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

Mapeamento Dinâmico de Tarefas em MPSoCs Heterogêneos baseados em NoC

EWERSON LUIZ DE SOUZA CARVALHO

Tese apresentada como requisito parcial para a obtenção de grau de Doutor em Ciência da Computação.

Prof. Dr. Fernando Gehm Moraes
Orientador

Porto Alegre, 10 de janeiro de 2009.

Dados Internacionais de Catalogação na Publicação (CIP)

C331m Carvalho, Ewerson Luiz de Souza
Mapeamento dinâmico de tarefas em MPSoCs heterogêneos baseados em NoC / Ewerson Luiz de Souza Carvalho. – Porto Alegre, 2009.
168 f.

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

1. Mapeamento Dinâmico de Tarefas (Informática). 2. Sistemas Multiprocessados. 3. Heurística (Informática). 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 "Mapeamento Dinâmico de Tarefas em MPSoCS Heterogêneos Baseados em Noc", apresentada por Ewerson Luiz de Souza Carvalho, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 10/03/09 pela Comissão Examinadora:

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

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

Prof. Dr. César Augusto Missio Marcon– — FACIN/PUCRS

Prof. Dr. Flávio Rech Wagner– — UFRGS

Homologada em 02/06/09, conforme Ata No. 009 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS

Campus Central
Av. Ipiranga, 6681 - P32 - sala 507 - CEP: 90619-900
Fone: (51) 3320-3611 - Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Aos meus pais.

"*You can fool some people sometimes, but*
you can't fool all the people all the time."

Robert Nesta Marley (1945 - 1981).

Agradecimentos

Certamente eu não teria concluído o Doutorado sem o auxilio de muitos, mas em primeiro lugar agradeço aos meus pais Edevagildo e Sônia e ao meu irmão Wagner pelo apoio nessa caminhada longa de 4 anos (7 desde o mestrado). VALEU FAMÍLIA! AMO VCS! Agradeço ao meu filho João Marcelo por me mostrar o quanto eu sou limitado cada vez que inventa mais uma de suas maravilhosas estórias (que imaginação fértil!). VALEU JM! TE AMO! Agradeço a mãe dele, a Lilian (Eu sei que vou te amar, por toda a minha vida eu ...) e a seus pais pela ajuda na criação do guri.

Alguém que também tem uma parte muito significativa nessa minha conquista é o meu orientador, Fernando Moraes. Vou resumir essa parte dos agradecimentos, pois não quero ser chamado de puxa-saco. Quem conhece o Moraes, e teve a oportunidade de trabalhar com ele, sabe que se trata de um profissional exemplar. Contudo, quem teve o privilégio como eu de conviver com ele sabe que além do profissional tu podes encontrar um amigo, com um coração enorme, sempre disposto a ajudar quando surgem problemas nas nossas vidas. VALEU MORAES!

Ao pessoal do GAPH o meu agradecimento. Nos últimos 7 anos convivi com muita gente no GAPH, e digo convivi porque não foi só compartilhar um espaço. Muito além foram as conversas, bebidas, jogos, churrascos etc. Eu tentei conhecer um pouco de cada um, porque esse é um dos meus lemas. Coisa boa conhecer gente e trocar experiências! Agradeço ao professor Ney Calazans com sua mania de perfeccionismo, e sempre com sua opinião bem formada sobre qualquer assunto que seja discutido. Agradeço ao Marcon pelos conselhos durante esses anos, e também por não ter me causado nenhuma lesão grave no futebol de cada semana. VALEU GAPH!

Meus amigos. Não consigo imaginar como seria a vida sem eles. Edson Moreno, ou seria Edson Carvalho? Que palavras usar? Esse é o cara! Só para dar uma noção, o defeito dele é ser altruísta. Nunca o vi deixar alguém na mão. 7 anos de amizade, e espero que os primeiros de muitos. VALEU EDSON! TU MORAS NO MEU CORAÇÃO! Querem saber quem é o próximo? Uma pista: Não tem uma vez que eu tenha visto esse cara triste. OST=ALEGRE, VALEU! Morei com vários nesses anos em PoA. Certa vez eu ia me mudar pra não ter que morar mais comigo, mas acho que não ia dar certo. Alguns são Edson, Márcio Garcia e o Julian Pontes. Eu e estes dois últimos compartilhamos opiniões tão parecidas com relação à vida, família, política, religião, futebol, mulheres etc. Já com o Edson as opiniões são meio que divergentes, mas eu sei que é a Silvia (noiva dele) que coloca essas idéias na cabeça dele. VALEU MÁRCIO! VALEU JULIAN!

Meus companheiros nas publicações: Rafael Soares, Leandro Möller e Ismael Grehs. VALEU PELA FORÇA! Foi muito legal trabalhar com vcs. Pena que a nossa antiga área entrou em extinção, a reconfiguração parcial e dinâmica. Ela que chamou minha atenção para a pesquisa, lá nos tempos de graduação quando o Ney foi apresentar uma palestra na semana acadêmica. Eu nem sonhava que era tanto trabalho assim. Quantas plataformas queimadas nessa brincadeira séria?

Agradeço ainda o suporte financeiro advindo do CNPq na forma da bolsa de fomento do Programa Nacional de Microeletrônica (PNM). O suporte operacional proporcionado pelo GAPH/FACIN/PUCRS também foi muito imprescindível. Ali não só encontrei os recursos materiais necessários, mas principalmente encontrei a amizade e o companheirismo de muitos outros estudantes (nem tem como citar nomes) e também dos funcionários (Thiago, Sandra, Zé Carlos).

Nas palavras de Mendes Ribeiro: "**Foi um privilégio ter estado com vocês**".

Resumo

MPSoCs são sistemas multiprocessados integrados na forma de um SoC. Eles são tendência no projeto de circuitos VLSI, pois minimizam a crise de produtividade de projeto, representada pelo descompasso entre a capacidade da tecnologia do silício e a capacidade atual de projeto de SoCs. Cita-se como exemplo de MPSoCs os propostos pela Intel e pela Tilera, compostos respectivamente por 80 e 64 núcleos de processamento. MPSoCs podem empregar NoCs para integrar diversos processadores, memórias, bem como núcleos de hardware específicos. O uso de NoCs deve-se a suas vantagens em relação a barramentos, entre as quais maior escalabilidade e paralelismo na comunicação.

A arquitetura alvo do presente trabalho consiste em um MPSoC heterogêneo, com utilização de NoC como meio interconexão entre os elementos de processamento, suportando a execução de tarefas de hardware via lógica reconfigurável, e a execução de tarefas de software via processadores. Um dos processadores da arquitetura alvo, denominado processador gerente, é responsável por: gerência da ocupação dos recursos do sistema, escalonamento, mapeamento, e configuração de tarefas. O mapeamento de tarefas define a posição de uma dada tarefa no sistema. A maioria dos trabalhos encontrados na literatura propõe técnicas de mapeamento estático, definido em tempo de projeto, no qual todas as tarefas de uma dada aplicação são mapeadas simultaneamente. Este mapeamento estático não é adequado para cenários com carga dinâmica de tarefas. Dado que aplicações executando em um MPSoC podem possuir um número variável de tarefas, e que tal número pode exceder os recursos disponíveis, é necessário realizar o mapeamento de tarefas em tempo de execução, mapeamento este denominado de mapeamento dinâmico.

O presente trabalho investiga o desempenho de heurísticas para mapeamento dinâmico de tarefas, com o objetivo de minimizar congestionamentos em NoCs. As tarefas são mapeadas sob demanda, de acordo com as requisições de comunicação e com a ocupação dos canais da NoC. Os algoritmos implementados aplicam estratégias gulosas, onde as tarefas são mapeadas uma por vez. Para isso, a decisão é baseada na informação local da aplicação, apenas relacionada à tarefa requisitada. O algoritmo utilizado como referência nos experimentos mapeia uma dada tarefa no primeiro recurso livre encontrado. Quatro heurísticas congestion-aware são propostas. Através de experimentos realizados com base na modelagem do sistema no nível RTL, pode-se observar redução de 31% na carga nos canais da NoC, de 15% na latência média, e de até 87% no nível médio de congestionamento. Tais resultados demonstram a eficiência das heurísticas propostas.

Palavras-Chave: Mapeamento dinâmico de tarefas, NoC, SoC, MPSoC.

DYNAMIC TASK MAPPING IN NOC-BASED HETEROGENEOUS MPSOCS

Abstract

MPSoCs are multi-processor systems integrated in a single chip. They are a trend in VLSI circuit design, since they minimize the design productivity crisis represented by the gap between the silicon technology and the actual SoC design capacity. Examples of MPSoCs include those proposed by Intel and Tilera, composed by 80 and 64 processing elements respectively. MPSoCs may employ NoCs to integrate several processors, memories, as well as specific hardware cores. NoCs may be used to replace busses, due to their advantages of higher scalability and communication parallelism.

The target architecture of the present work is a NoC-based heterogeneous MPSoC supporting hardware task execution through embedded reconfigurable logic, together with software tasks executed by programmable processors. One of the processors of the target architecture, named manager processor, is responsible for system resources management, task scheduling, task mapping, and configuration control. Task mapping defines the placement of a new task into the system. Most works in literature propose static mapping techniques defined at design time, where all application tasks are mapped simultaneously. This static mapping is not appropriate for dynamic workloads scenarios. Since applications running in MPSoCs may contain a varying number of tasks, and since their number may exceed the available resources, task mapping at run-time is necessary. Such task mapping method is named dynamic task mapping.

The present work investigates the performance of heuristics for dynamic task mapping, targeting NoC congestion minimization. Tasks are mapped on demand, according to the communication requests and the load in NoC channels. The implemented algorithms employ a greedy approach, where tasks are individually mapped. The mapping decision is based on local information, considering the communication constraints of the requested task. The algorithm used as the reference mapping strategy in experiments maps a task into the first free resource available. Four congestion-aware mapping heuristics are proposed. Through the experiments employing RTL abstraction level modeling, it is possible to observe reductions of 31% in channel load distribution, 15% in average latency, and 87% in congestion level. These results demonstrate the efficiency of proposed heuristics.

Key-Words: Dynamic Task Mapping, NoC, SoC, MPSoC.

Sumário

Lista de Abreviaturas

AMBA	Advanced Microcontroller Bus Architecture
APCG	APplication Characterization Graph
APG	APplication Graph
API	Application Programming Interface
ARM7	Advanced RISC Machine 7
BF	Best Fit
BN	Best Neighbor
CAD	Computer-Aided Design
CAFES	Communication Analysis for Embedded Systems
CDCG	Communication Dependence and Computation Graph
CDCM	Communication Dependence and Computation Model
CDG	Communication Dependence Graph
CDM	Communication Dependence Model
CELL BE	Cell Broadband Engine
CI	Circuito Integrado
CL	Clusterização Linear
CLP	Constraint Logic Programming
CRG	Communication Resource Graph
CTG	Communication Task Graph ou Conditional Task Graph
CWG	Communication Weighted Graph
CWM	Communication Weighted Model
DDR2	Double Data Rate 2
DMA	Direct Memory Access
DMEM	Data MEMory
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EC	East Channels
EDF	Early Deadline First
EIB	Element Interconnect Bus
FACIN	Faculdade de Informática
FF	First Fit ou First Free
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPMAC	Floating-Point Multiply-and-ACumulate

GALS	Globally Asynchronous Locally Synchronous
GAPH	Grupo de Apoio ao Projeto de Hardware
GI	Greedy Incremental
GMPSoC	Grafo de MPSoC
GPP	General-Purpose Processor
ILP	Integer Linear Programming
IMEM	Intruction MEMory
IP	Intellectual Property
ISP	Instruction Set Processor
ITRS	International Technology Roadmap for Semiconductors
JTAG	Joint Test Action Group
LCF	Largest Communication First
MAC	Medium Access Control
MACL	Minimum Average Channel Load
MDE	Multicore Development Environment
MMCL	Minimum Maximum Channel Load
MP	Manager Processor
MP3	MPEG-1/2 Audio Layer 3
MPEG	Moving Picture Experts Group
MPI	Message Passing Interface
MPSoC	Multi-Processor SoC
mSA	Marcon Simulated Annealing
mTS	Marcon Tabu Search
MWD	Multi-Window Display
NC	North Channels
NI	Network Interface
NN	Nearest Neighbor
NoC	Network-on-Chip
PCI	Peripheral Component Interconnect
PDA	Personal Digital Assistant
PE	Processing Element
PL	Path Load
PPE	Power Processor Element
PUCRS	Pontifícia Universidade Católica do Rio Grande do Sul
QAP	Quadratic Assignment Problem
QoS	Quality of Service
R	Roteador
RBERG	Integral de Romberg

RISC	Reduced Instruction Set Computer
RL	Reconfigurable Logic
RTEMS	Real-Time Executive for Multiprocessor Systems
RTL	Register Transfer Level
SA	Simulated Annealing
SC	South Channels
SO	Sistema Operacional
SoC	System-on-Chip
SP	Sample Period
SPE	Synergistic Processing Element
TDM	Time-Division Multiplexing
TGFF	Task Graphs For Free
TLM	Transaction-level modeling
TOPS	Trilhões de Operações Por Segundo
TS	Tabu Search
UDP	Unidade Decodificadora de Pacotes
UET	Unidade de Escalonamento de Tarefas
UMT	Unidade de Mapeamento de Tarefas
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit
VLSI	Very-Large-Scale Integration
VOPD	Video Object Plane Decoder
WC	West Channels
WF	Worst Fit
xy	Algoritmo de roteamento xy

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

1. INTRODUÇÃO

Inventado nos Laboratórios da *Bell Telephone* em dezembro de 1947, o transistor é componente fundamental da microeletrônica. Ao passar do tempo, a evolução da tecnologia submicrônica permite reduzir consideravelmente as dimensões dos transistores. Com isso, a indústria de semicondutores pode produzir circuitos integrados (CIs) mais complexos e de alto desempenho, a um custo relativamente baixo. A ITRS [SIA05] prevê que circuitos integrados fabricados na próxima década terão dezenas de bilhões de transistores, com dimensão em torno de 50*nm* e freqüência de operação acima de 10Ghz. Daí surge um importante desafio relacionado à capacidade de fazer uso eficiente dessa tecnologia.

A dificuldade em manter a produtividade de acordo com a crescente complexidade dos sistemas evidencia a *crise de produtividade de projeto* [HEN03], que representa justamente o espaço entre a tecnologia do silício e a capacidade de projeto de CIs. A Figura 1.1 demonstra o crescimento exponencial do número de transistores por CI desde a introdução dos computadores pessoais, em 1981 [SIA99]. Enquanto a produtividade cresce 21% ao ano, a complexidade apresenta um crescimento quase três vezes superior, de 58%.

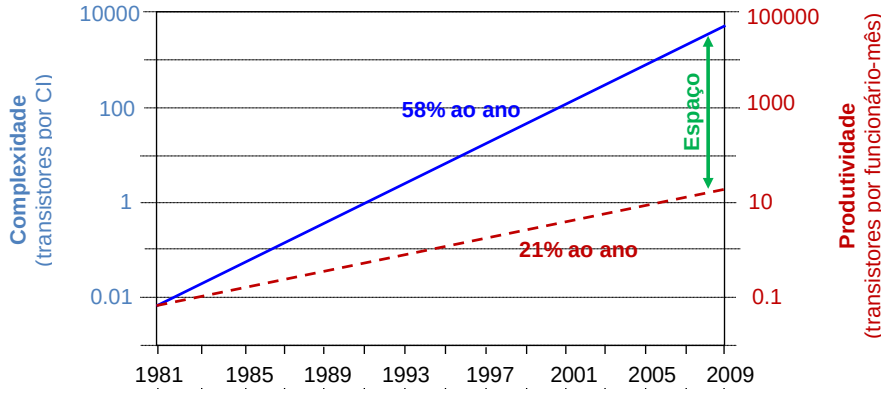


FIGURA 1.1. CRISE DE PRODUTIVIDADE DE PROJETO [SIA99].

Como o fator humano também influi na crise, surge como tendência o emprego de grupos grandes em projetos modulares, onde equipes menores desenvolvem partes do projeto de forma distribuída e paralela, de acordo com o conceito de divisão e conquista.

A crise também demanda o contínuo melhoramento dos métodos de projeto, que devem empregar níveis de abstração cada vez mais altos, bem como uma maior automação, através de ferramentas de CAD mais poderosas.

Uma solução apontada para reduzir da crise de projeto consiste no desenvolvimento de sistemas complexos integrados em um único circuito. Tal estratégia de integração recebe o nome de SoC (do inglês, *System-on-Chip*) [MAR01]. Sua primeira vantagem diz respeito ao desempenho do sistema com relação ao tempo de execução. As comunicações entre os componentes do SoC podem ocorrer mais rapidamente em comparação à estratégia convencional, pois eles se encontram todos no mesmo circuito. A principal vantagem do emprego de SoC é o tempo reduzido de projeto, contribuição considerável para a redução da crise de projeto. Geralmente, o projeto de SoC é baseado no reuso de núcleos de hardware (do inglês, *Intellectual Property Cores* ou IPs). Como os IPs são módulos pré-projetados e pré-validados [BER01], a técnica de reuso pode garantir um tempo menor para o produto chegar ao mercado, fator imprescindível no cenário competitivo atual.

Em geral, o aumento da complexidade das aplicações demanda maior capacidade de processamento. Tal fato impulsiona o desenvolvimento de sistemas computacionais composto por vários processadores, outrora implementados na forma de *clusters* (*i. e.* agregados de computadores [BAK99]), agora implementados na forma de um SoC. No domínio dos *clusters*, obtêm-se ganho de desempenho através da conexão de vários computadores por meio de redes rápidas de comunicação. As aplicações são então paralelizadas para executar nos vários processadores que compõem arquitetura. MPSoCs (do inglês, *Multi-Processor System-on-Chip*) são sistemas multiprocessados implementados na forma de um SoC [JER05]. Em geral, eles são compostos por vários elementos de processamento, memórias e núcleos de hardware específicos. Todos esses componentes são conectados por meio de uma infra-estrutura de comunicação, a qual requer flexibilidade para suportar a conexão de muitos e diversificados elementos de processamento.

Não só o projeto de sistemas complexos, tais como MPSoCs, requer atenção. Adicionalmente, o suporte à operação consiste em um assunto não menos complicado e igualmente relevante. Dois tópicos importantes em MPSoCs dizem respeito à sua infra-estrutura de comunicação intrachip e à gerência dos recursos do sistema. O desempenho da infra-estrutura de comunicação reflete diretamente no desempenho do sistema. Com isso, torna-se imprescindível que tal infra-estrutura suporte taxas consideráveis de comunicações, e alto paralelismo. Além disso, o crescimento do número de elementos processadores nos sistemas exige o emprego de infra-estruturas mais escaláveis. Porém, pode-se afirmar que estratégias tradicionais de comunicação, tais como barramento e ponto-a-ponto não são escaláveis o suficiente para as futuras arquiteturas [KUM02] [ZEF04]

[COR06]. As redes intrachip (do inglês, *Networks-on-Chip* ou simplesmente NoCs) [BEN02] consistem em uma nova abordagem de comunicação, baseada no reuso de conceitos bem conhecidos de redes de computadores no domínio intrachip.

De forma simplificada, uma *aplicação* é um *conjunto de tarefas*. Cada uma destas é responsável por executar parte da funcionalidade da aplicação, e em geral comunicam-se entre si para troca de dados. Em sistemas complexos, as tarefas podem executar no mesmo ou em diferentes elementos de processamento. Adicionalmente, diversas aplicações podem executar em paralelo. Muitas aplicações que executam em MPSoCs (*e. g.* no domínio de multimídia e redes) apresentam uma carga dinâmica de tarefas. Isso implica um número variável de tarefas executando em paralelo, podendo o número necessário exceder os recursos disponíveis no MPSoC. Daí surge a necessidade de controle da operação e dos recursos do sistema, incluindo o gerenciamento dinâmico da carga das tarefas, que corresponde às funções de escalonamento e mapeamento.

O *escalonador de tarefas* é responsável por determinar a *ordem* na qual as tarefas serão carregadas para execução em um dado elemento de processamento do MPSoC. Após decidir qual tarefa, é preciso decidir em qual *posição* ela deve ser carregada. Basicamente, esse é o objetivo do *mapeamento de tarefas*. Essa operação exige o controle total da ocupação dos recursos do sistema (*e. g.* elementos de processamento, memória, canais da NoC, etc), sob o risco de fazer mal uso dos recursos do MPSoC. Dentre os assuntos que envolvem o suporte operacional para um MPSoC, o mapeamento de tarefas desperta interesse, pois pode influenciar no desempenho do sistema, seja no tempo de execução das aplicações, seja na dissipação de potência.

A continuidade desse Capítulo introduz em maiores detalhes a abordagem de rede intrachip como uma alternativa de conexão (Seção 1.1). A seguir, na Seção 1.2 discute-se conceitos básicos de MPSoC, incluindo a motivação para seu estudo. A Seção 1.3 introduz e define de forma sucinta o problema de mapeamento de tarefas. O final do Capítulo dedica-se à apresentação dos objetivos do trabalho (Seção 1.4) e à organização dos Capítulos que compõem a presente Tese (Seção 1.5).

1.1. Redes Intrachip – NoCs

No que diz respeito a sua estrutura, uma rede intrachip é composta basicamente por um conjunto de roteadores e canais de comunicação que interconectam os núcleos de um sistema integrado (ver Figura 1.2(a)). Sua funcionalidade é suportar a comunicação entre tais núcleos, que ocorre através da troca de mensagens geralmente transmitidas na forma de pacotes ao longo da rede [ZEF03]. A flexibilidade da NoC deve permitir a cone-

xão de núcleos de diferentes naturezas, *e. g.* GPPs, memórias, dispositivos de entrada/saída ou ainda IPs específicos. Além disso, cada um deles pode ter características próprias de voltagem, freqüência de operação e/ou tecnologia.

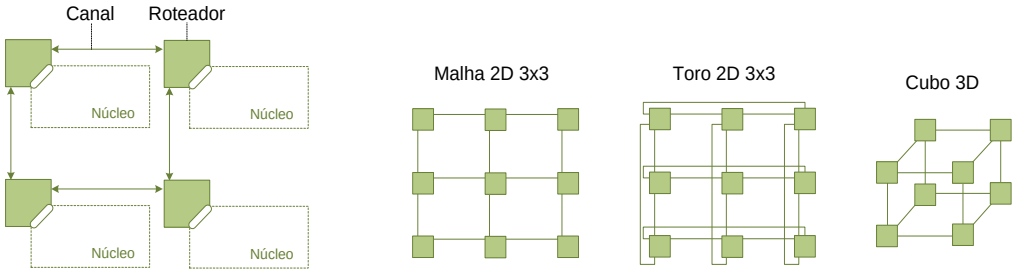


FIGURA 1.2. REDES INTRACHIP: ELEMENTOS BÁSICOS E EXEMPLOS DE TOPOLOGIAS.

Importantes conceitos herdados da área de redes computadores são aplicados no nível intrachip. Uma NoC pode ser caracterizada por sua *topologia* e pelos *mecanismos de comunicação* adotados. A *topologia* corresponde à organização dos seus roteadores de acordo com um grafo, onde os roteadores são representados pelos vértices, e os canais pelas arestas. As topologias mais comuns são malha e toro (ver Figura 1.2(b)), principalmente por serem regulares e planares, portanto mais facilmente implementadas em circuitos 2D.

Os *mecanismos de comunicação* de uma rede definem a forma como os pacotes trafegam através dela [ZEF03]. O *controle de fluxo* lida com a alocação dos recursos (*i. e. buffers* e canais) necessários para um pacote avançar pela rede. Exemplos são controle de fluxo baseado em créditos ou baseado em um protocolo de *handshake*. O algoritmo de *roteamento* define o caminho a ser utilizado por uma mensagem para atingir o seu destino. Alguns exemplos de roteamento são *xy*, *west-first*, *negative-first*, dentre outros. A política de *arbitragem* é responsável por resolver os conflitos internos na rede, quando duas ou mais mensagens competem por um mesmo recurso (e.g. *buffer* ou canal de saída). Ela pode ser baseada, por exemplo, em prioridades. O mecanismo de *chaveamento* define como uma dada mensagem é transferida da entrada de um roteador para um de seus canais de saída (i.e. circuito ou pacote). A *memorização* determina o esquema de filas utilizado para armazenar as mensagens. Como exemplo, pode ser utilizado *buffers* nas entradas, nas saídas ou em ambos os tipos de portas dos roteadores.

Como mencionado anteriormente, o emprego de NoCs é imprescindível frente às limitações impostas pelos barramentos, relativas à baixa escalabilidade e ao pouco paralelismo suportado na comunicação [BEN02]. Adicionalmente, as redes podem suportar o paradigma *Globalmente Assíncrono Localmente Síncrono* ou GALS [CHA84]. O aumento da freqüência de operação dos circuitos faz com que o atraso de propagação dos sinais exceda o período de relógio. O paradigma GALS promete amenizar tal problema, decompon-

do o sistema em um conjunto de núcleos síncronos que interagem de uma maneira assíncrona. Por tudo isso, o emprego de NoC é tendência conforme atesta o grande volume de trabalhos recentemente publicados nessa área [KUM02] [ZEF03] [MOR04] [COR06].

1.2. Sistemas Multiprocessados em um SoC – MPSoCs

MPSoCs são arquiteturas que buscam um compromisso entre restrições da tecnologia VLSI e as necessidades da aplicação. Eles representam tendência no cenário atual das pesquisas [JER05] [FET06] [WOL08]. Grande parte desse interesse deve-se ao fato do MPSoC representar uma evolução do conceito de SoC, beneficiando-se do reuso no projeto. Além disso, MPSoCs apresentam um conceito simples para obter alto desempenho, o uso de várias unidades de processamento em paralelo. Esse conceito foi empregado já em 1976 pelo processador vetorial Cray-1 [CRA77], evoluiu até as primeiras máquinas agregadas (*e. g.* projeto Beowulf [STE95]) no início de 1994, e continua até os processadores multi-núcleo atuais (*e. g. dual* e *quad core*). O gráfico esboçado na Figura 1.3 demonstra o crescimento do número de núcleos nos processadores nas últimas décadas, com destaque para o período posterior ao ano de 2005, quando o crescimento é acentuado e surgem implementações com mais de 16 núcleos.

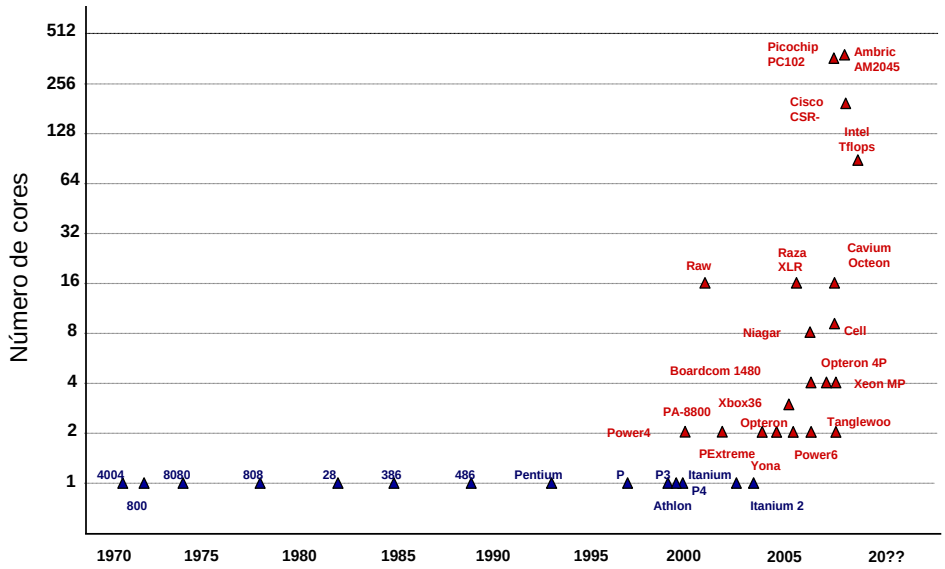


FIGURA 1.3. CRESCIMENTO DO NÚMERO DE NÚCLEOS POR PROCESSADOR AO LONGO DOS ANOS [GOR06].

Sob o ponto de vista do *multiprocessamento*, um MPSoC é dito *homogêneo* quando os elementos processadores que o compõem são todos da mesma natureza. Por exemplo, um sistema composto por processadores idênticos que permitem exclusivamente a execução de tarefas de software compiladas para tal arquitetura de processador. De outra forma, quando o MPSoC possui elementos de processamento diferentes (GPPs, DSPs, etc) ele é dito *heterogêneo*. Nesse caso as tarefas também serão de naturezas distintas.

Enquanto MPSoCs homogêneos tendem a simplificar a aplicação de técnicas como migração de tarefas, MPSoCs *heterogêneos* podem suportar uma variedade maior de aplicações. Para garantir qualidade e desempenho, um decodificador de TV digital, por exemplo, deve ser heterogêneo o suficiente para integrar processadores (*e. g.* RISC), núcleos de hardware dedicados (*e. g. upsampler*) e memórias (*e. g.* SDRAM). Além disso, cada um desses componentes possui diferentes funcionalidades, tamanhos e necessidades de comunicação, o que demonstra a complexidade desses sistemas.

MPSoCs podem ainda empregar lógica reconfigurável. Essa tecnologia emerge no projeto de CIs, pois sugere ao hardware flexibilidade similar ao software, permitindo a carga de tarefas de hardware no sistema até mesmo em tempo de execução. *Reconfiguração dinâmica* [WIR98] é a técnica que permite modificar o comportamento do hardware. Além disso, a implementação de funcionalidades de uma aplicação através de tarefas de hardware pode representar um ganho significativo de desempenho, sem, no entanto comprometer a área do circuito. Essa prática é denominada *hardware virtual* [DEH98].

Atualmente foram produzidos MPSoCs com tamanhos significativos, como os propostos pela Intel [VAN07b] e pela Tilera [TIL07]. O primeiro deles é composto por 80 núcleos de processamento idênticos, enquanto o outro por 64 núcleos também idênticos. No que diz respeito à comunicação, ambos MPSoCs citados empregam NoCs com topologia malha.

Além do proeminente ganho de desempenho, o estudo de MPSoCs também interessa porque renova a pesquisa em áreas clássicas da ciência da computação. Ele remete o pesquisador a tópicos relacionados a sistemas operacionais, processamento paralelo, redes de computadores, microeletrônica, dentre outros. O desenvolvimento e a implementação de um MPSoC é um processo bastante complexo que exige vasto conhecimento em todas essas áreas. O gerenciamento dinâmico dos recursos do MPSoC é crucial para o uso efetivo de todo o poder de processamento disponibilizado por essa tecnologia.

1.3. Mapeamento de Tarefas

Definido de maneira informal, o *mapeamento de tarefas consiste na escolha da melhor posição para uma dada tarefa*. Em sistemas heterogêneos, antes do mapeamento propriamente dito é necessário realizar um passo de *ligação* (do inglês, *binding*), responsável por assegurar a atribuição das tarefas apenas aos elementos de processamento adequados. Por exemplo, em um MPSoC composto por processadores e lógica reconfigurável, as tarefas de hardware só podem ser carregadas na lógica reconfigurável embarcada.

O problema de mapeamento é similar ao problema de atribuição quadrática (do inglês, *quadratic assignment problem* ou QAP) [GAR79]. O QAP é um dos problemas fundamentais de otimização combinatória, sendo um problema NP-completo, ou seja, não existe uma solução para resolvê-lo em tempo polinomial. Dessa forma, para pequenas instâncias do problema, *e. g.* mapeamento de tarefas em uma NoC de dimensões 2x2, o tempo para encontrar a melhor solução usando métodos exaustivos deve ser relativamente baixo, enquanto que para uma NoC 5x5 o tempo pode ser inaceitável. É importante citar aqui que algumas das arquiteturas propostas pela Intel [VAN07b] e pela Tilera [TIL07] possuem redes de dimensões 8x8 e 8x10, respectivamente. Dessa forma, torna-se obrigatório o emprego de heurísticas para resolução do problema de mapeamento. Algumas das principais heurísticas adotadas são baseadas em algoritmos genéticos [WU03], *Simulated Annealing* [ORS07] ou *Tabu Search* [MUR06b].

Com relação ao instante da decisão, o mapeamento é dito *estático* quando definido em tempo de síntese ou projeto, enquanto que o mapeamento *dinâmico* é aquele determinado em tempo de execução, em função de parâmetros tais como carga do sistema, estado de congestionamento na interface de comunicação dos roteadores, por exemplo.

A operação de mapeamento pode empregar diferentes critérios para otimização e diferentes funções custo. No que diz respeito à função custo adotada, pode-se considerar a fragmentação resultante do sistema, a dissipação de potência, e ainda a ocupação dos canais. Em sistemas com restrições de tempo real, o atendimento dos *deadlines* das tarefas ganha importância na tomada da decisão. Para o caso onde as aplicações apresentam necessidades específicas de comunicação, o gerenciamento da ocupação dos canais é crucial para manutenção da qualidade de serviço da rede. Geralmente a qualidade de serviço (QoS) de uma dada rede diz respeito à largura de banda disponibilizada para uma dada aplicação. Já no caso de dispositivos portáteis, o consumo de energia deve representar a principal função.

A maioria dos trabalhos encontrados na literatura propõe soluções para mapeamento estático de tarefas [HU03] [RHE04] [MUR04a] [RUG06] [MEH07], mas recentemente alguns trabalhos começam a investigar também o mapeamento dinâmico de tarefas [KIM05] [NGO06] [WRO06] [CHO07]. Dois trabalhos sobre mapeamento estático evidenciam-se por investigar algoritmos de mapeamento que consideram a energia consumida pelo sistema, técnicas geralmente denominadas *Power* ou *Energy-aware*. São eles os trabalhos de Hu e Marculescu [HU03] e Marcon e outros [MAR05a]. Os Autores destes trabalhos buscam reduzir a energia consumida do sistema relativa à comunicação.

No caso de aplicações de comunicação intensiva, tais como aplicações de fluxo de dados, a manutenção da ocupação da infra-estrutura de comunicação é fundamental. A ocorrência de congestionamentos pode acarretar problemas na transmissão de vídeo e áudio, por exemplo. Para essa classe de aplicações, os métodos *congestion-aware* são empregados. Eles visam uma ocupação inteligente do meio de comunicação, evitando ou ao menos reduzindo os congestionamentos. Para isso, o mapeamento *congestion-aware*, aplicado sobre uma NoC, considera a ocupação de cada um dos canais da rede. Como tal estratégia diminui a ocupação dos canais da rede, pode-se estimar que o tempo total de execução do sistema também será reduzido. Adicionalmente, tal mapeamento deve contribuir também para redução da energia consumida, isso porque assim como em [HU03] e [MAR05a], aqui congestionamentos são evitados através da diminuição do número de saltos que os pacotes percorrem entre sua fonte e destino. Entretanto, essa possível redução de energia é apenas um efeito colateral do mapeamento *congestion-aware*.

A estratégia de *migração* [NOL05a] [BER06] de tarefas também tem sido aplicada em MPSoCs para otimizar o desempenho em tempo de execução. Essa estratégia, muito utilizada no domínio de aplicações paralelas, visa realocar uma dada tarefa ou quando um gargalo de desempenho é identificado, ou para distribuir de maneira mais uniforme a carga de trabalho entre os elementos de processamento do sistema.

Em muitos trabalhos o termo *escalonamento* é substituído pelo termo *mapeamento temporal*. Enquanto isso, o *mapeamento espacial* aqui tratado, recebe o nome de *mapeamento*, *posicionamento* ou *ligação*. Para evitar confusão, no presente trabalho são mantidos os termos *escalonamento* para referenciar o mapeamento temporal, e apenas *mapeamento* quando se tratar de mapeamento espacial de tarefas.

1.4. Originalidade e Objetivos da Tese

A originalidade do trabalho consiste na proposta de uma abordagem de mapeamento que visa preencher a lacuna identificada entre as estratégias estáticas de mapeamento e as pesquisas sobre migração de tarefas. As primeiras geralmente aplicam heurísticas complexas de mapeamento, que, no entanto não se adéquam à natureza dinâmica das aplicações. De outro lado, embora apliquem estratégias dinâmicas, os estudos sobre migração refletem ainda passos iniciais, que em geral focam detalhes como salvamento de contexto e pontos de migração, sem qualquer ênfase nas heurísticas de mapeamento.

Dentre os objetivos estratégicos do presente trabalho encontram-se:

- Dominar a tecnologia de projeto de MPSoCs;
- Revisar o problema de mapeamento dinâmico de tarefas;
- Dominar e avaliar as heurísticas empregadas para resolver esse problema;
- Adequar o problema à natureza de MPSoCs baseados em NoC;
- Propor heurísticas de mapeamento adequadas a estes MPSoCs;
- Avaliar tais heurísticas.

Os objetivos específicos desse trabalho são:

- Propor um MPSoC factível de implementação;
- Definir sua organização (*i. e.* seus componentes básicos);
- Definir sua arquitetura (*i. e.* interface com as aplicações);
- Modelar seu comportamento para verificar sistemas maiores;
- Propor e avaliar um mapeamento unificado para tarefas de naturezas distintas.

O presente trabalho investiga o desempenho de seis algoritmos de mapeamento em MPSoCs baseados em NoC, considerando cenários com carga dinâmica de tarefas. O principal objetivo consiste em minimizar os congestionamentos na rede, através da otimização da ocupação dos seus canais. Os resultados são obtidos a partir de simulações do sistema composto por uma NoC descrita em VHDL, onde os recursos (*i. e.* elementos de processamento) são simulados através de *threads SystemC*. A avaliação de cada um dos algoritmos implementados considera as seguintes métricas: (*i*) tempo de execução total do sistema; (*ii*) ocupação dos canais da NoC; (*iii*) nível de congestionamento na rede; (*iv*) latência dos pacotes transmitidos; e (*v*) tempo de mapeamento.

1.5. Organização da Tese

O presente documento encontra-se organizado em sete capítulos. Neste primeiro apresentou-se a introdução ao assunto incluindo os conceitos de NoC, MPSoC e o problema de mapeamento a ser atacado. Além disso, os objetivos e a motivação do trabalho proposto foram discutidos.

No *Capítulo 2*, primeiramente apresenta-se um resumo do estado da arte em organizações de MPSoCs propostas, em ambos os domínios acadêmico e industrial. Na seqüência, apresenta-se um resumo dos trabalhos relacionados ao mapeamento de tarefas, estático e dinâmico. Além disso, trabalhos sobre a técnica de migração de tarefas também são abordados. Ao final de cada Seção, um quadro comparativo é apresentado.

O *Capítulo 3* apresenta a proposta de uma organização de MPSoC heterogêneo que suporta tanto a execução de tarefas no domínio de software, quando no domínio de hardware, de acordo com o conceito de reconfiguração parcial, acima exposto.

No *Capítulo 4*, apresenta-se a principal contribuição do presente trabalho. Ele propõe um conjunto de seis algoritmos para mapeamento dinâmico de tarefas, incluindo dois algoritmos de referência, sem avaliação de quaisquer funções custo, e outras quatro heurísticas ditas *congestion-aware*. Heurísticas *congestion-aware* avaliam a ocupação dos canais da rede na sua tomada de decisão.

Os *Capítulos 5* e *6* dedicam-se aos experimentos realizados. No *Capítulo 5*, cada um dos algoritmos de mapeamento implementado é avaliado com relação ao desempenho da NoC. As simulações realizadas são baseadas em uma plataforma descrita em nível RTL e simulada utilizando *ModelSim 6.4* da *Mentor Graphics*. Ela é composta pela NoC descrita em VHDL e elementos de processamento descritos em *SystemC*. Três cenários de simulações são considerados. Cada um deles visa a investigação de aplicações sintéticas com características de conectividade de tarefas e fluxo de dados diferentes.

O *Capítulo 6* dá continuação aos experimentos. Ele apresenta a avaliação da estratégia gulosa proposta para resolução do problema de mapeamento de tarefas. Para isso, as heurísticas aqui propostas são comparadas a dois algoritmos propostos por Marcon [MAR07]. Dois cenários de simulação são considerados, cada um representando sistemas com tamanhos diferentes. Dentre as aplicações mapeadas constam: decodificador MPEG-4, MWD, *Integral de Romberg*, e decodificar *VOP*.

O *Capítulo 7* primeiramente apresenta uma listagem das contribuições do presente trabalho. Em seguida, discute conclusões obtidas, e apresenta algumas sugestões para trabalhos futuros na área de mapeamento dinâmico de tarefas.

Ao final do documento, o *Apêndice A* apresenta uma descrição mais detalhada dos gráficos de aplicações empregados nos cenários dos experimentos, bem como os mapeamentos obtidos através da ferramenta CAFES, apresentada em [MAR07].

2. TRABALHOS RELACIONADOS

Cada vez mais, observa-se como tendência a pesquisa na área de MPSoCs [JER05] [FET06] [WOS07] [SAI07a]. Um dos catalisadores desse fenômeno refere-se às aplicações que demandam maior poder computacional. Além disso, o compromisso entre o custo de projeto de um MPSoC e seu potencial ganho de desempenho é outro atrativo. O projeto pode beneficiar-se das técnicas de reuso, e a tecnologia do silício suporta a concepção de circuitos cada vez mais complexos. O conceito é simples, a área de silício disponível é preenchida com módulos replicados. Contudo, a realização demanda grande esforço da comunidade científica na busca de melhores métodos de projeto e infra-estruturas para suporte operacional mais eficaz.

O presente Capítulo apresenta propostas de organizações para MPSoCs (Seção 2.1) encontradas na literatura. Na seqüência, a Seção 2.2 apresenta uma revisão dos trabalhos sobre mapeamento, abrangendo ambas as abordagens, estática (Subseção 2.2.1) e dinâmica (Subseção 2.2.2). Ao final de cada uma das Seções, apresenta-se considerações sobre os trabalhos revisados, incluindo tabelas que facilitam a comparação entre os mesmos. A migração de tarefas também está inclusa na discussão (Seção 2.3).

2.1. Organizações de MPSoCs

Como será apresentado abaixo, existem na literatura tanto propostas acadêmicas de organizações de MPSoCs (Subseção 2.1.1), quanto produtos baseados nesta tecnologia (Subseção 2.1.2), dentre eles [DUL05] [KIS06] [HAL06] [VAN07b] [TIL07].

2.1.1. Propostas de MPSoCs Acadêmicos

Lin e outros [LIN05] apresentam um MPSoC homogêneo, cuja infra-estrutura de comunicação consiste em uma NoC malha. O MPSoC proposto é composto por processadores e roteadores. Cada processador possui sua própria memória local, e está conectado a um roteador da NoC. Os roteadores empregam arbitragem *round-robin*, chaveamento

de circuitos e usam a técnica de canais virtuais. Nos experimentos realizados, foi empregada uma NoC com dimensões 4x4. Os elementos de processamento foram substituídos por geradores aleatórios de tráfego. O trabalho de Lin está focado na investigação de estratégias para mapeamento estático de tarefas, conforme discutido adiante, na Subseção 2.2.1 que apresenta trabalhos sobre mapeamento realizado tem tempo de projeto.

Woszezenki [WOS07] propõe um MPSoC homogêneo, e um conjunto de ferramentas que permitem a geração do sistema, sua simulação e avaliação de resultados. O MPSoC proposto é composto por um conjunto de processadores, conectados através da NoC Hermes [MOR04]. Os processadores do MPSoC proposto são baseados em uma versão modificada do processador PLASMA [RHO01]. Cada um deles possui memória local de 64 Kbytes, dividida em 4 páginas: uma delas dedicada ao *µkernel* desenvolvido e as demais para tarefas (*i. e.* processadores multitarefa). Um dos processadores do sistema (*i. e.* Plasma MP) é dedicado a realizar operações de controle (Figura 2.1). Uma unidade DMA é usada para acelerar a alocação das tarefas. Este trabalho fornece suporte à alocação dinâmica de tarefas, entretanto nenhuma heurística é investigada.

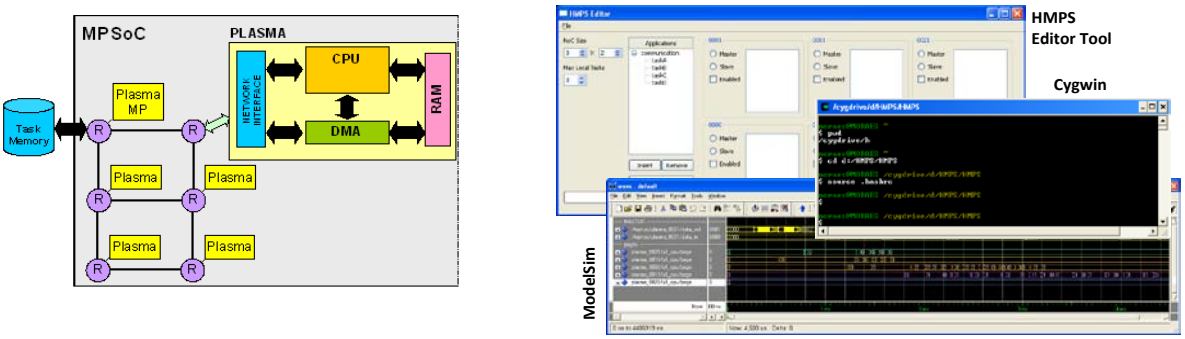


FIGURA 2.1. MPSOC HOMOGÊNEO PROPOSTO POR WOSZEZENKI [WOS07].

Saint-Jean e outros [SAI07a] [SAI07b] empregam um MPSoC homogêneo, que permite o emprego de técnicas para balanceamento de carga. O MPSoC proposto possui processadores conectados via NoC (Figura 2.2). Alguns dos processadores do sistema também estão conectados a um barramento, através do qual se comunicam com os demais componentes do MPSoC (*e. g.* computador hospedeiro, periféricos, etc). Cada processador executa um sistema operacional que permite processamento multitarefa. A NoC empregada é derivada da Hermes [MOR04]. Em [SAI07a], um MPSoC 2x3 é implementado em uma única plataforma de prototipação, ao passo que em [SAI07b] e [SAI08] são usadas 9 e 16 plataformas, respectivamente, cada uma contendo um par roteador mais processador.

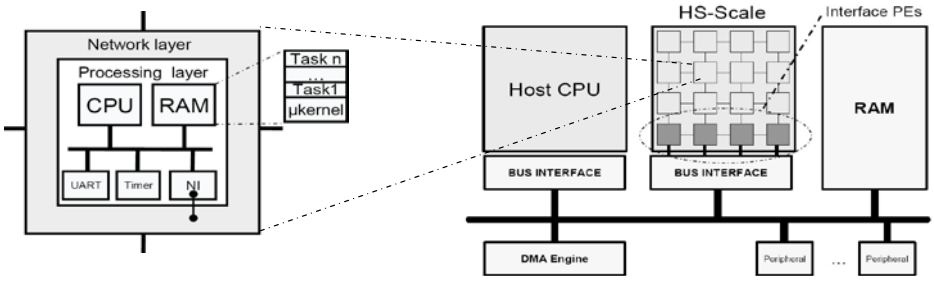


FIGURA 2.2. MPSOC HOMOGÊNEO PROPOSTO POR SAINT-JEAN E OUTROS [SAI07A].

2.1.2. Produtos Baseados em MPSoCs

O PC102 é um MPSoC comercial proposto para aplicação em redes sem fio. Ele é baseado na arquitetura *picoArray* [DUL05] (Figura 2.3), onde os processadores são interconectados através de barramentos *picoBus* (32bits) e chaves programáveis. O PC102 possui ao todo 322 processadores, organizados na forma de uma matriz. Cada processador executa um único processo, e possui suas próprias memórias de dados e instruções. A infra-estrutura de comunicação adotada possui largura de banda interna de 3.3Tbits/s, operando a uma freqüência de 160MHz. As comunicações se dão através de sinais de sincronização e dados transmitidos de acordo com um protocolo TDM.

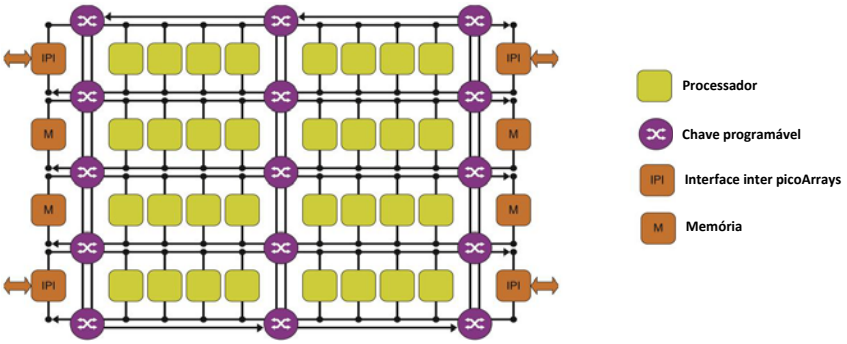


FIGURA 2.3. ORGANIZAÇÃO DE UM *PICOARRAY*, PROPOSTA PELA PICOCHIP [DUL05].

Em [KIS06], a IBM apresenta o MPSoC heterogêneo CELL, que visa a execução de diversas aplicações, incluindo processamento científico e multimídia. Seu primeiro alvo foi o videogame PlayStation3. Conforme a Figura 2.4, o CELL consiste de um processador de 64bits (PPE), oito processadores aceleradores (SPEs), um controlador de memória, um barramento de interconexão, memória e interfaces de E/S, tudo integrado na forma de um SoC. Operando a 3.3GHz, seu pico de desempenho teórico é 204.8Gflop/s para precisão simples e 14.6Gflop/s para precisão dupla. O PPE executa o sistema operacional que coordena todo o sistema. A interface de conexão (EIB) possui 4 anéis para transmissão de dados, e uma rede com topologia estrela que transporta instruções. O CELL conta com um ambiente *open source* para o desenvolvimento de aplicações [GSC07].

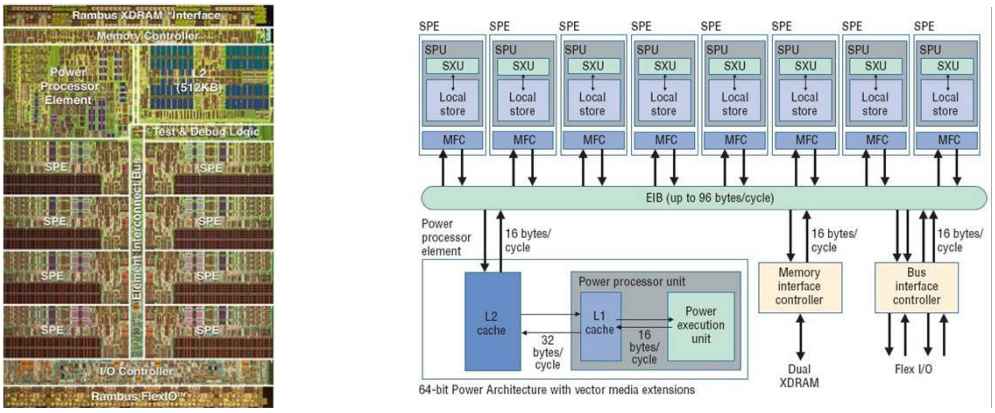
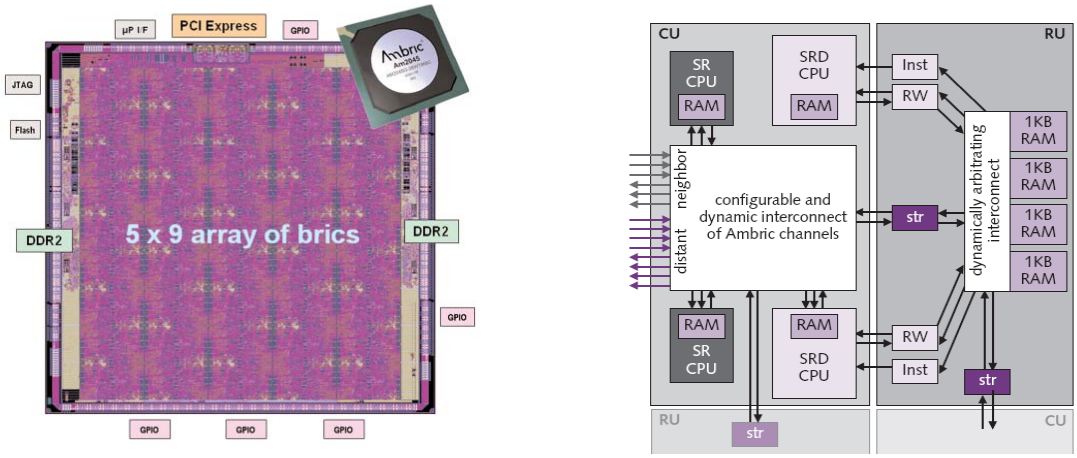


FIGURA 2.4. MPSOC CELL DESENVOLVIDO POR IBM, SONY E TOSHIBA PARA O PLAYSTATION3 [KIS06].

O MPSoC Am2045 [HAL06] é composto por processadores RISC de 32bits. Seu projeto visa substituir processadores embarcados, DSPs e FPGAs em aplicações que exigem alto desempenho no processamento de sinais digitais (*e. g.* codificação e decodificação H.264). O Am2045 consiste em uma matriz de *brics* 5x9 (Figura 2.5). Cada um deles possui oito núcleos processadores e 8KB de memória local. Ao todo são 360 processadores e 585KB de memória. Além disso, estão disponíveis controladores de memória DDR2, controlador PCI, 128 portas de E/S, interface serial para memória *flash*, e interface JTAG para depuração. O CI integra 117 milhões de transistores, fabricado com tecnologia 0.13µm. O Am2045 possui desempenho teórico máximo de 1TOPS a 333MHz. A programação do sistema é descrita em Java, entretanto, o código é traduzido para a linguagem proprietária pelo ambiente de desenvolvimento disponível para o Am2045.



SR – Processador RISC
SRD – Processador RISC com DSP

Cluster com 4 núcleos processadores, 4 memórias locais, suas interconexões e estruturas de controle.

FIGURA 2.5. MPSOC AM2045 COMPOSTO POR 360 PROCESSADORES RISC, DESENVOLVIDO PELA AMBRIC [HAL06].

A Intel apresenta um MPSoC com 80 elementos de processamento conectados via NoC [VAN07b] [VAN08]. A NoC apresenta topologia malha, e interfaces assíncronas. O sistema opera com freqüência de 4 GHz. Cada PE possui duas unidades independentes de

ponto flutuante de precisão simples (FPMAC), memória de instruções (IMEM), e memória de dados (DMEM). O roteador na NoC apresenta chaveamento *wormhole* e dois canais físicos para transmissão de pacotes. O MPSoC alcança o pico de desempenho de 1.0Tflops à 1V e 1.28Tflops à 1.2V. O consumo de potência estimado é de 98W à 1V e 181W à 1.2V. Todo MPSoC possui em torno de 100 milhões de transistores.

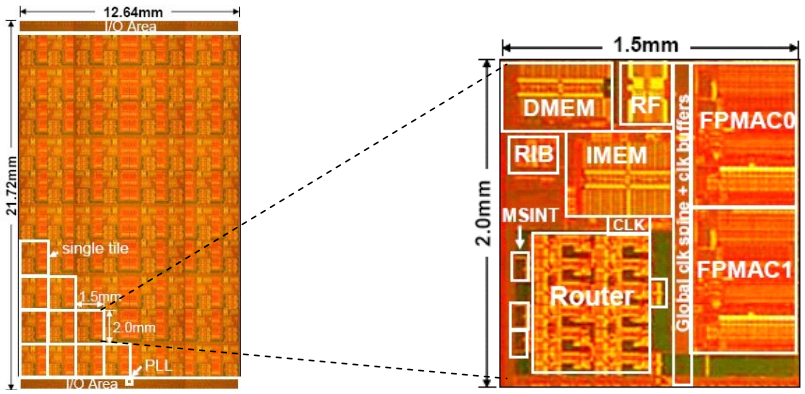


FIGURA 2.6. MPSOC COMPOSTO POR 80 PROCESSADORES IDÊNTICOS, DESENVOLVIDO PELA INTEL [VAN07B].

A Tilera apresenta o Tile64 [TIL07], um MPSoC composto por 64 PEs idênticos, conectados via NoC malha *iMESH* [WEN07] (Figura 2.7). Seu propósito inclui aplicações de rede, vídeo digital e telecomunicação. Além do processador, cada núcleo possui memórias *cache* L1 e L2. O Tile64 também possui quatro controladores DDR2; interfaces MAC e PCI. Cada PE pode executar seu próprio SO Linux. PEs operam em uma freqüência entre 500MHz e 866MHz com consumo de potência entre 15 e 22W a 700MHz. Para economizar energia, um dado PE pode entrar no modo *sleep* quando estiver ocioso. A programação de aplicativos para o Tile64 pode ser realizada através do *Multicore Development Environment* (MDE), um ambiente que além da descrição de código em C, permite ainda simulação, análise de desempenho e depuração.

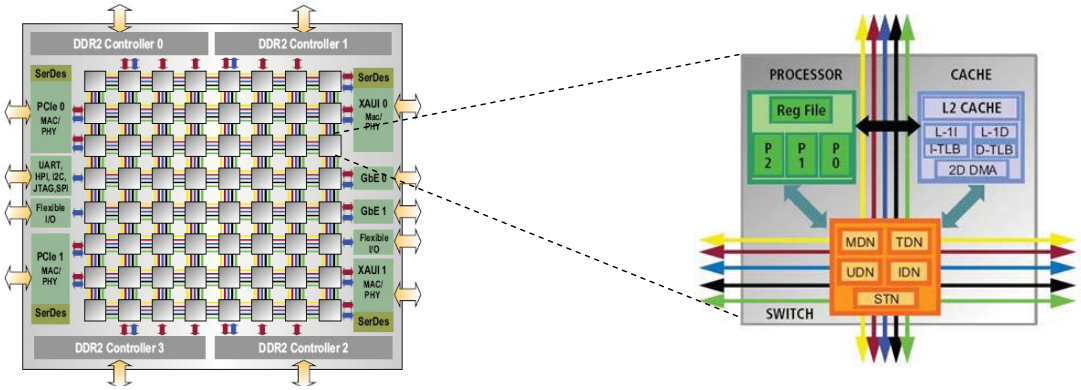


FIGURA 2.7. MPSOC COMPOSTO POR 64 PROCESSADORES IDÊNTICOS, DESENVOLVIDO PELA TILERA [TIL07].

CONSIDERAÇÕES SOBRE OS MPSOCS INVESTIGADOS

Em geral, as propostas tanto acadêmicas quanto comerciais são baseadas em MPSoCs homogêneos. Não existe um padrão para organizações para MPSoC, logo cada pesquisa acaba especificando a arquitetura que melhor lhe convém. Com relação à infraestrutura de comunicação, a maioria dos trabalhos investigados adota NoCs com topologias mais simples (*e. g.* malha) e algoritmos determinísticos de roteamento. O CELL apresenta-se como exceção visto que adota uma arquitetura de conexão hibrida composta por uma NoC de topologia anel para transmissão de dados, e outra com topologia estrela para instruções. Além disso, o CELL apresenta-se na forma de um MPSoC heterogêneo.

A Tabela 2.1 abaixo apresenta um resumo comparativo entre as principais características dos MPSoCs comerciais revisados na presente Seção.

TABELA 2.1. COMPARATIVO DOS MPSOCS INVESTIGADOS.

Nome	*Fabricante*	*Multiprocessamento*	*Infra-estrutura de comunicação*	*Número de PEs*	*Elemento de Processamento*	*Freqüência de Operação*	*Desempenho*
PC102 (2005)	*picoChip*	Homogêneo	Híbrida (barramentos picoBus conectados por chaves)	322	RISC (16bits) Monotarefa	150 MHz	-
CELL (2006)	*IBM*	Heterogêneo	Híbrida (NoC Anel + NoC estrela)	9	1 de 64bits + 8 aceleradores	3.3 GHz	204 GFlops/s
Am2045 (2006)	*Ambric*	Homogêneo	NoC malha	360	RISC 32bits	333 MHz	1 Trilhão de operações por segundo (TOPS)
- (2007)	*Intel*	Homogêneo	NoC malha	80	2 unidades de ponto Flutuante	4 GHz	1.28 TFlops/s
Tile64 (2007)	*Tilera*	Homogêneo	NoC malha (Tilera's iMesh)	64	3-way VLIW MIPS	866 MHz	443 bilhões de operações por segundo (BOPS)

2.2. Mapeamento de Tarefas

Geralmente, o *mapeamento de tarefas* é realizado em duas etapas. A primeira delas tem como principal função encontrar todas as *possíveis* posições onde uma dada tarefa pode ser inserida. Esta operação invariavelmente se vale da informação sobre a ocupação atual do sistema. Portanto, nota-se a necessidade de manter armazenada informação sobre quais recursos estão vazios e quais estão ocupados. Tal informação deve ser constantemente atualizada, seja quando uma tarefa deixa o sistema, ou porque uma nova tarefa foi mapeada. Na segunda fase do processo de mapeamento, a função principal é encontrar dentre as possíveis posições obtidas na fase anterior, qual delas é a *melhor*. A tomada dessa decisão pode considerar diferentes critérios inclusive valendo-se do conceito de qualidade de serviço (QoS). Além da carga dos processadores, pode-se também considerar o meio de comunicação e a energia consumida.

2.2.1. Mapeamento Estático de Tarefas

Conforme apresentado no primeiro Capítulo, de acordo com o instante no qual a decisão é tomada, o mapeamento pode ser classificado como *estático* ou *dinâmico*. A presente Seção revisa os trabalhos relacionados ao mapeamento estático de tarefas, ou seja, aquele definido ainda em tempo de projeto, com base em estimativas do comportamento das aplicações que devem executar na plataforma alvo.

Mihal e Keutzer [MIH03] propõem a modelagem de aplicações concorrentes em MPSoCs heterogêneos. Segundo os Autores, a solução desse problema relaciona-se principalmente à modelagem da concorrência da aplicação que implica na captura formal da comunicação em modelos computacionais. A abordagem proposta parte de uma aplicação descrita em alto nível, a qual é particionada manualmente em tarefas. O particionamento associa as tarefas da aplicação a PEs (*i. e. ligação*). Em etapas posteriores, realiza-se o mapeamento dos PEs na arquitetura alvo. O trabalho não investiga heurísticas para mapeamento, mas evidencia a importância de uma arquitetura que provê comunicação em camadas, e acaba por sugerir o uso de NoCs.

Wu e outros [WU03] apresentam um algoritmo genético para solucionar o problema de mapeamento de tarefas. Sua implementação é otimizada para explorar técnicas de variação dinâmica de voltagem, visando reduzir o consumo de energia do sistema. O modelo de aplicação adotado baseia-se em um grafo de tarefas denominado CTG. Os Autores obtêm uma redução de até 51% no consumo de energia quando aplicam variação dinâmica de voltagem juntamente ao mapeamento.

Lei e Kumar [LEI03a] [LEI03b] também apresentam um algoritmo genético para mapeamento de aplicações. Uma aplicação é descrita na forma de um grafo de tarefas, gerado com o auxilio da ferramenta TGFF [DIC98]. A arquitetura é representada por uma NoC malha, a qual conecta um conjunto heterogêneo de PEs. O objetivo do algoritmo proposto consiste em minimizar o tempo de execução da aplicação. Para isso, modelos de atraso de comunicação são empregados na estimativa do tempo de execução. Os resultados comparam apenas o desempenho do algoritmo proposto de acordo com diferentes parâmetros de entrada, dentre eles: a aplicação, dimensões da NoC, tamanho da população e o número de gerações do algoritmo.

Rhee e outros [RHE04] investigam o mapeamento de tarefas com o objetivo de reduzir o número de saltos dos pacotes na NoC e a ocupação dos canais. Uma aplicação é modelada como um grafo que representa a comunicação entre as tarefas através do volume de dados a serem transmitidos, e da largura de banda necessária. A arquitetura é re-

presentada por uma NoC malha com roteamento xy *wormhole*. Os PEs são idênticos. A principal contribuição do trabalho consiste em uma abordagem de mapeamento onde os roteadores da NoC podem conter vários núcleos/tarefas. Segundo os Autores, ela permite até 81,2% de redução de energia consumida, em comparação à estratégia convencional, a qual permite o mapeamento de apenas um núcleo por roteador.

Hu e Marculescu [HU03] [HU05] investigam o mapeamento das tarefas em sistemas baseados em NoC. A aplicação é modelada por um grafo denominado APCG. Esta abordagem é baseada em modelos de comunicação com pesos, onde o peso do canal de comunicação corresponde à quantidade de bits transmitidos através dele. Os Autores demonstram a possibilidade de redução de 60% no consumo de energia, comparado a soluções *ad hoc*. Em [HU04], estende-se a abordagem anterior, considerando como restrição a largura de banda máxima de cada canal da NoC para o escalonamento de tarefas. Os Autores introduzem um modelo que captura a comunicação (*i. e.* volume em bits) e a computação, baseado em um grafo de comunicação de tarefas CTG, o qual contém o tempo de execução de cada tarefa, seu consumo de energia, e *deadline*. Nos experimentos o gerador aleatório de grafos TGFF é empregado. Comparados ao escalonamento *Early Deadline First* (EDF), o algoritmo proposto apresenta um consumo de energia 44% melhor.

Murali e De Micheli [MUR04a] realizam o mapeamento de tarefas em NoCs de acordo com a largura de banda de comunicação. A aplicação é representada por um grafo de núcleos. A ferramenta NMAP proposta permite selecionar a topologia da NoC (malha ou *torus*), e a estratégia de roteamento (*caminho mínimo único* ou *múltiplos caminhos com divisão de tráfego*). Em [MUR04b], a ferramenta SUNMAP proposta possui uma biblioteca maior de topologias, mas o tipo de roteamento é fixado para *caminho mínimo*. A função custo considera a média de atraso de comunicação, e os consumos de área e energia. Em [MUR05], adiciona-se à ferramenta a capacidade de realizar posicionamento físico. Através do método *Tabu Search* (TS), o mapeamento visa otimizar o uso da largura de banda. Comparado a solução *ad hoc*, foram obtidas soluções 5 vezes melhores para largura de banda. Em [MUR06a], o mapeamento é realizado para um conjunto de aplicações alvo. A solução escolhida é aquela que melhor apresenta-se para todo conjunto. Conforme [MUR06b], essa abordagem resulta em NoCs muito grandes, além de funcionar apenas para aplicações com padrões de tráfego similares. Em [MUR06b], mantém-se uma estrutura para cada aplicação alvo, e avalia-se independentemente o mapeamento.

Manolache e outros [MAN05] investigam o mapeamento de tarefas em uma NoC com garantia de latência. A arquitetura é modelada através de uma matriz 2D de PEs homogêneos. Uma tarefa é representada por sua periodicidade, seu tempo de execução e *deadline*. Uma dada aplicação é descrita por um conjunto de tarefas, através de um grafo. Os

experimentos são baseados em uma NoC com dimensões 4x4. A melhor solução é encontrada através da técnica *Tabu Search*. A garantia de latência é atingida porque tanto a posição das tarefas quando o roteamento dos pacotes são definidos em tempo de projeto. Além disso, varias cópias de cada mensagem são enviadas. Trata-se de uma solução limitada. A duplicação de mensagens permite tolerância a falhas, no caso de perda de mensagens. Entretanto, essa estratégia pode sobrecarregar a rede, já que duplica o tráfego.

Lin e outros [LIN05] tratam o mapeamento de tarefas em um MPSoC homogêneo, baseado em uma NoC malha. A aplicação é representada por um grafo de tarefas, com informação sobre o volume de comunicação. O algoritmo proposto é composto de 4 fases: (*i*) um mapeamento é realizado de forma que as tarefas fiquem o mais perto possível; (*ii*) conexões são estabelecidas (*i. e.* chaveamento de circuito); (*iii*) através de simulação obtém-se um perfil da operação do sistema; e (*iv*) calcula-se a contenção da rede e o sistema é realimentado. Tal procedimento é repetido até que o resultado apresente o desempenho desejado. O mapeamento utiliza o algoritmo *Simulated Annealing (SA)*. Sua função custo considera a distância entre as tarefas, a ocupação dos canais e o volume de comunicação. Os experimentos são baseados em grafos de aplicações gerados no TGFF. Os resultados apresentam 20% de redução no tempo total de execução da aplicação, comparada a alternativa que não considera a comunicação e os efeitos de contenção.

Srinivasan e Chatha [SRI05], além da largura de banda, consideram também a latência resultante como restrição no mapeamento. Uma dada aplicação é modelada por um grafo composto por vértices que representam núcleos ou memórias; e por arestas que indicam a largura de banda e latência necessárias para cada comunicação. A NoC possui topologia malha, e cada roteador possui atributos tais como: larguras máximas de banda suportadas, energia consumida em cada porta. A solução implementada baseia-se no algoritmo Árvore de Corte (do inglês, *Cutting Tree*), que recursivamente divide o grafo da aplicação horizontalmente e verticalmente, até que a partição gere um vértice para cada roteador da NoC. Alguns dos grafos usados nos experimentos baseiam-se em aplicações reais (*e. g.* MPEG, MP3, H.263, VOPD, MWD). Os resultados obtidos demonstram que o algoritmo proposto possui complexidade inferior ao proposto em [MUR04a]. Entretanto, os resultados são equivalentes quando a latência não é tomada como restrição.

Marcon e outros [MAR05a] investigam o mapeamento de tarefas em uma NoC malha. A proposta baseia-se no modelo CDM que representa a dependência entre os pacotes. Esse trabalho manipula três grafos. O grafo CWG é composto por um conjunto de tarefas e as respectivas comunicações (volume de dados). O grafo CDG contém informação sobre todos os pacotes transmitidos e sua ordenação. O grafo CRG mantém informação sobre a topologia da NoC. Comparado à [HU03], o algoritmo proposto apresenta re-

dução de 21% na energia consumida, e de 42% no tempo de execução da aplicação. Em [MAR05b], a modelagem evolui com a proposta do modelo CDCM, que captura o tempo de computação de cada tarefa, além do volume de comunicação. O grafo CDCG é composto por um conjunto de pacotes transmitidos. A descrição de cada um deles contém informação sobre seu instante de envio. Comparado à [HU03], o algoritmo proposto apresentou redução de 20% na energia consumida e de 40% no tempo de execução da aplicação. Em [MAR07] [MAR08], apresenta-se uma avaliação mais completa de heurísticas para mapeamento. Nos trabalhos anteriores, os Autores empregaram a técnica *SA* e o método exaustivo. Agora, o método *Tabu Search* e dois métodos gulosos são também avaliados. O algoritmo *Greedy Incremental* (GI) apresentou bons resultados, enquanto o algoritmo *Largest Communication First* (LCF) combinado com estratégias estocásticas apresentou o melhor compromisso entre tempo de execução e economia de energia.

Orsila e outros [ORS05] modelam uma dada aplicação através de um grafo, com informação sobre a dependência entre as tarefas, seu tempo de computação e o volume de comunicação. Sua arquitetura reflete um sistema homogêneo. Os Autores propõem um algoritmo híbrido em contraproposta ao método *SA* puro. Eles combinam *SA* para otimizações globais e *Migração de Grupo* para otimizações locais. Seu sistema é simulado em um *cluster*. A proposta resulta em um ganho de 8,3% no tempo de execução. Em [ORS06], os Autores propõem um método automático de seleção de parâmetros para o algoritmo *SA*, visando diminuir o esforço de otimização. Em [ORS07], apresenta-se uma junção dos dois trabalhos anteriores, adicionando a avaliação da memória consumida pelos algoritmos. Os resultados são comparados com algoritmos *SA*, *Migração de Grupo* e mapeamento aleatório. *SA* apresenta-se como melhor alternativa.

Ruggiero e outros [RUG06] empregam um MPSoC homogêneo composto por processadores ARM7, interconectados via barramento AMBA. Cada processador tem uma memória local e pode acessar memórias remotas. O sistema operacional de tempo real utilizado é o RTEMS. As aplicações são modeladas como um grafo de tarefas, executadas na forma de um *pipeline*. O mapeamento de tarefas é dividido em um processo de duas etapas, incluindo mapeamento e escalonamento. O mapeamento emprega o modelo de programação linear (ILP), enquanto o escalonamento emprega o modelo de programação por restrições (CLP). O objetivo do mapeamento é reduzir o tráfego no barramento. Sua abordagem demonstra maior eficiência computacional comparada às abordagens puramente ILP ou puramente CLP.

Saeidi, Mehan e outros [SAE07] [MEH07] propõem a ferramenta SMAP, baseada no ambiente *Matlab*, para o mapeamento de tarefas em uma NoC malha. As funções custo adotadas consideram a distância em *hop*s entre tarefas comunicantes, e a ocupação dos

canais da NoC. As aplicações são modeladas através de um grafo de tarefas, o qual informa a dependência entre as tarefas e a largura de banda necessária para as comunicações. O algoritmo de mapeamento *Spiral* proposto mapeia as tarefas na forma de um espiral. Nos experimentos são utilizadas tarefas sintéticas geradas aleatoriamente. As NoCs usadas possuem algoritmo de roteamento xy, e dimensões que variam entre 3x3 e 6x6. Comparado a um algoritmo genético [SHI04] e outro aleatório, o *Spiral* apresenta respectivamente 8 e 10% de ganho na energia consumida na comunicação.

CONSIDERAÇÕES SOBRE MAPEAMENTO ESTÁTICO

Uma parte dos trabalhos revisados aborda sistemas homogêneos onde todas as tarefas são da mesma natureza. Alguns Autores apresentam soluções para mapeamento genérico de núcleos (ou tarefa de hardware) no sistema [MUR06b] [HU05] [MAR07] [SRI05], enquanto outros mapeiam tarefas de software em MPSoCs [LIN05] [RUG06]. Existem também pesquisas que visam o mapeamento em sistemas heterogêneos [LEI03a] [LEI03b]. Dentre as características empregadas nas funções custo, pode-se listar: fragmentação, largura de banda de comunicação, energia consumida na comunicação, proximidade entre as tarefas comunicantes. Isso denota a diversidade de trabalhos na área.

Na Tabela 2.2 (na próxima página) apresenta-se um quadro comparativo com um resumo das principais características dos trabalhos que investigam o mapeamento estático de tarefas. As propostas revisadas apresentam estratégias estáticas de mapeamento de tarefas. Tal abordagem é aceitável somente para arquiteturas cuja proposta seja executar um conjunto limitado e predefinido de tarefas que compõem aplicações especificas. Em sistemas dinâmicos, o comportamento ou não pode ser predito, ou a predição será bastante imprecisa. Nesse caso, para aplicações com comportamento dinâmico, a validade das estratégias estáticas é comprometida, pois estas invariavelmente necessitam de relativo conhecimento sobre o comportamento da aplicação.

2.2.2. Mapeamento Dinâmico de Tarefas

Kim e outros [KIM03] [KIM05] [KIM07] consideram um ambiente heterogêneo, no qual tarefas chegam aleatoriamente, e possuem diferentes prioridades e *deadlines*. Em [KIM03] [KIM07] são comparadas oito heurísticas para mapeamento, incluindo duas heurísticas gulosas; uma baseada em *look-up table*, dentre outras. Após a validação sobre diferentes cenários, os Autores concluem que com relação aos *dealine*s, heurísticas gulosas apresentam melhor desempenho. Contudo, se o tempo de mapeamento é importante, então técnicas *look-up table* são recomendadas. Em [KIM05], os Autores admitem os mesmos modelos de tarefa e ambiente, agora considerando o consumo de energia como alvo. Sete

heurísticas são avaliadas, algumas delas diferentes das tratadas anteriormente. A heurística gulosa ainda apresentou o melhor resultado, mas em um tempo maior. Os Autores não definem adequadamente a arquitetura alvo, e não consideram o tempo de mapeamento.

TABELA 2.2. COMPARATIVO DOS TRABALHOS INVESTIGADOS SOBRE MAPEAMENTO ESTÁTICO DE TAREFAS.

Autores	*Modelo de arquitetura*	*Infra-estrutura de comunicação*	*Algoritmo*	*Modelo de aplicação*	*Métrica*	*Observações*
Mihal e Keutzer (2003)	Heterogênea	Barramento	-	Modelo alto-nível para uma ferramenta tipo Ptolemy	-	Trabalho centrado na modelagem da concorrência de aplicações.
Wu et al. (2003)	Heterogênea	Barramento	Genético	Tempo de execução e energia dissipada	Energia consumida	Trata escalonamento e mapeamento de tarefas aliados a estratégias de variação dinâmica de voltagem.
Lei e Kumar (2003)	Heterogênea	NoC malha	Genético	Volume de comunicação	Tempo total de execução	Apresenta uma ferramenta para mapeamento e uma modelagem dos atrasos do sistema.
Rhee et al; (2004)	Homogênea	NoC malha (roteamento xy)	-	Volume de dados e largura de banda necessária	Número de saltos dos pacotes e ocupação dos canais	Mapeamento de núcleos na NoC. Roteadores podem conter vários núcleos.
Hu e Marculescu (2003 - 2005)	Homogênea	NoC malha (roteamento xy)	Divisão e conquista	Volume de comunicação, comunicação e computação	Comunicações (Energia)	Mapeamento dos núcleos de uma aplicação em NoCs.
Manolache et al. (2005)	Homogênea	NoC malha	Tabu Search	Periodicidade, tempo de execução e deadline	Energia e garantia de latência	Mapeamento integrado de Tarefas e comunicações, aliado a duplicação de mensagens.
Lin et al. (2005)	Homogênea	NoC malha	SA e Divisão e Conquista	Volume de comunicação	Comunicações (Latência e tempo de execução)	O algoritmo proposto é baseado em um perfil de operação.
Murali e De Micheli (2004 - 2006)	Homogênea	NoC malha ou torus	Tabu Search	Largura de banda de comunicação necessária	Largura de banda, atraso de comunicação, área e energia	Ferramenta permite selecionar a topologia, selecionar tipos de roteamento, mapear os núcleos e realizar *floor planning*.
Marcon et al. (2005 - 2008)	Homogênea	NoC malha	Força bruta,SA tabu search e métodos gulosos	Dependências e volume de comunicação; computações e temporização das mensagens.	Comunicações (Energia) e Tempo de execução	Mapeamento de núcleos em uma NoC.
Orsila et al. (2005 - 2007)	Homogênea	Barramento ou NoC	SA para otimizações globais e migração de grupo para locais	Dependências, computação; e volume de comunicação	Tempo de execução	Sistema descrito em phyton e simulado em um cluster
Srinivasan e Chatha (2005)	Homogênea	NoC malha	Cutting tree	Largura de banda e latência necessárias	Largura de banda e latência resultante	Mapeamento de núcleos na NoC. Mapeamento integrado de tarefas e de comunicações
Ruggiero et al. (2006)	Homogênea (processadores ARM7)	Barramento (AMBA)	Programação linear inteira	Grafo de tarefas no estilo pipeline	Tráfego no barramento	Processadores executam o SO RTEMS. Problemas de mapeamento e escalonamento são tratados juntos.
Saeidi et al. Mehran et al. (2007)	Homogênea	NoC malha	Spiral	Grafo de tarefas (largura de banda)	Proximidade e ocupação dos canais	Proposta da ferramenta SMAP baseada no Matlab

Smit e outros [SMI05] apresentam um método *iterativo hierárquico* para o mapeamento de aplicações em SoCs heterogêneos, baseados em NoC. Uma aplicação é representada por um conjunto de tarefas e suas respectivas comunicações. No método proposto, primeiramente cada tarefa é atribuída a um tipo de recurso do sistema (*e. g.* FPGA, DSP, ARM), de acordo com suas restrições de desempenho. Em seguida, cada tarefa é atribuída a um recurso específico. Todas as tarefas são rearranjadas visando diminuir a distância entre tarefas comunicantes. Ao final, o mapeamento resultante é verificado, e caso não atenda as necessidades da aplicação, uma nova iteração é necessária. Os experimentos ba-

seiam-se no mapeamento de uma aplicação com 13 tarefas sobre um SoC 4x4. Além do proposto, outros dois métodos de mapeamento são avaliados de acordo com a ocupação da NoC, e tempo de execução da aplicação. O primeiro deles é o método *exaustivo*, executado por 10 horas até encontrar a melhor solução. O segundo é o algoritmo *minWeight* apresentado em [SMI04a] [SMI04b]. O método iterativo encontrou sua melhor solução (idêntica à do exaustivo) em apenas três iterações, ao passo que *minWeight* apresentou uma solução 5% pior comparada ao método exaustivo. Segundo os Autores, *minWeight* possui baixas escalabilidade e flexibilidade para a inclusão de restrições.

Ngouanga e outros [NGO06] pesquisam o mapeamento de tarefas em um MPSoC homogêneo, composto por PEs conectadas via NoC. Três tipos de PEs são definidos: mestre, escravos e interface de E/S. O PE mestre é responsável pelo mapeamento de tarefas. Cada PE é monotarefa e possui um *μkernel*, modelado através de redes de Petri. Nenhuma informação sobre o mapeamento existe *a priori*, o mapeamento é realizado de acordo com padrões de localidade tal como compartilhamento de canais de comunicação. Os Autores investigam dois algoritmos para mapeamento dinâmico: *SA* e *Força Direcionada*. O último seleciona a posição para a nova tarefa de acordo com uma força de atração proporcional ao volume de comunicação e a distância entre tarefas. Os Autores não demonstram preocupação com o atraso da heurística de mapeamento empregada sob a justificativa de que o tempo de processamento da tarefa deve torná-lo insignificante. O algoritmo *Força Direcionada* executou em um tempo uma ordem de grandeza menor que *SA*, embora o caminho médio entre duas tarefas seja equivalente em ambos os algoritmos. Nos resultados, é possível identificar que quanto maior o sistema (número de PEs), melhor tende a ser os resultados das heurísticas comparados a um mapeamento aleatório.

Wronski e outros [WRO06] avaliam dois algoritmos *bin packing* [GAR73] para mapeamento de tarefas em um MPSoC baseado em NoC. São eles: *Best Fit* (BF) e *Worst Fit* (WF). Uma ferramenta descrita em *SystemC* simula o comportamento dinâmico do MPSoC e estima a energia consumida pelo sistema, de acordo com o mapeamento da aplicação. Os roteadores da NoC são descritos no nível RTL. A estimativa de consumo de energia de cada tarefa é baseada em um parâmetro aleatório, que representa o número de chaveamentos por ciclo de relógio. Cada tarefa é então representada pelo seu pior tempo de execução e por seu número de chaveamentos. A energia consumida pela NoC relativa à comunicação é obtida através da biblioteca *Orion* [BEN04]. Nos experimentos realizados, foram analisados casos onde: (*a*) todas as aplicações executam em um único processador, (*b*) cada processador executa uma aplicação e, (*c*) cada processador executa uma única tarefa. As aplicações foram geradas a partir de um gerador de grafos TGFF. Os Autores concluem que estratégias de mapeamento podem resultar em tempos de execução muito semelhantes e ainda apresentarem consumos de energia muito diferentes. Em [BRI08], as heu-

rísticas são combinadas com a estratégia de *Clusterização Linear* (CL) proposta, que une as tarefas com alto nível de comunicação no mesmo PE. A combinação WF+CL apresenta redução significativa, entre 80 e 100%, nas violações do *deadline* das tarefas. Entretanto, a avaliação de energia consumida aponta para BF como melhor solução.

Hölzenspies e outros [HÖL07] investigam métodos para estimar o desempenho do mapeamento de aplicações de fluxo de dados, em MPSoCs baseados em NoC. Uma aplicação é modelada por um grafo de tarefas, onde os vértices são as tarefas e as arestas representam FIFOs. A comunicação, por sua vez, é representada pelo envio de *token*s através das arestas do grafo. No trabalho proposto, os recursos do MPSoC são gerenciados por um sistema operacional que executa em um dos processadores do MPSoC. Seu alvo consiste em minimizar o consumo de energia. Para isso, o SO usa informações coletadas em tempo de projeto sobre a aplicação (*i. e.* latência necessária e *throughput*), e sobre a arquitetura (*e. g.* número de processadores). Em tempo de execução, o SO determina quando o mapeamento deve ser realizado, e se o mesmo deve ser validado. Esse trabalho concentra-se nos métodos de estimativa, tendo por base um exemplo simples de aplicação, composta apenas por 3 tarefas que se comunicam na forma de um *pipeline*. Em [HÖL08], os Autores apresentam em maiores detalhes o processo de mapeamento (*i. e. iterativo hierárquico* [SMI05]). Os mesmo modelos e aplicações são adotados. O algoritmo executado sobre um processador ARM levou 4ms para apresentar uma solução de mapeamento adequada. Os Autores não fornecem comparações com outros métodos.

Chou e Marculescu [CHO07] apresentam uma estratégia incremental para mapeamento de tarefas em MPSoCs homogêneos. Os processadores do sistema são interconectados por uma NoC malha com roteamento xy. Ela possui caminhos de dados e controle distintos. Os processadores podem executar em dois níveis de voltagem distintos. As aplicações são representadas por grafos de tarefas, com informação do volume de dados transmitidos e largura de banda necessária para cada comunicação. O processador gerente é responsável por encontrar uma área contigua no MPSoC onde caiba a aplicação a ser mapeada, e por definir a posição das tarefas dentro desta área. Essa estratégia evita a fragmentação do sistema. Os experimentos empregam um MPSoC 7x7, e aplicações sintéticas geradas no TGFF, compostas por 5 a 10 tarefas cada. Comparado à solução exaustiva, o método proposto apresentou penalidade de 21% no consumo de energia. Entretanto, seu tempo de execução é em torno de 40μs, ao passo que o método exaustivo necessita de 26 horas para um MPSoC com 13 PEs. Comparado a um mapeamento aleatório, o método proposto apresenta 45% de redução no consumo de energia. Em [CHO08], os Autores estendem o trabalho para considerar também um *modelo de comportamento do usuário* no mapeamento das tarefas. O comportamento do usuário alimenta um perfil de operação da aplicação, que contém dados sobre sua periodicidade e volume de dados comunicados en-

tre suas tarefas. Duas estratégias de mapeamento são investigadas. A primeira delas consiste no método adotado no trabalho anterior. A outra estratégia define um formato de região para uma dada aplicação, e realiza transformações geométricas de rotações se necessário, para em seguida mapear a aplicação no MPSoC. A proposta apresenta penalidade de 10 e 60% na energia consumida, comparado aos métodos exaustivo e aleatório, respectivamente. Tempos de mapeamento medidos para grafos de aplicações reais (telecomunicação, redes, automação e automotivas) são na rodem de 47μs.

Mehran e outros [MEH08] investigam o desempenho do mapeamento de tarefas em uma NoC malha, com dimensões 4x4. Esse trabalho apresenta uma evolução do proposto em [MEH07]. De acordo com os Autores, as tarefas de uma dada aplicação com comportamento dinâmico podem ser necessárias em diferentes instantes. Assim, quando uma nova tarefa é necessária, ou uma tarefa termina sua execução, um novo mapeamento da aplicação pode ser necessário. Para isso, os Autores propõem duas alternativas que consistem no mapeamento *total* ou *parcial* da aplicação. Ambas empregam o algoritmo *Spiral*, primeiramente proposto em [SAE07]. Os experimentos realizados empregam 10 aplicações sintéticas, com número de tarefas entre 9 e 25, num total de 100 cenários. Os resultados obtidos demonstram que o mapeamento *parcial* apresenta o melhor resultado em 82% dos experimentos, com redução entre 5 e 28% no tempo para mapear a aplicação.

Faruque e outros [ALF08] apresentam uma solução para mapear tarefas em MPSoCs heterogêneos baseados em NoC malha. Sua principal contribuição diz respeito a um método de *mapeamento distribuído* baseado em *agentes*, em contraproposta ao método que emprega um único processador como gerente centralizado. O MPSoC é dividido em *clusters virtuais*. Cada um deles possui um *agente local* responsável por mapear tarefas em seu interior. *Agentes globais* decidem em qual *cluster* as tarefas serão mapeadas. Assim, o processo de mapeamento reside na negociação entre agentes locais e globais. Além disso, o monitoramento dos recursos é distribuído entre os agentes locais. A modelagem das aplicações se dá através de um grafo de tarefas, o qual informa as dependências e as comunicações (largura de banda necessária) entre tarefas. Cada aplicação é mapeada por completo pelo agente local, segundo o algoritmo estático apresentado em [HAN05]. Os experimentos realizados comparam a estratégia proposta a alguns dos algoritmos aqui propostos (NN, MACL e PL discutidos no Capítulo 4), previamente publicados em [CAR07]. A partir de MPSoCs com dimensões 12x12, o método distribuído apresenta melhores resultados, em contraproposta ao centralizado. Para um MPSoC 32x64 o esforço de mapeamento é 7 vezes menor, e o volume de dados monitorados transmitidos através da NoC é 10 vezes menor. Esses resultados são facilmente justificados, pois a proposta permite paralelismo no mapeamento de aplicações caso sejam requisitadas simultaneamente. Entretanto dentro de uma mesma aplicação as tarefas ainda são mapeadas uma por vez.

CONSIDERAÇÕES SOBRE MAPEAMENTO DINÂMICO

A Tabela 2.3 apresenta um quadro comparativo com as principais características de cada um dos trabalhos revisados sobre mapeamento dinâmico. A última linha da Tabela adianta algumas características do trabalho aqui proposto. Como pode ser notado, surgiram muitos trabalhos nos últimos dois anos sobre mapeamento dinâmico [CHO07] [MEH08] [ALF08]. Alguns trabalhos revisados revisitam estratégias estáticas de mapeamento [WRO06] [NGO06], sem qualquer preocupação com o tempo de mapeamento, assumido como insignificante. A maioria dos trabalhos trata o mapeamento da aplicação por completo. No que diz respeito à arquitetura admitida pelos Autores, existem tanto trabalhos sobre MPSoCs homogêneos, quanto heterogêneos, em proporção similar. O uso de NoCs acontece em todos os trabalhos revisados. O gerenciamento de recurso, na maioria dos trabalhos revisados, é baseado em uma estratégia centralizada. Como exceção, [ALF08] sugere o gerenciamento é distribuído. Tal proposta é importante à medida que cresce as dimensões dos MPSoCs (*e. g.* 32x32), conforme comprovado pelo próprio Autor.

TABELA 2.3. COMPARATIVO DOS TRABALHOS INVESTIGADOS SOBRE MAPEAMENTO DINÂMICO DE TAREFAS.

Autores	*Modelo de arquitetura*	*Infra-estrutura de comunicação*	*Algoritmo*	*Modelo de Aplicação*	*Métrica*	*Observações*
Kim et al. (2003, 2005, 2007)	Heterogênea	-	Guloso, look-up table, etc	Conjunto de tarefas com prioridades e deadlines	Deadlines e energia consumida	Arquitetura não definida. Várias heurísticas são avaliadas.
Smit et al. (2005)	Heterogênea	NoC	Iterativo Hierarquico	Grafo de tarefas com volume de comunicação	Ocupação da NoC e Tempo de Execução	Comparações com métodos exaustivo e aleatório.
Ngouanga et al. (2006)	Homogênea	NoC (similar à Hermes)	SA e força direcionada	Grafo de tarefas com volume de comunicação	Tempo de Comunicação	Atraso de mapeamento é considerado insignificante. Replicação de tarefas.
Wronski et al. Brião et al. (2006, 2008)	Homogênea	NoC (Xpipes)	BF, WF, CL+BF e CL+WF	Pior tempo de execução e número de chaveamentos	Energia consumida e deadlines	Simulador em SystemC com roteador em nível RTL.
Hölzenspies et al. (2007)	Homogênea	NoC	Iterativo Hierárquico	Grafo de tarefas com volume de comunicação	Energia consumida	Estimativa de desempenho do mapeamento. Estudos de casos com poucas tarefas.
Chou e Marculescu (2007, 2008)	Homogênea (≠s voltagens)	NoC	Seleção de área + Mapeamento	Grafo de tarefas com volume de comunicação e largura de banda necessária	Energia consumida e Fragmentação do sistema	Mapeamento em dois estágios: Seleciona uma área contigua e mapeia as tarefas nessa área.
Mehran et al. (2008)	Homogênea	NoC	Spiral	-	Tempo de mapeamento	Mapeamento total ou parcial da aplicação quando uma nova tarefa da aplicação chega.
Faruque et al. (2008)	Heterogênea	NoC	Agentes distribuídos	Grafo de tarefas com volume de comunicação e largura de banda necessária	Tempo de mapeamento	Gerenciamento distribuído do sistema. Mapeamentos em paralelo. Comparações com NN, MACL e PL.
Proposta	**Heterogênea**	**NoC (Hermes)**	**FF, NN, MACL, MMCL, PL, BN**	**Grafo de tarefas com volume de comunicação e largura de banda necessária**	**Ocupação da NoC**	**Tarefas e Controlador em SystemC e NoC em VHDL.**

2.3. Outras Estratégias Dinâmicas

Migração consiste na transferência de tarefas em tempo de execução entre diferentes máquinas. Anteriormente, o interesse em migração de tarefas se deu no domínio da

computação paralela. O estudo de MPSoCs traz a migração novamente à pauta. Ela pode ser empregada para obter distribuição de carga dinâmica e assegurar tolerância a falhas. Além disso, quando um gargalo de desempenho é encontrado, a migração pode maximizar a economia de energia, gerenciar a temperatura do CI, dentre outros. Uma migração exige a interrupção da execução da tarefa no PE fonte, e o *salvamento de contexto* para que o mesmo seja retomado quando a tarefa reiniciar no PE destino. O processo que garante que uma dada tarefa possa continuar sua execução no destino denomina-se *relocação*. No caso onde os PEs fonte e destino são arquiteturas diferentes, o relocador seleciona o código objeto da tarefa compilado/sintetizado para o PE destino, e ainda se encarregar de carregar o contexto salvo neste PE. Outra estratégia empregada em tempo de execução é a *replicação de tarefas*, que visa evitar gargalos na computação/comunicação. Diferentemente da migração, na replicação, a tarefa continua executando no PE fonte.

Kalte e Porrmann [KAL05a] propõem dois métodos para salvamento de contexto para sistemas reconfiguráveis, denominados *shutdown process* e *readback*. O trabalho de Kalte é fundamentado no processo de relocação de tarefas. Em [KAL05b] e [KAL06a], apresenta-se dois módulos para relocação de módulos de hardware em FPGAs comerciais.

Nollet e outros [NOL05a] [NOL05b] apresentam um método para migrar tarefas entre os domínios hardware e software. A decisão do instante da migração é tomada em função do desempenho da aplicação. O método proposto emprega *pontos de migração* para definir quando uma dada tarefa pode ser migrada. A plataforma GECKO [MAR04], composta por um PDA e um FPGA, foi desenvolvida para validar a estratégia proposta.

Bertozzi e outros [BER06] empregam *checkpoints* para definir quando uma dada tarefa deve ser migrada. Um *middleware* implementado no *μClinux* permite migrações em pontos específicos, representados por *checkpoints*, inseridos manualmente no código da tarefa pelo programador do sistema. Para avaliar o desempenho do sistema e os custos da migração, diversos testes foram feitos, e sua viabilidade foi comprovada.

Streichert e outros [STE06] propõem um algoritmo que visa otimizar o mapeamento tarefas. Quando um nodo da rede falha, as tarefas que estavam neste nodo são remapeadas, réplicas se tornam tarefas principais, e novas rotas para comunicação são estabelecidas. Esse passo é denominado *reparação rápida*. Em seguida, no *passo de otimização*, um novo mapeamento é encontrado a fim de minimizar o tráfego de dados na rede.

Götz e outros [GÖT07] também investigam migração entre domínios de software e hardware. Segundo os Autores, a principal dificuldade é a diferença entre paradigmas seqüencial *versus* espacial. A computação de uma tarefa é representada como um grafo de

transição de estados, onde estado representam blocos de computação. Cada transição de estado representa um possível *ponto de migração*. Este define um ponto de encontro entre o código compilado para o processador e sua versão sintetizada para FPGA.

Sassatelli e outros [SAS07] exploram a adaptabilidade em MPSoCs. Esse trabalho é centrado no processo de *replicação de tarefas*, realizado quando um gargalo de desempenho é encontrado. A adaptação do sistema é baseada em monitores que coletam informações dos processadores, em tempo de execução. Segundo os Autores, o ganho de desempenho da aplicação é superior ao atraso causado na replicação. Em [SAI08], a API desenvolvida no trabalho anterior foi compatibilizada com o padrão MPI. Os experimentos baseiam-se em um sistema composto por 16 FPGAs, conectados segundo uma topologia malha.

Farag e outros [FAR07] apresentam o processo de migração como uma estratégia para desfragmentar o dispositivo, otimizando a utilização da área. O sistema é modelado na forma de uma matriz 2D, sem considerar a infra-estrutura de comunicação. Quando uma dada tarefa não pode ser mapeada porque existe área livre suficiente, mas ela está fragmentada, a migração das tarefas é realizada, a fim de tornar a área livre contigua.

Barcelos e outros [BAR07] investigam o impacto da organização de memória no atraso da migração de tarefas. Segundo os Autores, a estratégia distribuída requer muita memória, ao passo que a compartilhada apresenta um excessivo atraso na migração. Um modelo híbrido de organização de memória é proposto. Comparado às estratégias compartilhada e distribuída, ele reduz em 24% e 10% o consumo de energia, respectivamente.

Brião e outros [BRI07] investigam o impacto da migração em aplicações *soft real-time*. No trabalho, o atraso decorrente da migração corresponde ao tempo de transmissão do código e dos dados. O tempo para leitura da memória é desprezado. Após avaliar aplicações sintéticas e também um *benchmark* de telecomunicação, os Autores concluem que a migração pode ser aplicada para garantir o *deadline* de aplicações *soft real-time*.

CONSIDERAÇÕES SOBRE MIGRAÇÃO DE TAREFAS

Os trabalhos de migração investigados em geral indicam pontos específicos para migrar tarefas [BER06] [NOL05a]. Além disso, o processo de salvamento de contexto também desperta interesse [KAL05a]. Aplicações embarcadas possuem restrições de tempo real, e a migração transparente de tarefas pode causar violações na temporização do sistema em virtude do atraso inserido. Em adição, tarefas podem migrar com muita freqüência entre PEs do MPSoC, reduzindo a capacidade de predizer o tempo de execução do sistema. Por tudo isso, técnicas menos transparentes e mais controláveis de migração podem

ser necessárias neste contexto, tal como migração controlada pelo usuário ou pela aplicação. Na revisão realizada, os Autores na sua maioria optam por mecanismos menos transparentes [BER06] [NOL05a].

Os esforços para avaliar a validade do emprego da estratégia de migração encontram-se em estágio inicial. Alguns trabalhos estendem o conceito de migração para duplicação de tarefas, que pode visar tolerância a falhas [STE06], ou garantir o desempenho das aplicações [SAS07]. Alguns trabalhos investigam métodos para reduzir a energia consumida durante uma migração [BAR07], enquanto outros visam comprovar a viabilidade do emprego de migração em aplicações de tempo real [BRI07].

A Tabela 2.4 apresenta um quadro comparativo com um resumo das características dos trabalhos sobre migração de tarefas, aqui revisados.

TABELA 2.4. COMPARATIVO DOS TRABALHOS INVESTIGADOS SOBRE MIGRAÇÃO DE TAREFAS.

Autores	*Modelo de arquitetura*	*Salvamento de contexto*	*Instante de migração*	*Métrica*	*Observações*
Kalte et al. (2005, 2006)	HW controlador Barramento Área reconfigurável (Homogênea)	Shutdown process ou readback	-	Fragmentação do dispositivo	Trabalho baseado na relocação de tarefas usando o módulo REPLICA.
Nollet et al. (2005)	8 processadores Barramento (Homogênea)	-	Pontos de Migração	Largura de banda ocupada e Tempo de execução	Migração entre HW e SW de acordo com requisitos de desempenho.
Bertozzi et al. (2006)	Processadores ARMv7 conectados via barramento (Homogênea)	Controlada pelo usuário e através de mensagens.	Checkpoints inseridos manualmente no código	Balanceamento de carga	Suporte implementado na forma de um middleware para uClinux que permite migrações
Streichert et al. (2006)	Rede de sensores, atuadores e controladores (processador + FPGA)	-	Defeitos	Tráfego de dados	Migra tarefas de nodos defeituosos em dois passos: reparação rápida e otimização.
Götz et al. (2007)	Processador PPC Barramento HW auxiliar (FPGA)	Serviços do SO: get context, set context,	Ponto de migração	-	Migração entre HW e SW.
Sassatelli et al. (2006, 2007)	Processadores Plasma NoC (Homogênea)	-	Replicação realizada em um gargalo de desempenho	Tempo de execução	Trabalho centrado no processo de replicação de tarefas.
Farag et al. (2006, 2007)	Matriz 2D (Homogênea)	-	Se nova tarefa necessita ser mapeada mas o dispositivo está fragmentado	Fragmentação do dispositivo	Migração empregada para desfragmentar o sistema. (não considera o meio de comunic.)
Barcelos et al. Brião et al. (2006, 2007)	MPSoC baseado em NoC (Homogênea)	-	-	Energia consumida na migração do código	Investiga impacto da organização de memória, e da migração em sistemas de tempo real.

3. PROPOSTA DE ORGANIZAÇÃO DE MPSOC

O desenvolvimento de um MPSoC apresenta muitos graus de liberdade ao projetista. Dentre eles encontram-se: *(i)* o número e a natureza dos elementos de processamento; (*ii*) a infra-estrutura de comunicação; (*iii*) a organização de memória; e (*iv*) os mecanismos de controle da operação do sistema.

Com relação à *natureza dos PEs*, MPSoCs homogêneos são mais adequados para o uso de técnicas de migração e replicação de tarefas, ao passo que os heterogêneos tendem a suportar uma maior variedade de aplicações. Além disso, o uso de IPs específicos pode prover maior desempenho a MPSoCs heterogêneos. Enquanto a natureza dos PEs é função do domínio de aplicação alvo, o *número de PEs* a serem empregados no sistema deve considerar também a quantidade de lógica disponibilizada pela tecnologia atual.

No que diz respeito à *infra-estrutura de comunicação*, NoCs dominam devido a sua maior escalabilidade comparado a barramentos e conexões ponto-a-ponto dedicadas. Barramentos em geral suportam a comunicação em sistemas com até poucas dezenas de módulos, enquanto as NoCs podem suportar centenas. Além disso, o paralelismo na comunicação é outro fator atrativo de NoCs. Um barramento suporta apenas uma comunicação por vez. O emprego de uma hierarquia de barramentos permite paralelismo na comunicação quando elas ocorrem em graus diferentes da hierarquia. Entretanto, as comunicações entre os módulos de graus diferentes ainda limitam o paralelismo na comunicação.

Com relação à *organização de memória* empregada, segundo [BAR07], uma organização hibrida de memórias distribuídas e centralizadas representa a melhor alternativa para reduzir o tempo de migração de tarefas. Em geral, estratégias centralizadas de memória conduzem a gargalos de comunicação, enquanto as distribuídas sugerem maior complexidade de gerenciamento.

Os mecanismos usados para *controle da operação do sistema* são também importantes. Esses podem ser baseados no reuso de um SO conhecido (*e. g.* Linux); na personalização do SO para o sistema alvo; ou até mesmo no desenvolvimento de um SO novo. Este

pode valer-se até mesmo de complementos implementados em hardware para aumento do desempenho. Os métodos adotados para *controle do consumo de energia* também ganham importância, sobretudo no domínio de aplicações sem fio. Dentre eles, pode-se citar o emprego de estratégias para variação dinâmica de voltagem [BUR00] e *clock gating* [OH98]. *Modelos de programação e concorrência* do sistema são relevantes. A programação em sistemas paralelos, tal como *clusters*, há muito exige grande esforço do meio acadêmico na busca de melhores modelos de programação. Agora, parte desses trabalhos necessita ser adequada ao novo cenário imposto pelos MPSoCs.

Na Seção 3.1, apresenta-se a organização proposta para o MPSoC alvo da pesquisa em mapeamento de tarefas. A Seção 3.2, por sua vez, apresenta o modelo de aplicação adotado. Na Seção 3.3, apresenta-se a infra-estrutura de comunicação adotada. O protocolo definido para comunicação entre as tarefas e o controle do sistema é discutido na Seção 3.4. A representação dos recursos do sistema é discutida na Seção 3.5. A Seção 3.6 dedica-se a discussão de mecanismos para monitoramento da ocupação dos canais da NoC. Na Seção 3.7 apresenta-se as três estratégias de modelagem de sistema investigadas.

3.1. Proposta de Organização do MPSoC Heterogêneo

Segundo Grant Martin [MAR06]: "*Perhaps one of the most interesting challenges in developing an MPSoC architecture is to ensure it is sufficiently general for a variety of applications in the particular domain it is designed for. Incorporating many processors is one way of extending the life of an MPSoC platform, since they are programmable, and sizing system memory and communications resources to include some extra design margin will give the platform downstream flexibility. Another approach is to use reconfigurable logic on die to offer in-field programmability and flexibility that may offer greater performance than software-only approaches.*"

Motivado pela afirmação acima, a organização de MPSoC adotada neste trabalho é um MPSoC heterogêneo, composto por um conjunto de elementos de processamento (PEs) conectados através de uma NoC. A heterogeneidade do MPSoC consiste na natureza mista dos PEs, os quais suportam a execução de tarefas de software (via *instruction set processors* – ISPs) e tarefas de hardware (via IPs dedicados ou lógica reconfigurável embarcada – RL). O uso de lógica reconfigurável permite a carga de tarefas de hardware no sistema em tempo de execução, usando a estratégia de reconfiguração dinâmica.

A Figura 3.1 ilustra a organização genérica do MPSoC proposto. Quando o MPSoC inicia sua execução, somente tarefas inicialmente necessárias são alocadas no sistema. Novas tarefas são alocadas quando uma dada tarefa necessita se comunicar com tarefas ainda não alocadas. Dessa forma, ao longo do tempo, o número de tarefas alocadas no MP-

SoC varia, e os recursos disponíveis podem ser insuficientes para suportar a execução de todas as tarefas necessárias. Com a finalidade de controlar a operação do sistema, um dos processadores do MPSoC é reservado, denominado *processador gerente* (MP – *Manager Processor*). Dentre as *tarefas de controle* atribuídas ao MP, encontram-se:

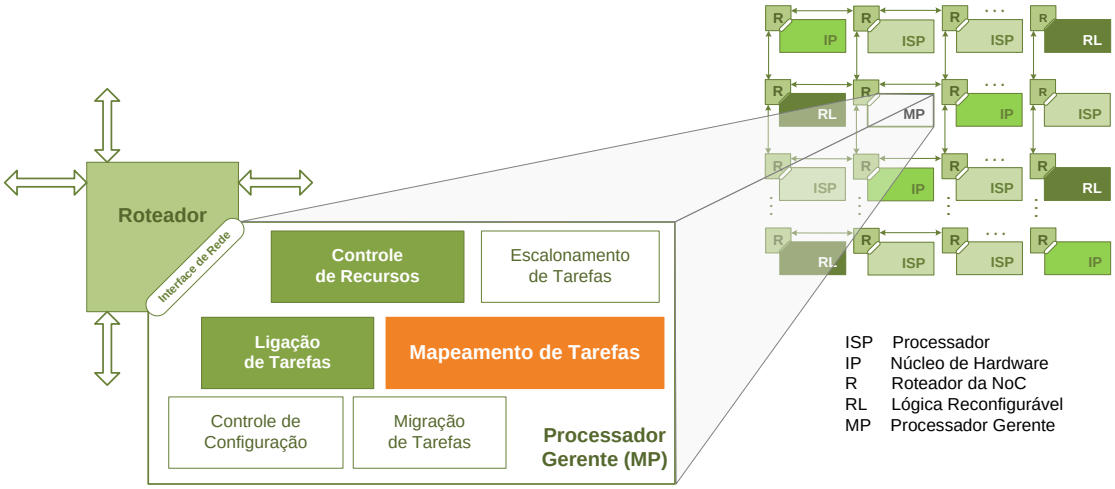


FIGURA 3.1. ORGANIZAÇÃO DE MPSOC HETEROGÊNEO ALVO DO PRESENTE TRABALHO.

Controle de recursos: responsável por manter informação atualizada sobre o estado de ocupação dos recursos do sistema, incluindo principalmente os elementos de processamento, e os canais da NoC. Como será apresentado adiante, no presente trabalho o controle de recursos é baseado em um esquema de monitoramento distribuído, que captura a ocupação dos recursos e envia tal informação ao processador gerente. Esse, por sua vez, armazena tal informação em um conjunto de matrizes de ocupação de recursos.

Escalonamento de tarefas: responsável por determinar a ordem (temporal) na qual as aplicações e suas respectivas tarefas serão executadas. O escalonamento de tarefas não faz parte do escopo desse trabalho. Ele é baseado em uma estratégia de filas, sem considerar políticas de preempção. São implementadas três filas, uma para cada tipo de tarefa possível (*i. e. hardware*, *software* e *inicial*). Uma dada tarefa é inserida em uma dada fila se não existem recursos do seu tipo disponíveis, e ela deve aguardar nesta fila até que esta condição mude.

Ligação de tarefas: precede o mapeamento de tarefas em sistemas heterogêneos. Ela é responsável por assegurar a correta alocação de tarefas aos recursos do sistema, como por exemplo, atribuição de tarefas de software a processadores, e atribuição de tarefas de hardware à lógica configurável embarcada. No caso de MPSoCs homogêneos, essa operação inexiste.

Mapeamento de tarefas: consiste na definição do posicionamento (espacial) de cada uma das tarefas no sistema. A estratégia proposta neste trabalho é descrita em detalhes no próximo Capítulo. Ela vale-se de informação sobre a ocupação dos canais da NoC para definir um mapeamento que reduz os congestionamentos.

Migração de tarefas: responsável pela modificação do posicionamento das tarefas alocadas no sistema, de acordo com diferentes objetivos, tais como balanceamento de carga e redução da potência dissipada. Tal como o escalonamento, a migração de tarefas também não faz parte do escopo do trabalho proposto.

Configuração de tarefas: operação de carga das tarefas nos recursos do sistema. No caso de tarefas de hardware, a configuração consiste no envio dos bits de configuração (i.e. *bitstream*) para a lógica configurável embarcada. O *bitstream* atribui determinado comportamento ao hardware reconfigurável. No caso de tarefas de software, a configuração consiste no envio e gravação de códigos objeto de uma dada tarefa na memória do processador alvo. O controle de configuração não faz parte do escopo do trabalho. Ele é simulado com base em experimentos prévios que permitem estimar os atrasos de configuração [MÖL06] [MÖL07].

O processador MP define uma abordagem centralizada de gerenciamento, que pode apresentar problemas relativos à escalabilidade, conforme mencionado em [FAR07]. Além disso, o gerenciamento centralizado sofre com o problema de ponto de falha único, e pode transformar o gerente em um gargalo do sistema, visto que todo o tráfego de pacotes de controle envolve o roteador no qual o MP foi alocado. Assim, o estudo de técnicas distribuídas de controle necessita ser investigado à medida que cresce o número de PEs em um MPSoC. Conforme, resultados prévios apresentado em [FAR07], para MPSoCs com dimensões 32x32, estratégias de gerenciamento distribuído otimizam o desempenho do sistema com relação ao tempo de mapeamento. Como nos experimentos realizados, apresentados mais adiante, foram simulados MPSoC com dimensões até 8x8, o emprego de uma estratégia de controle distribuído não se fez necessário. Além disso, a maioria dos trabalhos revisados considera a implementação ou simulação de MPSoCs ainda menores [SMI05] [NGO06] [CHO07] [MEH08] [HÖL08].

3.2. Modelagem de Aplicações

Antes de discutir em maiores detalhes o comportamento do sistema, é importante definir a modelagem adotada para a representação das aplicações. No estudo sobre MPSoCs, em geral aplicações são modeladas através de grafos. Hu e Marculescu [HU03]

[HU05] e Lin e outros [LIN05] empregam grafos, onde vértices representam tarefas, e pesos atribuídos às arestas representam o volume de dados transmitidos entre as tarefas conectadas. Marcon e outros [MAR05a] usam dois grafos, o primeiro com informação sobre a dependência e o volume de comunicação entre as tarefas, e o segundo que contém a ordenação dos pacotes. Em [MAR05b], os Autores apresentam um modelo mais detalhado que também considera os tempos de computação das tarefas. Além do volume de dados, Rhee e outros [RHE04] consideram a largura de banda necessária para as comunicações, enquanto Srinivasan e Chatha [SRI05] consideram tanto a largura de banda quanto a latência necessárias para a transmissão dos pacotes através da NoC.

No presente trabalho, uma dada *aplicação* é representada por um *grafo dirigido*, como esboçado na Figura 3.2. Neste grafo, *vértices* representam *tarefas*, e *arestas* indicam as *comunicações* realizadas entre as tarefas das aplicações. Os vértices representados por círculos com linha dupla referem-se às *tarefas iniciais das aplicações*. Cada aplicação possui apenas uma dada tarefa inicial, a qual é iniciada tão logo a aplicação seja disparada pelo usuário. Os demais vértices indicam *tarefas de software* e de *hardware*, representadas pelos círculos claros e escuros, respectivamente. As arestas entre as tarefas possuem pesos referentes ao *volume* e *taxas* de comunicação entre as tarefas, em ambos os sentidos. Cada aresta define assim, um *par de tarefas mestre-escravo*, onde a tarefa denominada *mestre* necessita requisitar o mapeamento da sua *escrava*, antes da comunicação propriamente dita ser realizada. Tal protocolo é discutido em detalhes na Seção 3.5. Conforme a Figura 3.2, cada aresta de um dado grafo de tarefas possui quatro pesos atribuídos {*Vms*, *Rms*, *Vsm*, *Rsm*}, os quais definem o volume *V* e a taxa *R* de transmissão de dados entre as tarefas no sentido mestre-escrava *ms* e escrava-mestre *sm*.

FIGURA 3.2. EXEMPLO DE APLICAÇÃO, REPRESENTADA POR UM GRAFO DE TAREFAS.

Na implementação do MPSoC os processadores executam apenas um vértice de uma dada aplicação por vez. Essa abordagem conduz à modelagem de processadores monotarefa. Contudo, cada vértice do grafo pode representar não só uma única tarefa, mas sim um conjunto de tarefas. A operação de *particionamento* [ARA08], realizada em

tempo de projeto, é responsável por dividir uma dada aplicação, em um conjunto de tarefas. Em geral, cada conjunto é definido de maneira a otimizar o uso dos recursos da plataforma alvo. Logo, é possível também admitir que cada PE pode executar um conjunto de tarefas, segundo um modelo multitarefa.

O presente trabalho está focado no domínio de aplicações de fluxo de dados, tais como processamento para transmissão sem fio de sinais de áudio e vídeo, e processamento multimídia. Aplicações neste domínio são caracterizadas por um processamento local simples, porém realizado sobre uma grande quantidade de dados. Além disso, faz-se necessária a garantia de desempenho na transmissão dos dados, tanto quanto no processamento dos mesmos. Nesse caso, o emprego de técnicas de controle de desempenho do sistema para garantia dos serviços é obrigatório, sobretudo aquelas que visam evitar congestionamentos na NoC.

3.3. NoC Hermes

Nesta Seção, discute-se a infra-estrutura de comunicação empregada no MPSoC alvo. Foi definido o emprego de uma infra-estrutura baseada em NoC devido a fatores de escalabilidade e paralelismo, corroborado pelo fato de que a maioria dos trabalhos investigados adotou tal abordagem [CHO07] [MEH07] [ALF08]. A experiência do grupo GAPH na área de NoCs contribuiu significativamente para o emprego da Hermes [MOR04]. Além disso, ela disponibiliza um conjunto de ferramentas [GAP07] que permite sua geração de acordo com diferentes parametrizações; geração de cenários de tráfego; simulação e prototipação do sistema; e avaliações da latência dos pacotes e dissipação de potência. Dentre as parametrizações possíveis constam as dimensões da NoC, o algoritmo de roteamento, o controle de fluxo, o tamanho dos *buffers*, e a largura de *flit*.

O roteador da Hermes é ilustrado na Figura 3.3. Ele possui cinco portas, quatro delas são conectadas aos roteadores vizinhos, formando uma topologia malha 2D. A outra porta, denominada local, é conecta ao elemento de processamento, ou IP. Cada enlace entre os roteadores da NoC possui dois canais de 16 bits de comunicação, um em cada um dos sentidos, permitindo a transmissão bidirecional simultânea entre os roteadores. Outros parâmetros adotados são: (*i*) chaveamento de pacotes *wormhole*; (*ii*) *buffers* de entrada; (*v*) controle de fluxo por *handshake* e (*iv*) roteamento xy determinístico. O protocolo *handshake* adotado permite a transmissão de um *flit* de 16 bits a cada dois ciclos de relógio.

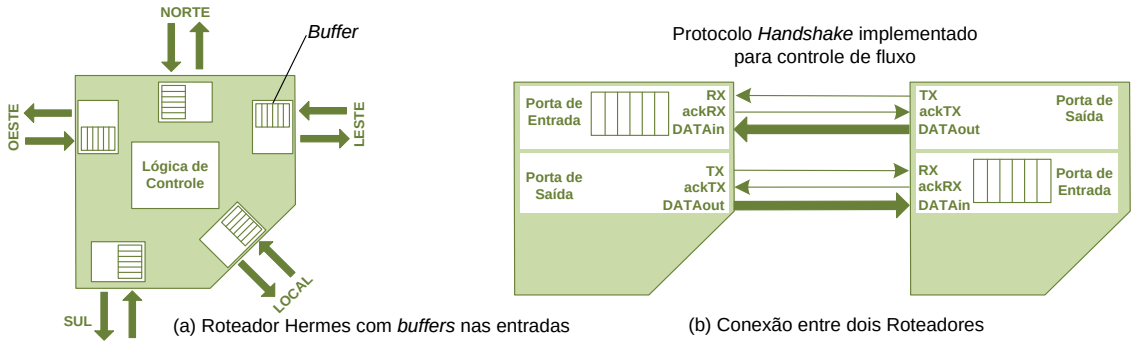


FIGURA 3.3. NOC HERMES [MOR04]: (A) ROTEADOR E (B) CONTROLE DE FLUXO HANDSHAKE.

3.4. Protocolo de Comunicação entre Tarefas

O protocolo de comunicação proposto baseia-se no uso de quatro tipos de pacotes, de acordo com a Figura 3.4. Para facilitar o entendimento, a NoC não é representada na figura, embora todos os pacotes sejam transmitidos através dela. Três pacotes, ditos de *controle* são empregados na comunicação entre as tarefas e o processador gerente MP. São eles: REQUEST, RELEASE e NOTIFY. A comunicação entre duas tarefas se dá através de pacotes do tipo GENERAL.

O pacote REQUEST informa ao MP o identificador de uma nova tarefa a ser inserida no sistema, e os respectivos volume e taxas de comunicação. Esse pacote é usado para a requisição de uma dada tarefa, tal como seu nome indica. O pacote de RELEASE, por sua vez, informa ao MP que um determinado PE teve sua tarefa concluída, liberando tal recurso para que outra tarefa seja nele alocada. Dois pacotes de controle do tipo NOTIFY são enviados pelo MP após o mapeamento de uma dada tarefa, um para cada tarefa envolvida na comunicação (*i. e.* escrava recém mapeada e sua mestre). Esses pacotes informam os endereços das tarefas a fim de possibilitar a correta transmissão dos pacotes.

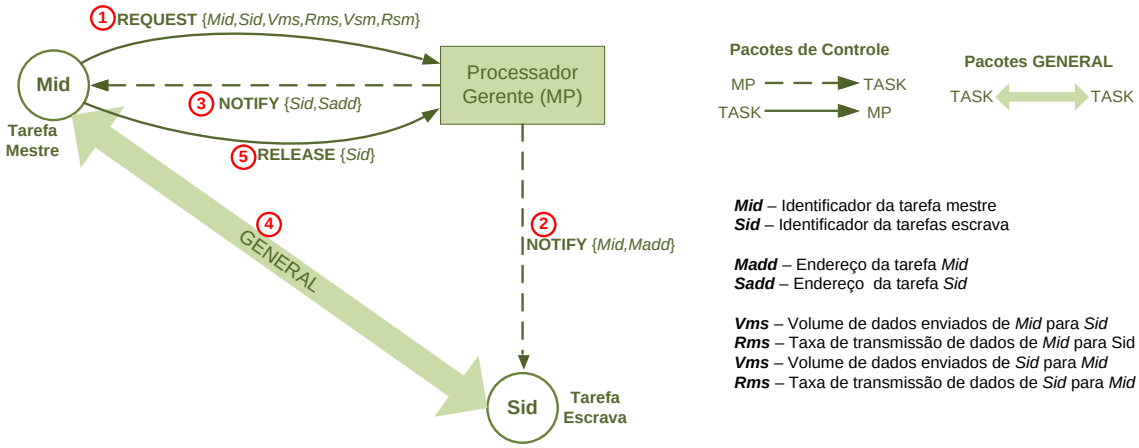


FIGURA 3.4. PROTOCOLO DE COMUNICAÇÃO ENTRE AS TAREFAS E O PROCESSADOR GERENTE MP.

Vistos os conceitos de aplicação e o protocolo definido para comunicação entre as tarefas, o comportamento genérico de uma dada tarefa é discutido agora. Ele compreende sete passos. São eles:

1. A tarefa aguarda um pacote NOTIFY, que contém o endereço de sua tarefa mestre, os volumes e as taxas de comunicação para iniciar sua operação. Este pacote só deve chegar quando essa tarefa for solicitada. Seu recebimento indica o sucesso no mapeamento. As tarefas iniciais das aplicações (*i. e.* não possuem mestre) são inicializadas diretamente pelo MP assim que a aplicação é disparada. Ou seja, o MP age como mestre de todas as tarefas iniciais de cada uma das aplicações. As tarefas iniciais avançam direto ao passo 3.

2. A tarefa recebe todos os pacotes GENERAL a partir de sua mestre. O número de pacotes é computado através dos devidos volumes de comunicação. No instante da simulação, as taxas de comunicação são empregadas para a geração de tráfego.

3. A tarefa envia pacotes de REQUEST para o MP, um para cada uma de suas tarefas escravas. As tarefas que não possuem escravas avançam diretamente ao passo 6.

4. A tarefa inicia a comunicação (usando pacotes GENERAL) com cada uma das suas escravas, quando o pacote NOTIFY com o endereço dessa escrava chegar a partir de MP. O número de tarefas que executam em paralelo é função dos recursos disponíveis, bem como da demanda da aplicação.

5. Quando a tarefa identifica que todos os pacotes foram trocados com uma dada escrava, ela envia um pacote RELEASE para MP, com o endereço dessa escrava para o recurso ser liberado, ou até mesmo reusado.

6. Após liberar todos os seus escravos, a tarefas realiza o processamento dos dados.

7. Se a tarefa não for uma tarefa inicial, ela envia os resultados do seu processamento (i.e. através de pacotes GENERAL) para sua tarefa mestre. As tarefas que são iniciais não possuem uma tarefa mestre, e assim não enviam resultados a outras tarefas. Ao invés disso, elas podem apresentar os resultados ao usuário através de uma dada interface de E/S, por exemplo.

A Figura 3.5 apresenta um diagrama de interação entre as tarefas de uma dada aplicação alvo, de acordo com os passos anteriores. Considerando as comunicações entre as tarefas *t0*, *t2* e *t3* do grafo alvo em destaque na figura, cada um dos 7 passos é indicado tendo *t2* como tarefa de referência. No passo 1, *t2* aguarda até receber o pacote NOTIFY a partir de MP com o endereço da sua mestre *t0*. No passo 2, *t2* recebe os pacotes GENERAL enviados por *t0*. Em seguida, *t2* requisita sua escrava *t3* ao MP, no passo 3. Após o mapeamento de *t3*, MP envia à *t2* o endereço da escrava mapeada através de um pacote NOTIFY, recebido por *t2* no passo 4. Após o término da comunicação com *t3*, *t2* envia o pacote RELEASE para MP no passo 5. No passo 6, *t2* realiza seu processamento, e no passo 7 ela envia os devidos resultados a sua mestre *t0*.

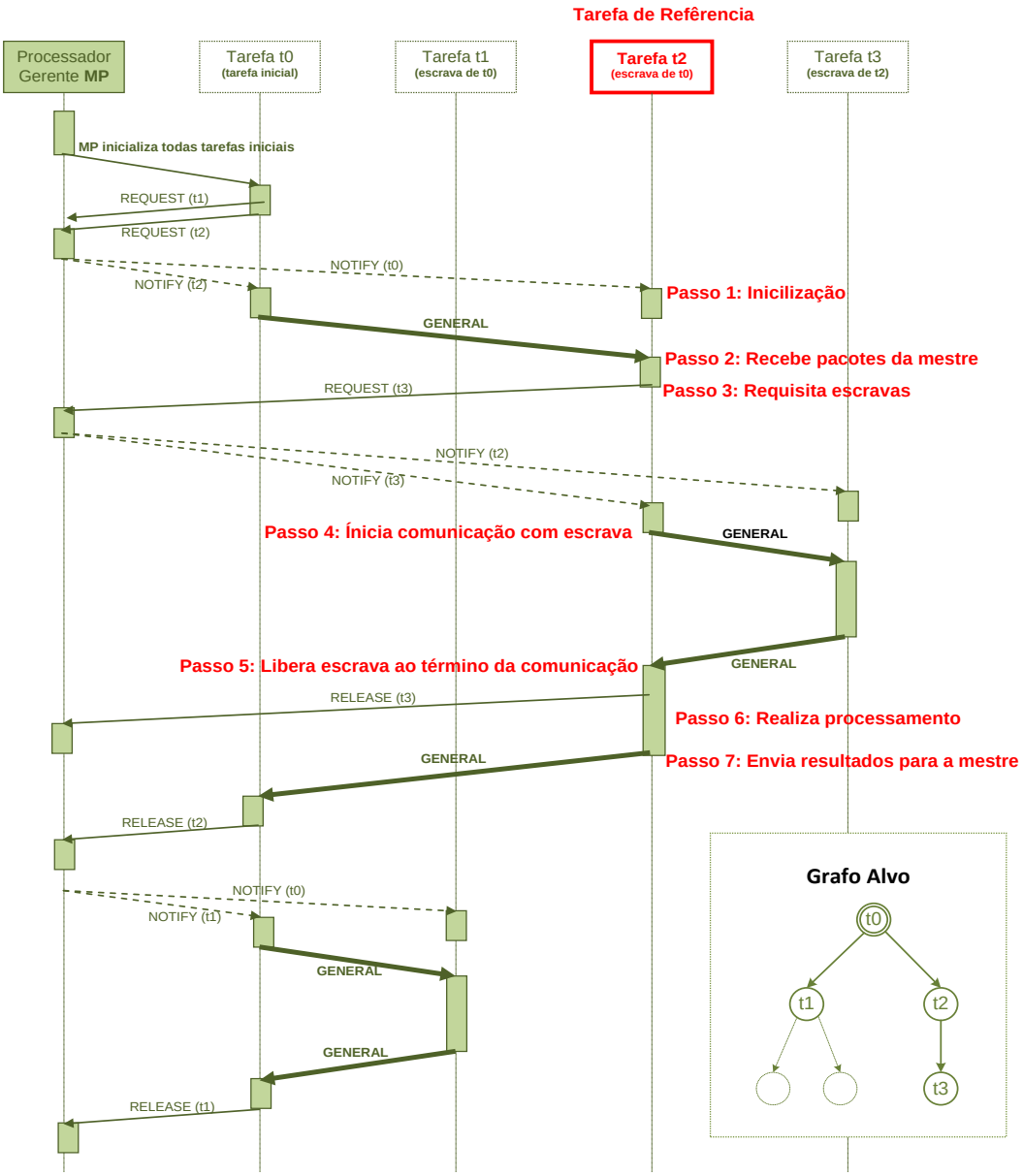


FIGURA 3.5. DIAGRAMA DE INTERAÇÃO ENTRE AS TAREFAS E O PROCESSADOR GERENTE MP.

Embora as comunicações entre *t0* e suas escravas *t1* e *t2* tenham sido apresentadas separadamente na Figura 3.5, elas podem ocorrer simultaneamente, dependendo diretamente de como a tarefa foi programada. Aqui, elas foram apresentadas assim, sem qualquer concorrência, apenas com o objetivo de facilitar o entendimento do comportamento da tarefa de referência *t2*.

3.5. Representação de Recursos do MPSoC

A representação dos recursos inclui os elementos de processamento e os canais da NoC. Alguns trabalhos encontrados na literatura tratam o problema de mapeamento genericamente, sem considerar a infra-estrutura de comunicação [HAN04a] [HAN04b] [AHM04] [BAZ00]. Nesses trabalhos, os PEs são modelados através de matrizes 2D, com a ocupação dos elementos representada por valores booleanos [HAN04b], inteiros [HAN04a], ou inteiros positivos [HAN04b]. O emprego de valores inteiros pode reduzir o tempo de procura de recurso livre, mas demanda mais memória, comparado aos valores booleanos. Ao invés de matrizes, os recursos são modelados através de listas encadeadas em [AHM04], e árvores em [BAZ00]. Em geral, o uso de matrizes tende a apresentar um tempo maior de busca em comparação às listas encadeadas. Contudo, as listas requerem um tempo maior para manutenção devido a sua complexidade.

No presente trabalho, a representação dos recursos, incluindo os elementos de processamento e os canais da NoC, é baseada em um conjunto de cinco matrizes. Adotando-se um MPSoC com *n* colunas e *m* linhas, tem-se então as seguintes matrizes:

Ocupação dos Elementos de Processamento (PE)

$$\begin{bmatrix} PE_{0,m-1} & PE_{1,m-1} & \cdots & PE_{n-1,m-1} \\ \vdots & \vdots & & \vdots \\ PE_{0,1} & PE_{1,1} & \cdots & PE_{n-1,1} \\ PE_{0,0} & PE_{1,0} & \cdots & PE_{n-1,0} \end{bmatrix}$$

Ocupação dos Canais para Sul (SC)

$$\begin{bmatrix} SC_{0,m-2} & SC_{1,m-2} & \cdots & SC_{n-1,m-2} \\ \vdots & \vdots & & \vdots \\ SC_{0,0} & SC_{1,0} & \cdots & SC_{n-1,0} \end{bmatrix}$$

Ocupação dos Canais para Norte (NC)

$$\begin{bmatrix} NC_{0,m-2} & NC_{1,m-2} & \cdots & NC_{n-1,m-2} \\ \vdots & \vdots & & \vdots \\ NC_{0,0} & NC_{1,0} & \cdots & NC_{n-1,0} \end{bmatrix}$$

Ocupação dos Canais para Oeste (WC)

$$\begin{bmatrix} WC_{0,m-1} & \cdots & WC_{n-2,m-1} \\ \vdots & & \vdots \\ WC_{0,1} & \cdots & WC_{n-2,1} \\ WC_{0,0} & \cdots & WC_{n-2,0} \end{bmatrix}$$

Ocupação dos Canais para Leste (LC)

$$\begin{bmatrix} EC_{0,m-1} & \cdots & EC_{n-2,m-1} \\ \vdots & & \vdots \\ EC_{0,1} & \cdots & EC_{n-2,1} \\ EC_{0,0} & \cdots & EC_{n-2,0} \end{bmatrix}$$

Um dado elemento de processamento é representado por *PEij*, onde *i* e *j* referem-se a coluna e linha em que o elemento encontra-se na *matriz PE*, respectivamente, conforme a distribuição apresentada na Figura 3.6. A informação armazenada em *PEij* contém seu tipo (*i. e. inicial*, *hardware* ou *software*) e seu estado (*i. e. livre* ou *ocupado*).

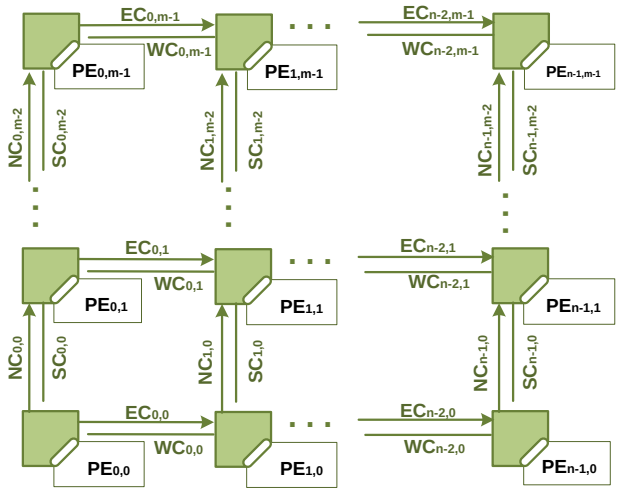


FIGURA 3.6. DISTRIBUIÇÃO ESPACIAL DOS RECURSOS DE UM MPSOC COM *N* COLUNAS E *M* LINHAS.

Os enlaces da NoC são compostos por um par de canais unidirecionais, com sentidos opostos. Os canais da NoC são representados por quatro matrizes: canais para leste (*East Channels - ECij*); canais para oeste (*West Channels - WCij*); canais para norte (*North Channels - NCij*); e canais para sul (*South Channels - SCij*), onde *i* e *j* possuem o mesmo significado usado na representação da matriz PE. Cada elemento das matrizes de canais informa o percentual de ocupação da largura de banda disponível.

3.6. Monitoramento da NoC

Monitoramento é um assunto importante em NoCs, pois permite capturar informações sobre o desempenho do sistema em tempo de execução. Por exemplo, Marescaux

e outros [MAR05c] empregam fios de controle (separados da NoC) para enviar informações sobre congestionamentos ao PE fonte do tráfego, permitindo que este ajuste sua taxa de injeção de pacotes. Enquanto isso, [VAN07a] e [CIO04] empregam a mesma rede para transmissão de pacotes de dados e controle. Em [VAN07a], os Autores usam uma ferramenta denominada *Model Predictive Controller*, a qual obtém informação sobre o estado de congestionamento da rede através de monitores distribuídos sobre a NoC. Ciordas e outros [CIO04] propõem um serviço de monitoramento genérico, com monitores anexos aos roteadores da NoC ou às interfaces de rede.

O processador gerente MP realiza o mapeamento das tarefas de acordo com a ocupação dos PEs e dos canais da NoC. Cada matriz de recurso necessita então ser atualizada em tempo de execução para prover ao MP uma informação precisa sobre a ocupação dos recursos. A estimativa da carga dos canais e de congestionamentos pode basear-se em dois diferentes métodos (Figura 3.7). O primeiro deles, empregado em [CAR07], monitora os pacotes de REQUEST e RELEASE recebidos pelo MP. Esta abordagem requer o uso de um único monitor, anexo à porta local do roteador conectado ao MP. Dessa forma, cada canal tem sua ocupação atualizada a partir das taxas informadas em pacotes REQUEST. De acordo com o algoritmo de roteamento (nesse caso o xy), a ocupação dos canais é devidamente incrementada com as taxas informadas. O processo inverso é realizado quando o monitor captura um pacote RELEASE. Essa estratégia reflete um esquema de *monitoramento centralizado* (lado esquerdo da Figura 3.7), onde o desempenho do mapeamento está diretamente relacionado às taxas estimadas, enviadas ao MP.

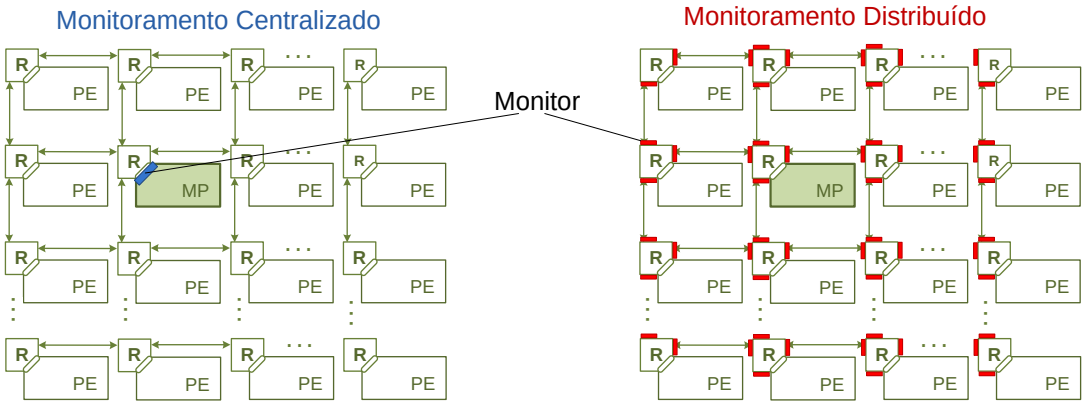


FIGURA 3.7. ESQUEMAS CENTRALIZADO E DISTRIBUÍDO PARA MONITORAMENTO DA OCUPAÇÃO DOS CANAIS DA NOC.

Conforme esboçado na parte direita da Figura 3.7, a segunda abordagem de monitoramento investigada insere um monitor em cada uma das portas dos roteadores da NoC. Este *monitoramento distribuído* permite medir o tráfego real em cada canal da NoC. Assim como em [VAN07a] e [CIO04], aqui também a informação capturada pelos monitores é enviada através da mesma rede de comunicação de dados. Diferentemente da abor-

dagem centralizada, a distribuída é independente do algoritmo de roteamento adotado. Além disso, ela suporta algoritmos não determinísticos. Entretanto, além de consumir área adicional, os monitores distribuídos também aumentam a dissipação de potência.

3.7. Modelagens do Sistema

Durante o desenvolvimento desse trabalho, três abordagens diferentes foram empregadas para modelagem do sistema. São elas baseadas em: (*i*) *POSIX Threads*; (*ii*) *SystemC_threads* no nível TLM; e (*iii*) *System_Cthreads* e VHDL em nível RTL. A seguir cada uma delas é introduzida.

3.7.1. Modelagem Comportamental

A primeira modelagem realizada visou uma implementação simplificada do sistema, a fim de validar a proposta do estudo sobre mapeamento dinâmico de tarefas. Ela é *baseada em Pthreads* (*POSIX Threads* [BUT97]). Cada uma das tarefas é representada por uma dada *thread*. Existem três tipos de tarefas no sistema: *tarefas de software* que executam em processadores, *tarefas de hardware* que executam em áreas reconfiguráveis, e *tarefas fixas* que executam em recursos fixos no sistema (*e. g.* memórias e interfaces de E/S). As últimas, por serem fixas, não necessitam ser mapeadas em tempo de execução.

A NoC, por sua vez, é modelada como uma matriz de *buffers* e quatro matrizes de canais, as quais mantém informação sobre a ocupação da largura de banda. Nenhum atraso é considerado na transmissão, visto que os pacotes não são transferidos através de enlaces, eles são gravados diretamente nos *buffers*. Para suportar a execução de todas as tarefas, os recursos do sistema podem ser de *hardware*, *software* ou *fixos*. Eles são representados por seu tipo e posição, dada pelas coordenadas da matriz de recursos. Cada recurso é associado a um dado *buffer de entrada* através dos seus índices na matriz.

O *processo de mapeamento* refere-se à atribuição de um recurso a uma dada tarefa, a qual pode ler os pacotes endereçados a ela a partir do *buffer* associado a esse recurso. Todas as escrita e leitura nos *buffers* são realizadas com base em um *mutex*, que assegura o acesso individual e atômico. Dois algoritmos de mapeamento foram implementados. O primeiro, denominado *First Free*, posiciona a tarefa na primeira posição livre, de acordo com seu tipo, sem considerar qualquer função custo. O segundo algoritmo emprega uma função custo que avalia o *somatório da ocupação dos canais* da NoC. Os algoritmos de mapeamento implementados são discutidos em maiores detalhes no próximo Capítulo.

Os experimentos validam a técnica de mapeamento baseada na ocupação de canais, bem como a estratégia de *clusterização*. A dificuldade nessa modelagem refere-se às medições de tempo (*i. e.* de mapeamento, de execução das tarefas), impossibilitadas já que o programador não tem acesso a informações de troca de contexto (controlado pelo SO) entre as *threads* executadas. Adicionalmente, as instruções disponíveis para medições de tempo possuem precisão na ordem de *m*segundos apenas, *e. g.* ferramentas tipo *GProf* [FEN08] que permitem obter um relatório detalhado da execução das funções do programa após sua execução. Assim, as medições são imprecisas nessa modelagem.

3.7.2. Modelagem em Nível TLM

A segunda modelagem realizada foi *baseada em SystemC_Threads* [GHE05], portanto em nível transacional TLM, onde o conceito de relógio de operação não é adotado. O objetivo dessa modelagem consiste em uma avaliação mais completa dos possíveis ganhos das heurísticas de mapeamento *congestion-aware*, em sistemas heterogêneos. Para isso, na avaliação, considera-se além da ocupação dos canais da NoC, o tempo total de execução das aplicações simuladas. Usando a biblioteca *SystemC* o programador pode controlar a troca de contexto entre as *SystemC_Threads*.

Os recursos são representados por uma matriz de ocupação. Cada um deles possui um *buffer de entrada* associado. A NoC em si é representada por: (*i*) uma função que estima o atraso dos pacotes; (*ii*) um conjunto de *buffers*; e (*iii*) matrizes que armazenam a ocupação de cada um dos seus canais. Os canais em si não existem. Quando uma tarefa deseja se comunicar com outra, ela grava os pacotes diretamente no *buffer* da tarefa destino da comunicação, assim como na modelagem anterior. Entretanto, o pacote será disponibilizado para a tarefa destino somente após o atraso calculado, tendo por base a ocupação dos canais usados na transmissão e o número de *hop*s necessários. Portanto, essa modelagem permite estimar a degradação no desempenho das aplicações relativa ao nível de congestionamento no sistema.

O processo de mapeamento consiste em atribuir a uma dada tarefa o endereço do *buffer* do qual ele deve ler seus pacotes. A avaliação do sistema compreende a execução de um conjunto de simulações cujos resultados permitem comparações entre diferentes estratégias de mapeamento, discutidas em detalhes no próximo Capítulo. Dentre as funções custo adotadas constam as ocupações máximas e médias dos canais da NoC. Nos experimentos realizados o tempo de mapeamento é considerado.

3.7.3. Modelagem em Nível RTL

A terceira modelagem é *baseada em SystemC_Cthreads e VHDL*, em nível RTL. Ela permite avaliações no nível de ciclo de relógio de operação. Seu objetivo consiste em obter avaliações precisas das heurísticas de mapeamento investigadas. A modelagem RTL é a principal modelagem do trabalho aqui proposto. Por isso, os resultados apresentados bem como os algoritmos implementados são todos discutidos tendo por base tal abordagem. Além disso, os modelos de aplicação e recursos apresentados anteriormente também fazem referência a ela.

Nas modelagens anteriores, cada tarefa do sistema era representada por sua própria *thread*. Na modelagem RTL, uma abordagem diferente é adotada, onde cada PE é modelado por uma *thread*. Eles são emulados por módulos descritos em *SystemC*, conectados as portas locais dos roteadores da NoC descrita em VHDL. Esses módulos reproduzem o comportamento padrão de uma dada tarefa (Figura 3.5). Esse comportamento é parametrizado em tempo de execução por um dado arquivo de configuração, o qual contém informação sobre os volumes e as taxas de comunicação, bem como sobre as topologias das aplicações simuladas. Dessa forma, cada tarefa é descrita por seu arquivo.

As avaliações são mais completas e precisas na modelagem RTL. Dentre os parâmetros de desempenho investigadas, além do tempo de execução e a ocupação dos canais, encontram-se o nível de congestionamento do sistema, as latências dos pacotes, e a energia consumida durante a operação do sistema. Como nessa abordagem a NoC é completamente modelada, é possível avaliar também o emprego de monitores distribuídos, além do monitoramento dos pacotes de REQUEST e RELEASE apenas.

A Tabela 3.1 apresenta um quadro comparativo entra as três modelagens investigadas no presente trabalho.

TABELA 3.1. COMPARATIVO ENTRE AS TRÊS ABORDAGENS ADOTADAS PARA MODELAR O SISTEMA.

Modelagens	*NoC*	*Aplicação*	*Elementos de Processamento*	*Algoritmo de Mapeamento*	*Métricas*	*Tempo de Simulação*
Comportamental *POSIX Thread*	Matriz de buffers (malha) Matriz de ocupação dos Canais Sem enlaces de comunicação Sem atraso nos pacotes	Grafo de tarefas Taxa de Comunicação **Tarefa = Thread**	MPSoC Heterogêneo Matriz de Ocupação Associado a um buffer	FF, PL	Ocupação dos Canais	- MPSoC 8x8 100 Apps
Nível TLM *SC_Thread*	Matriz de buffer (malha) Matriz de ocupação dos canais Sem enlaces de comunicação Estimativa de atraso de pacotes	Grafo de tarefas Taxa de Comunicação **Tarefa = Thread**	MPSoC Heterogêneo Matriz de Ocupação Associado a um buffer	FF, NN, MACL, MMCL, PL	Ocupação dos Canais Tempo Total de Execução	*m*seg MPSoC 8x8 100 Apps
Nível RTL *SC_Cthreads VHDL*	Hermes malha (Roteador + enlace) Matriz de ocupação dos canais Descrição VHDL	Grafo de tarefas Volume de Comunicação Taxa de Comunicação **Tarefa = Arquivo**	MPSoC Heterogêneo Matriz de Ocupação Associado a um Roteador **PE = Thread** Comportamento da tarefa	FF, NN, MACL, MMCL, PL, BN	Ocupação dos Canais Latência dos Pacotes Nível de Congestionamento Tempo Total de Execução Energia Total Consumida	*45 min - 1 h* MPSoC 8x8 20 Apps

4. HEURÍSTICAS PARA MAPEAMENTO DINÂMICO

Como mencionado anteriormente, o mapeamento é a operação que define o posicionamento das tarefas nos recursos do sistema. A posição de uma dada tarefa pode interferir diretamente no desempenho do sistema, afetando, por exemplo, parâmetros como energia consumida, congestionamento da interface interna de comunicação e tempo total de execução das aplicações. No domínio de aplicações que apresentam uma carga dinâmica de tarefas, o uso de estratégias estáticas de mapeamento não é adequado. Nesse caso, estratégias dinâmicas de mapeamento necessitam ser empregadas a partir do constante monitoramento dos recursos do sistema para otimizar o uso dos mesmos e ainda garantir um bom desempenho.

Em geral, no mapeamento estático de tarefas o tempo para definir o mapeamento não é tão importante quando comparado a um cenário onde as tarefas necessitam ser mapeadas em tempo de execução. Em cenários dinâmicos, o tempo de mapeamento ganha importância, visto que exerce influência direta sobre o desempenho do sistema. Nesse caso, o emprego de uma solução força bruta para o problema de mapeamento também não é indicado, pois tal problema é conhecido como um problema de atribuição NP-completo. De fato, até mesmo em tempo de projeto a solução força bruta pode ser proibitiva, visto que o aumento nas dimensões do MPSoC leva a um crescimento exponencial no tempo para computar a solução. Assim, surge a necessidade do estudo de heurísticas para o mapeamento de tarefas, que podem apresentar boas soluções de mapeamento, muitas vezes próximas da melhor solução, e que consomem um tempo reduzido para sua execução mesmo para instâncias maiores do problema.

Conforme a revisão apresentada no Capítulo 2, a maioria das pesquisas em mapeamento de tarefas investiga estratégias estáticas de mapeamento [LEI03a] [HU05] [SRI05] [LIN05] [RUG06] [MUR06b] [MAR07]. Entretanto, o mapeamento dinâmico começa a ser pesquisado [WRO06] [NGO06] [CHO07] [MEH08] [ALF08]. Em sua maioria, os Autores desconsideram o atraso de mapeamento, e investigam o mapeamento dinâmico de tarefas baseado nas mesmas estratégias antes adotadas em tempo de projeto, diferentemente do trabalho aqui proposto, que aplica algoritmos gulosos para o mapeamento de tarefas.

Na operação do MPSoC proposto no Capítulo 3, inicialmente somente as tarefas de controle do MP estão alocadas no sistema. Novas aplicações (*i. e.* tarefas iniciais) são disparadas pelo usuário, e mapeadas pelo MP. As demais tarefas do sistema são alocadas quando uma dada tarefa tenta se comunicar com uma tarefa ainda não alocada no sistema. Ou seja, as tarefas são mapeadas individualmente, sob demanda das aplicações. Mapear todas as tarefas de uma dada aplicação de uma única vez deve apresentar uma melhor solução de mapeamento, visto que tal abordagem vale-se do conhecimento global da aplicação, incluindo a topologia de seu grafo de representação. No entanto, em um cenário dinâmico, não é possível estimar precisamente quando cada tarefa será necessária. Assim, mapeando toda a aplicação, possivelmente, algumas de suas tarefas irão ocupar os recursos do sistema sem no entanto estar executando. Essa abordagem pode resultar em uma subutilização dos recursos do sistema. Ainda, se não existem recursos livres suficientes para as tarefas de uma aplicação, a mesma não será mapeada, problema que não pode ser amenizado se as tarefas de uma dada aplicação são mapeadas independentemente.

Nesse Capítulo discute-se o mapeamento dinâmico de tarefas, organizado como segue. Na Seção 4.1, apresenta-se formalmente o problema de mapeamento, bem como a função custo adotada, que considera a ocupação dos canais da NoC com o objetivo de reduzir possíveis congestionamentos. Na Seção 4.2, apresenta-se a estratégia de *clusterização* adotada para o mapeamento das tarefas iniciais das aplicações. A Seção 4.3 discute dois métodos de mapeamento usados como referências para avaliação das quatro heurísticas aqui propostas, apresentadas na Seção 4.4. No final do Capítulo, a Seção 4.5 discute a implementação de cada algoritmo, assim como sua complexidade.

4.1. Mapeamento de Tarefas

Anteriormente neste documento, os componentes da operação de mapeamento foram definidos informalmente, incluindo as descrições de tarefa, aplicação e arquitetura alvo. Tais definições foram importantes para auxiliar no entendimento dos trabalhos revisados e da proposta em si. Contudo, a definição do problema de mapeamento exige conhecimento aprofundado desses e de outros componentes, como apresentado a seguir.

> Definição 1. ***Tarefa**: Uma tarefa **T** é uma tripla **T = (Tid, Tex, Tti)**, onde **Tid** ∈ ℕ é o identificador da tarefa; **Tex** ∈ ℕ é o tempo de execução da tarefa, e **Tti** ∈ {software, hardware, inicial} é o tipo da tarefa.*

Conforme esta definição, os tipos de tarefa estão relacionados à arquitetura dos elementos de processamento para a qual ela foi compilada. No caso de um MPSoC homogêneo, por exemplo, todas as tarefas são do mesmo tipo porque só existe um tipo de ele-

mento de processamento no qual elas podem executar. No presente trabalho, como o MPSoC é heterogêneo, uma dada tarefa pode ter um entre três tipos, de acordo com a natureza da sua implementação. São eles: *software*, *hardware* e *inicial*. As *tarefas de software* são aquelas que devem executar nos processadores do MPSoC, ao passo que *tarefas de hardware* são configuradas na lógica configurável embarcada. Uma *tarefa inicial* é um caso especial de uma tarefa de software. Como o próprio nome indica, uma tarefa inicial é a primeira a ser executada, quando uma dada aplicação é disparada pelo usuário. No caso de um MPSoC com outros tipos de elementos de processamento, o conjunto de tipos possível deve ser alterado, portanto.

> Definição 2. ***Comunicação**: Uma comunicação **Cms** entre um par de tarefas **m** e **s** é uma quadrupla **Cms = (Vms, Rms, Vsm, Rsm)**, onde **Vms** ∈ ℕ é **volume de dados** enviados pela tarefa **m** para a tarefa **s**, segundo a **taxa de transmissão Rms**, enquanto **Vsm** e **Rsm** possuem os mesmos significados respectivos para o sentido oposto de comunicação (de **s** para **m**). O volume **V** é o número de flits enviados, ao passo que a taxa **R** é o percentual de ocupação da largura de banda disponível.*

> Definição 3. ***Grafo de Aplicação**: Um grafo de aplicação **APG** é um grafo dirigido **APG = (ST, SC)** onde **ST** é conjunto de vértices que representa o **conjunto de tarefas** que compõem a aplicação; e **SC** é o conjunto de arestas que descreve as **comunicações entre tarefas**. As arestas são definidas através de um par ordenado **(s, m)**, onde **s** é a **tarefa fonte** e **m** é a **tarefa destino** da comunicação.*

A topologia do grafo define a interdependência entre as tarefas da aplicação. Cada par de tarefas conectadas é denominado *par de tarefas comunicantes*, onde o sentido da aresta de conexão determina a ordem parcial de alocação das tarefas no sistema. A aresta dirigida indica que sua tarefa fonte necessita requisitar o mapeamento da tarefa destino, antes de iniciar a comunicação propriamente dita. Assim, a tarefa fonte recebe a denominação de *mestre*, ao passo que a tarefa destino é dita *escrava* na comunicação, formando um par de tarefas *mestre-escravo*. No presente trabalho, toda e qualquer aplicação pode conter apenas uma única tarefa inicial. Este pressuposto é adotado em concordância com a estratégia de *clusterização*, explicada em maiores detalhes a seguir, na Seção 4.2.

> Definição 4. ***Elemento de Processamento**: um elemento de processamento **PE** é uma quintupla **PE = (PEid; PEad; PEti; PEuse; ST)**, onde: **PEid** ∈ ℕ é o **identificador** do elemento de processamento; **PEad** ∈ ℕ é o seu **endereço** para envio de pacotes**; PEti** ∈ {software, hardware, inicial} é seu **tipo**; **PEuse** ∈ {ocu-*

pado, livre} é o ***estado de ocupação*** *do elemento de processamento; e* ***ST*** *é conjunto das tarefas nele mapeadas.*

Assim como no caso das tarefas, PEs podem ser classificados em um de três tipos: *software*, *hardware* ou *inicial*. No caso em que processadores do sistema são multitarefa, a ocupação dos recursos deve ser representada por um percentual de uso de memória, ou de capacidade de processamento, por exemplo. Assume-se no presente trabalho que os PEs podem conter apenas uma tarefa. Então, o valor de ocupação indica um entre dois estados possíveis: *ocupado* ou *livre*. Além disso, o conjunto de tarefas *ST* mapeadas no *PE* será composto por uma ou nenhuma tarefa.

Definição 5. ***Canal de Comunicação****: um canal de comunicação* ***C*** *é um conjunto de sinais/fios que transmitem dados e controles entre dois elementos de processamento. Ele é uma dupla* ***C = (Cw, Cuse)****, onde* $\boldsymbol{Cw} \in \mathbb{N}$ *é a* ***largura em bits*** *do canal, incluindo dados e controles; e* ***Cuse*** *é o* ***percentual de uso da largura de banda disponível*** *para transmissão de dados apenas.*

Definição 6. ***Enlace de Comunicação****: um enlace de comunicação* ***L*** *é uma tripla* ***L = (Lpe1, Lpe2, Lsc)****, onde* ***Lpe1*** *e* ***Lpe2*** *são os* ***elementos de processamento conectados*** *pelo enlace, e* ***Lsc*** *é o* ***conjunto de canais Lsc =*** $\boldsymbol{\{C_1, C_2, \ldots, C_n\}}$ *que interconectam Lpe1 e Lpe2, em qualquer sentido possível.*

No presente trabalho, assume-se que cada *enlace de comunicação* possui apenas *um par de canais* de comunicação que permitem a transmissão bidirecional simultânea (do inglês, *fullduplex*) de dados entre os PEs conectados.

Definição 7. ***Grafo do MPSoC****: Um grafo de MPSoC é um grafo* ***GMPSoC = <PE, L>*** *onde* ***PE*** *é o conjunto de vértices representando o conjunto dos* ***elementos de processamento*** *do MPSoC, e* ***L*** *é o conjunto de arestas que representa os* ***enlaces*** *que interconectam os PEs.*

PEs de um dado MPSoC recebem tarefas que são mapeadas para execução. Além de elementos de processamento programáveis/configuráveis que recebem tarefas, PEs podem ser IPs fixos que executam operações específicas, *e. g.* controle de E/S e armazenamento de dados (*i. e.* memórias embarcadas). Contudo, quando uma dada tarefa necessita se comunicar com um desses IPs fixos, a comunicação segue o mesmo protocolo definido no Capítulo anterior. A diferença nesse caso consiste no fato de que a operação de mapeamento não será realizada para IPs fixos.

No presente trabalho, MPSoCs são representados por GMPSoCs de topologia malha, onde cada PE é composto não só pelo elemento de processamento em si, mas também pela lógica que o interconecta aos enlaces de comunicação. Enlaces são dispostos nas direções horizontal e vertical de acordo com os eixos x e y em um sistema de coordenadas cartesianas. Um *roteador* é representado pelo seu algoritmo de roteamento. Aqui o *algoritmo de roteamento xy* é adotado, onde os pacotes são transmitidos entre fonte e destino, primeiro com deslocamentos na direção horizontal x, seguidos de deslocamentos na direção vertical y.

4.1.1. O Problema de Mapeamento

Definição 8. ***Mapeamento**: Seja **T = {t_1, t_2, ..., t_n}** o conjunto de tarefas da aplicaçã, e seja **PE = {pe_1, pe_2, ..., pe_m}** o conjunto de todos os elementos de processamento do **GMPSoC = <PE, L>**. Um **Mapeamento** é uma função injetora f: T → PE, que associa tarefas do domínio **T** à elementos de processamento do contradomínio **PE**. Nesse caso, denota-se por **f(t_x)** o elemento de processamento **pe_y** que a função **f** associa ao elemento **t_x**:*

$$f: t_x \in T \rightarrow pe_y = f(t_x)$$

Nesse trabalho, assume-se que as tarefas são mapeadas individualmente. As entradas da função de mapeamento tratada incluem a *tarefa a ser mapeada*, e a *arquitetura alvo*. A tarefa é descrita pelo seu tipo, suas taxas e volumes de comunicação, e ainda pela posição da sua tarefa mestre. A arquitetura é descrita de acordo com a Definição 7, ou seja, pela sua topologia, tipos e posição de cada PE, e ainda pela ocupação dos canais de comunicação. A saída da função de mapeamento constitui-se de uma dada posição, ou seja, um dado PE, no qual a tarefa deve executar.

O problema de mapeamento consiste em encontrar um mapeamento para a tarefa que ao mesmo tempo respeite as restrições e atenda aos requisitos do sistema. *Restrições* são informações que delimitam o sistema. Exemplos comuns compreendem a largura máxima de transmissão de dados, e o consumo máximo de energia. *Requisitos* são informações desejadas para os sistemas. Exemplos são baixo consumo de energia ou área reduzida. O requisito principal adotado aqui consiste em minimizar os congestionamentos nos enlaces do MPSoC. Com a redução dos congestionamentos, espera-se indiretamente minimizar o tempo de execução das tarefas, e ainda otimizar seu consumo de energia.

4.1.2. Função Custo para Mapeamento

Como em geral mais de um mapeamento é possível no sistema, faz-se necessária a elaboração de funções custo para avaliar a qualidade de um dado mapeamento de acordo com um dado critério. Através das funções custo pode-se escolher qual dentre as possíveis soluções será a escolhida. Funções custo podem considerar mais de uma métrica. Nos presente trabalho, apenas a ocupação dos canais é considerada. Para avaliar ocupação dos canais resultantes de um dado mapeamento, três funções custo são adotadas: (*i*) Ocupação máxima dos canais; (*ii*) Ocupação média dos canais; e (*iii*) Ocupação dos canais do caminho de comunicação.

De acordo com a Definição 7, a partir de um dado *GMPSoC* pode-se obter o conjunto ***L*** de todo os seus enlaces. Cada um desses é composto por um conjunto de canais conforme a Definição 6. Logo, o ***conjunto de todos os canais do MPSoC*** pode ser representado pela união de todos os conjuntos de canais ***Lsc*** referentes a cada um dos enlaces do MPSoC. Cada canal, por sua vez, possui informação sobre sua ocupação, representada por ***Cuse***, o percentual de uso da largura de banda disponível.

MÁXIMO DA OCUPAÇÃO DOS CANAIS

A função custo ***ocupação máxima dos canais*** representada por $\boldsymbol{max(M,t_s,t_m,pe_k)}$ simula o mapeamento da tarefa t_s, solicitada por sua mestre t_m, no elemento de processamento pe_k do MPSoC *M*, e retorna o maior percentual de uso da largura de banda disponível *Cuse* entre todos os canais de *M*. Assim, essa função retorna os picos de ocupação dos canais, que denotam possíveis congestionamentos na comunicação. Se o pico de ocupação apresenta um valor baixo, então pode-se concluir que não existe congestionamento no sistema. À medida que esse valor se aproxima da taxa máxima de transmissão suportada, a probabilidade de acontecerem congestionamentos aumenta, até o pico ultrapassar o limite, configurando-se uma situação de congestionamento.

MÉDIA DA OCUPAÇÃO DOS CANAIS

A função custo ***ocupação média dos canais*** representada por $\boldsymbol{avg(M,t_s,t_m,pe_k)}$, assim como a função *max*, simula o mapeamento da tarefa t_s, solicitada por sua mestre t_m, no elemento de processamento pe_k de *M*. Entretanto, *avg* retorna a média do percentual de uso da largura de banda *Cuse* para todos os canais de *M*. Assim, médias baixas podem ser obtidas ou quando poucos canais são usados, ou ainda se os canais usados são pouco ocupados. Na primeira situação onde poucos canais são usados, uma média baixa não significa que não existam congestionamentos no sistema, visto que os valores altos de ocupa-

ção serão diluídos pela operação de cálculo da média. No caso onde a média de ocupação é alta, então pode-se deduzir que existe um ou mais congestionamentos no sistema. A função média visa, sobretudo, uma distribuição uniforme na ocupação dos canais.

SOMATÓRIO DA OCUPAÇÃO DOS CANAIS DO CAMINHO DE COMUNICAÇÃO

A terceira função custo adotada é a ***ocupação dos canais do caminho de comunicação***. Representada por *sum(M,t_s,t_m,pe_k)*, ela simula o mapeamento da tarefa t_s, solicitada por sua mestre t_m, no elemento de processamento pe_k do MPSoC *M*. Como resultado, a função retorna o somatório dos percentuais de uso da largura de banda *Cuse*, considerando apenas os canais que fazem parte do caminho de comunicação entre as tarefas t_s e t_m, denotado por *CP(t_s,t_m)*. Tal caminho pode ser definido como segue.

> Definição 9. ***Segmento de Caminho de Comunicação**: Um segmento de caminho é um **conjunto de canais PS = {C_1, C_2, ... , C_n}** que possuem a **mesma direção** e o **mesmo sentido,** utilizados na comunicação entre duas tarefas **fonte** e **destino,** conforme o **algoritmo de roteamento**. Existem segmentos de caminho para cada direção: Leste (PS_{EC}), Oeste (PS_{WC}), Norte (PS_{NC}) e Sul (PS_{SC}).*

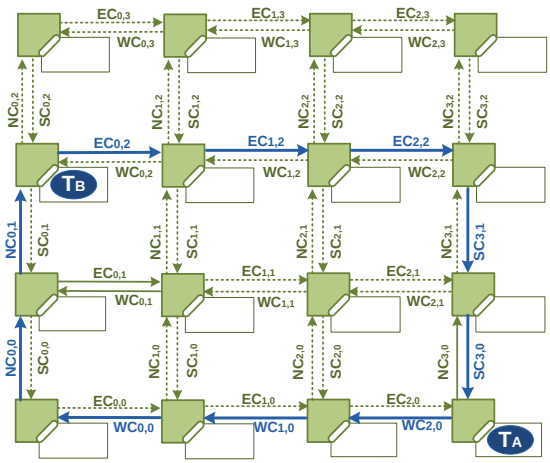
> Definição 10. ***Caminho de Comunicação**: Um dado caminho de comunicação é um **conjunto de quatro segmentos de caminhos** CP = {PS_{EC}, PS_{WC}, PS_{NC}, PS_{SC}}, um em cada direção, Leste, Oeste, Norte e Sul.*

Nesse trabalho, todas as comunicações entre as tarefas se dão através de um dado caminho de comunicação. Ele é formado por quatro segmentos de caminho, cada um em sua direção. Os canais pertencentes a cada segmento são determinados de acordo com a posição das tarefas comunicantes e com o algoritmo de roteamento xy adotado. Assim, caso as tarefas sejam alocadas na mesma coluna de recursos então os segmentos de caminho na direção horizontal serão conjuntos vazios. O mesmo acontece para os segmentos verticais se as tarefas estiverem alocadas na mesma linha de PEs.

Dadas duas tarefas, T_A e T_B, a Equação 4.1 representa o caminho de comunicação *CP(T_A,T_B)* entre tais tarefas, onde PS_{EC} (T_A,T_B) é o segmento de caminho composto pelos canais da matriz de canais para o Leste (Figura 3.6). PS_{WC}(T_A,T_B), PS_{NC} (T_A,T_B) e PS_{SC}(T_A,T_B) possuem significado similar. Ainda é importante notar que um dado caminho de comunicação contempla ambos os sentidos de transmissão.

EQUAÇÃO 4.1. $$CP(T_A, T_B) = PS_{EC}(T_A, T_B) \cup PS_{WC}(T_A, T_B) \cup PS_{NC}(T_A, T_B) \cup PS_{SC}(T_A, T_B)$$

A Figura 4.1 apresenta um exemplo de definição de um caminho de comunicação entre duas tarefas T_A e T_B, de acordo com o algoritmo de roteamento xy.



$PS_{EC}(T_A,T_B) = \{EC_{0,2}; EC_{1,2}; EC_{2,2}\}$

$PS_{WC}(T_A,T_B) = \{WC_{2,0}; WC_{1,0}; WC_{0,0}\}$

$PS_{NC}(T_A,T_B) = \{NC_{0,0}; NC_{0,1}\}$

$PS_{SC}(T_A,T_B) = \{SC_{3,1}; NS_{3,0}\}$

$CP(T_A,T_B) = \{EC_{0,2}; EC_{1,2}; EC_{2,2}; WC_{2,0}; WC_{1,0}; WC_{0,0}; NC_{0,0}; NC_{0,1}; SC_{3,1}; NS_{3,0}\}$

FIGURA 4.1. CAMINHO DE COMUNICAÇÃO $CP(T_A,T_B)$ ENTRE AS TAREFAS T_A E T_B, CONFORME ROTEAMENTO XY.

4.2. Mapeamento de Tarefas Iniciais das Aplicações

A estratégia empregada para definir o mapeamento de tarefas iniciais das aplicações apresenta um grande impacto no desempenho do mapeamento da aplicação como um todo. Duas estratégias podem ser empregadas nesse caso. A primeira delas realiza o mapeamento de tarefas iniciais sem qualquer critério. Como tarefas iniciais de aplicações diferentes podem ser mapeadas perto umas das outras, o mapeamento das demais tarefas de uma dada aplicação pode ser prejudicado com relação ao distanciamento entre as tarefas comunicantes. Os recursos ao redor das tarefas iniciais estarão possivelmente todos ocupados, e as aplicações estarão inteiramente misturadas e distribuídas sobre o MPSoC, causando assim uma utilização não otimizada dos canais e, possivelmente congestionamento na NoC.

Uma segunda estratégia, adotada no presente trabalho, define posições fixas, e distantes umas das outras para o mapeamento de tarefas iniciais. Assim, no mapeamento das aplicações, cada uma delas deve ocupar uma região diferente do MPSoC, reduzindo o número de canais compartilhados por comunicações de aplicações diferentes. Essa estratégia recebe aqui o nome de *clusterização*, pois simula a divisão do MPSoC em *clusters*. As aplicações podem ocupar recursos além daqueles disponibilizados por um dado *cluster*, visto que seus limites são virtuais apenas. Ou seja, o limite para mapear uma aplicação completa é função apenas da demanda da aplicação e da disponibilidade de recursos no MPSoC. Entretanto, o número de aplicações simultâneas é limitado pelo número de recursos dedicados a receber tarefas iniciais das aplicações.

Na Figura 4.2, apresenta-se um exemplo MPSoC 6x6 particionado em 4 *clusters*. Os PEs em destaque são aqueles reservados para o mapeamento das tarefas iniciais das aplicações. Eles são posicionados preferencialmente no centro do *cluster* gerado pelo particionamento. Assim deve-se reduzir a sobreposição entre as tarefas de aplicações diferentes. No caso deste MPSoC, é permitida a execução simultânea de 4 aplicações.

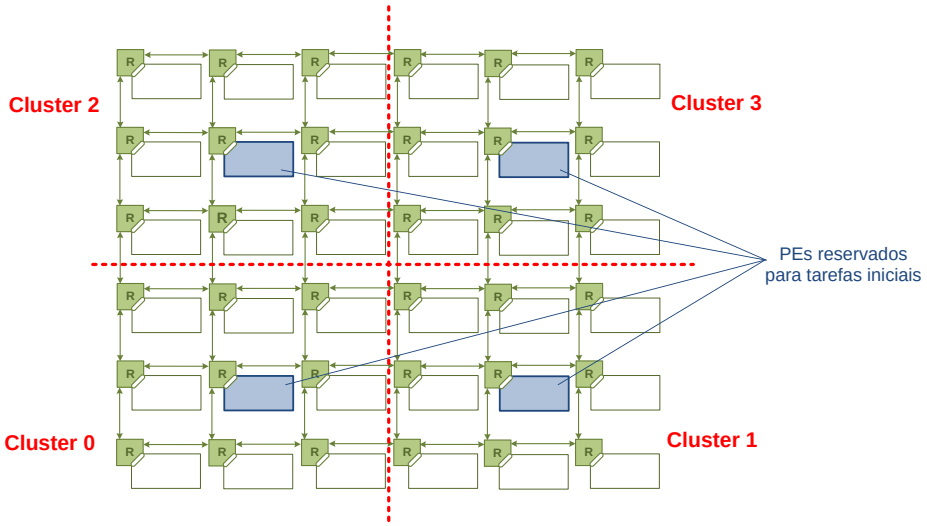
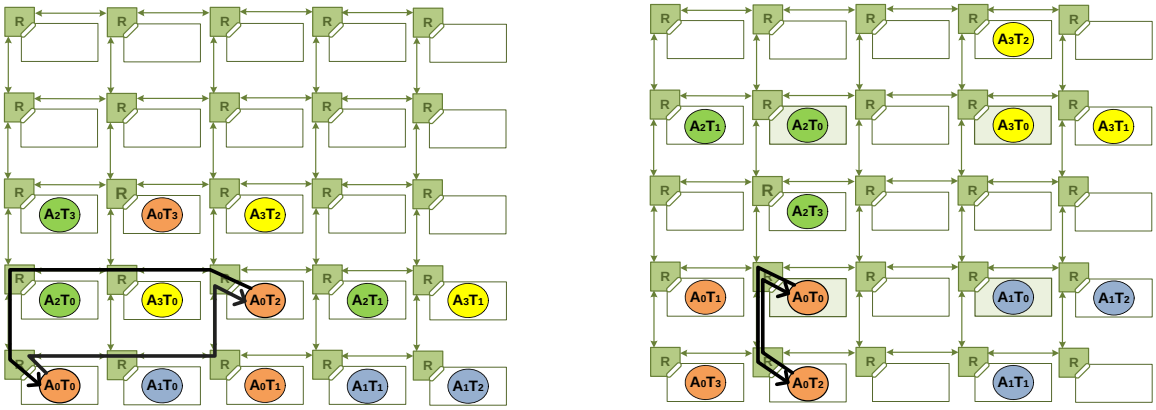


FIGURA 4.2. ESTRATÉGIA DE *CLUSTERIZAÇÃO* PARA O MAPEAMENTO DE TAREFAS INICIAIS DAS APLICAÇÕES.

A Figura 4.3 apresenta um exemplo do emprego das duas estratégias para o mapeamento de tarefas iniciais das aplicações, e seus efeitos sobre uma dada comunicação. As tarefas são representadas por círculos, onde *AiTj* refere-se a tarefa *j* da aplicação *i*. O algoritmo de roteamento xy é adotado na definição das rotas dos pacotes que compõem as comunicações. Em (a), as tarefas iniciais de quatro aplicações (*i. e.* A_0T_0, A_1T_0, A_2T_0 e A_3T_0) foram posicionadas como vizinhas. Em virtude da ordem do mapeamento e do mapeamento inicial empregado (*i. e.* sem qualquer critério), a comunicação entre as tarefas A_0T_0 e A_0T_2 deve ocupar seis canais da NoC.



(a) Sem *clustering* (b) Baseado em *clustering*

FIGURA 4.3. COMPARATIVO DE DUAS ESTRATÉGIAS PARA O MAPEAMENTO DE TAREFAS INICIAS.

Na mesma Figura, em (b), quando as tarefas iniciais estão posicionadas distantes de acordo com a estratégia *clusterização*, apenas dois canais da NoC são necessários para a comunicação entre as mesmas tarefas.

4.3. Métodos de Referência para Mapeamento Dinâmico

Dois métodos de mapeamento de tarefas que não possuem função custo de congestionamento são empregados no presente trabalho, são eles: *First Free* e *Nearest Neighbor*.

4.3.1. First Free – FF

O algoritmo de mapeamento de tarefas *First Free* (FF) seleciona o primeiro recurso livre capaz de suportar a execução da tarefa a ser mapeada, de acordo com os tipos de recursos e tarefas, sem considerar métricas de desempenho. A procura pelo alvo inicia pelo recurso R_{00}, no canto inferior esquerdo do MPSoC, e caminha pelos recursos coluna a coluna, sempre no sentido ascendente (veja Figura 4.4). O algoritmo FF deve apresentar resultados ruins de mapeamento com relação à ocupação dos canais. Entretanto, seu tempo de execução deve ser muito pequeno em comparação as demais heurísticas, apresentadas adiante. O resultado de FF não necessariamente representa o pior mapeamento possível, visto que mapeamentos realizados de maneira aleatória, como os usados nas comparações realizadas em [SHI04] [ORS07], devem apresentar soluções ainda piores. No entanto, o mapeamento aleatório não é considerado aqui.

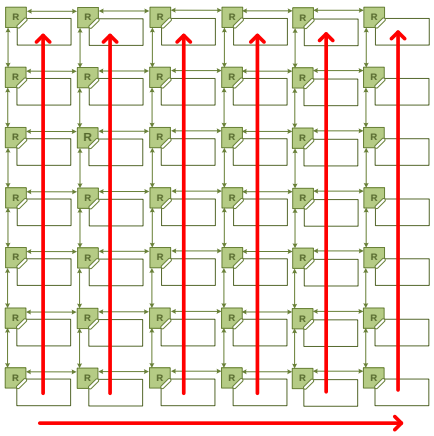


FIGURA 4.4. CAMINHO DA PROCURA POR RECURSO LIVRE REALIZADO POR FIRST FREE.

4.3.2. Nearest Neighbor – NN

Assim como FF, o algoritmo *Nearest Neighbor* (NN) também não considera os congestionamentos quando decide o mapeamento de uma dada tarefa. Contudo, a definição

do seu caminho de procura privilegia a proximidade entre as tarefas comunicantes, durante o processo de mapeamento. Para isso, conforme a Figura 4.5, NN inicia sua procura por um recurso livre, a partir do recurso onde a tarefa que solicitou (*i. e.* mestre) o mapeamento da escrava encontra-se alocada. A procura segue um caminho circular, onde os vizinhos são testados de acordo com o número de saltos (*hop*s) necessários para a comunicação. Ou seja, testa-se os vizinhos com 1 *hop* de distância, depois os vizinhos com 2 *hop*s, e assim por diante, variando o número de *hop*s (*NH* na Figura 4.5) até os limites do MPSoC. A procura termina assim que o primeiro vizinho livre, capaz de suportar o tipo da tarefa a ser executada, for encontrado.

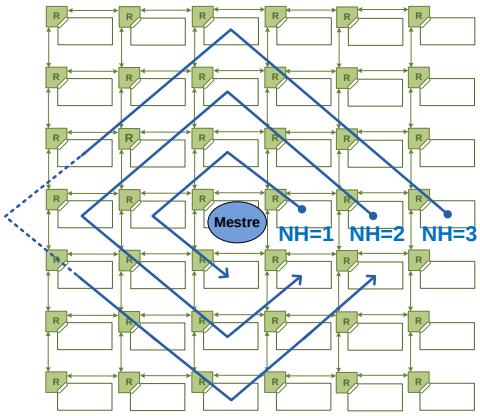


FIGURA 4.5. CAMINHO DA PROCURA POR RECURSO LIVRE REALIZADO POR *NEAREST NEIGHBOR*.

Esse algoritmo apresenta o mesmo objetivo do mapeamento de *Força Direcionada* apresentado em [NGO06], ou seja, posicionar as tarefas comunicantes tão próximas quanto possível. Estima-se que esta estratégia resulte em uma boa ocupação dos canais, mesmo que nenhuma métrica direta de congestionamento seja empregada como função custo. Além disso, o tempo de procura deve ser baixo, a exemplo do tempo de mapeamento apresentado por *First Free*. O caminhamento apresentado na Figura 4.5 assemelha-se ao apresentado pelo algoritmo *Spiral* proposto em [MEH07] [MEH08]. A diferença esta na origem, que aqui não consiste em uma posição fixa como no trabalho citado, onde uma única aplicação é mapeada por completo. Ambas as estratégias foram propostas ao mesmo tempo, no entanto sem qualquer interação entre os grupos de pesquisas.

4.4. Heurísticas para Mapeamento Dinâmico

Nessa Seção, apresenta-se quatro heurísticas para o mapeamento dinâmico de tarefas. Elas são ditas *congestion-aware* visto que visam reduzir os congestionamentos na infra-estrutura de comunicação. Para isso, avaliam constantemente a ocupação dos canais da NoC durante o processo de mapeamento.

4.4.1. Minimum Maximum Channel Load – MMCL

O primeiro algoritmo de mapeamento *congestion-aware* aqui proposto denomina-se *Minimum Maximum Channel Load* (MMCL). Esse algoritmo avalia todos os possíveis mapeamentos para cada um das tarefas a serem mapeadas no sistema. Sua função custo baseia-se na ocupação máxima dos canais da NoC. Dentre os possíveis mapeamentos, MMCL irá selecionar aquele que apresentar o *mínimo máximo de ocupação dos canais*. Dessa forma, estima-se que a abordagem adotada por MMCL possa reduzir a ocorrência de picos de ocupações nos canais, os quais indicam possíveis congestionamentos.

As equações a seguir consideram um MPSoC com dimensões lx e ly, onde lx corresponde ao seu número de colunas e ly ao número de linhas. Para cada mapeamento k, as taxas de comunicação entre as tarefas mestre (solicitante) e escrava (solicitada), Rms e Rsm (para ambos os sentidos de comunicação) são adicionadas à ocupação dos canais que pertence ao caminho de comunicação (Figura 4.1). Após a avaliação da nova ocupação dos canais, de acordo com as Equações 4.2 à 4.5, a ocupação máxima de cada conjunto de canais é calculada, dadas as matrizes de canais. Na Equação 4.2, $MaxR_{EC}(k)$ representa o *máximo local* obtido a partir da ocupação R_{EC} dos canais que pertencem a matriz de canais para Leste. $MaxR_{WC}(k)$, $MaxR_{NC}(k)$ e $MaxR_{SC}(k)$ seguem o mesmo raciocínio para os sentidos, Oeste, Norte e Sul, respectivamente.

Equação 4.2. $$MaxR_{EC}(k) = \max\left(R_{EC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \le i < lx-1;\ \ 0 \le j < ly$$

Equação 4.3. $$MaxR_{WC}(k) = max\left(R_{WC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \le i < lx-1;\ \ 0 \le j < ly$$

Equação 4.4. $$MaxR_{NC}(k) = \max\left(R_{NC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \le i < lx;\ \ 0 \le j < ly-1$$

Equação 4.5. $$MaxR_{SC}(k) = \max\left(R_{SC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \le i < lx;\ \ 0 \le j < ly-1$$

A Equação 4.6 permite calcular o *máximo global* a partir dos máximos locais anteriormente computados, para cada mapeamento k, resultando em um custo $C_{MMCL}(k)$.

Equação 4.6. $$C_{MMCL}(k) = \max\big(MaxR_{EC}(k),\ MaxR_{WC}(k),\ MaxR_{NC}(k),\ MaxR_{SC}(k)\big)$$

Finalmente, dentre os kP *possíveis mapeamentos*, o selecionado segundo o algoritmo MMCL será aquele com custo menor, ou seja, com o mínimo máximo computado, conforme a Equação 4.7.

Equação 4.7. $$Selected_{MMCL}(k) = \min\big(C_{MMCL}(k)\big)\ \forall\, k,\ \ 0 \le k < kP$$

4.4.2. Minimum Average Channel Load – MACL

A segunda heurística *congestion-aware* proposta para o mapeamento dinâmico de tarefas, denominada *Minimum Average Channel Load* (MACL), visa reduzir a ocupação média dos canais da NoC. MACL é muito semelhante ao algoritmo MMCL. Por substituir a função *máximo* pela função *média*, MACL deve reduzir não apenas picos de ocupação, bem como deve distribuir de forma mais homogênea a ocupação dos canais da NoC.

O algoritmo MACL também avalia o posicionamento da tarefa a ser mapeada em cada posição k dentre as kP possíveis posições. Para cada matriz de canais, a *ocupação média local* $AvgR_{EC}(k)$ é calculada, conforme as Equações 4.8 à 4.11.

Equação 4.8. $$AvgR_{EC}(k) = \text{avg}\left(R_{EC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \leq i < lx-1;\ \ 0 \leq j < ly$$

Equação 4.9. $$AvgR_{WC}(k) = \text{avg}\left(R_{WC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \leq i < lx-1;\ \ 0 \leq j < ly$$

Equação 4.10. $$AvgR_{NC}(k) = \text{avg}\left(R_{NC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \leq i < lx;\ \ 0 \leq j < ly-1$$

Equação 4.11. $$AvgR_{SC}(k) = \text{avg}\left(R_{SC(i,j)}(k)\right) \quad \forall\,(i,j), 0 \leq i < lx;\ \ 0 \leq j < ly-1$$

A partir dos valores para ocupação média local, o custo $C_{MACL}(k)$ do mapeamento k que representa sua *ocupação media global* pode ser obtida utilizando a Equação 4.12.

Equação 4.12. $$C_{MACL}(k) = \text{avg}\big(AvgR_{EC}(k),\ AvgR_{WC}(k),\ AvgR_{NC}(k),\ AvgR_{SC}(k)\big)$$

Conforme a Equação 4.13, o mapeamento selecionado será aquele que apresentar o custo mínimo, ou a mínima média entre todos os kP possíveis mapeamentos.

Equação 4.13. $$Selected_{MACL}(k) = \min\big(C_{MACL}(k)\big)\ \forall\, k,\ 0 \leq k < kP$$

4.4.3. Path Load – PL

Além do tempo elevado para o mapeamento, as heurísticas exaustivas MMCL e MACL apresentam ainda outros possíveis problemas. Em MMCL, ao computar cada um dos possíveis mapeamentos, a adição das taxas aos canais da NoC pode não representar um acréscimo ao máximo calculado. Assim, para todos os casos o custo C_{MMCL} calculado pode ser o mesmo, e a posição selecionada pode não representar a melhor solução.

Conforme a Figura 4.6, em (a) apresenta-se a ocupação dos canais da NoC no instante em que a tarefa T_A requisita ao MP a tarefa T_B, com taxas de transmissão *ms* e *mr* iguais a 30 e 33. MMCL deve encontrar a melhor posição dentre o dois nodos livres. Em ambas as alternativas (b) e (c), os custos C_{MMCL} calculados resultam em 90% porque em nenhum dos casos a adição das taxas de comunicação entre T_A e T_B interferem no canal que possui a máxima ocupação. Nesse caso a escolha pode ser aleatória, e a alternativa (c) pode ser selecionada mesmo causando outro pico de ocupação dos canais (*i. e.* 89%).

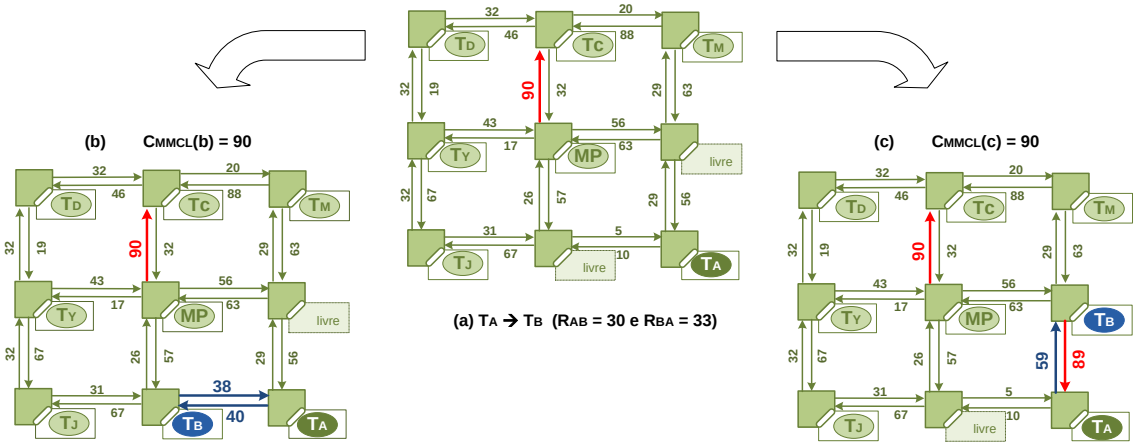


FIGURA 4.6. SITUAÇÃO DE FALHA NA HEURÍSTICA DE MAPEAMENTO MMCL.

A heurística MACL também pode ser comprometida, pois o uso da média como métrica pode não apresenta a melhor solução para mapeamento. Isso ocorre porque a opção que utiliza poucos canais muito ocupados pode resultar em uma média menor quando comparada a opções que usam muitos canais pouco ocupados. Entretanto, futuramente no sistema, a escolha da última alternativa deve acarretar em maiores congestionamentos a medida que as comunicações entre as tarefas começam a compartilhar uma quantia maior de canais.

Conforme a Figura 4.7, em (a) apresenta-se a ocupação dos canais da NoC no instante em que a tarefa T_A requisita ao MP a tarefa T_B. MACL deve encontrar a melhor posição dentre as duas possíveis (b) e (c). A alternativa (a) deve ser escolhida porque apresenta a menor média C_{MACL}, entretanto ela deve causar congestionamentos porque um dos canais tem ocupação 131%. Enquanto isso, a alternativa (b) embora resulte em uma média maior, apresenta uma distribuição mais justa quanto à ocupação dos canais.

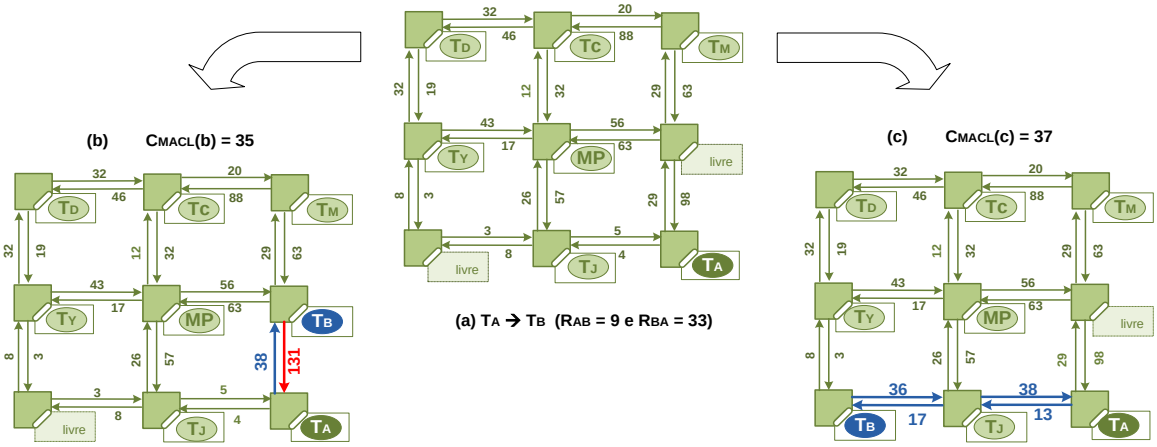


FIGURA 4.7. SITUAÇÃO DE FALHA NA HEURÍSTICA DE MAPEAMENTO MACL.

Como o tempo necessário para realizar o mapeamento pode ser proibitivo nas heurísticas que consideram todos os canais da NoC, tais como MMCL e MACL, o algoritmo *Path Load* (PL) é proposto. Ele considera somente os canais que serão usados pela tarefa que está sendo mapeada, ou seja, aqueles que compõem seu caminho de comunicação. Contudo ainda todos os possíveis mapeamentos são avaliados.

O mapeamento segundo o método PL computa a soma local $SumR_{EC}(k)$ das ocupações dos canais para cada uma das matrizes, de acordo com as Equações 4.14 à 4.17. Entretanto, tais somas consideram apenas os canais que compõem o *caminho de comunicação* (Definição 10) entre as tarefas mestre e escrava.

EQUAÇÃO 4.14. $$SumR_{EC}(k) = \sum\left(R_{EC(i,j)}(k)\right) \quad \forall\,(i,j) \in PS_{EC}, 0 \le i < lx-1;\ \ 0 \le j < ly$$

EQUAÇÃO 4.15. $$SumR_{WC}(k) = \sum\left(R_{WC(i,j)}(k)\right) \quad \forall\,(i,j) \in PS_{WC}, 0 \le i < lx-1;\ \ 0 \le j < ly$$

EQUAÇÃO 4.16. $$SumR_{NC}(k) = \sum\left(R_{NC(i,j)}(k)\right) \quad \forall\,(i,j) \in PS_{NC}, 0 \le i < lx;\ \ 0 \le j < ly-1$$

EQUAÇÃO 4.17. $$SumR_{SC}(k) = \sum\left(R_{SC(i,j)}(k)\right) \quad \forall\,(i,j) \in PS_{SC}, 0 \le i < lx;\ \ 0 \le j < ly-1$$

A partir das somas locais, o custo $C_{PL}(k)$ que representa a soma global é calculado através da Equação 4.18, para um dado mapeamento k.

EQUAÇÃO 4.18. $$C_{PL}(k) = SumR_{EC}(k) + SumR_{WC}(k) + SumR_{NC}(k) + SumR_{SC}(k)$$

Finalmente, o mapeamento selecionado é aquele que apresenta a menor soma global, conforme a Equação 4.19.

EQUAÇÃO 4.19. $Selected_{PL}(k) = \min\left(C_{PL}(k)\right)\ \forall\, k,\ 0 \leq k < kP$

Apresentado o método PL, pode-se retornar àquelas situações onde as heurísticas MMCL e MACL apresentam falhas. Na primeira delas, ilustrada na Figura 4.6, o custo C_{PL} computado para alternativa (b) e (c) são 78 e 148, respectivamente. Dessa forma, na heurística PL, a alternativa (b) será escolhida, diferentemente da heurística MMCL onde essa escolha poderia ser aleatória já que os dois casos apresentaram custos C_{MMCL} idênticos. Em seguida, no caso apresentado na Figura 4.7, os novos custos calculados de acordo coma heurística PL são $C_{PL}(b)$=169 e $C_{PL}(c)$=104. Logo, PL seleciona a alternativa (c), que não causaria congestionamento, ao contrário do resultado proposto quando a heurística MACL foi empregada.

4.4.4. Best Neighbor – BN

Como mencionado anteriormente, o algoritmo *Path Load* reduz o tempo de mapeamento por restringir a amplitude de visão global dos canais da NoC para uma visão apenas parcial, focada no caminho de comunicação. Entretanto PL ainda avalia todos os possíveis mapeamentos. Com o objetivo de reduzir ainda mais o tempo de mapeamento, a heurística *congestion-aware Best Neighbor* (BN) é proposta. Ela combina as estratégias *Nearest Neighbor* e *Path Load*. O caminho de procura de BN é idêntico ao adotado em NN (veja Figura 4.5), entretanto não mais o primeiro vizinho livre é selecionado como em NN. BN seleciona-se o melhor vizinho. Caso haja mais de um vizinho disponível no mesmo anel de verificação, ou seja, com o mesmo número de *hop*s de distância da tarefa solicitante, então todos serão avaliados e o melhor será eleito, segundo as Equações 14 à 19, utilizadas no algoritmo *Path Load*. Assim, estima-se que BN apresente resultados próximos aos encontrados por PL, mas em um tempo menor.

4.5. Descrição dos Algoritmos Implementados

Nessa Seção, discute-se os algoritmos de mapeamento propostos. Contudo, antes da execução do mapeamento em si, duas operações importantes são realizadas no sistema implementado. A primeira delas testa todos os elementos de processamento para verificar se a tarefa solicitada já não se encontra alocada no sistema. Em caso positivo, ela pode ser reutilizada, ou seja, torna-se desnecessário o passo de mapeamento e apenas seu endereço é enviado à tarefa mestre. Além disso, o reuso evita que mais de uma instância de uma dada tarefa seja alocada no sistema. Entretanto, se for interessante que várias instâncias executem em paralelo, então o programador pode replicar a tarefa e atribuir a elas identificadores diferentes. No Algoritmo 4.1, apresentado abaixo, o bloco ENQUANTO na linha 4 realiza o teste para reuso.

Adicionalmente, antes de mapear a tarefa faz-se necessário verificar se existem recursos disponíveis. Em caso positivo, o mapeamento pode seguir normalmente. Do contrário, a tarefa necessita ser escalonada. Conforme mencionado anteriormente, a estratégia de escalonamento implementada baseia-se em um conjunto de três filas, uma para cada tipo de tarefa, sem política de preempção. No Algoritmo 4.1, o bloco SE na linha 10 verifica a disponibilidade de recursos e escalona da tarefa se necessário.

ALGORITMO 4.1. TESTES REALIZADOS PARA REUSO E ESCALONAMENTO DE TAREFAS.

```
1  PE = 0  // INICIA PELO PE 0
2
3  // TENTA REUSAR A TAREFA
4  ENQUANTO (PE < NumeroDePEs) FAÇA
5     SE (PE.tarefaID == tarefa) ENTÃO Retorna(PE)
6     PE=PE+1
7  FIM ENQUANTO
8
9  // SE NÃO TEM RECURSO DESSE TIPO LIVRE ENTÃO ESCALONA
10 SE (NumeroDeRecursosLivres.Tipo(tarefa) == 0) ENTÃO
11    Escalona(tarefa)
12 FIM SE
```

Em contraproposta às estratégias que mapeiam simultaneamente todas as tarefas de uma dada aplicação, as heurísticas aqui apresentadas são algoritmos do tipo guloso [COR01], os quais realizam o mapeamento de uma única tarefa por vez. Assim pode-se obter soluções de mapeamento próximas à melhor solução em um tempo reduzido. De outro lado, a alternativa que mapeia aplicações por inteiro baseia-se em uma estratégia combinatória, a qual avalia todos os possíveis mapeamentos para todas as tarefas. Em geral, os resultados providos por essa abordagem são melhores, ao custo de um maior tempo de execução.

Na análise da complexidade dos algoritmos implementados admite-se um MPSoC com topologia malha, e com dimensões x e y. Além disso, outros parâmetros devem ser conhecidos para um melhor entendimento da complexidade dos algoritmos, visto que influenciam no número de iterações realizadas nos algoritmos discutidos a seguir. O primeiro deles corresponde ao *número total de elementos de processamento* que compõem um MPSoC, dado pelo produto apresentado na Equação 4.20.

EQUAÇÃO 4.20. $$NumPEs(x, y) = xy$$

O segundo parâmetro diz respeito ao *número total de canais* que compõem o MPSoC. Dado um MPSoC malha com dimensões x e y, o número total de canais pode ser obtido a partir da Equação 4.21.

EQUAÇÃO 4.21. $$NumCanais(x, y) = 4xy - 2x - 2y$$

O terceiro parâmetro para o cálculo da complexidade dos algoritmos é a distância máxima entre duas tarefas. De acordo com o algoritmo de roteamento xy adotado, pode-se inferir que o *número máximo de hops* entre duas tarefas é dado pela Equação 4.22, que refere-se ao caso onde as tarefas comunicantes estão posicionadas nos recursos extremos opostos do MPSoC. Como exemplo, pode-se citar a comunicação entre duas tarefas alocadas nos elementos $PE_{0,0}$ (*i. e.* canto inferior esquerdo) e $PE_{x-1,y-1}$ (*i. e.* canto superior direito).

EQUAÇÃO 4.22. $$maxDistancia(x, y) = x + y + 2$$

O quarto parâmetro empregado diz respeito ao *número máximo de vizinhos com a mesma distância de uma dada tarefa*. Dado um MPSoC com dimensões x e y, o número máximo de vizinhos pode ser obtido a partir da Equação 4.23. O resultado desta equação trata-se de um valor aproximado, onde o erro é de dois vizinhos a mais ou a menos. Contudo, no caso onde $x = y$, o valor resultante expressa o valor exato.

EQUAÇÃO 4.23. $$maxNumVizinhos(x, y) \approx x + y - 2$$

O método mais simples implementado para o mapeamento de tarefas é apresentado no Algoritmo 4.2. Para realizar o mapeamento de acordo com o método *First Free*, é necessário percorrer os elementos de processamento, e tão logo um recurso livre e compatível (*i. e.* do mesmo tipo da tarefa) seja encontrado, ele deve ser retornado. Considerando o Algoritmo 4.2, pode-se observar que o laço ENQUANTO da linha 4 é, no pior caso, executado para todos os PEs do MPSoC. Generalizando para um MPSoC com dimensões idênticas (*i. e.* $x = y$), então x^2 iterações serão realizadas conforme representado abaixo. Tal fato confere complexidade quadrática $O(x^2)$ ao algoritmo FF.

$$\begin{aligned} maxNumIteracoes(FF, x, y) &= NumPEs(x, y) \\ &= xy \\ &= x^2 \quad para \;\; x = y \quad \rightarrow \quad O(x^2) \end{aligned}$$

ALGORITMO 4.2. MAPEAMENTO SEGUNDO O MÉTODO FF DE REFERÊNCIA.

```
1  PE=0 // INICIA PELO PE 0
2
3  // ENQUANTO O PE FOR VÁLIDO E ATÉ ENCONTRAR UMA POSIÇÃO PARA AS TAREFAS
4  ENQUANTO (PE < NumeroDePEs) FAÇA
5
6     // SELECIONA PRIMEIRO RECURSO LIVRE COM TIPO COMPATÍVEL AO DA TAREFA A SER MAPEADA
7     SE (PE.estado == LIVRE) E (PE.tipo == tarefa.tipo) ENTÃO Retorna(PE)
8
9     PE =PE+1 // PROXIMO PE
10
11 FIM ENQUANTO
```

A segunda estratégia para mapeamento empregada como referência é apresentada no Algoritmo 4.3. O método *Nearest Neighbor* baseia-se na função `CalculaListaVizinhos()`, que gera uma lista com todos vizinhos da tarefa mestre que estão em uma mesma distância (mesmo número de *hop*s). O teste para encontrar um mapeamento possível, nesse caso o primeiro vizinho mais próximo, é idêntico ao FF. Entretanto, a lista de vizinhos, gerada a cada bloco PARA, permite o andamento circular conforme a Figura 4.4.

No Algoritmo 4.3, o bloco ENQUANTO (na linha 4) pode ser executado até o número máximo de *hop*s (Equação 4.22), enquanto o bloco PARA aninhado (na linha 10) pode ser executado até o número máximo de vizinhos possíveis (Equação 4.23). Assim o número máximo de iterações do algoritmo NN pode ser obtido através do produto entre estes dois parâmetros, como apresentado abaixo. Generalizando para o caso de um MPSoC com dimensões iguais, NN apresenta complexidade $O(x^2)$.

$$\begin{aligned} maxNumIteracoes(NN,x,y) &= maxDistancia(x,y) \times maxNumVizinhos(x,y) \\ &= (x+y+2) \times (x+y-2) \\ &= x^2 + y^2 + 2xy - 4 \\ &= x^2 + x^2 + 2x^2 - 4 \quad para \quad x = y \\ &= 4x^2 - 4 \qquad\qquad \rightarrow \qquad O(x^2) \end{aligned}$$

ALGORITMO 4.3. MAPEAMENTO SEGUNDO O MÉTODO NN DE REFERÊNCIA.

```
1  PE = MestrePE  // INICIA PELO PE DO MESTRE
2  NumeroHOPs = 1 // INICIA FAZENDO APENAS SALTO DE 1 HOP
3
4  ENQUANTO (1) FAÇA
5
6     // CRIA LISTA DE VIZINHOS DISTANTES NumeroHOPs DE MestrePE
7     CalculaListaVizinhos(NumeroHOPs,TaskMestre)
8
9     // PROCURA ENTRE OS VIZINHOS COM MESMA DISTANCIA
10    PARA (PE na ListaDeVizinhos)
11
12       // SELECIONA PRIMEIRO VIZINHO LIVRE COM TIPO COMPATÍVEL AO DA TAREFA A SER MAPEADA
13       SE (PE.estado == LIVRE) E (PE.tipo == tarefa.tipo) ENTÃO Retorna(PE)
14
15    FIM PARA
16
17    NumeroHOPs=NumeroHOPs+1
18
19 FIM ENQUANTO
```

A primeira heurística *congestion-aware* aqui proposta, *Minimum Maximum Channel Load* tem sua descrição apresentada no Algoritmo 4.4. Como pode ser visto, para cada possível mapeamento a função `CalculaCaminhoDeComunicação(T1,T2)` retorna o caminho entre as tarefas envolvidas na comunicação. Em seguida, somente os canais que fazem parte desse caminho são atualizados através da função `AtualizaCanal(C,T)`. Esta considera as taxas de comunicação passadas como parâmetro. Seguindo, a maior ocupação dentre todos os canais é obtida. O último bloco SE seleciona o mapeamento com mínimo máximo.

No Algoritmo 4.4, o bloco ENQUANTO (na linha 5) pode ser executado para todos os elementos de processamento, de acordo com a Equação 4.20. Enquanto isso, o bloco PARA aninhado (na linha 14) é executado para todos os canais do MPSoC de acordo com a Equação 4.21. O número máximo de iterações realizadas pelo algoritmo MMCL pode ser obtido através do produto entre estes dois parâmetros, como apresentado a seguir. Adicionalmente, generalizando para o caso onde o MPSoC possui ambas as dimensões iguais (*i. e.* $x = y$), o algoritmo MMCL apresenta complexidade $O(x^4)$.

$$\begin{aligned} maxNumIteracoes(MMCL, x, y) &= NumPEs(x, y) \times NumCanais(x, y) \\ &= (xy) \times (4xy - 2x - 2y) \\ &= 4x^2y^2 - 2x^2y - 2xy^2 \\ &= 4x^4 - 4x^3 \quad para \;\; x = y \quad \rightarrow \quad O(x^4) \end{aligned}$$

ALGORITMO 4.4. MAPEAMENTO SEGUNDO O MÉTODO MMCL PROPOSTO.

```
1  PE = 0  // INICIA PELO PE 0
2  minimoMAXIMO = 1000000; // MAXIMO INICIAL ALTO
3
4  // PERCORRE TODOS OS PEs CALCULANDO OS MAXIMOS PARA CADA UM DOS POSSÍVEIS MAPEAMENTOS
5  ENQUANTO (PE < NumeroDePEs) FAÇA
6
7     // SE FOR UMA POSIÇÃO LIVRE COMPATÍVEL ENTÃO CALCULA O MAXIMO
8     SE (PE.estado == LIVRE) E (PE.tipo == tarefa.tipo) ENTÃO
9
10       MAXIMO = 0
11
12       CP = CalculaCaminhoDeComunicação(Tarefa,TarefaMestre)
13
14       PARA Cada Canal // PARA TODOS OS CANAIS DAS MATRIZES DE CANAIS
15          SE (Canal pertence ao CP) ENTÃO AtualizaCanal(Canal,Taxa) // SE CANAL NO CAMINHO
16          SE (Canal.ocupacao > MAXIMO) ENTÃO MAXIMO = Canal.ocupacao
17       FIM PARA
18
19       SE (MAXIMO < minimoMAXIMO) ENTÃO // SE ENCONTROU UM MAXIMO MENOR
20          minimoMAXIMO = MAXIMO  // AUTALIZA MAXIMO
21          PEAlvo = PE  // MUDA PEAlvo TEMPORARIO
22       FIM SE
23
24    FIM SE
25
26    PE=PE+1 // PROXIMO PE
27
28 FIM ENQUANTO
29
30 Retorna(PEAlvo) // RETORNA O PE QUE APRESENTOU O MENOR MAXIMO
```

Como pode ser notado, os Algoritmos 4.4 e 4.5 são bastante semelhantes. Na heurística *Minimum Average Channel Load*, ao invés de considerar o máximo na ocupação dos canais, considera-se a média. Por isso, MACL (Algoritmo 4.5) acumula a ocupação de todos os canais da NoC para cada possível mapeamento, e em seguida calcula a media (na linha 19). No último bloco SE (na linha 21), o mapeamento que apresentou menor média é selecionado. Assim como no caso de MMCL, o algoritmo proposto para a heurística MACL também apresenta complexidade $O(x^4)$, visto que ambos apresentam os mesmos blocos aninhados, como os mesmo números de iterações para o pior caso.

$$
\begin{aligned}
maxNumIteracoes(MACL, x, y) &= maxNumIteracoes(MMCL, x, y) \\
&= NumPEs(x, y) \times NumCanais(x, y) \\
&= (xy) \times (4xy - 2x - 2y) \\
&= 4x^2y^2 - 2x^2y - 2xy^2 \\
&= 4x^4 - 4x^3 \quad para \;\; x = y \quad \rightarrow \quad O(x^4)
\end{aligned}
$$

ALGORITMO 4.5. MAPEAMENTO SEGUNDO O MÉTODO MACL PROPOSTO.

```
1  PE = 0  // INICIA PELO PE 0
2  minimaMEDIA = 1000000;  // MEDIA INICIAL ALTA
3
4  // PERCORRE TODOS OS PEs CALCULANDO AS MÉDIAS PARA CADA UM DOS POSSÍVEIS MAPEAMENTOS
5  ENQUANTO (PE < NumeroDePEs) FAÇA
6
7     // SE FOR UMA POSIÇÃO LIVRE COMPATÍVEL ENTÃO CALCULA A MÉDIA
8     SE (PE.estado == LIVRE) E (PE.tipo == tarefa.tipo) ENTÃO
9
10       Ocupacao = 0
11
12       CP = CalculaCaminhoDeComunicação(Tarefa,TarefaMestre)
13
14       PARA Cada Canal // PARA TODOS OS CANAIS DAS MATRIZES DE CANAIS
15          SE (Canal pertence ao CP) ENTÃO AtualizaCanal(Canal,Taxa) // SE CANAL NO CAMINHO
16          Ocupacao = Ocupacao + Canal.ocupacao
17       FIM PARA
18
19       MEDIAtmp = Ocupacao/NumeroDeCanais  // MÉDIA TEMPORÁRIA
20
21       SE (MEDIAtmp < MEDIA) ENTÃO // SE ENCONTROU UM MÉDIA MENOR
22          minimaMEDIA = MEDIAtmp  // AUTALIZA MÉDIA
23          PEAlvo = PE  // MUDA PEAlvo TEMPORARIO
24       FIM SE
25
26    FIM SE
27
28    PE=PE+1 // PROXIMO PE
29
30 FIM ENQUANTO
31
32 Retorna(PEAlvo) // RETORNA O PE QUE APRESENTOU O MENOR MPEDIA
```

O Algoritmo 4.6 representa a heurística *Path Load*. No bloco PARA da linha 14 pode-se notar que apenas os canais que compõem o caminho de comunicação são considerados. Para cada um deles, o mapeamento é emulado através da função `AtualizaCanal(C,T)`, e a ocupação resultante é acumulada na linha 16 do algoritmo. O mapeamento selecionado será aquele que apresentar o menor somatório da ocupação dos canais, conforme o último bloco SE (na linha 19).

Na linha 5, o bloco ENQUANTO pode ser executado, no pior caso, para todos os elementos de processamento do MPSoC (Equação 4.20). O bloco PARA aninhado (na linha 14), por sua vez, será executado no pior caso para o maior caminho de comunicação possível. Esse é dado pela Equação 4.22, que indica a distância máxima entre duas tarefas em um MPSoC malha, de acordo com o algoritmo de roteamento xy. Sendo assim, o número

máximo de iterações realizadas por PL pode ser obtido como apresentado abaixo. Veja que o algoritmo PL apresenta uma complexidade $O(x^3)$, intermediária àquelas apresentadas pelos algoritmos NN e MACL/MMCL, respectivamente $O(x^2)$ e $O(x^4)$.

$$\begin{aligned} maxNumIteracoes(PL,x,y) &= NumPEs(x,y) \times maxDistancia(x,y) \\ &= (xy) \times (x+y+2) \\ &= x^2y + xy^2 + 2xy \\ &= 2x^3 + 2x^2 \quad para \;\; x = y \quad \rightarrow \quad O(x^3) \end{aligned}$$

ALGORITMO 4.6. MAPEAMENTO SEGUNDO O MÉTODO PL PROPOSTO.

```
1  PE = 0  // INICIA PELO PE 0
2  minimoSOMATORIO = 1000000;  // SOMATORIO INICIAL ALTO
3
4  // PERCORRE TODOS OS PEs CALCULANDO A OCUPAÇÃO APENAS NO CAMINHO DE COMUNICAÇÃO
5  ENQUANTO (PE < NumeroDePEs) FAÇA
6
7     // SE FOR UMA POSIÇÃO LIVRE COMPATÍVEL ENTÃO CALCULA O SOMATORIO
8     SE (PE.estado == LIVRE) E (PE.tipo == Tarefa.tipo) ENTÃO
9
10       Ocupacao = 0
11
12       CP = CalculaCaminhoDeComunicação(Tarefa,TarefaMestre)
13
14       PARA cada Canal que pertence ao CP // PARA TODOS OS CANAIS CANAL NO CAMINHO
15          AtualizaCanal(Canal,Taxa)
16          Ocupacao = Ocupacao + Canal.ocupacao
17       FIM PARA
18
19       SE (Ocupacao < minimoSOMATORIO) ENTÃO // SE ENCONTROU UM SOMATORIO MENOR
20          minimoSOMATORIO = Ocupacao  // AUTALIZA SOMATORIO
21          PEAlvo = PE  // MUDA PEAlvo TEMPORARIO
22       FIM SE
23
24    FIM SE
25
26    PE=PE+1 // PROXIMO PE
27
28 FIM ENQUANTO
29
30 Retorna(PEAlvo) // RETORNA O PE QUE APRESENTOU O MENOR SOMATÓRIO
```

O Algoritmo 4.7 descreve a heurística *Best Neighbor*. Nesse caso, conforme a linha 9, o somatório da ocupação dos canais do caminho de comunicação é computado apenas para uma dada lista de vizinhos, obtida através da função `CalculaListaVizinhos()`, toda vez que uma nova iteração faz-se necessária. Assim como em NN, é a lista de vizinhos que permite o andamento circular na procura de uma posição para a tarefa.

Na linha 5 do algoritmo, o bloco ENQUANTO pode ser executado no pior caso para todas as distâncias possíveis, dada pela Equação 4.21. O bloco PARA (na linha 9) é executado para a lista de vizinhos cujo valor máximo é representado pela Equação 4.23. O bloco PARA da linha 18 é executado apenas uma única vez, e por isso é desconsiderado no cálculo do número de iterações máximo a serem realizadas por BN. Veja abaixo que BN e NN apresentam a mesma complexidade $O(x^2)$.

$$
\begin{aligned}
maxNumIteracoes(BN, x, y) &= maxNumIteracoes(NN, x, y) \\
&= maxDistancia(x, y) \times maxNumVizinhos(x, y) \\
&= (x + y + 2) \times (x + y - 2) \\
&= 4x^2 - 4 \qquad para \quad x = y \qquad \rightarrow \qquad O(x^2)
\end{aligned}
$$

ALGORITMO 4.7. MAPEAMENTO SEGUNDO O MÉTODO BN PROPOSTO.

```
1  NumeroHOPs = 1  // INICIA PELOS PEs COM 1 HOP DE DISTANCIA
2  minimoSOMATORIO = 1000000;  // SOMATORIO INICIAL ALTO
3
4  // PERCORRE PEs COMO NN CALCULANDO A OCUPAÇÃO APENAS NO CAMINHO DE COMUNICAÇÃO
5  ENQUANTO (1) FAÇA
6
7     CalculaListaVizinhos(NumeroHOPs,TaskMestre) // CRIA LISTA DE VIZINHOS
8
9     PARA (PE na ListaDeVizinhos) // TESTA OS VIZINHOS DA LISTA
10
11       // SE FOR UMA POSIÇÃO LIVRE COMPATÍVEL ENTÃO CALCULA O SOMATORIO
12       SE (PE.estado == LIVRE) E (PE.tipo == tarefa.tipo) ENTÃO
13
14          Ocupacao = 0
15
16          CP = CalculaCaminhoDeComunicação(Tarefa,TarefaMestre)
17
18          PARA cada Canal que pertence ao CP // PARA TODOS OS CANAIS CANAL NO CAMINHO
19             AtualizaCanal(Canal,Taxa)
20             Ocupacao = Ocupacao + Canal.ocupacao
21          FIM PARA
22
23          SE (Ocupacao < minimoSOMATORIO) ENTÃO // SE ENCONTROU UM SOMATORIO MENOR
24             minimoSOMATORIO = Ocupacao  // AUTALIZA SOMATORIO
25             PEAlvo = PE                 // MUDA PEAlvo TEMPORARIO
26          FIM SE
27
28       FIM SE
29
30    FIM PARA
31
32    // SE CALCULOU SOMATÓRIO ENTAO TEM UM VIZINHO LIVRE E O MELHOR FOI ESCOLHIDO
33    SE (minimoSOMATORIO < 100000) ENTÃO Retorna(PEAlvo)
34
35    NumeroHOPs=NumeroHOPs+1  // TESTA VIZINHO COM MAIOR DISTANCIA
36
37 FIM ENQUANTO
38
39 Retorna(PEAlvo) // RETORNA O PE VIZINHO QUE APRESENTOU O MENOR SOMATÓRIO
```

A Tabela 4.1 apresenta um quadro comparativo entre os algoritmos de mapeamento aqui propostos e implementados. Todos eles são empregados para o mapeamento individual das tarefas de acordo com uma visão local da aplicação, que inclui apenas a informação sobre a comunicação entre as tarefas solicitada e solicitante. Com relação à NoC, as heurísticas MACL e MMCL valem-se da visão global da ocupação de todos os canais, enquanto PL e BN atém-se apenas ao caminho de comunicação refletindo uma visão local, portanto. No caso da visão sobre os PEs, dentre os algoritmos *congestion-aware*, apenas BN pode não avaliar todos os possíveis mapeamentos (*i. e.* visão parcial), a exemplo do que deve acontecer quando os algoritmos de referência NN e FF são utilizados.

TABELA 4.1. COMPARATIVO ENTRE OS ALGORITMOS DE MAPEAMENTO PROPOSTOS.

Algoritmo	*Objetivo do algoritmo*	*Função Custo*	*Caminhamento na procura*	*Visão Aplicação (Grafo)*	*Visão da NoC* (Ocupação?)	*Visão dos PEs* (Livre?)	*Complexidade Estimada*
FF	Referência	Não se aplica	Fixo coluna por coluna (ver Figura 4.4)	Local	Não se aplica	Parcial	$O(x^2)$ [1,2]
NN	Referência Aproximar as tarefas comunicantes	Não se aplica	Variável e Circular (ver Figura 4.5)	Local	Não se aplica	Parcial	$O(x^2)$
MMCL	Reduzir os congestionamentos na NoC através da eliminação de picos de ocupação	Máxima ocupação dos canais	Mesmo de FF	Local	Global	Global	$O(x^4)$
MACL	Reduzir os congestionamentos na NoC através da distribuição homogênea da ocupação dos canais	Ocupação média dos canais	Mesmo de FF	Local	Global	Global	$O(x^4)$
PL	Reduzir os congestionamentos e o tempo de mapeamento considerando apenas os canais do caminho de comunicação	Somatório da ocupação dos canais	Mesmo de FF	Local	**Local**	Global	$O(x^3)$
BN	Reduzir os congestionamentos e o tempo de mapeamento considerando apenas alguns recursos vizinhos na sua procura	Somatório da ocupação dos canais	Mesmo de NN	Local	**Local**	**Parcial**	$O(x^2)$

[1] Algoritmo simples, que emprega apenas testes e nenhuma função custo.

[2] X e Y são as dimensões do MPSoC. No cálculo das complexidades assume-se x = y.

Quando existem muitas alternativas de mapeamento o tempo de procura é maior para as heurísticas MACL, MMCL e PL. No caso de NN, FF e BN o tempo de procura está mais relacionado com a distância do recurso livre. No caso em que o método FF é empregado, quanto mais a direita e acima estiver situado o primeiro recurso livre, maior será o tempo de procura, independentemente de quantos recursos estão livres. No caso de NN e BN, maior será o tempo de procura quando maior for o número de *hops* de distância entre o recurso livre e a posição da tarefa solicitante.

Vistos os detalhes sobre a arquitetura alvo, bem como os algoritmos de mapeamento propostos, pode-se prosseguir aos Capítulos 5 e 6. Estes apresentam cada um dos experimentos realizados, além de discutir os resultados obtidos a partir das simulações.

5. AVALIAÇÃO DAS HEURÍSTICAS *CONGESTION-AWARE*

Os experimentos realizados são divididos em duas partes. A primeira delas, apresentado no presente Capítulo, visa a comparação entre as heurísticas de mapeamento *congestion-aware* propostas (*i. e.* MMCL, MACL, PL e BN) frente aos algoritmos de referência FF e BN. Ao contrário dos propostos, os algoritmos *First Free* e *Nearest Neighbor* não possuem função custo de congestionamento para a definição do mapeamento de uma dada tarefa. Estes foram implementados justamente para seu emprego como referência nas avaliações. A segunda parte de experimentos é apresentada no próximo Capítulo. Ela visa a comparação entre as heurísticas propostas, com mapeamento tarefa a tarefa, e heurísticas globais, onde todas as tarefas de uma dada aplicação são mapeadas simultaneamente.

Antes da discussão dos experimentos, faz-se necessária a apresentação da arquitetura alvo (Seção 5.1), válida para as duas partes de experimentos. Ela discute ambas as modelagens, da NoC descrita em VHDL, e dos elementos de processamento descritos em *SystemC*. As co-simulações realizadas são baseadas no simulador *ModelSim SE 6.4* da *Mentor Graphics Corporation* [MEN08]. O relógio de operação adotado opera segundo uma freqüência fixada em 100MHz.

5.1. Organização de MPSoC Alvo

O MPSoC adotado possui topologia malha conforme a infra-estrutura de comunicação empregada (ver Figura 5.1). A NoC Hermes [MOR04] foi escolhida principalmente devido a disponibilidade de ferramentas para sua geração, síntese, geração de tráfego, simulação e avaliação. Dentre as funcionalidades da ferramenta ATLAS [GAP07], principalmente a geração (Maia [OST05]) foi utilizada aqui. Através dela foi possível obter a descrição VHDL da NoC em nível RTL.

Dentre os parâmetros adotados para a NoC encontram-se: chaveamento de pacotes *wormhole*; *buffers* de entrada nos roteadores; controle de fluxo *handshake*; e roteamento

segundo o algoritmo xy. O algoritmo de roteamento guia as heurísticas de mapeamento propostas, as quais consideram a ocupação dos canais que serão utilizados (*i. e.* definidos com base no algoritmo de roteamento) pela tarefa requisitada, para cada possível mapeamento. O controle de fluxo define quantos ciclos de relógio são necessários para transmitir um dado *flit* entre dois roteadores da NoC. O protocolo de *handshake* foi adotado em virtude da sua simplicidade. Contudo, ele suporta apenas a transmissão de um *flit* a cada dois ciclos de relógio, limitando a largura de banda máxima disponível a 50% da capacidade real de um dado canal.

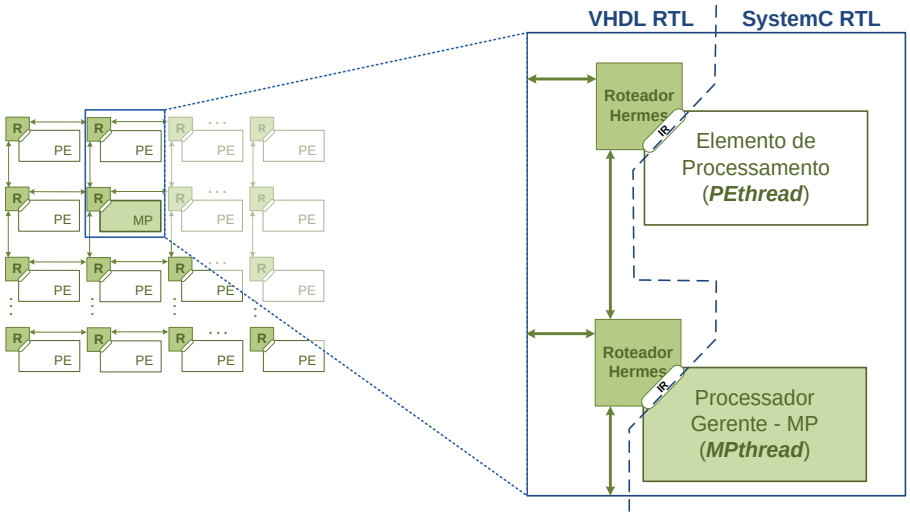


FIGURA 5.1. ABORDAGEM ADOTADA PARA MODELAGEM DA NOC, MP E PES.

5.1.1. Modelagem do Processador Gerente (MPthread)

Os elementos de processamento são modelados em nível RTL utilizando a linguagem *SystemC*, conforme descrito na Seção 3.7.3. Duas *SystemC_Cthreads* foram implementadas: *MPthread* e *PEthread*. A primeira delas é responsável pelas tarefas de controle atribuídas ao processador gerente. A Figura 5.2 apresenta-se os componentes do MP.

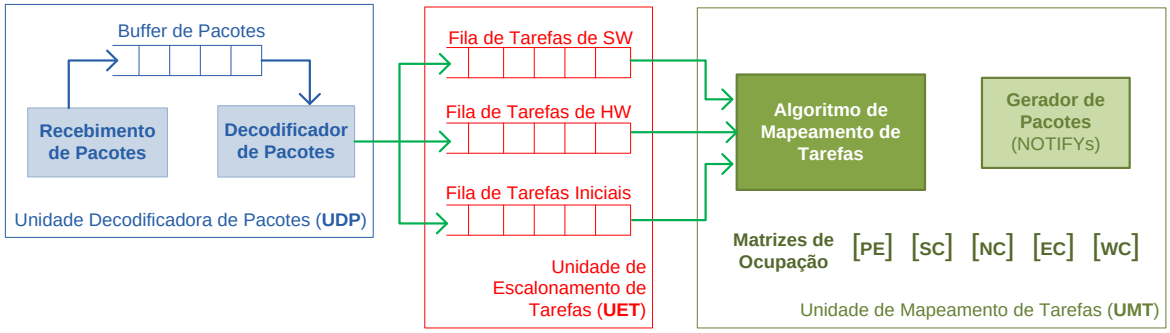


Figura 5.2. Componentes da implementação do processador gerente MP.

O comportamento do MP pode ser representado pelas três etapas seguintes:

Etapa 1. **Captura do Pacote**: Na primeira etapa, quando um dado pacote é recebido pela UDP, seus atributos são capturados. Os pacotes recebidos podem ser do tipo REQUEST ou RELEASE. Dentre os principais atributos dos pacotes constam o identificador do PE, o tipo do pacote e os identificadores das tarefas envolvidas na comunicação (*i. e.* mestre e escrava).

Etapa 2. **Decodificação do Pacote e Escalonamento**: Na segunda etapa, ainda na UDP, o pacote é decodificado. Se tratando de um pacote REQUEST, o identificador da tarefa requisitada deve ser então inserido na fila de escalonamento adequada (*i. e.* na UET) de acordo com o tipo dessa tarefa. Ambos os identificadores de mestre e escrava são inseridos na fila. A estratégia de escalonamento adotada não atribui prioridades às tarefas. Um mecanismo para evitar entradas duplicadas de tarefas em uma dada fila foi implementado. Quando o identificador da tarefa solicitada já se encontra na fila, então apenas um novo identificador de mestre é atribuído à mesma posição. Ao invés de mapear varias vezes uma mesma tarefa, uma para cara entrada na fila, agora apenas pacotes NOTIFYs adicionais serão gerados. No caso do recebimento de um pacote RELEASE, então o recurso identificado no pacote recebe o estado de livre. O identificador da tarefa que ocupava tal recurso é mantido para que ela possa ser reusada caso seja necessária.

Etapa 3. **Mapeamento da Tarefa**: A terceira etapa dedica-se ao mapeamento da tarefa. Para isso, verifica-se a existência de tarefas em cada uma das filas. De acordo com a Figura 5.3, se a tarefa lida da fila encontrar-se já mapeada no sistema, apenas os pacotes NOTIFYs são enviados. Com isso, os atrasos de mapeamento e configuração são eliminados. Caso contrário, deve-se verificar se existem recursos compatíveis livres. Em caso negativo, a tarefa continua na fila de escalonamento. De outra forma, a heurística de mapeamento é utilizada para selecionar um dos recursos livres para alocar a tarefa. Na seqüência, a tarefa é configurada nesse recurso. Após sua configuração, ela deve ser inicializada.

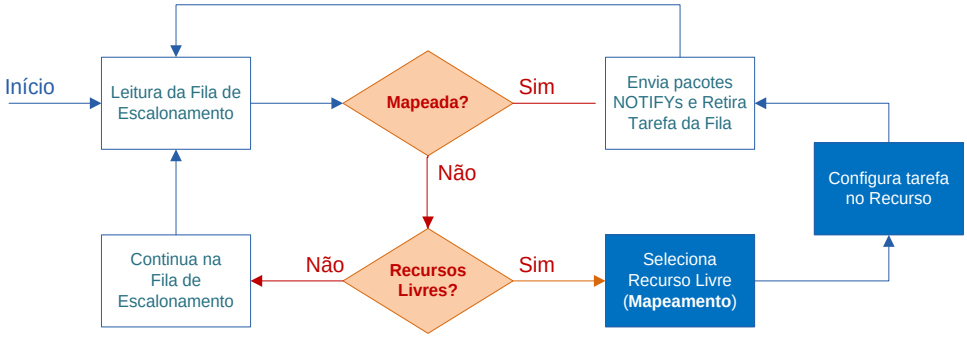


FIGURA 5.3. DIAGRAMA DE DECISÕES DURANTE O MAPEAMENTO DE TAREFAS NA ETAPA 3.

5.1.2. Modelagem dos Elementos de Processamento (PEthread)

A *PEthread* implementa o comportamento genérico de uma dada tarefa, de acordo com o protocolo de comunicação definido na Seção 3.4. A fim de permitir um comportamento especifico para cada uma das tarefas a executar no MPSoC, a *PEthread* é parametrizada por um dado arquivo de configuração. Esse arquivo informa detalhes sobre cada tarefa, incluindo: (*i*) tempo de execução; (*ii*) com quais outras tarefas ela se comunica; (*iii*) quais os volumes de comunicação, e ainda (*iv*) as respectivas taxas. Assim, de acordo com a tarefa (*i. e.* arquivo de configuração) atribuída a um dado PE (*i. e. PEthread*), diferentes tráfegos (*i. e.* taxas de injeção e destinos) serão gerados.

Um *período de amostragem* - SP foi definido para a geração de pacotes na *PEthread*. A cada período de amostragem, um determinado número de *flits* é gerado, de forma a atender as taxas e volumes contidos no arquivo de configuração. Por exemplo, assumindo um período de amostragem *SP = 100 ciclos de relógio*. Se a tarefa *ta* comunica-se com as tarefas *tb* e *tc*, segundo taxas de envio 7% e 25%, respectivamente, então a geração de tráfego segue o diagrama de tempo da Figura 5.4. Deve-se notar que a geração dos *flits* a uma taxa *tx* consome 2*tx* ciclos de relógio, devido à utilização do protocolo *handshake*.

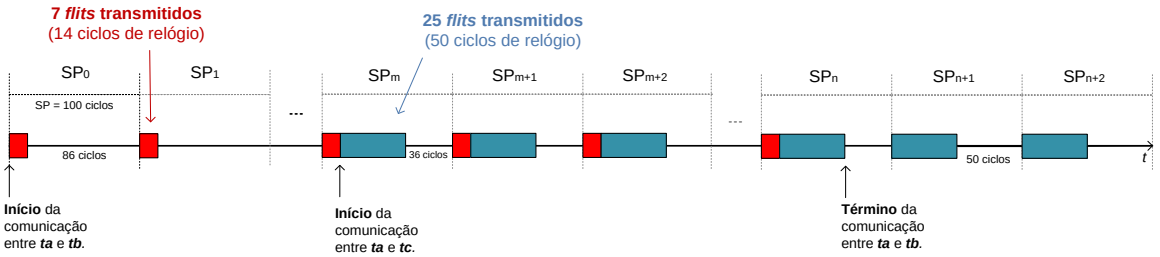


FIGURA 5.4. GERAÇÃO DE TRÁFEGO DE *TA* PARA *TB* (7%) E *TC* (25%), SEGUNDO UM SP = 100 CICLOS DE RELÓGIO.

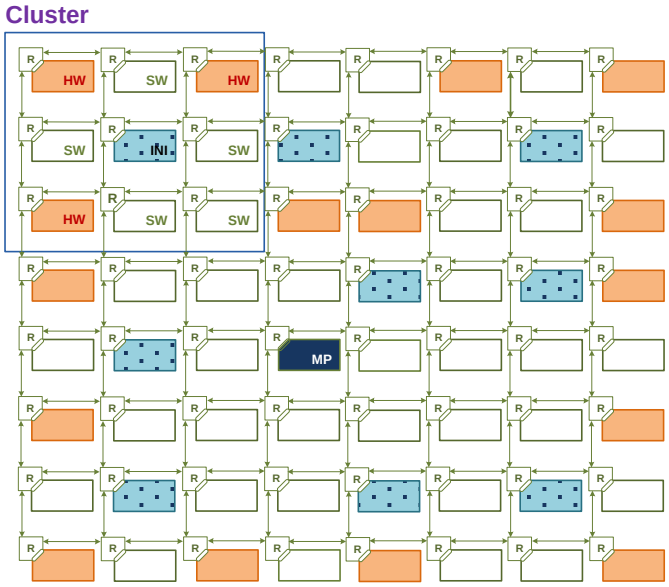
No inicio do diagrama da Figura 5.4, acontece apenas a comunicação entre *ta* e *tb* (SP_0 a SP_{m-1}). Em seguida, entre SP_m e SP_n, a cada período: (*i*) um pacote com 7 *flits* é enviado para *tb*; (*ii*) um pacote com 25 *flits* é enviado para *tc*; e (*iii*) no tempo restante o PE recebe pacotes de entrada ou processa os dados recebidos. Como cada *flit* requer dois ciclos de relógio para ser transmitido, o período "ocioso" será, nesse caso, de 36 ciclos de relógio. O tempo de processamento da tarefa está implícito, durante o tempo em que a tarefa não se encontra comunicando. Esse procedimento se repete até que o volume de dados seja completamente gerado e transmitido aos respectivos destinos. Quando *ta* termina os dados a serem enviados para *tb* (em SP_n), são produzidos apenas os 25 *flits* para *tc* a cada SP, com o tempo ocioso de 50 ciclos de relógio.

No exemplo da Figura 5.4, o período de amostragem de 100 ciclos de relógio foi adotado para facilitar a conversão das taxas em número de *flits*. Entretanto, nos experi-

mentos foram utilizados períodos na ordem de *10.000 ciclos de relógio*, no qual uma taxa de envio de 24% implica na transmissão de pacotes de 2400 *flits* a cada SP. A estratégia empregada para a geração de pacotes nas *PEthread* segue uma distribuição temporal *Pareto On-Off*.

5.2. Cenários de Simulação

Dentre as características importantes a serem discutidas sobre o cenário da simulação constam a modelagem do MPSoC, bem como as aplicações utilizadas. O MPSoC heterogêneo modelado nesse primeiro grupo de experimentos possui 8 colunas e 8 linhas de elementos de processamento (Figura 5.5). Nas avaliações, o emprego de MPSoCs menores pode comprometer os resultados visto que existem poucas possibilidades de mapeamento. Assim, provavelmente os resultados serão muito próximos, independentemente do algoritmo empregado para definir o mapeamento das tarefas.



R – Roteador MP – Processador Gerente HW – PE reservado para tarefas de hardware

INI – PE reservado para tarefas iniciais das aplicações SW – PE reservado para tarefas de software

FIGURA 5.5. MPSOC HETEROGÊNEO 8X8: POSICIONAMENTO DOS PES DE ACORDO COM SEU TIPO.

A disposição dos PEs da Figura 5.5 busca distribuir uniformemente os PEs sobre a área do MPSoCs, de acordo com seu tipo. Ao todo, são 64 PEs, sendo 16 dedicados à execução de tarefas de hardware, e 47 PEs que suportam a execução de tarefas de software. Um dos PEs é reservado para as funcionalidades do processador gerente. Dentre os 47 PEs de software, alguns são reservados ao mapeamento de tarefas iniciais das aplicações. A quantia desses recursos foi variada entre 9 e 15, indicando um paralelismo na execução de até 15 aplicações simultâneas.

Três cenários de aplicações sintéticas são considerados nesse primeiro grupo de experimentos. Dentre eles constam as simulações das seguintes aplicações:

Cenário A. **Pipeline**: Composto por *20 aplicações idênticas* com grafo cuja topologia representa um formato de *pipeline*, típico de aplicações orientadas a fluxo de dados. Conforme a Figura 5.6, cada uma das aplicações contém 10 tarefas, com taxas de injeção de dados idênticas em todas as arestas do grafo. As taxas *Rms* foram variadas entre 5% e 30% da largura de banda disponível nas seis simulações que compõem este cenário. No grafo da Figura 5.6, cada aresta possui um conjunto de atributos *{600, Rms, 10, 5}*, que indicam uma comunicação no sentido da aresta de acordo com um volume de 600 pacotes transmitidos a uma taxa de *Rms%*; e uma comunicação no sentido oposto como volume de 10 pacotes transmitidos a uma taxa de 5% da largura de banda.

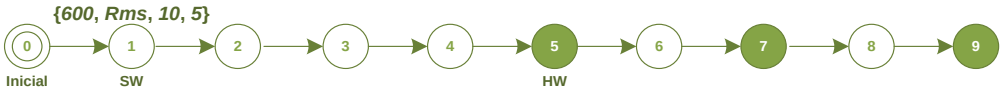


FIGURA 5.6. CENÁRIO A: APLICAÇÃO COM GRAFO COM TOPOLOGIA PIPELINE. A TAXA *RMS* É VARIADA ENTRE 5 E 30%.

Cenário B. **Árvore**: Composto por *20 aplicações idênticas* cujos grafos apresentam topologia de *árvore*. Elas representam aplicações com um maior grau de paralelismo entre as tarefas. Assim como no cenário anterior, cada aplicação árvore possui 10 tarefas (ver Figura 5.7). As taxas de injeção de pacotes na rede são idênticas em todas as arestas do grafo. São realizadas 4 simulações, com taxas de injeção variadas entre 5 e 20% da largura de banda disponível.

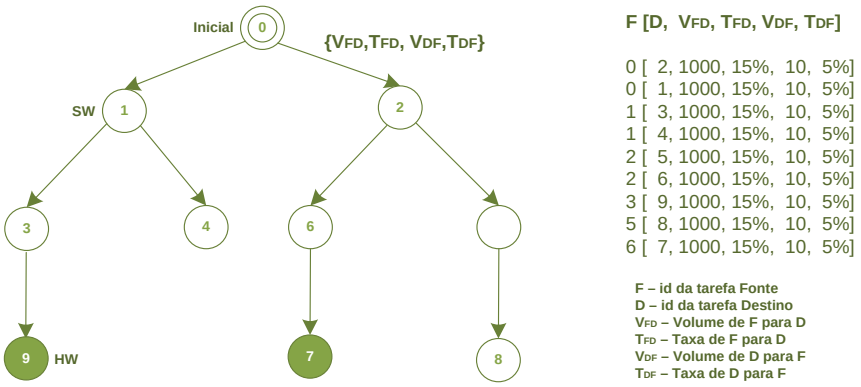


FIGURA 5.7. CENÁRIO B: APLICAÇÃO COM GRAFO COM TOPOLOGIA ÁRVORE. A TAXA É VARIADA ENTRE 5 E 20%.

Cenário C. **Geral**: Composto por *20 aplicações diferentes*, com grafos de topologias *variadas*. Essas aplicações foram geradas com o auxilio da ferramenta TGFF[1]. Nesses experimentos, foram utilizados grafos compostos por 5 a 10 vértices, e com taxas de injeção escolhidas aleatoriamente entre 5 e 30% da largura de banda disponível. Cada um dos grafos gerados pode ser visto na Figura 5.8. Maiores detalhes são apresentados no Apêndice A.

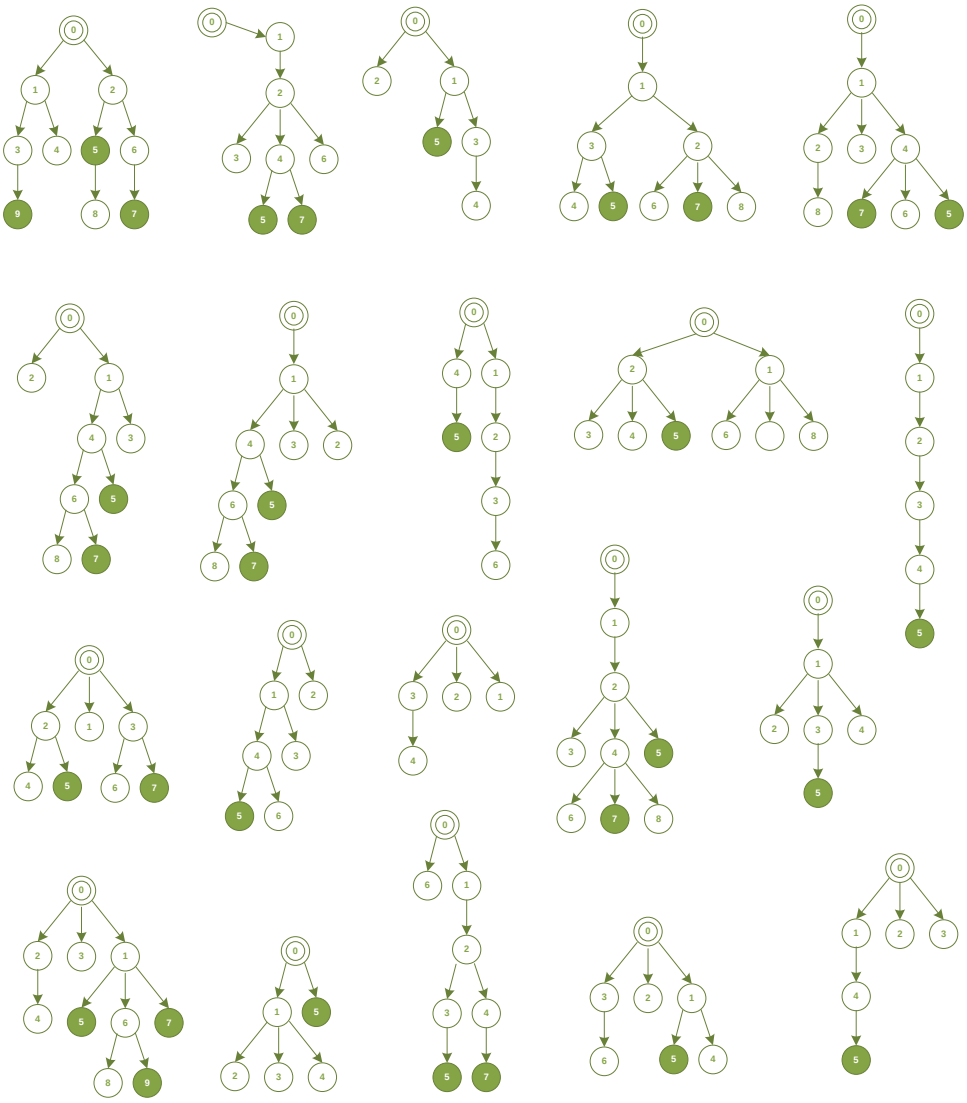


FIGURA 5.8. CENÁRIO C: 20 GRAFOS DE APLICAÇÕES GERADOS ALEATORIAMENTE NA FERRAMENTA TGFF.

[1] *Task Graph For Free* ou simplesmente TGFF [DIC98] é uma ferramenta que permite a geração automática de um conjunto de grafos. Para isso ela baseia-se em um conjunto de parâmetros, incluindo os números mínimo e máximo de vértices, e o intervalo para variação dos pesos nas arestas. O TGFF foi empregado em alguns dos trabalhos sobre mapeamento de tarefas revisados no Capítulo 2, dentre eles [LEI03b] [HU04] [LIN05] [WRO06] [CHO07].

5.3. Resultados Obtidos

Nesta primeira etapa de experimentos, os parâmetros de desempenho empregados incluem: (*i*) ocupação dos canais da NoC; (*ii*) latência dos pacotes; (*iii*) nível de congestionamento na NoC; e (*iv*) tempo de execução total do sistema.

5.3.1. Ocupação dos Canais da NoC

A carga dos canais é o primeiro parâmetro de desempenho investigado. Ela representa o estado de ocupação da NoC. Nos experimentos, toda informação sobre a ocupação dos canais foi obtida através de monitores distribuídos, inseridos em cada uma das portas de saída dos roteadores da NoC. Nas simulações, ambos os esquemas de monitores (*i. e.* centralizado e distribuído) foram avaliados. Nos experimentos com o monitor centralizado, onde o mapeamento é baseado em estimativas, os valores obtidos pelos monitores distribuídos são usados apenas nas medições de desempenho. No outro caso, nas avaliações com monitores distribuídos, suas medições são passadas ao gerente do sistema, que as considera para realizar o mapeamento das tarefas, conforme discutido na Seção 3.6.

Os gráficos da Figura 5.9 apresentam a ocupação dos canais da NoC ao longo do tempo de simulação, para cada um dos algoritmos de mapeamento implementado. Os dois valores plotados representam a média (linha mais escura) e o pico de ocupação dos canais (possíveis congestionamentos). A cada período de amostragem uma amostra de ocupação é plotada. Ela é medida como uma porcentagem da largura de banda disponível. O andamento do tempo de simulação se dá em milhões de ciclos de relógio (MCCs).

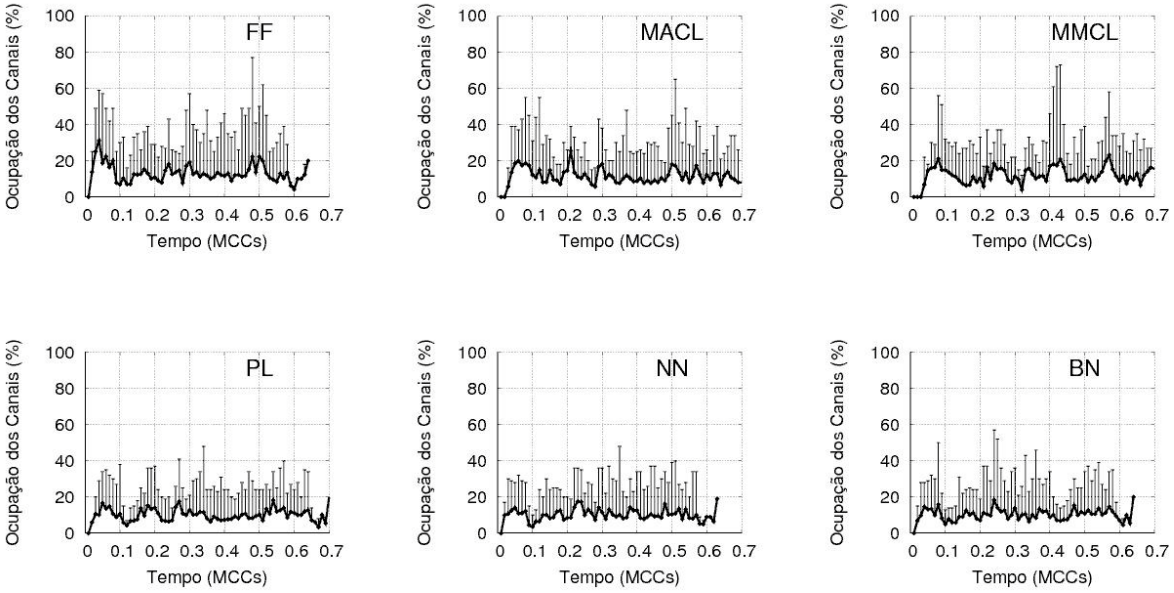


FIGURA 5.9. OCUPAÇÃO (MÉDIA E MÁXIMO) DOS CANAIS DA NOC AO LONGO DO TEMPO, PARA O CENÁRIO C.

A Figura 5.9 diz respeito às simulações do Cenário C, composto por grafos de aplicações gerados a partir da ferramenta TGFF. O algoritmo FF apresentou a pior solução em termos de ocupação dos canais. Enquanto isso, os algoritmos BN, NN e PL apresentam as melhores soluções, as quais resultam em uma baixa media de ocupação, e poucas saturações de canais, ou seja, poucos picos de ocupação maiores que 50% da largura de banda disponível. A complexidade dos algoritmos MMCL e MACL penaliza seus tempos de execução, o que pode ser observado nos gráficos destas heurísticas que gastam maior tempo para executar todas as 20 aplicações.

A avaliação das heurísticas apenas baseada nos gráficos da Figura 5.9 não permite precisão, visto que ela depende dos critérios adotados pelo observador, e ainda corresponde às medições obtidas de um único cenário. Uma avaliação mais completa é apresentada no gráfico da Figura 6.1, que considera todos os cenários simulados. Nos Cenários A e B deste gráfico é possível verificar que a *ocupação média dos canais da NoC* cresce à medida que a taxa de injeção aumenta.

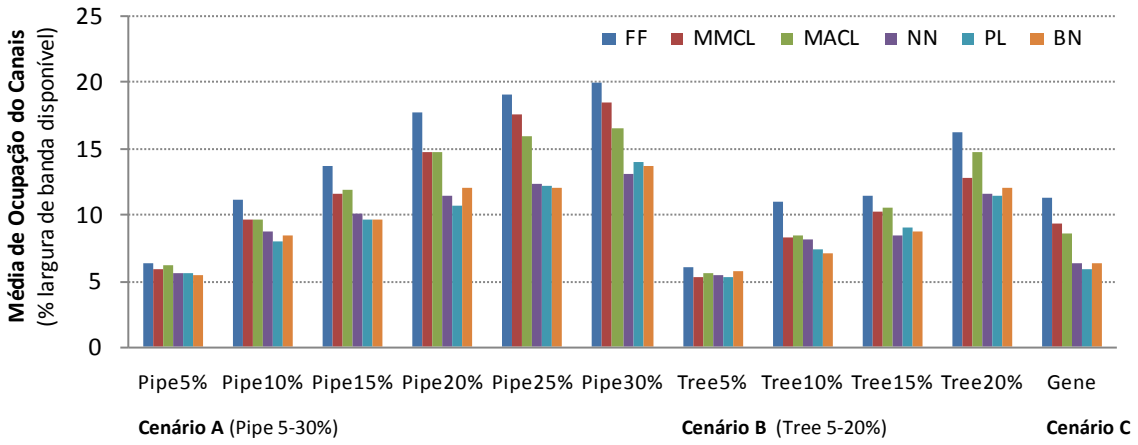


FIGURA 5.10. OCUPAÇÃO MÉDIA DOS CANAIS DA NOC (PERCENTUAL DA LARGURA DE BANDA DISPONÍVEL).

Além disso, com relação ao resultado apresentado quando o algoritmo FF é empregado, todos os demais algoritmos implementados reduzem a ocupação média. Conforme a média geral de todas as simulações, MMCL e MACL reduzem a ocupação dos canais em aproximadamente 14% (em comparação a FF). Esse resultado é bom, mas pior que o obtido quando NN, um algoritmo que não considera a ocupação dos canais é usado. Enquanto isso, *Best Neighbor* apresenta resultados semelhantes aos obtidos por *Nearest Neighbor*, com ganhos na ordem de 29% comparados a *First Free*.

A heurística PL, por sua vez, tem os melhores resultados nas simulações. De acordo com a média geral, PL apresenta um ganho de 31% na ocupação dos canais quando comparada a FF. Seu resultado é em média 2% melhor que aquele obtido por NN. Avali-

ando apenas o Cenário C, quando *Path Load* é empregado, pode-se observar um ganho de 48% relativo à FF, e até 6,5% relativo à NN.

Além da média, o *desvio padrão na ocupação* também é importante. Ele indica a distribuição do tráfego na NoC. Nesse caso, valores baixos representam uma distribuição homogênea de tráfego, enquanto valores altos sugerem que alguns canais possuem cargas altas, enquanto outros não estão sendo usados. Para os três cenários simulados (Figura 5.11), as heurísticas MMCL e MACL obtiveram um ganho de aproximadamente 10% com relação ao algoritmo FF. Novamente, NN, PL e BN foram os algoritmos com melhores resultados. Ainda assim, PL apresentou um menor desvio padrão na ocupação dos canais, em média 22% melhor que o apresentado por FF. Comparado ao NN, PL apresenta uma ganho de 2,5%. Nos experimentos do Cenário C, o ganho de *Path Load* atinge redução de 28% em relação ao resultado apresentado por *First Free*.

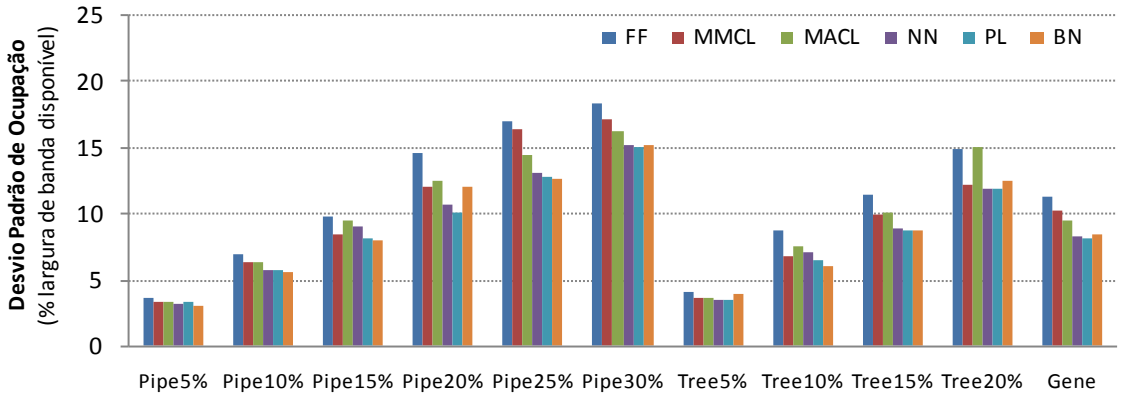


FIGURA 5.11. DESVIO PADRÃO NA OCUPAÇÃO DOS CANAIS DA NOC (PERCENTUAL DA LARGURA DE BANDA DISPONÍVEL).

5.3.2. Latência dos Pacotes Transmitidos

O segundo parâmetro de desempenho empregado na avaliação das heurísticas é a latência dos pacotes transmitidos durante a simulação. A latência de um dado pacote é uma função da distância entre os recursos nos quais as tarefas comunicantes estão mapeadas e, além disso, depende da ocupação dos canais pelos quais o pacote é transmitido. Quando muitos pacotes tentam compartilhar os mesmo canais da NoC, há tendência de que congestionamentos ocorram, aumentando a latência. No presente trabalho, as medições de latência são realizadas durante a simulação. Ela é dada em número de ciclos de relógio contados desde o envio do pacote em sua origem, até o seu recebimento no destino. Para isso, no envio do pacote, o instante de envio (*i. e. timestamp*) é inserido em uma dada posição do pacote. Quando o pacote chega ao seu destino, este valor é então capturado e subtraído do instante atual obtido a partir de uma variável global que mantém informação do número de ciclos de relógios simulados.

A Tabela 5.1 apresenta os valores capturados para a *latência média dos pacotes* transmitidos em cada uma das simulações. Na maioria delas, *Path Load* apresenta o melhor resultado, embora muito próximo daqueles apresentados pelos algoritmos NN e BN. A última linha da Tabela apresenta o ganho de cada uma das heurísticas compara ao algoritmo FF, calculado através da média de todos os experimentos. Os algoritmos MMCL e MACL apresentam redução de aproximadamente 6% e 8%, respectivamente no tempo para transmitir os pacotes. Enquanto isso, os demais algoritmos obtêm resultados mais significativos. NN, BN permitem redução nas latências na ordem de 15%, sendo que PL (16%) mais uma vez apresenta os melhores resultados.

TABELA 5.1. LATÊNCIA MÉDIA DOS PACOTES (EM CICLOS DE RELÓGIO).

Cenários	*Taxas*	FF	NN	MMCL	MACL	PL	BN
A. 20 aplicações com grafo Pipeline	*5%*	164	**141**	152	156	142	147
	10%	269	239	255	258	239	**238**
	15%	371	334	350	356	**328**	330
	20%	527	431	479	478	**425**	450
	25%	656	542	645	579	**531**	**531**
	30%	815	665	781	711	673	**664**
Ganho relativo ao FF (PIPE)		-	*16,00%*	*4,94%*	*9,40%*	***16,51%***	*15,75%*
B. 20 aplicações com grafo Árvore	*5%*	156	**143**	147	153	146	146
	10%	271	247	255	246	**233**	242
	15%	370	**327**	349	352	329	330
	20%	514	433	447	493	**430**	434
Ganho relativo ao FF (TREE)		-	*12,36%*	*8,56%*	*5,12%*	***13,28%***	*12,23%*
C. 20 Aplicações do TGFF		343	286	319	313	**285**	291
Ganho relativo ao FF (GENE)		-	*16,65%*	*6,82%*	*8,56%*	***16,88%***	*15,13%*
Ganho Global relativo ao FF		-	*14,98%*	*6,15%*	*8,07%*	***15,59%***	*14,66%*

Os resultados apresentados são relevantes, pois a latência consiste em um parâmetro que exerce grande influência sobre o desempenho de sistemas com requisitos de qualidade de serviço (QoS). Congestionamentos em sistemas dessa natureza podem acarretar na perda de pacotes, inaceitável em aplicações com *deadlines* estritos (*e. g.* de tempo real).

5.3.3. Congestionamentos na NoC

Tratando-se de heurísticas ditas *congestion-aware*, faz-se necessário avaliar o nível de congestionamento resultante no sistema. Os experimentos anteriores demonstraram reduções significativas na ocupação dos canais, principalmente quando as heurísticas PL e BN são empregadas, representando um ganho aproximado de 31% relativo ao algoritmo FF. Essa redução da ocupação acabou por se refletir sobre a latência dos pacotes, que também apresentou ganhos significativos, na ordem de 15%, como pode ser observado.

No presente trabalho, duas métricas relativas ao nível de congestionamento no sistema são empregadas. A primeira delas mede o *número de congestionamentos detectados* ao longo da simulação, ou seja, o *número de canais saturados*. O emprego dos monitores distribuídos foi imprescindível para esse tipo de medição. Os monitores detectam quando um determinado *flit* não pode ser enviado, identificando uma situação de congestionamento. Nesse caso, o uso dos monitores ganhou um significado similar ao definido por Marescaux [MAR05c], van den Brand [VAN07a] e Ciordas [CIO04], nos quais essas estruturas tem seu emprego fundamentado na descoberta de situações de congestionamento para posterior adequação de tráfego.

O gráfico da Figura 5.12 apresenta o número total de congestionamentos detectados durante cada uma das simulações. Conforme o gráfico, pode-se notar que houve uma redução significativa no número de congestionamentos. Considerando a média de todos os experimentos, quando *Minimum Average Channel Load* e *Minimum Maximum Channel Load* são empregados, pode-se obter 38% e 45% de redução no número de congestionamentos apresentados por FF. NN e BN, por sua vez, permitem uma redução de aproximadamente 70%, enquanto PL reduz em 77% os congestionamentos. Analisando apenas o Cenário C, a heurística PL permite a obtenção de resultados ainda melhores, com um ganho de 89% no número de congestionamentos comparado ao algoritmo FF, resultando em um ganho de até 40% relativo ao NN.

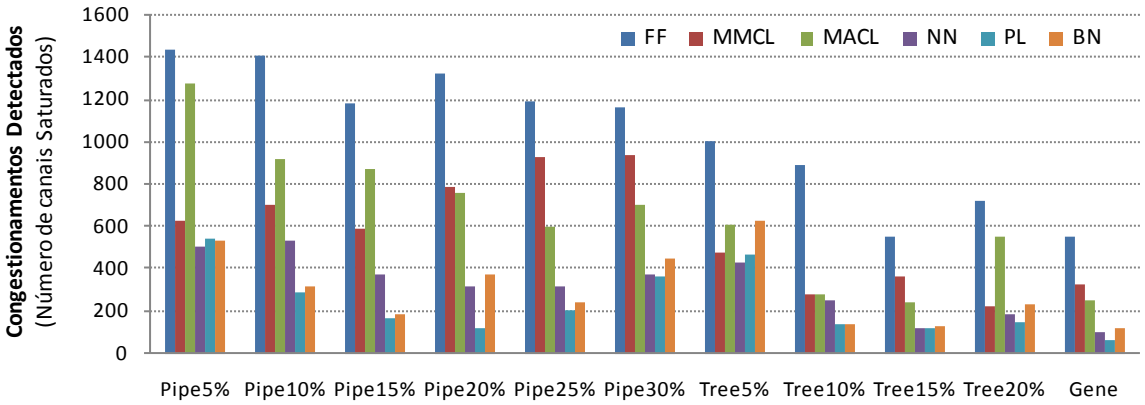


FIGURA 5.12. NÚMERO DE CONGESTIONAMENTOS DETECTADOS DURANTES AS SIMULAÇÕES DOS TRÊS CENÁRIOS.

O segundo parâmetro de desempenho relativo ao nível de congestionamento mede o *tempo perdido durante os congestionamentos*. Esta nova métrica visa avaliar o tempo perdido, dada a detecção de uma situação de congestionamento. Em suma, enquanto o primeiro parâmetro apresentado responsabiliza-se por identificar quando ocorrem congestionamentos, esse segundo parâmetro conta quando tempo cada *flit* teve de esperar até poder ser transmitido através da NoC.

Na Figura 5.13, o tempo perdido é quantificado em número de ciclos de relógio de operação, contados desde o instante em que um *flit* foi impedido de transmissão até o instante em que ele foi efetivamente inserido na NoC. Com exceção do algoritmo FF, todos os demais apresentaram um aumento nos congestionamentos à medida que a taxa de injeção cresce. Tomando a média de todos os 3 cenários foi possível alcançar uma redução de até 50% quando as heurísticas MMCL e MACL são empregadas, comparando ao algoritmo de referência *First Free*. No caso de NN e BN o ganho relativo é de até 82%.

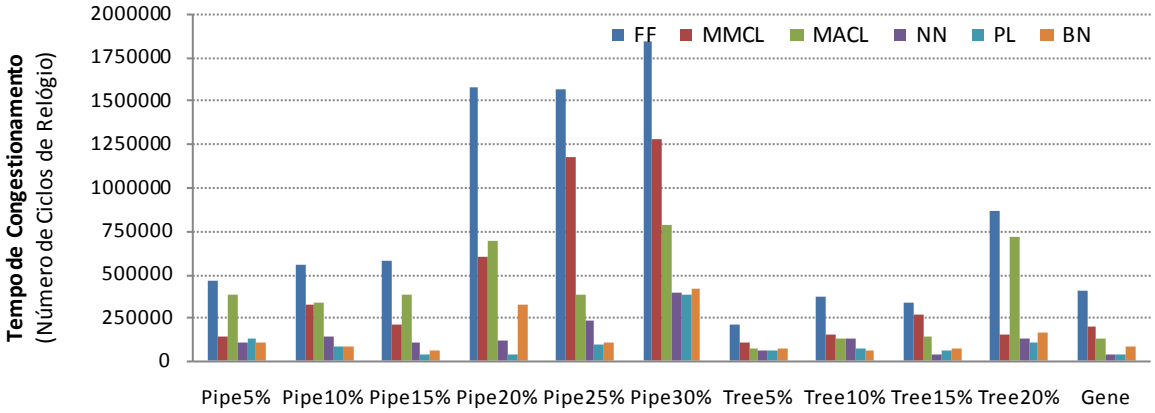


FIGURA 5.13. TEMPO PERDIDO DURANTE OS CONGESTIONAMENTOS PARA OS TRÊS CENÁRIOS DE SIMULAÇÃO.

A heurística *Path Load* apresenta um ganho relativo ao FF na ordem de 87%, e ainda com 30% de ganho comparado ao algoritmo *Nearest Neighbor*. Considerando apena os resultados do Cenário C, onde as aplicações são bastante diversificadas quanto à topologia, volumes e taxas de transmissão, o algoritmo *congestion-aware Path Load* permite reduzir o número de congestionamentos em 92%.

Os resultados de latência e congestionamentos são significativos. À medida que a ocupação dos canais da NoC foi reduzida, os pacotes podem ter menor latência de transmissão, e ainda os congestionamentos na rede podem ser reduzidos em até 90% como visto, cumprindo com o objetivo das heurísticas *congestion-aware*.

5.3.4. Tempo de Execução Total das Aplicações

Na presente Seção, discute-se o *tempo total de execução* de cada um dos experimentos. O objetivo aqui consiste em medir o impacto do emprego da heurística de mapeamento no tempo total de execução do sistema. Para essa avaliação, o volume de dados transmitidos nas simulações foi multiplicado 10 vezes. Nos experimentos anteriores, houve uma grande penalidade no tempo de execução devido à complexidade das heurísticas de mapeamento. Contudo, conforme apresentado na Tabela 5.2, pode-se perceber que à me-

dida que o volume de dados aumenta, o atraso inserido pelas heurísticas pode ser amortizado. Além disso, à medida que a taxa é aumentada, o tempo de execução total do experimento é reduzido. Isso acontece porque o volume de dados é conservado nos experimentos de cada cenário, apenas a taxa é alterada.

Como os *tempos de computação* das heurísticas MACL e MMCL são grandes, nos experimentos, o tempo total de execução apresentados quando essas heurísticas são empregadas foi ainda maior que o apresentado por *FF*. Contudo, o tempo de execução sofre em média uma penalidade de 0,3% para MMCL, e de 0,08% para o emprego de MACL.

TABELA 5.2. TEMPO DE EXECUÇÃO TOTAL DO SISTEMA (EM CICLOS DE RELÓGIO) PARA CADA SIMULAÇÃO REALIZADA.

Cenários	*Taxas*	FF	NN	MMCL	MACL	PL	BN
A. 20 aplicações com grafo Pipeline	*5%*	10354395	**10331940**	10422508	10385617	10381836	10337634
	10%	**5243932**	5268352	5349822	5299382	5285767	5261327
	15%	3631240	3581285	3617405	3542790	**3538877**	3650616
	20%	3206935	**2997971**	3159016	3323047	3064465	3094128
	25%	2812327	2700104	2818192	**2584887**	2595065	2724354
	30%	2695070	**2466313**	2507255	2664381	2568793	2710888
B. 20 aplicações com grafo Árvore	*5%*	9444023	**9441226**	9550022	9508053	9479805	9450054
	10%	4898498	**4882671**	4988130	4968061	4937693	4892786
	15%	3443958	**3409391**	3513331	3486856	3438662	3420585
	20%	3672616	**3540024**	3562495	3671807	3596780	3574463
C. 20 Aplicações do TGFF		3217520	3104678	3291088	3226601	3139558	**3084661**
Ganho Global relativo ao FF		-	***1,70%***	*-0,30%*	*-0,08%*	*1,13%*	*0,80%*

No caso dos demais algoritmos, foi possível obter uma redução no tempo total de execução. Na média de todas as simulações, as heurísticas PL e BN apresentam uma redução de aproximadamente 1% no tempo de execução, enquanto NN permite 1,7% de ganho comparado ao *First Free*. No Cenário C, o desempenho da heurística BN é melhor que as demais. Enquanto PL e NN apresentam respectivamente tempos de execução 2,4% e 3,5% menores que aquele apresentado por FF, *Best Neighbor* resulta em uma redução de 4,1%.

É importante notar que em todos os experimentos o tempo para mapear cada uma das tarefas foi considerado, assim com o tempo para configurá-la no sistema. De maneira contrária ao proposto por Kim [KIM05], Ngouanga [NGO06] e Wronski [WRO06], aqui não foi assumido que o tempo de execução das heurísticas pode ser desconsiderado. A Tabela 5.3 apresenta os tempos médios obtidos para cada algoritmo de mapeamento, bem como o tempo estimado para a configuração das tarefas no sistema.

TABELA 5.3. TEMPO DE EXECUÇÃO MÉDIO DOS ALGORITMOS IMPLEMENTADOS.

	Tempo (ciclos de relógio)	**Complexidade da Heurística** (MPSoC com dimensão x)
FF	20	$O(x^2)$
NN	15	$O(x^2)$
MMCL	1.000	$O(x^4)$
MACL	1.600	$O(x^4)$
PL	500	$O(x^3)$
BN	100	$O(x^2)$
Configuração HW	13.000	
Configuração SW	1.000	

A expectativa de que a redução dos congestionamentos implicasse na redução do tempo total de execução do sistema não se confirmou conforme esperado, visto que redução na ordem de 90% nos congestionamento, ainda assim permite que o tempo de execução seja penalizado. Nas primeiras simulações, a penalidade no tempo execução é em média de 8,8% para o algoritmo PL, e de 2,5% quando BN é empregado. Quando o volume de dados transmitidos nas simulações foi aumentado em 10 vezes, para o caso de BN o tempo de execução apresentado foi em 0,8% melhor que observado para FF. A heurística PL, por sua vez, apresentou um ganho de 1,13%, também comparado aquele obtido por FF. Assim, pode-se estimar que para volumes maiores a penalidade em termos de tempo de execução, e o tempo para mapear as tarefas será amortizada, de forma que a redução nos congestionamentos permita reduções significativas relativas ao tempo de execução.

O tempo de execução das heurísticas foi medido durante o processamento dos algoritmos através da execução da função 10 mil vezes consecutivas. Isso foi necessário porque as operações do sistema disponíveis para medição de tempo não fornecem a precisão necessária. Assim, executando por várias vezes o tempo médio de execução pode ser calculado com a precisão desejada. O atraso para configuração das tarefas foi baseado nos trabalhos de Möller e outros [MÖL06] [MÖL07]. Nesses trabalhos, os Autores investigam o problema de configuração dinâmica e parcial de módulos de hardware em sistemas que adotam NoCs como meio de comunicação. Esta etapa, que contou com a colaboração do Autor da Tese, foi essencial para o passo seguinte de modelagem do MPSoC.

5.4. Outras Considerações

A Tabela 5.4 apresenta um resumo dos resultados obtidos normalizados em função do algoritmo FF, considerando a média da simulação para os três cenários implementados. Pode-se notar que a heurística PL apresenta os melhores resultados na maioria dos os parâmetros avaliados, com exceção do tempo de execução total, onde NN se destaca.

No entanto, PL apresenta um tempo de execução muito próximo daquele apresentado por NN, visto que seu tempo médio para mapeamento de tarefa é de 500 ciclos de relógio, um tempo 33 vezes maior que o necessário pelo algoritmo NN. Os resultados apresentados apontam para uma redução significativa de quase 90% nos congestionamentos, de 31% na ocupação dos canais da NoC, e de 15% na latência dos pacotes.

TABELA 5.4. RESUMO DOS RESULTADOS DE CADA HEURÍSTICA, NORMALIZADOS EM FUNÇÃO DO ALGORITMO FF.

Parâmetros de Desempenho	***FF***	**NN**	**MMCL**	**MACL**	**PL**	**BN**
Ocupação dos Canais *(Média)*	*1,00*	0,70	0,86	0,85	**0,69**	0,70
Ocupação dos Canais *(Desvio Padrão)*	*1,00*	0,80	0,88	0,90	**0,78**	0,80
Latência dos Pacotes *(Média)*	*1,00*	0,85	0,94	0,92	**0,84**	0,85
Latência dos Pacotes *(Desvio Padrão)*	*1,00*	**0,66**	1,33	0,89	0,98	1,19
Congestionamentos *(Canais saturados)*	*1,00*	0,31	0,55	0,62	**0,23**	0,29
Congestionamentos *(Tempo perdido)*	*1,00*	0,17	0,53	0,47	**0,12**	0,17
Tempo Total de Execução *(1xVolume)*	*1,00*	**1,00**	1,25	1,14	1,09	1,03
Tempo Total de Execução *(10xVolume)*	*1,00*	**0,98**	1,00	1,00	0,99	0,99

Além do tempo de execução das heurísticas, outro fator que pode ter influenciado nos resultados de tempo total de execução diz respeito à modelagem adotada para as aplicações. Vale aqui ressaltar que as operações de escrita e leitura das tarefas foram ambas implementadas como *não-bloqueantes*. Se o recebimento das mensagens é não-bloqueante, então o efeito de um congestionamento sobre o tempo de execução das tarefas é menor que no caso onde a tarefa fica bloqueada esperando para enviar um dado pacote. O mesmo acontece no caso de um recebimento de um pacote. Infelizmente, essa possível falha foi detectada apenas ao final do trabalho. Sendo assim, o emprego de comunicações bloqueantes é uma sugestão de trabalho futuro para a obtenção de melhores resultados.

Além dos experimentos apresentados anteriormente, outras variações foram realizadas. A primeira delas aumenta o número de recursos dedicados ao mapeamento de tarefas iniciais das aplicações de 9 para ***15 PEs iniciais***. Conforme os resultados obtidos, o desempenho de todas as heurísticas piorou aproximadamente 40%. Em termos de *ocupação média dos canais*, os algoritmos NN, PL e BN apresentam ganhos com relação a FF de 15%, 14% e 16% respectivamente (valores anteriores eram em torno de 30%). A diferença entre os novos valores e os atuais pode ser obtida através da comparação com os valores relacionados na Tabela 5.4. O ganho em termos do *desvio padrão na ocupação* é de aproximadamente 9% (valores anteriores eram em torno de 20%), enquanto que a *latência dos pacotes* apresenta em média um ganho de 10% comparado ao FF (valores anteriores eram em torno de 15%). O nível de congestionamento também sofreu impacto semelhante, com redução de desempenho de 80% para 50% no *número de canais saturados*, e de 70% para 60% em termos de *tempo perdido em congestionamentos*.

No caso das simulações com 15 aplicações simultâneas executando sobre o MPSoC, os ganhos relativos a todos os parâmetros investigados são menores. Isso ocorre porque quanto mais aplicações compartilham o MPSoC, maior a tendência de que tais aplicações fiquem espalhadas sobre sua superfície. Assim sendo, independentemente do algoritmo empregado no mapeamento, se as possibilidades de mapeamento são reduzidas, então provavelmente os resultados obtidos serão similares.

Uma alternativa para melhorar um dado mapeamento das tarefas em tempo de execução consiste no emprego da estratégia de migração das tarefas. Em um determinado momento o mapeamento de uma dada tarefa não pode ser otimizado, por exemplo, porque só resta um recurso livre, e este se encontra distante da posição onde a tarefa mestre está alocada. Se algum tempo depois, um recurso mais próximo for liberado, a técnica de migração pode ser empregada para (re)mapear a tarefa que estava distante, neste novo recurso (mais próximo).

Uma segunda variação dos experimentos investiga o emprego das informações obtidas pelos ***monitores distribuídos*** na decisão de mapeamento. A expectativa de obter melhor desempenho não se confirmou. Nos experimentos, os resultados alternam valores melhores e piores que os obtidos nas simulações baseadas no esquema de monitor centralizado. Além disso, as variações entre os dois resultados foram mínimas. Uma causa identificada para tal comportamento consiste na modelagem das aplicações. Cada uma delas gera o tráfego a partir dos parâmetros de volume e taxa passados nos arquivos de configuração (modelagem RTL na Seção 3.7.3). Esse tráfego pode ser dito "bem comportado" porque representa exatamente as taxas e volumes passados nos arquivos. Assim, a ocupação medida nos monitores distribuídos é muito próxima daquela que foi passada nos pacotes REQUEST, a partir dos quais o monitor centralizado estima a ocupação dos canais.

Um possível experimento que pode resultar em uma melhor avaliação do uso dos monitores distribuídos consiste em inserir constantes de erro nas taxas informadas pelas aplicações. Assim, os valores informados não seriam exatos e as medições dos monitores distribuídos seriam mais precisas que aquelas estimadas pelo monitor centralizado. Esse experimento, no entanto, não foi realizado devido ao tempo restrito para defesa da tese. Assim sendo, sua realização torna-se uma sugestão para trabalho futuro.

6. AVALIAÇÃO DAS HEURÍSTICAS PROPOSTAS *VERSUS* MAPEAMENTO GLOBAL

A segunda parte dos experimentos realizados é discutida no presente Capítulo. Seu objetivo consiste em avaliar o desempenho dos algoritmos aqui implementados frente à estratégia global. No presente trabalho as tarefas são mapeadas uma por vez, refletindo uma abordagem gulosa, enquanto no *mapeamento global* todas as tarefas de uma dada aplicação são mapeadas de uma única vez. Enquanto algoritmos gulosos consideram apenas informação local sobre a comunicação entre as tarefas escrava (solicitada) e mestre (solicitante), a abordagem global vale-se de informação relativa a todas as comunicações da aplicação, incluindo a topologia do grafo que a representa. Dessa forma, o mapeamento global deve apresentar melhor desempenho em termos de ocupação dos canais, latência dos pacotes e congestionamentos na rede. Por outro lado, a abordagem gulosa deve consumir menor tempo de execução, tornando-se mais adequada para cenários dinâmicos.

Nesse Capítulo, investiga-se quais as penalidades relativas ao emprego da estratégia gulosa. Para isso, o desempenho das duas heurísticas que apresentaram melhores resultados nos experimentos anteriores (*i. e.* PL e BN) é comparado a outros dois algoritmos implementados por Marcon e outros em [MAR07]. Neste trabalho, o mapeamento das tarefas é feito de acordo com o modelo de mapeamento global discutido acima. Conforme a revisão do Capítulo 2, os Autores de [MAR07] pesquisam o mapeamento estático de tarefas em uma NoC. Trata-se de um trabalho amplamente divulgado conforme as publicações realizadas [MAR05a] [MAR05b] [MAR05d] [MAR08]. Sua principal contribuição consiste na modelagem do consumo de energia da NoC relativo ao volume de comunicação da aplicação, e ao mapeamento definido para suas tarefas. Além disso, a disponibilidade da ferramenta gráfica CAFES para o mapeamento de tarefas contribuiu para a escolha do trabalho como base das comparações.

O mapeamento realizado nas duas pesquisas visa diferentes parâmetros de desempenho. Enquanto Marcon e outros investigam métodos de mapeamento a fim de reduzir o consumo de energia do sistema como um todo, aqui os congestionamentos no sis-

tema são tidos como alvo. Como o volume de dados gerados pelas aplicações não são determinados pelo MPSoC, a redução do volume transmitido deve implicar a redução da distância (*i. e. hop*s) entre as tarefas comunicantes. Em ambos os trabalhos, portanto, a estratégia adotada para reduzir tanto a energia consumida na comunicação quanto os congestionamentos na NoC passa pela redução da distância entre as tarefas comunicantes.

Para comparar as estratégias de mapeamento é necessário portar o mapeamento realizado através do CAFES, para a plataforma de simulação aqui proposta. A estratégia adotada no presente trabalho baseia-se em um arquivo de mapeamento que informa ao processador gerente, durante a simulação, o mapeamento de cada uma das tarefas obtido no CAFES. Os parâmetros de desempenho empregados nesse grupo de experimentos incluem além daqueles utilizados nos experimentos anteriores, a energia total consumida na execução do sistema, obtida através das estimativas realizadas pela ferramenta CAFES. A arquitetura alvo desse segundo grupo de experimentos é a mesma descrita na avaliação apresentada no Capítulo anterior (Seção 5.1).

6.1. Algoritmos de Referência

Como dito, dois algoritmos proposto em [MAR07] são utilizados aqui. O primeiro deles, baseado no algoritmo ***Simulated Annealing*** **(SA)** [KIR83], utiliza dois laços aninhados. O laço externo implementa uma técnica de busca global, enquanto o interno realiza o refinamento local. Este otimiza o mapeamento inicial provido pelo laço externo. Para isso, o laço interno troca o mapeamento de duas tarefas escolhidas aleatoriamente, e computa o valor da energia consumida resultante. Os melhores valores são anotados. O laço externo aplica a troca do mapeamento de vários módulos aleatório de uma só vez.

O segundo algoritmo é baseado no método ***Tabu Search*** **(TS)** [GLO97]. Este também possui dois laços aninhados. O laço interno procura pares de tarefas cuja troca de posição resulta em uma menor energia consumida. Para isso, uma lista de possíveis trocas é gerada, sem entradas replicadas. As trocas são selecionadas aleatoriamente da lista, e aquelas que causam uma redução na energia são anotadas. A partir dessas trocas é criada uma lista denominada *lista tabu*, a qual é utilizada para atualizar o mapeamento atual.

Ambos os algoritmos TS e SA baseiam-se em um mapeamento inicial (semente), gerado aleatoriamente. Eles tentam melhorar o resultado (energia consumida) através de trocas aleatórias das posições iniciais das tarefas. A qualidade da semente deve influenciar na qualidade do resultado obtido, e no tempo necessário para obter a solução. A condição de parada depende de uma *temperatura*. Esta assume inicialmente um valor elevado, e após um número determinado de iterações é gradualmente diminuída. Assim, a chance de

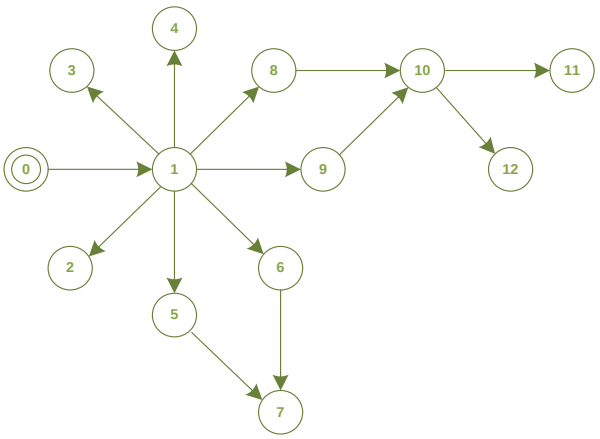
que seja selecionada uma solução em um mínimo local é reduzida. O método termina quando a temperatura estiver próxima de zero e nenhuma solução melhor for encontrada.

6.2. Cenários de Simulação

Duas restrições são impostas aos experimentos para suportar a comparação com a proposta de Marcon. Primeiramente, o MPSoC necessita ser homogêneo, ou seja, cada tarefa pode ser mapeada em qualquer elemento de processamento do sistema. No trabalho de Marcon foi investigado o mapeamento de núcleos IPs em uma NoC onde, cada um dos recursos representa um *tile*, o qual suporta o mapeamento de qualquer IP. Além disso, o número de IPs deve ser no máximo igual ao número de recursos disponíveis para mapeamento, visto que a natureza estática da solução de Marcon permite um mapeamento único das tarefas.

Quatro aplicações foram utilizadas como base para a geração dos grafos de aplicação empregados nos experimentos. Esta estratégia visa aproximar os cenários de simulação a um cenário real, onde as aplicações possuem topologias e também taxas aproximadas as de aplicações reais. Dentre elas constam três aplicações de vídeo: decodificadores MPEG-4 e VOP, MWD; e a Integral de Romberg. Além dessas, foram usadas nos experimentos outros 4 grafos selecionados do Cenário C (Figura 5.8).

O MPEG-4 é um padrão utilizado para compressão de dados digitais de áudio e vídeo. Ele foi definido pelo *Moving Picture Experts Group* (ou simplesmente MPEG), em 1998. No presente trabalho, o ***decodificador MPEG-4*** é representado pelo grafo da Figura 6.1, composto por 13 tarefas. Este decodificador foi empregado como estudo de caso em [VAN02] [MAR05d] [BER05], dentre outros. Conforme Murali e De Micheli [MUR04b], o grande volume de dados dessa aplicação pode ultrapassar a largura de banda disponível, tornando necessária a distribuição do tráfego através de diferentes caminhos.



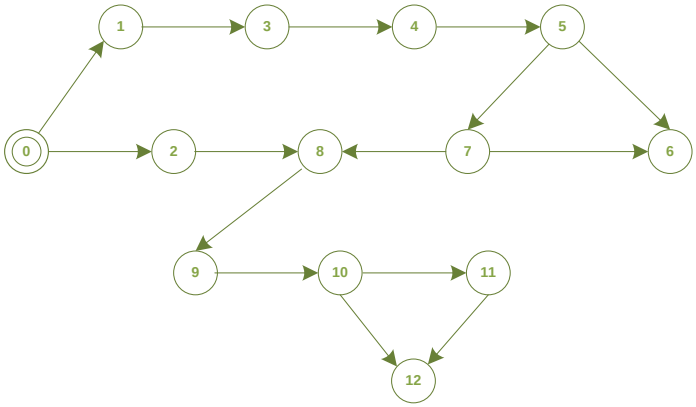
F [D, VFD, TFD, VDF, TDF]

0 [1, 10, 15%, 10, 1%]
1 [2, 10, 1%, 10, 1%]
1 [3, 100, 20%, 100, 2%]
1 [4, 10, 1%, 10, 1%]
1 [5, 600, 10%, 600, 10%]
1 [6, 100, 6%, 100, 6%]
1 [8, 100, 3%, 100, 3%]
1 [9, 400, 20%, 400, 20%]
5 [7, 100, 4%, 100, 4%]
6 [7, 100, 4%, 100, 4%]
8 [10, 200, 8%, 200, 8%]
9 [10, 450, 10%, 450, 10%]
10 [11, 680, 10%, 680, 10%]
10 [12, 240, 10%, 240, 10%]

FIGURA 6.1. GRAFO QUE REPRESENTA O DECODIFICADOR MPEG-4 COMPOSTO POR 13 TAREFAS.

Ao lado de cada grafo das Figuras, uma lista de comunicações é apresentada, onde cada uma das linhas representa uma dada aresta do grafo. A interpretação das linhas *F [D,VFD,TFD,VDF,TDF]* indica uma comunicação entre as tarefas *F* e *D* (*i. e.* fonte e destino), com volume *V* e taxa *T* de transmissão, em ambos os sentidos dados pelos índices *FD* e *DF* (*i. e.* fonte para destino e destino para fonte). Conforme apresentado na Seção 3.2, a qual discute a representação das aplicações, as comunicações podem ocorrer nos dois sentidos. Nesse caso, a indicação das arestas apenas informa qual a ordem parcial de mapeamento no sistema, em outras palavras, ela informa quem é a tarefa mestre da comunicação.

A segunda aplicação modelada, o ***decodificador VOP*** (do inglês, *Video Object Plane Decoder* ou VOPD) apresenta um alto nível de paralelismo entre suas tarefas. O decodificador requer alto desempenho e uma grande largura de banda para transmissões de vídeo em alta definição (*e. g.* HD). O grafo da Figura 6.2 representa a aplicação do decodificar VOP, composta por 13 tarefas. Esta aplicação foi empregada em vários trabalhos sobre mapeamento de tarefa, incluindo [VAN02] [MUR04a] [JAL04] [MAR05d] e [BER05].



F [D, VFD, TFD, VDF, TDF]

0 [1, 10, 1%, 10, 1%]
0 [2, 10, 1%, 10, 1%]
1 [3, 100, 7%, 10, 1%]
2 [8, 80, 2%, 10, 1%]
3 [4, 100, 36%, 10, 1%]
4 [5, 100, 36%, 10, 1%]
5 [6, 10, 1%, 100, 3%]
5 [7, 100, 36%, 100,15%]
6 [7, 10, 1%, 100, 5%]
7 [8, 100, 35%, 10, 1%]
8 [9, 100, 5%, 10, 1%]
9 [10, 100, 30%, 10, 1%]
10 [11, 100, 31%, 10, 1%]
10 [12, 10, 1%, 100, 4%]
11 [12, 100, 31%, 100, 9%]

FIGURA 6.2. GRAFO QUE REPRESENTA O DECODIFICADOR VOP COMPOSTO POR 13 TAREFAS.

A aplicação denominada ***Multi-Window Display*** (ou MWD) tem seu grafo exposto na Figura 6.3. Ele é composto por 12 vértices (ou tarefas). As taxas e volumes adotados são apresentados na listagem ao lado do grafo. Esta aplicação foi empregada nos estudos de caso dos trabalhos apresentados por Jalabert, Bertozzi e outros [JAL04] [BER05]. Contudo, nenhuma descrição da aplicação de suas tarefas foi encontrada nesses trabalhos ou mesmo em outros da literatura revisada.

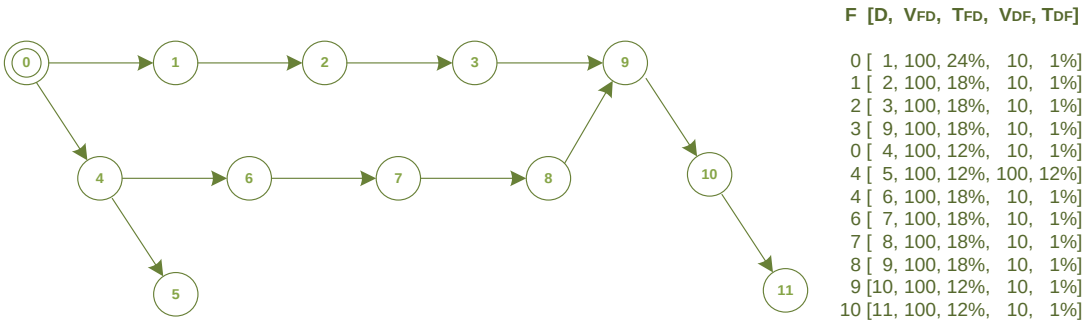


FIGURA 6.3. GRAFO QUE REPRESENTA A APLICAÇÃO MWD COMPOSTA POR 12 TAREFAS.

Romberg é um método baseado em aproximações sucessivas empregado no cálculo da integral de uma dada função [BUR04]. Conforme o grafo da Figura 6.4, a implementação desta aplicação sugere um fluxo de dados no formato de um triângulo retângulo. Embora um grafo com 10 tarefas seja adotado, o número de linhas empregadas no cálculo da ***Integral de Romberg*** (ou simplesmente RBERG) pode ser variado de acordo com a precisão desejada. Essa aplicação foi utilizada em [MAR05d].

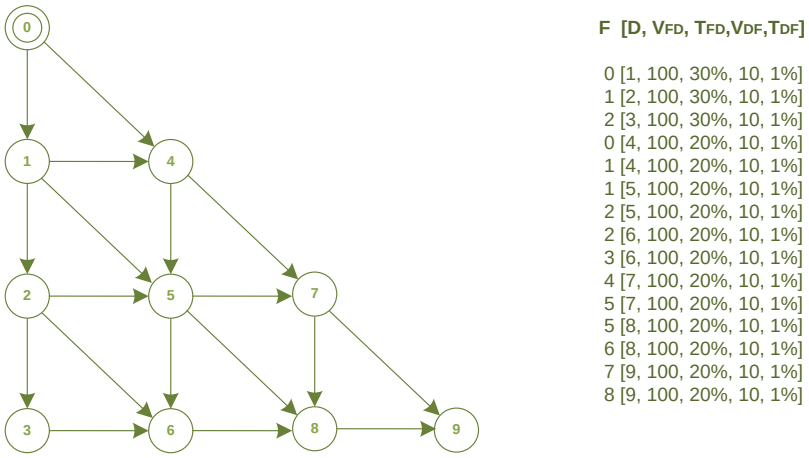


FIGURA 6.4. GRAFO QUE REPRESENTA A APLICAÇÃO DA INTEGRAL DE ROMBERG COMPOSTA POR 10 TAREFAS.

A modelagem do MPSoC alvo foi a mesma utilizada nos experimentos anteriores. Contudo, os recursos de software foram substituídos por recursos de hardware já que a estratégia de Marcon suporta apenas modelos onde todos os recursos são da mesma natureza. Um fator que permite a plena portabilidade dos estudos realizados por Marcon, incluindo sua modelagem de energia consumida consiste na NoC empregada. Tanto o trabalho aqui proposto, quanto o apresentado por Marcon empregam a NoC Hermes.

Dois cenários de experimentos foram investigados. São eles:

Cenário D. **MPSoC 5x4**: neste primeiro cenário foi empregado um MPSoC homogêneo com dimensões reduzidas, o qual suporta apenas o mapeamento individual das aplicações estudadas. Assim, pode-se avaliar os mapeamentos de acordo com a aplicação a ser mapeada, investigando qual o impacto no nível de conectividade das tarefas sobre o mapeamento, e ainda se os algoritmos gulosos apresentam resultados satisfatórios frente àqueles apresentado quando a ferramenta CAFES é usada.

Cenário E. **MPSoC 9x9:** neste cenário foi utilizado um MPSoC grande o suficiente para suportar o mapeamento de todos os 8 grafos de aplicações que compõem o experimento. Assim, espera-se avaliar como se dá o compartilhamento de recursos do MPSoC por várias aplicações, sendo utilizadas as heurísticas gulosas aqui propostas, bem como as apresentadas por Marcon.

6.3. Resultados para o MPSoC 5x4

Nessa Seção, discute-se os resultados dos experimentos realizados segundo os Cenários D e E. Para a avaliação da ocupação dos canais (Seção 6.3.2), da latência dos pacotes (Seção 6.3.3), do nível de congestionamento (Seção 6.3.4) e do tempo de execução (Seção 6.3.5), os mapeamentos segundo os algoritmos propostos por Marcon são realizado na ferramenta CAFES. Seguindo, os cenários são simulados no ambiente proposto no presente trabalho, baseado na NoC descrita em VHDL e nos PEs descritos em *SystemC*. Ao final, na Seção 6.3.7, apresenta-se um sumário dos resultados obtidos, bem como algumas considerações relevantes.

6.3.1. Distribuição das Tarefas

Nos experimentos do Cenário D, o mapeamento das tarefas segundo os algoritmos investigados resulta nas distribuições de tarefas exposta na Figura 6.5, onde *Ti* faz referência a tarefa *i* da aplicação. Na Figura, cada mapeamento possui um determinado *número total de hops*, dado pelo somatório das distâncias entre cada par de tarefas comunicantes. Cada distância é calculada conforme a posição das tarefas, baseando-se no algoritmo de roteamento xy. Por exemplo, para a comunicação entre as tarefas *T0* e *T1* da aplicação RBERG mapeada segundo o algoritmo *Tabu Search*, são necessários: 2 *hops* para a comunicação no sentido *T0-T1* (*i. e.* leste em *x* e norte em *y*); e 2 *hops* para a comunicação no sentido oposto (*i. e.* oeste em *x* e sul em *y*). No total são necessários 4 *hops*, portanto.

O mapeamento resultante para as heurísticas PL e BN foram idênticos para as quatro aplicações utilizadas. Comparando o *número de hops* necessários para realizar todas as comunicações, para todas as quatro aplicações o algoritmo *Simulated Annealing* apresentou o melhor resultado, seguido das heurísticas *congestion-aware* PL e BN. O pior desempenho foi anotado pelo algoritmo *Tabu Search* no que diz respeito ao número de *hop*s resultantes.

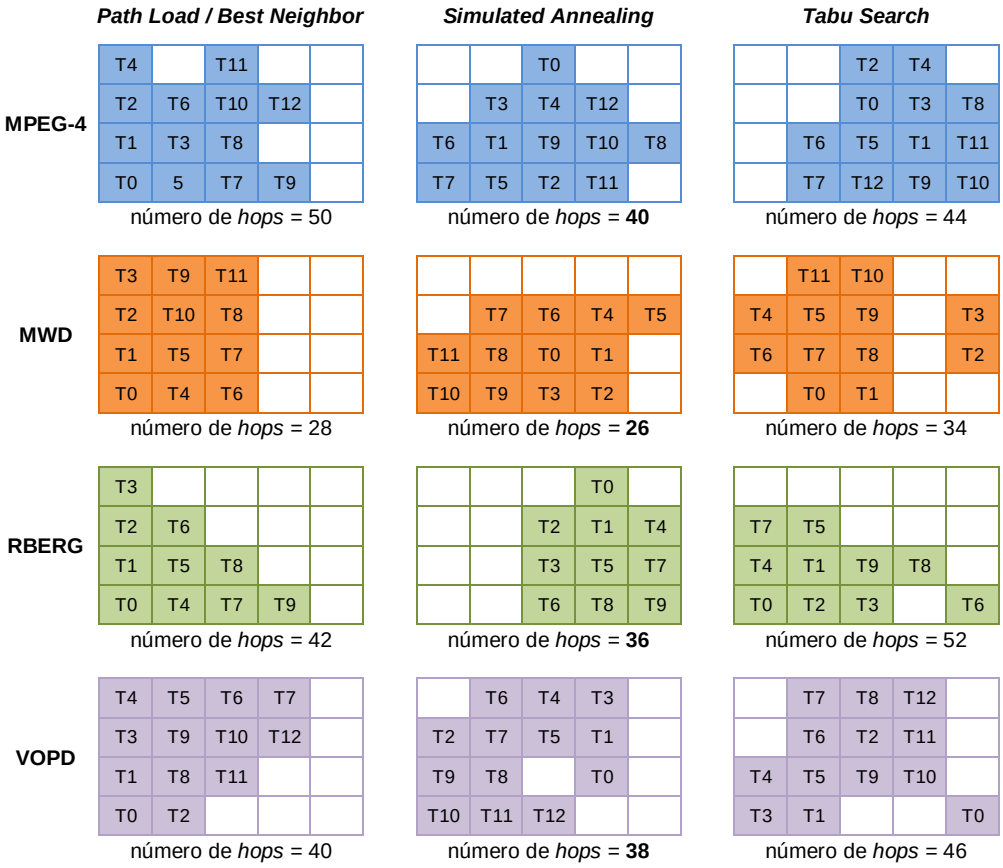


FIGURA 6.5. DISTRIBUIÇÃO DAS TAREFAS DE ACORDO COM OS ALGORITMOS DE MAPEAMENTOS INVESTIGADOS.

As heurísticas propostas obtiveram seu pior resultado para o mapeamento da aplicação MPEG-4, a qual possui uma tarefa que centraliza várias comunicações, ou seja, conectada a diversas outras tarefas (tarefa 1 no grafo da Figura 6.5). Esse caso expõe uma desvantagem no emprego de algoritmos gulosos. A visão local empregada por essa técnica não permite identificar que a tarefa *T1* necessita ser mapeada em uma posição central para que seja possível a alocação das suas escravas ao seu redor. O mesmo não acontece com o algoritmo *SA*, que possui conhecimento completo da topologia da aplicação.

A avaliação dos algoritmos apenas pelo número *hops* resultante não permite afirmar que um dado mapeamento será o melhor ou pior que outro em termos de congestionamentos ou energia consumida. No caso do congestionamento é preciso avaliar se as ta-

refas mapeadas mais próximas são realmente aquelas que demandam maior taxa de comunicação. No caso da energia, ao invés da taxa, o volume de comunicação ganha importância. Portanto, faz-se necessária uma avaliação mais aprofundada dos resultados, conforme apresentado a seguir.

6.3.2. Ocupação dos Canais da NoC

O resultado da ocupação dos canais, incluindo a média e o desvio padrão, para cada um dos algoritmos é apresentado nos gráficos da Figura 6.6. Embora os mapeamentos resultantes para o emprego de *Path Load* e *Best Neighbor* sejam idênticos (Figura 6.5), os valores obtidos para a ocupação dos canais não são os mesmos. Isso ocorre porque para cada heurística o tempo de mapeamento é diferente. Assim para a simulação relativa a cada algoritmo, num determinado instante idêntico para ambas as simulações, as tarefas alocadas no sistema não são necessariamente as mesmas.

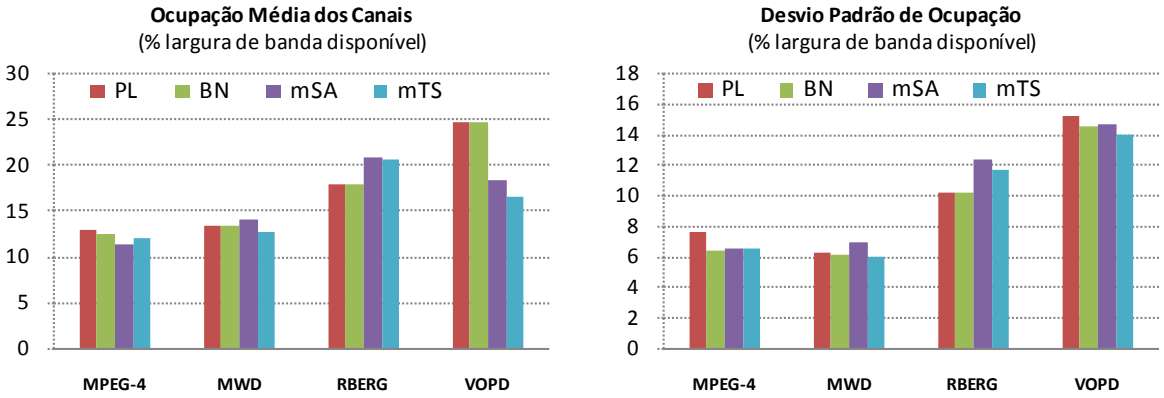


FIGURA 6.6. OCUPAÇÃO DOS CANAIS (MÉDIA E DESVIO PADRÃO) DA NOC PARA OS ALGORITMOS INVESTIGADOS.

Pode-se notar na Figura 6.6, que embora o algoritmo *Simulated Annealing* implementado por Marcon (*i. e.* mSA) tenha resultado em um número de *hops* menor para todas as aplicações, apenas para a MPEG-4 ele obteve o melhor desempenho para a ocupação dos canais, mesmo assim com pouca vantagem relativa aos demais algoritmos. Provavelmente, tal efeito ocorreu porque apenas nessa aplicação o algoritmo mSA teve um resultado muito melhor que os demais algoritmos.

Na aplicação da *Integral de Romberg*, os algoritmos gulosos PL e BN apresentaram melhores resultados. O algoritmo *Tabu Search* implementado por Marcon (*i. e.* mTS) apresentou a menor ocupação dos canais nas aplicações MWD e VOPD, mesmo resultando em um maior número de *hops* para tais aplicações. Os algoritmos *Path Load* e *Best Neighbor* não apresentaram bons resultados para a aplicação VOPD. Como pode ser visto na Figura 6.5,

o mapeamento da tarefa *T7* da aplicação VOPD não foi boa, porque os algoritmos gulosos realizaram a decisão com o conhecimento apenas da comunicação entre *T7* e *T5*, sem conhecer a relação dessa tarefa com as tarefas *T6* e *T8*. Logo em seguida, a tarefa *T7* requisita a tarefa *T8*, que será reusada, pois já se encontra alocada no MPSoC (se anteriormente requisitada por *T2*), em uma posição distante de *T7*.

6.3.3. Latência dos Pacotes Transmitidos

Nos gráficos da Figura 6.7, apresenta-se a latência dos pacotes de acordo com os mapeamentos obtidos para cada um dos algoritmos. A primeira constatação a ser realizada é a de que todos os algoritmos apresentaram resultados muitos semelhantes. Isso era esperado porque o MPSoC modelado neste experimento possui tamanho reduzido, limitando as distâncias entre as tarefas. Nesse caso, mesmo que o mapeamento realizado não seja o mais inteligente, as tarefas não serão mapeadas muito distantes da sua mestre.

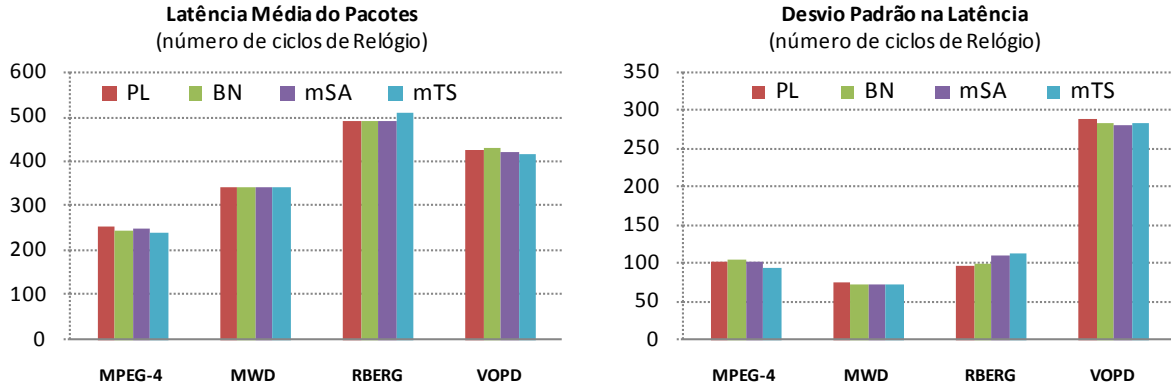


FIGURA 6.7. LATÊNCIA DOS PACOTES (MÉDIA E DESVIO PADRÃO) NA NOC PARA OS ALGORITMOS INVESTIGADOS.

Nos gráficos ainda é possível notar que a aplicação RBERG resultou nas maiores médias de latência. Este efeito acontece porque tal aplicação possui um alto grau de conectividade entre suas tarefas. Todas as suas tarefas estão conectadas a pelo menos outras duas tarefas, sendo que a tarefa *T5* (Figura 6.4), por exemplo, está conectada a seis tarefas. Com isso, a ocupação dos canais é alta, influindo diretamente sobre a latência dos pacotes.

Com relação aos desvios na latência, a aplicação VOPD apresenta os piores resultados dentre as aplicações investigadas. Como o desvio padrão alto não foi detectado apenas para o caso onde as heurísticas gulosas são aplicadas, então a causa desse efeito não pode ser atribuída ao mapeamento da tarefa T7. Assim sendo, este resultado deve ter sua justificativa baseada na natureza da aplicação, seja pela topologia do seu grafo ou pelas taxas e volumes aplicados.

6.3.4. Congestionamentos na NoC

Os gráficos da Figura 6.8 apresentam o nível de congestionamento na NoC, obtido para cada algoritmo investigado, incluindo o número de canais saturados e o tempo perdido durante todos os congestionamentos detectados. Em geral, os algoritmos mSA e mTS apresentaram melhores resultados de latência, embora não sejam *congestion-aware*.

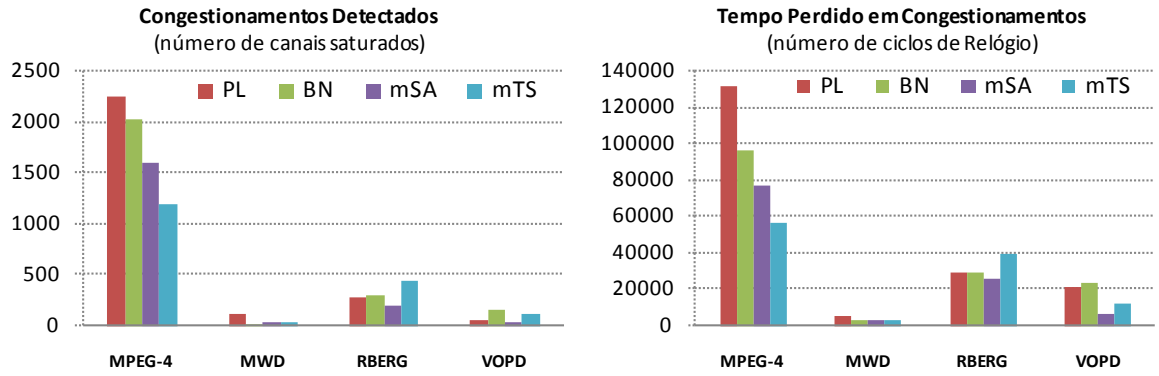


FIGURA 6.8. NÍVEL DE CONGESTIONAMENTOS NA NOC PARA OS ALGORITMOS INVESTIGADOS.

Dentre todas as aplicações, MPEG foi aquela com maiores valores de congestionamento. Uma possível justificativa para isso consiste no alto grau de conectividade da tarefa *T1* da aplicação (Figura 6.1). Além disso, MPEG apresenta a maior diferença de resultados entre as heurísticas propostas PL/BN e os algoritmos propostos por Marcon. A provável causa é também a tarefa *T1*. Quando ela é requisitada, o algoritmo guloso proposto define seu mapeamento baseado na comunicação entre ela e sua mestre, no caso a tarefas *T0*. Assim, quando a tarefas *T1* começa a requisitar suas escravas, faltam posições ao seu redor para posicionamento de todas as suas sete escravas.

6.3.5. Tempo de Execução Total das Aplicações

Aqui, discute-se o tempo total para execução de cada uma das aplicações, conforme o mapeamento resultante para cada um dos algoritmos investigados. Os valores plotados no gráfico da Figura 6.9 demonstram que os tempos de execução foram equivalentes para os diferentes mapeamentos realizados.

Cabe ressaltar que o mapeamento realizado segundo os algoritmos de Marcon foi aplicado nas simulações em tempo de projeto, ou seja, quando a simulação do ambiente utiliza ou o algoritmo mSA ou mTS, nenhum atraso é considerado para o mapeamento da tarefa, tanto para o seu mapeamento quanto a sua configuração. Enquanto isso, no caso das heurísticas gulosas, o mapeamento e configuração de uma tarefa consomem tempo.

Por exemplo, o algoritmo BN gasta em média 1.100 ciclos de relógio no mapeamento de uma única tarefa, sendo 100 ciclos para o mapeamento e 1.000 para a sua configuração no sistema (ver Tabela 5.3).

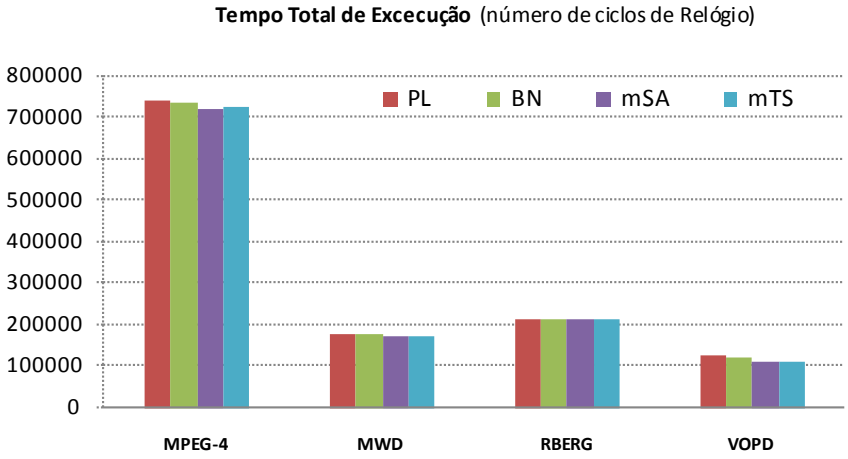


FIGURA 6.9. TEMPO DE EXECUÇÃO TOTAL PARA OS EXPERIMENTOS.

6.3.6. Energia Consumida

A presente seção discute a energia total consumida na NoC relativa às comunicações dado o mapeamento das tarefas. A estratégia adotada para estimativa da energia é baseada na ferramenta CAFES. Para isso, os mapeamentos resultantes das heurísticas *Path Load* e *Best Neighbor* foram anotados e aplicados na ferramenta CAFES. O modelo adotado para representação das comunicações é o CWM (ou *Communicating Weighted Model*). A discussão do modelo de energia adotado é apresentada a seguir, de maneira simplificada. Para maiores detalhes remete-se o leitor à [MAR05b].

Dada a transmissão de um *bit* através da NoC, a energia dinâmica consumida para tal transmissão pode ser calculada através da Equação 6.1. O consumo total é composto pelo conjunto de três parcelas de energia, referentes à energia $E_R bit$ consumida para roteamento do *bit*; a energia $E_L bit$ consumida pela sua transmissão através de um dado enlace da NoC; e a energia $E_C bit$ da transmissão no enlace local, entre PE e roteador.

EQUAÇÃO 6.1. $$Ebit = E_R bit + E_L bit + E_C bit$$

A energia de roteamento $E_R bit$ pode ser decomposta na energia consumida para seu armazenamento no buffer interno do roteador $E_B bit$, e na energia consumida no seu chaveamento $E_S bit$ para a porta de destino. A energia $E_L bit$ para transmissão através do enlace também pode ser decomposta em duas parcelas, uma primeira relativa à transmissão através de enlaces horizontais ($E_{LH} bit$), e a segunda relativa aos verticais ($E_{LV} bit$). Seguindo o exemplo de [MAR05b], para simplificar a modelagem, pode-se assumir uma

NoC quadrada onde ambas as parcelas são idênticas, ou seja, $E_{LV}bit = E_{LH}bit$. Além disso, a energia consumida no enlace local também é assumida como insignificante ($E_Cbit = 0$).

Dadas as afirmações anteriores, a energia consumida na transmissão de um *bit* entre duas tarefas mapeadas nos recursos *Ri* e *Rj* se dá através da Equação 6.2. Nessa equação, *NR* representa o número de roteadores intermediários necessários para a transmissão. Ele indica a distância entre as tarefas, definida pelas suas posições e ainda pelo algoritmo de roteamento adotado pela NoC.

EQUAÇÃO 6.2. $$Ebit_{Ri,Rj} = NR_{Ri,Rj} \times (E_B bit + E_S bit) + (NR_{Ri,Rj} - 1) \times E_L bit$$

Em geral, a comunicação entre duas tarefas contém vários *bits*. Para obter a energia total da comunicação E_{AB} entre duas tarefas *TA* e *TB*, a Equação 6.3 multiplica a energia de transmissão de um único *bit* pelo volume total de bits V_{AB} a serem transmitidos entre tais tarefas.

EQUAÇÃO 6.3. $$E_{AB} = V_{AB} \times Ebit_{Ri,Rj}$$

Por fim, para obter a energia dinâmica total relativa às comunicações entre todas as tarefas do sistema é necessário aplicar o somatório da energia para cada par de tarefas comunicantes, de acordo com Equação 6.4. Nessa equação, *CT* representa o conjunto de todas as tarefas que compõem o sistema. Esse conjunto contém informação sobre os volumes de comunicação.

EQUAÇÃO 6.4. $$E_{DyNoC} = \sum_{(A,B)} E_{AB} \quad \forall\ A,B \in CT,\ com\ V_{AB} > 0$$

Vista a modelagem de energia consumida, seguem os resultados. A Figura 6.10 apresenta os valores de energia consumida na comunicação resultante para os mapeamentos obtidos a partir do emprego de cada algoritmo investigado. Conforme esperado, o desempenho do algoritmo mSA é o melhor. Contudo, com exceção da aplicação MPEG-4, as demais heurísticas *congestion-aware* apresentaram resultados bons, muito próximos daqueles obtidos por mSA e mTS.

Nas aplicações RBERG e MWD, as heurísticas gulosas apresentaram resultados melhores que os obtidos com o algoritmo mTS. No caso da aplicação MPEG-4, o resultado das heurísticas propostas foi aproximadamente 40% pior. Como mencionado anteriormente, o problema ocorrido no caso das heurísticas gulosas foi o mapeamento da tarefa *T1* da aplicação MPEG-4. Como pode ser notado, seu efeito foi exercido em todos os parâmetros de desempenho investigados.

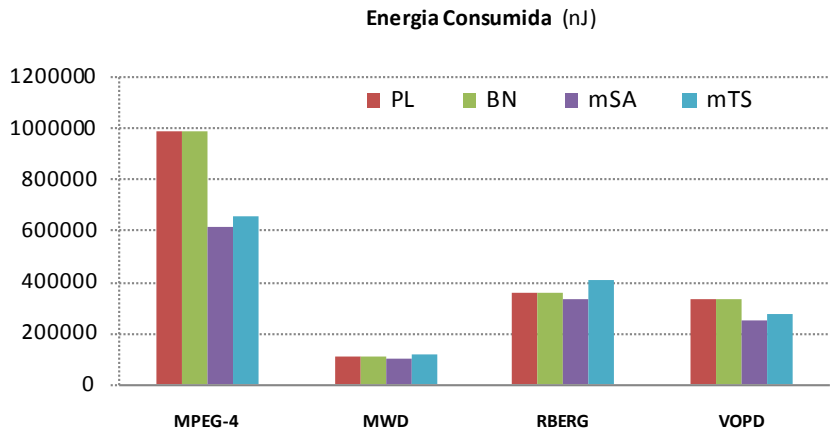


FIGURA 6.10. ENERGIA DINÂMICA DE COMUNICAÇÃO PARA OS MAPEAMENTOS REALIZADOS.

6.3.7. Discussão dos Resultados

A Tabela 6.1 apresenta para cada uma das heurísticas, o resultado obtido normalizado em função do algoritmo mSA, o qual apresentou, em média, a melhor solução. Em duas métricas importantes, latência dos pacotes e tempo de execução, as heurísticas propostas apresentaram uma perda mínima de desempenho na ordem de 1% e 4%, respectivamente. O tempo de execução é penalizado pelo tempo gasto pelos algoritmos de mapeamento, já que no emprego de mSA e mTS não foram inseridos quaisquer atrasos.

TABELA 6.1. RESUMO DOS RESULTADOS PARA O CENÁRIO D, NORMALIZADOS EM FUNÇÃO DO ALGORITMO MSA.

Parâmetros de Desempenho	**PL**	**BN**	***mSA***	**mTS**
Ocupação dos Canais *(Média)*	1,07	1,06	*1,00*	**0,96**
Ocupação dos Canais *(Desvio Padrão)*	0,97	**0,92**	*1,00*	0,94
Latência dos Pacotes *(Média)*	**1,01**	**1,01**	*1,00*	**1,01**
Latência dos Pacotes *(Desvio Padrão)*	**0,99**	**0,99**	*1,00*	1,00
Congestionamentos *(Canais saturados)*	1,45	1,34	*1,00*	**0,95**
Congestionamentos *(Tempo perdido)*	1,66	1,35	*1,00*	**0,98**
Número de hops Total	**1,14**	**1,14**	*1,00*	1,26
Tempo Total de Execução	1,04	1,03	*1,00*	**1,01**
Energia Consumida na Comunicação *(CAFES)*	1,38	1,38	*1,00*	**1,11**

Com relação ao nível de congestionamento, PL e BN sofreram uma penalidade relativamente alta, na ordem de até 45% no número de canais saturados, e de até 66% no tempo perdido devido a congestionamentos. Este resultado não reflete o esperado, já que as heurísticas propostas são orientadas ao congestionamento. Uma possível causa para esse efeito reside na estratégia gulosa, que se vale de informação parcial para o mapeamento das tarefas, uma a uma.

Por outro lado, como esperado as heurísticas propostas por Marcon, apresentam os melhores resultados em termos de energia consumida na comunicação. Neste parâmetro, PL e BN apresentam um consumo de energia 38% pior que o apresentado por mSA. O valor calculado para ambas as heurísticas é idêntico porque a estimativa de energia é realizada pela ferramenta CAFES em seu modelo CWM, que estima a energia com base no mapeamento das tarefas, sem considerar a natureza dinâmica das aplicações.

O emprego de um mapeamento global de aplicações pode comprometer o desempenho de sistemas dinâmicos, devido a sua complexidade. Além disso, ele exige um número de recursos livres suficiente para mapear toda a aplicação. Enquanto isso, as heurísticas propostas são rápidas, e ainda permitem que uma aplicação seja mapeada parcialmente, mesmo que ainda não existam recursos para todas as suas tarefas. Adicionalmente, as tarefas da aplicação podem começar a executar, até mesmo antes do mapeamento completo da aplicação. Nesse último caso, apenas as tarefas que necessitam executar num dado instante estarão mapeadas no MPSoC, evitando assim desperdício de recursos.

Os valores obtidos levam ao questionamento de qual estratégia poderia ser empregada para reduzir os congestionamentos no sistema, se as heurísticas *congestion-aware* não forem efetivas para um determinado caso. Uma alternativa sugere o uso da técnica de migração de tarefas. Tal estratégia é muito empregada visando a distribuição de carga nos processadores do sistema. No entanto, seu emprego pode ser adaptado para um cenário no qual a migração das tarefas é realizada para corrigir o mapeamento de uma dada tarefa, quando é detectado que sua posição está gerando congestionamentos no sistema. Como a migração de tarefas não faz parte do escopo do presente trabalho, uma sugestão para trabalhos futuros consiste na investigação do emprego desta técnica.

6.4. Resultados para o MPSoC 9x9

O Cenário E investiga o desempenho dos algoritmos quando um MPSoC maior é empregado. Agora, os recursos devem ser compartilhados por mais de uma aplicação. Além das aplicações utilizadas nos experimentos do Cenário D (*i. e.* MPEG-4, MWD, RBERG e VOPD), aqui outras quatro aplicações sintéticas do Cenário C (Figura 5.8) são empregadas para que o número total de tarefas esgote os recursos disponíveis.

Os diagramas expostos na Figura 6.11 representam cada um dos mapeamentos obtidos para as 8 aplicações. No caso do mapeamento obtido pelas heurísticas gulosas, o número de *hops* resultantes foi menor que o obtido pelos algoritmos estocásticos. Os resultados indicam uma redução de 20 saltos (*hops*) com relação ao mapeamento obtido através do algoritmo mTS, e de até 190 saltos comparado ao mapeamento de mSA. O algoritmo

mSA apresentou os melhores resultados nos experimentos do Cenário D, mas agora, quando o MPSoC necessita ser compartilhado, o desempenho foi o pior de todos, pelo menos em termos de número total de *hops*. Logicamente, assim com nos experimentos anteriores, outras métricas necessitam ser avaliadas para permitir conclusões mais precisas do desempenho dos algoritmos. Essas serão discutidas mais adiante no texto.

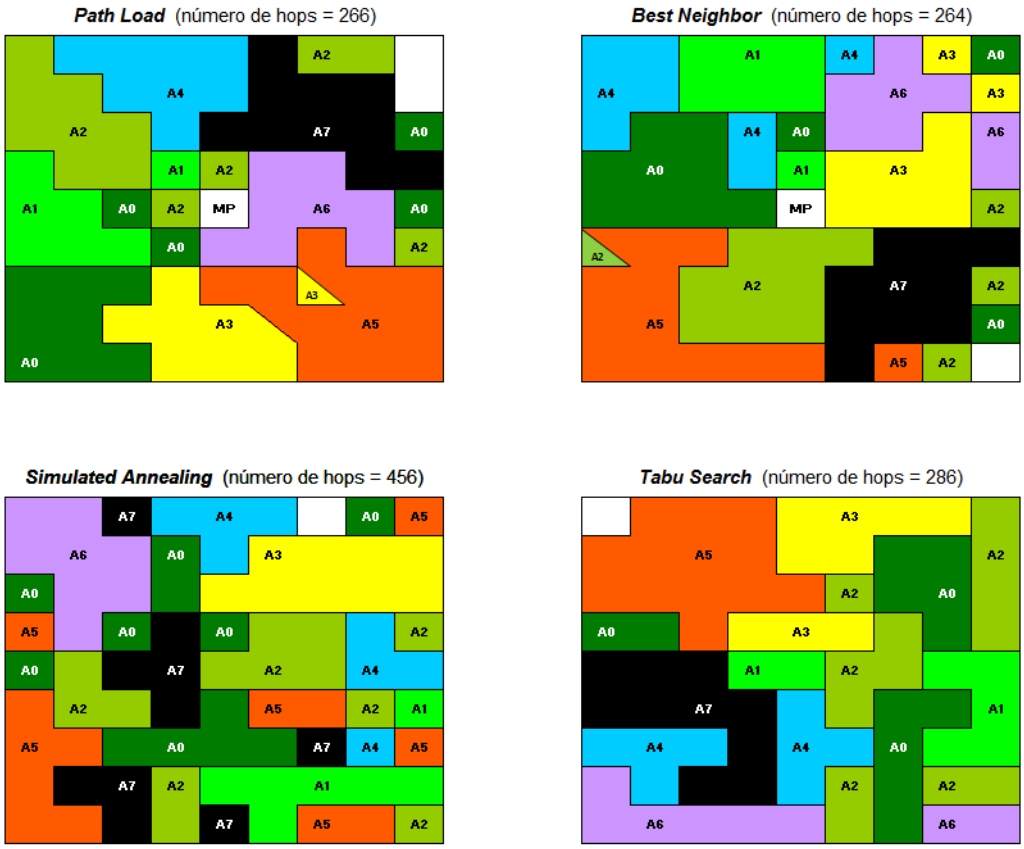


FIGURA 6.11. DISTRIBUIÇÃO DAS TAREFAS PARA O MAPEAMENTO SOB UM MPSOC 9X9.

Ao analisar separadamente cada mapa de tarefas da Figura 6.11, pode-se perceber que no mapeamento resultante de mSA, os blocos contínuos são menores que nos demais mapeamentos. Esses blocos agrupam tarefas de uma mesma aplicação. Quando maiores os blocos, significa que as tarefas de uma mesma aplicação estão mais próximas, e ainda que as comunicações entre as tarefas de aplicações diferentes irão compartilhar menos canais. Isso possivelmente levará a uma redução significativa nos congestionamentos resultantes no sistema, conforme discutido na Seção 4.2 deste documento.

O mapeamento mSA possui blocos pequenos, e apenas duas aplicações foram mapeadas na forma de um único bloco (*i. e. A3* e *A6*). No caso do mapeamento mTS, o tamanho dos blocos aumentou, mas ainda apenas duas aplicações foram mapeadas em um bloco contínuo (*i. e. A5* e *A7*). Para o mapeamento BN, apenas a aplicação *A7* encontra-se

em um bloco contínuo. Nesse mapeamento pode-se perceber um triangulo retângulo. Ele indica que o recurso sob o triangulo foi reusado durante a simulação. Em um dado instante, naquele recurso foi mapeada uma tarefas da aplicação *A5*, e em outro fora mapeada uma tarefa da aplicação *A2*. No mapeamento PL, são dois os triângulos, nesse caso compartilhados por tarefas das aplicações *A3* e *A5*. Quando as tarefas da aplicação *A3* estão mapeadas forma-se um bloco contínuo, o mesmo acontece para a aplicação *A5*. Assim, PL permite o mapeamento de 5 aplicações em um bloco contínuo (*i. e. A3*, *A4*, *A5*, *A6* e *A7)*.

Na Figura 6.11, existem blocos sem nenhum identificador, esses são blocos não utilizados. No caso dos mapeamentos mSA e mTS apenas em um recurso nenhuma tarefa é alocada. No mapeamento BN, dois recursos não receberam tarefas, mas um deles foi utilizado para o mapeamento do processador gerente MP. No caso do algoritmo PL, são dois os recursos livres, e ainda um recurso dedicado ao MP.

A Tabela 6.2 apresenta o resumo de todos os parâmetros de desempenho investigados, para os algoritmos empregados sobre o Cenário E. Os valores da tabela estão normalizados em função dos apresentados pelo algoritmo mTS. Veja que para a maioria dos parâmetros os melhores resultados foram obtidos a partir do emprego do algoritmo mTS. Em termos de ocupação dos canais e latência dos pacotes, os resultados do mapeamento PL são próximos daqueles obtidos com mTS. Enquanto isso, para o caso dos algoritmos BN e mSA, os resultados são ainda piores que os apresentados por PL.

TABELA 6.2. RESUMO DOS RESULTADOS PARA O CENÁRIO E, NORMALIZADOS EM FUNÇÃO DO ALGORITMO MTS.

Parâmetros de Desempenho	PL	BN	mSA	mTS
Ocupação dos Canais *(Média)*	**1,07**	1,13	1,13	*1,00*
Ocupação dos Canais *(Desvio Padrão)*	**1,18**	1,27	1,36	*1,00*
Latência dos Pacotes *(Média)*	**1,07**	1,10	1,16	*1,00*
Latência dos Pacotes *(Desvio Padrão)*	**1,03**	1,11	1,05	*1,00*
Congestionamentos *(Canais saturados)*	**1,90**	2,03	2,63	*1,00*
Congestionamentos *(Tempo perdido)*	**2,48**	3,17	4,58	*1,00*
Número de hops Total	0,93	**0,92**	1,59	*1,00*
Tempo Total de Execução	1,04	1,03	**1,02**	*1,00*
Energia Consumida na Comunicação *(CAFES)*	1,18	**1,13**	1,35	*1,00*

A avaliação dos congestionamentos apresenta ainda melhores resultados para o algoritmo mTS, com uma redução acima de 90% no nível de congestionamento obtido pela heurística *congestion-aware* PL. Esse efeito se da devido a abordagem de visão local adotadas nos algoritmos gulosos. Entretanto, não era esperada uma diferença tão grande nos resultados para congestionamentos.

Com relação ao tempo de execução, conforme esperado, os algoritmos gulosos apresentaram tempos maiores de execução para o sistema completo. É importante relembrar que também nesses experimentos os atrasos de mapeamento e configuração são considerados em PL e BN, enquanto que em mSA e mTS eles são nulos. Assim sendo, uma penalidade de 4% no tempo de execução dos algoritmos gulosos não é significativa, visto que os tempos de mapeamento para mSA e mTS devem ser grandes, embora não tenham sido medidos no presente trabalho. Essa medição exige uma manipulação de código fonte da ferramenta CAFES, atividade que se encontra fora do escopo do trabalho.

Em termos de energia consumida na comunicação, os algoritmos *energy-aware* não apresentaram resultados semelhantes. Enquanto mTS apresentou um valor baixo para consumo de energia, mSA apresentou o pior resultado. Deve-se lembrar que nos experimentos do Cenário D, o algoritmo mSA apresentou os melhores resultados, mas no Cenário E seus resultados foram os piores dentre os quatro mapeamentos obtidos.

6.5. Outras Considerações

A vantagem do método guloso é que ele deve consumir um tempo menor para seu processamento. Entretanto, seu resultado pode ser pior porque a decisão tomada para o mapeamento de uma dada tarefa deve ser a melhor dada uma visão local da aplicação. No futuro dessa aplicação, talvez essa decisão implique no mau mapeamento da próxima tarefa requisitada. Por exemplo, quando uma tarefa possui duas escravas e ela solicita primeiramente a escrava com quem necessita se comunicar "menos", então pode acontecer do melhor recurso ser alocado para essa primeira tarefa solicitada, restando um recurso "pior" para a segunda escrava. No caso do mapeamento global da aplicação, provavelmente este problema não ocorreria.

Mesmo existindo o problema da visão local, os resultados obtidos para os algoritmos gulosos foram próximos daqueles apresentados quando os algoritmos mSA e mTS são empregados, em alguns casos são até melhores como pode ser visto no Cenário E.

Como os algoritmos propostos por Marcon são baseados em sementes geradas aleatoriamente e ainda em parâmetros de temperatura que determinam quando o mapeamento satisfatório é encontrado, pode ser que os parâmetros do algoritmo mSA não sejam os mais adequados para o Cenário E, mas sim para o Cenário D. O caso do algoritmo mTS deve ser exatamente o contrário, sendo que seus parâmetros sejam mais adequados para o último cenário. Entretanto não foi possível a investigação do efeito dos parâmetros sobre os resultados dos algoritmos, pois tais parâmetros não estão acessíveis na interface da ferramenta CAFES.

7. CONCLUSÕES E TRABALHOS FUTUROS

No presente Capítulo, primeiramente apresenta-se uma relação das contribuições do trabalho, na Seção 7.1. Em seguida, apresenta-se um conjunto de conclusões (Seção 7.2), e apresenta-se algumas sugestões de trabalhos futuros na área de mapeamento dinâmico de tarefas (Seção 7.3).

7.1. Contribuições do Trabalho

Dentre as contribuições do trabalho constam:

Revisão do estado da arte – A primeira contribuição do trabalho consiste na investigação do estado da arte relacionado a MPSoCs. Para isso, são discutidas algumas propostas de MPSoCs comerciais e acadêmicos. No que diz respeito ao alvo desta pesquisa, o mapeamento de tarefas, são investigadas tanto abordagens estáticas (mapeamento definido em tempo de projeto), quando dinâmicas (mapeamento definido em tempo de execução), conforme apresentado no Capítulo 2.

Modelo de MPSoC – A segunda contribuição consiste na modelagem de uma organização de MPSoC heterogêneo baseado em NoC. Este suporta a execução tanto de tarefas de software, quando de tarefas de hardware. Enquanto as primeiras são mapeadas em processadores anexos aos roteadores da NoC, as tarefas de hardware podem ser carregadas em áreas configuráveis (*e. g.* FPGAs) embarcadas no sistema. Dentre os processadores do sistema, um foi selecionado para realizar tarefas de controle, tais como escalonamento, mapeamento e configuração de tarefas, e ainda a manutenção de estruturas que mantêm informação sobre a ocupação dos recursos (NoC e PEs) do sistema. Essa parte do trabalho compõe o Capítulo 3. A modelagem apresentada baseia-se no nível RTL, com a descrição VHDL da NoC Hermes, e com os elementos de processamento representados por *threads* descritas em *SystemC*.

Modelo de aplicação – A terceira contribuição consiste na modelagem das aplicações. Conforme o Capítulo 3, uma dada aplicação é representada aqui através de um grafo, que indica seu conjunto de tarefas, e as conexões entre as mesmas. Cada conexão contém informação sobre os volumes e taxas de comunicação entre as tarefas conectadas. Tais valores expressam estimativas, nas quais os algoritmos de mapeamento propostos baseiam sua decisão. Completando a descrição do sistema, foi definido o protocolo de comunicação entre as tarefas, baseado principalmente nos pacotes de controle, enviados ao processador gerente para *requisitar* uma dada tarefa, ou para *liberar* determinado PE.

Proposta de heurísticas *congestion-aware* – A principal contribuição do trabalho consiste na proposta de quatro heurísticas para o mapeamento de tarefas. Tal mapeamento é realizado em tempo de execução, sob demanda da aplicação. Os algoritmos implementados baseiam-se na ocupação dos canais da NoC para decidir o melhor mapeamento para a tarefa. Dessa forma, tais heurísticas visam reduzir o nível de congestionamento na NoC, assim evitando perda de desempenho nas aplicações. Dentre as heurísticas propostas no Capítulo 4, constam: MMCL, MACL, PL e BN. MMCL emprega como função custo a *ocupação máxima* em todos os canais da NoC, enquanto MACL avalia a *ocupação média* para todos os canais. PL e BN são heurísticas que avaliam a ocupação apenas considerando os *canais que pertencem ao caminho de comunicação* entre as tarefas mestre (que solicitou) e escrava (solicitada).

Uso de monitores – O emprego de monitores nos canais da NoC foi outra contribuição do trabalho. Esses módulos, inseridos nas portas dos roteadores da NoC, permitem a medição da ocupação real dos canais da NoC, em tempo de execução. Além disso, os monitores são empregados nas avaliações relacionadas à ocupação dos canais da NoC, e ainda na medição do nível de congestionamentos no sistema.

Estudo da complexidade dos algoritmos – Foi apresentado um estudo da complexidade dos algoritmos implementados, que permitem obter uma estimativa da curva de crescimento do tempo para obter um dado mapeamento, de acordo com o algoritmo implementado e com as dimensões do MPSoC alvo.

Avaliação das heurísticas – A avaliação das heurísticas apresentada no Capítulo 5 é outra contribuição. Ela permite a comparação entre os algoritmos implementados. Para isso, são considerados parâmetros de desempenho que incluem a ocupação dos canais da NoC, a latência resultante dos pacotes transmitidos, o nível de congestionamento na NoC e o tempo total de execução do sistema. Na segunda parte da avaliação (Capítulo 6), investiga-se o emprego dos algoritmos propostos, os quais aplicam uma abordagem gulosa (mapeiam uma tarefa por vez), frente os algoritmos que mapeiam a aplicação completa.

Publicações – O desenvolvimento do trabalho resultou nas publicações listadas abaixo. Dentre elas, as três primeiras relativas ao trabalho realizado logo no início do Doutorado, cujo objetivo consiste em investigar o comportamento de sistemas dinâmica e parcialmente configuráveis. As três seguintes apresentam a modelagem TLM desenvolvida, enquanto o último trabalho discute a modelagem RTL do MPSoC proposto. Outras submissões estão previstas ao final da escrita do volume.

1. Moraes, F.; Calazans, N.; Möller, L.; Brião, E.; Carvalho, E. **Dynamic and Partial Reconfiguration in FPGA SoCs: Requirements and Tools**. In: New Algorithms, Architectures and Applications for Reconfigurable Computing. Rosenstiel, W.; Lysaght, P. (Org.). Springer Verlag, 2005. ISBN: 1-4020-3127-0.

2. Möller, L.; Soares, R.; Carvalho, E.; Grehs, I.; Calazans, N.; Moraes, F. **Infrastructure for Dynamic Reconfigurable Systems: Choices and Trade-offs**. In: 19th Symposium on Integrated Circuits and System Design (SBCCI). Ouro Preto, MG, Brasil. Agosto, 2006. pp.44-49.

3. Möller, L.; Grehs, I.; Carvalho, E.; Soares, R.; Calazans, N.; Moraes, F. **A NoC-based Infrastructure to Enable Dynamic Self Reconfigurable Systems**. In: Reconfigurable Communication-centric SoCs (ReCoSoC). Montpellier, França. Junho, 2007. pp.23-30.

4. Carvalho, E.; Moraes, F. **Congestion-aware Task Mapping in Heterogeneous MPSoCs.** In: International Symposium on System-on-Chip (SoC). Tampere, Finlândia. Novembro, 2008. pp.-.

5. Carvalho, E.; Calazans, N.; Moraes, F. **Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs.** In: 18th Annual IEEE/IFIP International Workshop on Rapid Systems Prototyping (RSP). Porto Alegre, RS, Brasil. Maio, 2007. pp.34-40.

6. Carvalho, E.; Calazans, N.; Moraes, F. **Congestion-Aware Task Mapping in NoC-based MPSoCs with Dynamic Workload.** In: IEEE Computer Society Annual Symposium on VLSI 2007 (ISVLSI). Porto Alegre, RS, Brasil. Maio, 2007. pp.459-460.

7. Carvalho, E.; Moraes, F. **Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs.** In: XXII South Symposium on Microelectronics (SIM). Porto Alegre, RS, Brasil. Maio, 2007. pp.147-150.

7.2. Conclusões

No que diz respeito ao cenário atual em MPSoCs, pode-se concluir que, em geral, as propostas tanto acadêmicas quanto industriais são baseadas em MPSoCs homogêneos [TIL07] [HAL06] [WOS07] [LIN05]. Estes são mais adequados para operações de migração de tarefas, por exemplo. Contudo, o emprego de MPSoCs heterogêneos torna-se importante para suportar uma grande variedade de aplicações de um dado domínio.

Com relação à infra-estrutura de comunicação, a maioria dos trabalhos investigados adota NoCs [KUM02] [ZEF03] [MOR04] [COR06]. Seu emprego torna-se imprescindível em CIs de grande porte frente às limitações impostas pelos barramentos, principalmente relativas à baixa escalabilidade e ao pouco paralelismo suportado na comunicação.

Hoje, existem alguns MPSoCs industriais, como os propostos pela Intel [VAN07b], pela Tilera [TIL07], a arquitetura *picoArray* [DUL05], e o processador CELL da IBM [KIS06]. Por exemplo, os MPSoCs da Intel e da Tilera possuem 80 e 64 núcleos de processamento, respectivamente. Certamente, o desenvolvimento e a implementação de um sistema dessa complexidade consiste em um processo bastante desafiador. Além disso, o gerenciamento dinâmico dos recursos de um MPSoC também é crucial, visto que deve permitir o uso efetivo de todo o poder de processamento disponibilizado. A maioria desses MPSoCs grandes (*e. g.* da Intel e da Tilera), ainda não chegou ao consumidor. Talvez uma possível causa para tal efeito seja a falta de gerenciamento eficiente para tais sistemas, ou falta de modelos de programação adequados, dentre outras. De fato na literatura revisada, para o caso do MPSoC Intel, o mapeamento de tarefas é realizado manualmente nos estudos de caso apresentados. Isso evidencia a necessidade de pesquisas na área de gerenciamento dinâmico de MPSoCs.

O trabalho proposto foi desenvolvido no sentido de contribuir com o gerenciamento dinâmico de sistemas MPSoCs. Uma dentre as operações importantes é o mapeamento de tarefas. Na maioria dos trabalhos encontrados na literatura propõe-se soluções para o mapeamento estático de tarefas [HU03] [RHE04] [MUR04a] [RUG06] [MEH07], mas recentemente, alguns trabalhos começam a investigar também o mapeamento dinâmico [NGO06] [WRO06] [CHO07]. Enquanto as estratégias estáticas não são adequadas para o emprego em cenários onde a carga das aplicações e suas tarefas é dinâmica, as pesquisas em mapeamento dinâmico ainda refletem seus passos iniciais.

No presente trabalho, foram propostas e avaliadas quatro heurísticas de mapeamento ditas *congestion-aware*. Em aplicações de comunicação intensiva, tais como aplicações de fluxo de dados, a manutenção da ocupação da infra-estrutura de comunicação é

fundamental. A ocorrência de congestionamentos pode acarretar problemas na transmissão de vídeo e áudio, por exemplo. Para essa classe de aplicações os métodos *congestion-aware* são aplicáveis.

As heurísticas propostas atendem ao compromisso de redução de congestionamento *versus* tempo total de execução. Os mapeamentos *congestion-aware* propostos reduzem de forma efetiva os parâmetros congestionamento, ocupação dos canais e latência dos pacotes, frente à implementação de referência (FF). Um breve resumo aponta para uma redução média de quase 90% nos congestionamentos, de 31% na ocupação dos canais da NoC, e de 15% na latência dos pacotes.

Com relação ao tempo de execução total do sistema, no primeiro momento as heurísticas PL e BN apresentaram uma penalidade na ordem de 8% e 2%, respectivamente. Quando o volume de dados transmitidos nas simulações foi multiplicado 10 vezes, os resultados para as mesmas heurísticas apresentaram um ganho de 1,3% no tempo de execução. Assim, pode-se estimar que para grandes volumes de dados o tempo de execução será ainda menor. Ou seja, atrasos inseridos pela operação de mapeamento, bem como pela configuração, devem se tornar insignificantes. Volumes maiores de dados não foram simulados devido ao tempo excessivo para execução das simulações.

Na segunda avaliação realizada, a estratégia gulosa foi posta à prova. Nesse caso a expectativa era de que o resultado dos algoritmos gulosos fosse pior que aquele obtido quando o mapeamento global da aplicação fosse empregado. O mapeamento global é baseado no conhecimento de toda a aplicação, incluindo sua topologia e volumes de comunicação. Enquanto isso, os algoritmos implementados apenas baseiam-se na informação local sobre as comunicações da tarefa a ser mapeada. De fato, os resultados obtidos na estratégia gulosa são inferiores, contudo a diferença entre os resultados foi pequena, ao contrário do esperado. Assim sendo, o emprego da estratégia gulosa é indicado, ao passo que pode reduzir o tempo de mapeamento, e ainda suporta uma baixa penalidade comparada ao desempenho da estratégia global. O emprego de algoritmos como *Simulated Annealing* e *Tabu Search* provavelmente deve resultar em tempos proibitivos de mapeamento, dado um cenário dinâmico.

Como mencionado, o mapeamento *congestion-aware* é baseado na taxa de comunicação entre as tarefas. Enquanto isso, no trabalho de Marcon usado como referência [MAR07], o volume de comunicação é considerado. Possivelmente, uma forma de otimizar as heurísticas propostas consiste em considerar ambos, o volume e a taxa de comunicação durante a decisão de mapeamento. A taxa de comunicação influi nos congestionamento e também na dissipação de potência, enquanto o volume por sua vez, é importante

porque está diretamente relacionado à energia consumida. Baixa dissipação de potência e consumo de energia são importantes requisitos para sistemas portáteis, mas no caso de sistemas MPSoC esses parâmetros também devem receber atenção, visto que sistemas com alta densidade há uma tendência a um grande consumo de energia.

No que diz respeito ao monitoramento, não foi possível comprovar a efetividade no emprego dos monitores distribuídos. Como mencionado anteriormente, para o caso onde as estimativas das taxas de comunicação são precisas, o uso do monitor centralizado será satisfatório. Entretanto, se a estimativa não é precisa ou ainda se não existe estimativa, o uso dos monitores distribuídos é obrigatório. Logicamente, antes do emprego dos monitores distribuídos, faz-se necessária uma investigação mais aprofundada. São importantes informações relativas ao seu consumo de energia, seu consumo de área, entre outras. Nesse caso, é importante realizar compromissos com relação ao custo para obter uma medição precisa da ocupação dos canais da NoC. Uma alternativa que se vislumbra para reduzir esse fator custo-desempenho consiste em atribuir outras funcionalidades aos monitores, *e. g.* controle de *clock gating*, variação dinâmica de voltagem e freqüência, etc.

Outra questão importante a ser citada é o fato do MPSoC modelado aplicar a mesma rede para transmissão de pacotes de dados e pacotes de controle. Esse fato pode estimular questionamentos sobre a influência do tráfego de controle sobre os congestionamentos no sistema. No cenário idealizado, acredita-se que essa possibilidade seja mínima, visto que os pacotes de controle possuem tamanho pequeno, na ordem de 10 *flits* apenas. Enquanto isso, os pacotes de dados devem conter algumas centenas de *flits*. Além disso, acredita-se que o tempo de processamento e de comunicação de cada tarefa mapeada no sistema é relativamente alto, de forma que o tráfego de pacotes de controle será esporádico. Ainda, o fato do emprego de um processador gerente único (*i. e.* centralizado) pode levar a outro questionamento, sobre a possibilidade deste tornar-se um gargalo no sistema. Também para esta questão, pode-se justificar a estratégia centralizada devido ao pequeno fluxo de pacotes de controle esperado.

Em se tratando de um MPSoC com monitoramento distribuído, os pacotes de controle com informação da ocupação dos canais também podem vir a contribuir pra aumentar o congestionamento no sistema. Para evitar esse problema é importante definir de forma adequada o período de amostragem para captura de dados e geração de pacotes. Se esse período for reduzido, logicamente o volume de dados gerados e transmitidos será grande.

7.3. Trabalhos Futuros

Nessa Seção apresenta-se um conjunto de sugestões para trabalhos futuros.

A primeira sugestão para trabalho futuro consiste na otimização da modelagem do sistema. Algumas das possíveis melhorias são:

- Ajustar as heurísticas de mapeamento para utilizar uma função custo multi-objetivo, que considere além da ocupação dos canais, também o volume de dados comunicados, ou os *deadlines* das tarefas para o caso de aplicações de tempo real.

- Adequar a modelagem dos elementos de processamento para que seja considerada o execução de múltiplas tarefas em um dado recurso, de acordo com um modelo de processadores multitarefa. Nesse caso, o mapeamento deve considerar também a ocupação dos processadores. Logo, surge a necessidade da avaliação da relação custo-desempenho em mapear duas ou mais tarefas comunicantes em um mesmo processador, ou mapeá-las em processadores distintos. No primeiro caso provavelmente a comunicação entre as tarefas será mais eficiente, visto que não envolve a NoC. Por outro lado, o mapeamento no mesmo processador implica o compartilhamento do mesmo por várias tarefas, o que pode vir a comprometer o desempenho de uma dada tarefa em termos de processamento.

A segunda sugestão de trabalho consiste na modelagem e avaliação completa do consumo de energia no sistema, bem como da dissipação de potência. O emprego de estratégias de variação dinâmica de voltagem e freqüência, bem como *clock gating* é atrativo para emprego em MPSoCs que contêm muito núcleos, visto que sua heterogeneidade deve permitir que cada um deles opere de acordo com características diferentes de voltagem e freqüência.

A modelagem de aplicações para MPSoCs é complicada, visto que tais sistemas, em geral, visam um domínio vasto de aplicações. Nesse caso, o emprego de aplicações sintéticas geradas a partir da ferramenta TGFF parece satisfatório, já que podem ser obtidas aplicações com grafos bem variados e com taxas de transmissão distintas. Mas, além disso, o emprego de aplicações reais e de *benchmarks* faz-se necessário. Aqui, apenas quatro aplicações reais foram usadas. Assim sendo, é sugestão para trabalhos futuros a investigação de outras aplicações que podem ser adaptadas para execução no sistema. Adicionalmente, é necessária a avaliação do mapeamento proposto, conforme o emprego de possíveis *benchmarks*, a serem investigados.

Uma última sugestão para trabalho futuro é o desenvolvimento de uma ferramenta para automatizar a geração dos cenários de teste (*i. e.* características do MPSoC e da aplicações), a simulação do MPSoC de acordo com as heurísticas, e a avaliação dos resultados. Além disso, a ferramenta deve permitir que o usuário adicione seus próprios algoritmos para o mapeamento de tarefas. Enquanto a ferramenta não é disponibilizada, o acesso aos códigos fonte e maiores esclarecimentos podem ser obtidos diretamente com o Autor, via e-mail *ewerson.carvalho@pucrs.br*.

REFERÊNCIAS BIBLIOGRÁFICAS

[AHM04] Ahmadinia, A.; Bobda, C.; Bednara, M.; Teich, J. "*A New Approach for On-line Placement on Reconfigurable Devices*". In: International Parallel and Distributed Processing Symposium: Reconfigurable Architectures Workshop (IPDPS-RAW). New Mexico, EUA. Abril, 2004. pp.134a-140a.

[ALF08] Al Faruque, M.A.; Krist, R.; Henkel, J. "*ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication*". In: ACM/IEEE Design Automation Conference (DAC). CA, EUA. Julho, 2008. pp.760-765.

[ARA08] Arafeh, B.; Day, K.; Touzene, A. "*A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems*". Journal of Systems Architecture, Vol.54(5). Maio, 2008. pp.530-548.

[BAK99] Baker, M.; Buyya, R.; Hyde, D. "*Cluster Computing: A High-Performance Contender*". IEEE Computer, Vol.32(7). Julho, 1999. pp.79-83.

[BAR07] Barcelos, D.; Brião, E.; Wagner, F. "*A Hybrid Memory Organization to Enhance Task Migration and Dynamic Task Allocation in NoC-based MPSoCs*". In: Chip in Rio - Symposium on Integrated Circuits and Systems Design (SBCCI). Rio de Janeiro, Brasil. Setembro, 2007. pp.282-287.

[BAZ00] Bazargan, K.; Kastner, R.; Sarrafzadeh, M. "*Fast Template Placement for Reconfigurable Computing Systems*". IEEE Design & Test of Computers, Vol.17(1). Janeiro-Março, 2000. pp.68-83.

[BEN02] Benini, L.; De Micheli, G. "*Networks on chips: a new SoC paradigm*". IEEE Computer Magazine, Vol.35(1). Janeiro, 2002. pp.70-78.

[BEN04] Benini, L.; Bertozzi, D. "*Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 2004. pp.18-31.

[BER01] Bergamaschi, R.; Bhattacharya, S.; Wagner, R.; Fellenz, C.; Muhlada, M.; White, F.; Daveau, J.; Lee, W. "*Automating the design of SOCs using cores*". IEEE Design & Test of Computers, Vol.18(5). Setembro-Outubro, 2001. pp.32-45.

[BER05] Bertozzi, D.; Jalabert, A; Murali, S.; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G. "*NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip*". IEEE Transactions on Parallel and Distributed Systems. Vol.16(2). Fevereiro, 2005. pp.113-129.

[BER06] Bertozzi, S.; Acquaviva, A.; Bertozzi, D.; Poggiali, A. "*Supporting task migration in multi-processor systems-on-chip: a feasibility study*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2006. pp.15-20.

[BRI07] Brião, E.; Barcelos, D.; Wronski, F.; Wagner, F. "*Impact of Task Migration in NoC-based MPSoCs for Soft Real-time Applications*". In: IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC). Atlanta, EUA. Outubro, 2007. pp.296-299.

[BRI08] Brião, E.; Barcelos, D.; Wagner, F. "*Dynamic Task Allocation Strategies in MPSoC for Soft Real-time Applications*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2008. pp.1386-1389.

[BUR00] Burd, T.; Brodersen, R. "*Design issues for dynamic voltage scaling*". In: International Symposium on Low Power Electronics and Design (ISLPED). Rapallo, Italia. Julho, 2000. pp.9-14.

[BUR04] Burden, R.; Faires, J. "*Study Guide for Numerical Analysis*". McGraw-Hill. Ed. 8. New-York, EUA. Dezembro, 2004. ISBN 0-534-39200-8.

[BUT97] Butenhof, D.R. "*Programming with POSIX(R) Threads*". Addison-Wesley Professional. Maio, 1997. ISBN: 0201633922.

[CAR07] Carvalho, E.; Calazans, N.; MORAES, F. "*Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs*". In: Annual IEEE/IFIP International Workshop on Rapid Systems Prototyping (RSP). Porto Alegre, Brasil. Maio, 2007. pp.34-40.

[CHA84] Chapiro, D. "*Globally-Asynchronous Locally-Synchronous Systems*". Tese de Doutorado, Stanford University. CA. EUA. Outubro, 1984. 131p.

[CHO07] Chou, C-L.; Marculescu, R. "*Incremental Run-time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels*". In: IEEE/ACM/IFIP international conference on

Hardware/software codesign and system synthesis (CODES+ISSS). Salzburg, Austria. Outubro, 2007. pp.161-166.

[CHO08] Chou, C-L.; Marculescu, R. "*User-Aware Dynamic Task Allocation in Networks-on-Chip*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2008. pp.1232-1237.

[CIO04] Ciordas, C.; Basten, T.; Radulescu, A.; Goossens, K.; Meerbergen, J. "*An event-based network-on-chip monitoring service*". In: High-Level Design Validation and Test Workshop (HLDVT). California, EUA. Novembro, 2004. pp.149-154.

[COR01] Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. "*Introduction to Algorithms*". 2ª Ed. MIT Press and McGraw-Hill. Setembro, 2001. ISBN 978-0-262-53196-2.

[COR06] Cornelius, C.; Timmermann, D. "*Development and Operation of Networks-on-Chip*". In: IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC). Nice, França. Outubro, 2006. pp.19-23.

[CRA77] Cray Research, Inc. "*Cray-1 Computer System Hardware Reference Manual*". Technical Report 2240004 REV. C. Novembro, 1977. 204p.

[DEH98] DeHon, A. "*Comparing Computing Machines*"**.** SPIE Configurable Computing: Technology and Applications. Vol.3526. Novembro, 1998. pp.124-133.

[DIC98] Dick, R.; Rhodes, D.; Wolf, W. "*TGFF: task graphs for free*". In: International Workshop on Hardware/Software Co-design (CODES/CASHE). Washington, EUA. Março, 1998. pp.97-101.

[DUL05] Duller, A.; Towner, D.; Panesar, G.; Gray, A.; Robbins, W. "*picoArray Technology: The Tool's Story*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2005. pp.106-111.

[FAR07] Farag, A.; El-Boghdadi, H.; Shaheen, S. "*Improving utilization of reconfigurable resources using two dimensional compaction*". In: Design, Automation and Test in Europe (DATE). Nice, França. Abril, 2007. pp.135-140.

[FEN08] Fenlason, J.; Stallman, R. "*GNU gprof: The GNU Profiler*". Último acesso: Agosto, 2008. Disponível em http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html

[FET06] Fettweis, G. and Meyr, H. "*4G applications, architectures, design methodology and tools for MPSoC*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha.

Março, 2006. pp.830-831.

[GAP07] GAPH. "*Atlas - An Environment for NoC Generation and Evaluation*". Último acesso: Setembro, 2008. Disponível em http://www.inf.pucrs.br/~gaph/AtlasHtml/AtlasIndex_us.html

[GAR73] Garey, M.R.; Graham, R.L.; Ullman, J.D. "*An Analysis of Some Packing Algorithms*". In: Combinatorial Algorithms. New York: Algorithmics Press. Junho, 1973. pp.39-47.

[GAR79] Garey, M.R.; Johnson, D.S. "*Computers and Intractability: A Guide to the Theory of NP-Completeness*". W.H. Freeman. Janeiro, 1979. ISBN 0-7167-1045-5. A2.5: ND43.

[GHE05] Ghenassia, F. "*Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*". Springer. Novembro, 2005. ISBN 0387262326.

[GLO97] Glover, F., Laguna, M. "*Tabu Search*". Ed.1. Kluwer Academic Publishers. Norwell, MA. Junho, 1997. ISBN 079239965X, 978079239965.

[GOR06] Gordon, M.; Thies, W.; Amarasinghe, S. "*Exploiting Coarse-Grained Task, Data, and Pipeline Parallelism in Stream Programs*". In: International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). San Jose, EUA. Outubro, 2006. pp.151-162.

[GÖT07] Götz, M.; Dittmann, F. "*Dynamic Relocation of Hybrid Tasks: A Complete Design Flow*". In: Reconfigurable Communication-centric SoCs (ReCoSoc). Montpellier, França. Junho, 2007. pp.31-38.

[GSC07] Gschwind, M.; Erb, D.; Manning, S.; Nutter, M. "*An Open Source Environment for Cell Broadband Engine System Software*". IEEE Computer, Vol.40(6). Junho, 2007. pp.37-47.

[HAL06] Halfhill; T.R. "*Ambric's New Parallel Processor - Globally Asynchronous Architecture Eases Parallel Programming*". Microprocessors Report. Outubro, 2006. Disponível em http://www.ambric.com/pdf/MPR_Ambric_Article_10-06_204101.pdf

[HAN04a] Handa, M.; Vemuri, R. "*A Fast Algorithm for Finding Maximal Empty Rectangles for Dynamic FPGA Placement*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 2004. pp.744-745.

[HAN04b] Handa, M.; Vemuri, R. "*Hardware Assisted Two Dimensional Ultra Fast Placement*". In: International Parallel and Distributed Processing Symposium: Reconfigurable Architectures Workshop (IPDPS-RAW). New Mexico, EUA. Abril, 2004. pp.140-147.

[HAN05] Hansson, A.; Goossens, K.; Radulescu, A. "*A unified approach to constrained mapping and routing on networkon-chip architectures*". In: IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS). New Jersey, EUA. Setembro, 2005. pp.75-80.

[HEN03] Henkel, J. "*Closing the SoC design gap*". IEEE Computer, Vol.36(9). Setembro, 2003. pp.119-121.

[HÖL07] Holzenspies, P.K.F.; Smit, G.J.M.; Kuper, J. "*Mapping streaming applications on a reconfigurable MPSoC platform at run-time*". In: International Symposium on System-on-Chip (SoC). Tampere, Finlândia. Novembro, 2007. pp.1-4.

[HÖL08] Hölzenspies, P.K.F.; Hurink, J.L.; Kuper, J.; Smit, G.J.M. "*Run-time Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2008. pp.212-217.

[HU03] Hu, J.; Marculescu, R. "*Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Kitakyushu, Japão. Janeiro, 2003. pp.233-239.

[HU04] Hu, J.; Marculescu, R. "*Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 2004. pp.234-239.

[HU05] Hu, J.; Marculescu, R. "*Energy- and Performance-Aware Mapping for Regular NoC Architectures*". IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol.24(4). Abril, 2005. pp.551-562.

[JAL04] Jalabert, A.; Murali, S.; Benini, L.; De Micheli, G. "*XpipesCompiler: a Tool for Instantiating Application Specific Networks on Chip*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 2004. pp.884-889.

[JER05] Jerraya, A.; Tenhunen, H.; Wolf, W. "*Guest Editors' Introduction: Multiprocessor Systems-on-Chips*". IEEE Computer, Vol.38(7). Julho, 2005. pp.36-40.

[JER06] Jerraya, A.; Bouchhima, A. and Pétrot, F. "*Programming models and HW-SW interfaces abstraction for multi-processor SoC*". In: ACM/IEEE Design Automation Conference (DAC). San Francisco, EUA. Julho, 2006. pp.280-285.

[KAL05a] Kalte, H.; Porrmann, M. "*Context Saving and Restoring for Multitasking in Reconfigurable*

Systems". In: Conference on Field Programmable Logic and Applications (FPL). Tampere, Finlândia. Agosto, 2005. pp.223-228.

[KAL05b] Kalte, H.; Lee, G.; Porrmann, M.; Rückert, U. "*REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems*". In: International Parallel and Distributed Processing Symposium: Reconfigurable Architectures Workshop (IPDPS-RAW). Colorado, EUA. Abril, 2005. pp.151b-157b.

[KAL06a] Kalte, H.; Porrmann; M. "*REPLICA2Pro: task relocation by bitstream manipulation in virtex-II/Pro FPGAs*". In: Conference on Computing frontiers Conference On Computing Frontiers. Ischia, Itália. Maio, 2006. pp.403-412.

[KAL06b] Kalte, H.; Koester, M.; Kettelhoit, B.; Porrmann. M.; Rückert, U. "*A Comparative Study on System Approaches for Partially Reconfigurable Architectures*". In: Engineering of Reconfigurable Systems and Algorithms (ERSA). Las Vegas, EUA. Junho, 2006. pp.70-76.

[KIM03] Kim, J.; Shivle, S.; Siegel, H.; Maciejewski, A.; Braun, T.; Schneider, M.; Tideman, S.; Chitta, R.; Dilmaghani, R.; Joshi, R.; Kaul, A.; Sharma, A.; Sripada, S.; Vangari, P.; Yellampalli, S. "*Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines*". In: International Parallel and Distributed Processing Symposium (IPDPS). Nice, França. Abril, 2003. pp.98-112.

[KIM05] Kim, J.; Siegel, H.; Maciejewski, A.; Eigenmann, R. "*Dynamic Mapping in Energy Constrained Heterogeneous Computing Systems*". In: International Parallel and Distributed Processing Symposium (IPDPS). Colorado, EUA. Abril, 2005. pp.64-73.

[KIM07] Kim, J.; Shivle, S.; Siegel, H.; Maciejewski, A.; Braun, T.; Schneider, M.; Tideman, S.; Chitta, R.; Dilmaghani, R.; Joshi, R. "*Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment*". Journal of Parallel and Distributed Computing, Vol.67(2). Fevereiro, 2007. pp.154-169.

[KIR83] Kirkpatrick, S.; Gelatt, C.; Vecchi, M. "*Optimization by Simulated Annealing*". Science, Vol.220(4598). Maio, 1983. New York, EUA. pp.671-680.

[KIS06] Kistler, M.; Perrone, M.; Petrini, F. "*Cell Multiprocessor Communication Network: Built for Speed*". IEEE Micro, Vol.26(3). Maio/Junho, 2006. pp.10-23.

[KUM02] Kumar, S.; Jantsch, A.; Soininen, J.-P.; Forsell, M.; Millberg, J.-P.; berg, J.; Tiensyrj, K.; Hemani, A. "*A network on chip architecture and design methodology*". In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI). Pittsburgh, EUA. Abril, 2002. pp.105-112.

[LEI03a] Lei, T.; Kumar, S. "*A Two-step Genetic Algorithm for Mapping Task Graphs to Network on Chip Architecture*". In: Euromicro Symposium on Digital System Design: Architectures, Methods and Tools (DSD). Turquia. Setembro, 2003. pp.180-187.

[LEI03b] Lei, T.; Kumar, S. "*Algorithms and Tools for Networks on Chip based System Design*". In: Chip in Sampa - Symposium on Integrated Circuits and Systems Design (SBCCI). São Paulo, SP, Brasil. Setembro, 2003. pp.163-168.

[LIN05] Lin, L.; Wang, C.; Huang, P.; Chou, C.; Jou, J. "*Communication-driven task binding for multiprocessor with latency insensitive network-on-chip*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Shanghai, China. Janeiro, 2005. pp.39-44.

[MAN05] Manolache, S.; Eles, P.; Peng, Z. "*Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC*". In: ACM/IEEE Design Automation Conference (DAC). CA, EUA. Junho, 2005. pp.266-269.

[MAR01] Martin, G.; Chang, H. "*System-on-Chip design*". In: 4th International Conference on ASIC. Shanghai, China. Outubro, 2001. pp.12-17.

[MAR04] Marescaux, T.; Nolleta, V.; Mignoleta, J.; Bartica, A.; Moffata, J.; Avasarea, P.; Coenea, P.; Verkesta, D.; Vernaldea, S.; Lauwereinsa, R. "*Run-time support for heterogeneous multitasking on reconfigurable SoCs*". Integration, the VLSI Journal, Vol.38(1). Outubro, 2004. pp.107-130.

[MAR05a] Marcon, C.; Borin, A.; Susin, A.; Carro, L.; Wagner, F. "*Time and Energy Efficient Mapping of Embedded Applications onto NoCs*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Shanghai, China. Janeiro, 2005. pp.33-38.

[MAR05b] Marcon, C.; Calazans, N.; Moraes, F.; Susin, A.; Reis, I.; Hessel, F. "*Exploring NoC mapping strategies: an energy and timing aware technique*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2005. pp.502-507.

[MAR05c] Marescaux. T.; Rångevall, A.; Nollet, V.; Bartic, A.; Corporaal, H. "*Distributed congestion control for packet switched networks on chip*". In: Parallel Computing Conference (PARCO). Malaga, Espanha. Setembro, 2005. pp. 761-768.

[MAR05d] Marcon, C.; Palma, J.; Calazans, N.; Moraes, F.; Susin, A. "*Modeling the Traffic Effect for the Application Cores Mapping Problem onto NoCs*". In: IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC). Perth, Australia. Outubro, 2005. pp.179-194.

[MAR06] Martin, G. "*Multicore This, Multiprocessor That: It's all MPSoC*". SOCcentral. Julho, 2006. Último acesso: Setembro, 2008. Disponível em http://www.soccentral.com/results.asp?CatID=488&EntryID=19537

[MAR07] Marcon, C.; Moreno, E.; Calazans, N.; Moraes, F. "*Evaluation of Algorithms for Low Energy Mapping onto NoCs*". In: IEEE International Symposium on Circuits and Systems (ISCAS). Louisiana, EUA. Maio, 2007. pp.389-392.

[MAR08] Marcon, C.; Moreno, E.; Calazans, N.; Moraes, F. "*Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption*". IET Computers & Digital Techniques, V 2(6). Novembro, 2008. pp.471-482.

[MEH07] Mehran, A.; Saeidi, S.; Khademzadeh, A.; Afzali-Kusha, A. "*Spiral: A heuristic mapping algorithm for network on chip*". IEICE Electronics Express. Vol.4(15). Agosto, 2007. pp.478-484.

[MEH08] Mehran, A.; Khademzadeh, A.; Saeidi, S. "*DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip*". IEICE Electronics Express. Vol.5(13). Julho, 2008. pp.464-471.

[MEN08] Mentor Graphics Corporation. "*ModelSim® SE User's Manual*". Product Manuals. V6.4b. Oregon, EUA. Outubro, 2008. 1036p.

[MIH03] Mihal, A.; Keutzer, K. "*Mapping concurrent applications onto architectural platforms*". A. Jantsch, H. Tenhunen (Eds). Kluwer Academic Publishers. MA, EUA. 2003. pp.39-59.

[MÖL06] Möller, L.; Soares, R.; Carvalho, E.; Grehs, I.; Calazans, N.; Moraes, F. "*Infrastructure for Dynamic Reconfigurable Systems: Choices and Trade-offs*". In: Chip in the Mountains - Symposium on Integrated Circuits and Systems Design (SBCCI). Ouro Preto, Brasil. Agosto, 2006. pp.44-49.

[MÖL07] Möller, L.; Grehs, I.; Carvalho, E.; Soares, R.; Calazans, N.; Moraes, F. "*A NoC-based Infrastructure to Enable Dynamic Self Reconfigurable Systems*". In: Reconfigurable Communication-centric SoCs (ReCoSoC). Montpellier, França. Junho, 2007. pp.23-30.

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

[MUR04a] Murali, S.; De Micheli, G. "*Bandwidth-Constrained Mapping of Cores onto NoC Architectures*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro,

2004. pp.896-901.

[MUR04b] Murali, S.; De Micheli, G. "*SUNMAP: a tool for automatic topology selection and generation for NoCs*". In: ACM/IEEE Design Automation Conference (DAC). CA, EUA. Junho, 2004. pp.914-919.

[MUR05] Murali, S.; Benini, L.; De Micheli, G. "*Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Shanghai, China. Janeiro, 2005. pp.27-32.

[MUR06a] Murali, S.; Coenen, M.; Radulescu, A.; Goossens, K.; De Micheli, G. "*Mapping and configuration methods for multi-use-case networks on chips*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Yokohama, Japão. Janeiro, 2006. pp.146-151.

[MUR06b] Murali, S.; Coenen, M.; Radulescu, A.; Goossens, K.; De Micheli, G. "*A methodology for mapping multiple use-cases onto networks on chips*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2006. pp.118-123.

[NGO06] Ngouanga, A.; Sassatelli, G.; Torres, L.; Gil, T.; Soares, A.; Susin, A. "*A contextual resources use: a proof of concept through the APACHES platform*". In: IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS). Praga, República Tcheca. Abril, 2006. pp.42-47.

[NOL05a] Nollet, V.; Avasare, P.; Mignolet, J.; Verkest, D. "*Low Cost Task Migration Initiation in a Heterogeneous MP-SoC*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2005. pp.252-253.

[NOL05b] Nollet, V.; Marescaux, T.; Avasare, P.; Mignolet, J-Y. "*Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2005. pp.234-239.

[OH98] Oh, J.; Pedram, M. "*Gated Clock Routing Minimizing the Switched Capacitance*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 1998, pp.692-697.

[ORS05] Orsila, H.; Kangas, T.; Hämäläinen, T. "*Hybrid Algorithm for Mapping Static Task Graphs on Multiprocessor SoCs*". In: International Symposium on System-on-Chip (SoC). Tampere, Finlândia. Novembro, 2005. pp.146-150.

[ORS06] Orsila, H.; Kangas, T.; Salminen, E.; Hämäläinen, T. "*Parameterizing Simulated Annealing for Distributing Task Graphs on Multiprocessor SoCs*". In: International Symposium

on System-on-Chip (SoC). Tampere, Finlândia. Novembro, 2006. pp.73-76.

[ORS07] Orsila, H.; Kangas, T.; Salminen, E.; Hämäläinen, T.; Hännikäinen, M. "*Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips*". Journal of Systems Architecture, Vol.53(11). Novembro, 2007. pp.795-815.

[OST05] Ost, L.; Mello, A.; Palma, J.; Moraes, F.; Calazans, N. "*MAIA - A Framework for Networks on Chip Generation and Verification*". In: Asia and South Pacific Design Automation Conference (ASP-DAC). Shanghai, China. Janeiro, 2005. pp.49-52.

[RHE04] Rhee. C.; Jeong, H.; Ha, S. "*Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures*". In: IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD). CA, EUA. Outubro, 2004. pp.438-443.

[RHO01] Rhoads, S. "*Plasma - most MIPS I(TM) opcodes: Overview*". Setembro, 2007. Último acesso: Janeiro, 2008. Disponível em: http://www.opencores.org/projects.cgi/web/mips/overview

[RUG06] Ruggiero, M.; Guerri, A.; Bertozzi, D.; Poletti, F.; Milano, M. "*Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2006. pp.3-8.

[SAE07] Saeidi, S.; Khademzadeh, A.; Mehran, A. "*SMAP: An Intelligent Mapping tool for Network on Chip*". In: International Symposium on Signals, Circuits and Systems (ISSCS). Iasi, Romania. Julho, 2007. pp.1-4.

[SAI07a] Saint-Jean, N.; Sassatelli, G.; Benoit, P.; Torres, L.; Robert, M. "*HS-Scale: a Hardware-Software Scalable MP-SOC Architecture for embedded Systems*". In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI). Porto Alegre, Brasil. Março, 2007. pp.21-28.

[SAI07b] Saint-Jean, N.; Jalier, C.; Sassatelli, G.; Benoit, P.; Torres, L.; Robert, M. "*HS Scale: A run-time adaptable MP-SoC architecture*". In: Reconfigurable Communication-centric SoCs (ReCoSoC). Montpellier, França. Junho, 2007. pp.39-46.

[SAI08] Saint-Jean, N.; Benoit, P.; Sassatelli, G.; Torres, L.; Robert, M. "*MPI-Based Adaptive Task Migration Support on the HS-Scale System*". In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI). Montpellier, França. Abril, 2008. pp.105-110.

[SAS07] Sassatelli, G.; Saint-Jean, N.; Woszezenki, C.; Grehs, I.; Moraes, F. "*Architectural Issues in Homogeneous NoC-Based MPSoC*". In: Annual IEEE/IFIP International Workshop on Rapid Systems Prototyping (RSP). Porto Alegre, Brasil. Maio, 2007. pp.139-142.

[SHI04] Shin, D.; Kim, J. "*Power-aware communication optimization for networks-on-chips with voltage scalable links*". In: IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS). Stockholm, Sweden. Setembro, 2004. pp.170-175.

[SIA05] Semiconductor Industry Association (SIA). "*2005 International Technology Roadmap for Semiconductors*". (ITRS). Outubro, 2005.

[SIA99] Semiconductor Industry Association (SIA). "*1999 International Technology Roadmap for Semiconductors*". (ITRS). Dezembro, 1999.

[SMI04a] Smit, L.T.; Smit, G.J.M.; Hurink, J.L.; Broersma, H.; Paulusma, D.; Wolkotte. P.T. "*Run-time assignment of tasks to multiple heterogeneous processors*". In: PROGRESS workshop on embedded systems. Nieuwegein, Holanda. Outubro, 2004. pp.185-192.

[SMI04b] Smit, L.T.; Smit, G.J.M.; Hurink, J.L.; Broersma, H.; Paulusma, D.; Wolkotte. P.T. "*Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture*". In: International Conference on Field-Programmable Technology (FPT). Brisbane, Austrália. Dezembro, 2004. pp.421-424.

[SMI05] Smit, L.T.; Hurink, J.L.; Smit, G.J.M. "*Run-time mapping of applications to a heterogeneous SoC*". In: International Symposium on System-on-Chip (SoC). Tampere, Finlândia. Novembro, 2005. pp.78-81.

[SRI05] Srinivasan, K.; Chatha, K. "*A technique for low energy mapping and routing in network-on-chip architectures*". In: International Symposium on Low Power Electronics and Design (ISLPED). CA, EUA. Agosto, 2005. pp.387-392.

[STE06] Streichert, T.; Strengert, C.; Haubelt, C.; Teich, J. "*Dynamic task binding for hardware/software reconfigurable networks*". In: Chip in the Mountains - Symposium on Integrated Circuits and Systems Design (SBCCI). Ouro Preto, Brasil. Agosto, 2006. pp.38-43.

[STE95] Sterling, T.; Savarese, D.; Becker, D.; Dorband, J.; Ranawake, U.; Packer, C. "*BEOWULF: A Parallel Workstation for Scientific Computation*". In: International Conference on Parallel Processing (ICPP). Illinois, EUA. Agosto, 1995. pp.11-14.

[TIL07] Tilera Corporation. "*TILE64™ Processor*". Product Brief Description. Santa Clara, CA, EUA. Agosto, 2007. 2p.

[VAN02] Van der Tol, E.; Jaspers, E. "*Mapping of MPEG-4 Decoding on a Flexible Architecture Plat-*

form". In: SPIE Conference on Visualization and Data Analysis (SPIE 4674). San Jose, CA, USA; Janeiro, 2002. pp.1-13.

[VAN07a] van den Brand, J.W.; Ciordas, C.; Goossens, K.; Basten, T. "*Congestion-controlled best-effort communication for networks-on-chip*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Abril, 2007. pp.1-6.

[VAN07b] Vangal, S.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Iyer, P.; Singh, A.; Jacob, T.; Jain, S.; Venkataraman, S.; Hoskote, Y.; Borkar, N. "*An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS*". In: IEEE International Solid-State Circuits Conference (ISSCC). CA, EUA. Fevereiro, 2007. pp.5-7.

[VAN08] Vangal, S.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Singh, A.; Jacob, T.; Jain, S.; Erraguntla, V.; Roberts, C.; Hoskote, Y.; Borkar, N.; Borkar, S. "*An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS*". IEEE Journal of Solid-State Circuits, Vol.43(1). Janeiro, 2008. pp.29-41.

[WEN07] Wentzlaff, D.; Griffin, P.; Hoffmann, H.; Bao, L.; Edwards, B.; Ramey, C. "*On-Chip Interconnection Architecture of the Tile Processor*". IEEE Micro, Vol.27(5). Setembro-Outubro, 2007. pp.15-31.

[WIR98] Wirthlin, M.; Hutchings, B. "*Improving Functional Density Using Run-Time Circuit Reconfiguration*". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.6(2). Julho, 1998. pp.247-256.

[WOL08] Wolf, W.; Jerraya, A.; Martin, G. "*Multiprocessor System-on-Chip (MPSoC) Technology*". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.27(10). Outubro, 2008. pp.1701-1713.

[WOS07] Woszezenki, C. "*Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs*". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS. Orientador: Fernando Gehm Moraes. Porto Alegre, Brasil. Fevereiro, 2007. 121p. Disponível em http://www.inf.pucrs.br/~moraes/my_pubs/papers/dissertacao_cris.pdf

[WRO06] Wronski, F.; Brião, E.; Wagner, F. "*Evaluating Energy-aware Task Allocation Strategies for MPSoCS*". In: IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES). Braga, Portugal. Outubro, 2006. pp. 215-224.

[WU03] Wu, D.; Al-Hashimi, B.; Eles, P. "*Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems*". In: Design, Automation and Test in Europe (DATE). Munich, Alemanha. Março, 2003. pp.90-95.

[ZEF03] Zeferino, C.; Susin, A. "*SoCIN: a Parametric and Scalable Network-on-Chip*". In: Chip in Sampa - Symposium on Integrated Circuits and Systems Design (SBCCI). São Paulo, Brasil. Setembro, 2003. pp.169-174.

[ZEF04] Zeferino, C.; Kreutz, M.; Susin, A. "*RASoC: A router soft-core for networks-on-chip*". In: Design, Automation and Test in Europe (DATE). Paris, França. Fevereiro, 2004. pp.198-203.

APÊNDICE A. APLICAÇÕES EMPREGADAS NOS CENÁRIOS DE SIMULAÇÃO

Neste Apêndice, apresenta-se as aplicações empregadas em cada um dos cenários simulados, incluindo informação sobre sua fonte, sua representação conforme o modelo de aplicação adotado no presente trabalho, e ainda a sua representação conforme a modelagem da ferramenta CAFES para o caso dos Cenários D e E.

A.1. Aplicações Empregadas no Cenário A

No Cenário A, foram empregadas 20 aplicações idênticas com grafo cuja topologia representa um formato de *pipeline*, típico de aplicações orientadas a fluxo de dados. Cada uma das aplicações contém 10 tarefas, com taxas de injeção de dados idênticas para todas as arestas do grafo. A Figura abaixo esboça o grafo que representa tais aplicações, bem como os atributos adotados nas 6 simulações realizadas (*i. e.* PIPE5%, PIPE10%, PIPE15%, PIPE20%, PIPE25% e PIPE30%).

PIPE 5%
0 [1, 2000, **5%**, 10, 5%]
1 [2, 2000, **5%**, 10, 5%]
2 [3, 2000, **5%**, 10, 5%]
3 [4, 2000, **5%**, 10, 5%]
4 [5, 2000, **5%**, 10, 5%]
5 [6, 2000, **5%**, 10, 5%]
6 [7, 2000, **5%**, 10, 5%]
7 [8, 2000, **5%**, 10, 5%]
8 [9, 2000, **5%**, 10, 5%]

PIPE 10%
0 [1, 2000, **10%**, 10, 5%]
1 [2, 2000, **10%**, 10, 5%]
2 [3, 2000, **10%**, 10, 5%]
3 [4, 2000, **10%**, 10, 5%]
4 [5, 2000, **10%**, 10, 5%]
5 [6, 2000, **10%**, 10, 5%]
6 [7, 2000, **10%**, 10, 5%]
7 [8, 2000, **10%**, 10, 5%]
8 [9, 2000, **10%**, 10, 5%]

PIPE 15%
0 [1, 2000, **15%**, 10, 5%]
1 [2, 2000, **15%**, 10, 5%]
2 [3, 2000, **15%**, 10, 5%]
3 [4, 2000, **15%**, 10, 5%]
4 [5, 2000, **15%**, 10, 5%]
5 [6, 2000, **15%**, 10, 5%]
6 [7, 2000, **15%**, 10, 5%]
7 [8, 2000, **15%**, 10, 5%]
8 [9, 2000, **15%**, 10, 5%]

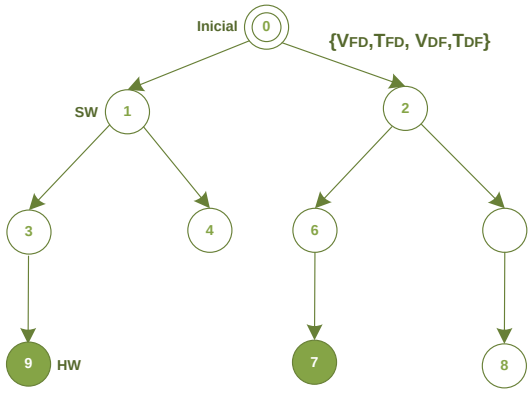
PIPE 20%
0 [1, 2000, **20%**, 10, 5%]
1 [2, 2000, **20%**, 10, 5%]
2 [3, 2000, **20%**, 10, 5%]
3 [4, 2000, **20%**, 10, 5%]
4 [5, 2000, **20%**, 10, 5%]
5 [6, 2000, **20%**, 10, 5%]
6 [7, 2000, **20%**, 10, 5%]
7 [8, 2000, **20%**, 10, 5%]
8 [9, 2000, **20%**, 10, 5%]

PIPE 25%
0 [1, 2000, **25%**, 10, 5%]
1 [2, 2000, **25%**, 10, 5%]
2 [3, 2000, **25%**, 10, 5%]
3 [4, 2000, **25%**, 10, 5%]
4 [5, 2000, **25%**, 10, 5%]
5 [6, 2000, **25%**, 10, 5%]
6 [7, 2000, **25%**, 10, 5%]
7 [8, 2000, **25%**, 10, 5%]
8 [9, 2000, **25%**, 10, 5%]

PIPE 30%
0 [1, 2000, **30%**, 10, 5%]
1 [2, 2000, **30%**, 10, 5%]
2 [3, 2000, **30%**, 10, 5%]
3 [4, 2000, **30%**, 10, 5%]
4 [5, 2000, **30%**, 10, 5%]
5 [6, 2000, **30%**, 10, 5%]
6 [7, 2000, **30%**, 10, 5%]
7 [8, 2000, **30%**, 10, 5%]
8 [9, 2000, **30%**, 10, 5%]

A.2. Aplicações Empregadas no Cenário B

No Cenário B, foram empregadas 20 aplicações idênticas cujos grafos apresentam uma topologia similar a uma árvore. Estas representam aplicações com um maior grau de paralelismo entre suas tarefas. Assim como no Cenário A, cada aplicação possui 10 tarefas, e as taxas de injeção de pacotes na rede são idênticas para todas as arestas do grafo. A Figura abaixo esboça o grafo que representa tais aplicações, bem como os atributos de cada uma das 4 simulações realizadas (*i. e.* TREE5%, TREE10%, TREE15%, TREE20%).



TREE 5%

0 [2, 1000, **5%**, 10, 5%]
0 [1, 1000, **5%**, 10, 5%]
1 [3, 1000, **5%**, 10, 5%]
1 [4, 1000, **5%**, 10, 5%]
2 [5, 1000, **5%**, 10, 5%]
2 [6, 1000, **5%**, 10, 5%]
3 [9, 1000, **5%**, 10, 5%]
5 [8, 1000, **5%**, 10, 5%]
6 [7, 1000, **5%**, 10, 5%]

TREE 10%

0 [2, 1000, **10%**, 10, 5%]
0 [1, 1000, **10%**, 10, 5%]
1 [3, 1000, **10%**, 10, 5%]
1 [4, 1000, **10%**, 10, 5%]
2 [5, 1000, **10%**, 10, 5%]
2 [6, 1000, **10%**, 10, 5%]
3 [9, 1000, **10%**, 10, 5%]
5 [8, 1000, **10%**, 10, 5%]
6 [7, 1000, **10%**, 10, 5%]

TREE 15%

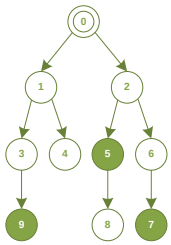
0 [2, 1000, **15%**, 10, 5%]
0 [1, 1000, **15%**, 10, 5%]
1 [3, 1000, **15%**, 10, 5%]
1 [4, 1000, **15%**, 10, 5%]
2 [5, 1000, **15%**, 10, 5%]
2 [6, 1000, **15%**, 10, 5%]
3 [9, 1000, **15%**, 10, 5%]
5 [8, 1000, **15%**, 10, 5%]
6 [7, 1000, **15%**, 10, 5%]

TREE 20%

0 [2, 1000, **20%**, 10, 5%]
0 [1, 1000, **20%**, 10, 5%]
1 [3, 1000, **20%**, 10, 5%]
1 [4, 1000, **20%**, 10, 5%]
2 [5, 1000, **20%**, 10, 5%]
2 [6, 1000, **20%**, 10, 5%]
3 [9, 1000, **20%**, 10, 5%]
5 [8, 1000, **20%**, 10, 5%]
6 [7, 1000, **20%**, 10, 5%]

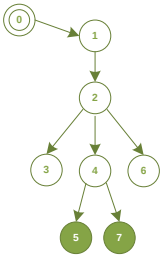
A.3. Aplicações Empregadas no Cenário C

No Cenário C, foram empregadas 20 aplicações diferentes, com grafos de topologias variadas, geradas com o auxilio da ferramenta TGFF. *Task Graph For Free* ou simplesmente TGFF [DIC98] é uma ferramenta que permite a geração automática de um conjunto de grafos. Para isso ela baseia-se em um conjunto de parâmetros, incluindo os números mínimo e máximo de vértices, e o intervalo para variação dos pesos para as arestas. Os grafos gerados para o Cenário C são compostos por entre 3 a 10 vértices, e as taxas de injeção foram escolhidas aleatoriamente. Os grafos da Figura abaixo apresentam cada uma das 20 aplicações simuladas, bem como os respectivos atributos referentes aos volumes de taxas de comunicação adotados.



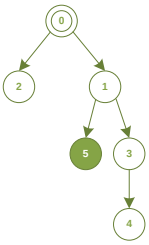
App0

0 [2, 150, 22%, 170, 20%]
0 [1, 340, 10%, 300, 15%]
1 [3, 470, 10%, 410, 20%]
1 [4, 300, 15%, 480, 15%]
2 [5, 380, 10%, 230, 10%]
2 [6, 240, 5%, 140, 5%]
3 [9, 210, 15%, 280, 25%]
5 [8, 190, 25%, 340, 20%]
6 [7, 120, 20%, 360, 25%]



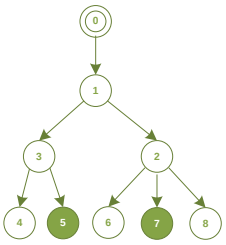
App1

0 [1, 260, 25%, 230, 20%]
1 [2, 330, 20%, 280, 5%]
2 [3, 310, 10%, 190, 5%]
2 [4, 190, 10%, 200, 5%]
2 [6, 180, 5%, 220, 15%]
4 [5, 190, 10%, 200, 10%]
4 [7, 260, 15%, 300, 15%]



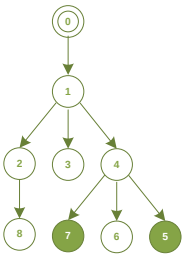
App2

0 [1, 240, 20%, 250, 10%]
0 [2, 260, 15%, 230, 20%]
1 [3, 330, 25%, 320, 10%]
1 [5, 280, 5%, 280, 5%]
3 [4, 260, 30%, 300, 15%]



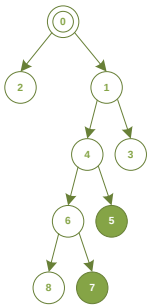
App3

0 [1, 260, 20%, 260, 10%]
1 [2, 200, 15%, 170, 5%]
1 [3, 330, 10%, 320, 10%]
3 [4, 180, 15%, 170, 10%]
3 [5, 190, 5%, 160, 15%]
2 [6, 260, 15%, 230, 15%]
2 [7, 320, 5%, 320, 5%]
2 [8, 310, 5%, 190, 5%]



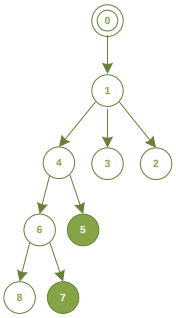
App4

0 [1, 240, 25%, 240, 10%]
1 [2, 190, 15%, 200, 5%]
1 [3, 180, 5%, 180, 5%]
1 [4, 240, 5%, 250, 5%]
4 [5, 310, 10%, 190, 6%]
4 [6, 200, 15%, 270, 10%]
4 [7, 200, 5%, 170, 10%]
2 [8, 260, 30%, 300, 20%]



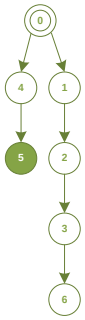
App5

0 [1, 330, 30%, 320, 10%]
0 [2, 330, 5%, 320, 15%]
1 [3, 170, 25%, 310, 5%]
1 [4, 170, 5%, 310, 6%]
4 [5, 180, 10%, 180, 15%]
4 [6, 310, 15%, 290, 10%]
6 [7, 250, 15%, 170, 15%]
6 [8, 180, 10%, 250, 10%]



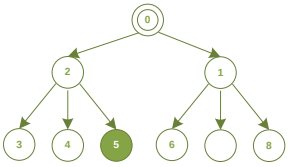
App6

0 [1, 320, 15%, 320, 10%]
1 [2, 160, 10%, 160, 5%]
1 [3, 190, 10%, 200, 6%]
1 [4, 310, 5%, 190, 7%]
4 [5, 310, 15%, 190, 15%]
4 [6, 180, 10%, 170, 10%]
6 [7, 330, 5%, 320, 25%]
6 [8, 300, 25%, 200, 5%]



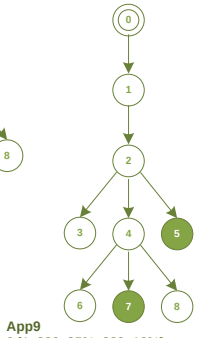
App7

0 [1, 280, 25%, 280, 25%]
0 [4, 150, 5%, 200, 10%]
1 [2, 260, 15%, 300, 5%]
2 [3, 260, 30%, 160, 15%]
3 [6, 180, 15%, 170, 5%]
4 [5, 260, 25%, 230, 15%]



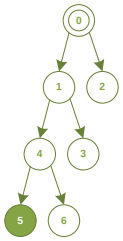
App8

0 [1, 150, 15%, 200, 5%]
0 [2, 310, 15%, 170, 15%]
2 [3, 150, 10%, 200, 5%]
2 [4, 290, 5%, 290, 10%]
2 [5, 180, 5%, 170, 6%]
1 [6, 310, 6%, 170, 15%]
1 [7, 190, 10%, 160, 10%]
1 [8, 150, 13%, 200, 5%]



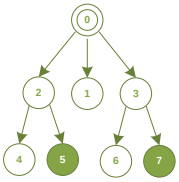
App9

0 [1, 280, 35%, 280, 10%]
1 [2, 200, 20%, 170, 5%]
2 [3, 160, 10%, 210, 10%]
2 [4, 270, 15%, 220, 5%]
2 [5, 280, 5%, 280, 6%]
4 [6, 190, 25%, 200, 6%]
4 [7, 180, 5%, 170, 7%]
4 [8, 190, 7%, 240, 7%]



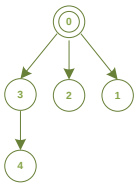
App10

0 [1, 190, 15%, 200, 20%]
0 [2, 160, 15%, 160, 15%]
1 [3, 330, 10%, 320, 5%]
1 [4, 260, 15%, 160, 15%]
4 [5, 190, 15%, 290, 5%]
4 [6, 260, 15%, 160, 10%]



App12

0 [1, 210, 14%, 230, 13%]
0 [2, 250, 13%, 300, 10%]
0 [3, 210, 5%, 230, 15%]
2 [4, 330, 17%, 320, 19%]
2 [5, 170, 10%, 280, 12%]
3 [6, 310, 14%, 250, 15%]
3 [7, 250, 10%, 300, 10%]

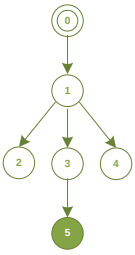


App13

0 [1, 240, 15%, 240, 15%]
0 [2, 200, 10%, 170, 5%]
0 [3, 250, 5%, 300, 5%]
3 [4, 260, 25%, 230, 35%]

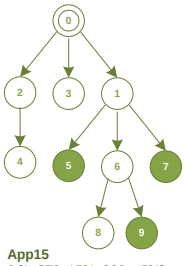
App11

0 [1, 250, 30%, 200, 25%]
1 [2, 310, 25%, 250, 5%]
2 [3, 200, 20%, 170, 10%]
3 [4, 240, 25%, 240, 15%]
4 [5, 180, 10%, 250, 15%]



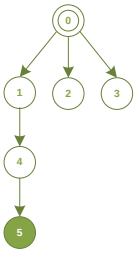
App14

0 [1, 230, 20%, 160, 4%]
1 [2, 230, 5%, 160, 5%]
1 [3, 240, 12%, 250, 15%]
1 [4, 240, 5%, 250, 5%]
3 [5, 190, 10%, 200, 27%]



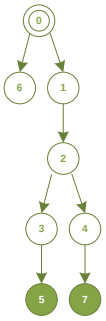
App15

0 [1, 270, 15%, 200, 5%]
0 [2, 190, 15%, 210, 15%]
0 [3, 260, 5%, 260, 10%]
2 [4, 330, 10%, 280, 25%]
1 [5, 280, 5%, 280, 5%]
1 [6, 260, 13%, 210, 10%]
1 [7, 270, 8%, 220, 5%]
6 [8, 240, 15%, 250, 10%]
6 [9, 310, 15%, 190, 25%]



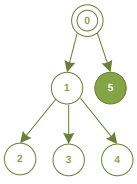
App16

0 [1, 190, 30%, 200, 5%]
0 [2, 310, 5%, 290, 5%]
0 [3, 330, 5%, 280, 25%]
1 [4, 190, 15%, 290, 25%]
4 [5, 190, 15%, 200, 40%]



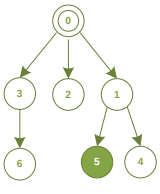
App18

0 [1, 190, 30%, 200, 20%]
0 [6, 310, 5%, 250, 15%]
1 [2, 300, 20%, 270, 10%]
2 [3, 190, 15%, 240, 5%]
2 [4, 200, 5%, 170, 7%]
3 [5, 250, 20%, 170, 20%]
4 [7, 200, 25%, 270, 35%]



App17

0 [1, 290, 25%, 290, 5%]
0 [5, 150, 15%, 200, 35%]
1 [2, 190, 10%, 210, 3%]
1 [3, 330, 15%, 320, 7%]
1 [4, 190, 5%, 240, 5%]



App19

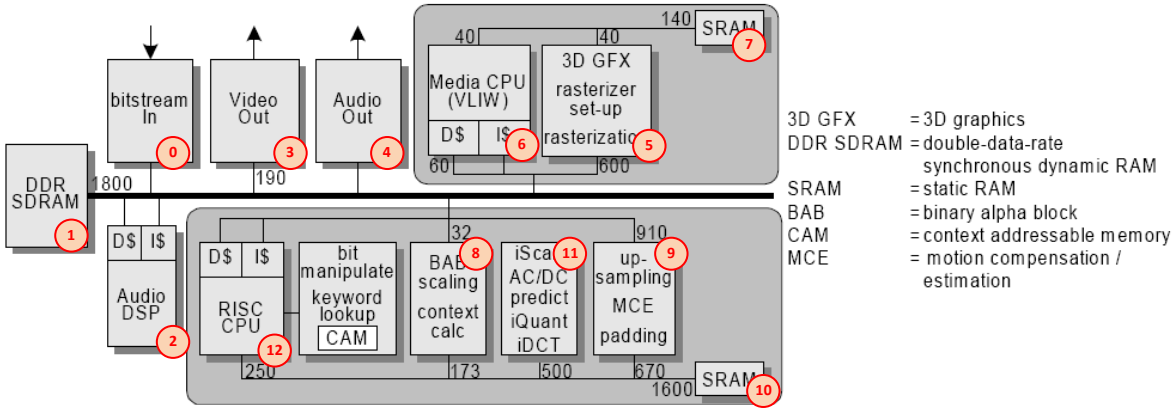
0 [1, 190, 15%, 200, 10%]
0 [2, 190, 10%, 200, 12%]
0 [3, 210, 5%, 230, 4%]
1 [4, 330, 10%, 280, 15%]
1 [5, 310, 12%, 190, 10%]
3 [6, 280, 25%, 280, 15%]

A.4. Aplicações Empregadas no Cenário D

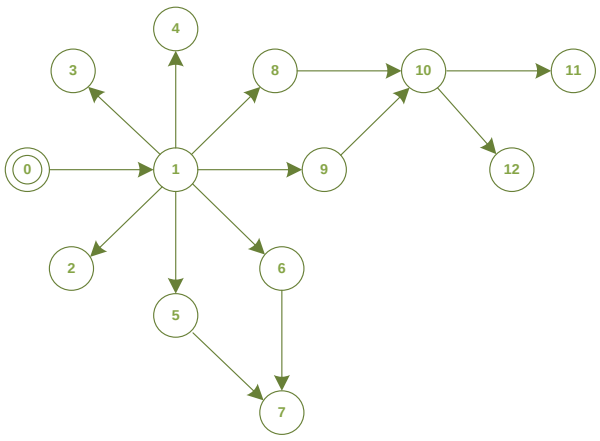
Nesta parte do Apêndice, são discutidas as principais aplicações empregadas no Cenário D. Além disso, suas representações conforme o modelo adotado aqui, e aquele adotado pela ferramenta CAFES são apresentadas, bem como o mapeamento resultante da mesma ferramenta. Essas aplicações são empregadas para avaliar o desempenho dos algoritmos proposto de acordo com grafos que representam topologias, volumes e taxas aproximadas de um cenário real.

A.4.1. MPEG-4

A primeira aplicação empregada nos Cenários D e E dos experimentos foi a MPEG-4. Seu grafo é baseado no diagrama de blocos desta aplicação, apresentado em [VAN02], e reproduzido na Figura abaixo, onde os círculos indicam os identificadores atribuídos às tarefas.



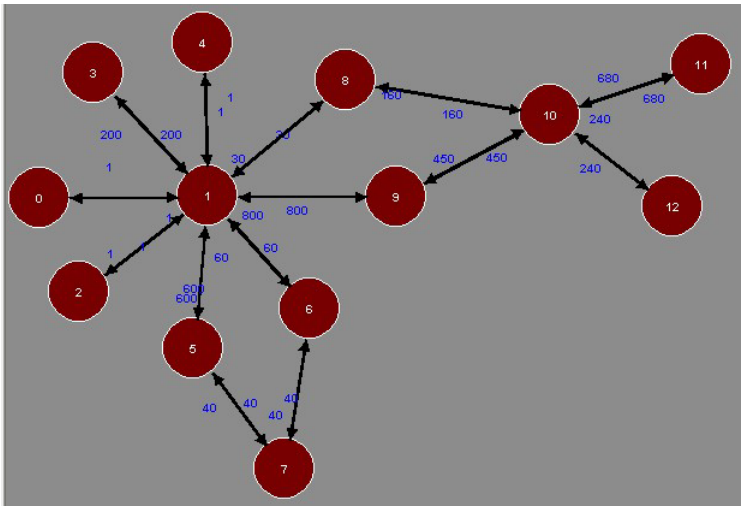
A Figura a seguir apresenta o mesmo diagrama de blocos da aplicação MPEG-4, agora representado para o MPSoC modelado no presente trabalho. Trata-se da mesma Figura 6.1. Ela foi apresentada aqui para facilitar a comparação com a estrutura da Aplicação, esboçada na Figura anterior.



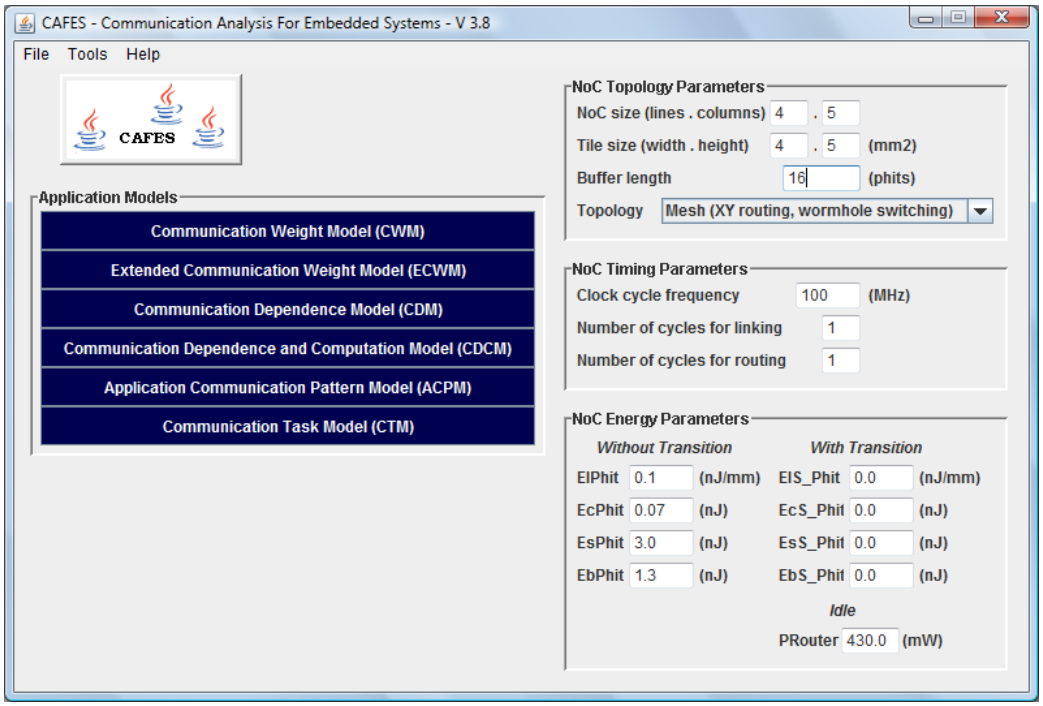
F [D, VFD, TFD, VDF, TDF]

0 [1, 10, 15%, 10, 1%]
1 [2, 10, 1%, 10, 1%]
1 [3, 100, 20%, 100, 2%]
1 [4, 10, 1%, 10, 1%]
1 [5, 600, 10%, 600, 10%]
1 [6, 100, 6%, 100, 6%]
1 [8, 100, 3%, 100, 3%]
1 [9, 400, 20%, 400, 20%]
5 [7, 100, 4%, 100, 4%]
6 [7, 100, 4%, 100, 4%]
8 [10, 200, 8%, 200, 8%]
9 [10, 450, 10%, 450, 10%]
10 [11, 680, 10%, 680, 10%]
10 [12, 240, 10%, 240, 10%]

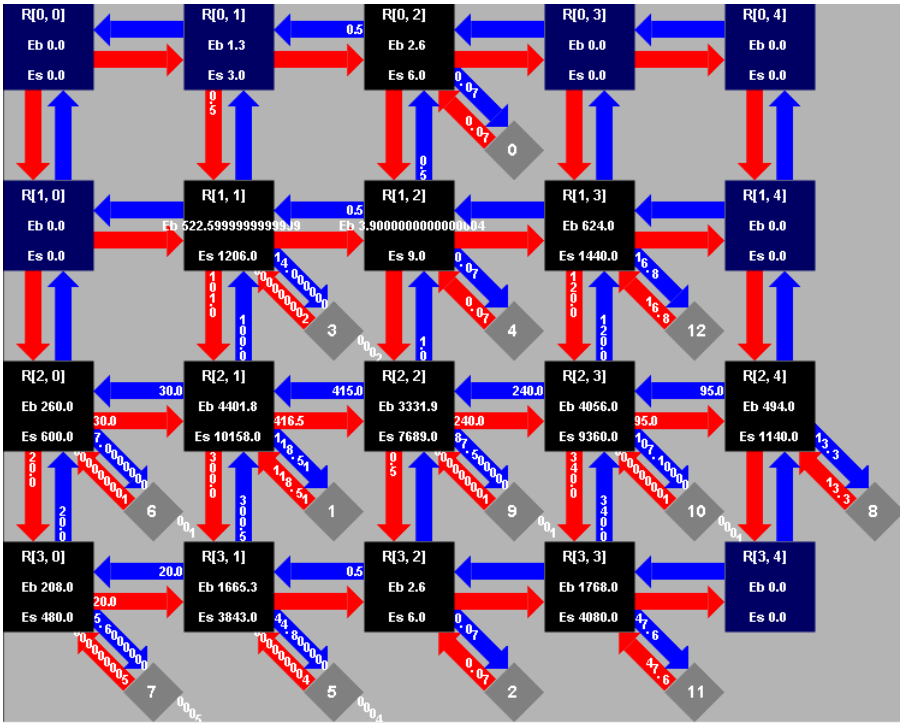
A Figura abaixo apresenta o mesmo diagrama de blocos da aplicação MPEG-4, agora conforme a modelagem adotada pela ferramenta CAFES, quando o modelo CWM é empregado.



Todos os mapeamentos realizados no Cenário D foram baseados na configuração padrão da ferramenta CAFES. O modelo empregado foi o mais simplificado, o CWM. A Figura abaixo apresenta a tela inicial da ferramenta CAFES bem como os valores padrões de energia consumida relativa à transmissão através dos enlaces entre roteadores (*i. e. ElPhit*), nos enlaces locais (*i. e. EcPhit*), relativa ao chaveamento (*i. e. EsPhit*), e inda ao armazenamento no *buffer* inter no do roteador (*i. e. EbPhit*). Após pressionar o modelo escolhido, é apresentada a tela onde deve-se inserir o grafo da aplicação, conforme o apresentado na Figura anterior.

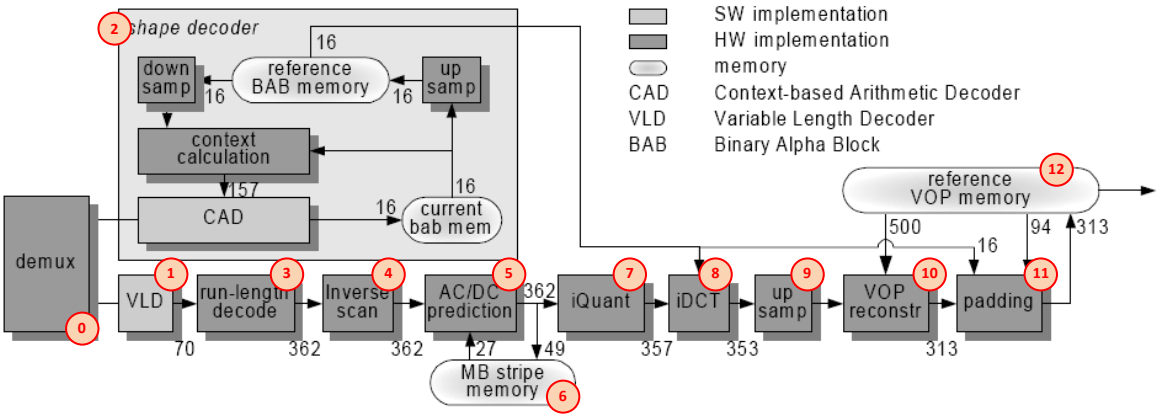


A Figura a seguir apresenta o resultado de mapeamento obtido para a aplicação MPEG-4 através da ferramenta CAFES, quando o algoritmo *Simulated Annealing* foi empregado. Este resultado foi base para os mapeamentos apresentados anteriormente na Figura 6.5.

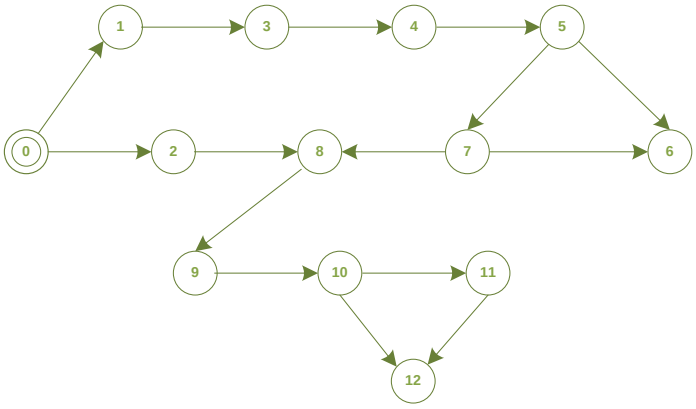


A.4.2. VOPD

A aplicação VOPD também foi empregada nos Cenários D e E dos experimentos do Capítulo 6. O grafo dessa aplicação anteriormente apresentado (Figura 6.2) foi baseado no diagrama de blocos desta aplicação, apresentado abaixo, obtido a partir de [VAN02]. Os círculos indicam os identificadores atribuídos às tarefas.



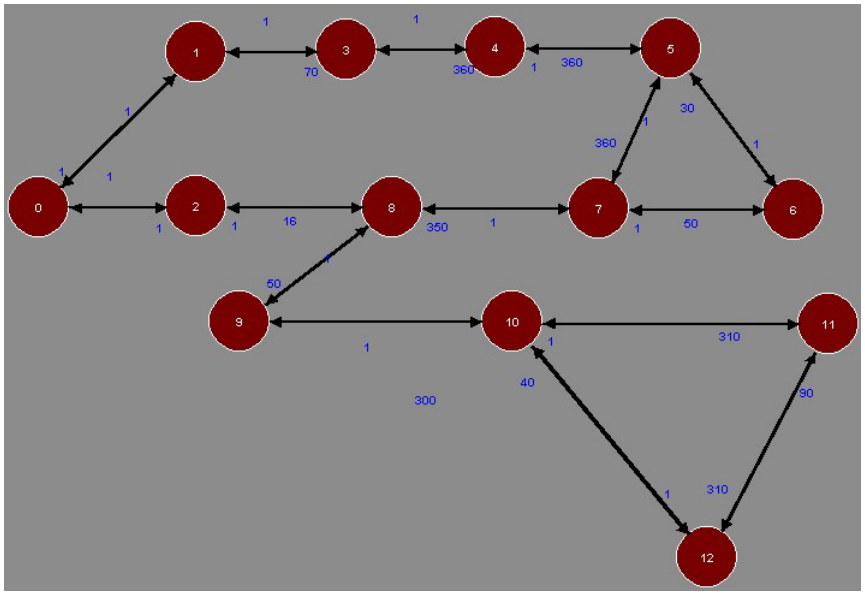
A Figura abaixo apresenta o mesmo diagrama de blocos, agora em sua representação conforme a modelagem de aplicações adotada no presente trabalho. Trata-se da mesma Figura 6.2. Ela é apresentada aqui, apenas com o objetivo de facilitar sua comparação com a Figura anterior.



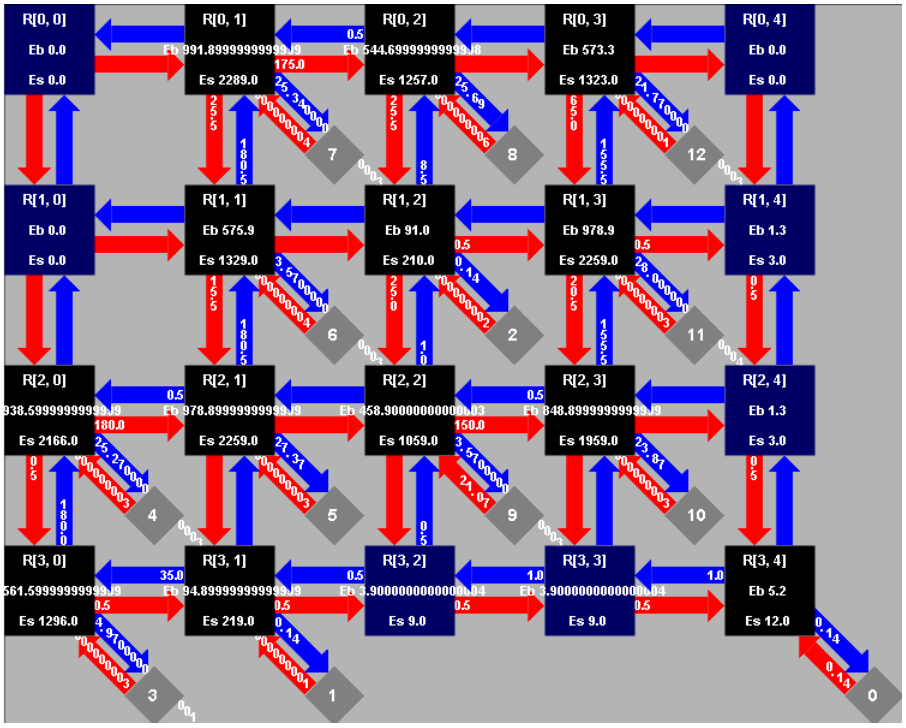
F [D, VFD, TFD, VDF, TDF]

0 [1, 10, 1%, 10, 1%]
0 [2, 10, 1%, 10, 1%]
1 [3, 100, 7%, 10, 1%]
2 [8, 80, 2%, 10, 1%]
3 [4, 100, 36%, 10, 1%]
4 [5, 100, 36%, 10, 1%]
5 [6, 10, 1%, 100, 3%]
5 [7, 100, 36%, 100,15%]
6 [7, 10, 1%, 100, 5%]
7 [8, 100, 35%, 10, 1%]
8 [9, 100, 5%, 10, 1%]
9 [10, 100, 30%, 10, 1%]
10 [11, 100, 31%, 10, 1%]
10 [12, 10, 1%, 100, 4%]
11 [12, 100, 31%, 100, 9%]

Na Figura a seguir, apresenta-se o diagrama de blocos da aplicação na forma como ele foi representado conforme a modelagem CWM adotada na ferramenta CAFES.

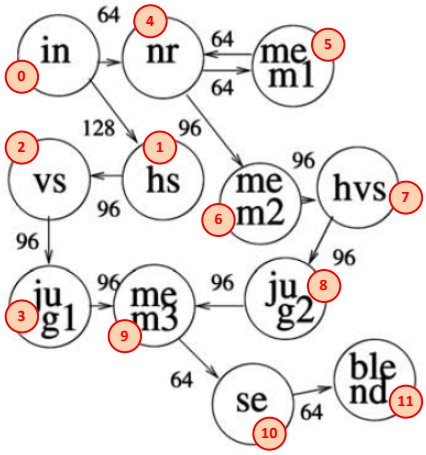


Em seguida, na Figura abaixo, apresenta-se o resultado obtido para o mapeamento da aplicação VOPD, realizado pela ferramenta CAFES. O algoritmo de mapeamento adotado foi o *Tabu Search*. Este resultado foi base para os mapeamentos apresentados anteriormente na Figura 6.5.

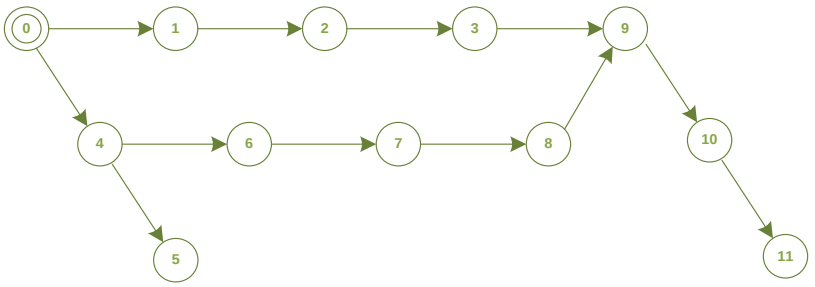


A.4.3. MWD

O grafo abaixo representa a aplicação MWD (*Multi-Window Display*). Este grafo foi obtido a partir do trabalho de Bertozzi e outros [BER05]. Os círculos menores dizem respeito aos identificadores atribuídos a cada uma das tarefas que compõem esta aplicação.



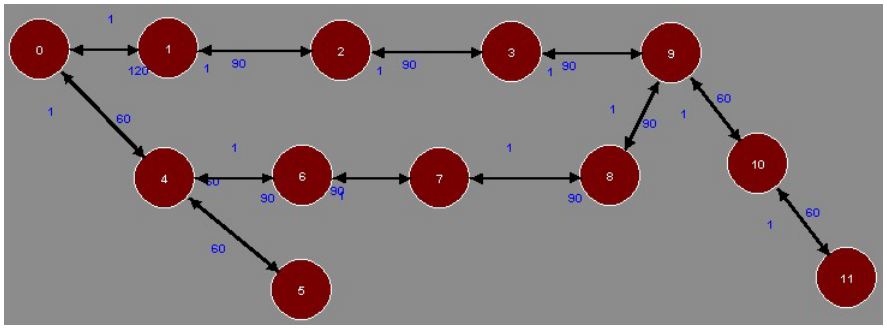
A Figura a seguir apresenta o mesmo grafo anterior, agora representado conforme a modelagem aplicada no presente trabalho. Trata-se da mesma Figura 6.3, apresentada aqui para facilitar sua comparação a Figura anterior, onde foi apresentado cada um dos identificadores das tarefas.



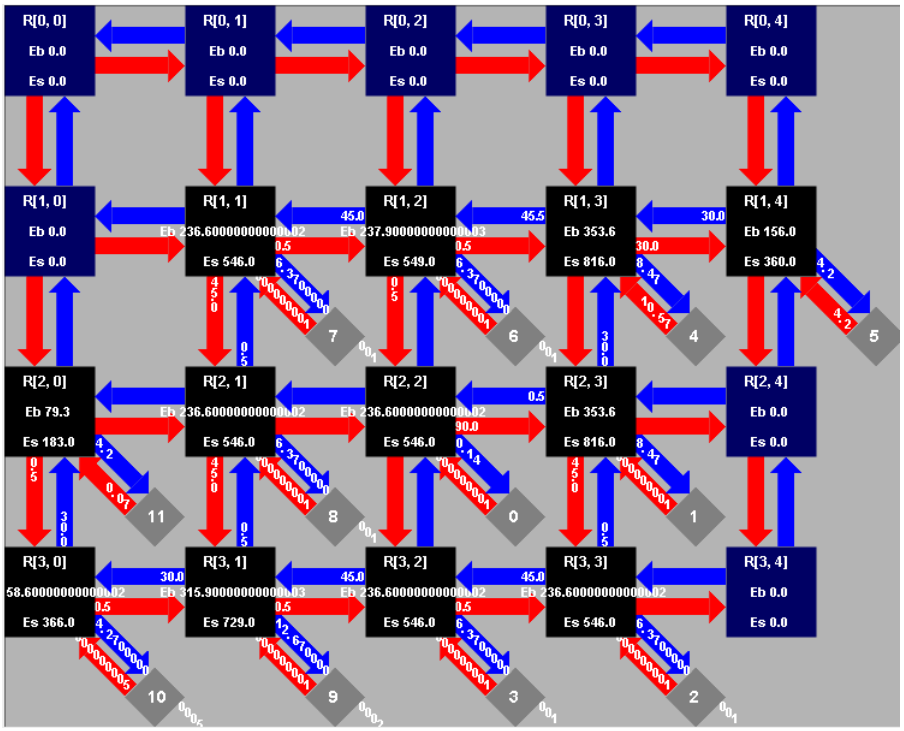
F [D, VFD, TFD, VDF, TDF]

0 [1, 100, 24%, 10, 1%]
1 [2, 100, 18%, 10, 1%]
2 [3, 100, 18%, 10, 1%]
3 [9, 100, 18%, 10, 1%]
0 [4, 100, 12%, 10, 1%]
4 [5, 100, 12%, 100, 12%]
4 [6, 100, 18%, 10, 1%]
6 [7, 100, 18%, 10, 1%]
7 [8, 100, 18%, 10, 1%]
8 [9, 100, 18%, 10, 1%]
9 [10, 100, 12%, 10, 1%]
10 [11, 100, 12%, 10, 1%]

Na Figura abaixo apresenta-se o mesmo grafo da aplicação MWD, nesse caso descrito conforme a modelagem CWM da ferramenta CAFES.

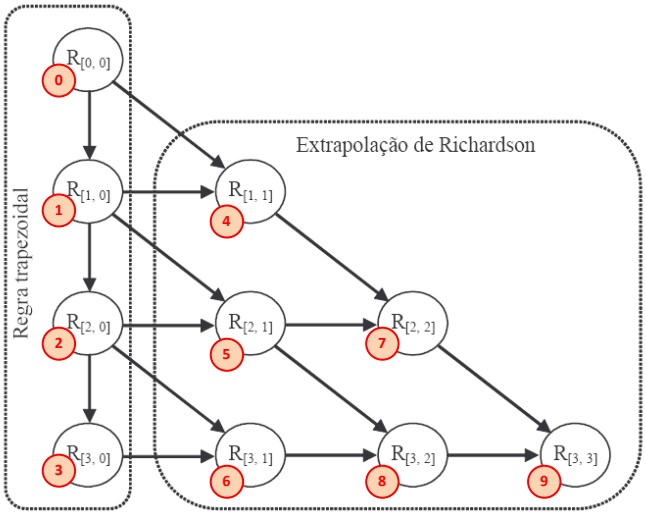


A Figura a seguir apresenta o resultado de mapeamento obtido através da ferramenta CAFES para a aplicação MWD, de acordo com o algoritmo *Simulated Annealing*.

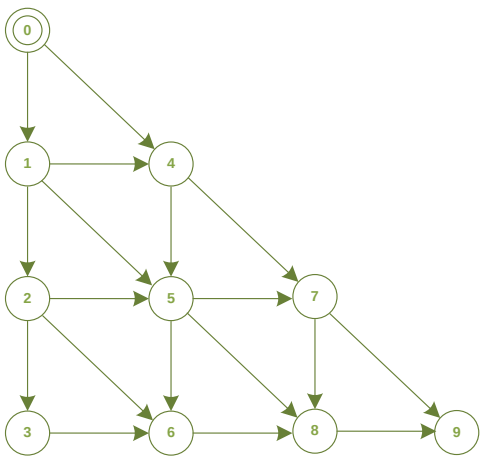


A.4.4. Integral de Romberg

O grafo abaixo representa a aplicação *Integral de Romberg*, obtida a partir do trabalho de Marcon e outros [MAR05d]. Os círculos menores dizem respeito aos identificadores atribuídos a cada uma das tarefas que compõem esta aplicação.



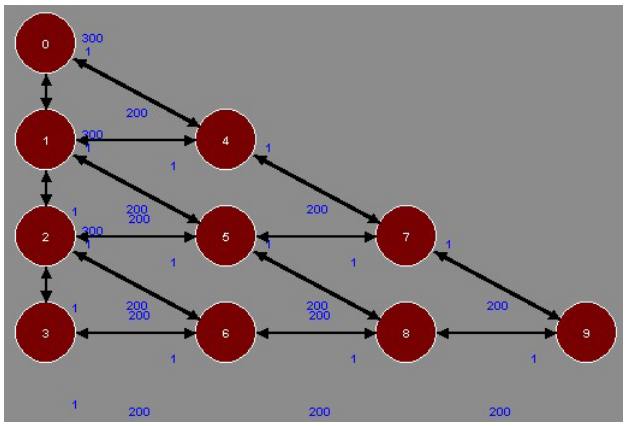
A Figura abaixo apresenta o mesmo grafo anterior, representado agora conforme a modelagem aplicada no presente trabalho. Trata-se da mesma Figura 6.4, apresentada aqui para facilitar sua comparação com a Figura acima.



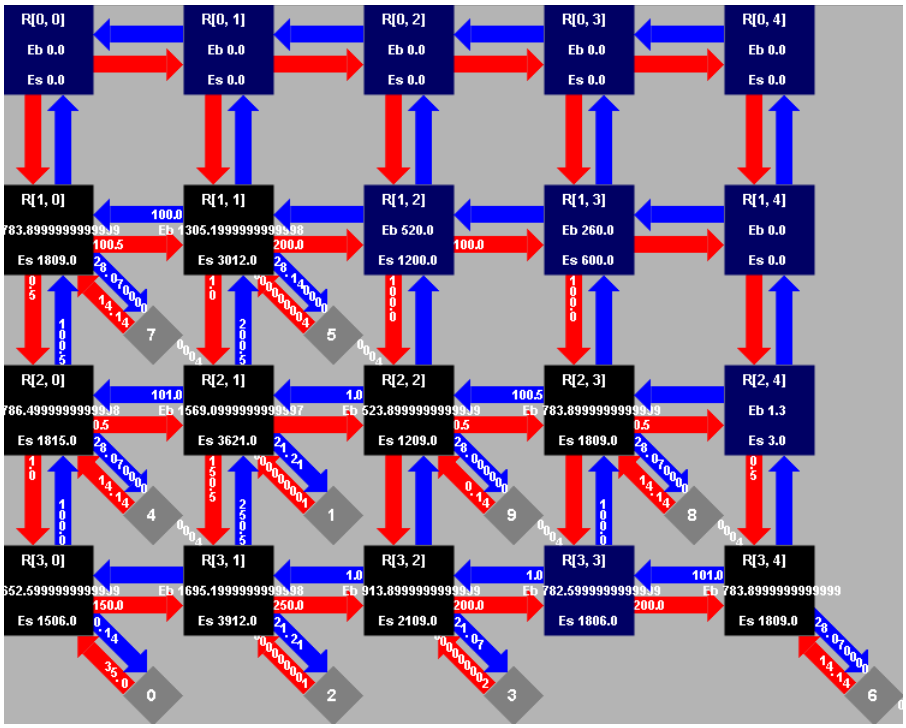
F [D, VFD, TFD,VDF,TDF]

0 [1, 100, 30%, 10, 1%]
1 [2, 100, 30%, 10, 1%]
2 [3, 100, 30%, 10, 1%]
0 [4, 100, 20%, 10, 1%]
1 [4, 100, 20%, 10, 1%]
1 [5, 100, 20%, 10, 1%]
2 [5, 100, 20%, 10, 1%]
2 [6, 100, 20%, 10, 1%]
3 [6, 100, 20%, 10, 1%]
4 [7, 100, 20%, 10, 1%]
5 [7, 100, 20%, 10, 1%]
5 [8, 100, 20%, 10, 1%]
6 [8, 100, 20%, 10, 1%]
7 [9, 100, 20%, 10, 1%]
8 [9, 100, 20%, 10, 1%]

O grafo apresentado na Figura abaixo diz respeito a mesma aplicação *Integral de Romberg*, agora representada conforme a modelagem CWM da ferramenta CAFES.



A Figura a seguir apresenta o resultado de mapeamento obtido através da ferramenta CAFES para a aplicação Romberg, considerando o algoritmo de mapeamento *Tabu Search*.



A.5. Aplicações Empregadas nos Cenários E

No Cenário E foram empregadas as mesmas 4 aplicações do Cenário D (*i. e.* MPEG-4, VOPD, MWD, RBERG). Além dessas, outras 4 aplicações foram geradas através da ferramenta TGFF.

A Figura abaixo apresenta os 8 gráficos mapeados em um MPSoC com dimensões 9x9. O mapeamento resultante da ferramenta CAFES não é apresentado aqui, visto que são geradas várias telas, entretanto o resultado pode ser observado na Figura 6.11.