Power Manager
PMC	Power Management Controller
PTT	Path To Target
PSO	Power Shut-Off

PU	Power-Up
QoS	Quality of Service
RCT	Router Congestion Table
RTL	Register Transfer Level
SC	Supply level Changes
SoC	System-on-a-Chip
TCB	Task Control Block
TCT	Target Congestion Table
VFI	Voltage and Frequency Island
VHDL	VHSIC Hardware Description Language
VLSI	Very Large-Scale Integration
VOPD	Video Object Plane Decoder

SUMÁRIO

1 INTRODUÇÃO

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

Muitas aplicações exigem SoCs com vários processadores para poder suprir seus requisitos de desempenho computacional. Um SoC que contém diversos elementos de processamento (PEs, do inglês *Processing Element*) é denominado de MPSoC (do inglês, *Multiprocessor System-on-a-Chip*). Um MPSoC consiste de uma arquitetura composta por recursos heterogêneos, que podem incluir múltiplos processadores, módulos de hardware dedicados, memórias e um meio de interconexão [WOL04].

MPSoCs com dezenas de PEs interconectados por uma NoC (do inglês, *Network-on-Chip*), são a realidade no projeto dos atuais sistemas embarcados [HOW10]. De acordo com a ITRS (*International Technology Roadmap for Semiconductors*), os MPSoCs poderão agregar mais de mil PEs até o ano de 2025.

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

1.1 A Dissipação de Potência em Dispositivos Móveis

Sistemas de alto desempenho como os MPSoCs usualmente necessitam de um elevado consumo de energia para executar suas aplicações. Em alguns dispositivos móveis, tais como *smartphones* e *tablets*, que precisam de um alto poder de processamento para entregar ao usuário múltiplos serviços e funcionalidades simultâneas, o consumo de energia é limitado pela autonomia da bateria do dispositivo. Além disto, problemas como o aumento da corrente de fuga e da densidade de potência, agravam-se a cada avanço no processo tecnológico dos circuitos integrados. Concomitantemente, o avanço na tecnologia de fabricação de baterias não é compatível com o aumento crescente da dissipação de potência em circuitos mais complexos. Desta forma, projetistas precisam selecionar o melhor desempenho considerando o menor consumo de energia possível para o conjunto de aplicações que será executado no sistema em desenvolvimento [BEN00].

A maneira mais comum para atingir a melhor relação entre desempenho e consumo

de energia é projetar o sistema para respeitar os requisitos mínimos de desempenho para um dado conjunto de aplicações, e a seguir utilizar alguma técnica de redução do consumo de energia (ou dissipação de potência) para fazer com que o circuito respeite os limites de consumo de energia. O consumo de energia em circuitos CMOS está relacionado à frequência de operação, à tensão de alimentação, à capacitância de carga e à atividade de chaveamento do circuito. A maioria das técnicas de redução do consumo de energia controlam a frequência de operação e/ou a tensão de alimentação, e desta forma conseguem reduzir significativamente o consumo de energia de um circuito CMOS.

Para garantir a satisfação do usuário, dispositivos computacionais precisam disponibilizar o máximo de desempenho quando solicitados. Entretanto, este nível de desempenho só é necessário em alguns intervalos de tempo. Desta forma, em um dado ponto de execução do dispositivo, o sistema pode ajustar suas condições de operação a fim de consumir menos energia e aumentar a vida útil de sua bateria. A habilidade de ajustar o desempenho do sistema de acordo com o desempenho solicitado é fundamental para o desenvolvimento dos sistemas com consumo de energia eficiente [BEN00].

O controle das condições de operação de um dispositivo a fim de reduzir seu consumo de energia é denominado de *gerência de dissipação de potência*. Este pode ter suas decisões guiadas por características de carga de trabalho extraídas em tempos de projeto ou de execução. Para se desenvolver uma gerência de dissipação de potência eficaz utilizando informações de tempo de projeto é necessário explorar de forma abrangente o espaço de projeto, o que exige uma quantidade de tempo significativo para circuitos mais complexos. Além disto, a variação no processo de fabricação e o envelhecimento do circuito podem levar a funcionamento defeituoso das técnicas que utilizam esta metodologia. Em contrapartida, uma gerência dinâmica da dissipação de potência, que monitora a carga de trabalho em tempo de execução, pode apresentar melhores resultados quando aplicada em sistemas com alta variação na carga de trabalho, e está inserida na maioria dos sistemas modernos.

1.2 A Adaptabilidade na Gestão dos Recursos e do Desempenho dos MPSoCs

MPSoCs podem receber uma carga dinâmica de trabalho [CAR07a], ou seja, aplicações podem ser carregadas em tempo de execução. Por exemplo, um dado MPSoC pode estar executando o tratamento de um fluxo de dados para processamento de telefonia 4G (LTE) [JAL10], e durante o processamento desta aplicação o usuário inicia o processamento de vídeo em alta resolução.

Futuramente, MPSoCs com centenas de PEs executarão múltiplas aplicações (com inúmeras tarefas) admitindo carga dinâmica de trabalho. As tarefas que executarão nestes MPSoCs, estarão sujeitas a disputa por recursos como o meio de comunicação e um PE para sua execução. No caso do MPSoC admitir em seu PE a execução multitarefa, as tarefas poderão disputar, também, o tempo de execução em um dado PE. Tais

disputas por recursos degradam o desempenho das tarefas que compõem uma dada aplicação, e desta forma o desempenho da mesma.

Assim, MPSoCs, com estas características devem integrar em seu projeto mecanismos que realizem auto-adaptação em tempo de execução para atender as demandas das aplicações, através da gestão eficiente de seus recursos. A adaptabilidade é inserida no MPSoC através do uso de diferentes técnicas, como escalonamento de tarefas baseada em prioridades, prioridades nas comunicações, migração de tarefas, DVFS (Escalonamento Dinâmico de Tensão e Frequência, do inglês *Dynamic Voltage and Frequency Scaling*) e a adoção de chaveamento por circuito. Tal adaptabilidade é utilizada para lidar com a degradação de desempenho que ocorre em possíveis cenários de execução de aplicações, como o descrito anteriormente.

As técnicas empregadas no conceito de adaptabilidade podem ser utilizadas em conjunto, porém cada uma delas possui seu foco em uma determinada métrica de qualidade de serviço (QoS – *Quality of Service*). Como exemplos podemos citar a prioridade nas comunicações para atender a métrica de vazão, ou a utilização de DVFS para reduzir o consumo de energia do MPSoC.

A técnica de DVFS é a mais utilizada para reduzir o consumo de energia do MPSoC se baseia no fato de que mudanças no domínio de frequência possuem um impacto linear no consumo de energia e no domínio de tensão produzem um impacto quadrático. Desta forma, ao se controlar estas duas variáveis pode-se adaptar o consumo de energia de um circuito CMOS. Entretanto, a diminuição de tamanho das novas tecnologias atrelado ao aumento da variabilidade no processo de fabricação dos circuitos CMOS, pode interferir no funcionamento da técnica de DVFS. Tal interferência pode fazer com que um esquema de DVFS nominalmente correto falhe em atingir suas metas de frequência ou consumo de energia [HER09] [GAR09].

A técnica de DFS (Escalonamento Dinâmico de Frequência, do inglês *Dynamic Frequency Scaling*), é uma alternativa ao uso do DVFS. No DFS, a tensão do circuito CMOS é mantida constante, enquanto a frequência dos PEs é alterada em função de sua carga de trabalho. Em [ROS12a], a técnica de DFS foi utilizada a partir do sinal de relógio do sistema. O emprego do DFS ocasionou um pequeno aumento na execução total das aplicações executadas no MPSoC, porém produziu uma redução no consumo de energia.

1.3 Objetivos da Tese

Os objetivos estratégicos da presente Tese de Doutorado incluem:

- Explorar a utilização da técnica de DFS, com controle realizado através do sistema operacional, em um método distribuído de gerência de recursos;
- Desenvolver um modelo de estimativa do consumo de energia de MPSoCs baseados em NoCs;

- Explorar qualidade de serviço (QoS) no nível de aplicação e de tarefas;
- Utilizar a técnica de DFS para controlar o atendimento de QoS em MPSoCs;
- Dominar o projeto de MPSoCs baseados em NoC desde o nível de rede até o nível de aplicação.

Tais objetivos estratégicos devem ser alcançados a partir dos seguintes objetivos específicos:

- Monitoramento
  - Desenvolver monitores de estabilidade do DFS, para avaliar quando a gerência de DFS estabiliza os níveis de frequência dos processadores, para a partir deste momento aplicar técnicas de QoS;
  - Desenvolver monitores de parâmetros de QoS, a fim de avaliar violações no atendimento de um dado requisito de QoS;
  - Explorar diversas técnicas de monitoramento, incluindo monitoramento no nível da aplicação ou de suas tarefas;
  - Explorar duas métricas de QoS, vazão e latência.
- DFS
  - Modificar a técnica de DFS proposta em [ROS12b] para ser controlada pelo sistema operacional;
  - Modificar esta técnica de DFS para que existam dois algoritmos de controle, um com foco na redução do consumo de energia do MPSoC e outro com foco na manutenção dos requisitos de QoS de algumas aplicações.
- Sistema Operacional
  - Definição de chamadas de sistema para habilitar o monitoramento e/ou o controle do QoS de uma dada tarefa;
  - Definição de chamadas de sistema para configurar os parâmetros de QoS de uma dada tarefa;
  - Desenvolvimento de um algoritmo de controle do DFS baseado nas violações nos parâmetros de QoS monitorados;
  - Explorar a utilização de gerentes de aplicação, a fim de administrar a execução de uma dada aplicação.
- Ferramentas de Projeto
  - Desenvolver uma ferramenta de avaliação do consumo de energia na Plataforma HeMPS.

1.4 Originalidade da Tese

A originalidade do trabalho está no desenvolvimento de um método de controle da qualidade do serviço de um MPSoC baseado na utilização da técnica de DFS. Tal método irá atender duas métricas de QoS, vazão e latência. A técnica de DFS é aplicada nos PEs e na NoC independentemente, permitindo flexibilidade na estrutura de rede mantendo os PEs na frequência correta. No cenário de execução do MPSoC em que uma aplicação com requisitos de QoS é afetada por outra aplicação de propósito geral, a técnica de DFS é utilizada para ajustar a frequência do PE em que a aplicação com requisitos de QoS é executada. Desta forma, o atendimento do QoS poderá ser restabelecido, afetando minimamente o desempenho global do MPSoC.

O presente trabalho é desenvolvido no escopo de MPSoCs homogêneos com memória distribuída (troca de mensagens) e NoCs com topologia malha bidimensional, tendo como base o MPSoC acadêmico HeMPS (Capitulo 2, Seção 2.2.6).

1.5 Contribuição Original

Este trabalho deixa como contribuição original para a comunidade de pesquisa em MPSoCs um esquema de controle do QoS baseado na técnica de DFS. Mostrou-se, através dos resultados, que é possível obter uma sensível redução na energia consumida e simultaneamente atender aos requisitos de QoS da aplicação.

1.6 Organização do texto

O restante do texto está organizado em mais dez capítulos:

- O Capítulo 2 apresenta trabalhos relacionados ao tema da presente Tese de Doutorado, como MPSoCs baseados em NoCs e soluções que visam garantir os requisitos de desempenho das aplicações. O Capítulo encerra com a descrição do MPSoC HeMPS, o qual serve de base para a realização do presente trabalho e diversos outros no grupo de pesquisa GAPH;
- O Capítulo 3 – apresenta a técnica de DFS utilizada como base para o controle de QoS. Neste Capítulo são exibidos os resultados do emprego desta técnica na plataforma HeMPS, bem como as modificações sugeridas para que esta seja utilizada no controle de QoS de tarefas/aplicações;
- O Capítulo 4 – apresenta um estudo dos diversos tipos de monitoramento utilizados em MPSoCs, bem como as estruturas de monitoramento inseridas na plataforma HeMPS. O monitoramento adotado na presente Tese de Doutorado é classificado de acordo com a terminologia estudada neste Capítulo;
- O Capítulo 5 – apresenta um levantamento dos mecanismos de controle de

QoS existentes na HeMPS, a fim de realizar uma comparação de características gerais entre a técnica proposta neste trabalho e as demais existentes;

- O Capítulo 6 – apresenta o modelo de consumo de energia nos MPSoCs baseados em NoCs, considerado a primeira contribuição desta Tese de Doutorado. Neste Capítulo todos os componentes do modelo são descritos, como a estimativa do consumo de energia nos PEs, NoC e fios de interconexão;
- O Capítulo 7 – apresenta o foco principal da presente Tese de Doutorado, a técnica proposta de controle do QoS na HeMPS através do emprego de DFS. Neste Capítulo a técnica é descrita, e são apresentados módulos que realizam uma implementação desta técnica na plataforma HeMPS;
- O Capítulo 8 – apresenta a configuração da plataforma HeMPS utilizada, além de descrever os cenários de teste avaliados;
- O Capítulo 9 – apresenta os resultados dos cenários de teste avaliados com o objetivo de controlar a vazão de uma dada aplicação, utilizando o controle de QoS proposto nesta Tese de Doutorado;
- O Capítulo 10 – apresenta os resultados dos cenários de teste avaliados com o objetivo de controlar a latência de uma dada aplicação, utilizando o controle de QoS proposto nesta Tese de Doutorado;
- O Capítulo 11 – encerra esta Tese de Doutorado com uma conclusão acerca dos resultados obtidos e da utilização da técnica proposta. Neste Capítulo também são apresentadas direções para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Neste Capítulo são apresentados artigos relacionados com o trabalho. Este Capítulo é dividido em três sessões: (*i*) trabalhos relacionados a métodos de monitoramento em MPSoCs; (*ii*) trabalhos relacionados à qualidade de serviço em um MPSoC; e (*iii*) descrição do MPSoC HeMPS, o qual é utilizado como base para a implementação do esquema de QoS através de DFS.

2.1 Monitoramento em MPSoCs

2.1.1 Fattah et. al.

Em [FAT11] os Autores propõem um sistema de gerência hierarquizada, com gerentes em cada unidade de processamento (denominada *célula*), sendo cada uma destas células composta por: PE, roteador, memória cache e módulo de DMA. Um gerente de aplicação recebe informações de cada um dos monitores a ele associado, sendo capaz de atuar sobre suas células e consequentemente sobre suas tarefas. Um gerente global recebe informações de todos os gerentes de aplicação, sendo capaz de atuar sobre cada gerente de aplicação e assim gerir de forma geral o MPSoC. A Figura 1 mostra a arquitetura de monitoramento proposta.

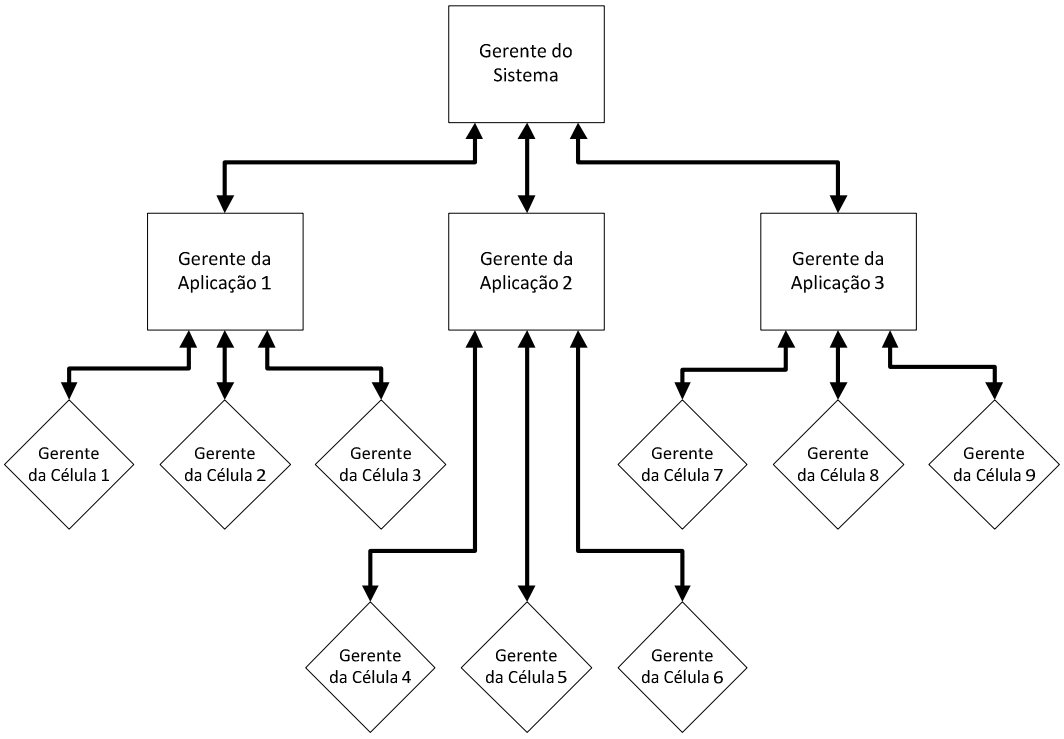


Figura 1 - Arquitetura de gerencia de um MPSoC do tipo hierárquica [FAT11].

Os Autores utilizam dois métodos distintos de forma combinada, o monitoramento centralizado e o distribuído, para construir a arquitetura de gerência hierárquica como descrita na Figura 1. Desta forma, o sistema de gerenciamento começa sua avaliação

diretamente em hardware, envia dados de monitoramento e relatórios para níveis de gerência superiores que retornam comandos a serem executados.

As *células* (Figura 2) constituem a menor unidade na hierarquia do sistema. De acordo com a necessidade do MPSoC, cada célula monitora diferentes variáveis, tais como: temperatura, potência e falhas. Além destes parâmetros monitorados, as células podem receber diferentes atuadores, tais como: módulos DVFS e módulos de reconfiguração. As células possuem um módulo de gerência local que avalia as variáveis monitoradas e envia relatórios para os gerentes superiores, além disso, as células implementam os comandos recebidos por tais gerentes.

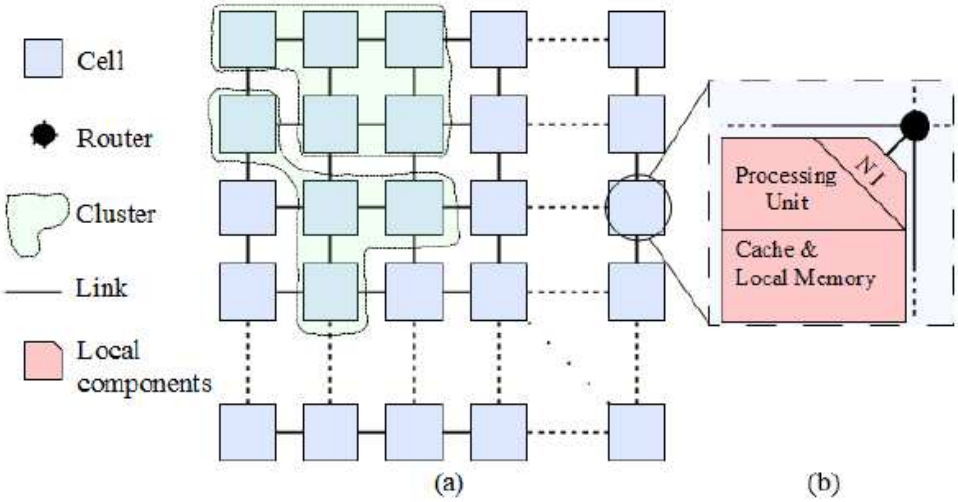


Figura 2 - (a) Uma rede 2D baseada na arquitetura de vários PEs (*células*), (b) possíveis componentes de uma célula [FAT11].

As células podem ser agrupadas, e este agrupamento (do inglês, *cluster*) é gerenciado por um gerente de nível intermediário. *Clusters* diferentes podem adotar políticas de controle diferentes: uma aplicação pode estar em múltiplos *clusters*, cada qual executando um conjunto de tarefas; ou cada *cluster* pode executar diferentes aplicações. Os gerentes de nível intermediário são denominados de gerente de aplicação. Cada gerente de aplicação administra os requisitos de sua aplicação, enviando comandos para seus gerentes de célula a fim de cumprir com todos os requisitos da aplicação.

Um gerente de alto-nível administra o MPSoC de forma global, coordenando as ações dos gerentes de aplicação. Este gerente é executado no nível de sistema operacional, sendo responsável por alocar novos gerentes de aplicação à medida que novas aplicações são alocadas no MPSoC.

Os Autores concluem que o sistema de gerência hierárquico atende aos requisitos dos futuros MPSoCs, explorando métodos de gerencia centralizado no nível de sistema, e distribuído no nível de células. Porém, os Autores não possuem nenhuma implementação, e afirmam que irão realizar futuramente seus testes na plataforma HeMPS ou SESC.

2.1.2 Stan et. al.

Em [STA11], os Autores 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 Figura 3 mostra a estrutura de monitoramento implementada em um dispositivo mapeado no espaço de endereçamento do processador.

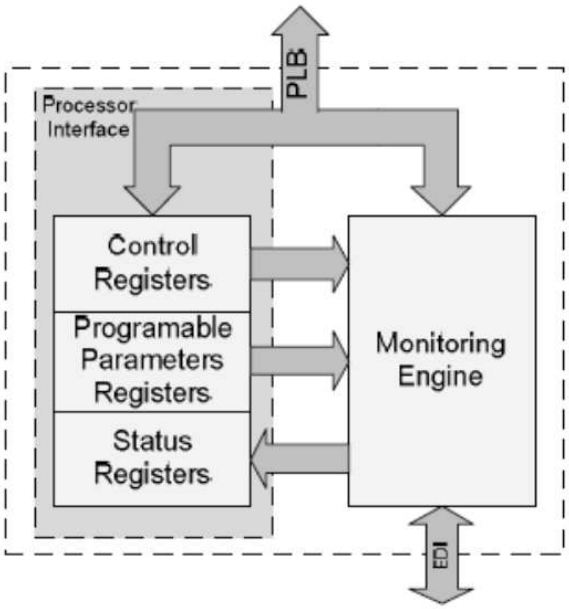


Figura 3 - Arquitetura da estrutura de monitoramento [STA11].

A estrutura de monitoramento implementa um mecanismo *watchdog* que conta a quantidade de eventos que são executados antes do tempo limite, entre dois limites de tempo ou depois de um tempo limite. Os intervalos de tempo utilizados para verificar um determinado evento são armazenados em parâmetros programáveis, que são configurados nas aplicações; estes 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.

2.1.3 Madalozzo et. al.

Em [MAD13], os Autores utilizam a plataforma HeMPS e apresentam um mecanismo de monitoramento de tarefas, que busca 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 respeitada; (*iii*) tarefas sendo executadas distantes de suas comunicantes, aumentando o consumo de energia na NoC. Os monitores propostos realizam, periodicamente, a coleta dos dados relativos aos

parâmetros de desempenho monitorados (como deadlines, vazão, latência). A técnica de monitoramento foi desenvolvida para as plataformas com gerência de recursos centralizada e distribuída, sendo o mesmo protocolo usado para ambas as plataformas.

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* 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 (do inglês, *Task Control Block*) da tarefa. O campo deadline foi adicionado na TCB, pois originalmente este não existia. Durante a análise de dados na etapa de *profiling*, deve-se também 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. 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 *deadline* antes de solicitar a operação de migração de tarefas. Logo, na etapa de *profiling* devem ser configurados três parâmetros: *deadline*, *max_miss*, e *max_miss_migra*.

No Figura 4, 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. Como explicado anteriormente, utiliza-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. Nesta chamada de sistema, atribui-se os valores parametrizáveis, na função, aos respectivos campos da TCB.

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

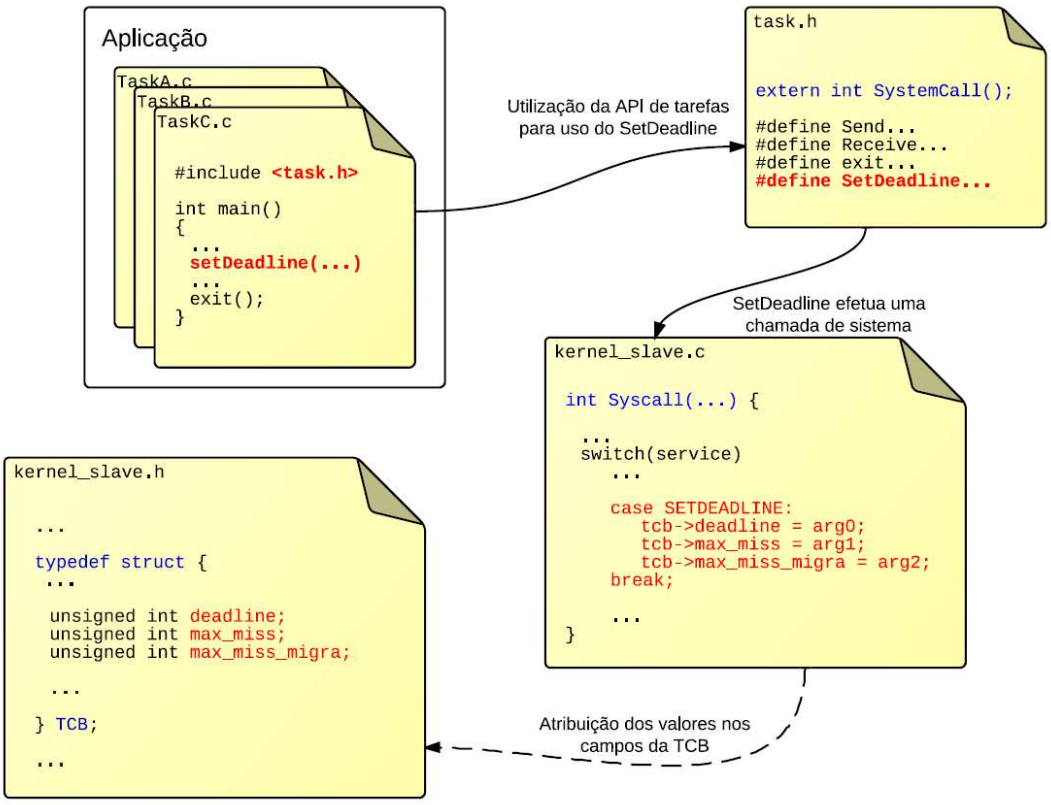


Figura 4 - Fluxo de configuração de tarefas para realizar o monitoramento [MAD13].

2.1.4 Considerações sobre monitoramento

Os trabalhos avaliados mostram que a utilização de estruturas de monitoramento é fundamental para se implantar nos MPSoCs políticas de controle do QoS nas aplicações. Diferentes tipos de monitores são utilizados, com implementações em hardware, em software ou uma combinação destes. Em relação ao controle dos parâmetros de QoS das aplicações, cada trabalho estudado avalia e controla um parâmetro específico de QoS (vazão, latência, *jitter* ou *deadlines*). O estudo dos trabalhos de monitoramento deste Capítulo permitiu o desenvolvimento do sistema de monitoramento utilizado no mecanismo de controle de QoS proposto.

No Capítulo 4 será apresentada uma classificação dos diferentes tipos de monitores que podem ser empregados nos MPSoCs, e o Capítulo 7 descreve o sistema de monitoramento desenvolvido no contexto desta Tese de Doutorado.

2.2 QoS em MPSoCs

2.2.1 Goosens et. al.

Em [GOO10], os Autores apresentam mecanismos de gerência de energia combinados com gerência de *deadlines* de tarefas de tempo real, porém visando a manutenção da modularidade (do inglês, *composability*) dos elementos compõem o MPSoC. A modularidade é necessária para isolar as aplicações, permitindo que seu

projeto, construção e verificação sejam realizados de forma independente facilitando sua implementação.

Para eliminar a interferência entre as aplicações que executam em um mesmo PE, é utilizada a política de multiplexação em divisões de tempo (do inglês, *Time Division Multiplexing*, ou TDM). Ao se utilizar esta política, é criado um escalonamento estático com espaços de tempo de execução constante para cada aplicação. Em todos os PE, o sistema operacional executa a política de TDM, permitindo o compartilhamento do PE por diversas aplicações com restrições de tempo de execução.

Os Autores utilizam como aplicação um conjunto de tarefas que se comunicam entre si em hardware ou software. Cada aplicação é independente, e baseado em suas restrições temporais pode ser classificada em um de três grupos: *hard real-time*, *soft real-time* e sem tempo-real.

As aplicações de *hard real-time*, ou FRT (do inglês, *Firm Real-Time*) são aquelas as quais a perda de um deadline ocasiona uma degradação inaceitável na qualidade da aplicação. Alguns exemplos de tais aplicações são decodificadores de áudio e *soft-radio*. O mapeamento das aplicações FRT em um conjunto de recursos exige uma análise formal (por exemplo, análise do fluxo de dados, cálculo de tempo real, etc.). Tais análises levam em consideração as condições de pior caso (pior tempo de execução das tarefas e pior tempo de recebimento dos dados de entrada) e geram mapeamentos e alocações de recursos que garantem o cumprimento dos *deadlines* em tempo de execução. Os recursos são alocados de acordo com o pior caso de comportamento da aplicação.

As aplicações de *soft real-time*, ou SRT (do inglês, Soft Real-Time) são aquelas as quais a perda ocasional de *deadlines* causa uma degradação aceitável na qualidade da aplicação. Exemplos de tais aplicações são alguns codificadores / decodificadores de vídeo, os quais a perda de um *deadline* pode ser compensada com a repetição do quadro anterior. As aplicações SRT são usualmente projetadas utilizando uma análise estática ou baseadas em um conjunto representativo de entradas. O mapeamento das aplicações e a alocação de recursos garantem que a aplicação irá cumprir seus *deadlines* na maior parte dos casos. Neste caso, os recursos são usualmente alocados para cobrir o pior caso da aplicação (de todas as suas tarefas), e não de cada tarefa individualmente. Em cada aplicação, o tempo ocioso de uma tarefa pode ser utilizado por outra tarefa. Esta ação reduz o risco da aplicação perder seu *deadline*, pois uma tarefa de baixa complexidade pode terminar mais cedo liberando tempo para uma tarefa mais complexa começar mais cedo e terminar dentro do período de seu *deadline*.

Finalmente, um sistema embarcado pode executar aplicações sem requisitos de tempo, as aplicações NRT (do inglês, *Non Real-Time*). Exemplos destas aplicações são as interfaces de usuário, download de arquivos, navegação de internet, etc. Neste caso, é importante maximizar a vazão global do sistema.

Os autores apresentam algoritmos de escalonamento de tempo real, mostrando

vantagens e desvantagens de cada um deles. A seguir, exemplificam algumas técnicas de gestão de energia em tempo de execução. Os autores mostram o modelo de MPSoC adotado, semelhante à HeMPS, com execução multitarefa e prioridades. Os autores fazem uma breve discussão de exemplos de modularidade nos diferentes recursos do MPSoC adotado, descrevendo como fazer uma gerência de energia modular. Basicamente este gerenciamento deve conter elementos cujos sinais de relógio sejam independentes. Para isso os autores sugerem que o processador, o DMA e a memória possuam sinais de relógio independentes.

Os Autores utilizam em seus PEs módulos de DFS, e apresentam o conceito de gerência de energia modular. Nesta gerência, a responsabilidade de gerir os recursos que não são compartilhados entre as aplicações é repassada para cada aplicação. Em contrapartida, os recursos que são compartilhados por mais de uma aplicação deverão manter uma frequência constante ou ser geridas por decisões alinhadas entre os sistemas operacionais dos PEs em que estão sendo executadas.

Os processadores são geridos no nível de escalonamento, ou seja, o sistema operacional realiza o escalonamento de uma tarefa, decide em que frequência irá executá-la de acordo com o tipo de aplicação (FRT, SRT ou NRT). Antes que uma aplicação seja trocada no processador, a frequência é alterada para a frequência da nova tarefa. O sistema operacional é executado com a máxima frequência para minimizar seu tempo de execução. Desta forma, quando a execução de uma tarefa é interrompida, o sistema operacional habilita a máxima frequência no processador antes de iniciar um novo escalonamento ou gerência de energia.

Os Autores afirmam que a sua gerência de energia é aplicada por recurso e por aplicação (inclusive por tarefa), ao invés de ser aplicada de forma global no MPSoC. Desta forma, diferentes políticas de gestão de energia podem ser aplicadas para cada aplicação, dependendo de cada tipo de aplicação.

2.2.2 Mansouri et. al.

Em [MAN10], os Autores propõem uma política de ajuste da técnica de DVFS dos PEs em um MPSoC baseado em troca de mensagens e consenso entre os PEs. O consenso é derivado da pesquisa em teoria do controle cooperativo, e foi desenvolvida para o processamento de rede de sensores e coordenação de multi-agentes. De forma sucinta, o consenso é definido como um processo iterativo que utiliza protocolos de trocas de mensagem, o que leva um conjunto de elementos comunicantes a concordarem em um dado valor ou certo comportamento. Os Autores pretendem empregar o consenso para chegar a um acordo que otimize o consumo global do MPSoC.

O sistema avaliado pelos Autores é a arquitetura MPSoC apresentada na Figura 5, que é composta por *N* PEs interconectados por uma NoC assíncrona. O modelo de consumo de energia é construído com base no seguinte cenário. Um mecanismo externo

é utilizado para mapear as tarefas nos PEs de maneira que os requisitos de tempo real sejam atendidos, e cada aplicação é executada em um intervalo de tempo pré-fixado. Para efeito de simplificação, todos os PE somente pode executar apenas uma tarefa.

O mecanismo de DVFS é controlado por um módulo de decisão local, que é executado como um processo em todos os PE quando houver necessidade de uma reconfiguração (nova alocação de tarefa, por exemplo). Este módulo de decisão local coleta informações de seus vizinhos para chegar a um consenso sobre a nova frequência.

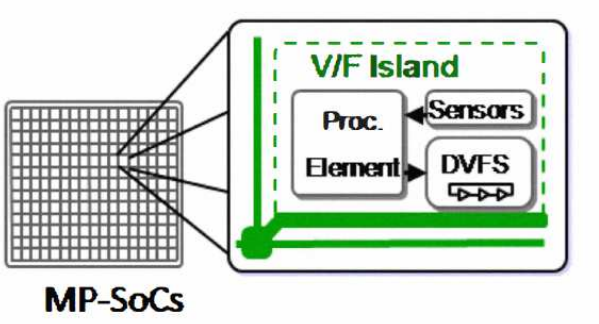


Figura 5 - Modelo de MPSoC adotado em [MAN10].

Os Autores utilizam como aplicação um receptor digital do tipo 3GPP-LTE, mapeado em um MPSoC baseado em NoC (como descrito em Figura 6). Os resultados exibidos mostram um ganho de até 87% em comparação com o pior caso de execução da aplicação.

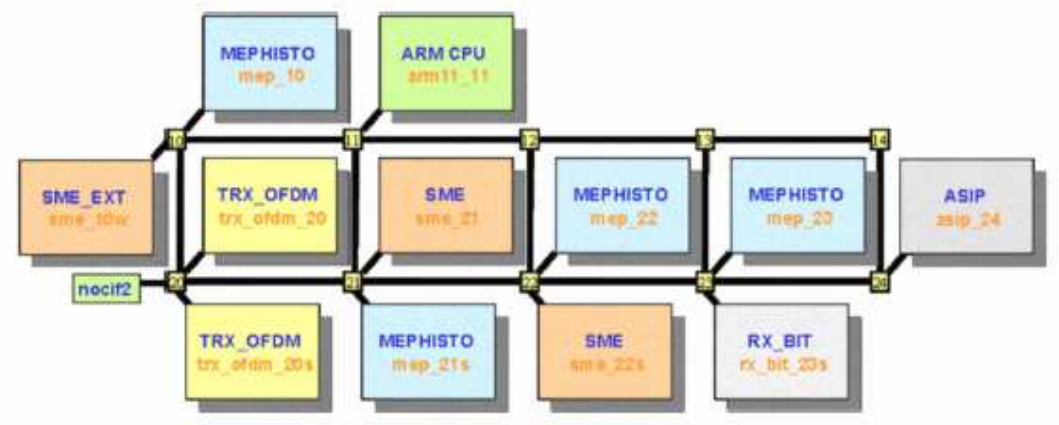


Figura 6 - Um possível mapeamento da aplicação do receptor 3GPP-LTE.

2.2.3 Garg et. al.

Em [GAR10], os Autores propõem uma política de controle de DVFS em ilhas de tensão e frequência, ou VFIs (do inglês, *Voltage and Frequency Islands*). O projeto de um MPSoC, baseado na utilização de VFIs, permite que cada ilha possua um gerente independente da política de DVFS. Esta combinação pode ser utilizada como uma solução em potencial para o aumento da dissipação de potência nos MPSoCs.

Um dos principais desafios no uso de MPSoCs com VFIs é o projeto de algoritmos de controle do DVFS efetivos e escaláveis. Estes algoritmos são utilizados para regular a

tensão e a frequência em uma VFI de acordo com a aplicação que ela executa. Este algoritmo deverá ser sensível à carga de trabalho da aplicação e às variações de processo e ambiente. Uma solução promissora para este algoritmo é a utilização de laços de controle com realimentação, que monitoram o estado do sistema e ajusta de forma adequada o DVFS de cada VFI. Os estados monitorados dependem do objetivo do controle, por exemplo dissipação de potência no MPSoC, controle da temperatura ou ocupação de filas de comunicação entre os processadores a fim de controlar o desempenho da aplicação.

A maior parte dos trabalhos anteriores na área de gerência da dissipação de potência em um circuito integrado possui foco em duas escolhas arquiteturais distintas, o controle totalmente centralizado, ou FC (do inglês, *fully-centralized*), e o controle totalmente descentralizado, ou FD (do inglês, *fully-descentralized*). Os controladores FC utilizam realimentação com o estado global para a tomada de decisões, neste caso cada VFI utiliza informações de todo o MPSoC para ajustar seu DVFS num dado intervalo de controle. Em contrapartida, em um controlador FD cada VFI utiliza apenas seu estado local para a tomada de decisões de controle. Enquanto o controlador FC proporciona o melhor desempenho, o acréscimo no custo global de comunicação atrelado ao número elevado de VFIs em um mesmo circuito resulta em um controle FC que possui implementação com custos proibitivos. Já o controle FD sacrifica o desempenho em favor de uma redução no custo de implementação.

Os controles FC e FD representam os opostos extremos no gráfico entre desempenho *versus* custo de implementação do controle de DVFS em MPSoCs baseados em VFIs (Figura 7). Os Autores exploram o espaço de projeto entre estes dois extremos, e propõem o algoritmo de controle com realimentação personalizado, ou CF (do inglês, *Custom Feedback*). Neste algoritmo, cada VFI realiza decisões de controle baseado na combinação do estado local com um estado global parcial (ao invés da utilização do estado global do MPSoC). Para criar um estado global parcial, o algoritmo utiliza algumas informações de VFIs vizinhas, sendo esta quantidade de informações parametrizável.

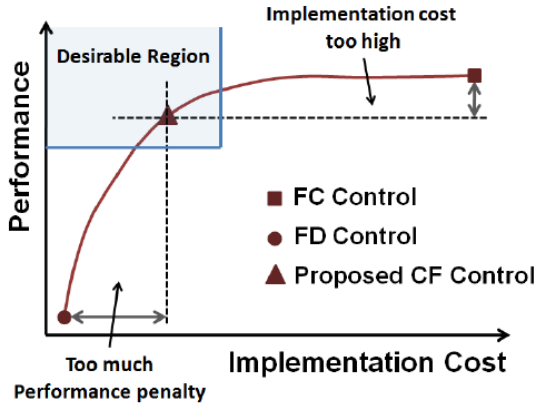


Figura 7 - Curva do desempenho x custo de implementação dos algoritmos de controle com realimentação [GAR10].

Os Autores avaliaram o algoritmo de controle proposto através de simulações utilizando uma aplicação de codificador de vídeo MPEG-2. Esta aplicação foi dividida em nove módulos funcionais, e cada qual foi mapeada em um processador ou memória. O grafo de tarefas da aplicação de codificação MPEG-2 é exibida na Figura 8. Os Autores utilizaram como objetivo do algoritmo de controle, a ocupação das filas de comunicação entre os processadores.

Os resultados obtidos mostram que o algoritmo FC manteve a ocupação com variação de 3% sobre o valor de referência (valor em que o algoritmo de controle deverá manter o sistema). Já o algoritmo FD manteve a ocupação das filas com variação de 30% em relação ao valor de referência. O algoritmo CF, com conexão a oito vizinhos para gerar um estado global, obteve como resultado uma variação máxima de 7% em relação ao valor de referência.

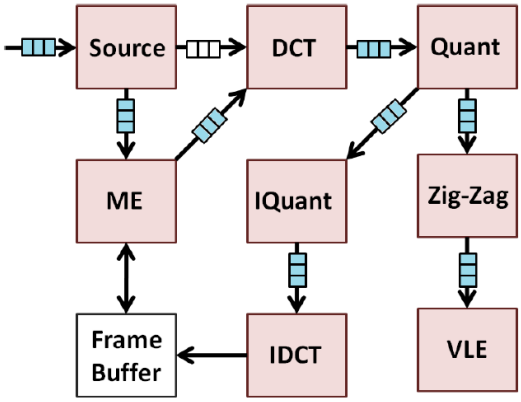


Figura 8 - O grafo de tarefas de um codificador de vídeo MPEG-2.

2.2.4 Rosa et. al.

Em [ROS12a] os Autores propõem o uso da técnica de DFS na plataforma HeMPS, que ao contrário da técnica de DVFS realiza a variação da frequência do MPSoC mantendo sua tensão fixa. Os Autores aplicam a técnica proposta nos elementos do MPSoC (NoC + PE) de forma independente. No PE a frequência é ajustada de acordo com as cargas de processamento e comunicação. Já na NoC, para cada pacote recebido no roteador a técnica escolhe a frequência adequada.

A Figura 9, exibe a arquitetura de um PE com o controlador DFS associado. Na interface entre o roteador e o processador foi utilizado o paradigma globalmente assíncrono e localmente síncrono, GALS (do inglês, *Globally Assynchronous Locally Synchronous*), utilizando uma estrutura de FIFO bissíncrona [CHE00] e sincronizadores do tipo two-flop para os sinais de controle. Os Autores modificaram o sistema operacional do processador, denominado de *microkernel*, a fim de realizar o monitoramento da carga do processador e a ocupação da fila de entrada de dados. O controlador DFS utiliza estas informações para tomar decisões e trocar a frequência do processador dinamicamente.

Cada módulo gerador de relógio local recebe o sinal de relógio do sistema como referência, e a partir deste produz um novo sinal de relógio. O principal benefício desta arquitetura é que o relógio global do sistema é utilizado somente para alimentar os módulos geradores de relógio local, reduzindo ambas a carga no sinal de relógio global e a árvore de relógio do sistema. Esta árvore é responsável por, pelo menos, 40% da dissipação de potência em MPSoCs totalmente síncronos [ROS12a].

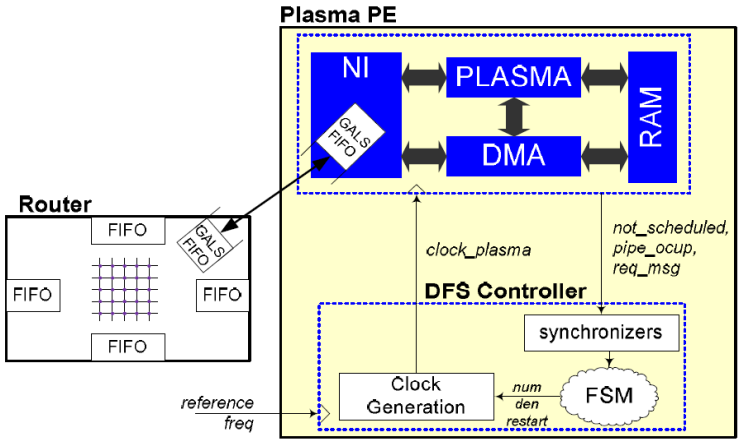


Figura 9 - Arquitetura de PE proposta em [ROS12a], destacando-se a interface GALS entre o processador e o roteador.

O módulo gerador de relógio local é capaz de suprimir uma porção determinada do sinal de relógio de referência, criando assim o novo sinal de relógio. Para uma frequência correspondente a 50% do sinal de referência, o módulo gerador é capaz de suprimir 1 ciclo de relógio a cada 2 ciclos de relógio de referência.

Os Autores avaliaram a arquitetura proposta utilizando três aplicações: (*i*) uma sintética, sendo um *pipeline* com 6 tarefas; (*ii*) duas reais, VOPD e MPEG. O MPSoC utilizado é descrito em VHDL, sintetizado para uma biblioteca *standard cells* (ST 65 nm) e simulada no nível RTL. Os Autores selecionaram em sua técnica de DFS nove níveis de frequência que variam de 6,67 a 100 MHz. Estas frequências correspondem a 6,67%, 10%, 25%, 40%, 50%, 60%, 75%, 90% e 100% do valor da frequência de referência.

Os resultados para a simulação da aplicação sintética apresentaram uma redução de 20,85% na dissipação de potência dos processadores e 73% de redução na dissipação de potência da NoC. Para as simulações das aplicações reais (VOPD e MPEG), os resultados mostram uma redução média de 27,7% e 26,6% na dissipação de potência para os processadores e 75% para a NoC. Pode-se notar que a técnica proposta foi capaz de produzir uma significativa redução na dissipação da NoC, porém esta dissipação corresponde somente a 5% da dissipação global do MPSoC. A Tabela 1 e a Figura 10 apresentam os resultados da avaliação da dissipação de potência com e sem a utilização da técnica de DFS na HeMPS ao se executar a aplicação de MPEG. Os resultados abaixo exemplificam apenas uma simulação da aplicação MPEG, e não a média das simulações avaliadas em [ROS12b].

Tabela 1 - Resultados da dissipação de potência para a aplicação MPEG com e sem a utilização da técnica de DFS [ROS12b].

	Dissipação Total (mW)				
	RAM	**Processador**	**Controlador DFS**	**PEs**	**NoC**
Com DFS	39,04	9,05	1,77	51,17	1,61
Sem DFS	76,34	21,09	-	103,63	6,98
Redução	46,86%	57,08%	-	50,62%	76,09%

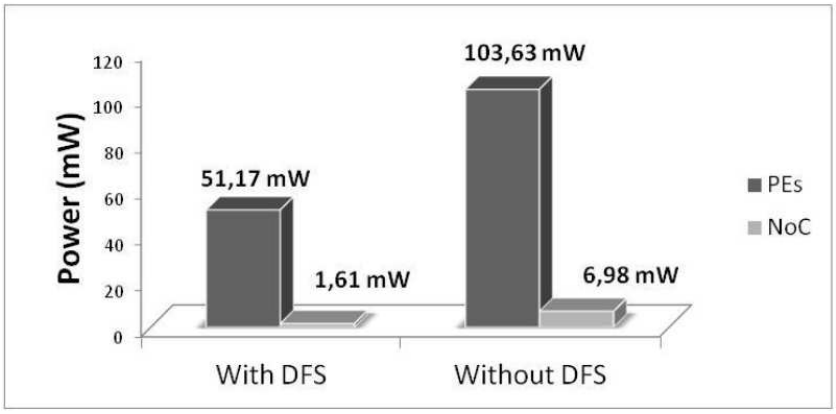


Figura 10 - Comparação entre as dissipações de potência da NoC e dos PEs para a aplicação MPEG [ROS12b].

Os Autores concluem afirmando que sua técnica foi capaz de reduzir a frequência global do MPSoC, reduzindo assim sua dissipação de potência. Entretanto, a redução na frequência dos PEs produz um aumento no tempo de execução das tarefas variando de 3,2% a 18%. A redução da dissipação de potência é maior na NoC do que nos processadores. Isto se dá pelo fato de que a NoC fica em espera com uma frequência muito baixa, enquanto os processadores executam suas tarefas.

2.2.5 Höppner et. al.

Em [HOP12] os Autores propõem um controlador de DVFS implementado em cada um dos PEs de um MPSoC heterogêneo. Este controlador permite a troca da tensão do PE entre diversas fontes de tensão presentes no interior do circuito. As perturbações nas redes de distribuição de tensão provocadas por tais trocas são evitadas utilizando um número configurável de chaveadores de pré-carga. O controlador proposto permite a utilização de corte de tensão, PSO (do inglês, *Power Shut-Off*), religamento, PU (do inglês, *Power-Up*) e trocas de níveis de tensão, SC (do inglês, *Supply level Changes*).

A Figura 11 exibe o *wrapper* do PE proposto pelos Autores para um MPSoC heterogêneo. Diversas fontes de tensão são geradas fora do circuito integrado e são distribuídas por todo o MPSoC. Os PEs são conectados a um destes domínios de tensão utilizando os chaveadores de potência. Estes chaveadores são construídos utilizando dispositivos PMOS com alto limiar de tensão e são integrados a biblioteca de células padrão.

O sinal de relógio é gerado a partir de um circuito ADPLL (do inglês, *All Digital*

Phase-Locked Loop) local conectado a um gerador de relógio com laço aberto, que é capaz de trocar a frequência de saída de forma rápida [HOP11]. A troca de tensão ou frequência do PE é controlada através do controlador de dissipação de potência, PMC (do inglês, *Power Management Controller*), que recebe comandos da NoC ou de um JTAG utilizado para testes e medições. Os Autores afirmam que o *wrapper* proposto é utilizado em um MPSoC heterogêneo do tipo GALS.

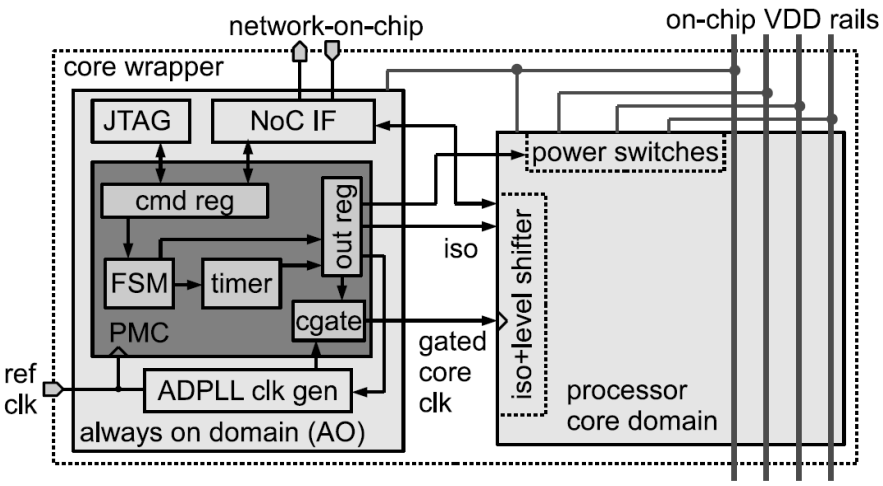


Figura 11 - *Wrapper* de gerência da dissipação de potência de um PE no MPSoC.

Os Autores apresentam resultados obtidos diretamente no circuito integrado, que foi construído utilizando a tecnologia de 65 mm. Este circuito possui 4 domínios de tensão diferentes, onde um deles opera como um domínio sempre ligado. Os resultados obtidos pelos Autores mostram que sua arquitetura proporciona uma troca rápida entre domínios de tensão, com reduzida perturbação nas redes de distribuição de tensão do circuito integrado. Estas trocas de tensão são realizadas por um controlador que gerencia a aplicação do DVFS no MPSoC.

2.2.6 Considerações sobre QoS em MPSoCs

Os trabalhos avaliados sobre o tema do controle de parâmetros de QoS em MPSoCs, mostram que diversos trabalhos utilizam diferentes técnicas, tais como: DFS, DVFS, algoritmos de roteamento e TDM, para cumprir com o objetivo de controlar diversos parâmetros de QoS em aplicações do MPSoC. Estas técnicas podem ser aplicadas em tempo de projeto (algoritmos de roteamento) ou em tempo de execução (DFS, DVFS e TDM).

A avaliação dos trabalhos de controle de QoS em MPSoCs realizada neste Capítulo foi importante para a escolha da técnica de adaptabilidade implementada no mecanismo proposto bem como a escolha dos parâmetros de QoS a serem controlados. No Capítulo 5 são apresentados os mecanismos de controle de QoS implementados na plataforma HeMPS, e no Capítulo 7 é descrito o mecanismo proposto de controle do QoS em MPSoCs (tendo como plataforma alvo a HeMPS).

2.3 HeMPS

HeMPS (*Hermes Multiprocessor System*) [CRI07][CAR09a] é um MPSoC homogêneo baseado no processador Plasma e na NoC Hermes. A Figura 12 ilustra uma instância 2x3 do MPSoC mostrando seus principais componentes de hardware. O elemento de processamento Plasma-IP contém um processador RISC baseado no processador MIPS (Plasma), interface de rede (NI – *Network Interface*), DMA e memória local (*scratch pad*). A NoC Hermes emprega uma topologia malha 2D e chaveamento por pacotes (*wormhole*). Os roteadores têm *buffers* de entrada, um árbitro centralizado, um *crossbar* e até cinco portas bidirecionais. O árbitro serve as portas de entrada utilizando escalonamento *round-robin* e os pacotes são roteados utilizando o algoritmo de roteamento XY. O sistema é conectado a uma memória externa chamada de repositório (*task repository*), a qual armazena os códigos objetos das tarefas que compõem as aplicações.

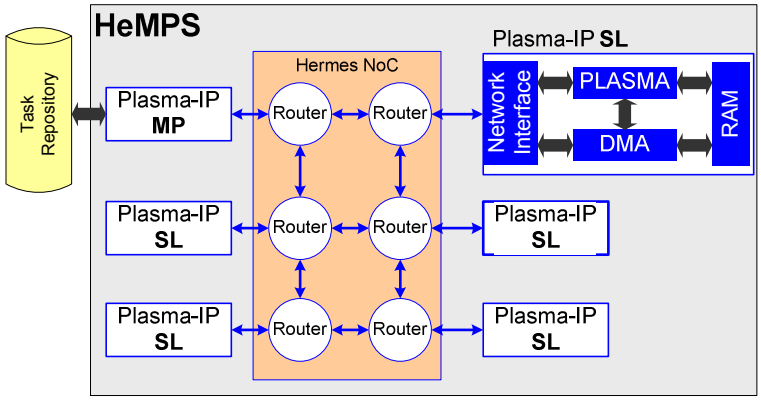


Figura 12 – Instância 2x3 do MPSoC HeMPS[CAR09a].

Tipicamente os sistemas computacionais são divididos em camadas com diferentes níveis de abstração a fim de gerenciar sua complexidade. Cada camada tem uma funcionalidade específica e se comunica com as camadas adjacentes. Dessa maneira, cada camada se serve dos serviços oferecidos pelas camadas inferiores e fornece serviços às camadas superiores. A Figura 13 ilustra as diferentes camadas do MPSoC HeMPS e as entidades correspondentes.

As camadas destacadas na Figura 13 possuem a seguinte funcionalidade:

- Camada de aplicação (*Application*): contém a descrição das aplicações a serem mapeadas no sistema. Cada aplicação é descrita como um grafo de tarefas;
- Camada de tarefa (*Task*): contém a descrição de cada tarefa usando as primitivas implementadas pela API fornecida pelo sistema operacional;
- Camada de sistema operacional (*OS*): é composta por um conjunto de *drivers* responsáveis pelo carregamento de tarefas, gerenciamento da memória local e do DMA, empacotamento/desempacotamento de

mensagens. Além disso, realiza o escalonamento de tarefas e implementa a API;

- Camada de transporte (*Transport*): executa a injeção/recepção de pacotes e controle de fluxo;
- Camada de rede (*Network*): responsável pela transmissão de pacotes sem perda de dados.

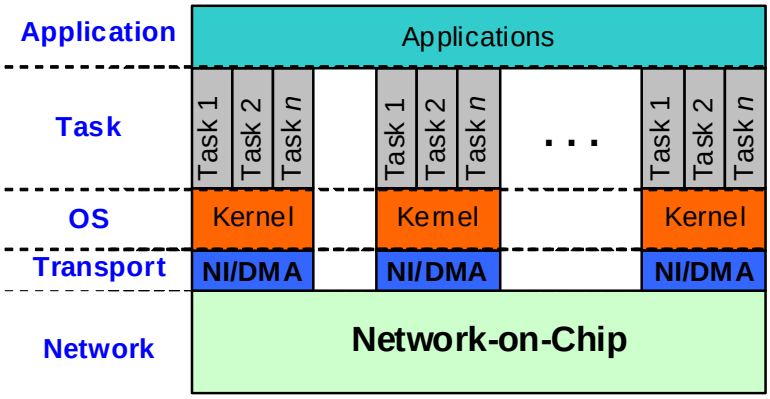


Figura 13 - Camadas do MPSoC HeMPS e suas entidades [CAR09b].

Aplicações que executam em MPSoCs possuem muitas vezes uma carga dinâmica, onde as tarefas que as compõem são alocadas sob demanda. Isso implica um número variável de tarefas executando simultaneamente, o qual pode exceder a quantidade de recursos de processamento disponível. Para lidar com esta questão, a arquitetura da HeMPS assume que: (*i*) aplicações são modeladas usando grafos de tarefas e (*ii*) somente um subconjunto de tarefas é inicialmente alocado no sistema (mapeamento estático). As demais tarefas são armazenadas no repositório e alocadas sob demanda (mapeamento dinâmico).

O sistema tem um processador mestre (Plasma-IP MP) que é responsável pelo gerenciamento dos recursos de computação. Esse é o único processador do sistema que possui acesso ao repositório. Quando o sistema inicia sua execução, o processador mestre aloca nos processadores escravos (Plasma-IP SL) as tarefas estaticamente mapeadas. Durante a execução, as tarefas já alocadas requisitam ao processador mestre novas tarefas, as quais são dinamicamente alocadas. À medida que as tarefas alocadas terminam de executar, recursos de processamento tornam-se disponíveis.

Para otimizar o desempenho nos PEs, a arquitetura do Plasma-IP separa a computação da comunicação. A interface de rede e o DMA são responsáveis por enviar e receber pacotes (comunicação) enquanto o processador Plasma executa as tarefas (computação). A memória local é uma RAM dupla porta que permite acesso simultâneo do processador e do DMA, habilitando esta separação computação-comunicação.

2.3.1 Microkernel

Cada processador escravo executa um *microkernel* (sistema operacional de pequeno porte) que suporta a execução de múltiplas tarefas e a comunicação entre estas. A memória local do Plasma-IP é dividida em páginas, das quais as primeiras são alocadas pelo *microkernel* e as subsequentes pelas tarefas das aplicações. O escalonamento das tarefas é preemptivo e baseado em um política *round-robin*. O *microkernel* possui uma tabela de tarefas com a localização de todas as tarefas alocadas no sistema (locais e remotas).

As páginas são protegidas pelo *microkernel* e toda comunicação entre tarefas é realizada através de troca de mensagens. A comunicação é suportada através de um vetor global de mensagens alocado no *microkernel* (*pipe*) e primitivas de comunicação (`Send()` e `Receive()`) compõem a API de comunicação. O *pipe* armazena mensagens enviadas pelas tarefas locais. As primitivas de comunicação são implementadas pelo *microkernel* e invocadas pelas tarefas através de chamadas de sistemas.

O modelo de computação que garante a sincronização entre tarefas é baseado em redes de processos de Kahn (KPN - Kahn Process Networks) [KAH74]. KPN é um modelo de computação distribuído onde canais de comunicação baseados em FIFOs infinitas (*pipes*) conectam os processos uns aos outros, formando uma rede. Esse modelo se baseia no princípio fundamental de que a leitura do canal de comunicação deve ser bloqueante e a escrita deve ser não-bloqueante.

O protocolo de comunicação adotado é o read request [BAG08], o qual garante o controle de fluxo fim-a-fim e o ordenamento de mensagens. Quando uma tarefa qualquer executa a primitiva `Send()`, a mensagem é armazenada no *pipe* local e a computação continua, caracterizando uma escrita não-bloqueante. Ao se executar a primitiva `Receive()`, a mensagem pode ser lida do *pipe* local ou de um *pipe* remoto. Se as tarefas comunicantes estão alocadas no mesmo PE, a mensagem é lida do *pipe* local. Caso contrário, o *microkernel* envia uma requisição de mensagem para o PE onde a mensagem está armazenada, e a tarefa entra em estado de espera (perde o processador). Quando a mensagem requisitada é recebida, o *microkernel* a armazena na página da tarefa destino e muda seu estado para pronta. A Figura 14 ilustra a comunicação remota entre tarefas. Na Figura 14(a) é assumido que a tarefa *t2* armazenou uma mensagem no *pipe* (*global_pipe*) endereçada à tarefa *t5* (`Send(&msg,t5)`), a qual requisita uma mensagem à tarefa *t2* (`Receive(&msg,t2)`). Na Figura 14(b) a mensagem requisitada (*msg*) é transmitida e armazenada na página da tarefa *t5*. O *microkernel* garante o ordenamento das mensagens numerando-as à medida que são armazenadas no *pipe*.

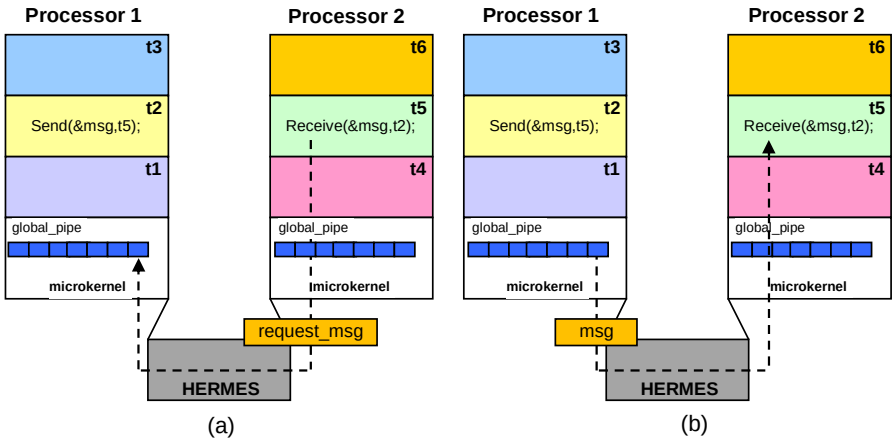


Figura 14 – Comunicação remota entre tarefas [CAR09a].

2.3.2 Mapeamento dinâmico de tarefas

O gerenciamento realizado pelo processador mestre durante a execução do sistema inclui (*i*) alocação de tarefas; (*ii*) broadcast de mensagens de controle (baseado em *unicast*) como localização das tarefas alocadas e notificação de tarefas desalocadas; (*iii*) tratar mensagens de controle enviadas pelos processadores escravos como fim de tarefa e requisição de tarefa. Ele não executa tarefas de aplicações.

A carga de trabalho dinâmica ocorre através da alocação de tarefas sob demanda durante a execução das aplicações. O desenvolvedor de uma aplicação define o subconjunto de tarefas necessário para o início da execução. O evento que dispara a alocação de uma nova tarefa é a execução da primitiva `Send()`. Cada vez que uma tarefa executa a primitiva `Send()`, o microkernel verifica na tabela de tarefas se a tarefa destino da mensagem está alocada no sistema. Se a tarefa está alocada, a mensagem é armazenada no *pipe*. Caso contrário, o microkernel envia ao processador mestre uma requisição de tarefa. Esse processo é transparente ao nível para a apliação..

Ao receber uma mensagem de requisição de tarefa, o processador mestre busca a tarefa no repositório de tarefas e a transmite a um processador escravo que tenha uma página disponível. Concluída a transmissão, o processador mestre notifica todos os escravos sobre a localização da nova tarefa alocada. O algoritmo de mapeamento utilizado para selecionar o processador escravo procura minimizar a distância entre as tarefas comunicantes [MAN11].

3 GERÊNCIA DA DISSIPAÇÃO DE POTÊNCIA EM UM MPSOC UTILIZANDO A TÉCNICA DE DFS

Neste Capítulo é apresentada a técnica de DFS utilizada como base para a presente Tese. Esta técnica foi desenvolvida no escopo da dissertação de mestrado de [ROS12b], e foi implementada no MPSoC HeMPS.

3.1 Gerência Dinâmica da Dissipação de Potência

A gerência dinâmica da dissipação de potência, ou DPM (do inglês, *Dynamic Power Management*), é empregada para reconfigurar um sistema eletrônico, a fim de prover os serviços requeridos e os níveis de desempenho adequados com o mínimo de componentes ativos e/ou o mínimo de carga nestes componentes [BEN98] [LOR98]. O DPM é composto por um conjunto de técnicas e/ou mecanismos para a construção de ambientes com consumo de energia eficientes. Isto é realizado com o desligamento ou com a redução do desempenho de alguns componentes do sistema, os quais não estão sendo totalmente utilizados. Existem duas premissas principais para a aplicação da DPM: (*i*) o sistema e seus componentes apresentam durante a sua operação uma carga de trabalho não-uniforme e (*ii*) é possível prever, com certo nível de exatidão, a oscilação na carga de trabalho do sistema.

Os mecanismos de DPM são diferenciados de acordo com o nível de elemento em que foi associado (e.g., componente, sistema, rede), além do estilo de realização física em que foi realizado (e.g., temporizador, controlador físico, rotina de software) [BEN00]. O agente responsável pela tomada de decisões é chamado de gerente de potência, ou PM (do inglês, *Power Manager*), e suas decisões são tomadas baseadas em um conjunto de parâmetros ou regras, chamadas de política. Não obstante, a maneira pela qual o sistema emprega o DPM, com reconfiguração dinâmica e com a escolha de uma política adequada, pode causar um grande impacto na dissipação de potência global do sistema. Desta forma, o projeto de um PM eficiente é decisivo na maximização da redução do consumo de energia de um sistema.

3.2 DFS e DVFS

A técnica de DVFS realiza o controle da tensão de alimentação e a frequência de operação em tempo de execução. O controle apenas da frequência de operação é uma alternativa ao uso do DVFS, e é à base de funcionamento do DFS. O DFS é uma solução interessante quando o acréscimo de área gerado pelo módulo de controle de tensão é um problema, quando projetistas precisam utilizar gerentes de modos de operação mais rápidos e simples, ou quando não estão disponíveis mecanismos de hardware para realizar a troca de várias fontes de alimentação (o que acontece nos dispositivos FPGAs modernos).

As técnicas de DVFS e DFS podem ser controladas por apenas componentes de hardware, por algoritmos em software, ou por uma configuração híbrida de hardware e software. No controle puramente implementado em hardware, sensores ou monitores dedicados são utilizados para coletar informações do hardware avaliado (e.g., temperatura, vazão, latência), e enviá-las para o PM. Em contrapartida, o sistema operacional ou o *microkernel* é o principal atuador em uma implementação de controle puramente em software. Neste caso, o sistema operacional precisa processar todo o conjunto de dados disponível, trabalhando como um PM, além de informar ao hardware qual tensão e/ou frequência foi selecionada.

Em uma solução híbrida, o hardware pode coletar diversas informações e enviá-las ao software, que é responsável somente pelo processamento dos dados de controle. Esta solução apresenta um pequeno acréscimo de hardware (visto que o hardware não precisa ter nenhuma capacidade de processamento) que mantém um nível preciso de informações enviadas ao software, este processa estas informações e informa o hardware quais ações deverão ser adotadas. Além disso, o esquema de controle pode ser centralizado ou distribuído. O controle centralizado utiliza informações globais do sistema para a tomada de decisões, já o descentralizado utiliza apenas informações locais do hardware de cada PE.

Nos MPSoCs baseados em NoCs, o esquema de DVFS pode ser inserido no elemento de processamento (PE), no roteador da NoC, ou em ambos. A utilização do DVFS em apenas um componente do MPSoC, pode levar a uma arquitetura que não explora todo o seu potencial de redução no consumo de energia. Desta forma, na maioria do trabalhos relacionados, tratam o PE e o roteador como um único módulo de hardware, no qual a técnica de DVFS atua. Entretanto, para se obter a máxima redução no consumo de energia, a técnica de DVFS deve ser aplicada em cada módulo separadamente explorando as características específicas de cada componente.

As novas tecnologias de circuitos CMOS apresentam alta variabilidade no processo de fabricação, que limitam o emprego da técnica DVFS. Estas variações podem produzir circuitos que possuem um DVFS nominalmente correto, incapazes de atingir seus objetivos de frequência ou dissipação de potência [GAR09] [HER09]. Desta forma, as técnicas de DVFS devem lidar com a variação de projetos em tecnologias com escala nanométricas, para que produzam os resultados esperados. Além disso, as tecnologias mais recentes (como a de 45 nm, por exemplo) estão restringindo as margens para tensões de alimentação, o que é o principal parâmetro de redução de dissipação de potência da técnica de DVFS [CHA11]. Consequentemente, projetar o sistema para trabalhar com uma tensão de alimentação fixa, juntamente com o uso da técnica de DFS pode se tornar uma opção eficiente de gestão de consumo de energia em MPSoCs.

3.3 DFS aplicado à arquitetura HeMPS

Diversas modificações foram realizadas na arquitetura HeMPS original, a fim de realizar a implementação da técnica de DFS. A Figura 15 apresenta a arquitetura da HeMPS com suporte a DFS. A técnica de DFS é aplicada em ambos os PEs e roteadores, e as principais modificações no hardware da HeMPS estão destacados na Figura 15. As filas de entrada dos roteadores foram substituídas por filas bisíncronas, exibidas na Figura 15(a) como retângulos vermelhos. Um esquema de sincronização foi inserido para evitar problemas de metaestabilidade [RAB03], já que os módulos que trabalham com frequências e domínios de relógio diferentes devem se comunicar.

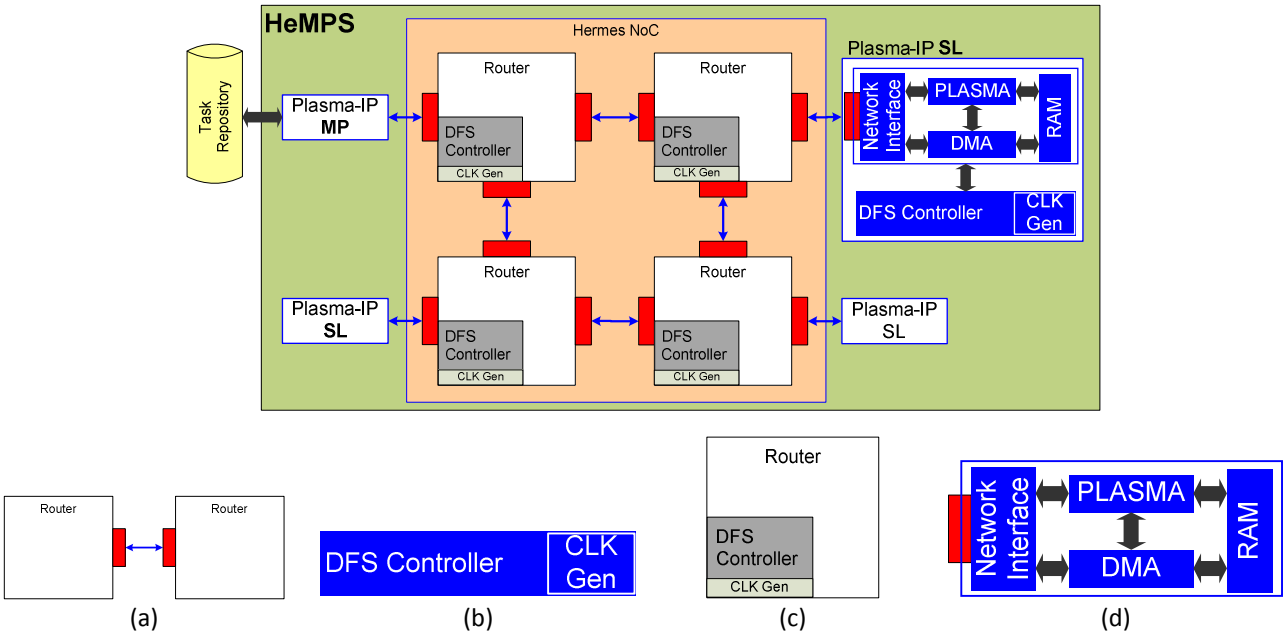


Figura 15 - Arquitetura HeMPS com suporte a DFS [ROS12b].

As Figura 15(b) e Figura 15(c) mostram os dois tipos de controladores de DFS utilizados. O primeiro é inserido junto ao PE e seu funcionamento é descrito na Seção 3.3.2, já o segundo é inserido junto aos roteadores e seu funcionamento é descrito na Seção 3.3.3. No módulo de interface de rede, ou NI (do inglês, *Network Interface*), foi substituída a fila de entrada por uma fila bisíncrona. Além disto, a NI foi modificada para que a frequência de operação de um processador remoto seja disponibilizada para o controlador de DFS do PE, através de informações contidas nos pacotes da NoC. Diversos registradores mapeados em memória foram adicionados ao PE, porém não estão exibidos na Figura 15. Além destas modificações, monitores foram inseridos no *microkernel* a fim de avaliar a utilização do processador e a ocupação do vetor de comunicação, ou *pipe*, e armazená-los em registradores mapeados em memória.

3.3.1 DFS no Nível do Processador

Esta Seção apresenta o esquema de DFS aplicado nos PEs da HeMPS. O controlador de DFS calcula as cargas de comunicação e de processamento, de acordo

com valores obtidos pelo *microkernel*. O controlador utiliza estes valores calculados para definir a frequência de operação do PE. O controlador sempre opera na frequência de referência, sendo esta a maior frequência do sistema, e provê ao PE uma frequência derivada da frequência de referência. A frequência fornecida ao PE é gerada no módulo de geração de relógio local (CLK Gen, na Figura 15), e sua operação é detalhada na Seção 3.3.1.1. A Seção 3.3.1.2 apresenta o funcionamento do controlador DFS e a política adotada para o controle da frequência do PE.

3.3.1.1 Módulo Gerador do Sinal de Relógio

O módulo de geração de relógio local é o mesmo utilizado para prover um sinal de relógio tanto para o PE, quanto para o roteador. Este módulo é responsável por gerar um sinal de relógio local, derivando o sinal de relógio de referência. O processo de geração do sinal de relógio local é baseado na divisão do sinal de relógio de referência através do corte de alguns de seus ciclos. Por exemplo, para gerar um sinal de relógio local que corresponde a 40% (2/5, dois quintos) do sinal de relógio de referência, basta habilitar 2 ciclos de relógio para cada 5 ciclos do relógio de referência. O exemplo descrito acima pode ser visualizado na Figura 16.

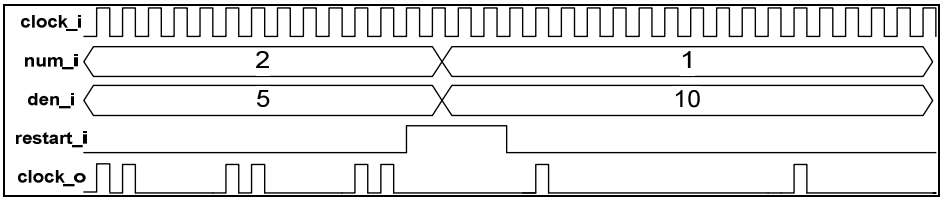


Figura 16 - Exemplo do processo de geração do sinal de relógio local. O sinal *clock_i* é o sinal de relógio de referência, e o sinal *clock_o* é o sinal de relógio local.

O controle da geração de frequências locais é realizado através de seus sinais de entrada, *num_i*, *den_i* e *restart_i*. Para cada *den_i* ciclos do relógio de referência, *num_i* ciclos são propagados para o sinal de relógio local. As exceções para esta regra geral são as seguintes, o sinal *den_i* não pode ser nulo (*den_i*=0), se o sinal *num_i* for nulo o sinal de relógio local será desligado, e para se gerar um sinal de relógio local $num\_i \leq den\_i$. Antes de se alterar os valores de *num_i* e *den_i* o sinal *restart_i* deve ser ativado para momentaneamente parar o sinal de relógio local e reinicializar os registradores internos do módulo gerador. Após a desativação do sinal *restart_i*, o sinal de relógio local será gerado com base nos sinais *num_i* e *den_i*.

Este esquema de geração do sinal de relógio local pode ser traduzido em uma lógica simples, que permite multiplicar o sinal de relógio de referência por valores selecionados no intervalo fechado [0, 1]. A quantidade de valores distintos presentes neste intervalo é definida pelos valores definidos por *num_i* e *den_i*, os quais definem o tamanho máximo dos registradores internos que irão armazená-los. Este esquema de geração de relógio local pode ser classificado como *ratiochronous* [CHA09], já que qualquer relação entre

duas frequências utilizadas no sistema é um número racional.

A principal desvantagem deste esquema é a maneira de se conectar com a técnica de gerência de tensão, e o projeto dos caminhos críticos de todos os módulos no sistema que utilizam o sinal de relógio de referência. Como exemplifica a Figura 16, para quaisquer frequências geradas, a duração de um período do sinal de relógio será o mesmo que o período do sinal de relógio de referência.

3.3.1.2 Controle do DFS no PE

A arquitetura do PE da HeMPS com suporte a DFS é apresentada na Figura 9. A interface NoC-processador foi modificada para trabalhar de acordo com o paradigma GALS. Os *buffers* locais da NoC e a interface de rede (NI) foram modificados, inseriu-se em seu lugar filas bissíncronas e sincronizadores do tipo *two-flop* nos sinais de controle. A interface NoC-processador possui dois *buffers* (filas do tipo GALS), uma no roteador, para receber dados da NI e outra na NI, para receber dados da NoC. Nesta nova arquitetura da HeMPS, a técnica de DFS foi implementada apenas nos PEs escravos. O PE mestre sempre trabalha na frequência de referência, a fim de evitar o surgimento de um gargalo no sistema.

As métricas utilizadas no esquema de DFS do PE são a carga de comunicação e a utilização do processador. Ao se monitorar somente a carga de comunicação podem-se encontrar problemas quando o *pipe* estiver vazio ou com poucas mensagens armazenadas, neste caso a frequência deveria ser aumentada. Entretanto, se a tarefa estiver bloqueada, e.g. esperando por uma mensagem de outra tarefa, e o *pipe* estiver com uma ocupação baixa, a frequência pode ser mantida ou até mesmo reduzida (cenário 4 na Tabela 2). Da mesma maneira, ao se monitorar apenas a utilização da CPU podem-se causar problemas quando esta estiver próxima de 100%, neste caso a frequência deveria ser aumentada. Entretanto, quando se obtém uma alta utilização de CPU e alta ocupação no *pipe* (cenário 1 na Tabela 2) a frequência não precisa ser aumentada, já que as mensagens estão sendo produzidas em uma taxa maior do que estão sendo consumidas.

Tabela 2 - Possíveis cenários de monitoramento dos parâmetros e ações de DFS.

		Utilização da CPU	
		Alta	Baixa
Carga de Comunicação	Alta	Diminuir Frequência (cenário 1)	Diminuir Frequência (cenário 2)
	Baixa	Aumentar/Manter a Frequência (cenário 3)	Diminuir/Manter a Frequência (cenário 4)

O mesmo raciocínio pode ser aplicado para os outros dois cenários da Tabela 2. No cenário 2, uma alta utilização do *pipe* combinada a uma baixa utilização de CPU significa

que, mesmo com uma baixa taxa de processamento a produção de mensagens é maior que o consumo, podendo-se baixar a frequência de operação. No cenário 3, duas ações podem ser tomadas: (*i*) aumentar a frequência do processador se a utilização estiver próxima a 100%, para evitar que o *pipe* fique vazio; e (*ii*) manter a mesma frequência se o processador não estiver próximo a 100% de utilização, isto significa que o processador ainda não foi totalmente utilizado para a atual frequência de operação.

A Figura 17 exibe o controlador DFS e suas interfaces. Este módulo utiliza os seguintes dados de entrada:

- *pipe_ocup* e *req_msg* estão relacionados ao parâmetro de carga de comunicação, e representam o número de mensagens armazenadas no *pipe* (expresso em um número inteiro) e a notificação do recebimento de uma requisição para uma mensagem que ainda não foi produzida pelo processador (expresso em uma variável Booleana).
- *not_scheduled* (expresso em uma variável Booleana): quando verdadeiro, somente o *microkernel* está sendo executado. Isto significa que não existem tarefas sendo executadas. Quando este sinal for falso, pelo menos uma tarefa está sendo executada. O controlador DFS pode avaliar a utilização da CPU contando o número de ciclos de relógio em que este sinal é verdadeiro em um dado período de avaliação.
- *msg_transfer*, *dma_active* (expressos em variáveis Booleanas), *rem_freq* (expresso em dois números inteiros): este conjunto de sinais é utilizado para aumentar a frequência do processador, se necessário, quando estiver transmitindo um pacote de dados.

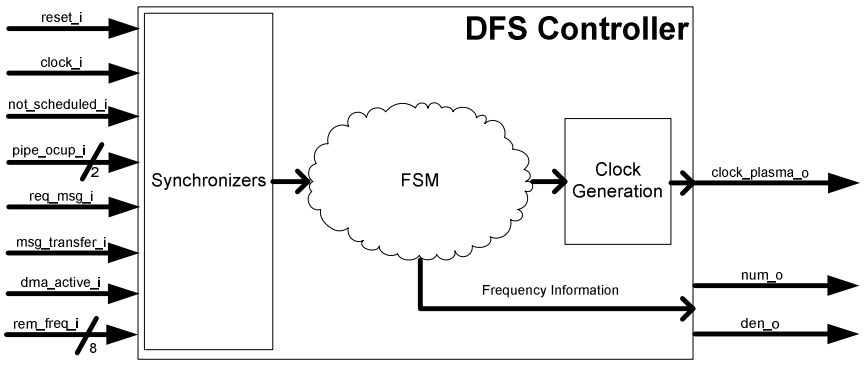


Figura 17 - O controlador DFS e suas interfaces.

Como o controlador DFS trabalha com a frequência de referência e o processador em uma frequência diferente, um mecanismo de sincronização é utilizado na interface entre eles. Esta sincronização é realizada através do bloco de sincronizadores, como mostra a Figura 17. O controlador DFS utiliza o módulo gerador de sinal de relógio descrito na Seção 3.3.1.1, para prover a frequência do processador. A máquina de estados finita, ou FSM (do inglês, *Finit States Machine*), representada na Figura 17

corresponde ao comportamento detalhado na Tabela 3. O trabalho da FSM é escolher uma frequência de operação para o processador baseado na carga de comunicação e na utilização da CPU. Esta frequência é selecionada avaliando-se as seguintes informações:

- **Requisições de mensagens pendentes de outras tarefas.** Esta situação ocorre quando o processador não está produzindo dados para a tarefa consumidora (*req_msg* = 1). Esta informação está relacionada à carga de comunicação do PE.
- **Ocupação do *pipe*.** Se o *pipe* possui uma alta ocupação, o processador está produzindo mensagens em uma taxa maior do que as tarefas consumidoras conseguem absorver. Já a situação contrária significa que existe um *déficit* na produção de mensagens. Limiares parametrizáveis superior e inferior, determinados pelo projetista, definem os estados de alta e baixa ocupação do *pipe*, respectivamente. Os valores de ocupação descritos entre estes dois limiares definem o estado operacional do *pipe*. Esta informação está relacionada à carga de comunicação do PE.
- **Utilização da CPU.** Quando a utilização é baixa, o processador não está executando nenhuma tarefa (*not_scheduled* = 1) ou todas as suas tarefas estão bloqueadas aguardando mensagem(ns) de outras tarefas. Já quando a utilização é alta, as tarefas estão utilizando o processador em sua taxa máxima de processamento. Dois limiares parametrizáveis, determinados pelo projetista, definem os estados de alta, baixa e utilização operacional.

Tabela 3 - Comportamento do controlador DFS (↓/↑ significam um passo de redução/aumento, ↑↑ significa um aumento de dois passos, = significa que a frequência é mantida e - significa uma condição não importante (*don't care*).

Ação na Frequência	Mensagem Pendente	Ocupação Atual do *Pipe*	Ocupação Anterior do *Pipe*	Utilização da CPU
1 - ↓	0	Alta	-	-
2 - ↓	0	Operacional	Baixa	-
3 - ↓	0	Baixa	-	Baixa
4 - =	0	Operacional	Operacional	-
5 - =	0	Baixa	-	Operacional
6 - =	1	-	-	Baixa
7 - ↑↑	1	-	-	Operacional/Alta
8 - ↑	0	Baixa	-	Alta
9 - ↑	0	Operacional	Alta	-

O primeiro parâmetro avaliado é a presença de requisições de mensagens pendentes. A seguir, o controlador DFS avalia a ocupação do *pipe* e a utilização da CPU, respectivamente. Como exibido na Tabela 3, o controlador é guiado principalmente pela carga de comunicação, o que leva as tarefas a produzir e consumir seus dados de forma uniforme, sem afetar o desempenho geral da aplicação. O parâmetro da utilização da CPU previne que o controlador DFS aumente a frequência do PE nas situações em que, a

carga de comunicação poderia levar a um aumento normal na frequência (ações 3 e 6 da Tabela 3). Além disto, os limiares parametrizáveis de utilização de CPU alteram a reatividade do controlador DFS, ou seja o controlador irá reagir mais rápido ou mais lentamente.

A frequência do PE é reduzida em três situações: (*i*) o *pipe* está quase cheio (ação 1 na Tabela 3); (*ii*) a ocupação do *pipe* está aumentando, ou seja, na avaliação anterior o estado era baixo e o atual é operacional (ação 2). Nas situações (*i*) e (*ii*) o processador está produzindo mensagens em uma taxa maior do que estão sendo consumidas; (*iii*) a ocupação do *pipe* está quase vazia e a utilização da CPU é baixa, significando que mesmo com uma baixa frequência os dados no *pipe* estão sendo consumidos (ação 3).

A frequência do PE é aumentada em três situações: (*i*) na existência de mensagens pendentes com utilização de CPU alta ou operacional (ação 7), neste caso o gerador de relógio aumenta a frequência em dois passos; (*ii*) o *pipe* está quase vazio e a utilização de CPU é alta (ação 8); (*iii*) a ocupação do *pipe* está diminuindo, ou seja na avaliação anterior o estado era alta e o atual é operacional (ação 9). Nestas três situações, a frequência é aumentada para evitar que a tarefa remota fique esperando por longos períodos até receber a mensagem solicitada, o que afeta o desempenho da aplicação. Desta forma, na situação (*i*), quando a mensagem já foi solicitada mas não foi produzida ainda, o aumento da frequência é ainda maior para produzir a mensagem solicitada o mais rápido o possível.

O período entre duas avaliações consecutivas também é parametrizável, e definido pelo projetista. Nesta arquitetura, o período de avaliação corresponde a quatro fatias de tempo (do inglês, *time slices*). Uma fatia de tempo é um intervalo de tempo fixo, definido no *microkernel*, no qual o sistema executa uma dada tarefa. Outros períodos de avaliação foram testados, mas valores muito altos (como 8 ou 16) levam a respostas lentas no controlador DFS já que poucas avaliações irão ocorrer em um mesmo período de tempo, e valores muito baixos (como 1 ou 2) tornam o controlador instável realizando muitas trocas de frequência, mesmo quando não são necessárias.

Quando uma avaliação é disparada, o controlador armazena os valores gerados pelo *microkernel* (ocupação atual do *pipe* e utilização da CPU) e avalia os parâmetros monitorados. Os valores de ocupação do *pipe* e utilização da CPU utilizados pelo controlador são uma média dos valores anteriores e atuais. Utilizando esta média, o controlador é capaz de realizar trocas suaves de frequência na presença de grandes variações na ocupação do *pipe* e/ou na utilização da CPU.

Adicionalmente, o controlador DFS implementa um mecanismo de comunicação diferenciado, utilizado para balancear a dissipação de potência e o desempenho do MPSoC. Este mecanismo leva em consideração o fato de que um processador consumidor deve receber seus dados em uma frequência que não é inferior a sua própria frequência. Desta forma, se o processador produtor estiver operando um uma frequência

maior do que a do consumidor, a mensagem poderá ser enviada com uma frequência menor para reduzir a dissipação de potência. Por outro lado, se o produtor está operando com uma frequência inferior do que o consumidor, sua frequência deverá ser temporariamente aumentada a fim de evitar uma degradação no desempenho do consumidor. O conjunto de sinais *msg_transfer*, *dma_active* e *rem_freq* é responsável por realizar estas ações.

O recebimento de um pacote com o serviço REQUEST_MESSAGE habilita o sinal *msg_transfer* do controlador DFS. Este pacote contém, além de outros campos, a frequência do PE que está solicitando dados. Esta frequência está codificada no sinal *rem_freq* (frequência remota) do controlador DFS. De acordo com estes sinais, as seguintes situações podem ocorrer:

- Se o sinal *rem_freq* é menor que a frequência do PE, a frequência do PE não precisa ser alterada. O pacote é enviado através da rede com o sinal *rem_freq* codificado nele.
- Se o sinal *rem_freq* é maior que a frequência do PE e existem dados no *pipe*, a frequência do PE é trocada para o valor de *rem_freq* a fim de realizar a transmissão do pacote de DELIVER_MESSAGE. A frequência do PE irá retornar a seu valor anterior assim que o sinal *dma_active* retornar a zero, sinalizando o final da transmissão do pacote DELIVER_MESSAGE.
- Se o sinal *rem_freq* é maior que a frequência do PE e não existem mensagens no *pipe*, as regras da FSM do controlador DFS serão aplicadas. Neste caso, é mais provável que a FSM aplique a regra 7, aumentando em dois passos a frequência do PE a fim de produzir mais rapidamente os dados para o consumidor.

Todos os demais pacotes de controle (como REQUEST_MESSAGE, REQUEST_TASK, TASK_ALLOCATION, etc) são inseridos na NoC com a máxima frequência de operação, fazendo com que estes pacotes estejam disponíveis nos destinos o mais rápido o possível.

3.3.2 Controle do DFS no Roteador

O projeto de um roteador que poderá utilizar diferentes frequências e comunicar com roteadores vizinhos, os quais utilizam frequências diferentes, exige adaptações nos *buffers* de entrada e saídas de todas as portas. Da mesma maneira que foi demonstrado na Seção 3.3.1.2, todos os *buffers* das portas de entrada (norte, sul, leste e oeste) do roteador foram substituídos por filas bísincronas a fim de sincronizar a comunicação do roteador. A Figura 18 exibe estas filas como FIFO GALS. Além disto, é necessário realizar modificações no árbitro do roteador, que é responsável por solicitar trocas de frequência no roteador.

O controlador DFS do roteador, *DFS controller* na Figura 18, é responsável por definir a frequência de operação do roteador e de colocá-lo em um modo de baixa dissipação de potência no caso de inatividade do roteador. O controlador DFS recebe as informações de frequência e atividade de cada uma das filas de entrada, além de sinais de controle do árbitro (*clk_change, activity, header, selection, ack_header*), e a partir destas informações seleciona a frequência de operação necessária para o funcionamento do roteador (*clk_r*).

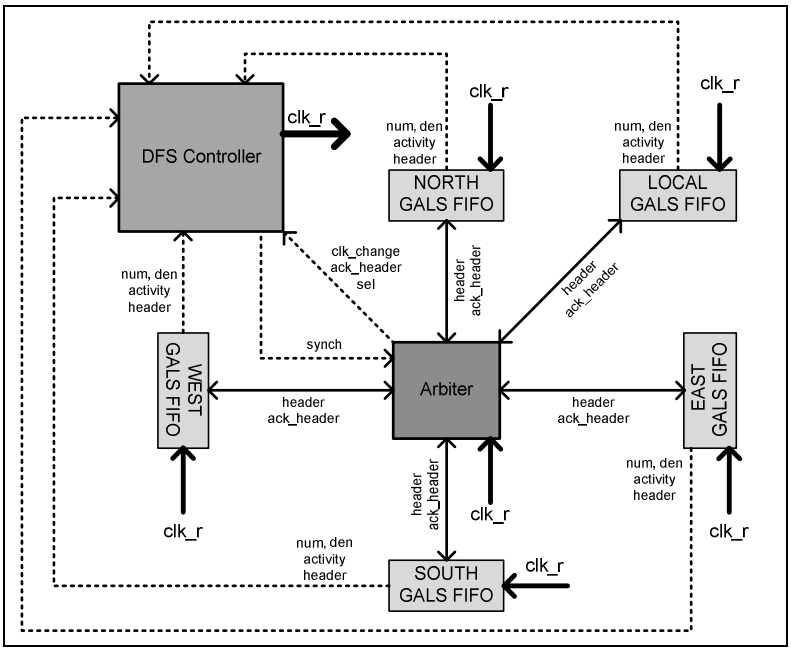


Figura 18 - Estrutura do controlador DFS no roteador.

Os valores de frequência dos PEs são obtidos através dos pacotes enviados na NoC, os quais trazem em seu cabeçalho um campo com o valor de frequência que este pacote deve ser transmitido (Figura 19). Este valor de frequência é representado em um campo de 8 bits, 4 bits para o numerador e 4 bits para o denominador, o que está de acordo com a codificação do gerador de sinal de relógio descrito na Seção 3.3.1.1.

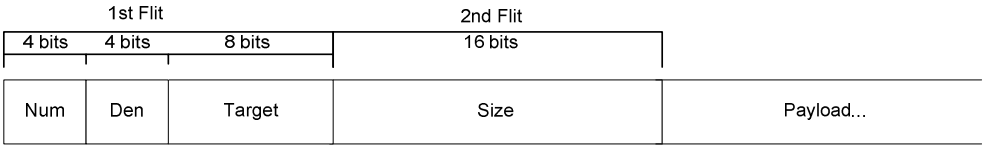


Figura 19 - Exemplo de um pacote da NoC Hermes com informação de frequência.

Quando um pacote é recebido, a FSM da fila de entrada extrai a informação de frequência do pacote e a envia para o controlador DFS (Figura 20(a)). Logo após, a fila de entrada envia uma requisição de arbitragem para o árbitro do roteador e aguarda uma confirmação deste (Figura 20 (b)). Quando o árbitro recebe uma requisição de arbitragem, ele envia uma requisição de troca de frequência para o controlador DFS (*clk_change_i*), e aguarda o sinal de sincronização do relógio enviado pelo controlador DFS (*clk_synch_o*) (Figura 20 (c)). Assim, utilizando as informações vindas de todas as filas de entrada e da

requisição de troca de frequência, o controlador DFS verifica de qual porta partiu a requisição de roteamento, e desativa o sinal de relógio (Figura 20 (d)). Esta desativação é necessária para que o controlador DFS realize a geração do novo sinal de relógio. Após gerar o novo sinal de relógio, o controlador DFS sinaliza a finalização da troca de frequência ao árbitro através do sinal *clk_synch_o* (Figura 20 (e)). O árbitro, então, responde a requisição de arbitragem da fila de entrada (Figura 20 (f)).

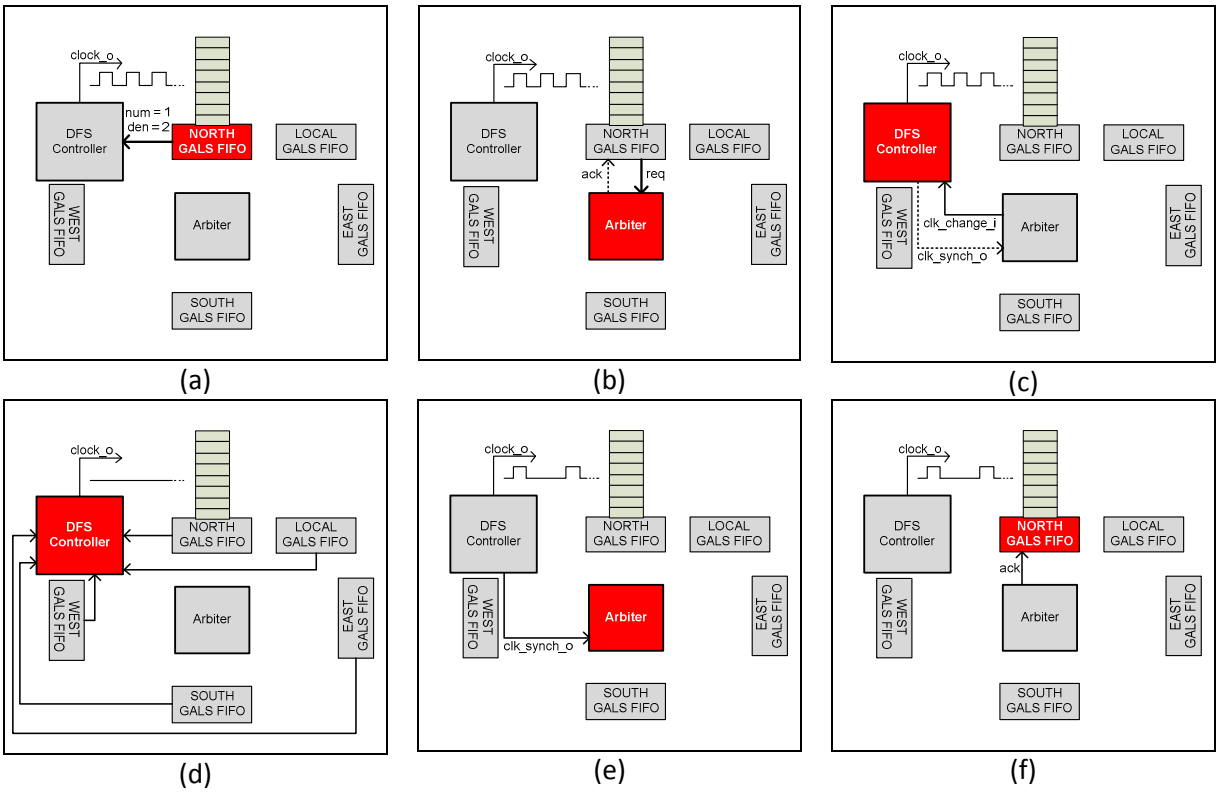


Figura 20 - Mecanismo de troca de frequência no roteador da NoC Hermes.

Quando não existe nenhum tráfego no roteador, o controlador DFS reduz a frequência de operação para a menor possível. Para realizar esta ação, o controlador DFS monitora a atividade do roteador, ou seja, ele monitora a ocupação das filas de entrada e as recepções de pacotes. A inatividade do roteador se caracteriza através do esvaziamento das filas de entrada e do não recebimento de quaisquer pacotes em todas as portas do roteador. Além disto, ao final da transmissão de um pacote, a fila de entrada reinicializa os valores do numerador e denominador para aqueles que correspondem a menor frequência disponível para o roteador. Esta ação é necessária para evitar que o roteador fique operando em altas frequências logo após a transmissão de pacotes de alta frequência.

3.4 Conclusão

Estas modificações na plataforma HeMPS permitem a utilização da técnica de DFS, e são aplicadas as métricas de ocupação do *pipe* e utilização de CPU como guias para a aplicação do controle do DFS. A utilização de tais métricas resulta na redução da

dissipação de potência da HeMPS, porém esta redução é limitada pela dissipação do PE que executa a tarefa com mais poder computacional. Neste PE, a frequência será elevada para evitar atrasos na execução da aplicação.

A presente Tese de Doutorado utiliza esta métrica original do controlador DFS em períodos iniciais da execução de uma aplicação. Desta forma, após um período de estabilização, se a aplicação não estiver atendendo os requisitos de QoS definidos (como vazão, por exemplo) troca-se a política de controle de DFS por outra que atenda a diferentes métricas para satisfazer os requisitos de QoS da aplicação.

4 MONITORAMENTO EM MPSOCS

Este Capítulo aborda diversos tipos de monitoramento utilizados em MPSoCs, utilizando como referência principal o artigo [KOR12]. Além disso, são apresentados os tipos de monitoramento e métricas utilizadas na plataforma HeMPS. O Capítulo termina com a seleção do tipo de monitoramento e métricas que serão utilizadas no Capítulo 7

4.1 O Significado do Monitoramento na Era dos Circuitos de Múltiplos Núcleos

Os circuitos com múltiplos núcleos estão se tornando mais diversificados tanto no nível de arquitetura quanto no nível de programação. A variabilidade nos requisitos das aplicações em tempo de execução abre um leque de possibilidades onde o desempenho, previsibilidade, diagnóstico de falhas e medidas de recuperação combinados ao baixo consumo de energia, exigem novas metodologias de projeto de circuitos integrados.

Para atender a estas necessidades, o monitoramento, o controle dos estados e a dinâmica de todo o sistema estão se tornando cada vez mais importantes. O monitoramento das aplicações está ligado à tentativa dos projetistas de entender e otimizar o comportamento destas, além de permitir a exploração de características intrínsecas ao código objeto. Ao mesmo tempo, o monitoramento dos SoCs, traz outros benefícios para os sistemas como, a adaptabilidade de recursos para atender a carga de trabalho dinâmica; melhorias na eficiência do consumo de energia e desempenho desejado respeitando limites de dissipação de potência ou térmicos.

Ambientes com limitações da dissipação de potência mudaram a abordagem do projeto de SoCs. Já que os projetos com o maior desempenho possível dissipam muita potência, o desempenho está se tornando uma preocupação secundária para a maior parte dos projetistas [FLY05]. Além disto, como o tamanho e complexidade dos sistemas continuam a aumentar, o monitoramento do sistema em tempo de execução oferece um melhor desempenho, escalabilidade e flexibilidade para o projeto de circuitos com múltiplos núcleos.

4.2 Tipos de Monitores

Os monitores são os principais atores em um mecanismo de monitoramento. Eles realizam a coleta dos dados monitorados e os entregam para outro módulo que fará o processamento destes. Em alguns casos, os monitores coletam, processam os dados e os enviam para os sistemas de controle que os auxiliarão na tomada de decisão. Os monitores realizam a coleta de informações de diversas formas, dentre as principais pode-se citar: os circuitos de monitoramento direto e indireto, o monitoramento via software ou via hardware e o monitoramento híbrido.

4.2.1 Circuitos de Monitoramento Direto

O monitoramento através de vários sensores de propósito geral integrados no chip vem se desenvolvendo de forma lenta e gradual nos últimos vinte anos, mas atualmente recebeu um destaque maior já que os projetistas procuram novas maneiras de se construir circuitos analógicos ou digitais utilizando os processos CMOS mais modernos. Sensores e circuitos de monitoramento estão avançando para lidar com desafios tradicionais nos circuitos VLSI, tais como: integridade e qualidade de sinal, ruído na fonte de tensão; além de tratar dos desafios referentes aos novos processos de fabricação.

Atualmente, os efeitos da variabilidade no processo de fabricação dos circuitos integrados estão se tornando mais importantes, fazendo com que a sensibilidade de um dispositivo às condições de operação aumentem no decorrer do ciclo de vida. Desta forma, centenas, ou talvez milhares, de sensores são utilizados para estimar as fronteiras de degradação de desempenho ou características físicas de um circuito integrado.

A Figura 21(i) exibe um sensor térmico digital, ou DTS (do inglês, *Digital Thermal Sensor*) que utiliza um diodo térmico como elemento sensor, e a Figura 21(ii) exibe um sensor de temperatura CMOS, totalmente digital, baseado em atraso nos pulsos de relógio. Este último converte atrasos sofridos por uma série de inversores em um valor de temperatura.

Diferentes circuitos, de diferentes tipos, são desenvolvidos para medir diferentes variáveis tais como: temperatura, ruído na fonte de tensão, detectores de ruído, sensores de *jitter*, sensores de variação de processo (incluindo envelhecimento, confiabilidade, NTBI [*Negative Bias Temperature Instability*], etc...), sensores de transição e sensores de atrasos.

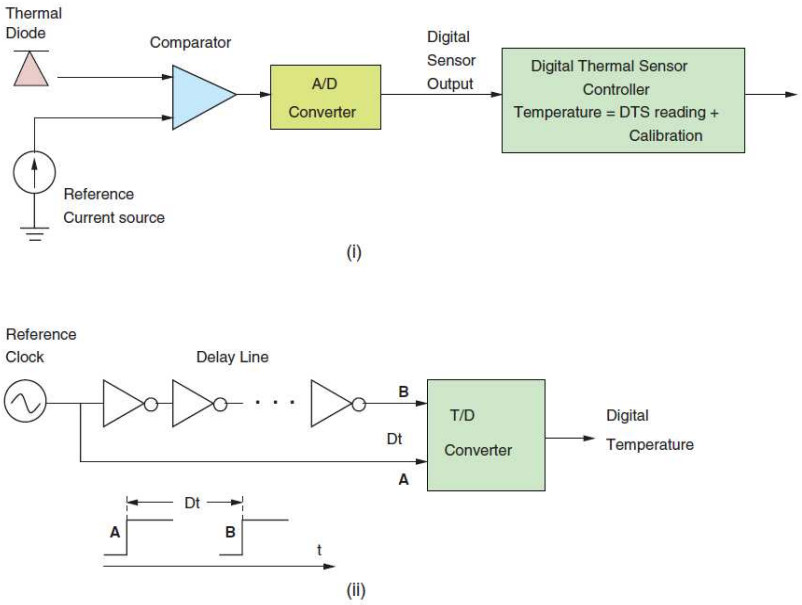


Figura 21 - Sensores digitais de temperatura construídos com (i) um diodo térmico, conversor analógico-digital (A/D) e calibração; e (ii) uma ponte de inversores para gerar um atraso e um temporizador digital (T/D) [KOR12].

4.2.2 Circuitos de Monitoramento Indireto

Monitores de desempenho são as estruturas de monitoramento indiretas mais comuns para se inferir a dissipação de potência ou consumo de energia de um componente do circuito, já que o desempenho varia de acordo com a dissipação de potência. O acompanhamento de estatísticas sobre os dados arquiteturais, tais como ciclos por instrução (CPI, do inglês *Cycles Per Instruction*), *cache miss*, erros de referência nos acessos à memória e utilização da largura de banda na comunicação intra-chip, aliadas a métricas e padrões do nível da aplicação, podem ser utilizados para ajustar o desempenho do sistema, o compromisso entre desempenho e consumo de energia, o consumo de energia ou os requisitos de temperatura do componente deste circuito.

O monitoramento indireto normalmente envolve a construção de um modelo da dissipação de potência de um processador [ISC03], de um GPU (do inglês, *Graphical Processor Unit*) [HON10] ou de uma NoC [GUI08a]. Este tipo de monitoramento também é utilizado através do emprego de funções preditivas que modelam o comportamento específico de um sistema, e como este comportamento pode ser utilizado no controle indireto do desempenho do sistema ou na sua dissipação de potência. Este tipo de monitoramento foi utilizado por Wang [WAN11] na forma de um módulo de predição simples, baseado no histórico de utilização de recursos do GPU, para determinar os requisitos de recursos utilizados por uma aplicação de sombreamento, ou *shader*, em computação gráfica. Neste exemplo o *shader* deve calcular o próximo quadro, e utiliza este módulo para prever a alocação de recursos e aplicar mecanismos de redução de energia, ou *power-gating*, a fim de reduzir a dissipação de potência de fuga em um GPU.

4.2.3 Monitoramento via Software

O conceito de monitoramento via software utiliza as metodologias de desenvolvimento de monitores (instrumentação via hardware ou software), para resolver problemas do software ou para avaliar o comportamento da aplicação a fim de otimizar o desempenho desta. O monitoramento via software determina quais componentes de software (construtores, objetos, estruturas de dados, ferramentas e técnicas) serão utilizados para analisar o comportamento do sistema, a fim de avaliar o desempenho do software ou a eficiência do processador.

Os monitores via software utilizam técnicas de instrumentação de software [LUK05], para avaliar a aplicação original utilizando códigos adicionais que fazem o monitoramento. Infelizmente, os principais problemas na utilização de monitoramento via software são, a velocidade na avaliação e a ineficiência deste tipo de monitoramento quando aplicado a ambientes multitarefa.

4.2.4 Monitoramento via Hardware

O conceito de monitoramento via hardware utiliza diversos tipos de estruturas de hardware, tais como: contadores, sensores de temperatura, medidores de tensão, etc., para coletar informações da execução do hardware avaliado. Estas informações são repassadas para outros módulos que podem atuar no controle das variáveis monitoradas. A quantidade de monitores inseridos nos projetos de SoCs vem aumentando cada vez mais. Além disto, as técnicas de gerenciamento dinâmico de dissipação de potência ou de temperatura surgem como solução para se evitar o aparecimento de pontos quentes, *hot-spots*, temporais ou espaciais, atendendo os requisitos de desempenho dos SoCs.

Desta forma, as arquiteturas adaptáveis a quaisquer tipos de variações tendem a estabelecer uma nova categoria de SoCs. Tais arquiteturas dependem fortemente das informações obtidas dos circuitos de monitoramento inseridos do sistema. Sensores e controladores em hardware, que monitoram desempenho, degradação de tensão, temperatura ou dissipação de potência, na medida em que o circuito envelhece estão se tornando um subsistema fundamental para os SoCs [SYL06].

4.2.5 Monitoramento Híbrido

As melhorias de hardware no processamento dos monitores trabalham sinergicamente com ferramentas de software desenvolvidas para ajudar na coleta e filtragem de informações nos monitores, e para prover interfaces padrão ou métodos de amostragem mais precisos. Através desta ferramenta, os usuários podem extrapolar medidas obtidas de amostras colhidas em pontos predeterminados na aplicação ou durante interrupções disparadas por condições estabelecidas pelo usuário como, por exemplo, *N* falhas de acesso à memória cache (cache miss). Geralmente, o uso deste tipo de monitoramento introduz erros de forma inversamente proporcional à frequência de amostragem.

O monitoramento híbrido combina as vantagens do monitoramento em hardware, como a não intrusão no processo e a latência mínima, com a flexibilidade do monitoramento em software. Desta forma, se torna mais fácil relacionar os eventos identificados através dos dados dos monitores, com o sistema em monitoramento. Em contrapartida, o processamento da parte em software do monitoramento provoca um acréscimo na carga de trabalho do processador, o que ocasiona um impacto indesejável no comportamento do sistema monitorado. Estes acréscimos de processamento que são provocados pela execução da rotina do monitor podem ser controlados através do uso de diferentes tipos de monitores: de amostragem, de acompanhamento ou estendidos.

Estes tipos de monitoramento variam com o detalhamento das informações que são coletadas pelos monitores, o que leva a um compromisso entre desempenho ou latência e precisão dos monitores. Os monitores de acompanhamento produzem registros de eventos com informações de tempo associadas além das amostras mais simples, estes

registros podem ser utilizados imediatamente pelo software de controle do sistema ou armazenados para análises futuras. Os monitores estendidos podem realizar um processamento básico das informações coletadas, como filtros ou combinações, antes de disponibilizá-las ao software. Os monitores por amostragem são aqueles que produzem o menor impacto na latência e necessitam de menos espaço de armazenamento que os demais tipos. Entretanto, o uso combinado dos três tipos permite ao usuário fazer um balanceamento entre um baixo impacto na latência contra requisitos de precisão dos monitores.

4.3 Objetivos dos Monitores

Os monitores podem ser classificados de acordo com as funções ou objetivos para os quais foram projetados, como por exemplo: depuração, desempenho, qualidade de serviço (QoS), utilização de recursos, dissipação de potência, consumo de energia, temperatura, tolerância a falhas e outras aplicações específicas como reconfiguração dinâmica ou assistência na migração de tarefas.

4.3.1 Monitores para Depuração

Uma metodologia de depuração é observar o componente, seja ele hardware ou software, para depurar seu comportamento no ambiente em que será executado, ou controlar sua execução (parar, passo-a-passo, etc.) a fim de localizar o motivo de comportamentos indesejáveis. Monitoramento e depuração computacional são duas áreas maduras nas quais diversas ferramentas e técnicas foram desenvolvidas, especialmente para processadores simples (ou de um único núcleo).

Na indústria, várias soluções são empregadas para esta finalidade; A empresa de FPGAs Xilinx desenvolveu o Chipscope [XIL13] para permitir a depuração intra-chip e visualização do sistema que está sendo projetado, em tempo real para plataformas reconfiguráveis. Paradigmas de processadores embarcados tais como o ARM CoreSight [ARM13] e a tecnologia de instrumentação intra-chip do MIPS [LEA97] apresentam módulos de monitoramento não intrusivo integrados nos núcleos do processador, permitindo uma análise de sua lógica através dos barramentos de comunicação. Além disto, a Organização IEEE para a Padronização e Tecnologias da Indústria propôs um padrão global de interface para depuração em processadores embarcados, denominado de Nexus 5001 [IEE09]. O padrão Nexus define quatro classes de operação: Classes 1, 2, 3 e 4. Quanto maior a classe de operação, maior o suporte para operações de depuração mais complexas ao custo de mais recursos do chip.

4.3.2 Monitores de Desempenho

A mais conhecida estrutura de monitoramento é o contador de desempenho, que é, geralmente, uma classe de dispositivos inseridos na maioria dos processadores

modernos. Estes contadores podem ser programados para contar o número de vezes (fixa ou dinâmica) que um dado evento ocorre no processador. Um evento bem comum que é monitorado é o número de instruções executadas pelo processador desde que a contagem foi ligada. Através da exposição destas contagens ao software, estas podem ser utilizadas para procurar e remover ineficiências do software ou gargalos arquiteturais.

O monitor de desempenho observa o comportamento do sistema, coletando informações estatísticas de vazão e latência as quais auxiliam o sistema em execução a corrigir possíveis violações de limiares previamente estabelecidos. Monitores de desempenho em hardware são comumente inseridos nos processadores, enquanto os monitores em software são comumente disponibilizados como bibliotecas nativas. Um monitor de desempenho é uma variável escalar que muda em função do tempo, enquanto que os contadores de desempenho são um tipo especial de monitor de desempenho (utilizados para contar a quantidade de certos eventos). O valor de um contador de desempenho reflete uma contagem, geralmente uma contagem de eventos.

4.3.3 Monitores para Qualidade de Serviço

O projeto dos SoCs modernos envolve a integração de diferentes componentes de hardware, ou IPs (do inglês, *Intelectual Property*), tais como processadores, aceleradores gráficos, decodificadores de áudio/vídeo, memórias intra-chip, etc. que cooperam para atender os requisitos gerais de desempenho de diferentes aplicações. Geralmente, cada um destes IPs possui requisitos de vazão e latência muito diferentes, e estes requisitos devem ser garantidos para permitir seu correto funcionamento.

O monitoramento dos subsistemas vem sendo cada vez mais utilizado nos SoCs multi-núcleos para realizar a gerência de alocação de recursos destes. Um dos principais objetivos dos monitores para qualidade de serviço é prover um conjunto razoável de serviços para garantir que os requisitos de QoS sejam atendidos. Classes de serviços, canais virtuais e controladores dinâmicos de largura de banda são algumas soluções desenvolvidas para prover uma qualidade de serviço eficiente em SoCs.

A Figura 22 exibe um esquema de monitoramento de QoS, onde os monitores estão nos roteadores da NoC e monitoram o congestionamento do roteador a partir da estimativa da utilização do enlace ou contando quantos pacotes estão armazenados nos *buffers*. Quando alguns limites (definidos pelo projetista) são ultrapassados, uma notificação é enviada para um controlador ou um módulo de ajuste de tráfego para que a taxa de produção de pacotes seja ajustada [MAR07].

Em outro esquema de monitoramento de QoS, algoritmos de roteamento adaptativos são utilizados por NoCs adaptativas onde um subconjunto de tarefas e seu mapeamento pode ser alterado em tempo de execução [TED09]. Os monitores são responsáveis por fazer um levantamento espacial das tarefas em execução, isto é, realocar as tarefas das aplicações em diferentes PEs, além da introdução de novas tarefas e a reconfiguração do

algoritmo de roteamento.

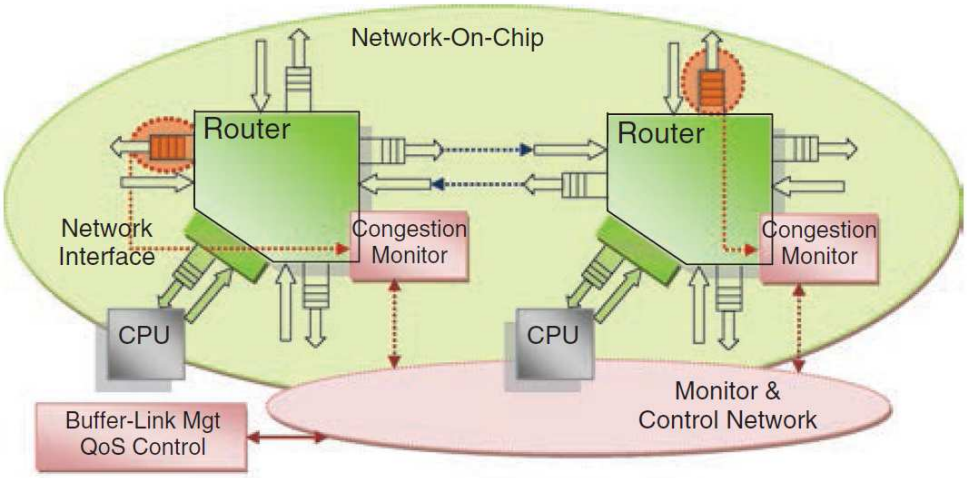


Figura 22 - Abordagem de controle do QoS distribuído em uma NoC, através do monitoramento dos buffers ou estimativa da utilização dos enlaces.

O monitoramento para manter o QoS pode ser inerente a alguns domínios de aplicação. Os servidores que possuem um serviço de transmissão de vídeo exploram a adaptatividade inerente as aplicações de vídeo para executar ajustes controlados na qualidade perceptual de uma transmissão de vídeo MPEG, estes ajustes são utilizados para compensar as flutuações no QoS entregue pela camada infra-estrutural.

4.3.4 Monitores para Dissipação de Potência, Consumo de Energia e Temperatura

O aumento da dissipação de potência na medida em que se diminuem as tecnologias de transistores pode levar ao aparecimento de pontos quentes, *hot-spots*, de temperatura e a grandes variações de temperatura no circuito integrado, estes comportamentos devem ser evitados ou pelo menos reduzidos. Os mecanismos de monitoramento possuem um papel fundamental ao se construir arquiteturas adaptativas que, tentam atingir o melhor desempenho teórico com uma configuração de DVFS termicamente segura aplicada a cargas de trabalhos específicas ou dinâmicas.

Estratégias de projeto dos monitores, intrusiva ou não-intrusiva, envolve a exploração dos contadores de desempenho que existem na maioria dos processadores e que podem ser utilizados para monitorar dados sobre a atividade (contadores de acesso) das unidades funcionais do sistema. A interpretação dos valores destes contadores pode ser utilizada para monitorar a temperatura do processador [ISC03], ou o seu consumo de energia [WEI04]. Em outra abordagem, monitores são utilizados na teoria de controle do circuito com realimentação que, controla o desempenho do sistema utilizando previsões de temperatura, descritas em tabelas, e leituras da carga de trabalho dinâmica coletadas nos monitores.

Além do uso dos monitores de desempenho e de controladores com realimentação para realizar a gerência térmica ou do consumo de energia do sistema, outros tipos de

monitores são integrados nos SoCs modernos tais como: térmicos, de atraso ou de deterioração. A maior parte dos monitores térmicos possuem sensores construídos com osciladores em anel ou circuitos com diodos. Matrizes de sensores térmicos ou de tensão são conectados a controladores intra-chip para registrar as condições do ambiente em que o circuito está presente. As informações dos monitores podem ser utilizadas para controlar a alocação de frequência, tensão e largura de banda do sistema [MCG06].

4.3.5 Monitores para Tolerância a Falhas e Confiabilidade

Para se garantir sistemas confiáveis, devem-se utilizar diversos níveis de abstração desde o nível de circuito/arquitetura até o nível de algoritmo/aplicação. Desta forma, diversas metodologias de monitoramento foram desenvolvidas para prover aos sistemas auto-diagnósticos, adaptabilidade e tolerância a falhas. A maior parte destes mecanismos assume uma confiabilidade assimétrica, que é uma visão conservativa onde alguns componentes são praticamente livres de falhas, logo tais mecanismos devem proteger componentes individuais do sistema tais como: barramentos, NoC, memórias, etc. contra erros transientes ou permanentes.

As unidades de monitoramento ou de diagnóstico possuem poucos requisitos de desempenho, desta forma podem operar em baixa tensão e frequência; além de serem construídas com redundâncias, tornando-se praticamente imunes a falhas [SYL06]. Entretanto, esquemas de monitoramento centralizados representam um risco, pois possuem uma vulnerabilidade de ponto único de falha.

Monitores em tempo de execução são utilizados para extrair métricas do circuito a fim de calcular com precisão modelos de envelhecimento, realizar uma análise de confiabilidade [ATI08] ou prover características de confiabilidade dos componentes do sistema. Novas arquiteturas serão construídas a partir de componentes com características de confiabilidade variáveis, tais como núcleos de processamento com diferentes precisões aritméticas ou memórias com diferentes garantias de confiança. As aplicações poderão fazer um compromisso entre alta confiabilidade em hardware por alto desempenho, através da troca do modo de operação em sistemas de múltiplos núcleos com diversos modos de operação devido à reconfiguração dinâmica das redundâncias do sistema [WEL09].

No contexto das NoCs, o monitoramento pode ser aplicado para realizar a observação dos eventos de comunicação, mesmo no nível de transação. Da mesma forma que as redes tradicionais, diversas técnicas foram desenvolvidas a fim de prover a comunicação intra-chip tolerante a falhas. Algumas destas utilizam códigos de detecção/correção de erros (como CRC ou Hamming), retransmissão em comunicações ponto-a-ponto [MUR05], suportam comunicação com múltiplos caminhos, roteamento adaptativo e reconfiguração. Os monitores, que são inseridos nas interfaces de rede ou nos roteadores, são responsáveis por indicar a presença de erros, após verificar cada

pacote transmitido.

4.3.6 Monitores para Segurança

A segurança é considerada como outro domínio nos requisitos do sistema, tais como desempenho, área e dissipação de potência. Este domínio inclui subsistemas de monitoramento, que são os componentes principais nas arquiteturas de segurança. Tais subsistemas são utilizados para verificar se o processador realmente executou as operações as quais lhe foram atribuídas, as detecções de anomalias e de intrusões são comumente realizadas através da comparação do comportamento monitorado contra modelos pré-definidos. Os subsistemas de monitoramento operam em paralelo ao processador, e utilizam valores *hash* por bloco ou controle de fluxo de dados para detectar desvios na execução de um programa [ARO05].

As arquiteturas dos monitores voltados à segurança são, usualmente, desenvolvidos utilizando conceitos de resistência à falsificação de dados ou criptografia. Ao se construir um ambiente seguro existem diversas escolhas de variáveis que podem ser monitoradas. Um caso particular é o monitoramento de padrões intuitivos, que envolve monitorar o fluxo de controle, endereçamento e informações de *load/store* dos processadores. Estes padrões consomem espaço na memória, desta forma esquemas de *hash* podem ser utilizados. Diversas informações como, endereço da instrução e a palavra da instrução, podem ser comparadas a pequenos valores *hash* e assim verificar se estão corretas.

4.3.7 Monitores para Aplicações Específicas

O monitoramento para aplicações específicas envolve mecanismos desenvolvidos para atender diversos objetivos discutidos anteriormente, porém são personalizados a fim de considerar comportamentos específicos de algumas aplicações. Nesta categoria estão localizados os esquemas de monitoramento para a exploração, em tempo de execução, dos melhores valores de desempenho/consumo de energia que podem ser encontrados para aplicações com perfis personalizados de utilização de memória.

Em outra perspectiva, um monitoramento é considerado para aplicações específicas quando personalizado para o emprego em um sistema computacional específico. Ao invés de utilizar contadores de desempenho e sensores, alguns monitores utilizados em tempo de execução adotam circuitos complexos. Gordon-Ross et. al. [GOR07] descrevem um módulo de hardware para o ajuste da *cache* que, de forma não intrusiva, monitora os padrões de acesso à memória da aplicação e prevê analiticamente a melhor configuração de *cache* para estes padrões. Desta forma, se a melhor configuração de *cache* prevista for diferente daquela em uso, um módulo de ajuste da *cache* a reconfigura diretamente para a melhor configuração prevista.

4.4 O Monitoramento na HeMPS

Dois esquemas de monitoramento foram inseridos na HeMPS, um para auxiliar o controle da dissipação de potência do sistema [ROS12b] e outro para controlar o desempenho das aplicações que executam no sistema [MAD13]. Estes dois monitoramentos foram inseridos separadamente na plataforma HeMPS, cada um utilizado no escopo de uma dissertação de mestrado. A seguir estes dois esquemas de monitoramento são classificados a partir das definições estudadas neste Capítulo.

O monitoramento descrito em [MAD13] e apresentado na Seção 2.1.3, é realizado via software, com seu monitor inserido no *microkernel* do PE. Somente um monitor de desempenho foi utilizado, que avalia se a aplicação está cumprindo requisitos de latência previamente alocados. Diferentemente de outros monitores de desempenho, este é ativado através de uma chamada de sistema e cada tarefa pode ativar ou não seu monitoramento.

O monitoramento descrito em [ROS12b] e apresentado na Capítulo 3, também é realizado via software, e os monitores são inseridos no SO, ou *microkernel*, do PE. Foram desenvolvidos dois monitores de desempenho, um avalia a utilização da CPU, e o outro registra um histórico da ocupação das filas de entrada da interface de rede. Ambos os monitores compartilham a mesma janela de avaliação, que é parametrizável.

4.5 Conclusão

Neste Capítulo foram estudados os principais conceitos sobre monitoramento em MPSoCs, descrevendo os tipos de monitoramento e a classificação dos monitores quanto a seu objetivos. Mencionou-se também duas estruturas de monitoramento já utilizadas na plataforma HeMPS. Estes esquemas utilizam somente monitoramento via software, com monitores de desempenho.

O trabalho descrito na presente Tese de Doutorado utiliza as estruturas destes dois tipos de monitoramento inseridos na HeMPS, porém modificando sua estrutura para cumprir com o objetivo de controle de QoS. Desta forma, o esquema de monitoramento apresentado no Capítulo 7 é do tipo híbrido, utilizando monitores em hardware e software. Os monitores utilizados são de desempenho (utilizando os monitores dos trabalhos anteriores) e de QoS (contribuição deste trabalho), e possuem janela de avaliação parametrizável e inserida através de uma chamada de sistema.

5 MECANISMOS DE CONTROLE DA QUALIDADE DE SERVIÇO NA HEMPS

Este Capítulo apresenta mecanismos para o controle de QoS inseridos na plataforma HeMPS, e realiza uma comparação entre estes mecanismos e o controle de QoS inserido no escopo desta Tese de Doutorado.

Em [TED10], os Autores não modificam a plataforma HeMPS diretamente mas sim a NoC. Em [CAR11], os Autores propõem diversos serviços para garantir QoS na plataforma HeMPS, e inseriram serviços de conexão baseados em prioridades de conexão. Já em [MAD13], os Autores combinam a técnica de migração de tarefas com um esquema de monitoramento, permitindo um controle mais efetivo do QoS na HeMPS.

5.1 Tedesco et. al. [TED10]

O foco principal deste trabalho não é a plataforma HeMPS, e sim a NoC Hermes que a compõe. Entretanto, é relevante avaliar outros trabalhos relacionados ao controle de QoS na NoC Hermes além da técnica de DFS utilizada no mecanismo desenvolvido nesta Tese de Doutorado.

A literatura na área de NoCs apresenta diversas propostas de adaptação da rede para atender aos requisitos de aplicações. Estas soluções normalmente objetivam o atendimento de requisitos de níveis de QoS nas comunicações entre os núcleos (latência mínima, *jitter*, *throughput* e *deadlines*), mínimo acréscimo de área de silício e energia consumida pela rede. Segundo [TED10], a adaptabilidade em NoCs pode ocorrer em dois níveis: sistema e arquitetura. A adaptação em nível de sistema ocorre em tempo de execução, onde parâmetros funcionais são alterados de acordo com as características de tráfego da aplicação em curso, que pode por sua vez apresentar características de dinamicidade. A adaptação em nível de arquitetura ocorre em tempo de projeto, onde parâmetros estruturais são dimensionados de acordo com a descrição da aplicação-alvo.

Neste contexto de adaptabilidade, o foco do mecanismo de controle de QoS proposto em [TED10] reside na otimização do desempenho em fluxos de dados entre pares de núcleos que se comunicam através de uma NoC, com requisitos de qualidade de serviço. Tal otimização utiliza um sistema de monitoração de tráfego em roteadores pertencentes ao caminho de um fluxo com requisitos de QoS, que oferece suporte a um algoritmo de roteamento adaptativo na origem. O objetivo geral é tratar em tempo de execução a ocorrência de congestionamento, gerado por comunicações que tipicamente ocorrem em MPSoCs. Os Autores também abordam a configuração de *buffers* de interfaces de redes externas, onde é realizado o desacoplamento comunicação-computação.

Para atender a adaptabilidade, tanto em tempo de projeto quanto em tempo de execução, os Autores utilizam um sistema com monitoração de tráfego, roteamento

adaptativo na origem e *buffers* de desacoplamento (*D-buffers*) dimensionados de acordo com padrões de produção de dados pela rede e consumo pela aplicação. A Figura 23 apresenta uma visão geral do trabalho desenvolvido em [TED10], sendo ilustradas as etapas no projeto de NoCs que foram abordadas: geração de tráfego, definição de caminho, desacoplamento comunicação-computação, coleta e propagação de informações de congestionamento, detecção de congestionamento, envio de pacote com informações de congestionamento e definição de nova rota.

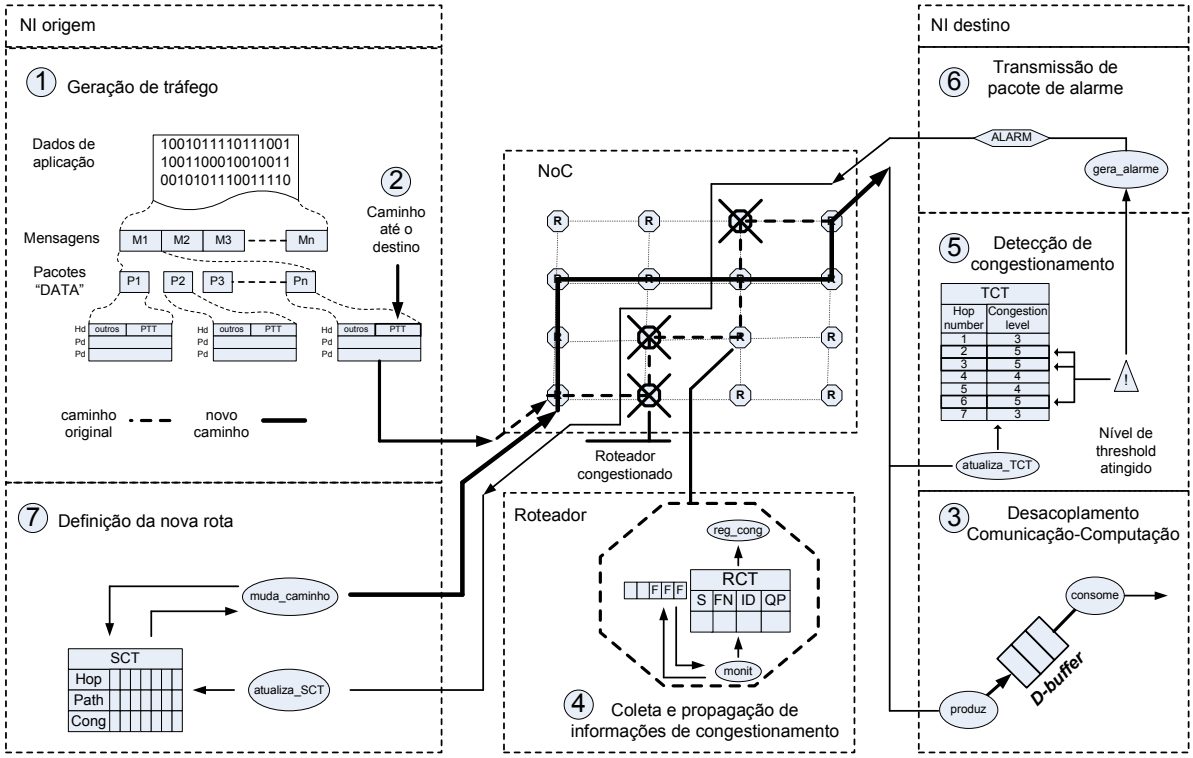


Figura 23 - Visão geral do mecanismo de controle de QoS na NoC Hermes proposto em [TED10].

As etapas 1 e 2 mostram a geração de tráfego de um IP origem para um IP destino. O objetivo é estruturar os dados da aplicação no formato de pacotes, de maneira a viabilizar a sua transmissão no meio de comunicação. Inicialmente os dados da aplicação são quebrados em mensagens, sendo posteriormente formatados em pacotes. Todo pacote é formado por *flits* de *header* e *payload*. O *header* (Hd na Figura 23) contém informações para execução do protocolo de comunicação, além de indicar o destino dos dados através do campo PTT (do inglês, *path to target*). De fato, o campo PTT define a distribuição espacial do tráfego, onde são especificadas as portas de saída em cada roteador pertencente ao caminho do fluxo. O *payload* (Pd) contém informações específicas da aplicação ao qual o fluxo pertence. A taxa de injeção de dados é outro importante parâmetro na etapa de geração de tráfego, onde se especifica a frequência na qual se inserem dados na rede. Taxas típicas de aplicações foram utilizadas como referência, trazendo aos experimentos maior possibilidade de se avaliar a utilização de recursos em cenários cujo comportamento seja similar a situações reais. Quatro classes de pacotes são transmitidas sobre a arquitetura-alvo proposta. Pacotes do tipo SREQ iniciam uma sessão QoS, configurando os roteadores intermediários para o

armazenamento de informações sobre o fluxo. Pacotes do tipo DATA são aqueles que carregam informações da aplicação. Ainda, a funcionalidade deste pacote é estendida para conter dados gerados pelo processo de monitoração da rede. Pacotes do tipo ALARM informam quais roteadores estão congestionados, além de habilitarem a transmissão de mensagens. Pacotes do tipo CLEAN limpam registros de fluxos QoS em roteadores após a transmissão de mensagens ou devido a falhas de inicialização de caminhos.

Em 3 é ilustrado o processo de desacoplamento comunicação-computação, em que os pacotes que chegam à taxa de transmissão da rede são armazenados em um *buffer* (tratado na Figura 23 como *D-buffer*) para serem posteriormente consumidos pelo núcleo IP receptor na taxa da aplicação. Portanto, o *D-buffer* tem como função adaptar os eventos que ocorreram na rede à demanda de processamento pelo núcleo IP. O empacotamento de dados na origem, o processamento de roteamento e a concorrência de outros fluxos que utilizam a NoC provocam a descaracterização do fluxo original dos dados. Os Autores mostram como é dimensionado o *D-buffer* pertencente às NIs de módulos IP, de modo que sejam evitadas ocorrências de falta de espaço para inserção de novos dados que chegam, assim como ocorrências de esvaziamento precoce do *D-buffer*, o que faz como que a aplicação não consiga consumir dados no momento em que desejar.

A parte 4 da Figura 23 ilustra o processo monitoração de tráfego na rede. Nesta etapa é observado o estado de utilização de recursos da rede. Ocupação de *buffers* e taxa de encaminhamento de pacotes são algumas das métricas que podem ser observadas. A Tabela RCT (do inglês, *Router Congestion Table*) armazena informações monitoradas para cada fluxo que utiliza o roteador. A mesma tabela é utilizada no momento em que a informação deve ser enviada ao destino, informando o seu grau de congestionamento.

A etapa 5 ilustra a detecção de um valor de *threshold* (limite) atingido, indicando a necessidade de haver mudança na rota dos pacotes das próximas mensagens, visto que congestionamento em um dado ponto do caminho foi detectado. A tabela TCT (do inglês, *Target Congestion Table*) armazena para cada fluxo QoS o nível de congestionamento encontrado em cada roteador de seu caminho. Em 6 é ilustrada a transmissão de um pacote ALARM, o qual notifica congestionamento, informando quais roteadores estão com dificuldades para encaminhamento de pacotes.

Em 7 é mostrado no IP origem do tráfego a redefinição da rota de um pacote gerado na etapa 1. Tal redefinição toma como referência o dado contido no pacote ALARM recebido. Uma vez que os dados são extraídos do pacote, eles são armazenados na tabela SCT (do inglês, *Source Congestion Table*), a qual é consultada pela função que gera o novo caminho. Assim sendo, o próximo pacote é transmitido em um novo caminho, que evita *hot-spots* gerados pelos tráfegos concorrentes.

5.1.1 Comparativo

O trabalho apresentado nesta Seção exibe diversas modificações na NoC Hermes para atender a requisitos de QoS das aplicações de um MPSoC. Estas modificações envolvem a utilização de algoritmos de roteamento adaptativos e dimensionamento dos *buffers* das NIs. Entretanto, estas modificações não foram inseridas na plataforma HeMPS.

As modificações na NoC Hermes para a utilização da técnica de DFS, descritas na Seção 3.3.2 e desenvolvidas em [ROS12b], foram utilizadas no escopo desta Tese de Doutorado a fim de realizar o controle de QoS das tarefas de uma aplicação no nível da NoC. Ao contrário do trabalho apresentado nesta Seção, a técnica utilizada nesta Tese de Doutorado ajusta a frequência de operação dos roteadores para cada tipo de pacote por eles roteados. Desta forma, não são necessárias mudanças na NI ou na API de comunicação das tarefas para realizar este ajuste.

5.2 Carara et. al. [CAR11]

O Autores em [CAR11] propõem serviços para o controle de QoS na plataforma HeMPS: (*i*) serviços de comunicação baseados em prioridades e conexões; e (*ii*) serviço de comunicação com roteamento diferenciado.

5.2.1 Serviços De Comunicação Baseados em Prioridades E Conexões

Originalmente a NoC Hermes emprega uma topologia malha bidimensional construída a partir de roteadores com até cinco portas bidirecionais. Os roteadores localizados nas bordas da malha têm menos portas que aqueles localizados entre as bordas.

A principal modificação realizada pelos Autores na arquitetura do roteador para suportar um serviço de comunicação baseado em prioridades foi à duplicação dos canais físicos [CAR09b]. O roteador resultante suporta até dez portas bidirecionais. A partir dessa nova arquitetura um mecanismo de prioridades pode ser usado na alocação dos canais físicos. A Figura 24 ilustra a arquitetura do novo roteador. Os dois canais físicos em uma mesma direção são numerados de 0 a 1. O mesmo vale para porta local.

A replicação dos canais físicos é uma abordagem que vem se popularizando recentemente [GIL10][YOO10][KAK11]. Tal abordagem é empregada principalmente como uma alternativa a canais virtuais. A implementação de ambas as abordagens implica custos de área relacionados aos *buffers* de entrada e ao *crossbar*. No entanto, a abordagem de canais virtuais necessita ainda um custo extra relativo à multiplexação dos canais físicos (TDM), o que torna seu custo total em área superior à replicação dos canais físicos, considerando o número de canais virtuais e físicos iguais [CAR07b][CAR08]. Além disso, a largura de banda agregada do roteador é diretamente proporcional ao fator de replicação dos canais físicos, ao passo que o aumento do número de canais virtuais não

altera a largura de banda. A potência dissipada por ambas as abordagens é similar [YOO10].

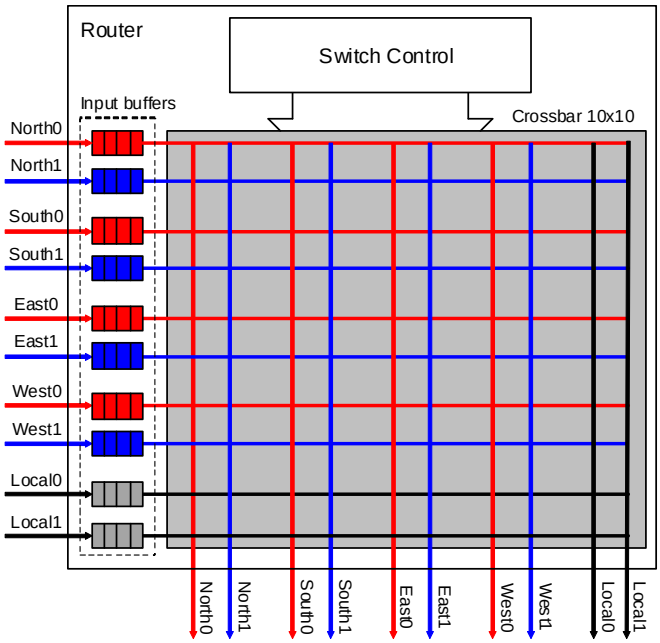


Figura 24 - Arquitetura do roteador com dez portas bidirecionais [CAR11].

Canais virtuais foram introduzidos por Dally e Seitz [DAL87], visando resolver o problema de *deadlock* em redes que implementam *wormhole* e não visando desempenho, apesar de contribuírem para uma melhor utilização dos canais físicos. Essa abordagem é uma herança das redes de computadores, onde há uma limitação significativa no número de conexões que conectam dois elementos comunicantes. Nesse caso a multiplexação do meio físico torna-se a alternativa mais adequada. Por outro lado, dentro de um chip, a abundância de espaço disponível para conexões favorece a replicação dos canais físicos.

O mecanismo de prioridades implementado nesse roteador é baseado em prioridades fixas. Duas classes de tráfego são distinguidas pela NoC: (*i*) pacotes com prioridade alta e (*ii*) pacotes com prioridade baixa. Um canal físico (canal 0) é reservado para transmitir exclusivamente pacotes com prioridade alta, ao passo que o outro canal (canal 1) pode transmitir pacotes com prioridade alta ou baixa. O compartilhamento de um dos canais físicos (canal 1) entre as duas classes de tráfego aumenta o suporte da rede a pacotes com prioridade alta, pois possibilita a transmissão de dois fluxos de alta prioridade simultaneamente na mesma direção. O mecanismo de prioridades oferece um serviço de comunicação diferenciado a tráfegos de alta prioridade através da reserva virtual de recursos (recursos em cinza escuro na Figura 24). Todavia, quando mais de dois fluxos de alta prioridade competem por um mesmo caminho, a interferência entre eles nessa implementação é inevitável. De fato, NoCs que empregam algum tipo de mecanismo de prioridades tendem a atuar como NoCs BE (do inglês, *Best-Effort*) à medida que o tráfego de alta prioridade se intensifica [MEL06]. Esse mecanismo explora a tolerância de algumas aplicações a variações modestas no desempenho da rede, onde a perda de alguns *deadlines* não é um problema (e.g. *soft real time*).

O serviço de comunicação baseado em conexões é suportado a partir do modo de chaveamento por circuito. Esse chaveamento coexiste juntamente com o chaveamento por pacotes, de maneira que a NoC Hermes suporta simultaneamente ambos os chaveamentos. Uma conexão física é estabelecida entre um único par origem/destino e os recursos da rede permanecem alocados durante todo o tempo da comunicação. As conexões são unidirecionais e estabelecidas/encerradas pela origem da comunicação através de pacotes de controle (dois *flits*).

Essa abordagem de chaveamento por circuito é a mais simples que pode ser implementada, considerando a arquitetura da NoC Hermes. Ela exige alterações mínimas na arquitetura do roteador e apresenta um baixo custo em área. Tal simplicidade deve-se ao fato de que o chaveamento por circuito foi implementado sobre o chaveamento por pacotes. No chaveamento por pacotes, os *flits* de *payload* de um pacote comum seguem o caminho alocado pelo *flit* de cabeçalho. Esse caminho permanece alocado até o último *flit* de *payload* ser transmitido. Na abordagem de chaveamento por circuito implementada, o pacote de controle que estabelece uma conexão é o *flit* de cabeçalho de um pacote comum e as mensagens transmitidas pela conexão representam o *payload* desse pacote.

Nesta arquitetura, existe um pacote de controle que estabelece a conexão e outro para liberar a conexão. Uma vez estabelecida à conexão, os dados são transmitidos utilizando o caminho selecionado. Toda a largura de banda do caminho entre origem e destino é alocada pela conexão, permitindo às aplicações atingir a máxima vazão possível sem qualquer tipo de interferência proveniente de outras comunicações. Visto que a alocação total da largura de banda pode subutilizar os recursos quando a vazão das aplicações é baixa, conexões são restritas somente ao canal 0. Assim o canal 1 está sempre disponível para transmitir pacotes usando chaveamento por pacotes.

As portas locais do roteador (local 0 e local 1) são utilizadas pelo PE dependendo do serviço de comunicação. A porta local 0 é utilizada pelo serviço baseado em conexões enquanto a porta local 1 serve o serviço baseado em prioridades. Pacotes injetados na NoC pela porta local 0 são transmitidos a partir do modo de chaveamento por circuito, enquanto os pacotes injetados na porta local 1 são transmitidos utilizando chaveamento por pacotes. Um PE pode manter uma conexão através da porta local 0, enquanto envia pacotes pela porta local 1 utilizando chaveamento por pacotes. Visto que a porta local 0 permite a conexão entre apenas dois PEs, o objetivo dessa abordagem é possibilitar aos PEs conectados comunicarem-se com outros PEs utilizando chaveamento por pacotes através da porta local 1, mantendo a conexão estabelecida.

A NoC diferencia os pacotes injetados a partir de campos específicos no cabeçalho. Quando um pacote chega a um roteador, o módulo *Switch Control* (Figura 24) extrai informações do cabeçalho para executar o algoritmo de roteamento e a alocação/desalocação dos canais físicos.

5.2.2 Serviço De Comunicação Com Roteamento Diferenciado

Os Autores propõem um esquema de roteamento orientado a fluxo que permite rotear fluxos de pacotes de diferentes classes através de diferentes versões do algoritmo de roteamento. Esse esquema visa unir as vantagens dos algoritmos de roteamento determinístico e adaptativo com o propósito de criar um serviço de comunicação com roteamento diferenciado. Visto que um algoritmo adaptativo pode fornecer vários caminhos entre um par origem/destino, ele oferece um serviço de comunicação com uma qualidade superior a um algoritmo determinístico, pois além de estar apto a evitar bloqueios, ele reduz a contenção, ainda que possa acarretar aumento da latência devido ao uso de um caminho mais longo, no caso de algoritmos não-mínimos. A partir do esquema proposto pelos Autores, a NoC passa a suportar simultaneamente os dois tipos de algoritmos. A ideia é utilizar o algoritmo adaptativo no roteamento de mensagens com restrições temporais flexíveis, enquanto as demais são roteadas deterministicamente.

O roteamento orientado a fluxo é um esquema que pode ser implementado tendo como base qualquer algoritmo de roteamento adaptativo. A condição básica é que exista uma versão determinística do algoritmo adaptativo selecionado. Pode ser provado que esta versão sempre existe fixando um único caminho para cada par origem/destino a partir do conjunto de caminhos fornecidos pelo algoritmo adaptativo. Visto que um algoritmo adaptativo oferece caminhos alternativos, ele pode ser aplicado a fluxos de alta prioridade, enquanto fluxos de baixa prioridade são roteados usando a versão determinística do mesmo algoritmo. Os roteadores são responsáveis por selecionar a versão do algoritmo a ser aplicada para cada pacote durante a transmissão.

O roteamento orientado a fluxo pode ser implementado tendo como base algoritmos adaptativos conhecidos como *odd-even* [CHI00] ou aqueles baseados no modelo *turn model* (e.g. *west first* e *north last*) [GLA94]. Este trabalho implementa o roteamento orientado a fluxo sobre a NoC Hermes tendo como base o algoritmo de roteamento Hamiltoniano [LIN94], o qual substitui o algoritmo XY presente na HeMPS. O algoritmo Hamiltoniano foi escolhido devido à simplicidade em obter-se uma versão determinística a partir da versão adaptativa, além de servir de base para a implementação de algoritmos *multicast* (e.g. *dual-path* e *multipath*). Em [EBR10], uma versão mínima adaptativa desse algoritmo chamada HAMUM (*Hamiltonian Adaptive Multicast Unicast Method*) foi comparada com os algoritmos XY, *odd-even* e o esquema DyAD [HU04]. O HAMUM apresentou um desempenho superior em termos de latência média sobre malhas de dimensões 8x8 e 14x14, considerando uma distribuição de tráfego com um *hot-spot*.

O controle sobre qual versão do algoritmo de roteamento deve ser utilizada foi integrada à API de comunicação da plataforma HeMPS através do parâmetro *priority* da primitiva *Send()*. Assim, a plataforma passa a dispor de um serviço de comunicação com roteamento diferenciado, cuja versão do algoritmo a ser utilizada depende apenas do valor do parâmetro *priority*. Durante o processamento da primitiva *Send()*, o valor desse

parâmetro é utilizado para definir o valor do bit de roteamento no cabeçalho do pacote que carrega a mensagem. Definindo o parâmetro *priority* como *HIGH*, indica que a versão adaptativa do algoritmo de roteamento deve ser aplicada. Ao utilizar o valor *LOW*, os pacotes são definidos para serem roteados deterministicamente.

Uma questão a ser tratada ao utilizar um algoritmo de roteamento adaptativo é o ordenamento das mensagens, pois estas podem tomar caminhos diferentes e chegarem ao destino em uma ordem diferente da qual foram enviadas. A premissa para a ocorrência de um desordenamento é a transmissão simultânea de duas ou mais mensagens de uma origem para um mesmo destino. A NoC Hermes não implementa nenhum tipo mecanismo que garanta o ordenamento das mensagens. Entretanto, na plataforma HeMPS, esse ordenamento é assegurado em software pelo protocolo de comunicação *read request* implementado pelo *microkernel* (Seção 2.3.1). Esse protocolo garante que um PE origem somente envia uma mensagem *n* depois que o PE destino recebeu a mensagem *n-1*. Visto que a primitiva *Receive()* é bloqueante, é impossível requisitar uma segunda mensagem antes de ter recebido a primeira. Isto garante que nunca há mais de uma mensagem sendo transmitida para um mesmo destino em um determinado instante, portanto o ordenamento é assegurado.

5.2.3 Comparativo

O trabalho apresentado nesta Seção, inseriu na plataforma HeMPS dois serviços que contribuem para o controle de QoS das aplicações. As aplicações com requisitos de QoS podem contar com uma transmissão adicional, que utilizam algoritmo de roteamento e canais físicos diferenciados. Estes serviços de transmissão são acionados através da informação de prioridade na primitiva SEND().

No trabalho proposto na presente Tese de Doutorado o controle dos parâmetros de QoS de uma aplicação é realizado através de uma malha fechada de controle, onde tais parâmetros são monitorados e controlados através do ajuste da frequência de operação dos PEs. As aplicações com requisitos de QoS são diferenciadas das demais através de uma chamada de sistema que aciona o monitoramento dos parâmetros de QoS e de outra chamada de sistema que seleciona os limiares de operação destes parâmetros.

5.3 Madalozzo et. al. [MAD13]

O Autores em [MAD13] propõem um mecanismo para o controle de QoS na HeMPS baseado em malha fechada de controle. Os Autores utilizam o esquema de monitoramento descrito na Seção 2.1.3 para realizar uma avaliação dos parâmetros de QoS (latência e vazão) de uma dada tarefa, e modificam estes parâmetros através das técnicas de prioridades do algoritmo de escalonamento e migração de tarefas. As seções a seguir descrevem estas duas técnicas utilizadas no ajuste do QoS de uma aplicação.

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

Nesta Seção apresenta-se o algoritmo de escalonamento adaptado utilizado em [MAD13]. Este algoritmo tem por base o escalonamento *Round-Robin* com prioridade de tarefas, descrito em [LI03]. 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 25, 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. O algoritmo escalona as tarefas em uma fila circular regular, executando a Task2 e na sequência 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.

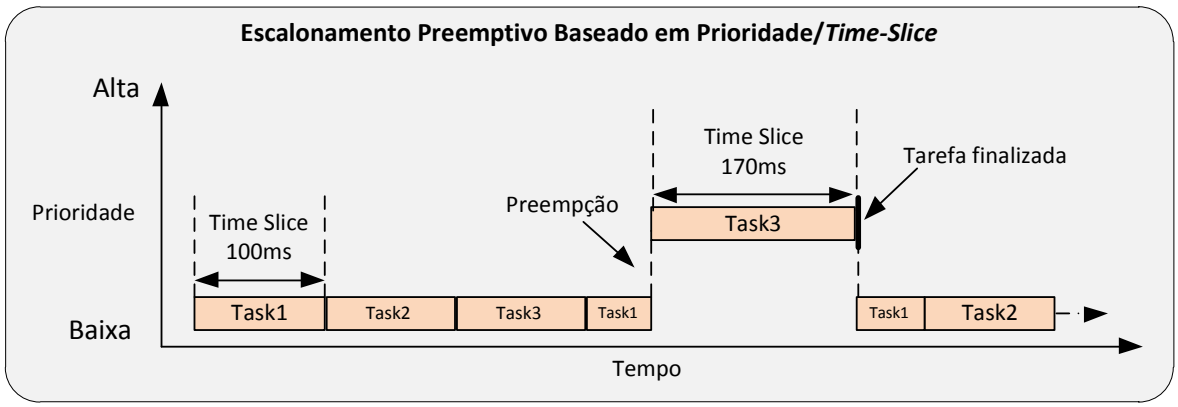


Figura 25 - Escalonamento preemptivo baseado em prioridade/*time-slice* [MAD13].

O monitoramento de tarefas é utilizado para realizar o controle de violações de *deadlines*, ajustando 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 aumentado 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.

5.3.2 Migração de Tarefas

Nesta Seção apresenta-se a técnica de migração de tarefas desenvolvida em [MAD13]. Esta técnica tem por objetivo 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 pode-se reduzir a energia consumida na comunicação e otimizar o desempenho das aplicações do sistema.

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 26, 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 (PE que está monitorando uma ou mais tarefas nele executando) verifica violações de *deadline*, resultando na necessidade de executar uma técnica de adaptabilidade, a fim de realizar o controle de QoS da aplicação. 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 executa a heurística para saber qual tarefa deve ser migrada, computando qual será a nova posição dessa tarefa. Note que a tarefa que será migrada continua sua execução, em paralelo.
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, efetuou-se 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 (do protocolo de comunicação da HeMPS), com sua nova posição, e o PE receptor atualiza sua tabela de tarefas.

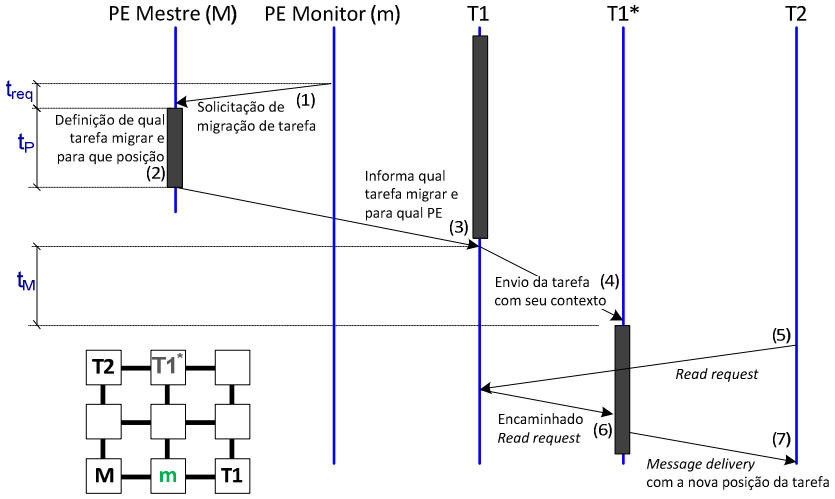


Figura 26 - Protocolo de migração de tarefas com atividades por processador [MAD13].

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

5.3.2.1 Interação do Monitoramento com Migração de Tarefas

Em [MAD13], 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 (Seção 5.3.1). 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 27, apresenta-se a interação do monitoramento com a migração de tarefas. Para ilustrar a migração de tarefas foi utilizada uma arquitetura distribuída de recursos. A etapa 1 da Figura ilustra a configuração inicial da aplicação. O processador gerente local, ou LMP_0 (do inglês, *Local Manager Processor*), 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. Na etapa 3 são exibidos 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 na Seção 5.3.1, afetando diretamente o escalonamento.

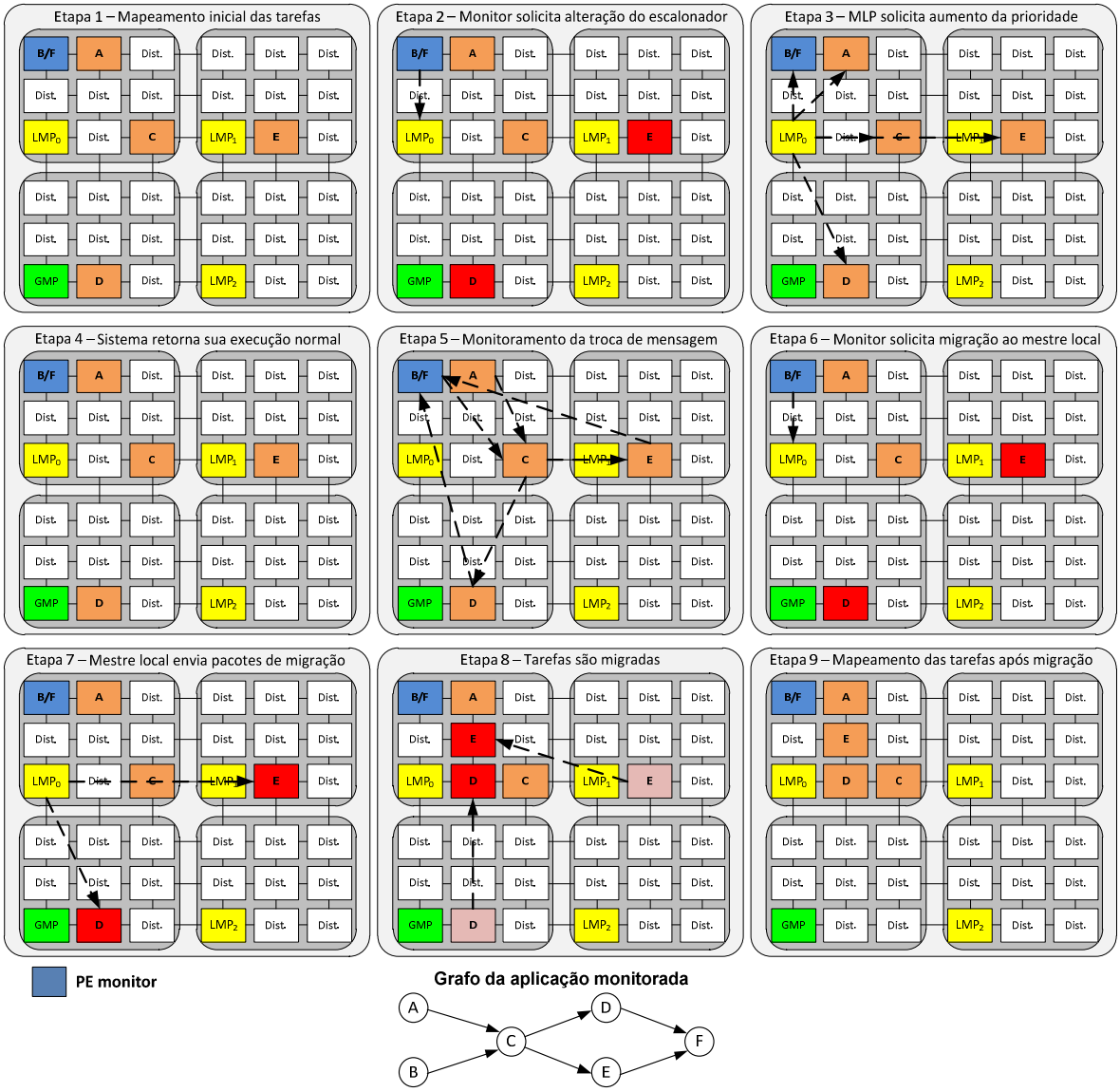


Figura 27 - Mecanismo de monitoramento com migração de tarefas [MAD13].

Depois 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, comportamento descrito na etapa 4. Na Etapa 5, o PE monitor verifica as mensagens de comunicação das tarefas. Na Etapa 6, o PE monitor avalia que a quantidade de perdas de *deadline* é superior àquela configurada no *max_miss_migra* da TCB da tarefa F. Desta forma, é gerada uma solicitação da execução da técnica de adaptabilidade ao PE mestre local. 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 tarefas D e E. Na Etapa 8 ocorre à migração das tarefas e logo em seguida, na Etapa 9, as tarefas migradas são escalonadas em suas novas posições.

5.3.3 Comparativo

O trabalho apresentado nesta Seção utiliza uma malha de controle fechada para realizar o controle de QoS das aplicações. Este controle avalia parâmetros de QoS das aplicações através de PEs de monitoração, que comandam mudanças na política de escalonamento e migrações de tarefas da aplicação.

O trabalho desta Tese de Doutorado, também utiliza uma malha fechada para realizar o controle de QoS das aplicações. Entretanto, o trabalho desenvolvido utiliza a técnica de DFS para controlar o QoS das aplicações.

5.4 Conclusão

Os trabalhos apresentados neste Capítulo utilizam diversos mecanismos para controlar os parâmetros de QoS das aplicações que executam na NoC Hermes ou plataforma HeMPS. Com foco na plataforma HeMPS, os dois trabalhos apresentados buscam gerenciar o QoS das aplicações com serviços dedicados [CAR11] ou através de ajustes na política de escalonamento com prioridades e migração de tarefas [MAD13].

O trabalho realizado no escopo desta Tese de Doutorado, está mais próximo ao trabalho de [MAD13] utilizando uma malha fechada de controle para controlar parâmetros de QoS das aplicações. O trabalho desenvolvido, descrito no Capítulo 7, utiliza a técnica de DFS implementada na plataforma HeMPS em [ROS12b] para controlar o QoS das aplicações.

6 MODELO DO CONSUMO DE ENERGIA NA HEMPS

Neste Capítulo é apresentado o modelo de consumo de energia implementado e avaliado na plataforma HeMPS. Este modelo é baseado em taxas de transmissão nos enlaces da NoC e nas instruções executadas pelo processador. Foi desenvolvida uma ferramenta que automatiza o modelo de consumo de energia na HeMPS e realiza o cálculo deste consumo. A Figura 28 descreve o fluxo de informações para a avaliação do consumo de energia na plataforma HeMPS.

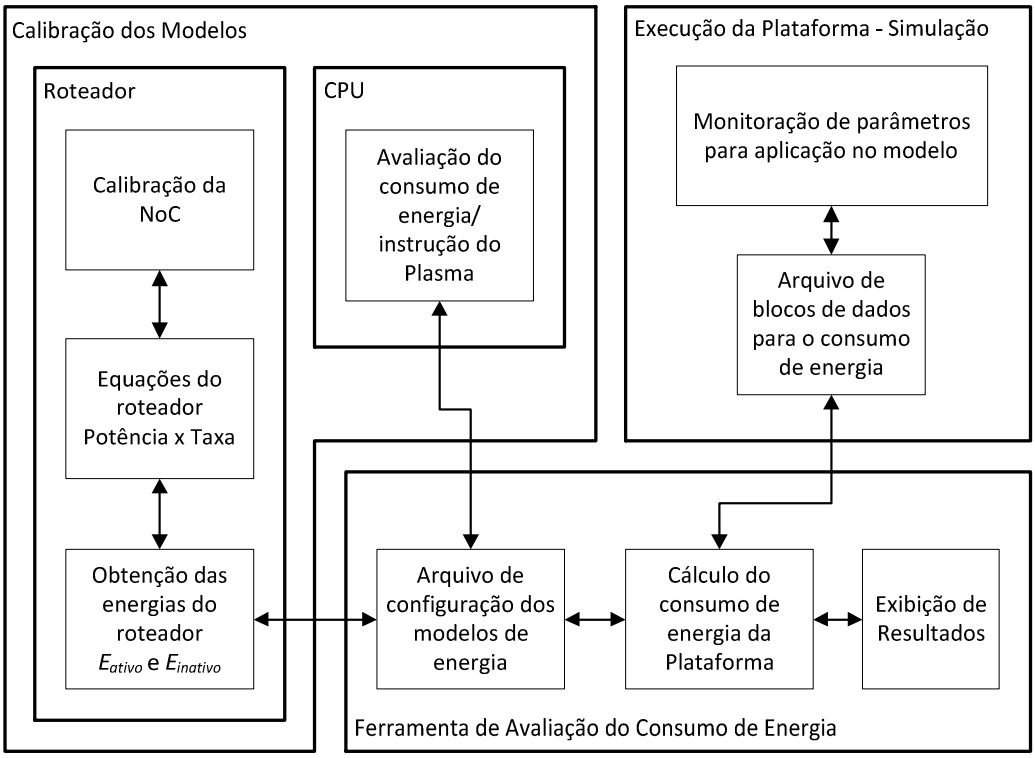


Figura 28 - Fluxo de avaliação do consumo de energia na plataforma HeMPS.

O restante deste Capitulo está organizado como segue. A Seção 6.1 descreve o modelo do consumo de energia nos roteadores da NoC HERMES, e a Seção 6.2 descreve o modelo do consumo de energia nos processadores Plasma. A Seção 6.3 descreve o modelo do consumo de energia na HeMPS, ou seja, utiliza ambos os modelos descritos nas seções anteriores, além de modelar o consumo de energia nos fios de interconexão entre os roteadores. A Seção 6.4 descreve a ferramenta de estimativa do consumo de energia da plataforma HeMPS.

6.1 Modelo do consumo de energia na NoC HERMES

O modelo de consumo de energia na NoC HERMES utilizado neste trabalho foi originalmente desenvolvido em [GUI08b], e adaptado para o presente trabalho. Neste modelo, o consumo de energia da NoC é determinada pela soma dos consumos de energia nos seus roteadores, e estes consumos de energia são calculados multiplicando-

se a dissipação de potência do roteador pelo tempo de simulação executado (em ciclos de relógio).

Segundo [GUI08b], a dissipação de potência de um roteador da NoC pode ser dividida em três componentes: (1) dissipação nos buffers; (2) dissipação na lógica de controle e roteamento; (3) dissipação no crossbar interno ao roteador. Como analisado em [PAL05], a maior dissipação de potência do roteador ocorre nos buffers, e análises posteriores, mostraram que estes dissipam até 90% da potência total nos roteadores da NoC Hermes. Desta forma, a dissipação de potência geral do roteador está fortemente ligada à dissipação de cada um dos seus buffers. Caso a NoC possua uma arquitetura sem buffers, com roteamento do tipo "*batata quente*" (hot-potato) [IL05], as dissipações de potência do crossbar e lógica de controle e roteamento poderão ser uma função da taxa média de transmissão, assim como a taxa de transmissão afeta a dissipação de potência nos buffers.

A dissipação de potência em um buffer de entrada deve-se a dois componentes principais: o chaveamento no sinal de relógio e o chaveamento do dado armazenado. O sinal de relógio é um requisito de base em um circuito síncrono, como é o caso no roteador da NoC Hermes. A transição periódica do sinal de relógio do buffer gera a dissipação de potência base do mesmo. Soma-se a esta, a dissipação gerada pela troca do valor lógico que este buffer armazena. Sempre que houver uma mudança de pelo menos um bit no valor armazenado pelo buffer, ocorrerá uma maior dissipação do que a potência de base. Esta dissipação, adicionada à dissipação de base, irá aumentar na medida em que mais bits do valor armazenado troquem de valor lógico [PAL07].

A troca dos valores lógicos nos buffers está ligada à taxa de recepção destes. Quanto mais rápido o buffer receber valores, maior será a probabilidade de que algum bit armazenado mude de valor lógico, ocasionando assim uma dissipação de potência maior do que a de base. Desta forma, a taxa de recepção de um buffer irá refletir diretamente em alterações na dissipação de potência deste. O modelo utilizado para a estimativa da dissipação de potência em NoCs opera em duas etapas: calibração e aplicação.

A primeira etapa do método corresponde à calibração (Figura 29 à esquerda) do modelo. Em uma NoC 3x3 o roteador central, com 5 portas de entrada, é sintetizado utilizando a tecnologia 0,65 nm da IBM [IBM14], tecnologia disponível no laboratório GAPH durante os estudos apresentados nesta Tese de Doutorado. Após a síntese lógica, a descrição VHDL original do roteador central é substituída pela descrição RTL sintetizada deste roteador, utilizando a biblioteca de células desta tecnologia. Com esta nova descrição, é feita a simulação para diferentes padrões de tráfego (os quais variam a carga nos buffers do roteador de 0% à 50% do máximo), obtendo-se através destas simulações um arquivo com a atividade de chaveamento das portas lógicas. Estes tráfegos simulados são desenvolvidos com trocas da maioria dos bits de um pacote para o outro, sendo assim criado um tráfego de pior caso de chaveamento dos bits de um pacote. Com este

arquivo é possível obter-se a potência média de cada elemento do roteador. As equações 1 a 3 apresentam a potência média, em mW@100MHz, para cada componente da NoC, para largura do flit igual a 16 bits, e profundidade do buffer igual a 8 flits. Tais equações foram obtidas utilizando-se a etapa de calibração descrita acima.

$$P_{buffer} = (91,9468 * taxa) + 49,5853 \quad (1)$$

$$P_{cross-bar} = (24,2008 * taxa) + 6,68903 \quad (2)$$

$$P_{lógica\_de\_controle} = (69,0515 * taxa) + 43,9689 \quad (3)$$

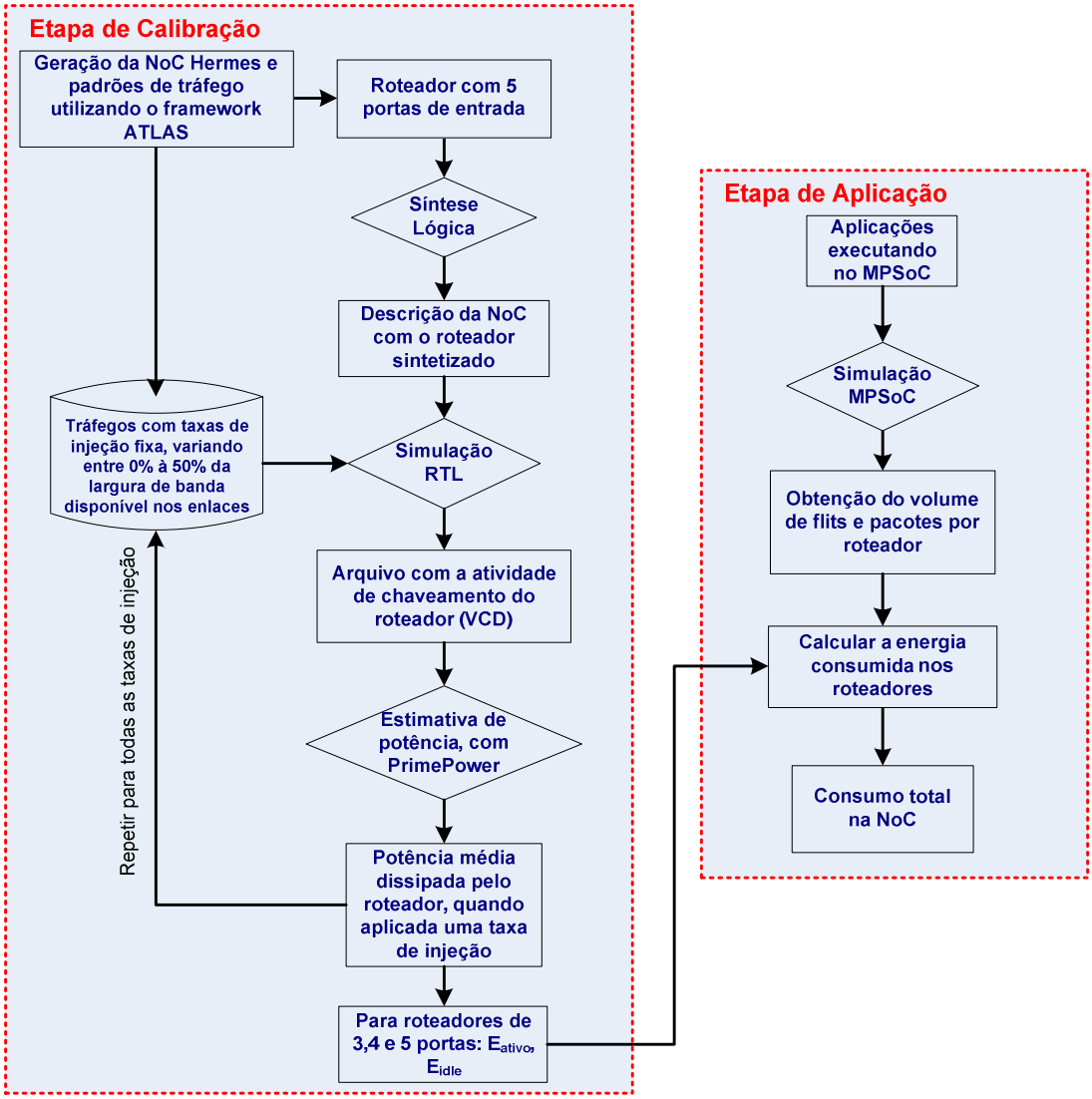


Figura 29 - Fluxo de aplicação do modelo de estimativa da dissipação de potência na NoC HERMES.

As equações de potência, descritas anteriormente, são utilizadas para o cálculo da energia consumida pelo roteador. Uma importante característica do MPSoC é a forma como os dados são transmitidos. Os pacotes são armazenados integralmente na memória do elemento de processamento, para posteriormente serem transmitidos em rajada. O efeito prático desta característica é haver duas taxas de transmissão: (*i*) uma igual a 100%, quando o pacote está sendo injetado em uma dada porta local; (*ii*) outra igual a

0%, quando não há injeção de dados na rede. Assim, dois consumos de energia são considerados: (*i*) o consumo do roteador quando transmitindo um *flit* – E_{ativo}; (*ii*) o consumo do roteador em modo inativo – $E_{inativo}$.

Obtém-se o valor E_{ativo} , para cada flit transmitido, através da equação 4.

$$E_{ativo} = \left[(n\_portas - 1) * P_{buffer}(0) + P_{buffer}(1) + P_{cross-bar}(1) + P_{lógica\_de\_controle}(1) \right] * T \quad \textbf{(4)}$$

Onde: *n_portas* corresponde ao número de portas do roteador, $P_{buffer}(0)$ potência consumida pelo buffer quando não há transmissão de dados, $P_{elemento}(1)$ potência consumida por um dado elemento do roteador quando há transmissão de dados (taxa=100%), T é o período utilizado na etapa de calibração em ns.

De forma análoga obtém-se o consumo em modo inativo, conforme equação 5.

$$E_{inativo} = \left[(n\_portas) * P_{buffer}(0) + P_{cross-bar}(0) + P_{lógica\_de\_controle}(0) \right] * T \quad \textbf{(5)}$$

Visando reduzir a energia consumida na NoC, o roteador opera com duas frequências distintas. Quando há transmissão de dados em alguma das portas do roteador, a frequência utilizada é igual a 100 MHz. Quando o roteador não está transmitindo dados, a frequência do mesmo é reduzida para 10 MHz. A Tabela 4 apresenta os valores para E_{ativo} e $E_{inativo}$, considerando as duas frequências de operação.

Tabela 4 - Valores de E_{ativo} e $E_{inativo}$ para o roteador da NoC HERMES.

Frequência / Período	Número de portas do roteador	Energia - pJ
100 MHz / 10 ns	3	E_{ativo} = 3,815
	4	E_{ativo} = 4,304
	5	E_{ativo} = 4,793
10 MHz / 100 ns	3	$E_{inativo}$ = 13,481
	4	$E_{inativo}$ = 16,278
	5	$E_{inativo}$ = 19,076

A segunda etapa do modelo corresponde à utilização dos valores obtidos na Tabela 4 (Figura 29 à direita – *etapa de aplicação*). O MPSoC é simulado no nível RTL. Para calcular a energia consumida é necessário determinar o tempo no qual o roteador é utilizado para transmissão de pacotes, e o volume total de flits/pacotes transmitidos por cada roteador. O tempo no qual um roteador permanece ativo, em ciclos de relógio, é dado pela equação 6. O tempo no qual o roteador está ativo corresponde ao somatório do tempo gasto com a transmissão de flits, e o tempo necessário para arbitragem e roteamento de cada pacote (na NoC HERMES é de 5 ciclos de relógio). O tempo no qual o roteador permanece inativo é dado pela equação 7. Este tempo é obtido através da

diferença entre o tempo total de simulação e o tempo do roteador ativo, considerando-se a relação entre os períodos ativo e inativo (o tempo $ciclos_{inativo}$ deve corresponder ao número de ciclos na menor frequência).

$$ciclos_{ativo} = \sum flits + \sum pacotes * 5 \quad \textbf{(6)}$$

$$ciclos_{inativo} = \left(ciclos_{simulação} - ciclos_{ativo}\right) * \left(\frac{T_{ativo}}{T_{inativo}}\right) \quad \textbf{(7)}$$

Utilizando-se as equações anteriores, obtém-se a energia consumida por um roteador com *k* portas de entrada (equação 8). A soma da energia consumida por todos os roteadores corresponde à energia total consumida na NoC.

$$E_{rot_k} = \left(E_{ativo} \times ciclos_{ativo}\right) + \left(E_{inativo} \times ciclos_{inativo}\right) \quad \textbf{(8)}$$

6.2 Modelo do consumo de energia no processador Plasma

O modelo de consumo de energia do processador Plasma utilizado neste trabalho é baseado na quantidade de instruções que são executadas pelo processador. Aferiu-se em [FIL12] a energia consumida pelo processador Plasma para as diferentes instruções por ele executadas em uma freqüência de 100 MHz, e verificou-se que instruções de uma dada categoria (lógica, saltos, aritméticas, etc) possuem consumo de energia similar. Desta forma, as principais instruções executadas pelo processador Plasma foram categorizadas, e o consumo de energia de cada categoria está descrito na Tabela 5 (valores atualizados para tecnologia de 65nm).

O número de execuções de cada categoria de instrução é contabilizado, separando-se por instruções do *kernel* ou de tarefa (usuário). A seguir multiplica-se o total de instruções de uma categoria pelo seu consumo de energia (potência dissipada multiplicada pelo tempo de execução daquele conjunto de instruções), realizando este cálculo para todas as categorias da Tabela 5. Obtêm-se, então, o consumo de energia tanto para a execução do *kernel* quanto para as tarefas, e somando-se estes dois consumos têm-se o consumo de energia do processador ao executar uma dada aplicação. O modelo de consumo de energia do processador não avalia o consumo devido a memória, e somente o consumo devido a execução das instruções.

Ao se empregar o mecanismo de DFS para o processador, pode-se admitir diferentes frequências de operação. Desta forma os valores de dissipação de potência descritos na Tabela 5 não poderão ser aplicados diretamente. Podem-se extrapolar os valores de dissipação de potência para o processador em diferentes frequências de operação, derivando-se a equação de dissipação de potência a partir de duas dissipações de potência conhecidas.

Tabela 5 - Dissipação de potência das diferentes categorias de instruções do processador Plasma (frequência do processador: 100 MHz, tecnologia 65 nm).

Categoria	Instruções	Dissipação de Potência Média (mW)
Aritmética	MULT, MULTU, DIV, ADD, ADDU, SUB, SUBU, SLT, SLTU, DADDU, ADDI, ADDIU e SLTI, SLTIU	5,099
Saltos	JR, JALR, BLTZAL, BLTZ, BGEZ, BLTZALL, BLTZL, BGEZALL, BGEZL, JAL, J, BEQ, BNE, BLEZ, BGTZ, BEQL, BNEL, BLEZL e BGTZL	5,869
Load/Store	LB, LH, LW, LBU, LHU, SB, SH, SW, LL e SC	3,994
Lógica	AND, OR, XOR, NOR, ANDI, ORI, XORI e LUI	4,363
Move	MOVZ, MOVN, MFHI, MTHI, MFL0, MTL0 e COP0	3,143
Shift	SLL, SRL, SRA, SLLV, SRLV e SRAV	3,824
Demais	SYSCALL, BREAK, SYNC, TGEU, TLT, TLTU, TEQ, TNE, LWL, LWR, SWL, SWR, e SWC1	5,099

Dado que a dissipação de potência é linear em função da frequência [RAB03], de posse do consumo em 100 MHz e 20 MHz é possível derivar a potências para as demais frequências. Assim, pode-se realizar a aproximação linear e derivar a equação que indica a dissipação de potência de uma categoria de instrução do processador Plasma para frequências entre 1 e 100 MHz. A Tabela 6 exibe os resultados obtidos através desta aproximação linear.

Tabela 6 - Valores da dissipação de potência média das instruções do Plasma obtidos através de aproximação linear (tecnologia 65 nm).

Frequência	Aritmética	Saltos	Load/Store	Lógica	Move	Shift
100 MHz	5,099	5,869	3,994	4,363	3,143	3,824
87,5 MHz	4,623	5,321	1,116	3,956	2,850	3,467
75 MHz	4,147	4,774	1,105	3,549	2,556	3,110
62,5 MHz	3,672	4,226	1,095	3,142	2,263	2,754
50 MHz	3,196	3,678	1,084	2,735	1,970	2,397
37,5 MHz	2,720	3,131	1,073	2,327	1,677	2,040
25 MHz	2,244	2,583	1,063	1,920	1,383	1,683
12,5 MHz	1,769	2,036	1,052	1,513	1,090	1,326
6,25 MHz	1,531	1,762	1,047	1,310	0,943	1,148

6.3 Consumo de energia na HEMPS

Para aferir o consumo de energia na plataforma HeMPS utilizam-se os dois modelos descritos nas Seções 6.1 e 6.2. Outros módulos do PE, como DMA ou NI, não são considerados na estimativa do consumo de energia, visto que o consumo de energia destes é constante e inferior ao do processador ou do roteador (valores de potência média em 65 nm para os módulos NI e DMA são 0,683 e 0,111 mW, respectivamente). São monitorados, em tempo de execução os seguintes parâmetros:

- Quantidade de *flits* e pacotes recebidos em cada porta de cada roteador;
- Quantidade de instruções de cada classe executadas pelo *kernel* e pelas tarefas;
- A frequência em que o PE opera em cada momento da simulação.

Estes parâmetros são armazenados em uma estrutura de dados acrescida ao ISS (do inglês Intruction Set Simulator) do processador Plasma. Esta estrutura de dados é construída todas as vezes em que o processador realizar uma instrução de SYSCALL. Tal estrutura é divida nos campos descritos na

Tabela 7 abaixo.

Com base nesta estrutura de dados, aplicam-se os modelos de consumo do processador e do roteador após a simulação, a fim de se calcular o consumo de energia na plataforma HeMPS.

Como mencionado na Seção 6.3, no modelo de cálculo de energia abstraiu-se a influência de outros módulos do PE, como DMA ou NI. Entretanto, é importante se avaliar o consumo nos fios que interconectam os PEs, pois estes são uma componente que apresenta um consumo variável conforme o volume de dados transmitido.

Para se determinar o consumo em um determinado fio deve-se conhecer o comprimento do fio, sua capacitância, tensão do nível lógico ‘1’. Assume-se um tamanho do PE igual a 1 mm^2 (resultado obtido com a síntese física do PE para tecnologia de 65 nm), tensão de alimentação igual a 1V, roteamento em metal 5 (0,2194 fF/micron), largura dos fios igual a 0,3 μm e distância entre fios igual a 0,3 μm. As equações 9 a 11 apresentam o consumo para um fio com estas características. Para determinar o consumo em um determinado fio não precisamos da noção de tempo, pois precisamos saber apenas o número de vezes em que as capacitâncias dos fios carregam e descarregam.

$$capacitância\ para\ fio\ de\ 1\ mm\ em\ metal\ 5 = 219{,}4\ fF \quad \textbf{(9)}$$

$$energia\ no\ fio\ (E_{fio}) = C*v^2 = 219{,}4\ fJ = 0{,}2194\ pJ \quad \textbf{(10)}$$

$$energia\ no\ enlace\ (wire\_energy) = E_{fio} * 16 = 3{,}5104\ pJ \quad \textbf{(11)}$$

Tabela 7 - Estrutura do bloco de dados para a avaliação do consumo de energia na HeMPS, gerado durante uma execução.

Campo da Estrutura de Dados	Descrição do Campo
#BLOCK	Delimitador do início de uma nova estrutura de dados
cycles_sim	Tempo de execução, em ciclos de relógio, até o momento da gravação desta estrutura de dados
freq_num_cpu	Campo *"numerador"* do valor da frequência do PE
freq_den_cpu	Campo *"denominador"* do valor da frequência do PE
arith_inst_kernel	Número de instruções da categoria aritmética executadas pelo *kernel*
branch_inst_kernel	Número de instruções da categoria saltos executadas pelo *kernel*
load_inst_kernel	Número de instruções da categoria *load/store* executadas pelo *kernel*
logical_inst_kernel	Número de instruções da categoria lógica executadas pelo *kernel*
move_inst_kernel	Número de instruções da categoria *move* executadas pelo *kernel*
shift_inst_kernel	Número de instruções da categoria *shift* executadas pelo *kernel*
other_inst_kernel	Número de instruções de outras categorias executadas pelo *kernel*
arith_inst_user	Número de instruções da categoria aritmética executadas pelas tarefas
branch_inst_ user	Número de instruções da categoria saltos executadas pelas tarefas
load_inst_ user	Número de instruções da categoria *load/store* executadas pelas tarefas
logical_inst_ user	Número de instruções da categoria lógica executadas pelas tarefas
move_inst_ user	Número de instruções da categoria *move* executadas pelas tarefas
shift_inst_ user	Número de instruções da categoria *shift* executadas pelas tarefas
other_inst_ user	Número de instruções de outras categorias executadas pelas tarefas
east_flits	Número de flits recebidos na porta leste do roteador deste PE
west_flits	Número de flits recebidos na porta oeste do roteador deste PE
north_flits	Número de flits recebidos na porta norte do roteador deste PE
south_flits	Número de flits recebidos na porta sul do roteador deste PE
local_flits	Número de flits recebidos na porta local do roteador deste PE
east_pckts	Número de pacotes recebidos na porta leste do roteador deste PE
west_pckts	Número de pacotes recebidos na porta oeste do roteador deste PE
north_pckts	Número de pacotes recebidos na porta norte do roteador deste PE
south_pckts	Número de pacotes recebidos na porta sul do roteador deste PE
local_pckts	Número de pacotes recebidos na porta local do roteador deste PE
#END_BLOCK	Delimitador de final da estrutura de dados

O consumo de energia de um dado enlace é obtido pela equação 12:

$$Pwire_{port} = wire\_energy \times flits_{port} * \alpha \quad \textbf{(12)}$$

onde: *wire_energy* é a energia gasta para transmitir um flit no enlace; $flits_{port}$ é a quantidade de flits recebida em uma dada porta do roteador; α é a atividade de chaveamento do fio (assume-se um valor igual a 0,4, o que é bastante elevado, pois se considera que 40% dos bits estão chaveando).

Somente são consideradas no cálculo de energia dos fios as conexões entre roteadores, visto que a conexão entre o processador e o roteador (porta local) é muito curta e, sendo assim, possui um consumo de energia negligenciável. O cálculo do consumo de energia dos fios de conexão entre PEs é realizado através da soma da energia consumida em todos os enlaces do MPSoC.

6.4 Ferramenta de avaliação do consumo de energia na HEMPS

No escopo desta Tese de Doutorado e no intuito de automatizar o cálculo de energia da plataforma HeMPS, foi desenvolvida uma ferramenta, na linguagem de programação C++, que automatiza as etapas de aplicação dos modelos de consumo de energia sobre as estruturas de dados obtidas através da simulação da plataforma. A esta ferramenta deu-se o nome de *hemps_power*.

A *hemps_power* utiliza um arquivo de configuração, denominado de *config_file.cfg* que possui blocos de configuração similares àqueles construídos com os dados de monitoração. A Tabela 8 descreve cada campo deste arquivo de configuração.

Tabela 8 - Descrição do arquivo de configuração utilizado na ferramenta hemps_power.

Campo	Descrição
#CPU	Delimitador que inicia o bloco de configuração do processador
#BLOCK	Delimitador que inicia um bloco de configuração
freq_num	Campo “numerador” da configuração de frequência do processador
freq_den	Campo “denominador” da configuração de frequência do processador
inst_class	Categoria da instrução executada (aritmética, salto, lógica, etc...)
inst_power	Potência média dissipada pelo processador na execução desta categoria de instrução para esta frequência
#END_BLOCK	Delimitador que finaliza o bloco de configuração
<vários blocos>	Existem blocos de configuração para todas as categorias de instrução do processador em cada frequência que ele pode executar
#END_CPU	Delimitador que finaliza o bloco de configuração do processador
#NOC	Delimitador que inicia o bloco de configuração da NoC
#BLOCK	Delimitador que inicia um bloco de configuração
freq_num	Campo “numerador” da configuração de frequência do roteador
freq_den	Campo “denominador” da configuração de frequência do roteador
numb_ports	Número de portas de comunicação deste roteador (na NoC HERMES pode-se ter de 3 a 5 portas)
e_active	Consumo de energia quando o roteador está ativo
e_idle	Consumo de energia quando o roteador está inativo
#END_BLOCK	Delimitador que finaliza o bloco de configuração
<vários blocos>	Existem blocos de configuração para todas as frequências de operação do roteador e quantidade de portas por ele admitidas
#END_NOC	Delimitador que finaliza o bloco de configuração da NoC
#WIRE	Delimitador que inicializa o bloco de configuração do consumo de energia no fio de interconexão
w_energy	Energia consumida na transmissão de um flit através de um enlace
alpha	Constante que descreve a atividade de chaveamento nos fios
#END_WIRE	Delimitador que finaliza o bloco de configuração do consumo de energia no fio de interconexão

A cada execução da simulação a ferramenta lê este arquivo de configuração, e configura os modelos de energia para receber os dados de monitoramento. Este arquivo permite a flexibilização da aplicação do modelo de consumo de energia, pois se pode avaliar diferentes variantes da plataforma HeMPS bastando uma calibração prévia para a extração dos parâmetros de configuração.

Para a execução da ferramenta *hemps_power*, utiliza-se a seguinte sintaxe: ***hemps_power <caminho do cenário de teste> <nome do cenário de teste>.hmp***. O caminho do cenário de teste garante que a ferramenta terá acesso aos arquivos de

monitoramento gerados durante a execução da plataforma, já o arquivo *hmp* permite que a ferramenta saiba qual a configuração da HeMPS foi utilizada e quais tarefas foram executadas.

Os resultados gerados pela ferramenta são gerados conforme a Tabela 9. A Figura 30 apresenta o resultado de uma simulação sobre a plataforma. O consumo pelos dois PEs apresentados deve-se majoritariamente aos processadores, pois neste exemplo o volume de dados transmitido entre os PEs é muito baixo.

Tabela 9 - Apresentação dos resultados da ferramenta *hemps_power*.

==========				
PE: *<numero do PE>*				
Task: *<nome da tarefa>*				
SIM TIME(cycles);	PE Energy (uJ);	CPU Energy(uJ);	Router Energy(uJ);	Wire Energy(uJ)
tempo x;	*energia x;*	*energia x;*	*energia x;*	*energia x*
tempo y;	*energia y;*	*energia y;*	*energia y;*	*energia y*
==========				
PE: *<numero do PE>*				
Task: *<nome da tarefa>*				
SIM TIME(cycles);	PE Energy (uJ);	CPU Energy(uJ);	Router Energy(uJ);	Wire Energy(uJ)
tempo x;	*energia x;*	*energia x;*	*energia x;*	*energia x*
tempo y;	*energia y;*	*energia y;*	*energia y;*	*energia y*

```
========================================================================
PE:50
Task: D12.c
SIM TIME(cycles);PE Energy(uJ);CPU Energy(uJ);Router Energy(uJ);Wire Energy(uJ)
34535;366,966;366,44;0,524488;0,0014498
35313;378,654;377,591;1,06092;0,0014498
36197;387,765;386,153;1,61096;0,0014498
36767;396,533;394,361;2,16974;0,0014498
37659;410,009;407,265;2,74221;0,0014498
39107;431,28;427,941;3,33689;0,00146735
43549;489,908;485,908;3,99919;0,00160776
44433;499,146;494,469;4,6751;0,00160776
44995;507,909;502,547;5,35962;0,00160776
45797;520,091;514,033;6,05645;0,00160776
45797;520,091;514,033;6,05645;0,00160776
========================================================================
PE:16
Task: taskF_1.c
SIM TIME(cycles);PE Energy(uJ);CPU Energy(uJ);Router Energy(uJ);Wire Energy(uJ)
7613;99,338;99,2271;0,109266;0,00155511
8423;110,849;110,616;0,231009;0,00155511
9249;122,577;122,21;0,365474;0,00155511
9671;128,976;128,468;0,50644;0,00155511
10111;135,615;134,96;0,654183;0,00155511
11057;149,307;148,489;0,816452;0,00155511
11941;158,045;157,051;0,992337;0,00155511
12509;166,409;165,231;1,17693;0,00155511
13393;179,376;177,999;1,37509;0,00155511
14145;190,412;188,826;1,58483;0,00155511
14897;201,46;199,653;1,80616;0,00155511
15649;212,52;210,479;2,03907;0,00155511
16401;223,591;221,306;2,28356;0,00155511
17153;234,674;232,133;2,53963;0,00155511
17905;245,768;242,959;2,80729;0,00155511
28447;375,977;372,739;3,23733;0,00155511
38989;506,349;502,518;3,82975;0,00155511
60009;765,712;760,965;4,74585;0,00157266
640594;4829,39;4814,78;14,6011;0,00201497
650781;5074,89;5050,28;24,6108;0,0024292
661165;5319,69;5284,91;34,7781;0,00284342
671341;5560,88;5515,78;45,0995;0,00325765
681509;5802;5746,42;55,5751;0,00367188
691688;6043,5;5977,29;66,2049;0,00408611
```

Figura 30 - Exemplo da apresentação dos resultados da ferramenta *hemps_power*.

6.5 Considerações Finais

Neste Capítulo apresentou-se o mecanismo de avaliação do consumo de energia na Plataforma HeMPS que foi desenvolvido nesta Tese de Doutorado. Este mecanismo utiliza-se de três modelos de consumo de energia, um para os roteadores da NoC HERMES, um para o processador Plasma e o último para os fios de interconexão. Os parâmetros para o cálculo dos modelos são gerados em tempo de execução, porém a aplicação dos modelos é realizada após a execução através de uma ferramenta desenvolvida no escopo desta Tese.

Somente através do desenvolvimento desta ferramenta para o cálculo de energia consumida pelo MPSoC é possível avaliar a eficácia dos mecanismos de controle propostos nos Capítulos subsequentes. A avaliação dos mecanismos propostos através de uma simulação no nível de portas lógicas, e a correspondente avaliação de potência através de um arquivo com atividade de chaveamento, é inviável computacionalmente.

Assim, este Capítulo representa uma primeira contribuição desta Tese, que é uma ferramenta capaz de avaliar a energia consumida por um conjunto de aplicações executando na plataforma HeMPS. Simplificações foram feitas no modelo, porém estas representam um consumo constante que está sendo negligenciado (DMA, NI e lógica de interconexão entre os módulos), não comprometendo a avaliação das heurísticas que serão apresentadas na sequência.

7 CONTROLE DE PARÂMETROS DE QOS DAS APLICAÇÕES NA HEMPS UTILIZANDO DFS

Neste Capítulo é apresentado o mecanismo de controle de parâmetros de QoS das aplicações na HeMPS utilizando a técnica de DFS, desenvolvido no escopo deste trabalho e contribuição principal da presente Tese de Doutorado.

Como exemplificado nos Capítulos 2 e 5, diferentes técnicas podem ser utilizadas para atender aos requisitos de QoS das aplicações, tanto em tempo de projeto quanto em tempo de execução. Nos MPSoCs o não atendimento a estes requisitos pode ser uma constante, já que várias aplicações disputam uma quantidade limitada de recursos. A técnica de DFS pode ser empregada neste contexto, alterando a frequência de operação dos PEs que executam tarefas de uma dada aplicação com requisitos de QoS a fim de atender tais requisitos.

7.1 Requisitos de QoS e *Profiling* de uma Aplicação

Uma aplicação pode ser modelada através de um grafo de tarefas, como exemplificado na Figura 31. Neste grafo, uma aplicação possui:

- Uma ou mais tarefas de entrada (T1 na Figura 31), responsáveis por adquirir os dados da aplicação e enviá-los às tarefas computacionais;
- *N* tarefas computacionais (T2, T3, e T4 na Figura 31), responsáveis por realizar o processamento da aplicação sobre os dados de entrada, e enviá-los para outra tarefa computacional ou para a tarefa de saída;
- Uma tarefa de saída (T5 na Figura 31), responsável por agregar todos os dados já processados na aplicação e disponibilizá-los em uma estrutura de saída de dados (memória de vídeo, interface de comunicação, canal de comunicação com outra aplicação, etc.).

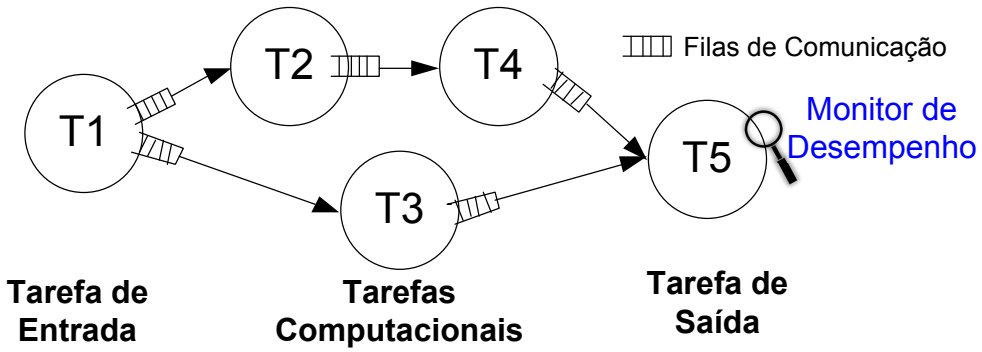


Figura 31 - Exemplo de aplicação modelada como um grafo de tarefas, sendo que a tarefa de saída é monitorada para realizar um controle de QoS.

Pode-se utilizar técnicas de *profiling*, em tempo de projeto, para verificar se os requisitos de QoS de uma aplicação podem ser atendidos pela plataforma. Monitores de

desempenho podem ser utilizados nesta etapa, avaliando os parâmetros de QoS da aplicação. Na etapa de *profiling* a aplicação tem sua execução simulada no MPSoC alvo, sem a execução de qualquer outra aplicação. Ao final desta simulação, a partir dos dados monitorados, define-se os parâmetros de QoS que serão utilizados em tempo de execução. Se na etapa de *profiling* a plataforma não atendeu os requisitos de QoS, a aplicação deve ser remodelada de forma a atender a estes requisitos. Os projetistas podem utilizar estes parâmetros para realizar o controle de QoS em tempo de execução, ou aplicar sobre estes uma margem de perda de desempenho aceitável para esta aplicação. Em tempo de execução, os requisitos de QoS estabelecidos na etapa de *profiling* deverão ser respeitados.

7.2 Adaptabilidade Utilizando DFS

A Seção 3.3 descreve o esquema de adaptabilidade aplicado à plataforma HeMPS, proposto em [ROS12b]. Neste esquema, a técnica de DFS é empregada na HeMPS a fim de realizar um balanceamento de carga nas filas de entrada dos PEs, reduzindo a dissipação de potência do MPSoC. O mecanismo de adaptabilidade desenvolvido nesta Tese de Doutorado estende este esquema, aplicando a técnica de DFS no controle de parâmetros de QoS das aplicações que executam na HeMPS.

O mecanismo de adaptabilidade desenvolvido nesta Tese de Doutorado pode ser interpretado como um laço fechado de controle, como mostra a Figura 32. O mecanismo compreende: (*i*) um sistema de monitoramento dos parâmetros de QoS avaliados; (*ii*) um sistema de avaliação dos parâmetros de QoS contra as referências obtidas na etapa de *profiling*; e (*iii*) um módulo de adaptabilidade, utilizado para realizar o controle dos parâmetros de QoS monitorados utilizando a técnica de DFS.

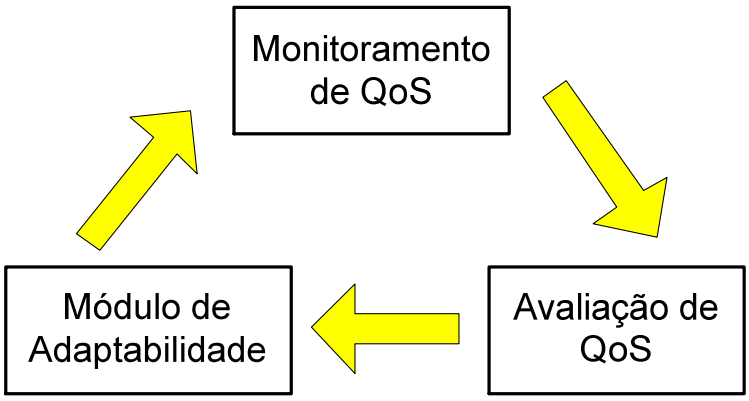


Figura 32 - Esquema conceitual do mecanismo de adaptabilidade proposto.

O sistema de monitoramento é responsável por aferir os parâmetros de QoS das aplicações, em tempo de execução, e enviá-los para o sistema de avaliação de QoS de uma aplicação ou tarefa. O sistema de monitoramento está descrito na Seção 7.2.1.

O sistema de avaliação de QoS pode ser aplicado nas tarefas individualmente ou na

aplicação como um todo, e é responsável por avaliar se esta aplicação/tarefa está atendendo aos seus requisitos de QoS. Ao verificar falhas de atendimentos aos requisitos de QoS, este sistema comanda o módulo de adaptabilidade que atua a fim de corrigir tais falhas. Este sistema está descrito na Seção 7.2.2.

O módulo de adaptabilidade é responsável por, utilizando a técnica de DFS, alterar o comportamento de uma dada tarefa. Este módulo aumenta ou diminui a frequência de operação do PE em que a tarefa está executando, acelerando ou diminuindo o desempenho desta. O módulo de adaptabilidade também é responsável por receber e propagar as mensagens de adaptação. Estas mensagens são capazes de atuar sobre os PEs os quais executam as tarefas de uma dada aplicação. O módulo de adaptabilidade está descrito na Seção 7.2.3.

7.2.1 Sistema de Monitoramento

O sistema de monitoramento desenvolvido na presente Tese de Doutorado é similar àquele descrito na Seção 2.1.3, porém foi modificado a fim de se ajustar ao mecanismo de adaptabilidade proposto neste trabalho. A Figura 33 exibe a arquitetura do sistema de monitoramento proposto.

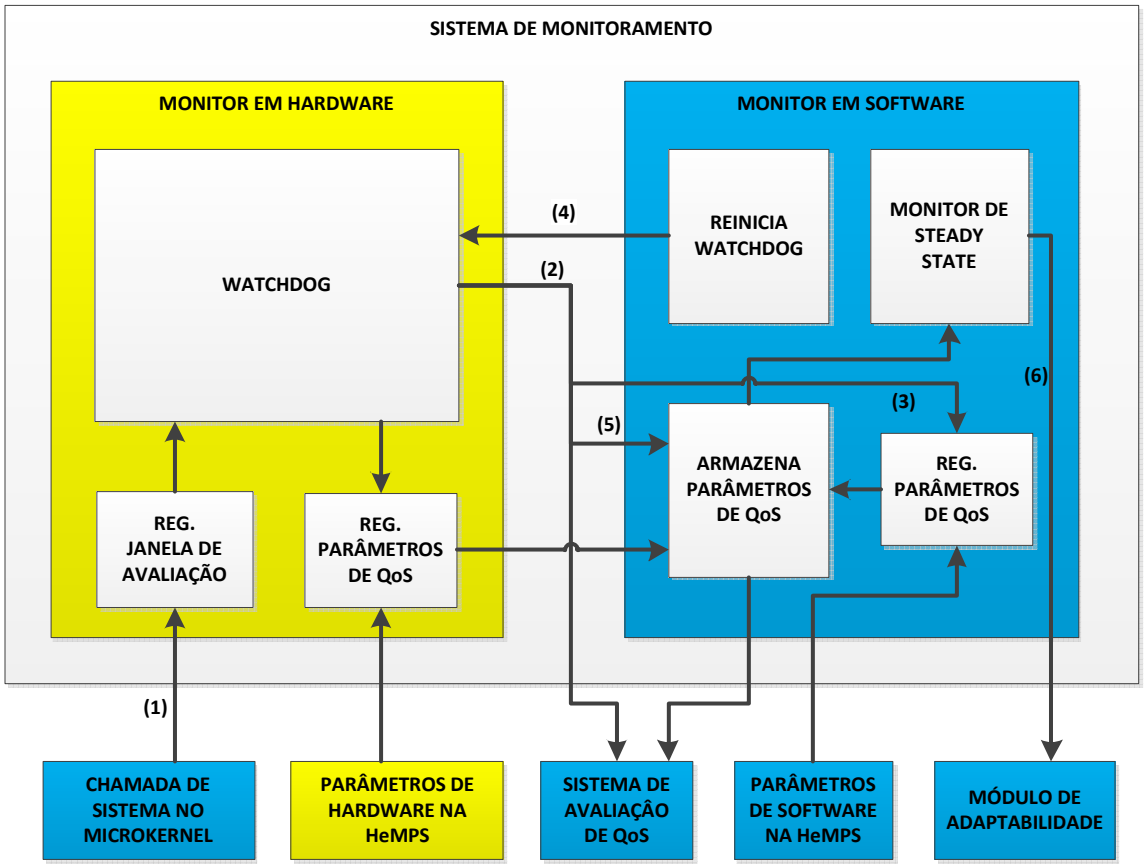


Figura 33 - Arquitetura do sistema de monitoramento proposto.

De acordo com o Capítulo 4, pode-se classificar o sistema de monitoramento como um monitoramento de desempenho híbrido, com monitores em hardware e em software.

O monitor em hardware é utilizado para sincronizar o tempo de amostragem dos parâmetros de QoS, além de registrar alguns parâmetros de QoS em hardware. A sincronização do monitor de hardware é realizada através de um temporizador do tipo *watchdog*, e sua janela de avaliação configurada por meio de uma chamada de sistema adicionada no *microkernel* (1 na Figura 33). Esta chamada de sistema é denominada *EnableMonitoring (task, eval_window)*, onde *task* é a tarefa que se comunica com a tarefa em execução na qual a comunicação deverá ser monitorada e *eval_window* é a janela de avaliação, em ciclos de relógio, do monitor, também utilizada para disparar pela primeira vez o temporizador *watchdog*. Ao atingir a janela de avaliação, o *watchdog* produz um sinal de interrupção para o processador que aciona a rotina do monitor em software (2 na Figura 33). O monitor em hardware pode ser utilizado sem o acionamento do monitor em software, a fim de realizar um traçado de perfil da aplicação e avaliar seus parâmetros de QoS.

O monitor em software, quando executado no processador, realiza a leitura de alguns parâmetros de QoS em software e os parâmetros de QoS capturados pelo monitor em hardware (3 na Figura 33). Este monitor é responsável, também, por reiniciar o temporizador *watchdog* para que este inicie uma nova avaliação (4 na Figura 33). Os parâmetros de QoS capturados são armazenados em memória, para que possam ser usados pelo sistema de avaliação de QoS (5 na Figura 33). O monitor em software possui uma rotina que avalia se o PE atingiu a estabilidade no algoritmo original de DFS (6 na Figura 33). Esta estabilidade é avaliada através da informação de frequência do PE, e é considerada atingida quando o PE repete sua frequência mais de *N* janelas de avaliação consecutivas (valor definido em tempo de projeto). O monitor de estabilidade descarta algumas janelas de avaliação, que são utilizadas na inicialização do PE. Este número de janelas a serem descartadas é também configurado em tempo de projeto. A aplicabilidade da rotina de detecção de estabilidade é descrita na Seção 7.3.

A principal desvantagem do monitor em software é o fato de ele estar ligado a cada comunicação realizada em cada tarefa da aplicação. Desta forma, se as tarefas de uma aplicação possuírem um perfil de muita comunicação e pouca computação os monitores serão ativados diversas vezes, e cada ativação o tratamento do monitor será executado. Estas execuções dos monitores inserem uma carga computacional adicional no PE, o que pode aumentar seu tempo de execução e seu consumo de energia.

Para fins de sincronização no nível de MPSoC, o temporizador *watchdog*, presente no monitor em hardware, utiliza o sinal de relógio geral do MPSoC e não aquele que será gerado pelo módulo de adaptabilidade utilizando a técnica de DFS. Desta forma, todos os monitores, em todos os PEs, possuem a mesma referência de tempo.

7.2.2 Sistema de Avaliação de QoS

O sistema de avaliação de QoS desenvolvido na presente Tese de Doutorado é

baseado naquele descrito na Seção 2.1.3, porém foi modificado a fim de se ajustar ao mecanismo de adaptabilidade proposto neste trabalho. A Figura 34 exibe a arquitetura do sistema de avaliação de QoS proposto.

O sistema de avaliação de QoS é totalmente implementado em software, e é executado em conjunto com o monitor em software durante o tratamento da interrupção gerada pelo monitor em hardware. Este sistema é composto por: (*i*) um módulo avaliador de violações de QoS; (*ii*) registradores para armazenar os parâmetros de QoS de referência; e (*iii*) pelo leitor dos parâmetros de QoS.

Os parâmetros de QoS de referência são recebidos através da chamada de sistema *Set<QoS_Parameter> (task, min_threshold, max_threshold)* adicionada no *microkernel*, onde *task* é a tarefa a qual será monitorado o QoS, *min_threshold* e *max_threshold* são os limiares mínimo e máximo do parâmetro de QoS monitorado. Estes limiares são utilizados para que o controle da qualidade de serviço de uma aplicação possa ser realizado sobre um intervalo específico de valores, sendo que o valor máximo pode ser suprimido dependendo do parâmetro de QoS monitorado. Esta chamada de sistema também aciona o módulo avaliador de violações de QoS, que passa a avaliar o parâmetro de QoS monitorado.

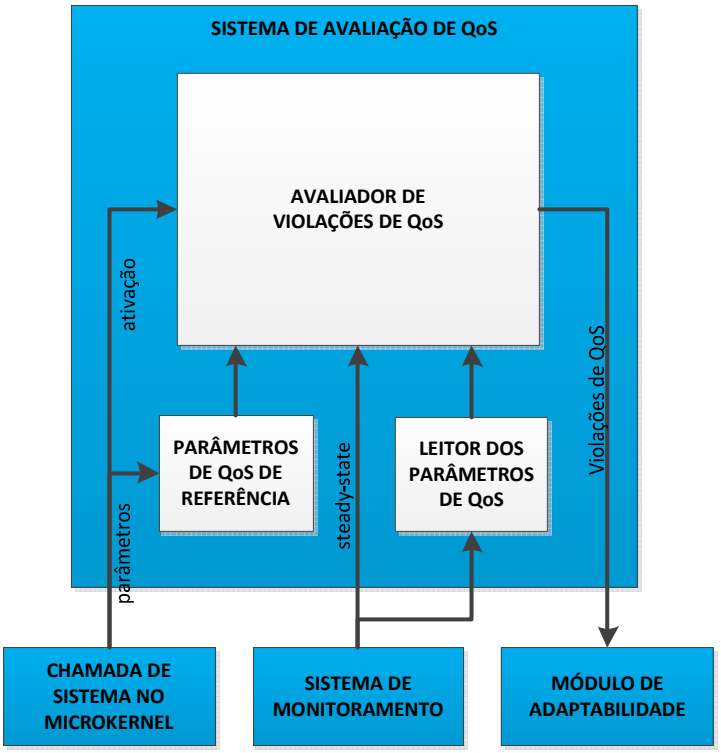


Figura 34 - Arquitetura do sistema de avaliação de QoS proposto.

O leitor dos parâmetros de QoS apenas realiza uma leitura de memória, onde estão armazenados os parâmetros de QoS avaliados pelo sistema de monitoramento. O leitor disponibiliza estes parâmetros de QoS para o módulo de avaliação de violações de QoS.

O módulo de avaliação de violações de QoS somente é ativado quando o sistema de

monitoramento é ativado e quando o PE atingir um estado de estabilidade na troca de frequência. Este módulo realiza uma comparação dos parâmetros de QoS obtidos pelo leitor dos parâmetros de QoS, com os valores de referência armazenados no armazenador dos parâmetros de QoS de referência. O valor obtido pelos monitores é comparado com os limiares mínimo e máximo de referência. Quando o valor obtido é menor que o limiar mínimo de referência, diz-se que ocorreu uma violação de QoS para baixo. Por outro lado, quando o valor obtido é maior que o limiar máximo de referência, diz-se que ocorreu uma violação de QoS para cima. Estas violações são contabilizadas, e quando ultrapassam um limite (estabelecido em tempo de projeto), o módulo de adaptabilidade é acionado e a contagem de violações é zerada. O módulo de avaliação de violações de QoS repassa para o módulo de adaptabilidade qual comportamento deve ser corrigido, ou seja, violações para baixo ou para cima.

7.2.3 Módulo de Adaptabilidade

O módulo de adaptabilidade desenvolvido na presente Tese de Doutorado é baseado no controlador de DFS descrito na Seção 3.3, porém foi modificado a fim de se ajustar ao mecanismo de adaptabilidade proposto neste trabalho. A Figura 35 exibe a arquitetura do módulo de adaptabilidade proposto.

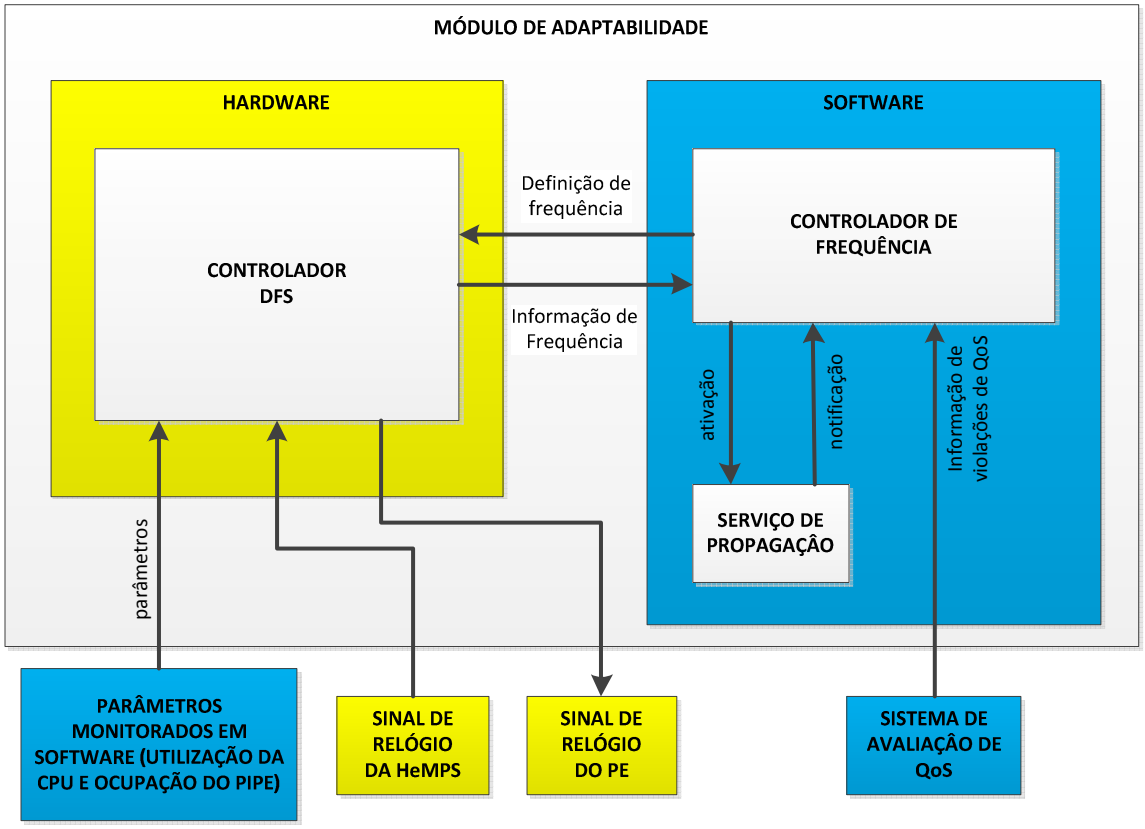


Figura 35 - Arquitetura do módulo de adaptabilidade proposto, onde em amarelo estão os módulos em hardware e em azul os módulos em software.

O módulo de adaptabilidade é composto por uma parte em hardware e outra em software. A parte em hardware é basicamente constituída pelo controlador DFS, e alguns

registradores que realizam a interface mapeada na memória do processador para a troca de política do controlador DFS. O controlador DFS teve sua estrutura modificada, a fim de aceitar o comando de troca de frequências realizadas pelo software do módulo de adaptabilidade. Assim sendo, o controlador DFS pode utilizar tanto a política de controle do DFS original, descrito na Seção 3.3, quanto à política de controle do DFS realizada através do software do módulo de adaptabilidade.

A parte do módulo de adaptabilidade desenvolvida em software é composta por uma rotina de controle da frequência do PE e um serviço que realiza a propagação da adaptação em todas as tarefas que compõem aplicação. A rotina de controle da frequência é acionada através do sistema de avaliação de QoS ou pelo serviço de propagação, e recebe a informação de qual mudança de frequência deverá ser realizada (aumento ou diminuição). Quando acionada, esta rotina envia um comando de troca de frequência para o controlador DFS, que então efetua a troca de frequência. A seguir, a rotina de controle da frequência aciona o serviço de propagação informando qual frequência foi comandada ao controlador DFS.

O serviço de propagação é acionado de duas formas, pela rotina de controle de frequência e pelo recebimento de uma mensagem de adaptação. Quando acionada pela rotina de controle da frequência, este serviço é responsável por enviar uma mensagem de adaptação para as tarefas que se comunicam com a tarefa que disparou o processo de adaptação. Neste caso, o serviço de propagação adiciona um identificador único na mensagem de adaptação. Este identificador é utilizado no descarte de mensagens de adaptação duplicados. Quando este serviço é acionado através do recebimento de uma mensagem de adaptação, ele é responsável por executar a rotina de controle da frequência, passando como parâmetro a informação de qual mudança de frequência deverá ser realizada. A Figura 36 exibe um exemplo de atuação deste módulo em diversos PEs.

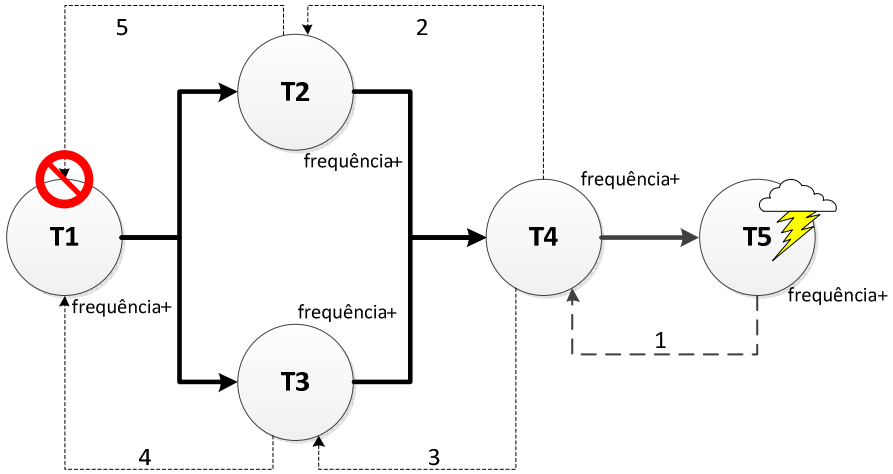


Figura 36 - Exemplo da atuação do mecanismo de adaptabilidade em uma aplicação. As linhas contínuas representam o fluxo de mensagens da aplicação, e as pontilhadas representam o fluxo das mensagens de adaptação.

Neste exemplo, somente a tarefa T5 está com o sistema de monitoramento ativado, ou seja, esta tarefa possui requisitos de QoS, que representam os requisitos de QoS da aplicação. No decorrer da execução da aplicação, o sistema de avaliação de QoS detecta que a tarefa T5 está perdendo desempenho (várias violações de QoS consecutivas). A partir deste momento este sistema comanda o aumento da frequência na aplicação. Ao término do aumento de frequência local o serviço de propagação é acionado, enviando uma mensagem de adaptação comandando o aumento da frequência para os demais PEs que se comunicam com a tarefa T5, ou seja, somente a tarefa T4. Esta mensagem irá possuir um identificador único, por exemplo o número 1.

A tarefa T4 recebe a mensagem de adaptação e o serviço de propagação verifica a existência de um comando de aumento da frequência, e compara o identificador que acompanha a mensagem com seu contador de mensagens. Se o identificador é maior que o contador local significa que esta mensagem é única, desta forma o módulo de adaptabilidade que cumprirá o comando de aumento da frequência é acionado neste PE. Ao final do aumento da frequência, o serviço de propagação é acionado e identifica as tarefas T.e T3 como alvo das novas mensagens de adaptação, com o mesmo identificador recebido na mensagem recebida. Este serviço envia as mensagens de adaptação para os PEs que executam as tarefas T2 e T3. Esse mesmo processo é realizado nas tarefas T2 e T3, que enviam mensagens de adaptação para o PE que executa a tarefa T1. Notar que este processo é executado ao se receber a mensagem de adaptação, sem verificação da eventual não necessidade de continuar o processo às tarefas filhas. Pode ocorrer que a adaptação apenas em algumas tarefas seja suficiente. Entretanto, dado o custo de monitoração e o tempo para esta verificação, optou-se por sempre enviar as mensagens de adaptação.

O serviço de propagação do PE que executa a tarefa T1 recebe duas mensagens de adaptação, e estas duas possuem o mesmo comando de aumento da frequência. A cada recebimento o serviço de propagação aumenta seu contador de mensagens, e verifica se a mensagem recebida ainda possui um identificador menor que o seu contador. Neste caso, a T1 irá receber a primeira mensagem com aumento de frequência, verificar que o identificador é maior que seu contador (que é zero) e irá executar o comando de aumento de frequência. Ao término da execução do comando de aumento de frequência, o seu contador local é incrementado. Quando a segunda mensagem de adaptabilidade é recebida, com o mesmo comando de aumento de frequência, o identificador recebido na mensagem não é maior que o contador local da tarefa T1. Neste caso a segunda mensagem é descartada, sem que o comando de aumento de frequência seja executado.

7.3 Adaptabilidade no Controle da Vazão das Aplicações na HeMPS

O mecanismo de controle da vazão das aplicações na HeMPS desenvolvido no escopo desta Tese de Doutorado, combina o controle da frequência via DFS descrito no

Capítulo 3 e o esquema de adaptabilidade descrito na Seção 7.2. Neste mecanismo, a vazão das aplicações é verificada nas tarefas de saída das aplicações (Figura 31).

Na plataforma HeMPS, as tarefas de uma aplicação se comunicam através de troca de mensagens, utilizando as primitivas *SEND* e *RECEIVE*. Como descrito na Seção 2.2.6, as primitivas de *SEND*, não-bloqueantes, inserem mensagens nas filas de comunicação, enquanto que as primitivas *RECEIVE*, bloqueantes, realizam leituras das filas de comunicação. Este esquema de comunicação reduz o tráfego na NoC, já que uma mensagem só é inserida na rede quando é requisitada. Se as filas de comunicação forem colocadas no lado do receptor, as mensagens podem bloquear a NoC quando as filas ficarem totalmente cheias.

No mecanismo proposto, a frequência na HeMPS é controlada desde o início de sua operação utilizando uma política que procura manter uniforme a ocupação das filas de entrada nos PEs, e assim reduzindo a dissipação de potência global da plataforma [ROS12b]. O objetivo geral desta política é gerenciar as filas de comunicação, trazendo-as a um nível de ocupação médio onde os PEs sempre encontrarão dados a serem consumidos quando requisitados. Esta política realiza diversas trocas de frequência nos PEs de uma aplicação até atingir um estado de estabilidade, onde a frequência é praticamente mantida estável. Ao atingir a estabilidade, as aplicações que ativaram o monitoramento de vazão, irão avaliar a violação do cumprimento deste parâmetro. No caso de uma sequência de violações de vazão, o PE troca a política de controle de DFS ajustando a frequência do PE de acordo com o esquema de adaptabilidade descrito na Seção 7.2.

O mecanismo de controle da vazão é acionado através de duas chamadas de sistema, *EnableMonitoring(task, eval_window)* e *SetThroughputQoS(task, min_threshold, max_threshold)* inseridas na aplicação, como exibe a Figura 37. A primeira chamada de sistema ativa e configura o mecanismo de monitoramento como descrito na Seção 7.2.1, já a segunda ativa e configura o sistema de avaliação de QoS como descrito na Seção 7.2.2.

O requisito de vazão de uma aplicação pode deixar de ser atendido devido as seguintes situações: (*i*) a vazão obtida ao se atingir o estado de estabilidade é inferior àquela definida como requisito; ou (*ii*) tráfegos de outras aplicações interferem na comunicação entre as tarefas da aplicação monitorada, reduzindo a vazão na tarefa de saída. A vazão pode ser também superior àquela especificada como limiar superior, o que permite a redução na frequência das tarefas da aplicação. O sistema de avaliação de QoS executa o comportamento descrito na Seção 7.2.2, aumentando a frequência se muitas violações de vazão são detectadas, ou reduzindo a frequência quando a vazão monitorada é superior a um limiar específico. Este comportamento garante que os requisitos de vazão da aplicação sejam atendidos, mantendo-se uma baixa dissipação de potência nos PEs que executam as tarefas da aplicação.

```
#include <task.h>
#include <stdlib.h>

Message msg;

int main()
{

  int i,t;
  EnableMonitoring(taskE_0, 10000);
  SetThroughputQoS(taskE_0, 192, 400);

  Echo("synthetic0 task F started.");
  Echo(itoa(GetTick()));

  for(i=0;i<50000;i++){
    for(t=0;t<500;t++){
    }
    Receive(&msg, taskE_0);
  }
  Echo(itoa(GetTick()));
  Echo("synthetic0 task F finished.");

  exit();

}
```

Figura 37 - Exemplo da tarefa final de uma aplicação com controle e monitoramento da vazão.

É importante salientar que ao atingir o estado de estabilidade, os PEs podem estar executando as tarefas de uma dada aplicação com diferentes frequências. Para evitar o uso de um gerente global, responsável por manter o estado de cada um destes PEs, uma estratégia de gerência de frequência distribuída foi utilizada, sendo implementada pelo módulo de adaptabilidade (como descrito na Seção 7.2.3). Esta estratégia utiliza mensagens de adaptação, que comandam as trocas de frequência nos PEs de uma dada aplicação.

7.4 Adaptabilidade no Controle da Latência das Aplicações na HeMPS

De forma equivalente ao controle de vazão nas aplicações da HeMPS descrito na Seção 7.3, o uso combinado do controle da frequência via DFS descrito no Capítulo 3 e o esquema de adaptabilidade descrito na Seção 7.2 foi utilizado para o desenvolvimento do mecanismo de controle da latência das aplicações na HeMPS desta Tese de Doutorado. Diferentemente do controle de vazão, a latência é verificada em cada par de tarefas comunicantes. Se em uma comunicação o requisito mínimo de latência não for atendido em cinco janelas de avaliação consecutivas (quantidade parametrizável), o mecanismo de adaptabilidade é acionado aumentando a frequência dos PEs anteriores às duas tarefas comunicantes.

O mecanismo de controle da latência é acionado através de duas chamadas de sistema, *EnableMonitoring(task, eval_window)* e *SetLatencyQoS(task, min_threshold)* inseridas na aplicação, como exibe a Figura 38. A primeira chamada de sistema ativa e configura o mecanismo de monitoramento como descrito na Seção 7.2.1, já a segunda ativa e configura o sistema de avaliação de QoS como descrito na Seção 7.2.2.

```
#include <task.h>
#include <stdlib.h>

Message msg;

int main()
{

  int i,t;
  EnableMonitoring(taskE_1, 10000);
  EnableMonitoring(taskD_1, 10000);
  //SetThroughputQoS(taskE_1, 190, 400);
  //SetThroughputQoS(taskD_1, 190, 400);
  SetLatencyQoS(taskE_1, 300);
  SetLatencyQoS(taskD_1, 300);

  Echo("synthetic0 task F started.");
  Echo(itoa(GetTick()));

  for(i=0;i<100;i++){
    for(t=0;t<50;t++);
    Receive(&msg, taskD_1);
    Receive(&msg, taskE_1);
  }
  Echo(itoa(GetTick()));
  Echo("synthetic0 task F finished.");

  exit();

}
```

Figura 38 - Exemplo de uma aplicação com controle da latência através de DFS.

Para realizar a medição de latência em uma comunicação entre as tarefas A e B (onde a tarefa A envia dados para a tarefa B) utiliza-se mensagens de avaliação de latência antes e após a comunicação entre as duas tarefas. Antes de iniciar a comunicação, a tarefa A envia uma pequena mensagem de avaliação de latência com o valor do relógio comum (o mesmo utilizado pelo monitoramento em hardware), e após a comunicação a tarefa A envia outra mensagem de latência com o valor do relógio comum.

Ao receber uma mensagem de latência, a tarefa B decrementa o valor do seu relógio comum do valor recebido. Isto permite avaliar somente o tempo gasto na transmissão dos pacotes de comunicação somando-se as duas mensagens de latência. Ao receber a segunda mensagem de latência, a tarefa B realiza o mesmo decremento da primeira mensagem. De posse destes dois tempos, a tarefa B calcula a latência na comunicação diminuindo-se o tempo inicial do tempo final. A Tabela 10 abaixo sumariza os passos realizados pelas tarefas A e B para se avaliar a latência na comunicação de A para B.

Se a latência avaliada na tarefa B é superior àquela introduzida como parâmetro de QoS, considera-se que ocorreu uma violação de QoS. Se o número de violações de QoS for superior ao limiar determinado em tempo de projeto, o mecanismo de adaptabilidade é executado a fim de aumentar a frequência dos PEs das tarefas.

Tabela 10 - Atores e ações no cálculo da latência na comunicação entre duas tarefas.

Ator da Ação	Ação Realizada	Ordem
Tarefa A	Envio da mensagem de latência com o valor do relógio comum ($R_{inicial}$)	1
Tarefa B	Recebimento da mensagem de latência e decremento do valor recebido pelo seu relógio local (*Início* = $R_{local} - R_{inicial}$)	2
Tarefa A	Envio dos pacotes referentes aos dados a serem transmitidos	3
Tarefa B	Recebimento dos pacotes referentes aos dados a serem recebidos	4
Tarefa A	Envio da mensagem de latência com o valor do relógio comum (R_{final})	5
Tarefa B	Recebimento da mensagem de latência e decremento do valor recebido pelo seu relógio local (*Final* = $R_{local} - R_{final}$)	6
Tarefa B	Cálculo da latência na transmissão dos dados (*Lat = Final – Inicial*)	7

7.5 Conclusão

Este Capítulo descreveu a principal contribuição desta Tese de Doutorado, o mecanismo de controle de parâmetros de QoS utilizando a técnica de DFS nos PEs do MPSoC. Este mecanismo foi desenvolvido com o foco na plataforma HeMPS, porém ele pode ser aplicado em outras plataformas realizadas as devidas modificações.

O modelo idealizado possui uma limitação de sua implementação, que é o fato dos monitores em software estarem ligados a cada comunicação entre as tarefas da aplicação monitorada. Isto leva a um aumento da carga computacional nos PEs, o que irá gerar aumentos no tempo de execução das tarefas e no consumo de energia dos PEs que compõem a aplicação.

Foram utilizados como exemplos de parâmetros de QoS a vazão da aplicação e a latência das tarefas desta aplicação. Outros parâmetros podem ser avaliados, desde que um monitor específico para estes parâmetros seja desenvolvido, e que o sistema de controle de QoS avalie as violações deste novo parâmetro disparando o mecanismo de adaptabilidade.

8 PLATAFORMA E CENÁRIOS DE TESTES AVALIADOS

Este Capítulo descreve a plataforma HeMPS utilizada para avaliar o mecanismo de adaptabilidade desenvolvido (Seção 8.1), as aplicações avaliadas (Seção 8.2), os cenários de teste executados (Seção 8.3) e as métricas de QoS exploradas (Seções 8.4 e 8.5).

8.1 Plataforma HEMPS Utilizada

A plataforma HeMPS utilizada para se avaliar o mecanismo de controle de vazão e latência desenvolvido nesta Tese de Doutorado, possui as seguintes características:

- HeMPS 4 x 4, sendo 1 PE mestre e 15 PEs escravos;
- 16 KB de tamanho da página dos processadores;
- Máximo de duas tarefas por escravo;
- 64 KB de memória por PE;
- 100 MHz de frequência de operação na HeMPS.

Esta plataforma foi modelada utilizando descrição em código VHDL no nível RTL (*Register Transfer Level*) para a maioria dos componentes do MPSoC, exceto o processador, o qual é modelado com um ISS (*Instruction Set Simulator*). O ISS possui precisão de ciclo de relógio, tendo precisão igual à descrição VHDL [PET12]. O uso de ISS justifica-se pelo menor tempo de simulação. A plataforma foi simulada com a ferramenta MODELSIM da empresa MentorGraphics. Quatro configurações da plataforma foram avaliadas:

- *HeMPS*: HeMPS com frequência nominal (100MHz);
- *HeMPS-D*: HeMPS com DFS;
- *HeMPS-DQ*: HeMPS com DFS e controle de QoS;
- *HeMPS-DQd*: HeMPS com DFS, controle de QoS e descarte de mensagens de adaptabilidade.

8.2 Aplicações Avaliadas

Três aplicações tiveram seus parâmetros de vazão e latência controlados, duas aplicações sintéticas e uma aplicação real.

A primeira aplicação sintética, denominada de *SYNTH0*, é uma aplicação *pipeline* com seis estágios. O grafo desta aplicação é apresentado na Figura 39. Cada tarefa desta aplicação possui um laço que irá emular o tempo de computação gasto por cada tarefa. Este laço possui valores de iteração que variam entre 12 e 125, estes valores foram utilizados para que as tarefas executassem um tempo de computação diferente. Esta aplicação foi utilizada nos primeiros testes do mecanismo, e seus resultados foram

atualizados a cada nova modificação do mecanismo de controle do QoS na HeMPS.

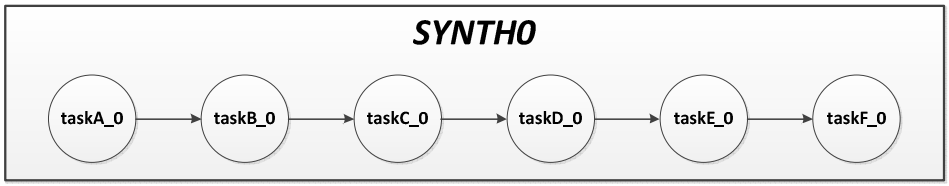


Figura 39 - Grafo das tarefas da aplicação *SYNTH0*.

A segunda aplicação sintética, denominada de *SYNTH1*, é uma aplicação com diversas bifurcações (*forks*) e sincronizações (*joins*). O grafo desta aplicação é apresentado na Figura 40. Da mesma forma que a aplicação *SYNTH0*, as tarefas desta aplicação possuem laços que irão emular o tempo de computação gasto em cada uma. Estes laços também possuem de 12 a 125 iterações, fazendo com que as tarefas possuam tempos de computação diferentes. Esta aplicação foi utilizada para a validação do descarte de mensagens de adaptação.

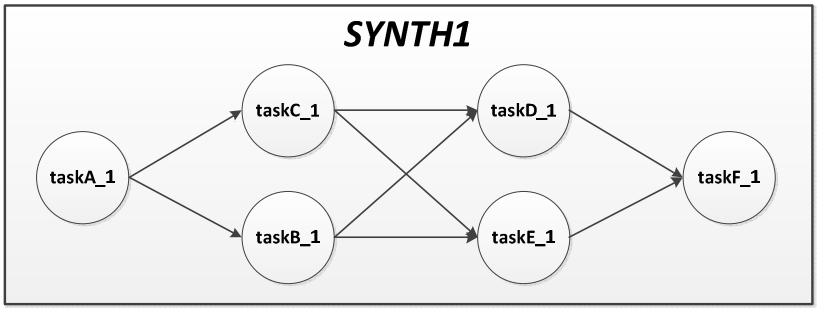


Figura 40 - Grafo das tarefas da aplicação *SYNTH1*.

A aplicação real utilizada é uma aplicação que implementa o algoritmo do MPEG2. Esta aplicação está disponível na ferramenta de geração automática da plataforma HeMPS. O grafo da aplicação *MPEG2* é apresentado na Figura 41. Esta aplicação possui um vetor de entrada com 128 posições, que representa uma parte de uma imagem a ser tratada. Esta aplicação foi utilizada a fim de avaliar o funcionamento do controle de parâmetros de QoS em aplicações reais.

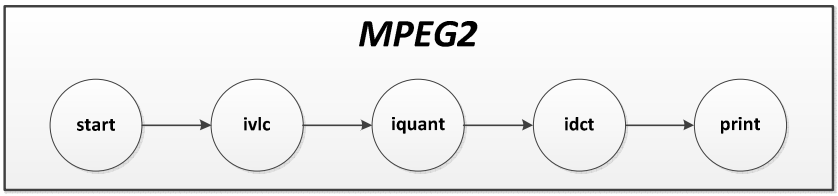
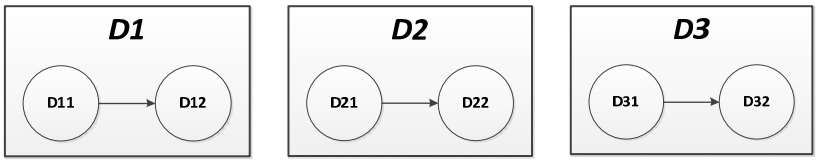


Figura 41 - Grafo das tarefas da aplicação *MPEG2*.

Além destas aplicações, três outras aplicações sintéticas foram utilizadas como fonte de ruído na plataforma. Estas aplicações são denominadas de *D1*, *D2* e *D3*, e são aplicações com apenas duas tarefas que somente se comunicam. Estas aplicações foram introduzidas no caminho de comunicação das tarefas das aplicações *SYNTH0*, *SYNTH1* e *MPEG2*, visando influenciar a execução destas aplicações.



(a) (b) (c)

Figura 42 - Grafo das tarefas das aplicações (a) *D1*, (b) *D2* e (c) *D3*.

8.3 Cenários de Teste Avaliados

Quatro cenários de teste foram simulados:

1. A aplicação controlada (*SYNTH0*, *SYNTH1* ou *MPEG2*) sendo executada sozinha na plataforma HeMPS. Este cenário é utilizado para realizar um *profiling* desta aplicação, a fim de se extrair o seu requisito de vazão. Como exemplo, a Figura 43 exibe as tarefas de A até F da aplicação *SYNTH0*;
2. A aplicação controlada sendo executada juntamente com a aplicação *D1* (tarefas D11 e D12 na Figura 43);
3. A aplicação controlada sendo executada juntamente as aplicações *D1* e *D2* (tarefas D11, D12, D21 e D22 na Figura 43);
4. A aplicação controlada sendo executada juntamente com as aplicações *D1*, *D2* e *D3* (tarefas D11, D12, D21, D22, D31 e D32 na Figura 43).

A Figura 43 exibe o mapeamento das tarefas dos cenários de teste descritos acima, as setas mostram o fluxo de mensagens geradas pelas tarefas de cada aplicação. Nos cenários de teste acima descritos, a janela de avaliação do sistema de monitoramento é de 10.000 ciclos de relógio, ou seja, uma avaliação a cada 0,1 ms.

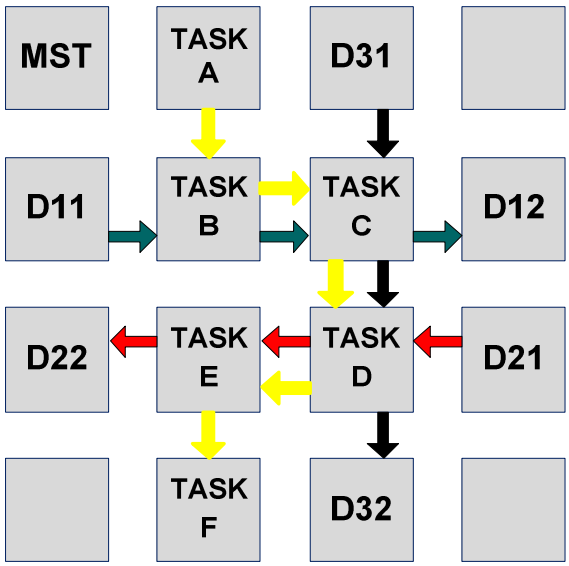


Figura 43 - Mapeamento das tarefas dos cenários de teste avaliados. As tarefas A até F correspondem à aplicação SYNTH0. As tarefas D10, D11, D21, D21, D31, D32 correspondem às tarefas das aplicações D1, D2 e D3, respectivamente.

O tempo de simulação para todos os cenários de teste é variável e depende do final da execução da aplicação avaliada. Este tempo de simulação é avaliado utilizando-se o

sinal de relógio global para todos os PEs, desta maneira o referencial de tempo será o mesmo para todas as plataformas independente do mecanismo de DFS. Entretanto, os tempos das violações de QoS são calculados com base nos sinais de relógio de cada PE, e estão sujeitos as trocas de frequência realizadas pelo mecanismo de DFS.

8.4 Controle de Vazão das Aplicações Controladas

Nas avaliações de controle da vazão, as aplicações controladas (*SYNTH0*, *SYNTH1* e *MPEG2*) configuram os limiares mínimo e máximo de vazão da seguinte maneira:

- *SYNTH0* → vazão mínima de 380 bits/ciclo e máxima de 600 bits/ciclo;
- *SYNTH1* → vazão mínima de 190 bits/ciclo e máxima de 400 bits/ciclo;
- *MPEG2* → vazão mínima de 192 bits/ciclo e máxima de 400 bits/ciclo.

O mecanismo de controle da vazão proposto, quando habilitado, é configurado em tempo de projeto com os seguintes parâmetros:

- *STEADY_STATE_WINDOW = 13* – Este é o número de janelas de avaliação consecutivas com a mesma frequência, utilizada para determinar o estado de estabilidade do PE;
- *PE_INIT_WINDOW = 5* – Este é o número de janelas de avaliação desconsiderado pelo monitor de estabilidade durante a inicialização do PE;
- *MIN_THROUGHPUT_PERMITTED = 5* – Este é o número máximo de violações do limiar mínimo de vazão permitido;
- *MAX_THROUGHPUT_PERMITTED = 5* – Este é o número máximo de violações do limiar máximo de vazão permitido.

Os quatro cenários de teste (descritos na Seção 8.3) foram executados para cada uma das três aplicações controladas (descritas na Seção 8.2), utilizando-se das quatro possíveis configurações de plataforma HeMPS (descritas na Seção 8.1) avaliando-se o controle da vazão das mesmas.

8.5 Controle da Latência nas Aplicações Controladas

Nas avaliações de controle da latência, as aplicações controladas (*SYNTH0*, *SYNTH1* e *MPEG2*) configuram os limiares mínimos de latência da seguinte maneira:

- *SYNTH0* → latência mínima de 350 ciclos de relógio;
- *SYNTH1* → latência mínima de 300 ciclos de relógio para as tarefas *taskE_1* e *taskD_1*;
- *MPEG2* → latência mínima de 300 ciclos de relógio.

O mecanismo de controle da latência proposto, quando habilitado, é configurado em tempo de projeto com os seguintes parâmetros:

- *STEADY_STATE_WINDOW = 13* – Este é o número de janelas de avaliação

consecutivas com a mesma frequência, utilizada para determinar o estado de estabilidade do PE;

- *PE_INIT_WINDOW = 5* – Este é o número de janelas de avaliação desconsiderado pelo monitor de estabilidade durante a inicialização do PE;
- *MIN_LATENCY_PERMITTED = 5* – Este é o número máximo de violações do limiar mínimo de vazão permitido.

Os quatro cenários de teste (descritos na Seção 8.3) foram executados para cada uma das três aplicações controladas (descritas na Seção 8.2), utilizando-se das quatro possíveis configurações de plataforma HeMPS (descritas na Seção 8.1) avaliando-se o controle da latência das mesmas.

8.6 Conclusão

Neste Capítulo foram definidas as aplicações, as configurações da plataforma HeMPS e os cenários de teste executados para a avaliação do controle de vazão e latência utilizando-se o mecanismo proposto nesta Tese de Doutorado.

Os resultados obtidos nestas avaliações são descritos nos Capítulos 9 (para avaliação de vazão) e 10 (para avaliação de latência), respectivamente.

9 RESULTADOS OBTIDOS PARA AVALIAÇÃO DE VAZÃO

Neste Capítulo são apresentados os resultados obtidos durante a execução de todas as simulações descritas no Capítulo 8 para o controle da vazão das aplicações. Os resultados estão assim organizados:

(i) a Seção 9.1 exibe os resultados obtidos nas simulações com a aplicação SYNTH0;

(ii) a Seção 9.2 exibe os resultados obtidos nas simulações com a aplicação MPEG2;

(iii) a Seção 9.3 exibe os resultados obtidos nas simulações com a aplicação SYNTH1;

Os resultados de vazão das aplicações são medidos na tarefa final de cada aplicação, e as avaliações de consumo de energia das aplicações são o somatório do consumo de energia de todos os PEs que executam as tarefas da aplicação até o final da execução desta.

Ao início de cada Seção é apresentada uma tabela com o resumo dos resultados obtidos, e ao final do Capítulo é apresentada uma breve conclusão sobre os resultados obtidos.

9.1 Resultados da aplicação SYNTH 0

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação SYNTH0. O grafo de tarefas desta aplicação está descrito na Figura 39. Esta aplicação foi simulada utilizando-se três versões da plataforma HeMPS, a primeira é a plataforma original sem qualquer mecanismo de controle de frequência ou QoS – HeMPS; a segunda é a plataforma com o mecanismo de DFS controlado pelo preenchimento do canal de comunicação (*pipe*) – HeMPS-D; e a terceira é a plataforma com controle da vazão da aplicação utilizando o mecanismo de DFS – HeMPS-DQ.

A Tabela 11 exibe um resumo dos resultados obtidos nas simulações acima descritas. Os requisitos de vazão para esta aplicação foram um mínimo de 380 bits/ciclo, e máximo de 600 bits/ciclo (como descrito no Capítulo 8, Seção 8.4). Nesta tabela pode-se verificar que a HeMPS sempre atende os requisitos de QoS impostos no menor tempo de execução possível, porém esta plataforma possui o maior consumo de energia na execução da aplicação. As simulações na HeMPS-D exibem reduções de até 44% no consumo de energia e aumentos no tempo de execução de até 123% (este aumento elevado no tempo de execução deve-se à natureza da aplicação, a qual possui muito mais comunicação que computação). Os requisitos de vazão da aplicação não foram respeitados. Já os resultados obtidos na HeMPS-DQ mostram reduções no consumo de energia de até 61% com os requisitos de vazão atendidos, porém o tempo de simulação foi até 80% maior que a HeMPS.

Tabela 11 - Resumo dos resultados das simulações utilizando a aplicação SYNTH0.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	140.727	94,72	-	-	Sim
	HeMPS-D	274.698	52,29	95,20%	-44,79%	Não
	HeMPS-DQ	**246.089**	**36,48**	**74,87%**	**-61,48%**	**Sim**
Com 1 interferência	HeMPS	140.927	95,64	-	-	Sim
	HeMPS-D	315.166	75,16	123,64%	-21,41%	Não
	HeMPS-DQ	**254.705**	**38,77**	**80,74%**	**-59,46%**	**Sim**
Com 2 interferências	HeMPS	147.177	100,38	-	-	Sim
	HeMPS-D	316.778	75,90	115,24%	-24,38%	Não
	HeMPS-DQ	**264.738**	**38,56**	**79,88%**	**-61,59%**	**Sim**
Com 3 interferências	HeMPS	167.529	108,93	-	-	Sim
	HeMPS-D	279.548	68,09	66,87%	-37,49%	Não
	HeMPS-DQ	**255.904**	**58,18**	**52,75%**	**-46,59%**	**Sim**

Adicionalmente, comparando-se os resultados obtidos entre as simulações na HeMPS-D e na HeMPS-DQ mostram reduções de até 19% no tempo de execução e de até 49% no consumo de energia na plataforma HeMPS-DQ.

A Seção 9.1.1 detalha os resultados obtidos na simulação da aplicação SYNTH0 sobre a plataforma HeMPS, a Seção 9.1.2 detalha os resultados nas simulações sobre a plataforma HeMPS-D e a Seção 9.1.3 detalha os resultados obtidos nas simulações sobre a plataforma HeMPS-DQ.

9.1.1 Simulações com a Plataforma HeMPS

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

9.1.1.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS, sem a interferência de quaisquer outras aplicações. A Figura 44 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste permanece a maior parte do tempo em 384 bits/ciclo com alguns picos de 576 bits/ciclo.

A avaliação do consumo de energia da aplicação resultou em 94,72 mJ e um tempo de execução total de 140.727 ciclos. Os componentes do consumo de energia desta aplicação são: 94,12 mJ para os processadores, 0,6 mJ para os roteadores e 0,0039 mJ para os fios de interconexão.

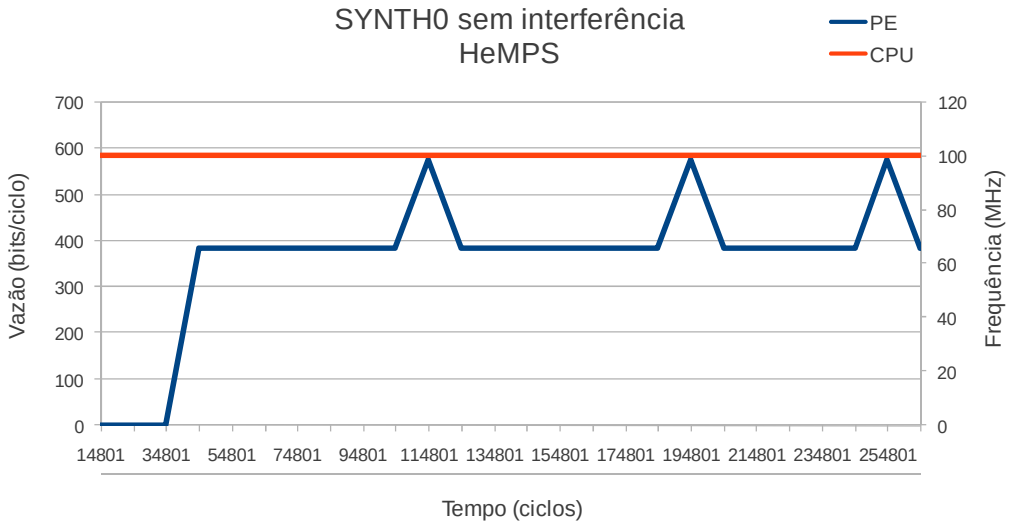


Figura 44 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS.

Os resultados de vazão descritos acima foram utilizados para determinar o perfil de QoS para a aplicação SYNTH0. A partir deste perfil de QoS, pode-se determinar os parâmetros de QoS que deverão ser respeitados nas simulações com a plataforma HeMPS-DQ.

9.1.1.2 Cenário de Teste 2

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS juntamente com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 45 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste permanece a maior parte do tempo em 384 bits/ciclo com alguns picos de 576 bits/ciclo.

A avaliação do consumo de energia da aplicação resultou em 95,64 mJ e um tempo de execução total de 140.927 ciclos. Os componentes do consumo de energia desta aplicação são: 95,01 mJ para os processadores, 0,62 mJ para os roteadores e 0,0042 mJ para os fios de interconexão.

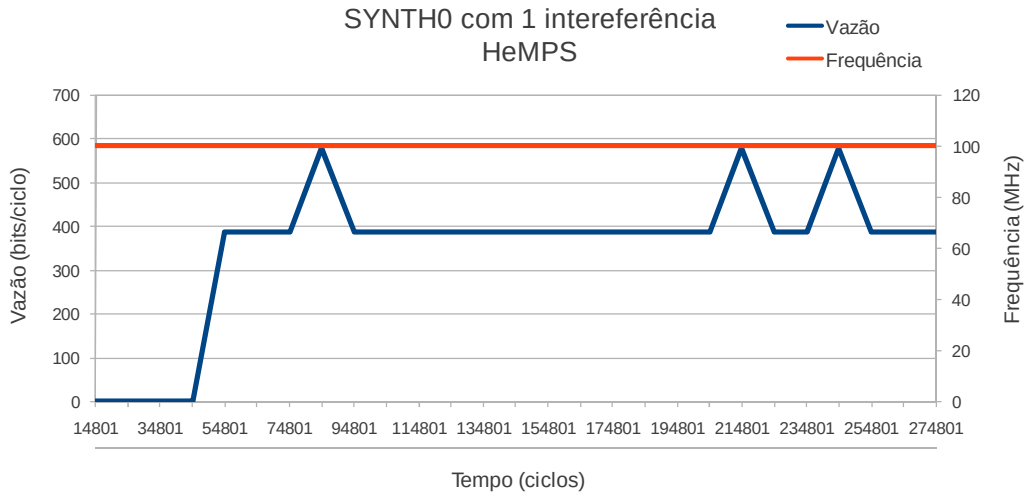


Figura 45 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 2, sobre a plataforma HeMPS.

9.1.1.3 Cenário de Teste 3

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 46 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste permanece a maior parte do tempo em 384 bits/ciclo com alguns picos de 576 bits/ciclo.

A avaliação do consumo de energia da aplicação resultou em 100,38 mJ e um tempo de execução total de 147.177 ciclos. Os componentes do consumo de energia desta aplicação são: 99,67 mJ para os processadores, 0,71 mJ para os roteadores e 0,0047 mJ para os fios de interconexão.

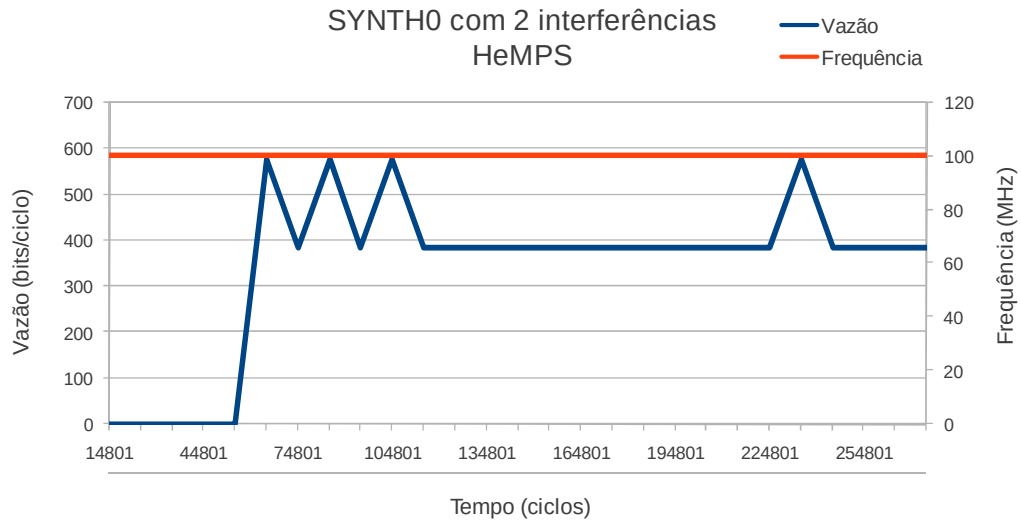


Figura 46 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 3, sobre a plataforma HeMPS.

9.1.1.4 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 47 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste fica oscilando entre 392 e 576 bits/ciclo.

A avaliação do consumo de energia da aplicação resultou em 108,93 mJ e um tempo de execução total de 167.529 ciclos. Os componentes do consumo de energia desta aplicação são: 107,9 mJ para os processadores, 1,02 mJ para os roteadores e 0,0072 mJ para os fios de interconexão.

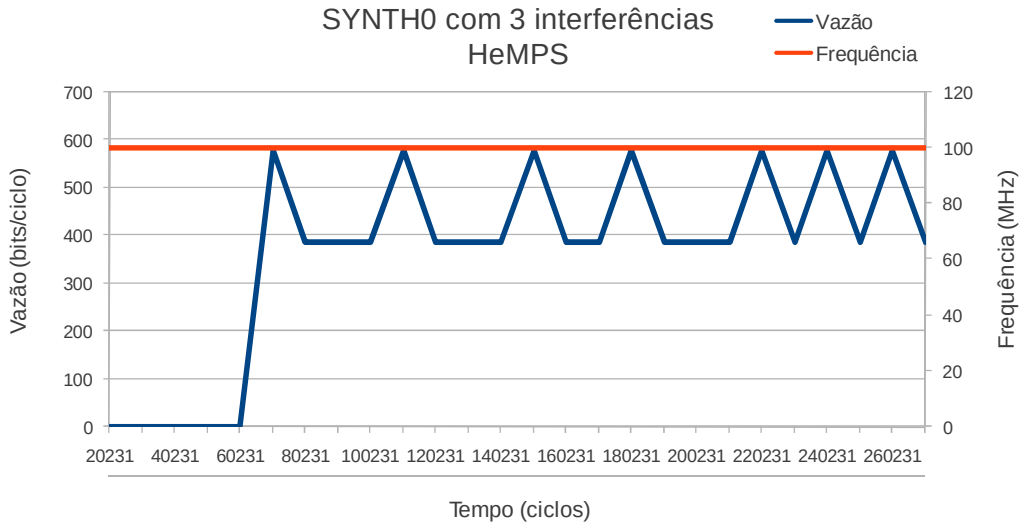


Figura 47 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS.

Verifica-se que nos cenários de teste 2, 3 e 4 um aumento significativo no tempo de execução da aplicação (de 140.727 para 167.529 ciclos). Isto deve-se à interferência gerada pelas tarefas adicionais que atrapalham o fluxo de mensagens da aplicação avaliada.

Nota-se também um maior número de picos de vazão à medida que se aumentam as interferências na comunicação. Estes picos são consequência de pacotes que foram bloqueados, e posteriormente enviados em rajada.

9.1.2 Simulações com a Plataforma HeMPS-D

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS-D, com o mecanismo de DFS proposto em [ROS12b]. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

9.1.2.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS-D, sem a interferência de quaisquer outras aplicações. A Figura 48 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS. A vazão irá oscilar entre 0 e 384 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 52,29 mJ e um tempo de execução total de 274.698 ciclos. Os componentes do consumo de energia desta aplicação são: 51,34 mJ para os processadores, 0,95 mJ para os roteadores e 0,0029 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 95,20%;
- Consumo de energia da aplicação: redução de 44,79%;
- Consumo de energia nos processadores: redução de 45,45%;
- Consumo de energia nos roteadores: aumento de 58,56%;
- Consumo de energia nos fios de interconexão: redução de 26,70%.

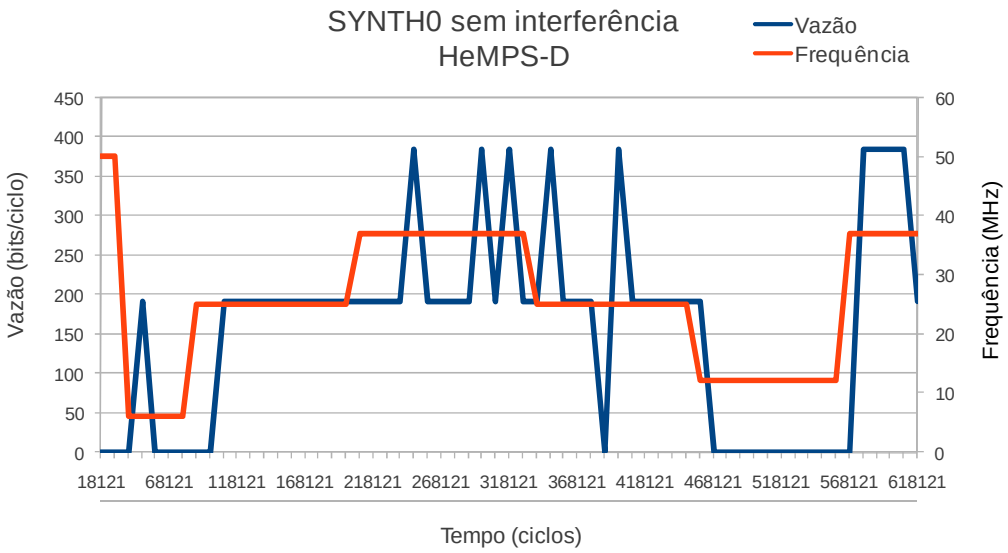


Figura 48 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS-D.

O mecanismo de DFS proposto em [ROS12b] busca fazer com que as filas de comunicação (*pipes*) sempre tenham dados para serem consumidos. Desta forma os processadores têm sua frequência de operação reduzida para o mínimo possível, garantindo apenas um estado operacional das filas de comunicação.

9.1.2.2 Cenário de Teste 2

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-D juntamente

com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 49 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS. A vazão irá oscilar entre 0 e 384 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 75,16 mJ e um tempo de execução total de 315.166 ciclos. Os componentes do consumo de energia desta aplicação são: 73,99 mJ para os processadores, 1,17 mJ para os roteadores e 0,0033 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 123,64%;
- Consumo de energia da aplicação: redução de 21,41%;
- Consumo de energia nos processadores: redução de 22,13%;
- Consumo de energia nos roteadores: aumento de 87,08%;

Consumo de energia nos fios de interconexão: redução de 20,82%.

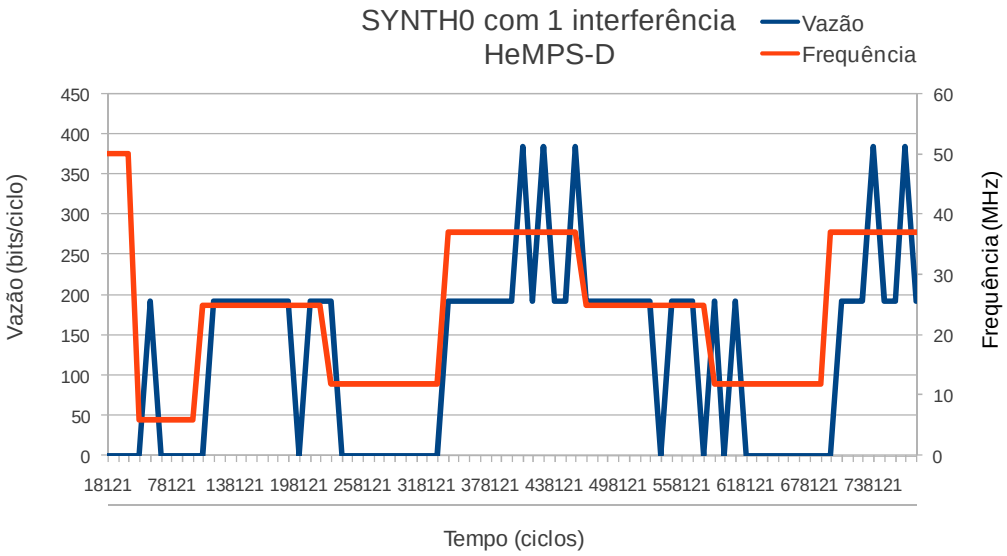


Figura 49 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 2, sobre a plataforma HeMPS-D.

9.1.2.3 Cenário de Teste 3

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-D juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 50 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS. A vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 75,90 mJ e um tempo de execução total de 316.778 ciclos. Os componentes do consumo de energia desta aplicação são: 74,69 mJ para os processadores, 1,21 mJ

para os roteadores e 0,0036 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 115,24%;
- Consumo de energia da aplicação: redução de 24,38%;
- Consumo de energia nos processadores: redução de 25,06%;
- Consumo de energia nos roteadores: aumento de 70,43%;
- Consumo de energia nos fios de interconexão: redução de 24,53%.

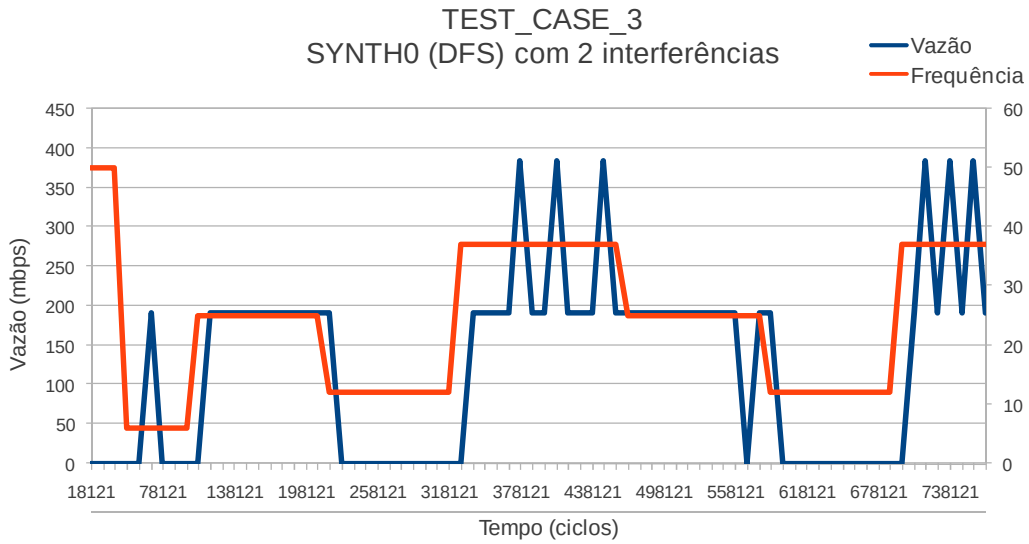


Figura 50 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 3, sobre a plataforma HeMPS-D.

9.1.2.4 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-D juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 51 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. Neste gráfico verifica-se que a frequência varia entre 12 e 50 MHz, devido ao mecanismo de DFS. A vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 68,09 mJ e um tempo de execução total de 279.548 ciclos. Os componentes do consumo de energia desta aplicação são: 67 mJ para os processadores, 1,08 mJ para os roteadores e 0,0043 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 66,87%;
- Consumo de energia da aplicação: redução de 37,49%;

- Consumo de energia nos processadores: redução de 37,90%;
- Consumo de energia nos roteadores: aumento de 5,98%;
- Consumo de energia nos fios de interconexão: redução de 40,64%.

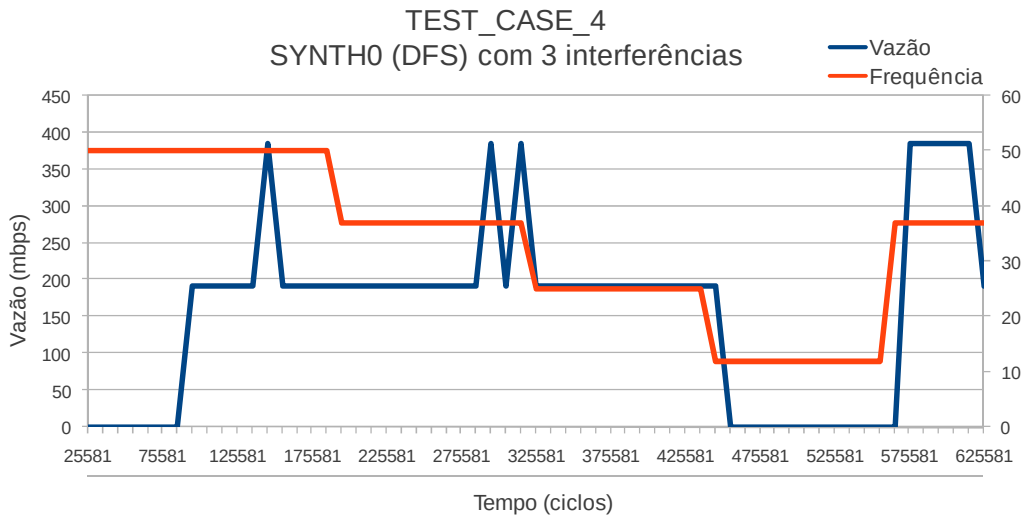


Figura 51 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS-D.

Verifica-se o aumento do consumo de energia no roteador, devido ao mecanismo de DFS do roteador que mantém o roteador ativo por mais tempo, e utilizando as frequências dos transmissores nos repasses de pacotes.

9.1.3 Simulações com a Plataforma HeMPS-DQ

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS-DQ, com o controle da vazão proposto nesta Tese de Doutorado. Os cenários de teste aqui avaliados são descritos na Seção 8.3. Os requisitos de vazão da aplicação são um mínimo de 380 bits/ciclo, e máximo de 600 bits/ciclo (descrito no Capitulo 8, Seção 8.4).

9.1.3.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS-DQ, sem a interferência de quaisquer outras aplicações. A Figura 52 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 188.141 ciclos, como demonstrado na Figura 52. Logo após, em 238.141 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 288.141 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao

mecanismo de DFS controlado pelo módulo de adaptabilidade, e a vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 36,48 mJ e um tempo de execução total de 246.089 ciclos. Os componentes do consumo de energia desta aplicação são: 35,54 mJ para os processadores, 0,94 mJ para os roteadores e 0,0029 mJ para os fios de interconexão.

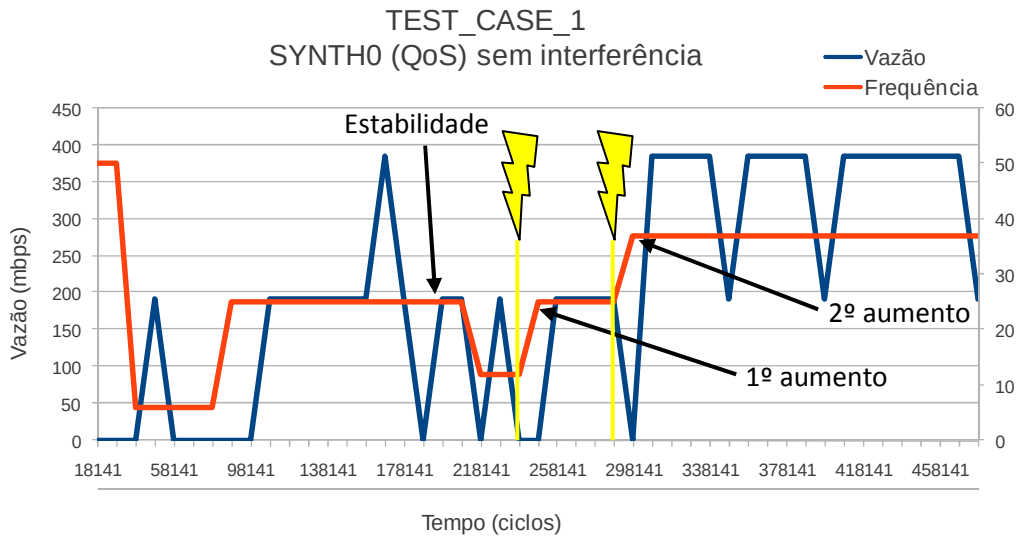


Figura 52 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS-DQ.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 74,87%;
- Consumo de energia da aplicação: redução de 61,48%;
- Consumo de energia nos processadores: redução de 62,24%;
- Consumo de energia nos roteadores: aumento de 57,15%;
- Consumo de energia nos fios de interconexão: redução de 25,60%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: redução de 10,41%;
- Consumo de energia da aplicação: redução de 30,23%;
- Consumo de energia nos processadores: redução de 30,78%;
- Consumo de energia nos roteadores: redução de 0,89%;
- Consumo de energia nos fios de interconexão: aumento de 1,49%.

9.1.3.2 Cenário de Teste 2

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-DQ juntamente com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A **Erro! Fonte de referência não encontrada.**

exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 198.141 ciclos, como demonstrado na Figura 52. Logo após, em 248.141 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 298.141 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 38,77 mJ e um tempo de execução total de 254.705 ciclos. Os componentes do consumo de energia desta aplicação são: 37,73 mJ para os processadores, 1,04 mJ para os roteadores e 0,0033 mJ para os fios de interconexão.

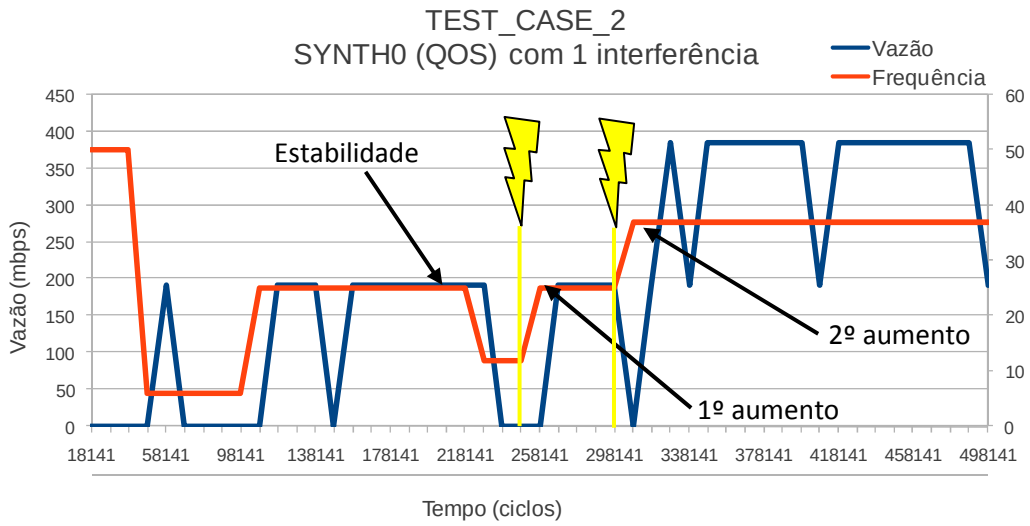


Figura 53 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 2, sobre a plataforma HeMPS-DQ.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 80,74%;
- Consumo de energia da aplicação: redução de 59,46%;
- Consumo de energia nos processadores: redução de 60,29%;
- Consumo de energia nos roteadores: aumento de 66,92%;
- Consumo de energia nos fios de interconexão: redução de 21,13%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: redução de 19,18%;
- Consumo de energia da aplicação: redução de 48,41%;

- Consumo de energia nos processadores: redução de 49,01%;
- Consumo de energia nos roteadores: redução de 10,77%;
- Consumo de energia nos fios de interconexão: redução de 0,39%.

9.1.3.3 Cenário de Teste 3

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-DQ juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 54 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 188.141 ciclos, como demonstrado na Figura 52. Logo após, em 238.141 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 288.141 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 38,56 mJ e um tempo de execução total de 264.738 ciclos. Os componentes do consumo de energia desta aplicação são: 37,46 mJ para os processadores, 1,93 mJ para os roteadores e 0,0036 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 79,88%;
- Consumo de energia da aplicação: redução de 61,59%;
- Consumo de energia nos processadores: redução de 62,41%;
- Consumo de energia nos roteadores: aumento de 54,20%;
- Consumo de energia nos fios de interconexão: redução de 24,13%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: redução de 16,43%;
- Consumo de energia da aplicação: redução de 49,20%;
- Consumo de energia nos processadores: redução de 49,85%;
- Consumo de energia nos roteadores: redução de 9,53%;
- Consumo de energia nos fios de interconexão: aumento de 0,53%.

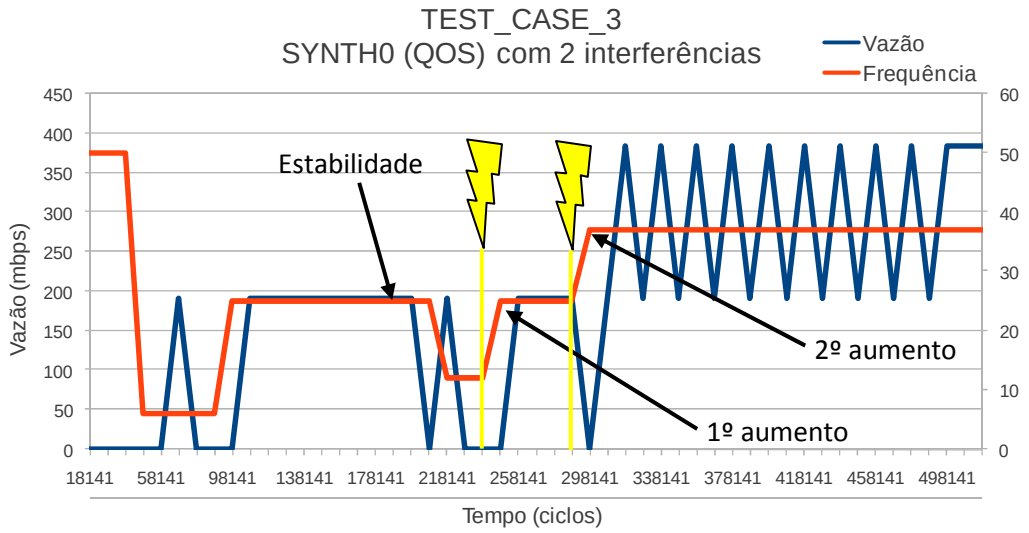


Figura 54 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 3, sobre a plataforma HeMPS-DQ.

9.1.3.4 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-DQ juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 55 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 175.601 ciclos, como demonstrado na Figura 52. Logo após, em 225.601 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 355.601 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 37 e 62 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 392 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 58,18 mJ e um tempo de execução total de 255.904 ciclos. Os componentes do consumo de energia desta aplicação são: 57,13 mJ para os processadores, 1,04 mJ para os roteadores e 0,0044 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 52,75%;
- Consumo de energia da aplicação: redução de 46,59%;
- Consumo de energia nos processadores: redução de 47,06%;
- Consumo de energia nos roteadores: aumento de 2,27%;
- Consumo de energia nos fios de interconexão: redução de 39,17%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: redução de 8,46%;
- Consumo de energia da aplicação: redução de 14,56%;
- Consumo de energia nos processadores: redução de 14,74%;
- Consumo de energia nos roteadores: redução de 3,51%;
- Consumo de energia nos fios de interconexão: aumento de 2,47%.

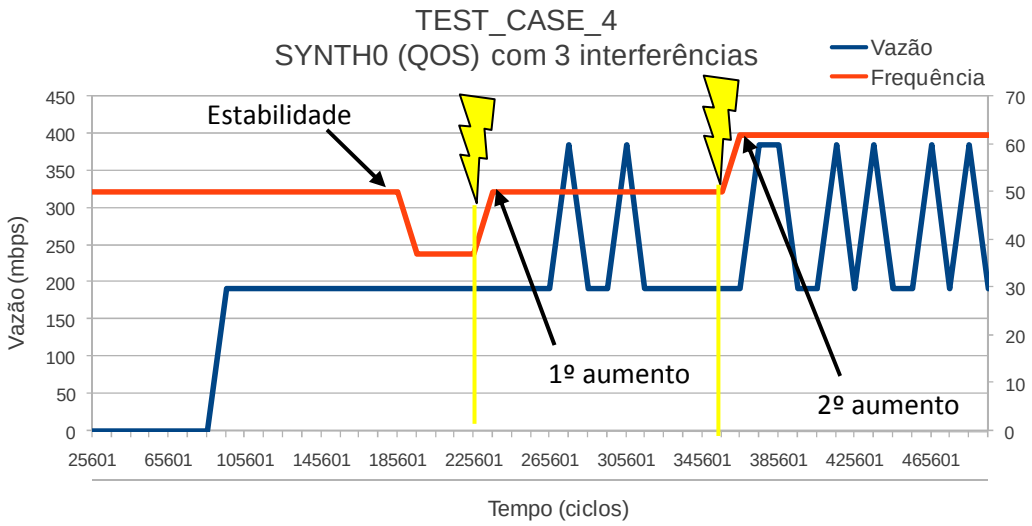


Figura 55 - Resultados de vazão e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS-DQ.

9.2 Resultados da aplicação MPEG2

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação MPEG2 (Figura 41). A Tabela 12 resume os resultados obtidos nas simulações dos cenários de teste descritos no Capítulo 8, Seção 8.3. Os requisitos de vazão para esta aplicação foram um mínimo de 192 bits/ciclo, e máximo de 400 bits/ciclo (como descrito no Capítulo 8, Seção 8.4).

Na Tabela 12 pode-se verificar que a HeMPS sempre atende os requisitos de QoS impostos no menor tempo de execução possível, porém esta plataforma possui o maior consumo de energia na execução da aplicação. As simulações na HeMPS-D exibem reduções de até 75% no consumo de energia e aumentos no tempo de execução de até 50%, e os requisitos de vazão da aplicação não foram respeitados. Já os resultados obtidos na HeMPS-DQ mostram reduções no consumo de energia de até 67% com os requisitos de vazão atendidos, porém o tempo de simulação foi até 56% maior que a HeMPS.

Tabela 12 - Resumo dos resultados das simulações utilizando a aplicação MPEG2.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	1.441.069	8.877,68	-	-	Sim
	HeMPS-D	1.456.794	2.213,27	1,09%	-75,07%	Não
	HeMPS-DQ	**1.491.089**	**2.868,82**	**3,47%**	**-67,69%**	**Sim**
Com 1 interferência	HeMPS	1.191.842	7.219,23	-	-	Sim
	HeMPS-D	1.486.373	2.229,77	24,71%	-69,11%	Não
	HeMPS-DQ	**1.652.961**	**3.528,68**	**38,69%**	**-51,12%**	**Sim**
Com 2 interferências	HeMPS	1.139.155	6.889,38	-	-	Sim
	HeMPS-D	1.715.273	2.590,41	50,57%	-62,40%	Não
	HeMPS-DQ	**1.787.447**	**4.017,88**	**56,91%**	**-41,68%**	**Sim**
Com 3 interferências	HeMPS	1.500.778	8.875,78	-	-	Sim
	HeMPS-D	1.705.467	2.327,32	13,64%	-73,78%	Não
	HeMPS-DQ	**1.807.938**	**4.203,99**	**20,47%**	**-52,64%**	**Sim**

A comparação entre os resultados obtidos entre as simulações na HeMPS-D e na HeMPS-DQ mostram aumentos de até 11% no tempo de execução e de até 80% no consumo de energia.

A Seção 9.2.1 detalha os resultados obtidos na simulação da aplicação MPEG2 sobre a plataforma HeMPS, a Seção 9.2.2 detalha os resultados nas simulações sobre a plataforma HeMPS-D e a Seção 9.2.3 detalha os resultados obtidos nas simulações sobre a plataforma HeMPS-DQ.

9.2.1 Simulações com a Plataforma HeMPS

9.2.1.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS, sem a interferência de quaisquer outras aplicações. A Figura 56 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste oscila entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 8.877,68 mJ e um tempo de execução total de 1.441.069 ciclos. Os componentes do consumo de energia desta aplicação são: 8.865,73 mJ para os processadores, 11,93 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

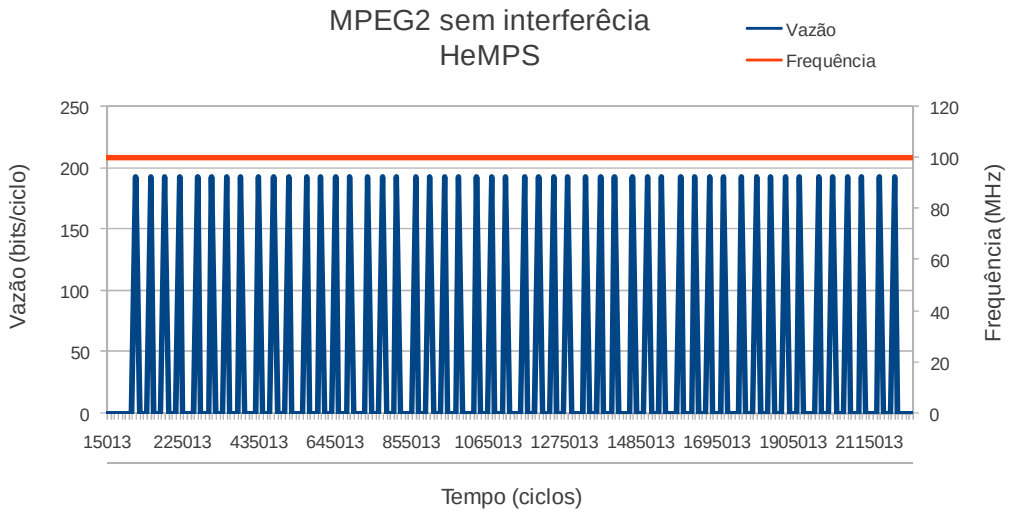


Figura 56 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS.

Os resultados de vazão descritos acima foram utilizados para determinar o perfil de QoS para a aplicação MPEG2. A partir deste perfil de QoS, pode-se determinar os parâmetros de vazão que deverão ser respeitados nas simulações com a plataforma HeMPS-DQ, estes parâmetros estão descritos no Capítulo 8, Seção 8.4.

9.2.1.2 Cenário de Teste 2

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS juntamente com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 57 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste oscila entre 0 e 192 bits/ciclo.

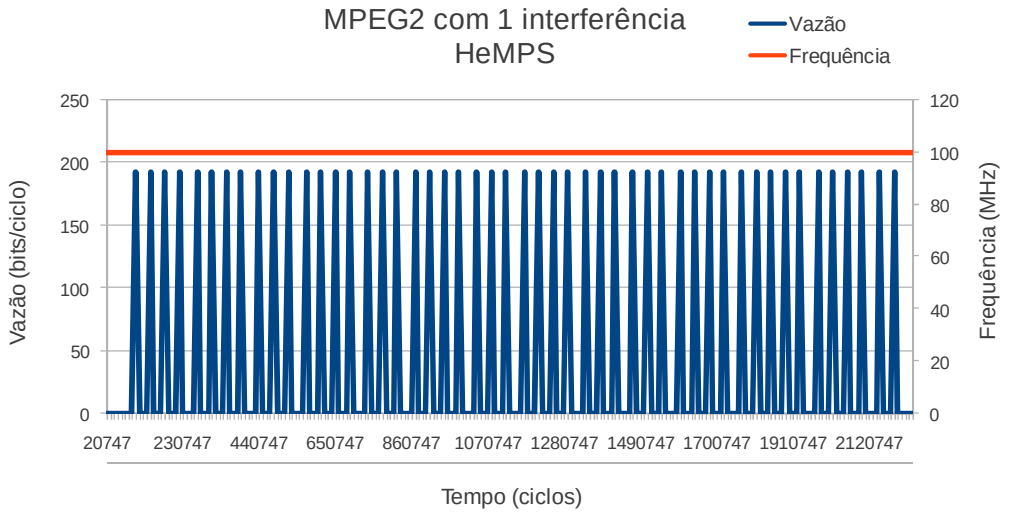


Figura 57 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 2, sobre a plataforma HeMPS.

A avaliação do consumo de energia da aplicação resultou em 7.219,23 mJ e um tempo de execução total de 1.191.842 ciclos. Os componentes do consumo de energia desta aplicação são: 7.212,68 mJ para os processadores, 6,55 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

9.2.1.3 Cenário de Teste 3

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 58 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste oscila entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 6.889,38 mJ e um tempo de execução total de 1.139.155 ciclos. Os componentes do consumo de energia desta aplicação são: 6.883,51 mJ para os processadores, 5,86 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

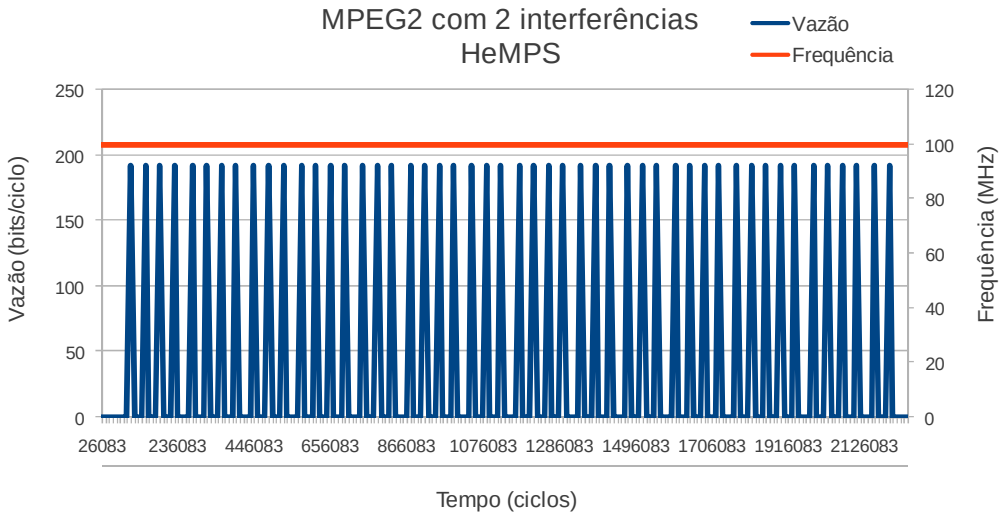


Figura 58 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 3, sobre a plataforma HeMPS.

9.2.1.4 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 59 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a vazão neste cenário de teste oscila entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 8.875,78 mJ e um tempo de execução total de 1.500.778 ciclos. Os componentes do consumo de energia desta aplicação são: 8.864,20 mJ para os processadores, 11,56 mJ para os roteadores e 0,02 mJ para os fios de

interconexão.

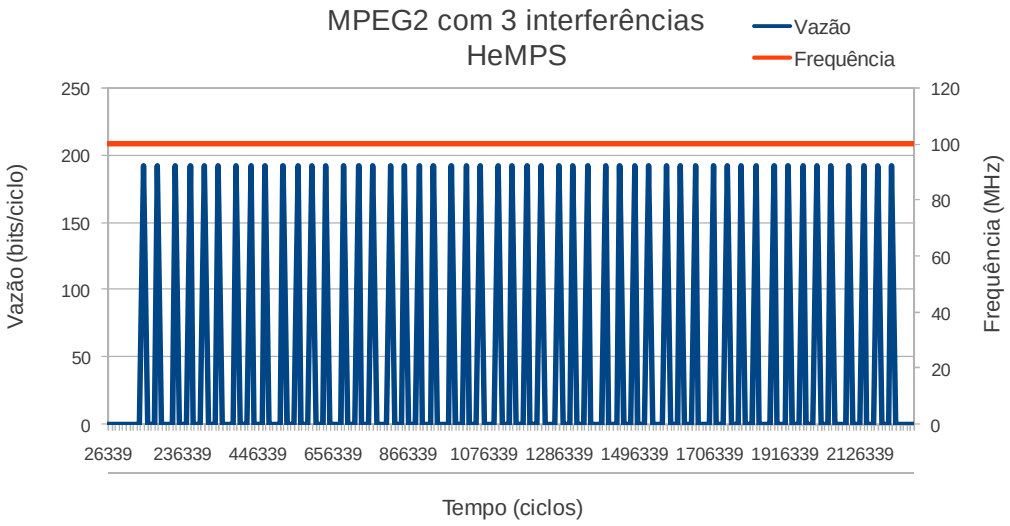


Figura 59 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS.

Verifica-se nos cenários de teste um menor impacto no tempo de execução – 4,1% (de 1.441.069 para 1.500.778). Isto deve-se ao fato que as tarefas das aplicações consomem uma maior parcela do tempo em computação, diferentemente do observado na aplicação sintética SYNTH0.

9.2.2 Simulações com a Plataforma HeMPS-D

Nesta Seção são apresentados os resultados das simulações da aplicação MPEG2 sobre a plataforma HeMPS-D, com o mecanismo de DFS proposto em [ROS12b]. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

9.2.2.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS-D, sem a interferência de quaisquer outras aplicações. A Figura 60 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. Neste gráfico verifica-se que a frequência varia entre 6,66 e 62 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 2.213,27 mJ e um tempo de execução total de 1.456.794 ciclos. Os componentes do consumo de energia desta aplicação são: 2.198,71 mJ para os processadores, 14,55 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

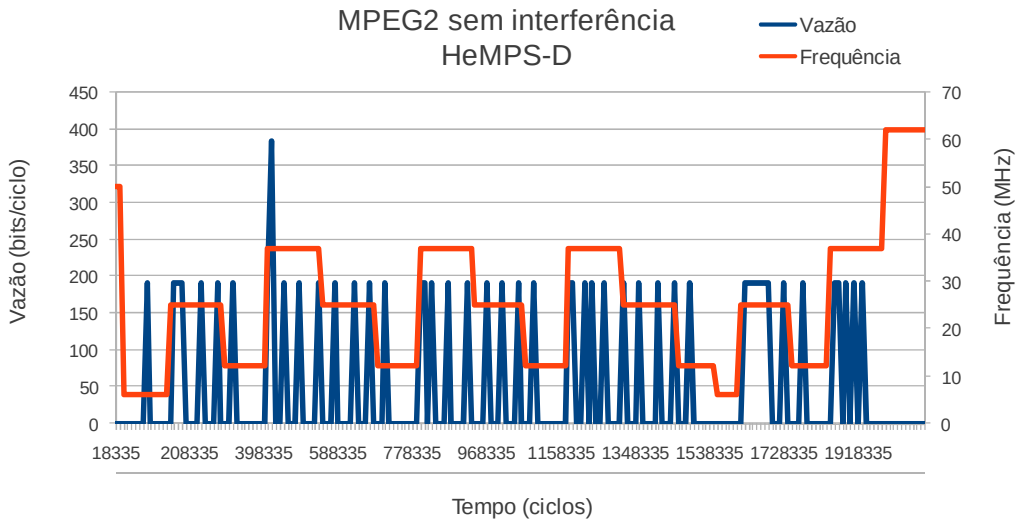


Figura 60 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS-D.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 1,09%;
- Consumo de energia da aplicação: redução de 75,07%;
- Consumo de energia nos processadores: redução de 75,20%;
- Consumo de energia nos roteadores: aumento de 21,94%;
- Consumo de energia nos fios de interconexão: redução de 7,28%.

Observa-se agora, com uma aplicação real, um menor impacto no tempo total de execução quando aplicada a técnica de DFS (+1,09%).

9.2.2.2 Cenário de Teste 2

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-D juntamente com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 61 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. Neste gráfico verifica-se que a frequência varia entre 6,66 e 62 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 2.229,77 mJ e um tempo de execução total de 1.486.373 ciclos. Os componentes do consumo de energia desta aplicação são: 2.216,73 mJ para os processadores, 13,03 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 24,71%;
- Consumo de energia da aplicação: redução de 69,11%;
- Consumo de energia nos processadores: redução de 69,27%;

- Consumo de energia nos roteadores: aumento de 99,04%;
- Consumo de energia nos fios de interconexão: aumento de 1,66%.

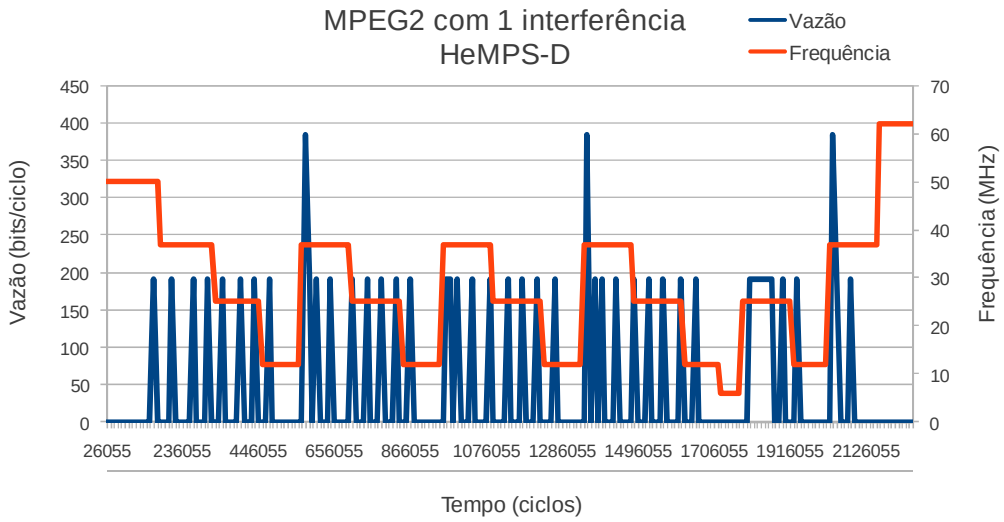


Figura 61 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 2, sobre a plataforma HeMPS-D.

9.2.2.3 Cenário de Teste 3

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-D juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 62 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. Neste gráfico verifica-se que a frequência varia entre 6,66 e 50 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 2.590,41 mJ e um tempo de execução total de 1.715.273 ciclos. Os componentes do consumo de energia desta aplicação são: 2.571,16 mJ para os processadores, 19,24 mJ para os roteadores e 0,02 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 50,57%;
- Consumo de energia da aplicação: redução de 62,40%;
- Consumo de energia nos processadores: redução de 62,65%;
- Consumo de energia nos roteadores: aumento de 228,40%;
- Consumo de energia nos fios de interconexão: aumento de 35,59%.

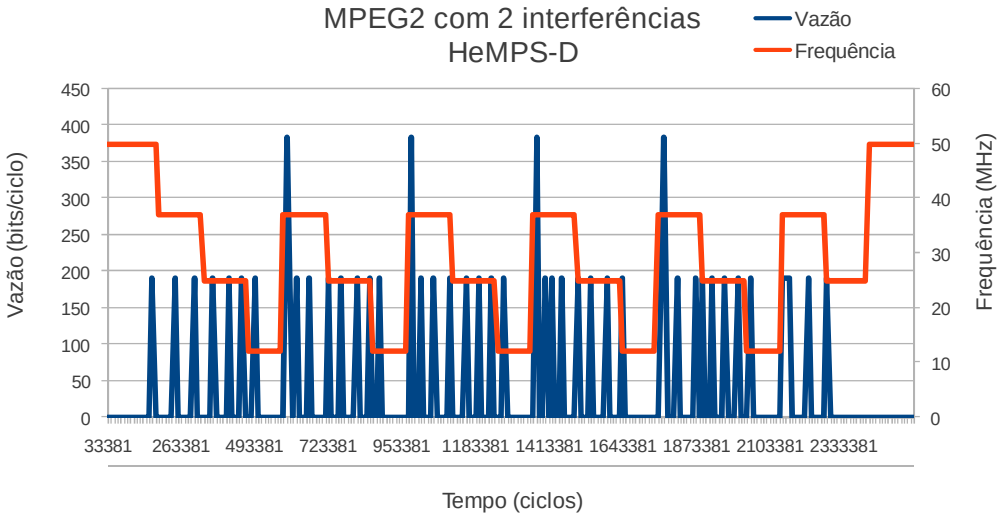


Figura 62 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 3, sobre a plataforma HeMPS-D.

9.2.2.4 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-D juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 63 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. Neste gráfico verifica-se que a frequência varia entre 6,66 e 62 MHz, devido ao mecanismo de DFS, e a vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 2.327,32 mJ e um tempo de execução total de 1.705.467 ciclos. Os componentes do consumo de energia desta aplicação são: 2.308,49 mJ para os processadores, 18,81 mJ para os roteadores e 0,02 mJ para os fios de interconexão.

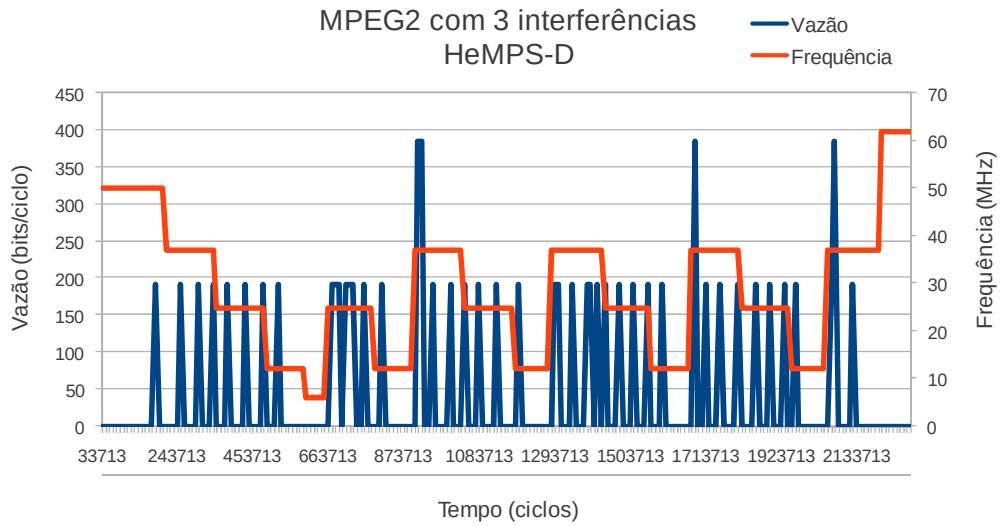


Figura 63 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS-D.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 13,64%;
- Consumo de energia da aplicação: redução de 73,78%;
- Consumo de energia nos processadores: redução de 73,96%;
- Consumo de energia nos roteadores: aumento de 62,76%;
- Consumo de energia nos fios de interconexão: aumento de 0,50%.

9.2.3 Simulações com a Plataforma HeMPS-DQ

Nesta Seção são apresentados os resultados das simulações da aplicação MPEG2 sobre a plataforma HeMPS-DQ, com o controle da vazão proposto nesta Tese de Doutorado. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

9.2.3.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS-DQ, sem a interferência de quaisquer outras aplicações. A Figura 64 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 278.355 ciclos, como demonstrado na Figura 64. Logo após, em 368,355 ciclos, ocorrem cinco violações consecutivas da vazão mínima. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final da execução da aplicação.

Ao final da execução da aplicação, o PE irá enviar uma mensagem para o Plasma Mestre avisando do término da aplicação, o módulo de avaliação de QoS não entende que ocorreu o fim da aplicação e a cada cinco janelas de vazão 0 um novo aumento de frequência é comandado ao módulo de adaptabilidade. A resolução deste problema é elencado como um trabalho futuro.

Neste gráfico verifica-se que a frequência varia entre 6,66 e 25 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. Após a execução da aplicação a frequência sobe de 25 MHz para 75 Mhz. A vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 2.868,82 mJ e um tempo de execução total de 1.491.089 ciclos. Os componentes do consumo de energia desta aplicação são: 2.854,91 mJ para os processadores, 13,90 mJ para os roteadores e 0,01 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 3,47%;
- Consumo de energia da aplicação: redução de 67,69%;
- Consumo de energia nos processadores: redução de 67,80%;

- Consumo de energia nos roteadores: aumento de 16,48%;
- Consumo de energia nos fios de interconexão: redução de 12,37%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: aumento de 2,35%;
- Consumo de energia da aplicação: aumento de 29,62%;
- Consumo de energia nos processadores: aumento de 29,84%;
- Consumo de energia nos roteadores: redução de 4,48%;
- Consumo de energia nos fios de interconexão: redução de 5,49%.

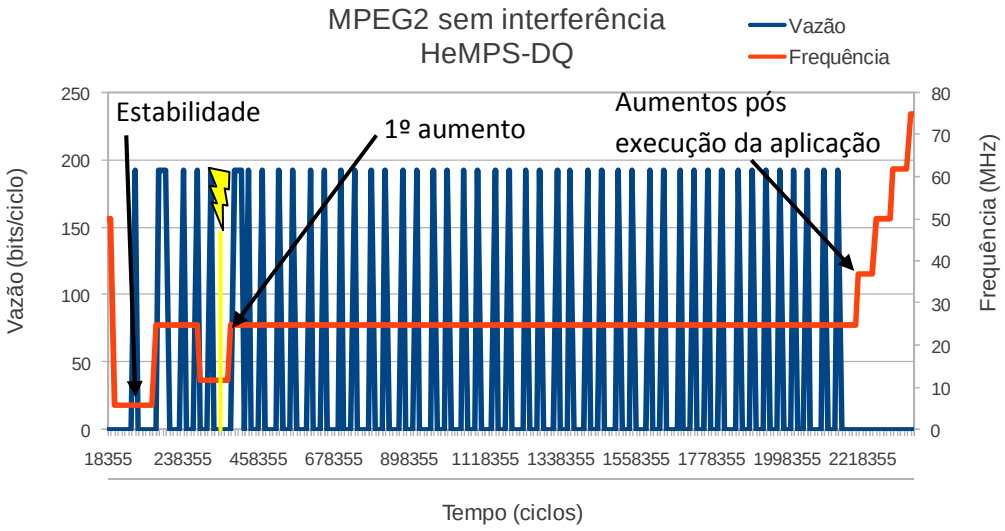


Figura 64 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS-DQ.

9.2.3.2 Cenário de Teste 2

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-DQ juntamente com a aplicação D1. A aplicação D1 gera uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 65 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 2. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 296.075 ciclos, como demonstrado na Figura 65. Logo após, em 526.075 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final da execução da aplicação. Ao final da execução da aplicação, o PE irá enviar uma mensagem para o Plasma Mestre avisando do término da aplicação, o módulo de avaliação de QoS não entende que ocorreu o fim da aplicação e a cada cinco janelas de vazão 0 um novo aumento de frequência é comandado ao módulo de adaptabilidade.

Neste gráfico verifica-se que a frequência varia entre 6,66 e 25 MHz durante a

execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. Após a execução da aplicação a frequência sobe de 25 MHz para 75 Mhz. A vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 3.528,68 mJ e um tempo de execução total de 1.652.961 ciclos. Os componentes do consumo de energia desta aplicação são: 3.510,07 mJ para os processadores, 18,59 mJ para os roteadores e 0,02 mJ para os fios de interconexão.

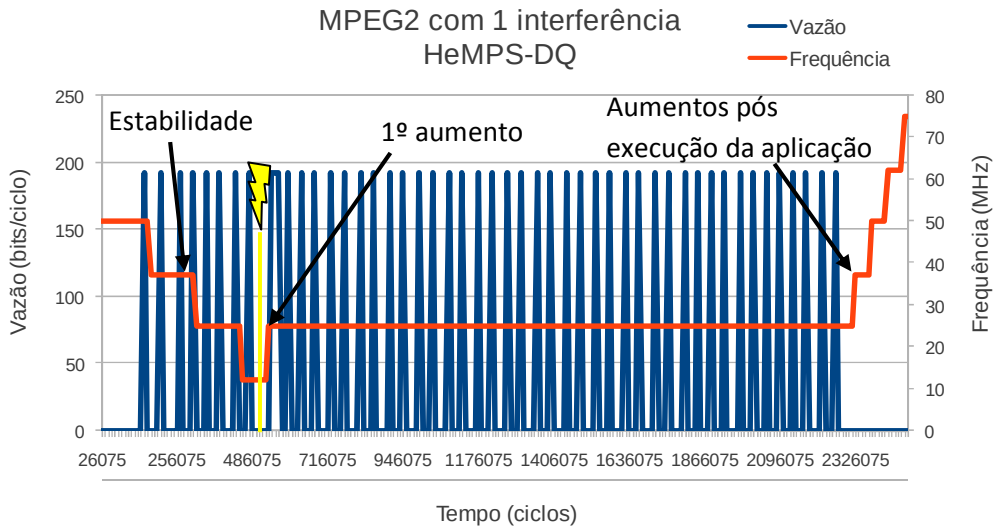


Figura 65 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 2, sobre a plataforma HeMPS-DQ.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 38,69%;
- Consumo de energia da aplicação: redução de 51,12%;
- Consumo de energia nos processadores: redução de 51,33%;
- Consumo de energia nos roteadores: aumento de 184,01%;
- Consumo de energia nos fios de interconexão: aumento de 33,92%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: aumento de 11,21%;
- Consumo de energia da aplicação: aumento de 58,25%;
- Consumo de energia nos processadores: aumento de 58,34%;
- Consumo de energia nos roteadores: aumento de 42,69%;
- Consumo de energia nos fios de interconexão: aumento de 31,74%.

9.2.3.3 Cenário de Teste 3

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-DQ juntamente com as aplicações D1 e D2. Estas aplicações geram uma interferência no fluxo de

comunicação de algumas tarefas da aplicação MPEG2. A Figura 66 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 3. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 313.401 ciclos, como demonstrado na Figura 66. Logo após, em 543.401 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final da execução da aplicação. Ao final da execução da aplicação, o PE irá enviar uma mensagem para o Plasma Mestre avisando do término da aplicação, o módulo de avaliação de QoS não entende que ocorreu o fim da aplicação e a cada cinco janelas de vazão 0 um novo aumento de frequência é comandado ao módulo de adaptabilidade.

Neste gráfico verifica-se que a frequência varia entre 6,66 e 25 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. Após a execução da aplicação a frequência sobe de 25 MHz para 75 Mhz. A vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 4.017,88 mJ e um tempo de execução total de 1.787.447 ciclos. Os componentes do consumo de energia desta aplicação são: 3.995,54 mJ para os processadores, 22,32 mJ para os roteadores e 0,02 mJ para os fios de interconexão.

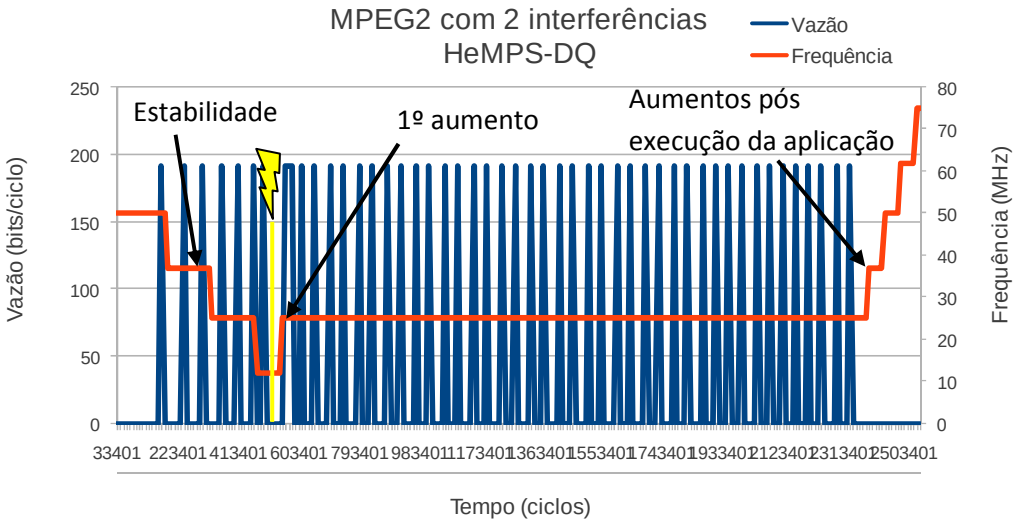


Figura 66 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 3, sobre a plataforma HeMPS-DQ.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 56,91%;
- Consumo de energia da aplicação: redução de 41,68%;
- Consumo de energia nos processadores: redução de 41,95%;
- Consumo de energia nos roteadores: aumento de 281,08%;
- Consumo de energia nos fios de interconexão: aumento de 54,24%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: redução de 4,21%;
- Consumo de energia da aplicação: aumento de 55,11%;
- Consumo de energia nos processadores: aumento de 55,40%;
- Consumo de energia nos roteadores: aumento de 16,04%;
- Consumo de energia nos fios de interconexão: aumento de 13,75%.

9.2.3.4 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-DQ juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 67 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 333.733 ciclos, como demonstrado na Figura 67. Logo após, em 563.733 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. A partir deste ponto, a vazão permanece no interior da faixa permitida, e a frequência permanece constante até o final da execução da aplicação. Ao final da execução da aplicação, o PE irá enviar uma mensagem para o Plasma Mestre avisando do término da aplicação, o módulo de avaliação de QoS não entende que ocorreu o fim da aplicação e a cada cinco janelas de vazão 0 um novo aumento de frequência é comandado ao módulo de adaptabilidade.

Neste gráfico verifica-se que a frequência varia entre 6,66 e 25 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. Após a execução da aplicação a frequência sobe de 25 MHz para 75 Mhz. A vazão irá oscilar entre 0 e 192 bits/ciclo. A avaliação do consumo de energia da aplicação resultou em 4.203,99 mJ e um tempo de execução total de 1.807.938 ciclos. Os componentes do consumo de energia desta aplicação são: 4.182,42 mJ para os processadores, 21,54 mJ para os roteadores e 0,02 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 20,47%;
- Consumo de energia da aplicação: redução de 52,64%;
- Consumo de energia nos processadores: redução de 52,82%;
- Consumo de energia nos roteadores: aumento de 86,35%;
- Consumo de energia nos fios de interconexão: aumento de 9,81%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQ têm-se:

- Tempo de execução: aumento de 6,01%;
- Consumo de energia da aplicação: aumento de 80,64%;
- Consumo de energia nos processadores: aumento de 81,18%;
- Consumo de energia nos roteadores: aumento de 14,49%;
- Consumo de energia nos fios de interconexão: aumento de 9,26%.

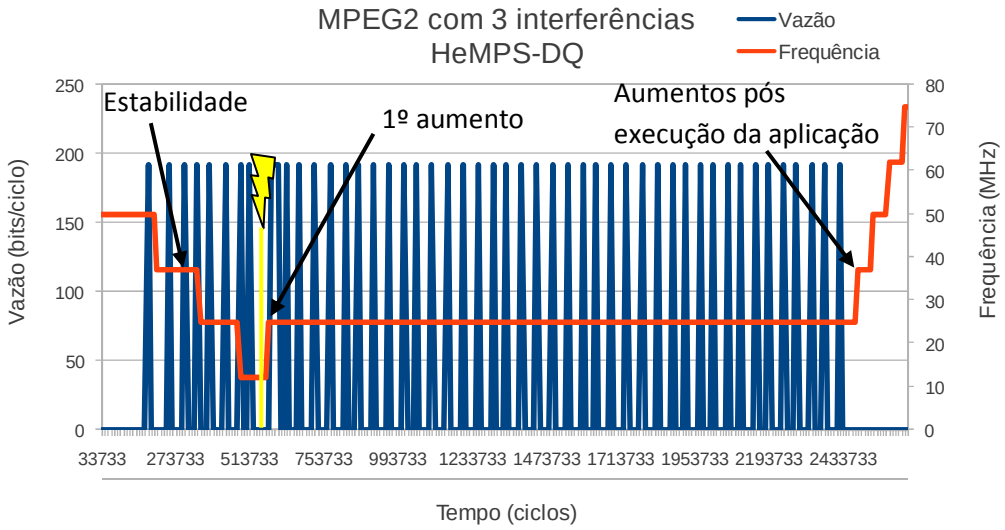


Figura 67 - Resultados de vazão e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS-DQ.

9.3 Resultados da aplicação SYNTH1

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação SYNTH1. O grafo de tarefas desta aplicação está descrito na Figura 40. Esta aplicação foi simulada utilizando-se quatro versões da plataforma HeMPS: (*i*) HeMPS original; (*ii*) HeMPS-D; (*iii*) HeMPS-DQ; (*iv*) a plataforma com controle da vazão da aplicação utilizando o mecanismo de DFS com descarte de mensagens de troca de frequência – HeMPS-DQd.

A Tabela 13 exibe um resumo dos resultados obtidos com aplicação SYNTH1. Notar que os resultados obtidos neste experimento não foram positivos em termos de redução de energia, porém o atendimento aos requisitos de QoS foram respeitados. A discussão destes resultados é feita na sequência desta seção.

Os requisitos de vazão para esta aplicação foram um mínimo de 190 bits/ciclo, e máximo de 400 bits/ciclo (como descrito no Capítulo 8, Seção 8.4). Nesta tabela pode-se verificar que a HeMPS sempre atende os requisitos de QoS impostos no menor tempo de execução possível, e com a exceção do cenário de teste com 3 interferências, com o menor consumo de energia possível com atendimento dos requisitos de vazão. A plataforma HeMPS-D aumentou o tempo de execução em até 299%, e o consumo de energia em até 96%, sem atendimento dos requisitos de vazão. Os resultados

apresentados na plataforma HeMPS-DQ mostram um aumento de até 144% no tempo de execução e de até 35% maiores em consumo de energia, com exceção do cenário de teste com 3 interferências onde o tempo de execução aumentou em 122% e com uma redução de 0,19% no consumo de energia. Esta plataforma sempre atende os requisitos de vazão impostos à aplicação. Já a execução na plataforma HeMPS-DQd, quando comparada com a HeMPS, apresenta aumentos de até 205% no tempo de execução e de até 222% no consumo de energia. A exceção é o cenário de teste com 3 interferências, onde o tempo de execução foi 136% maior com redução de 10% no consumo de energia. Em todos os cenários de teste avaliados, esta plataforma conseguiu manter os requisitos de vazão impostos à aplicação.

Tabela 13 - Resumo dos resultados das simulações utilizando a aplicação SYNTH1.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	767.782	355,46	-	-	Sim
	HeMPS-D	3.067.935	634,65	299,58%	78,54%	Não
	HeMPS-DQ	**1.733.495**	**447,92**	**125,78%**	**26,01%**	**Sim**
	HeMPS-DQd	**2.342.232**	**849,01**	**205,06%**	**138,85%**	**Sim**
Com 1 interferência	HeMPS	791.553	358,82	-	-	Sim
	HeMPS-D	3.096.917	676,67	291,25%	88,58%	Não
	HeMPS-DQ	**1.938.068**	**485,95**	**144,84%**	**35,43%**	**Sim**
	HeMPS-DQd	**1.941.466**	**469,51**	**145,27%**	**30,85%**	**Sim**
Com 2 interferências	HeMPS	789.677	349,80	-	-	Sim
	HeMPS-D	3.064.978	686,44	288,13%	96,23%	Não
	HeMPS-DQ	**1.643.869**	**423,15**	**108,17%**	**20,97%**	**Sim**
	HeMPS-DQd	**2.146.465**	**1.127,80**	**171,82%**	**222,41%**	**Sim**
Com 3 interferências	HeMPS	788.567	369,24	-	-	Sim
	HeMPS-D	2.836.177	619,69	259,66%	67,83%	Não
	HeMPS-DQ	**1.755.343**	**368,53**	**122,60%**	**-0,19%**	**Sim**
	HeMPS-DQd	**1.865.868**	**331,80**	**136,62%**	**-10,14%**	**Sim**

Adicionalmente, comparando-se os resultados obtidos entre as simulações na HeMPS-D e na HeMPS-DQ mostram reduções de até 46% no tempo de execução e de até 40% no consumo de energia. Comparando-se as plataformas HeMPS-DQ e HeMPS-DQd, os resultados apontam reduções de até 9% no tempo de execução e aumentos de

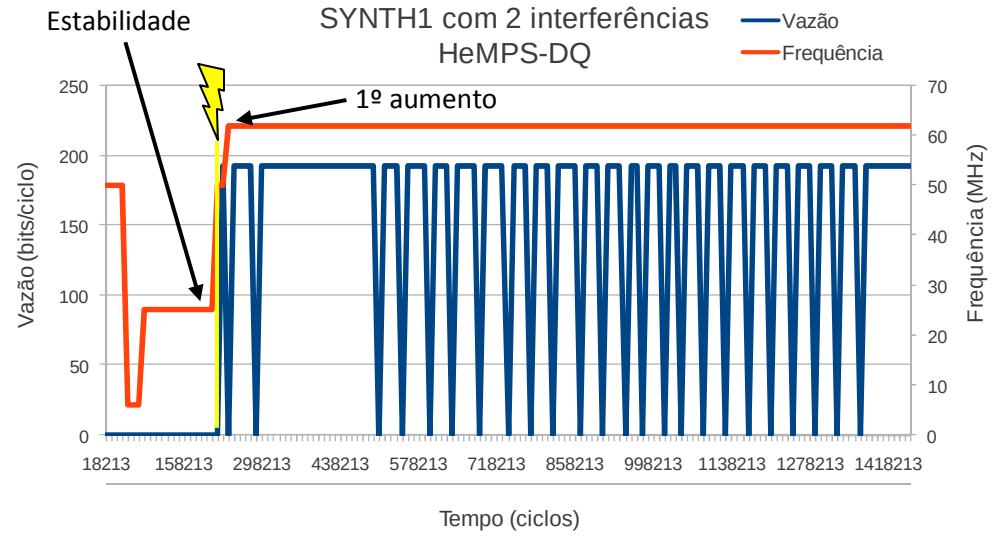
até 63% no consumo de energia.

Os resultados acima descritos mostram um aumento no tempo de execução e no consumo de energia das plataformas HeMPS-D, HeMPS-DQ e HeMPS-DQd em relação à HeMPS. Isto se dá pelo fato de que os monitores em software inseridos nas plataformas, os quais são ativados em cada comunicação de tarefas, inserem uma carga computacional na execução das tarefas. Em todas as plataformas estes monitores foram inseridos a fim de avaliar os resultados de vazão e frequência, mesmo nas plataformas HeMPS e HeMPS-D que não possuem o mecanismo de controle de QoS. Os aumentos de tempo de execução e consumo de energia ficam evidenciados nas plataformas onde os processadores admitem variação de frequência, onde o aumento na carga computacional produz um impacto no tempo de execução e consumo de energia.

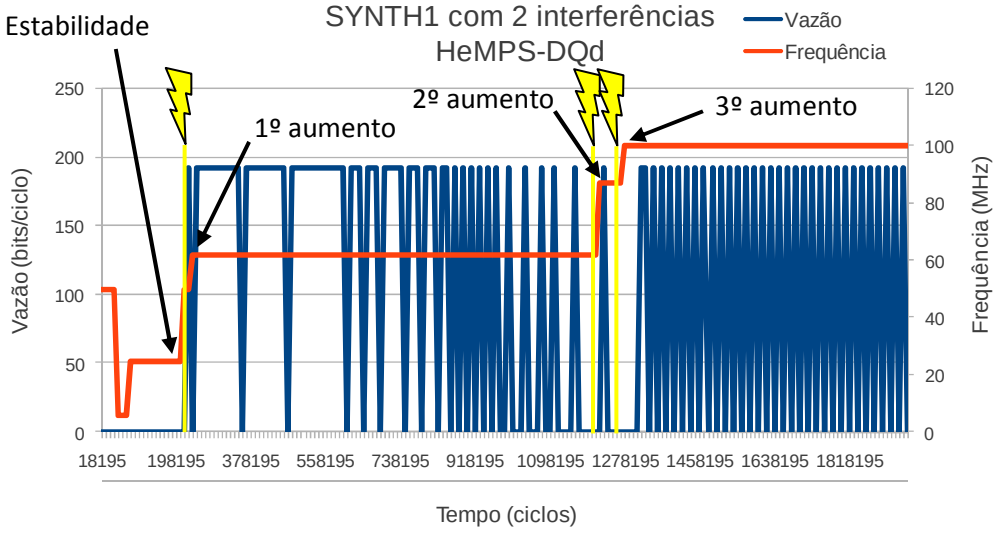
Desta forma, os resultados para esta aplicação mostram um maior consumo de energia devido ao aumento no tempo de execução. No cenário de teste com três interferências, os resultados mostram uma redução no consumo de energia. Este comportamento ocorre devido à interferência nas comunicações entre as tarefas da aplicação SYNTH1, o que desacelera o fluxo de comunicações entre as tarefas e consequentemente o acionamento frequente dos monitores de vazão. O menor acionamento dos monitores de vazão reduz a carga computacional causada pela ativação destes monitores, e os PEs irão consumir menos energia na execução da aplicação.

A Figura 68 exibe os resultados de vazão para a aplicação SYNTH1 quando executada nas plataformas sem descarte de mensagens de adaptabilidade (HeMPS-DQ) e com este descarte (HeMPS-DQd). Na simulação sobre a plataforma HeMPS-DQ (Figura 68(a)), o módulo de adaptabilidade dispara uma mensagem de aumento na frequência das tarefas da aplicação e esta mensagem é propagada pelas tarefas. As tarefas B e C (descritas no grafo da aplicação apresentado no Capítulo 8, Seção 8.2), recebem esta mensagem de duas tarefas distintas: D e E. O recebimento desta mensagem duplicada irá disparar dois aumentos na frequência dos PEs das tarefas B e C, e o reflexo direto desta ação aparece no gráfico de vazão da tarefa final F, que verifica o cumprimento dos requisitos de vazão com somente um disparo de mensagem de adaptabilidade.

Já na simulação sobre a plataforma HeMPS-DQd (Figura 68(b)), verifica-se que a aplicação não conseguiu respeitar seu requisito de vazão com somente um aumento de frequência. Isto ocorre, pois o descarte de mensagens de adaptabilidade irá eliminar a mensagem duplicada e o aumento duplo de frequência não ocorre. Desta forma a aplicação não conseguiu manter seus requisitos de vazão e a tarefa final solicitou dois novos aumentos de frequência de toda a aplicação. A partir do último aumento de frequência os requisitos de vazão são atendidos.



(a)



(b)

Figura 68 - Resultados de vazão para as plataformas (a) HeMPS-DQ e (b) HeMPS-DQd, exemplificando o efeito do descarte de mensagens de adaptabilidade.

Apesar dos resultados adversos no consumo de energia e tempo de execução, as simulações com a aplicação SYNTH1 validaram o mecanismo de descarte de mensagens de adaptabilidade duplicadas. Esta melhoria no mecanismo proposto de controle de QoS, visa otimizar os aumentos na frequência dos PEs e assim reduzir o consumo de energia da aplicação.

9.4 Conclusão

Os resultados apresentados neste Capítulo mostram que o mecanismo proposto de controle de vazão consegue cumprir com o seu objetivo, evitando que o MPSoC desrespeite o requisito de QoS imposto às aplicações.

Os resultados para a aplicação SYNTH0 mostram que o mecanismo proposto permitiu controlar o requisito de vazão e reduzir o consumo de energia do MPSoC, apesar

do maior tempo de execução da aplicação controlada.

Os resultados para a aplicação MPEG2 também mostram que o mecanismo proposto permitiu o controle da vazão com redução no consumo de energia da aplicação (até 54% menor) e pouco aumento no tempo total de execução em comparação com a HeMPS-D (até 6% maior).

Já os resultados com a aplicação SYNTH1 mostram que o mecanismo proposto insere uma carga computacional adicional devido aos monitores, que são acionados em cada comunicação entre as tarefas da aplicação. Para a aplicação SYNTH1, que possui tarefas com muita comunicação e pouca computação, esta carga adicional aumenta o tempo de execução e consumo de energia da aplicação. Os resultados mostram que esta deficiência é mitigada quando o fluxo de comunicações entre as tarefas é desacelerado, e assim o mecanismo proposto produz os melhores resultados no consumo de energia. Nos resultados apresentados nas simulações da aplicação SYNTH1, pode-se verificar a validade do descarte de mensagens de adaptabilidade. O descarte destas mensagens duplicadas permitiu uma redução de até 9% no tempo de execução da aplicação SYNTH1 em comparação com os resultados obtidos na plataforma sem a utilização do descarte destas mensagens duplicadas.

O aumento da carga computacional inserida pelos monitores ficou evidenciado nas aplicações com tarefas com muita comunicação e pouca computação. Esta deficiência será tratada em novas versões da plataforma HeMPS-DQd, onde parte dos monitores será implamentada em hardware.

10 RESULTADOS OBTIDOS PARA AVALIAÇÃO DE LATÊNCIA

Neste Capítulo são apresentados os resultados obtidos durante a execução dos cenários de teste 1 e 4 descritos no Capítulo 8, visando o controle da latência das aplicações. Os resultados estão assim organizados:

(i) a Seção 10.1 exibe os resultados obtidos nas simulações com a aplicação SYNTH0;

(ii) a Seção 10.3 exibe os resultados obtidos nas simulações com a aplicação MPEG2;

(iii) a Seção 10.4 exibe os resultados obtidos nas simulações com a aplicação SYNTH1;

Os resultados de latência das aplicações são medidos em todas as tarefas de cada aplicação, porém somente os gráficos das tarefas finais das aplicações são exibidos. As avaliações de consumo de energia das aplicações são o somatório do consumo de energia de todos os PEs que executam as tarefas da aplicação até o final da execução desta.

Ao início de cada Seção é apresentada uma tabela com o resumo dos resultados obtidos, e ao final do Capítulo é apresentada uma breve conclusão sobre os resultados obtidos.

10.1 Resultados da aplicação SYNTH 0

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação SYNTH0. O grafo de tarefas desta aplicação está descrito na Figura 39. A Tabela 14 exibe um resumo dos resultados obtidos nas simulações acima descritas.

Tabela 14 - Resumo dos resultados das simulações utilizando a aplicação SYNTH0.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	559.190	510,50	-	-	Sim
	HeMPS-D	1.10.2 524.753	988,83	172,67%	93,70%	Não
	HeMPS-DQd	**1.072.853**	**777,58**	**91,86%**	**52,32%**	**Sim**
Com 3 interferências	HeMPS	580.427	548,76	-	-	Sim
	HeMPS-D	1.152.475	808,69	98,56%	47,37%	Não
	HeMPS-DQd	**750.040**	**475,03**	**29,22%**	**-13,44%**	**Sim**

O requisito de latência imposto para esta aplicação foi de no mínimo 350 ciclos em cada comunicação (como descrito no Capítulo 8, Seção 8.5). Nesta tabela pode-se

verificar que a HeMPS sempre atende os requisitos de latência impostos no menor tempo de execução possível. As simulações na HeMPS-D exibem aumentos de até 93% no consumo de energia e de 172% no tempo de execução, e os requisitos de latência da aplicação não foram respeitados. Já os resultados obtidos na HeMPS-DQd mostram aumentos de até 91% no tempo de execução, porém o consumo de energia aumentou em 52% no primeiro cenário de este e diminuiu em 13% no cenário de teste com várias interferências. Nesta plataforma os requisitos de latência foram atendidos.

Adicionalmente, os resultados obtidos entre as simulações na HeMPS-D e na HeMPS-DQq mostram reduções de até 34% no tempo de execução e de até 41% no consumo de energia, na comparação entre estas duas plataformas.

A Seção 10.2.1 detalha os resultados obtidos na simulação da aplicação SYNTH0 sobre a plataforma HeMPS, a Seção 10.2.2 detalha os resultados nas simulações sobre a plataforma HeMPS-D e a Seção 10.2.3 detalha os resultados obtidos nas simulações sobre a plataforma HeMPS-DQd.

10.2.1 Simulações com a Plataforma HeMPS

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.2.1.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS, sem a interferência de quaisquer outras aplicações. A **Erro! Fonte de referência não encontrada.** exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste permanece a maior parte do tempo em 219 ciclos de relógio.

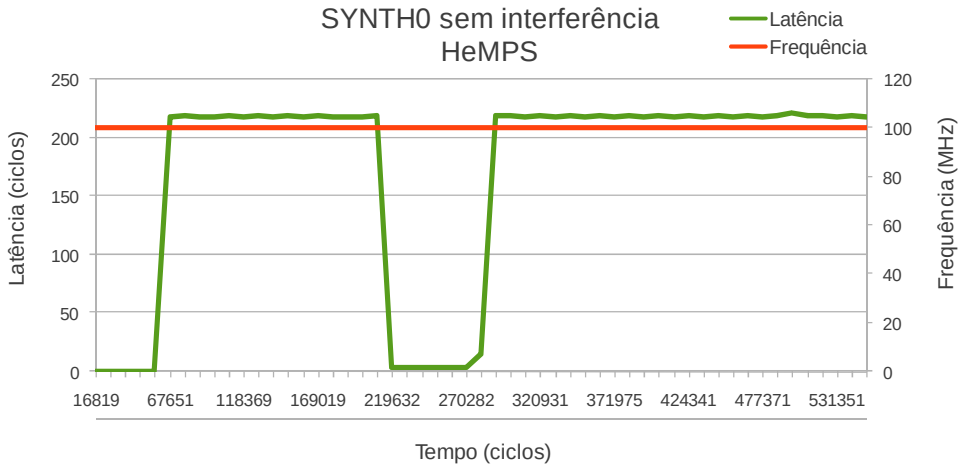


Figura 69 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS.

A avaliação do consumo de energia da aplicação resultou em 510,504 mJ e um tempo de execução total de 559.190 ciclos. Os componentes do consumo de energia desta aplicação são: 507,872 mJ para os processadores, 2,623 mJ para os roteadores e 0,009 mJ para os fios de interconexão.

Os resultados de latência descritos acima foram utilizados para determinar o perfil de QoS para a aplicação SYNTH0. A partir deste perfil de QoS, pode-se determinar os parâmetros de QoS que deverão ser respeitados nas simulações com a plataforma HeMPS-DQd.

10.2.1.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 70 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste permanece a maior parte do tempo em 219 ciclos de relógio.

A avaliação do consumo de energia da aplicação resultou em 548,764 mJ e um tempo de execução total de 580.427 ciclos. Os componentes do consumo de energia desta aplicação são: 544,967 mJ para os processadores, 3,783 mJ para os roteadores e 0,014 mJ para os fios de interconexão.

Verifica-se que nos cenários de teste 1 e 4, um aumento no tempo de execução da aplicação de 3,8% (de 559.190 para 580.427 ciclos). Isto deve-se à interferência gerada pelas tarefas adicionais que atrapalham o fluxo de mensagens da aplicação avaliada.

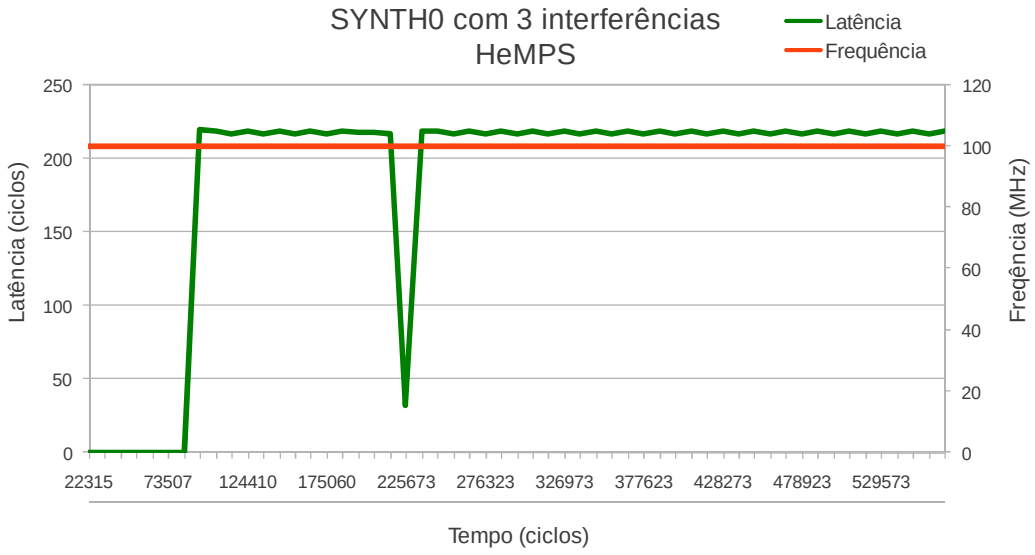


Figura 70 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS.

10.2.2 Simulações com a Plataforma HeMPS-D

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS-D, com o mecanismo de DFS proposto em [ROS12b]. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.2.2.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS-D, sem a interferência de quaisquer outras aplicações. A Figura 71 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. Neste gráfico verifica-se que a frequência varia entre 6,67 e 50 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 254 e 868 ciclos, com picos de 6.825 ciclos. A avaliação do consumo de energia da aplicação resultou em 988,831 mJ e um tempo de execução total de 1.524.753 ciclos. Os componentes do consumo de energia desta aplicação são: 982,237 mJ para os processadores, 6,585 mJ para os roteadores e 0,009 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 172,67%;
- Consumo de energia da aplicação: aumento de 93,70%;
- Consumo de energia nos processadores: aumento de 93,40%;
- Consumo de energia nos roteadores: aumento de 151,04%;
- Consumo de energia nos fios de interconexão: aumento de 5,14%.

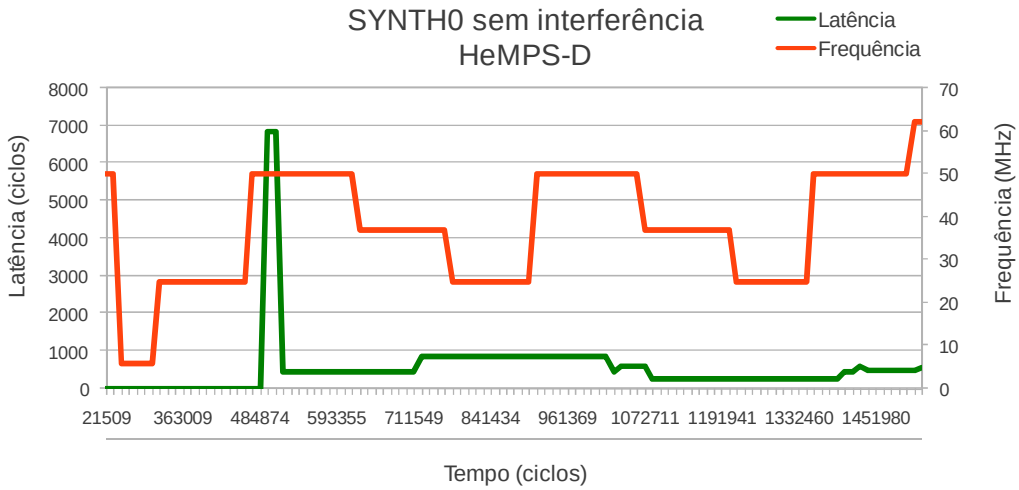


Figura 71 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS-D.

Neste cenário de teste verifica-se um aumento significativo no consumo de energia dos processadores. Isto se dá pelo acionamento dos monitores de vazão e latência, em software, os quais adicionam uma carga adicional de processamento aos processadores. Em tarefas com perfis de alta comunicação e baixa comunicação (como as aplicações SYNTH0 e SYNTH1) esta carga adicional é evidenciada, pois os monitores são acionados

a cada comunicação com as demais tarefas. Desta forma, o sistema de monitoramento proposto leva a um elevado consumo de energia em cenários de teste com aplicações altamente comunicativas e com processadores com DFS.

10.2.2.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-D juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 72 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. Neste gráfico verifica-se que a frequência varia entre 25 e 62 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 271 e 610 ciclos, com um pico de 1.550 ciclos. A avaliação do consumo de energia da aplicação resultou em 808,686 mJ e um tempo de execução total de 1.152.475 ciclos. Os componentes do consumo de energia desta aplicação são: 803,122 para os processadores, 5,554 mJ para os roteadores e 0,011 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 98,56%;
- Consumo de energia da aplicação: aumento de 47,37%;
- Consumo de energia nos processadores: aumento de 47,37%;
- Consumo de energia nos roteadores: aumento de 46,83%;
- Consumo de energia nos fios de interconexão: redução de 23,80%.

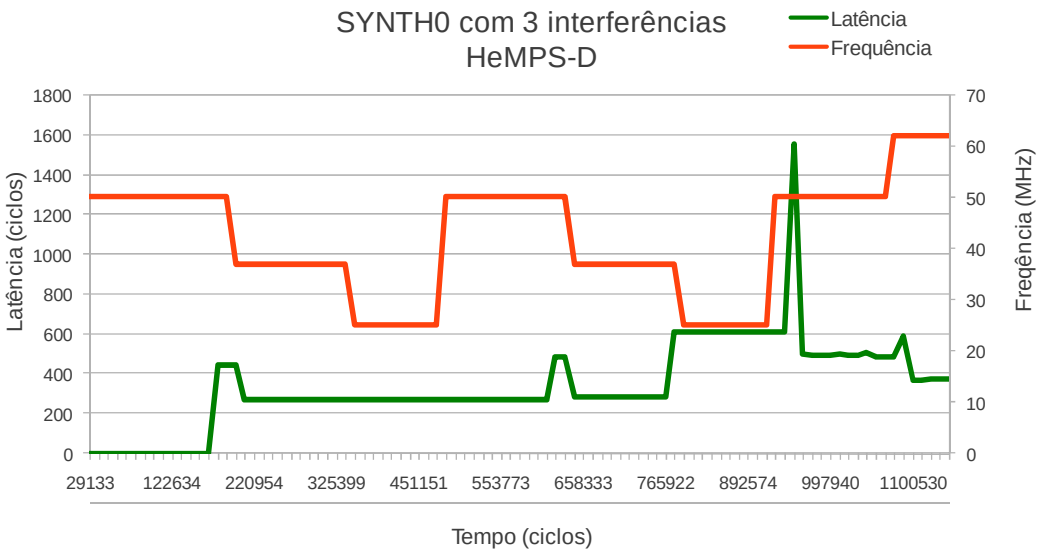


Figura 72 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS-D.

O aumento no consumo de energia é explicado pela carga adicional gerada pelos monitores em software nos processadores. Esta deficiência é mais impactante em aplicações com perfil de alta comunicação.

10.2.3 Simulações com a Plataforma HeMPS-DQ

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH0 sobre a plataforma HeMPS-DQd, com o controle da latência proposto nesta Tese de Doutorado. Os cenários de teste aqui avaliados são descritos na Seção 8.3. O requisito de latência imposto às tarefas desta aplicação é de 350 ciclos em cada comunicação (descrito no Capitulo 8, Seção 8.5).

10.2.3.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH0 é executada sozinha na HeMPS-DQd, sem a interferência de quaisquer outras aplicações. A Figura 73 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 452.730 ciclos, como demonstrado na Figura 73. Logo após, em 546.002 ciclos, ocorrem cinco violações consecutivas da latência mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 879.204 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste ponto, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 75 MHz, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 310 e 349 ciclos quando controlada, e com picos de até 6.016 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 777,577 mJ e um tempo de execução total de 1.072.853 ciclos. Os componentes do consumo de energia desta aplicação são: 771,871 mJ para os processadores, 5,696 mJ para os roteadores e 0,010 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 91,86%;
- Consumo de energia da aplicação: aumento de 52,32%;
- Consumo de energia nos processadores: aumento de 51,98%;
- Consumo de energia nos roteadores: aumento de 117,15%;
- Consumo de energia nos fios de interconexão: aumento de 6,69%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 29,64%;

- Consumo de energia da aplicação: redução de 21,36%;
- Consumo de energia nos processadores: redução de 21,42%;
- Consumo de energia nos roteadores: redução de 13,50%;
- Consumo de energia nos fios de interconexão: aumento de 1,48%.

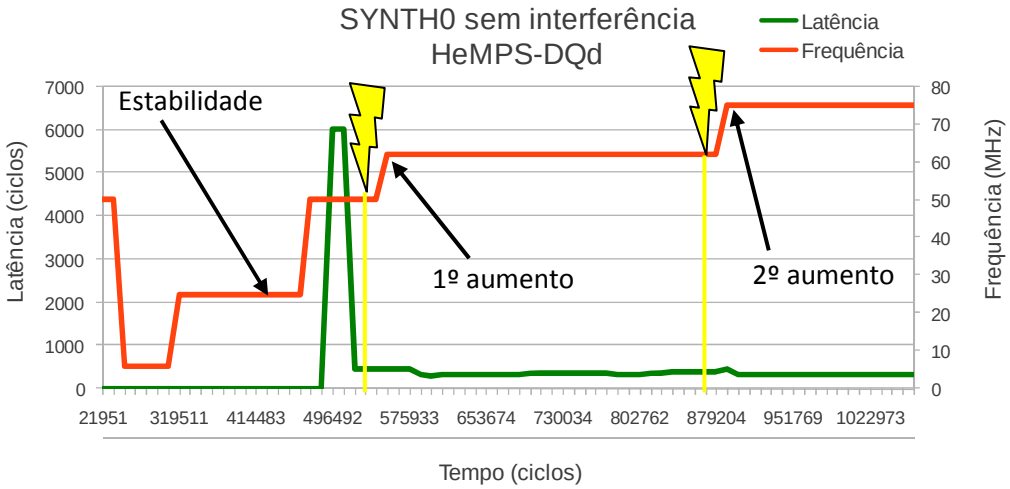


Figura 73 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 1, sobre a plataforma HeMPS-DQd.

10.2.3.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH0 é executada na HeMPS-DQd juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH0. A Figura 74 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 186.686 ciclos, como demonstrado na Figura 74. Logo após, em 245.034 ciclos, ocorrem cinco violações consecutivas da latência mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 298.883 e 352.213 ciclos, o que dispara novamente dois aumentos na frequência de toda a aplicação. A partir do último aumento de frequência, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 37 e 75 MHz, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 310 e 314 ciclos quando controlada, e com picos de até 595 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 475,031 mJ e um tempo de execução total de 750.040 ciclos. Os componentes do consumo de energia desta aplicação são: 470,369 mJ para os processadores, 4,651 mJ para os roteadores e 0,011 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 29,22%;
- Consumo de energia da aplicação: redução de 13,44%;
- Consumo de energia nos processadores: redução de 13,69%;
- Consumo de energia nos roteadores: aumento de 22,96%;
- Consumo de energia nos fios de interconexão: redução de 20,21%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 34,92%;
- Consumo de energia da aplicação: redução de 41,26%;
- Consumo de energia nos processadores: redução de 41,43%;
- Consumo de energia nos roteadores: redução de 16,25%;
- Consumo de energia nos fios de interconexão: aumento de 4,70%.

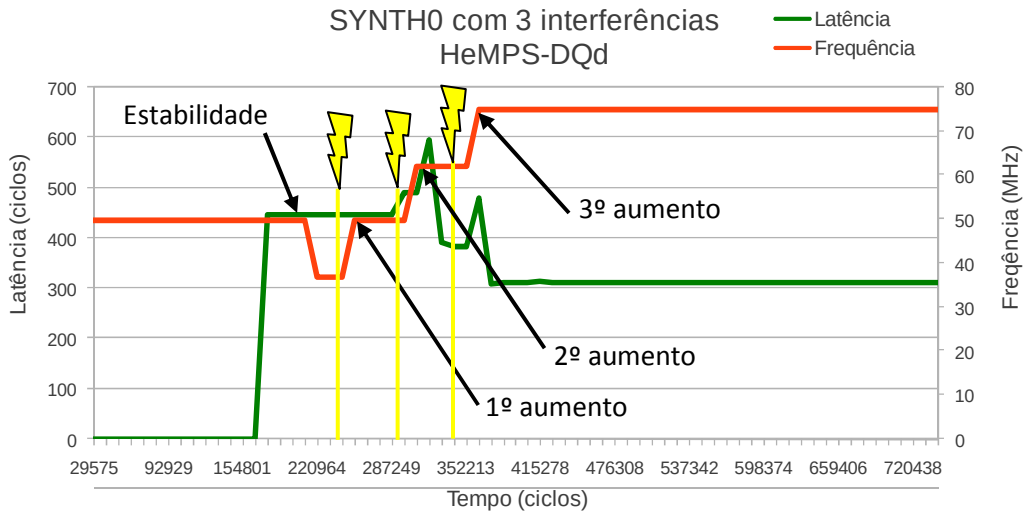


Figura 74 - Resultados de latência e frequência da aplicação SYNTH0 no cenário de teste 4, sobre a plataforma HeMPS-DQd.

O consumo de energia neste cenário de teste foi menor que na plataforma HeMPS, apesar do aumento da carga computacional nos processadores inserida pelos monitores. Isto se dá por conta da interferência gerada pelas aplicações D1, D2 e D3, que desaceleram o fluxo de comunicação da aplicação SYNTH0. Ao desacelerar a comunicação entre as tarefas, o número de vezes em que os monitores são acionados é reduzido e a carga computacional inserida é menor.

10.3 Resultados da aplicação MPEG2

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação MPEG2. O grafo de tarefas desta aplicação está descrito na Figura 41. Esta aplicação foi simulada utilizando-se três versões da plataforma HeMPS, a primeira é a

plataforma original sem qualquer mecanismo de controle de frequência ou QoS – HeMPS; a segunda é a plataforma com o mecanismo de DFS controlado pelo preenchimento do canal de comunicação (*pipe*) – HeMPS-D; e a terceira é a plataforma com controle da vazão da aplicação utilizando o mecanismo de DFS e descarte de mensagens de adaptabilidade – HeMPS-DQd.

A Tabela 15 exibe um resumo dos resultados obtidos nas simulações dos cenários de teste descritos no Capítulo 8, Seção 8.3. O requisito de latência imposto para esta aplicação foi de no mínimo 300 ciclos em cada comunicação (como descrito no Capítulo 8, Seção 8.5). Nesta tabela pode-se verificar que a HeMPS sempre atende os requisitos de QoS impostos no menor tempo de execução possível. As simulações na HeMPS-D exibem reduções de até 52% no consumo de energia e aumentos no tempo de execução de até 18%. O requisito de latência da aplicação não foi respeitado. Já os resultados obtidos na HeMPS-DQd mostram uma redução no consumo de energia do cenário de teste 1 de 5%, e aumento de 0,78% no cenário de teste 4. Os resultados mostram aumentos no tempo de execução da aplicação de até 7%, e o requisito de latência foram atendidos.

Tabela 15 - Resumo dos resultados das simulações utilizando a aplicação MPEG2.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	3.444.760	23.675,59	-	-	Sim
	HeMPS-D	3.967.964	11.308,16	15,19%	-52,24%	Não
	HeMPS-DQd	**3.585.413**	**22.437,04**	**4,08%**	**-5,23%**	**Sim**
Com 3 interferências	HeMPS	3.460.458	23.575,88	-	-	Sim
	HeMPS-D	4.098.921	11.234,23	18,45%	-52,35%	Não
	HeMPS-DQd	**3.732.581**	**23.760,09**	**7,86%**	**0,78%**	**Sim**

A comparação entre os resultados obtidos entre as simulações na HeMPS-D e na HeMPS-DQd mostram redução de até 9% no tempo de execução e aumento de até 111% no consumo de energia.

A Seção 10.3.1 detalha os resultados obtidos na simulação da aplicação MPEG2 sobre a plataforma HeMPS, a Seção 10.3.2 detalha os resultados nas simulações sobre a plataforma HeMPS-D e a Seção 10.3.3 detalha os resultados obtidos nas simulações sobre a plataforma HeMPS-DQd.

10.3.1 Simulações com a Plataforma HeMPS

Nesta Seção são apresentados os resultados das simulações da aplicação MPEG2

sobre a plataforma HeMPS. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.3.1.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS, sem a interferência de quaisquer outras aplicações. A Figura 75 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste oscila entre 74 e 222 ciclos, mas fica a maior parte do tempo em 218 ciclos.

A avaliação do consumo de energia da aplicação resultou em 23.675,585 mJ e um tempo de execução total de 3.444.760 ciclos. Os componentes do consumo de energia desta aplicação são: 23.596,925 mJ para os processadores, 78,605 mJ para os roteadores e 0,056 mJ para os fios de interconexão.

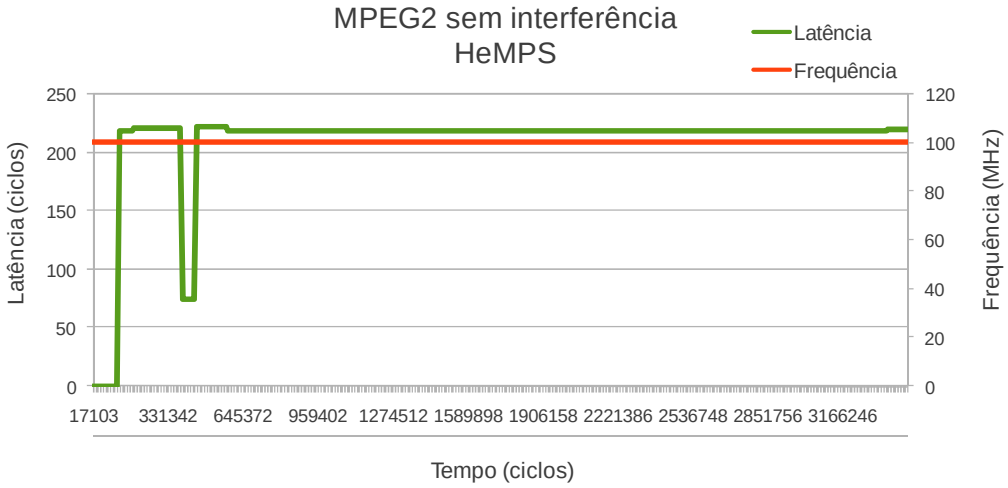


Figura 75 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS.

Os resultados de latência descritos acima foram utilizados para determinar o perfil de QoS para a aplicação MPEG2. A partir deste perfil de QoS, pode-se determinar os parâmetros de latência que deverão ser respeitados nas simulações com a plataforma HeMPS-DQd, estes parâmetros estão descritos no Capítulo 8, Seção 8.5.

10.3.1.2 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 76 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste oscila entre 218 e 234 ciclos, mas fica a maior parte do tempo em 218 ciclos.

A avaliação do consumo de energia da aplicação resultou em 23.575,875 mJ e um tempo de execução total de 3.460.458 ciclos. Os componentes do consumo de energia desta aplicação são: 23.496,434 mJ para os processadores, 79,365 mJ para os roteadores e 0,076 mJ para os fios de interconexão.

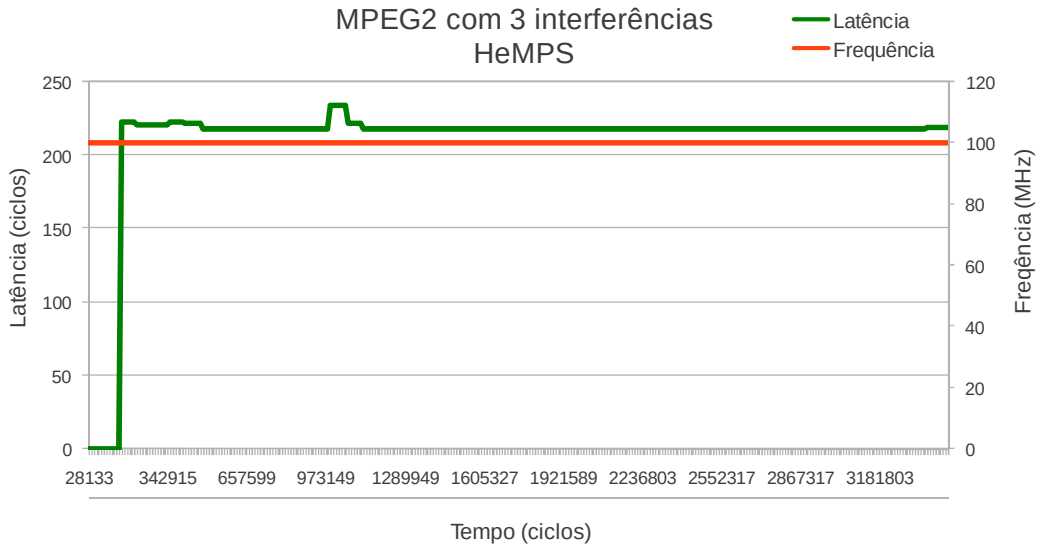


Figura 76 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS.

Verifica-se que nos cenários de teste 1 e 4 um menor impacto no tempo de execução – 0,46% (de 3.444.760 para 3.460.458). Isto deve-se ao fato que as tarefas das aplicações consomem uma maior parcela do tempo em computação, diferentemente do observado na aplicação sintética SYNTH0.

10.3.2 Simulações com a Plataforma HeMPS-D

Nesta Seção são apresentados os resultados das simulações da aplicação MPEG2 sobre a plataforma HeMPS-D, com o mecanismo de DFS proposto em [ROS12b]. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.3.2.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS-D, sem a interferência de quaisquer outras aplicações. A Figura 77 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. Neste gráfico verifica-se que a frequência varia entre 6,67 e 75 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 107 e 4358 ciclos. A avaliação do consumo de energia da aplicação resultou em 11.308,161 mJ e um tempo de execução total de 3.967.964 ciclos. Os componentes do consumo de energia desta aplicação são: 11.277,221 mJ para os processadores, 30,923 mJ para os roteadores e 0,017 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 15,19%;
- Consumo de energia da aplicação: redução de 52,24%;
- Consumo de energia nos processadores: redução de 52,21%;
- Consumo de energia nos roteadores: redução de 60,66%;
- Consumo de energia nos fios de interconexão: redução de 69,41%.

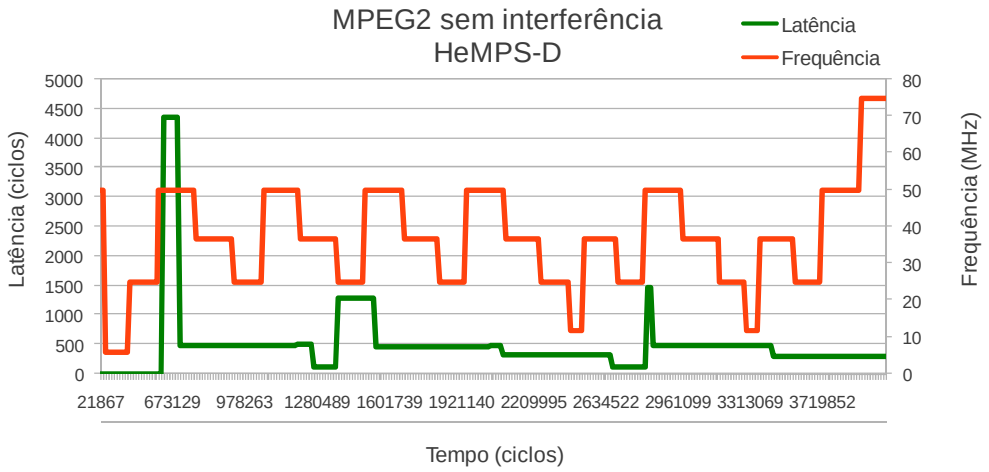


Figura 77 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS-D.

Observa-se agora, com uma aplicação real, um menor impacto no tempo total de execução quando aplicada a técnica de DFS (+15,19%).

10.3.2.2 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-D juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 78 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. Neste gráfico verifica-se que a frequência varia entre 25 e 87 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 2 e 1554 ciclos. A avaliação do consumo de energia da aplicação resultou em 11.234,225 mJ e um tempo de execução total de 4.098.921 ciclos. Os componentes do consumo de energia desta aplicação são: 11.202,250 mJ para os processadores, 31,954 mJ para os roteadores e 0,021 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 18,45%;
- Consumo de energia da aplicação: redução de 52,35%;
- Consumo de energia nos processadores: redução de 52,32%;
- Consumo de energia nos roteadores: redução de 59,74%;
- Consumo de energia nos fios de interconexão: redução de 71,61%.

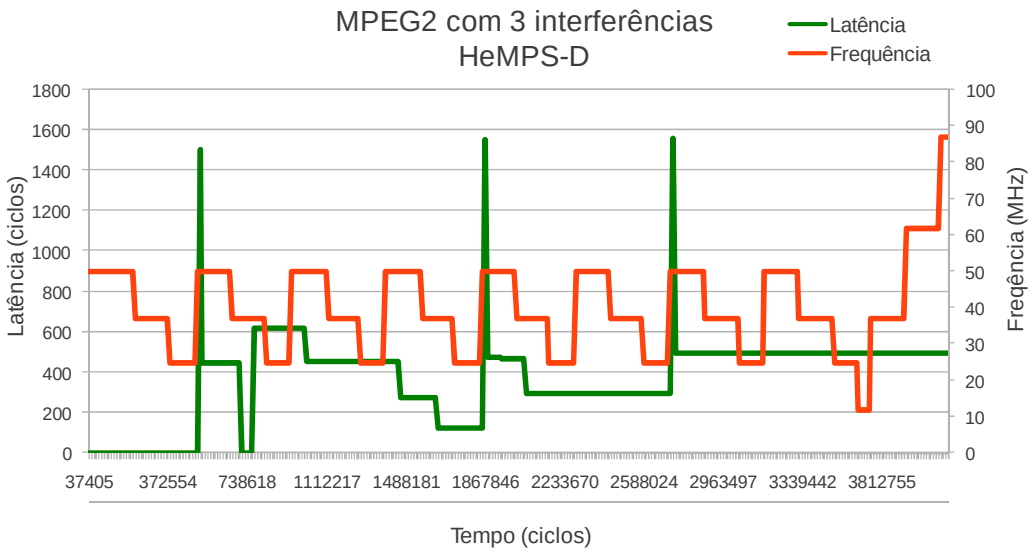


Figura 78 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS-D.

10.3.3 Simulações com a Plataforma HeMPS-DQd

Nesta Seção são apresentados os resultados das simulações da aplicação MPEG2 sobre a plataforma HeMPS-DQd, com o controle da latência proposto nesta Tese de Doutorado. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.3.3.1 Cenário de Teste 1

Neste cenário de teste a aplicação MPEG2 é executada sozinha na HeMPS-DQd, sem a interferência de quaisquer outras aplicações. A Figura 79 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 597.407 ciclos, como demonstrado na Figura 79. Logo após, em 686.097 ciclos, ocorrem cinco violações consecutivas de latência. Mais cinco violações consecutivas são detectadas em 738.947, 789.397, 840.051 e 890.327 ciclos, o que dispara novamente quatro aumentos na frequência de toda a aplicação. A partir do último aumento de frequência, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 100 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 223 e 588 ciclos quando controlada, e com picos de até 3502 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 22.437,035 mJ e um tempo de execução total de 3.585.413 ciclos. Os componentes do consumo de energia desta aplicação são: 22.377,558 mJ para os processadores, 59,443 mJ para os roteadores e 0,034 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-

se:

- Tempo de execução: aumento de 4,08%;
- Consumo de energia da aplicação: redução de 5,23%;
- Consumo de energia nos processadores: redução de 5,17%;
- Consumo de energia nos roteadores: redução de 24,38%;
- Consumo de energia nos fios de interconexão: redução de 39,33%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 9,64%;
- Consumo de energia da aplicação: aumento de 98,41%;
- Consumo de energia nos processadores: aumento de 98,43%;
- Consumo de energia nos roteadores: aumento de 92,23%;
- Consumo de energia nos fios de interconexão: aumento de 98,32%.

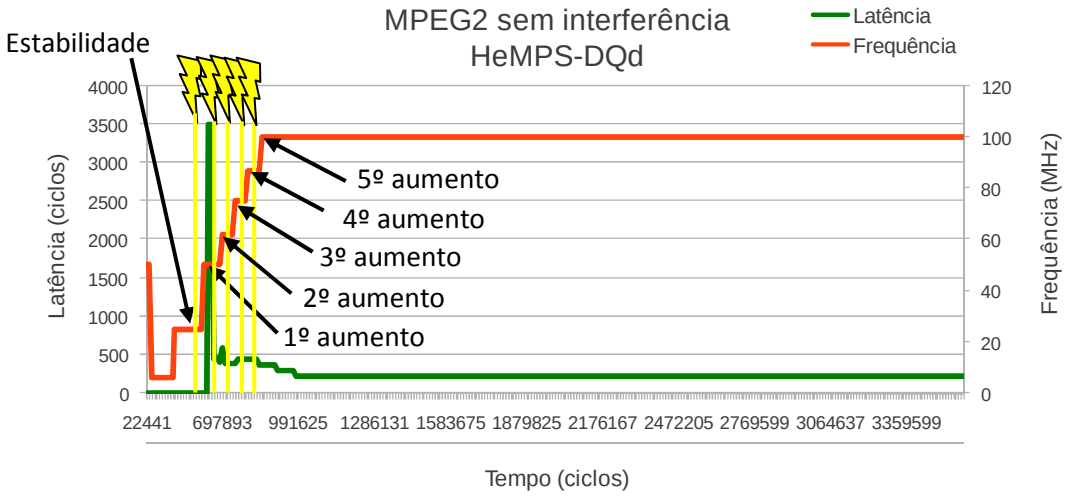


Figura 79 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 1, sobre a plataforma HeMPS-DQd.

10.3.3.2 Cenário de Teste 4

Neste cenário de teste a aplicação MPEG2 é executada na HeMPS-DQd juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação MPEG2. A Figura 80 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 193.329 ciclos, como demonstrado na Figura 80. Logo após, em 581.213 ciclos, ocorrem cinco violações consecutivas de latência. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 632.536, 682.986, 733.429 e 783.915 ciclos, o que dispara novamente quatro aumentos na frequência de toda a aplicação. A partir do último aumento de frequência, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 25 e 100 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 173 e 450 ciclos quando controlada, e com picos de até 1.498 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 23.760,082 mJ e um tempo de execução total de 3.732.581 ciclos. Os componentes do consumo de energia desta aplicação são: 23.686,703 mJ para os processadores, 73,324 mJ para os roteadores e 0,055 mJ para os fios de interconexão.

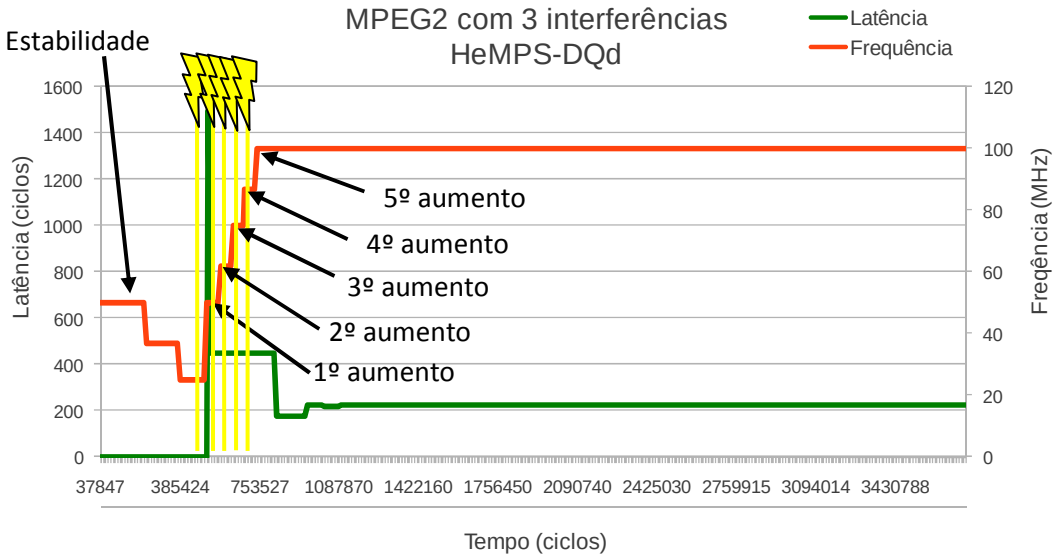


Figura 80 - Resultados de latência e frequência da aplicação MPEG2 no cenário de teste 4, sobre a plataforma HeMPS-DQd.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 7,86%;
- Consumo de energia da aplicação: aumento de 0,78%;
- Consumo de energia nos processadores: aumento de 0,81%;
- Consumo de energia nos roteadores: redução de 7,61%;
- Consumo de energia nos fios de interconexão: redução de 27,15%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 8,94%;
- Consumo de energia da aplicação: aumento de 111,50%;
- Consumo de energia nos processadores: aumento de 111,45%;
- Consumo de energia nos roteadores: aumento de 129,47%;
- Consumo de energia nos fios de interconexão: aumento de 156,63%.

Os resultados neste cenário de teste mostram um elevado consumo de energia em relação à plataforma HeMPS-D, inserido pelo mecanismo de controle da latência. A

plataforma HeMPS-D prioriza a otimização dos canais de comunicação, mantendo a frequência em níveis reduzidos nos PEs. Já a plataforma HeMPS-DQd realiza o controle da latência, modificando a frequência dos PEs de acordo com a latência nas comunicações entre duas tarefas. Desta forma, a frequência é fixada em um valor maior que na HeMPS-D, o que explica o consumo de energia maior na plataforma HeMPS-DQd.

10.4 Resultados da aplicação SYNTH1

Nesta Seção são apresentados os resultados obtidos nas simulações com a aplicação SYNTH1. O grafo de tarefas desta aplicação está descrito na Figura 40. Esta aplicação foi simulada utilizando-se quatro versões da plataforma HeMPS, a primeira é a plataforma original sem qualquer mecanismo de controle de frequência ou QoS – HeMPS; a segunda é a plataforma com o mecanismo de DFS controlado pelo preenchimento do canal de comunicação (*pipe*) – HeMPS-D; e a terceira é a plataforma com controle da vazão da aplicação utilizando o mecanismo de DFS com descarte de mensagens de adaptabilidade – HeMPS-DQd.

A Tabela 16 exibe um resumo dos resultados obtidos nas simulações acima descritas. O requisito de latência imposto para esta aplicação foi de no mínimo 300 ciclos em cada comunicação (como descrito no Capítulo 8, Seção 8.5). Nesta tabela pode-se verificar que a HeMPS sempre atende o requisito de latência imposto à aplicação no menor tempo de execução possível, e com o menor consumo de energia possível atendendo o requisito de latência imposto à aplicação. A plataforma HeMPS-D possui os piores resultados de tempo de execução (até 174% maior) e energia (até 311% maior) sem atendimento do requisito de latência. Os resultados apresentados na plataforma HeMPS-DQd são até 115% maiores em tempo de execução e até 213% maiores em consumo de energia. Esta plataforma sempre atende o requisito de latência imposto à aplicação.

Tabela 16 - Resumo dos resultados das simulações utilizando a aplicação SYNTH1.

Cenário	Plataforma	Tempo de Simulação (ciclos)	Energia Dissipada (mJ)	Diferença Tempo	Diferença Energia	Atendeu Requisito de QoS?
Sem interferência	HeMPS	2.138.754	2.332,81	-	-	Sim
	HeMPS-D	5.861.273	9.610,99	174,05%	311,99%	Não
	HeMPS-DQd	**4.613.569**	**7.310,13**	**115,71%**	**213,36%**	**Sim**
Com 3 interferências	HeMPS	2.149.369	2.335,15	-	-	Sim
	HeMPS-D	4.565.725	5.973,47	112,42%	155,81%	Não
	HeMPS-DQd	**2.553.050**	**3.166,92**	**18,78%**	**35,62%**	**Sim**

Os resultados acima mostram um aumento considerável no consumo de energia,

isto acontece por conta do aumento na carga computacional inserida devido à execução dos monitores de latência e vazão. Estes monitores são acionados em cada comunicação entre duas tarefas, e para tarefas com um perfil de alta comunicação, esta deficiência no mecanismo de controle do QoS na HeMPS fica mais evidenciado.

A Seção 10.4.1 detalha os resultados obtidos na simulação da aplicação SYNTH1 sobre a plataforma HeMPS, a Seção 10.4.2 detalha os resultados nas simulações sobre a plataforma HeMPS-D e a Seção 10.4.3 detalha os resultados obtidos nas simulações sobre a plataforma HeMPS-DQd.

10.4.1 Simulações com a Plataforma HeMPS

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH1 sobre a plataforma HeMPS. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.4.1.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH1 é executada sozinha na HeMPS, sem a interferência de quaisquer outras aplicações. A Figura 81 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste irá oscilar entre 165 e 233 ciclos. A avaliação do consumo de energia da aplicação resultou em 2.332,807 mJ e um tempo de execução total de 2.138.754 ciclos. Os componentes do consumo de energia desta aplicação são: 2.272,400 mJ para os processadores, 60,289 mJ para os roteadores e 0,118 mJ para os fios de interconexão.

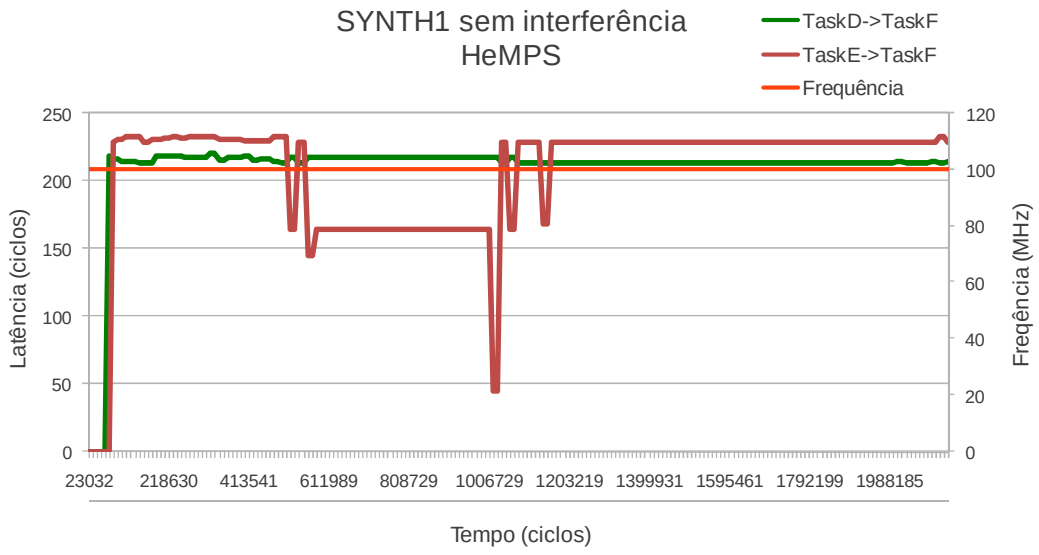


Figura 81 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 1, sobre a plataforma HeMPS.

Os resultados de latência descritos acima foram utilizados para determinar o perfil de QoS para a aplicação SYNTH1. A partir deste perfil de QoS, pode-se determinar os parâmetros de QoS que deverão ser respeitados nas simulações com a plataforma HeMPS-DQd.

10.4.1.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH1 é executada na HeMPS juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH1. A **Erro! Fonte de referência não encontrada.** exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. A frequência permanece fixa em 100 MHz, pois esta plataforma não possui mecanismo de DFS, e a latência neste cenário de teste irá oscilar entre 23 e 233 ciclos.

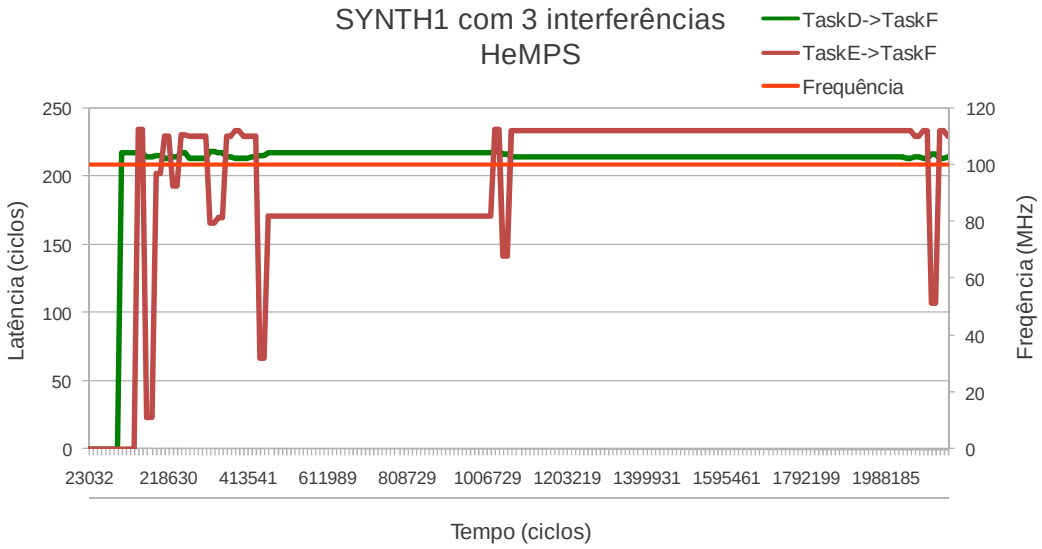


Figura 82 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 4, sobre a plataforma HeMPS.

A avaliação do consumo de energia da aplicação resultou em 2.335,153 mJ e um tempo de execução total de 2.149.369 ciclos. Os componentes do consumo de energia desta aplicação são: 2.273,655 mJ para os processadores, 61,368 mJ para os roteadores e 0,130 mJ para os fios de interconexão.

10.4.2 Simulações com a Plataforma HeMPS-D

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH1 sobre a plataforma HeMPS-D, com o mecanismo de DFS proposto em [ROS12b]. Os cenários de teste aqui avaliados são descritos na Seção 8.3.

10.4.2.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH1 é executada sozinha na HeMPS-D, sem a interferência de quaisquer outras aplicações. A **Erro! Fonte de referência não**

encontrada. exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 1. Neste gráfico verifica-se que a frequência varia entre 6,67 e 75 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 0 e 662 ciclos. A avaliação do consumo de energia da aplicação resultou em 9.610,999 mJ e um tempo de execução total de 5.861.273 ciclos. Os componentes do consumo de energia desta aplicação são: 9.521,132 mJ para os processadores, 89,779 mJ para os roteadores e 0,088 mJ para os fios de interconexão.

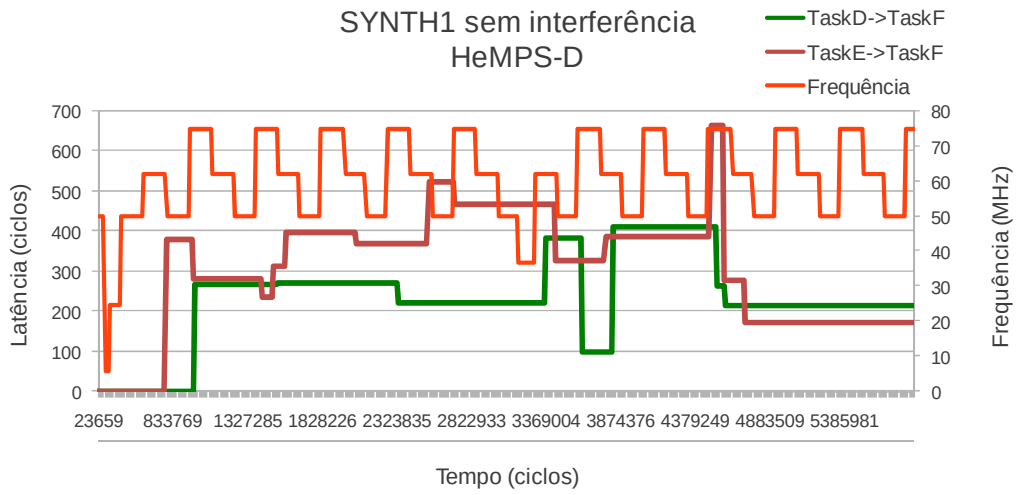


Figura 83 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 1, sobre a plataforma HeMPS-D.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 174,05%;
- Consumo de energia da aplicação: aumento de 311,99%;
- Consumo de energia nos processadores: aumento de 318,99%;
- Consumo de energia nos roteadores: aumento de 48,91%;
- Consumo de energia nos fios de interconexão: redução de 25,56%.

10.4.2.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH1 é executada na HeMPS-D juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH1. A Figura 84 exibe o gráfico com os perfis de latência e frequência obtidos na execução do cenário de teste 4. Neste gráfico verifica-se que a frequência varia entre 37 e 87 MHz, devido ao mecanismo de DFS. A latência irá oscilar entre 119 e 431 ciclos. A avaliação do consumo de energia da aplicação resultou em 5.973,472 mJ e um tempo de execução total de 4.565.725 ciclos. Os componentes do consumo de energia desta aplicação são: 5.900,056 mJ para os processadores, 73,325 mJ para os roteadores e 0,091 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 112,42%;
- Consumo de energia da aplicação: aumento de 155,81%;
- Consumo de energia nos processadores: aumento de 159,50%;
- Consumo de energia nos roteadores: aumento de 19,49%;
- Consumo de energia nos fios de interconexão: redução de 30,17%.

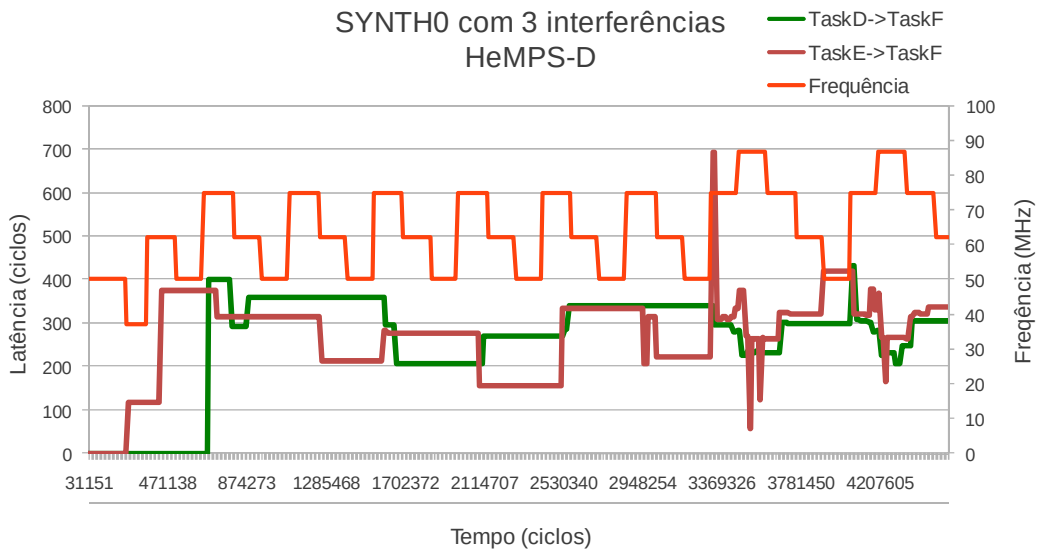


Figura 84 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 4, sobre a plataforma HeMPS-D.

O alto consumo de energia se dá pelo aumento da carga computacional inserida pelos monitores de vazão e latência. Estes monitores são ativados em cada comunicação entre as tarefas da aplicação, e para tarefas com o perfil de alta comunicação esta carga computacional é elevada.

10.4.3 Simulações com a Plataforma HeMPS-DQd

Nesta Seção são apresentados os resultados das simulações da aplicação SYNTH1 sobre a plataforma HeMPS-DQd, com o controle da vazão proposto nesta Tese de Doutorado. Os cenários de teste aqui avaliados são descritos na Seção 8.3. O requisito de latência imposto à aplicação é de 300 ciclos nas comunicações entre tarefas (descrito no Capitulo 8, Seção 8.5).

10.4.3.1 Cenário de Teste 1

Neste cenário de teste a aplicação SYNTH1 é executada sozinha na HeMPS-DQd, sem a interferência de quaisquer outras aplicações. A Figura 85 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 1. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 593.170 ciclos, como demonstrado na Figura 85. Logo após, em 697.387 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 748.193 ciclos, o que dispara novamente um aumento na frequência de toda a aplicação. A partir deste último aumento de frequência, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 6,67 e 100 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 181 e 215 ciclos quando controlada, e com picos de até 2538 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 7.310,125 mJ e um tempo de execução total de 4.613.569 ciclos. Os componentes do consumo de energia desta aplicação são: 7.151,738 mJ para os processadores, 158,246 mJ para os roteadores e 0,142 mJ para os fios de interconexão.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 115,71%;
- Consumo de energia da aplicação: aumento de 213,36%;
- Consumo de energia nos processadores: aumento de 214,72%;
- Consumo de energia nos roteadores: aumento de 162,48%;
- Consumo de energia nos fios de interconexão: aumento de 20,32%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 21,29%;
- Consumo de energia da aplicação: redução de 23,94%;
- Consumo de energia nos processadores: redução de 24,89%;
- Consumo de energia nos roteadores: aumento de 76,26%;
- Consumo de energia nos fios de interconexão: aumento de 61,64%.

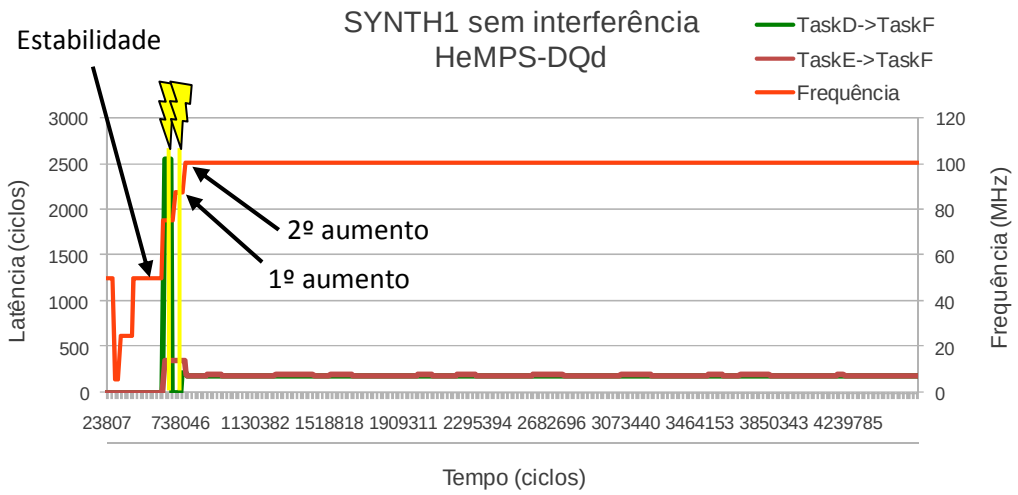


Figura 85 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 1, sobre a plataforma HeMPS-DQd.

10.4.3.2 Cenário de Teste 4

Neste cenário de teste a aplicação SYNTH1 é executada na HeMPS-DQd juntamente com as aplicações D1, D2 e D3. Estas aplicações geram uma interferência no fluxo de comunicação de algumas tarefas da aplicação SYNTH1. A Figura 86 exibe o gráfico com os perfis de vazão e frequência obtidos na execução do cenário de teste 4. Neste cenário de teste, o monitor de *steady-state* identificou o estado de estabilidade em 194.776 ciclos, como demonstrado na Figura 86. Logo após, em 425.126 ciclos, ocorrem cinco violações consecutivas da vazão mínima. Isto irá disparar o módulo de adaptabilidade que irá aumentar a frequência de toda a aplicação. Mais cinco violações consecutivas são detectadas em 437.160 e 1.020.526 ciclos, o que dispara novamente dois aumentos na frequência de toda a aplicação. A do último aumento de frequência, a latência permanece abaixo do valor solicitado como requisito, e a frequência permanece constante até o final do cenário de teste.

Neste gráfico verifica-se que a frequência varia entre 37 e 100 MHz durante a execução da aplicação, devido ao mecanismo de DFS controlado pelo módulo de adaptabilidade. A latência irá oscilar entre 222 e 346 ciclos quando controlada, e com picos de até 946 ciclos antes do controle de latência começar. A avaliação do consumo de energia da aplicação resultou em 3.166,918 mJ e um tempo de execução total de 2.553.050 ciclos. Os componentes do consumo de energia desta aplicação são: 3.098,224 mJ para os processadores, 68,576 mJ para os roteadores e 0,118 mJ para os fios de interconexão.

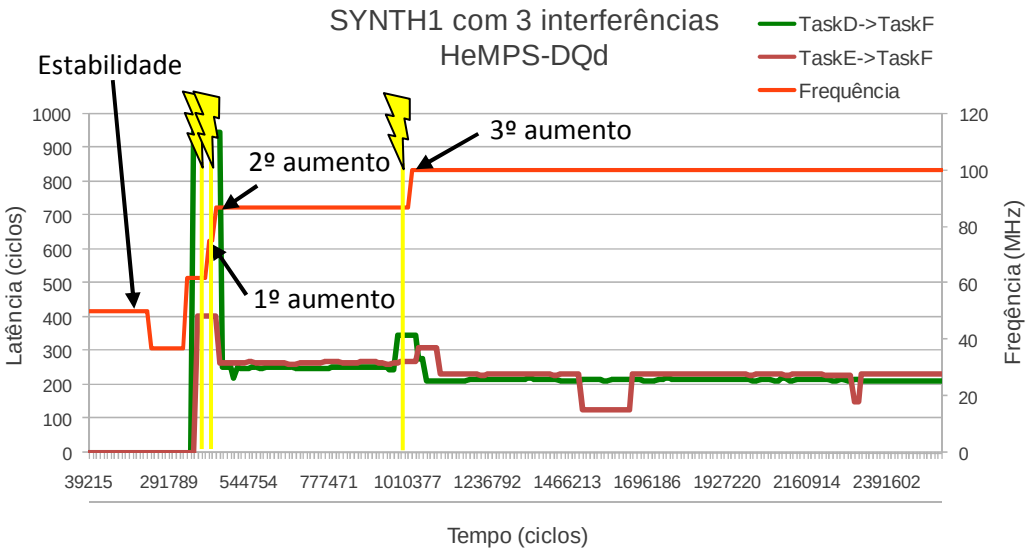


Figura 86 - Resultados de latência e frequência da aplicação SYNTH1 no cenário de teste 4, sobre a plataforma HeMPS-DQ.

Comparando-se estes valores com aqueles obtidos com a plataforma HeMPS têm-se:

- Tempo de execução: aumento de 18,78%;
- Consumo de energia da aplicação: aumento de 35,62%;
- Consumo de energia nos processadores: aumento de 36,27%;
- Consumo de energia nos roteadores: aumento de 11,75%;
- Consumo de energia nos fios de interconexão: redução de 8,85%.

Comparando-se os resultados das plataformas HeMPS-D e HeMPS-DQd têm-se:

- Tempo de execução: redução de 44,08%;
- Consumo de energia da aplicação: redução de 46,98%;
- Consumo de energia nos processadores: redução de 47,49%;
- Consumo de energia nos roteadores: redução de 6,48%;
- Consumo de energia nos fios de interconexão: aumento de 30,53%

10.5 Conclusão

Os resultados apresentados neste Capítulo mostram que o mecanismo proposto de controle de latência consegue cumprir com o seu objetivo, evitando que o MPSoC desrespeite o requisito de QoS imposto às aplicações.

Os resultados para a aplicação SYNTH0 mostram que o mecanismo proposto permitiu controlar o requisito de latência e reduzir o consumo de energia do MPSoC, apesar do maior tempo de execução da aplicação controlada.

Os resultados para a aplicação MPEG2, a única aplicação real avaliada, também

mostram que o mecanismo proposto permitiu o controle da latência, objetivo principal deste.

Já os resultados com a aplicação SYNTH1 mostram que o mecanismo proposto insere uma alta carga computacional devido aos monitores, em aplicações com tarefas de alta comunicação e pouca computação. Os resultados mostram que esta deficiência é mitigada quando o fluxo de comunicações entre as tarefas é desacelerado, e assim o mecanismo proposto produz os melhores resultados no consumo de energia.

Da mesma forma que no Capítulo anterior, o aumento da carga computacional inserida pelos monitores penalizaram os resultados no que se refere ao tempo de execução das aplicações, e por consequência na energia consumida destas. Entretanto, o principal objetivo, atendimento aos requisitos de QoS foi mantido. Assim, o desenvolvimento de monitores não intrusivos é uma direção correta de trabalho futuro a seguir.

11 CONCLUSÃO

A ideia deste trabalho originou-se em duas necessidades nos MPSoCs modernos, a redução no consumo de energia e a qualidade nos serviços prestados. Os MPSoCs modernos possuem um grande poder computacional e podem executar dezenas de aplicações em paralelo. Entretanto, tais aplicações disputam a utilização dos elementos que compõem este MPSoC, tais como elementos de processamento, de comunicação e de armazenamento. Tal disputa leva a conflitos de utilização entre as aplicações, e destes conflitos derivam as perdas na qualidade de serviço das aplicações. Estes MPSoCs estão sendo inseridos nos dispositivos móveis, como smartphones e tablets, e sendo assim necessitam controlar seu consumo de energia.

O presente trabalho buscou uma solução que satisfizesse estas duas necessidades, um controle de qualidade de serviço em MPSoCs com baixo consumo de energia. Foi escolhida a plataforma HeMPS como o MPSoC alvo para este trabalho, devido à proximidade do Autor a esta plataforma e ao fato de ela contar com um mecanismo de escalabilidade dinâmica de frequência (DFS) proposto em [ROS12b]. Este mecanismo reduz o consumo de energia na HeMPS, porém o controle do DFS visa à otimização dos canais de comunicação na plataforma e não a garantia da qualidade de serviço das aplicações.

Buscou-se então, desenvolver um controle adicional do DFS da HeMPS, aliando-se o baixo consumo de energia já implementado originalmente com o controle efetivo da qualidade de serviço nas aplicações. Para este fim foram adaptados os monitores de outros trabalhos da HeMPS, a fim de verificar os parâmetros de vazão e latência na HeMPS. Os dados destes monitores foram alimentados nos módulos do mecanismo de controle dos parâmetros de QoS, e este mecanismo controla o DFS nos elementos de processamento a fim de manter os valores de vazão e latência dentro dos requisitos alocados à aplicação. Um modelo de consumo de energia foi desenvolvido para aferir o consumo de energia da HeMPS. Este modelo foi utilizado para determinar se o mecanismo proposto consegue reduzir o consumo de energia da plataforma.

Testou-se a plataforma utilizando duas aplicações sintéticas com um perfil de mais comunicação e pouca computação, e uma aplicação real que possui muita computação e pouca comunicação. Em alguns cenários de teste estas aplicações executam sozinhas na HeMPS, e em outras elas disputam a rede intra-chip com uma, duas e três outras aplicações. Estas outras aplicações são utilizadas somente para interferir nos fluxos de comunicações das aplicações avaliadas.

Os resultados destes cenários de testes mostraram que o mecanismo proposto cumpre o objetivo de controlar os parâmetros de vazão e latência, mantendo-os nos limites impostos como requisitos de QoS às aplicações. Os resultados também mostraram que o mecanismo proposto possui como deficiência a inserção de uma maior carga

computacional adicional nos processadores, devido aos monitores de vazão e latência que são ativados em cada comunicação entre tarefas. Esta deficiência fica evidenciada na execução de aplicações com tarefas com alta demanda de comunicação e pouca computação. Nestes casos, os monitores são ativados tantas vezes que a carga computacional que eles inserem aumentam significativamente o consumo de energia da HeMPS.

11.1 Direcionamentos para Trabalhos Futuros

O desenvolvimento de estruturas de monitoramento híbrido (parte em hardware e parte em software) é extremamente complexo, pois a parte em hardware pode se tornar complexa demais e atrasar o fluxo de dados do circuito; e a parte em software pode inserir uma carga de processamento adicional tão grande que inviabiliza a execução das aplicações monitoradas. Este último problema ocorre no mecanismo de controle de QoS em MPSoCs proposto nesta Tese de Doutorado. Os monitores aqui propostos estão sendo ativados em cada comunicação entre tarefas da aplicação, e a execução destes introduz uma carga computacional no processador. Para aplicações com tarefas com baixa comunicação, este problema não afeta muito o processador pois este já executa tarefas com muita computação. Para aplicações com tarefas de muita comunicação este problema fica evidenciado, inserindo uma carga computacional adicional muito grande nos processadores.

O desenvolvimento de novos monitores, com ativação independente da comunicação entre as tarefas é elencado como o principal trabalho futuro desta Tese de Doutorado. Novos monitores permitirão a redução no consumo de energia dos processadores, e assim, a redução do consumo de energia nas aplicações com os requisitos de QoS controlados.

O modelo de avaliação do consumo de energia utilizou uma aproximação linear para estimar o consumo de energia nos processadores em diferentes frequências. Um trabalho futuro será a avaliação do consumo de energia para diferentes tensões de alimentação, de forma a estender o trabalho para DVFS. O consumo de energia da memória RAM do processador não foi avaliado. Outro trabalho futuro será a utilização de um modelo de consumo de energia para memórias RAM, como o exemplo do modelo CACTI descrito em [CAC14].

O mecanismo de controle de parâmetros de QoS proposto nesta Tese de Doutorado foi desenvolvido com foco em processadores que admitem a execução de diversas tarefas (multi-tarefas). O funcionamento do mecanismo para esta característica não foi avaliado, e esta avaliação é elencada como um trabalho futuro desta Tese de Doutorado.

REFERÊNCIAS BIBLIOGRÁFICAS

[ARM13] ARM Coresight, capturado em *http://www.arm.com/products/solutions/CoreSight.html*, em junho de 2013.

[ARO05] Arora, D.; Ravi, S.; Raghunathan, A.; Jha, N. K. “Secure embedded processing through hardware-assisted run-time monitoring”. In: DATE, 2005, pp. 178-183.

[ATI08] Atienza, D.; De Micheli, G.; Benini, L.; Ayala, J. L.; Valle, P. G. D.; DeBole, M.; Narayanan, V. “Reliability-aware design for nanometer-scale devices”. In: ASP-DAC, 2008, PP. 549-554.

[BAG08] Bagherzadeh, N.; Matsuura, M. “Performance Impact of Task-to-Task Communication Protocol in Network-on-Chip”. In: ITNG, 2008, pp. 1101-1106.

[BEN98] Benini, L.; De Micheli, G. “Dynamic Power Management: Design Techniques and CAD Tools”. Kluwer Academic Publishers, Boston, MA, 1998.

[BEN00] Benini, L.; Bogliolo, A.; De Micheli, G. "A survey of design techniques for system-level dynamic power management". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.8, no.3, pp. 299-316, Jun. 2000.

[CAC14] CACTI - An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model, capturado em *http://www.hpl.hp.com/research/cacti/*, em julho de 2014.

[CAR07a] Carvalho, E.; Calazans, N.; Moraes, F. “Congestion-aware task mapping in NoC-based MPSoCs with dynamic workload”. In: ISVLSI, 2007, pp. 459-460.

[CAR07b] Carara, E.; Calazans, N.; Moraes, F.; “Router Architecture for High-Performance NoCs”. In: Symposium on Integrated Circuits and Systems (SBCCI'07), 2007, pp. 111-116.

[CAR08] Carara, E.; Calazans, N.; Moraes, F.; “A New Router Architecture for High-Performance Intrachip Networks”. Journal of Integrated Circuits and Systems, v.3(1), 2008, pp. 23-31.

[CAR09a] Carara, E. et al. “HeMPS - A Framework for NoC-Based MPSoC Generation”, In: ISCAS, 2009, pp. 1345–1348.

[CAR09b] Carara, E.; Calazans, N.; Moraes, F. “Managing QoS Flows at Task Level in NoC-Based MPSoCs”. In: VLSI-SoC, 2009, pp. 133-138.

[CAR09c] Carro, L.; Beck, A. “Adaptability: the Key for Future Embedded Systems”. Capturado em: http://esweek09.inrialpes.fr/tutorials/tutorial2.shtml, Junho 2013.

[CAR11] Carara, E. “Serviços de comunicação diferenciados em sistemas multiprocessados em chip baseados em redes intra-chip”. Tese de Doutorado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 107p.

[CHA09] Chabloz, J.-M.; Hemani, A. "A flexible communication scheme for rationally-related clock frequencies". In: ICCD, 2009, pp. 109-116.

[CHA11] Chakraborty, K.; Roy, S. "Topologically Homogeneous Power-Performance Heterogeneous Multicore Systems". In: DATE, 2011, pp. 125-130.

[CHE00] Chelcea, T.; Nowick, S. “A low latency FIFO for mixed-clock systems”. In: Computer Society Workshop on VLSI, 2000, pp. 119-126.

[CHI00] Chiu, G. “The Odd-Even Turn Model for Adaptive Routing”. IEEE Transactions on Parallel and Distributed Systems, v.7(11), 2000, pp. 729-738.

[CRI07] Woszezenki, C. R. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2007, 121p.

[DAL87] Dally, W. J.; Seitz, C. L. “Deadlock-Free Message Routing in Multiprocessors Interconnection Networks”. IEEE Transactions on Computers v.36(5), 1987, pp.547-553.

[EBR10] Ebrahimi, M.; Daneshtalab, M.; Liljeberg, P.; Tenhunen, H. “HAMUM – A Novel Routing Protocol for Unicast and Multicast Traffic in MPSoCs”. In: 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP’10), 2010, pp. 525 – 532.

[FAT11] Fattah, M.; Daneshtalab, M.; Liljeberg, P.; Plosila, J. “Exploration of MPSoC monitoring and management systems”. In: ReCoSoC, 2011, pp. 1-3.

[FIL12] Johann F. S.; “Suporte para aplicações dinâmicas em sistemas multiprocessados intra-chip homogêneos”. Tese de Doutorado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2012, 160p.

[FLY05] Flynn, M. J.; Hung, P. “Microprocessor design issues: Thoughts on the road ahead”. In: IEEE Micro, Vol. 25, 2005, pp. 16-31.

[GAR09] Garg, S.; Marculescu, D.; Marculescu, R.; Orgras, U. “Technology-driven limits on DVFS controlability of multiple voltage-frequency island designs: A system-level perpective.” In: DAC, 2009, pp. 818-821.

[GAR10] Garg, S.; Marculescu, D.; Marculescu, R. “Custom feedback control: Enabling truly scalable on-chip Power management for MPSoCs”. In: ISLPED, 2010, pp. 425-430.

[GIL10] Gilabert, F.; Gómez, M.E.; Medardoni, S.; Bertozzi, D. “Improved Utilization of NoC Channel Bandwidth by Switch Replication for Cost-Effective Multi-Processor Systems-on-Chip”. In: International Symposium on Networks-on-Chip (NOCS’10), 2010, pp. 165 – 172.

[GLA94] Glass, C. J.; Ni, L. M. “The Turn Model for Adaptive Routing”. Journal of the Association for Computing Machinery, v.41(5), 1994, pp. 874-902.

[GOO10] Goossens, K.; Molnos, A.; Ambrose, J. A.; Nelson, A.; Stefan, R.; Cotofana, S. “A composable, energy-managed, real-time MPSoC platform”. In: OPTIM, 2010, pp. 870-876.

[GOR07] Gordon-Ross, A.; Viana, P.; Vahid, F.; Najjar, W.; Barros, E. “A one-shot configurable-cache tuner for improved energy and performance”. In: DATE, 2007, pp. 755-760.

[GUI08a] Guindani, G.; Reinbrecht, C.; da Rosa, T.; Calazans, N.; Moraes, F. “NoC Power Estimation at the RTL Abstraction Level.” In: ISVLSI, 2008, pp. 475-478.

[GUI08b] Guindani, Guilherme Montez. “Estimativa e redução da dissipação de potência em redes intra-chip com chaveamento por pacotes”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2008, 95p.

[HER09] Herbert, S.; Marculescu, D. “Variation-aware dynamic voltage/frequency scaling”. In: HPCA, 2009, pp.301-312.

[HON10] Hong, S.; Kim, H. “An integrated GPU power and performance model”. In: ISCA, 2010, pp. 280-289.

[HOP11] Höppner, S.; Henker, H.; Eisenreich, H.; Schüffny, R. “An open-loop clock generator for fast frequency scaling in 65nm CMOS technology”. In: MIXDES, 2011, pp. 264-269.

[HOP12] Höppner, S.; Shao, C.; Eisenreich, H.; Ellguth, G.; Ander, M.; Schüffny, R. “A power management architecture for fast per-core DVFS in heterogeneous MPSoCs”. In: ISCAS, 2012, pp. 261-264.

[HOW10] Howard, J.; Dighe, S.; Hoskote, Y. “A 48-Core IA-32 message passing processor with DVFS in 45nm CMOS”. In: ISSCC, 2010, pp. 108-109.

[HU04] Hu, J.; Marculescu, R. “DyAD - Smart Routing for Networks-on-Chip”. In: Design Automation Conference (DAC’04), 2004, pp. 260-263.

[IBM14] IBM 10LPE/10RFE CMOS Process. Disponível em *http://www.mosis.com/vendors/view/ibm/10lprfe*. 2014.

[IEE09] IEEE-ISTO. “The Nexus 5001 forum standard for a global embedded processor debug interface”. Disponível em *http://www.nexus5001.org*, 2009.

[IL05] Il-Gu L; Jin L; Sin-Chong P. "Adaptive routing scheme for NoC communication architecture". In: International Conference on Advanced Communication Technology (ICACT), 2005. pp.1180-1184.

[ISC03] Isci, C.; Martonosi, M. “Runtime power monitoring in high-end processors: Methodology and empirical data”. In: MICRO, 2003, pp. 93-104.

[JAL10] Jalier, C.; Lattard, D.; Jerraya, A.A.; Sassatelli, G.; Benoit, P.; Torres, L. "Heterogeneous vs Homogeneous MPSoC Approaches for a Mobile LTE Modem". In: DATE, 2010, pp. 184-189.

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

[KAH74] G. Kahn. “The semantics of a simple language for parallel programming”. In: Information Processing, 1974, pp 471-475.

[KAK11] Kakoee, M. R.; Bertacco, V.; Benini, L. “ReliNoC: A Reliable Network for Priority-Based on-Chip Communication”. In: Design, Automation and Test in Europe (DATE’11), 2011, pp. 491-496.

[KOR12] Kornaros, G.; Pnevmatikatos, D. “A survey and taxonomy of on-chip monitoring of multicore systems-on-chip”. In: ACM Transactions on Design Automation of Electronic Systems, Vol. 18, No. 2, Article 17, 2012, 38p.

[LEA97] Leatherman, R. “On-chip instrumentation approach to system-on-chip development”. First Silicon Solutions, 1997.

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

[LIN94] Lin, X.; McKinley, P. K.; Ni, L. M. “Deadlock-free Multicast Wormhole Routing in 2-D Mesh Multicomputers”. IEEE Transactions on Parallel and Distributed Systems, v.5(8), 1994, pp. 793-804.

[LOR98] Lorch, J.; Smith, A. “Software strategies for portable computer energy management”. IEEE Personal Communications, vol.5, no.3, pp. 60–73, Jun. 1998.

[LUK05] Luk, C. K.; Cohn, R.; Muth, R.; Path, H.; Klauser, A.; Lowney, G.; Wallace, S.; Reddi, V. J.; Hazelwood, K. “Pin: building customized program analysis tools with dynamic instrumentation”. In: SIGPLAN Not., Vol. 40, 2005, pp. 190-200.

[MAD13] Madalozzo, G. “Controle adaptativo para atendimento a requisitos de aplicações em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 68p.

[MAN10] Mansouri, I.; Clermidy, F.; Benoit, P.; Torres, L. “A run-time distributed cooperative approach to optimize power consumption in MPSoCs”. In: SoCC, 2010, pp. 25-30.

[MAN11] Mandelli, M.; Moraes, F.G. “Mapeamento dinâmico de aplicações para MPSoCs homogêneos”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 106p.

[MAR07] Marescaux, T.; Corporaal, H. “Introducing the supergt network-on-chip: Supergt QoS, more than just GT”. In: DAC, 2007, pp. 116-121.

[MEL06] Mello, A.; Tedesco, L.; Calazans, N.; Moraes, F. "Evaluation of Current QoS Mechanisms in Networks on Chip". In: International Symposium on Systemon-Chip (SOC’06), 2006, 4p.

[MUR05] Muralli, S.; Theocharides, T.; Vijaykrishnan, N.; Irwin, M. J.; Benini, L.; De Micheli, G. “Analysis of error recovery schemes for networks on chips”. In: IEEE Des. Test, Vol. 22, 2005, pp. 434-442.

[MCG06] McGowen, R.; Poirier, C. A.; Bostak, C.; Ignowski, J.; Millican, M.; Parks, W. H.; Naffziger, S. “Power and temperature control on a 90nm itanium family processor”. In: IEEE J. Solid-State Circuits, Vol. 41, No. 1, 2006, pp. 229-237.

[PAL05] Palma, J; Marcon, C; Moraes, F; Calazans, N; Reis, R; Susin, A; “Mapping Embedded Systems onto NoCs – The Traffic Effect on Dynamic Energy Estimation”. In: Symposium on Integrated Circuits and System Design (SBCCI), 2005, pp. 196-201.

[PAL07] Palma J; Indrusiak, L; Moraes, F; Ortiz, A; Glesner, M; Reis, R. “Inserting Data Encoding Techniques into NoC-Based Systems”. In: Annual Symposium on VLSI (ISVLSI), 2007, pp. 299-304.

[PET12] Petry, C; Wachter, E; Moraes, F; Calazans, N; Castilhos, G.”A Spectrum of MPSoC Models for Design and Verification Spaces Exploration”. In: RSP, 2012, pp. 30-35.

[RAB03] Rabaey, J. M.; Chandrakasan A.; Nikolic, B. “Digital Integrated Circuits a Design Perspective”. Upper Saddle River: Pearson Education, 2003, 761p.

[ROS12a] da Rosa, T. R.; Larrea, V.; Calazans, N.; Moraes, F. G. “Power consumption reduction in MPSoCs through DFS.” In: SBCCI, 2012, pp. 1-6.

[ROS12b] da Rosa, T. R. “Reduction of energy consumption in MPSoCs through a dynamic frequency scaling technique”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2011, 85p.

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

[SYL06] Sylvester, D.; Blaauw, D.; Karl, E. “ElastIC: An adaptive self-healing architecture for unpredictable silicon”. In: IEEE Des. Test, Vol. 23, 2006, pp. 484-490.

[TED09] Tedesco, L.; Clermidy, F.; Moraes, F. G. “A monitoring and adaptive routing mechanism for QoS traffic on mesh NoC architectures”. In: CODES+ISSS, 2009, pp. 109-118.

[TED10] Tedesco, L. “Monitoração e roteamento adaptativo para fluxos QoS em NoCs”. Tese de Doutorado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2010, 132p.

[WAN11] Wang, P. H.; Yang, C. L.; Chen, Y. M.; Cheng, Y. J. “Power gating strategies on GPUs”. In: ACM Trans. Archit. Code Optim., Vol. 8, No. 3, 2011, pp. 13:1-13:25.

[WEI04] Weissel, A.; Bellosa, F. “Dynamic thermal management for distributed systems”. In: TACS, 2004, 11p.

[WEL09] Wells, P. M.; Chakraborty, K.; Sohi, G. S. “Mixed-mode multicore reliability”. In: ASPLOS, 2009, pp. 169-180.

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

[XIL13] Xilinx Chipscope, capturado em *http://www.xilinx.com/tools/cspro.htm*, em junho de 2013.

[YOO10] Yoon, Y. J.; Concer, N.; Petracca, M.; Carloni, L. “Virtual Channels vs. Multiple Physical Networks: A Comparative Analysis”. In: Design Automation Conference (DAC’10), 2010, pp. 162-165.

APÊNDICE A – PUBLICAÇÕES

A Tabela 1 apresenta o conjunto das publicações realizadas desde o início do doutorado, juntamente da respectiva classificação atual (junho/2014) no Qualis. A coluna descrição relaciona a publicação com o texto da presente proposta de tese, quando se aplica, ou o tema principal da publicação.

Tabela 17 - Publicações realizadas desde o início do doutorado.

	Publicação	Descrição
1	GUINDANI, GUILHERME MONTEZ; MORAES, FERNANDO GEHM. Achieving QoS in NoC-based MPSoCs through Dynamic Frequency Scaling In: SoC, 2013, pp.1-6. (Qualis B4)	Publicação relacionada ao Capítulo 7
2	MANDELLI, MARCELO; OST, LUCIANO; CARARA, EVERTON ALCEU; GUINDANI, GUILHERME MONTEZ; ROSA, THIAGO GOUVEA DA; MEDEIROS, GUILHERME; MORAES, FERNANDO GEHM. Energy-Aware Dynamic Task Mapping for NoC-based MPSoCs In: ISCAS, 2011, p. 1676-1679. (Qualis A1)	Publicação relacionada à parte do Capítulo 6
3	ROSA, THIAGO RAUPP DA; GUINDANI, GUILHERME MONTEZ; CARDOSO, DOUGLAS; CALAZANS, NEY; MORAES, FERNANDO GEHM. A Self-adaptable Distributed DFS Scheme for NoC-based MPSoCs In: SBCCI, 2011, p. 203-208. (Qualis B1)	Publicação relacionada ao Capítulo 3
4	OST, LUCIANO; GUINDANI, GUILHERME; INDRUSIAK, LEANDRO; MAATTA, SANNA; MORAES, FERNANDO GEHM. Exploring NoC-Based MPSoC Design Space with Power Estimation Models IEEE Design and Test of Computers, vol. 28, no. 2, pp. 16-28, 2011 (Qualis A2)	Publicação relacionada à parte do Capítulo 6
5	OST, LUCIANO; GUINDANI, GUILHERME MONTEZ; INDRUSIAK, LEANDRO SOARES; REINBRECHT, CEZAR; ROSA, THIAGO RAUPP DA; MORAES, FERNANDO GEHM. A High Abstraction, High Accuracy Power Estimation Model for Networks-on-Chip (BEST PAPER) In: SBCCI, 2009, p. 193-198. (Qualis B1)	Publicação relacionada à parte do Capítulo 6
6	GUINDANI, GUILHERME MONTEZ; REINBRECHT, CEZAR; ROSA, THIAGO RAUPP DA; MORAES, FERNANDO GEHM. Increasing NoC Power Estimation Accuracy through a Rate-Based Model In: NOCS, 2009. p. 89-89. (Qualis B1)	Publicação relacionada à parte do Capítulo 6
7	GUINDANI, GUILHERME MONTEZ; FERLINI, FREDERICO; OLIVEIRA, J.; CALAZANS, NEY; MORAES, FERNANDO GEHM. A 10 Gbps OTN Framer Implementation Targeting FPGA Devices In: Reconfig, 2009, p. 30-35. (Qualis B4)	Publicação relacionada ao projeto X10-GIGA, realizado em parceria com a empresa Datacom