Universidade Federal do Rio Grande do Sul
Instituto de Informática
Programa de Pós-Graduação em Computação

CÉSAR AUGUSTO MISSIO MARCON

Modelos para o Mapeamento de Aplicações em Infra-estruturas de Comunicação Intrachip

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

Altamiro Amadeu Susin
Orientador

Ney Laert Vilar Calazans
Co-Orientador

Porto Alegre, dezembro de 2005

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Marcon, César Augusto Missio

Modelos para o Mapeamento de Aplicações em Infra-estruturas de Comunicação Intrachip / César Augusto Missio Marcon - Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

192p.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2005. Orientador: Altamiro Amadeu Susin; Co-orientador: Ney Laert Vilar Calazans.

1. Modelos computacionais. 2. Algoritmos de mapeamento. 3. Network-on-Chip (NoC). I. Susin, Altamiro Amadeu. II. Calazans, Ney Laert Vilar. III. Titulo.

AGRADECIMENTOS

A minha esposa que sempre me deu força, estímulo e acreditou no meu sucesso, por mais longa que fosse a jornada. Te amo Vânia.

Aos meus pais Natalino e Lourdes e a minha irmã Denise. Uma família de dar inveja. Sempre estiveram do meu lado, apoiando com todo o amor, carinho e infra-estrutura que puderam propiciar para eu chegar aqui.

A minha afiliada Andrea e ao meu cunhado Lalo pelo acolhimento e brincadeiras em diversos fins de semana.

A família de minha esposa, Volmi, Leda, Chico, Sinara, Veraldo, Volmar, Cristina, Vilton, Titi e Valéria, e a toda a gurizada pelo companheirismo, por desejarem o sucesso deste trabalho, e por momentos de alegria e diversão.

Ao Igor Reis por ter auxiliado no desenvolvimento da primeira versão do *framework* CAFES, sendo este a principal ferramenta aqui utilizada.

Ao amigo Fabiano Hessel, pelas trocas de idéias relativas a trabalhos relacionados com o período de doutoramento na PUCRS, e por ceder bolsistas que auxiliaram na elaboração de ferramentas utilizadas aqui.

Ao meu amigo Luís Ries por contribuições em diversos trabalhos realizados durante o período de doutoramento na PUCRS.

Aos meus colegas e amigos do GAPH, pela amizade, pelo ambiente de trabalho alegre e companheiro, e por diversos auxílios técnicos que contribuíram para a elaboração desta. Em especial a Aline Mello e o Leandro Moller por diversas contribuições técnicas dadas com NoC Hermes e ferramental de apoio, sendo a Hermes a principal infra-estrutura de comunicação aqui avaliada. Ao Luciano Ost que participou na elaboração da ferramenta Maia, sendo de grande importância na geração de NoCs Hermes parametrizáveis. Ao Leonel Tedesco que auxiliou na geração de tráfego para avaliar o consumo de energia e tempo de comunicação de NoCs. Ao Edson Moreno e ao Ewerson Carvalho por discussões técnicas em diversos assuntos relacionados a este trabalho.

Aos meus colegas e amigos da UFRGS. Em especial ao José Carlos Palma pela troca de idéias e trabalhos relacionados com modelos para a avaliação de energia de redes intrachip, sendo estes fundamentais para a avaliação da qualidade dos mapeamentos. Ao Márcio Kreutz por diversos trabalhos práticos e teóricos que auxiliaram na elaboração e concretização deste trabalho. Ao André Borin, por elaborar uma aplicação de imagens que auxiliou na avaliação do primeiro modelo de aplicação utilizado aqui. Ao Renato Hentschke que projetou o simulador Dragon Lemon, servido como ferramenta inicial para estimar o consumo de energia em redes intrachip. A todos os demais colegas Julio Mattos, Lisiane Brisolara, Edgard Farias, Eduardo Brião,

Rodrigo e outros pelo companheirismo e trocas de idéias relacionadas ou não com o tema da Tese.

A professora Beatriz Franciosi pelas dicas de modelos matemáticos usados tanto no período que foi realizado o doutorado na PUCRS quanto no período de realização deste trabalho pela UFRGS.

Aos professores Luigi Carro e Flávio Wagner. Pela troca de idéias e experiências e, em especial, pelo trabalho sugerido e orientado na disciplina de Sistemas Embarcados que foi essencial para impulsionar este trabalho.

Ao César Zeferino pela revisão do volume de Proposta de Tese e diversas sugestões para melhoria e andamento deste trabalho.

Ao meu amigo Fernando Moraes, que participou como co-orientador durante o período de doutoramento na PUCRS, e continuou contribuindo para a elaboração deste trabalho, com idéias e críticas essenciais. Sendo este, junto com o Ney, mentor de diversos outros trabalhos aqui utilizados, como a NoC Hermes e a ferramenta Maia.

Ao meu amigo Ney Calazans, que me orientou durante o período de doutoramento na PUCRS, e passou a ser co-orientador desta Tese de Doutorado, tendo uma contribuição essencial nas fundamentações teóricas e revisão de trabalhos e do volume final desta Tese. Valeu Ney!

Ao meu amigo e orientador Altamiro Susin que me acolheu pela segunda vez. Uma como orientador de mestrado há uma década atrás e outra como orientador de doutorado para a realização deste trabalho. Graças aos seus direcionamentos a Tese tomou rumos que considero terem premiado um bom trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS

ACFSM	Abstract Codesign Finite State Machine
ACP	Application Communication Pattern
ACPM	Application Communication Pattern Model
APCG	APplication Characterization Graph
ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
CAFES	Communication Analysis for Embedded Systems
CDA	Communication Dependence Algorithm
CDCG	Communication Dependence and Computation Graph
CDCM	Communication Dependence and Computation Model
CDF	Control-Data Flow
CDFG	Control-Data Flow Graph
CDG	Communication Dependence Graph
CDM	Communication Dependence Model
CDMA	Code Division Multiple Access
CDL	Communication Dependence List
CF	Control Flow
CI	Circuito Integrado
CTG	Communication Task Graph
CTM	Communication Task Model
CWA	Communication Weighted Algorithm
CWG	Communication Weighted Graph
CWM	Communication Weighted Model
DE	Discrete Event
DF	Data Flow
DSP	Digital Signal Processing
ECWG	Extended Communication Weighted Graph
ECWM	Extended Communication Weighted Model

FLI	Foreign Language Interface
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous, Locally Synchronous
IP core	Intellectual Property core
ITRS	International Technology Roadmap for Semiconductors
NoC	Network-on-Chip
PE	Processing Element
PN	Petri Nets
RTL	Register Transfer Level
SDL	Specification and Description Language
SoC	System-on-Chip
VCC	Virtual Component Codesign
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

LISTA DE SÍMBOLOS

b_i	i-ésimo bloco
B	Conjunto de todos os blocos da aplicação
c_{ij}	Aresta dirigida que caracteriza a dependência de controle e comunicação entre t_i e t_j
C	Conjunto de todos os c_{ij} de um grafo de tarefas da comunicação
d_i	Deadline da tarefa t_i
$d_{ij,abq}$	Atraso total do q-ésimo pacote que parte do núcleo n_a indo até o núcleo n_b, estando n_a mapeado no tile τ_i e n_b mapeado no tile τ_j
d_{Rij}	Atraso de roteamento de um pacote que parte do tile τ_i indo até o tile τ_j
$d_{Pij,abq}$	Atraso de transmissão do q-ésimo pacote que parte do núcleo n_a indo até o núcleo n_b, estando n_a mapeado no tile τ_i e n_b mapeado no tile τ_j
D	Conjunto de todas as dependências entre mensagens dos modelos CDM e CDCM
e_{ij}	Estimativa do consumo de energia da tarefa t_i, quando executada no j-ésimo PE
E_i	Vetor de todos as e_{ij} do núcleo n_i para todos os j PEs
E_{ij}	Energia dinâmica consumida por todos os pacotes transmitidos do núcleo n_i para o núcleo n_j
E_{ijq}	Energia dinâmica consumida pela mensagem q, transmitida do núcleo n_i para o núcleo n_j
$Ebit$	Energia dinâmica consumida devido à transição de um bit
$EBbit$	Energia dinâmica consumida em um meio de armazenamento (buffer) devido à transição de um bit
$ECbit$	Energia dinâmica consumida em uma conexão entre um roteador e um núcleo local ao tile devido à transição de um bit
$EDynRouter$	Energia dinâmica consumida em cada roteador devido aos circuitos que estão operando independente da existência de tráfego na rede
$EDyApp$	Energia dinâmica consumida pela aplicação
$EDyNoC$	Energia dinâmica consumida pela NoC
$EDyTask$	Energia dinâmica consumida pelas tarefas da aplicação

E_{Lbit}	Energia dinâmica consumida em uma conexão entre roteadores (tiles quadrados) devido à transição de um bit
E_{LVbit}	Energia dinâmica consumida em uma conexão vertical entre roteadores devido à transição de um bit
E_{LHbit}	Energia dinâmica consumida em uma conexão horizontal entre roteadores devido à transição de um bit
E_{NoC}	Energia total consumida na NoC
E_{Rbit}	Energia dinâmica consumida em um roteador devido à transição de um bit
E_{Sbit}	Energia dinâmica consumida no circuito de controle e de chaveamento de um roteador devido à transição de um bit
$E_{StDynNoC}$	Parcela de energia consumida na NoC, composta pela potência estática consumida por todos os elementos ativos da rede e composta pela potência dinâmica dos circuitos que operam mesmo na ausência de tráfego
k_{ij}	Número de mensagens transmitidas do núcleo n_i para o núcleo n_j
H	Conjunto de todas as comunicações de uma aplicação modelada via ECWM
h_{ij}	Aresta do grafo ECWG que parte do vértice n_i para o vértice n_j
l	Medida de um lado de um tile quadrado
l_V	Altura de um tile
l_H	Largura de um tile
l_R	Medida de um lado de um roteador quadrado
l_{RH}	Largura de um roteador
l_{RV}	Altura de um roteador
L_R	Medida do lado de um roteador de nível 1 de uma NoC com topologia árvore-gorda
L_{RH}	Largura de um roteador de nível 1 de uma NoC com topologia árvore-gorda
L_{RV}	Altura de um roteador de nível 1 de uma NoC com topologia árvore-gorda
l_L	Medida de uma conexão entre dois roteadores (tiles quadrados)
l_{LV}	Altura de uma conexão entre dois roteadores
l_{LH}	Largura de uma conexão entre dois roteadores
L_L	Medida de uma conexão longa entre dois roteadores de uma NoC com topologia toro dobrado (tiles quadrados)
L_{LV}	Altura de uma conexão longa entre dois roteadores de uma NoC com topologia toro dobrado
L_{LH}	Largura de uma conexão longa entre dois roteadores de uma NoC com topologia toro dobrado

m_q	q-ésima mensagem de uma aplicação
M	Conjunto de todas as mensagens trocadas entre os núcleos de uma aplicação
n_i	i-ésimo núcleo da aplicação
$n_{i \rightarrow j}$	Comunicação que parte do i-ésimo núcleo para o j-ésimo núcleo da aplicação
n_{BI}	Número total de bytes de uma imagem
n_{BPX}	Número de bytes na dimensão X por processador auxiliar
n_{BPY}	Número de bytes na dimensão Y por processador auxiliar
n_{BS}	Número de bytes de cada segmento de uma imagem
n_{BX}	Número de bytes de uma imagem na dimensão X
n_{BY}	Número de bytes de uma imagem na dimensão Y
n_{CV}	Número de bytes de controle de vizinhança
n_{PA}	Número de processadores auxiliares
n_{PAX}	Número de processadores auxiliares em X
n_{PAY}	Número de processadores auxiliares em Y
n_{Fijq}	Número de phits do q-ésimo pacote, sendo este transmitido do núcleo n_i para o núcleo n_j
N	Conjunto de todos os núcleos da aplicação
P_A	Conjunto de todos os pontos de acesso da infra-estrutura de comunicação
p_{Ai}	i-ésimo ponto de acesso
p_R	porta de roteamento
$P_{DynRouter}$	Potência dissipada em cada roteador devido aos circuitos que operam mesmo na ausência de tráfego
P_{Router}	Totalização de $P_{StRouter}$ com $P_{DynRouter}$
$P_{StRouter}$	Potência estática dissipada em cada roteador
$P_{StDynNoC}$	Parcela de potência dissipada na NoC, composta pela potência estática consumida por todos os elementos ativos da rede e composta pela potência dinâmica dos circuitos que operam mesmo na ausência de tráfego
s_{ijq}	Número de transições de bits de m_q enviados no meio físico
t_i	i-ésima tarefa da aplicação
$t_i \rightarrow t_j$	Dependência entre tarefas, indicando que a tarefa t_j não pode iniciar antes da tarefa t_i ter terminado sua execução
T	Conjunto de todas as tarefas da aplicação
t_a	Tempo de computação de um núcleo IP

t_{aq}	Tempo de computação do núcleo que, após terem sido satisfeitas todas as dependências do vértice, precede a transmissão da q-ésima mensagem
t_c	Tempo de contenção gasto na transmissão de um pacote
$texec$	Tempo de execução da aplicação
TOP	Topologia de uma NoC
t_{CPM}	Taxa de comunicação de cada processador auxiliar com a memória
t_{CPPC}	Taxa de comunicação de cada processador auxiliar com o processador central
t_{CPPX}	Taxa de comunicação de cada processador auxiliar com seu vizinho em X
t_{CPPY}	Taxa de comunicação de cada processador auxiliar com seu vizinho em Y
$t_{E_{ij}}$	Tempo de execução da tarefa t_i no j-ésimo PE
t_P	Taxa de processamento
t_R	Número de ciclos necessários para a decisão de roteamento de um pacote em um roteador
t_L	Número de ciclos necessários para transmitir um phit entre roteadores ou entre roteador e núcleo local
t_i	i-ésimo instante de tempo
t_{ij}	Tempo de computação anterior à geração da mensagem de índice j no núcleo n_i
T_i	Vetor de todos os t_{ij} do núcleo n_i para todos os j PEs
w_{ij}	Aresta de CWG que parte do vértice n_i para o vértice n_j
w_{ijq}	Número de bits de m_q enviados no meio físico
W	Número total de bits que trafega na infra-estrutura de comunicação durante a execução da aplicação
α_{TOP}	Função para estimar a energia consumida nos roteadores por onde trafegam os pacotes de uma NoC com topologia TOP
ϕ	Função que associa tarefas a núcleos
η	Número de roteadores de um caminho em uma NoC
φ_{TOP}	Função para estimar a energia consumida nas conexões por onde trafegam os pacotes de uma NoC com topologia TOP
λ	Duração de um período de relógio
σ_{ij}	Número total de transições que ocorre na comunicação do núcleo n_i para o núcleo n_j
τ_i	i-ésimo tile
τ	Conjunto de todos os tiles da infra-estrutura de comunicação

ω_{ij}	Número total de bits enviados da tarefa t_i ou núcleo n_i para a tarefa t_j ou núcleo n_j

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

RESUMO

O projeto de sistemas intrachip (SoCs) é uma atividade de alto grau de complexidade, dados a dimensão de SoCs, na ordem do bilhão de transistores, os requisitos de tempo de desenvolvimento e de consumo de energia, entre outros fatores. A forma de dominar a complexidade de projeto de SoCs inclui dividir a funcionalidade do sistema em módulos de menor complexidade, denominados de núcleos de propriedade intelectual (núcleos IP), interligados por uma infra-estrutura de comunicação.

Enquanto núcleos IP podem ser reusados de outros projetos ou adquiridos de terceiros, a infra-estrutura de comunicação deve sempre ser desenvolvida de forma personalizada para cada SoC. O presente trabalho volta-se para o projeto de infra-estruturas de comunicação eficientes. Questões importantes neste contexto são a eficiência da comunicação, refletida e.g. em medidas de vazão e latência, a redução de área de silício para implementar a comunicação, e a redução da energia consumida na comunicação. Estas questões dependem da escolha da infra-estrutura de comunicação. Barramentos são as infra-estruturas mais usadas nas comunicações intrachip, mas têm sido consideradas como pouco adequadas para servir a necessidade de comunicação de SoCs futuros. Redes intrachip vêm emergindo como um possível melhor candidato. Nesta infra-estrutura de comunicação, um problema a ser resolvido é o posicionamento relativo de núcleos IP dentro da rede, visando otimizar desempenho e reduzir o consumo de energia, no que se denomina aqui *problema de mapeamento*. Dada a complexidade deste problema, considera-se fundamental dispor de modelos para capturar as características da infra-estrutura de comunicação, bem como da aplicação que a emprega.

A principal contribuição deste trabalho é propor e avaliar um conjunto de modelos de computação voltados para a solução do problema de mapeamento de núcleos de propriedade intelectual sobre uma infra-estrutura de comunicação. Três modelos são propostos (CDM, CDCM e ECWM) e comparados, entre si e com três outros disponíveis na literatura (CWM, CTM e ACPM). Embora os modelos sejam genéricos, os estudos de caso restringem-se aqui a infra-estruturas de comunicação do tipo rede intrachip. Dada a diversidade de modelos de mapeamento, propõe-se uma segunda contribuição, o metamodelo Quantidade, Ordem, Dependência (QOD), que relaciona modelos de mapeamento usando os critérios expressos na denominação QOD. Considerando o alto grau de abstração dos modelos empregados, julga-se necessário prover uma conexão com níveis inferiores da hierarquia de projeto. Neste sentido, uma terceira contribuição original do presente trabalho é a proposta de modelos de consumo de energia e tempo de comunicação para redes intrachip. Visando demonstrar a validade de todos os modelos propostos, foram desenvolvidos métodos de uso destes na solução do problema de mapeamento, o que constitui uma quarta contribuição. Estes métodos

incluem algoritmos de mapeamento, estimativas de tempo de execução, consumo de energia e caminhos críticos em infra-estruturas de comunicação. Como quinta contribuição, propõe-se o *framework* CAFES, que integra os métodos desenvolvidos e os modelos de mapeamento em algoritmos computacionais. Uma última contribuição do presente trabalho é um método habilitando a estimativa de consumo de energia para infra-estruturas de comunicação e sua implementação como uma ferramenta computacional.

Palavras-Chave: Mapeamento, modelos, redes intrachip, consumo de energia, tempo de execução.

Models for Application Mapping onto Intrachip Communication Infrastructure

ABSTRACT

The design of Systems-on-Chip (SoCs) is a highly complex task, given the current dimension of SoCs, in the order of a billion transistors, their design time and energy consumption requirements, among other factors. Mastering SoC design complexity requires dividing the system functionality into less complex modules, called intellectual property cores (IP cores), interconnected by a communication infrastructure.

While IP cores can be reused from previous designs or acquired from other parties, the communication infrastructure must always be developed for each SoC mostly from scratch. The present work addresses the efficient design of communication infrastructures. Important issues in this context are communication efficiency, measured e.g. by throughput and latency values, reduction of silicon area and reduction of energy consumption in the communication infrastructure. These issues depend on the choice of the communication infrastructure. Buses are by far the most used infrastructure for intrachip communication today. However, they are growingly considered unsuitable to supply the communication needs of future SoCs. Networks-on-chip (NoCs) are emerging as a possible better candidate to fulfill communication requirements. In NoCs, a problem to be solved is the relative position of each IP core, aiming to optimize performance and to reduce energy consumption. This is called here the *mapping problem*. Given the complexity of this problem, it is fundamental to use models to capture the characteristics of both, the communication infrastructure and the associated application.

The main contribution of this work is to propose and to evaluate a set of models of computation directed to solve the IP core mapping problem on communication infrastructures. Three models are proposed (CDM, CDCM and ECWM) and compared, among themselves and with three other models available in the literature (CWM, CTM and ACPM). Although these are generic models, most case studies here are restricted to NoC communication infrastructures. Given the diversity of mapping models, there is a second contribution, the *Quantity*, *Order* and *Dependence* metamodel (QOD). This metamodel relates mapping models using the criteria expressed in its denomination (QOD). Considering the high level of abstraction of the employed models, it is necessary to provide a connection with lower levels of design abstraction. A third original contribution of the present work is the proposition of energy consumption and communication time models for NoCs. Aiming to demonstrate the validity of all mentioned models, methods based on them to solve the mapping problem have been developed, which constitutes a fourth contribution. These methods include mapping algorithms, execution time estimation, energy consumption estimation and critical path estimation in communication infrastructures. As a fifth contribution, the

CAFES framework is proposed to integrate the developed methods and the mapping models into computational algorithms. A last contribution of the present work is a method enabling the estimation of energy consumption for communication infrastructures and its implementation as a computational tool.

Keywords: Mapping, models, network-on-chip, energy consumption, execution time.

1 INTRODUÇÃO

A crescente complexidade dos sistemas eletrônicos tem conduzido projetistas e pesquisadores da área de ferramentas de apoio ao projeto a elevar cada vez mais o nível de abstração de atividades como especificação, validação e síntese destes sistemas. O objetivo primordial é tornar gerenciável a complexidade do processo de projetar tais sistemas, e conseqüentemente reduzir o tempo necessário para o produto chegar ao mercado (em inglês, *time-to-market*). Trabalhando em alto nível e com o apoio de ferramentas adequadas, o projetista pode se concentrar em atividades mais relevantes, como a exploração do espaço de soluções dos algoritmos que serão usados, deixando as etapas de implementação de baixo nível de abstração para serem realizadas de forma automática ou semi-automática por ambientes de CAD (em inglês, *Computer Aided Design*).

O projeto de sistemas eletrônicos de grande porte é tarefa por demais complexa devido, entre outros motivos, ao número elevado de componentes e variáveis interdependentes necessários para a sua realização. A implementação destes sistemas é facilitada pelo uso de ferramentas de projeto, geralmente compondo um sistema de desenvolvimento integrado. Estes sistemas de desenvolvimento incluem métodos que permitem a captura, validação e síntese dos sistemas eletrônicos, através de um conjunto de modelos. De Micheli (1995) lembra que a maioria dos sistemas eletrônicos digitais modernos são programáveis, consistindo de componentes de software e componentes de hardware, tais como memórias, barramentos e unidades lógicas e aritméticas. Utiliza-se aqui o termo *sistemas computacionais* para designar sistemas programáveis, e projeto integrado de *hardware* e *software* (em inglês, *hardware*/*software codesign*, ou apenas *codesign*), para se referir ao projeto de sistemas computacionais. Sistemas embarcados ou sistemas embutidos (em inglês, *embedded systems*) são considerados aqui como sistemas computacionais que executam uma função específica. Eles possuem a mesma estrutura geral de um computador, mas a especificidade de suas tarefas faz com que não sejam nem usados nem percebidos como tal (DE MICHELI, 1995).

Sistemas embarcados são divididos em classes, de acordo com a estrutura das aplicações a que se destinam. A partir daí, sistemas de apoio a codesign podem ser desenvolvidos para estas classes. Exemplos são os sistemas embarcados reativos (BENVENISTE; BERRY, 1991), aqueles que reagem continuamente a estímulos provenientes do ambiente, retornando outros estímulos, e cujas reações devem ocorrer em um ritmo ditado pelo ambiente; e os sistemas embarcados de tempo real, aqueles sujeitos a limitações de temporização externa (LAVAGNO; SANGIOVANNI-VINCENTELLI; HSIEH, 1995). Estas classes, por sua vez, podem ser combinadas e/ou subdivididas em domínios de aplicação, produzindo os sistemas embarcados para a área de processamento digital de sinais, os sistemas embarcados para as áreas de telecomunicações, entre outros.

1.1 Descrição do Problema

Requisitos são aqui definidos como informações não funcionais desejadas para um sistema, tal como minimização do consumo de energia, redução da área ocupada e redução do tempo de operação. *Restrições* são definidas como informações que delimitam um sistema, tal como o máximo consumo de energia e o máximo número de pinos de entrada e saída. O projeto de um sistema computacional é guiado por estes requisitos e restrições, que determinam a natureza dos elementos computacionais que o implementarão. Sciuto (2000) expõe que o mercado força a redução de custos e tempo de projeto, levando os projetistas a deslocar a maior parte da funcionalidade dos sistemas embarcados para software. Por outro lado, as funcionalidades que necessitam de alto desempenho são normalmente implementadas em componentes de hardware dedicado. Os pesquisadores de ferramentas de CAD dedicam-se a conceber métodos para automatizar as diversas atividades de projeto, permitindo assim uma distribuição mais adequada da funcionalidade do sistema computacional entre os componentes de software e de hardware, de forma a atender requisitos e restrições de projeto. Várias são as atividades de projeto para a construção de sistemas computacionais embarcados, incluindo: especificação, particionamento, escalonamento, alocação, síntese, mapeamento e validação.

Um fluxo típico de projeto para sistemas computacionais está ilustrado na Figura 1.1. Nesta, elipses representam descrições, retângulos simbolizam ferramentas que operam sobre descrições, arestas cheias simbolizam o fluxo de projeto, arestas tracejadas indicam representações do sistema sobre as quais ocorrem atividades de validação, retângulos pontilhados com bordas arredondadas (1, 2, 3, 4 e 5) representam diferentes descrições de hardware e software, e paralelogramos representam os componentes onde serão implementadas as descrições. Na Figura está destacado o retângulo que contém a atividade de mapeamento, por ser o foco deste trabalho.

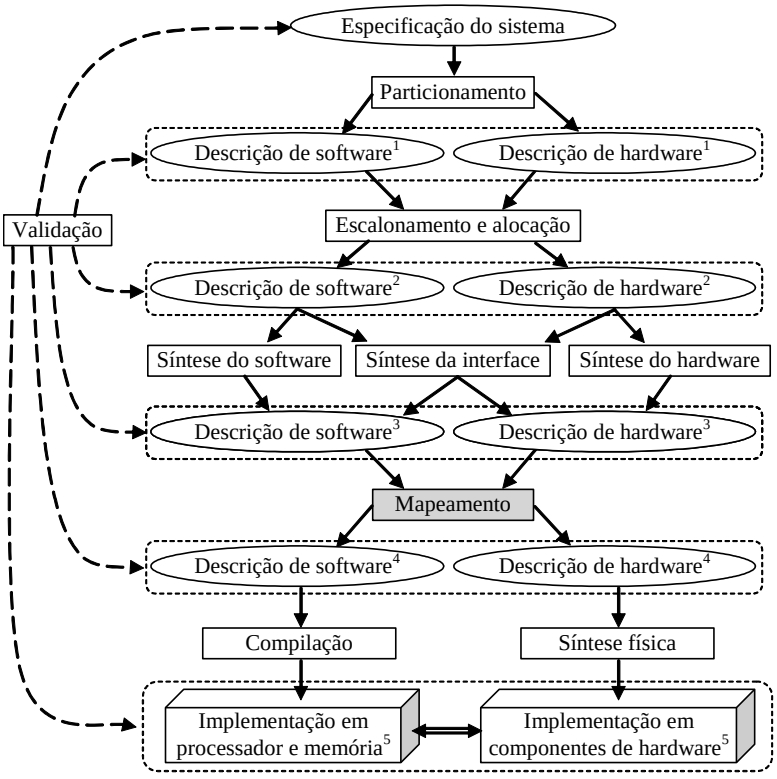


Figura 1.1: Fluxo típico de projeto para sistemas computacionais.

Para este trabalho, um *modelo* é uma abstração de um objeto, empregada para viabilizar o raciocínio sobre este. Uma *descrição* é uma representação deste modelo. Uma *especificação* é a descrição de mais alto nível de um sistema. Dela é que normalmente parte o processo de concepção deste sistema. A modelagem de um sistema computacional inclui um formalismo de descrição utilizado pelos projetistas para definir as funcionalidades deste. O formalismo de descrição pode provir de uma linguagem de programação, tal como C ou Occam, uma linguagem de descrição de hardware, tal como Verilog ou VHDL, ou uma linguagem específica para o projeto conjunto de hardware e software, derivada ou não das anteriores, tais como SystemC (SYSTEMC COMMUNITY, 2005), HardwareC (GUPTA, 1994), Esterel (BENVENISTE; BERRY, 1991) e SDL (MARCON et al., 2002c). A descrição de um componente com estas linguagens não implica que o mesmo seja forçosamente implementado de acordo com o modelo subjacente à linguagem, muito embora haja uma tendência natural para que isto ocorra. As descrições dos componentes devem ser passíveis de tratamento automatizado ou semi-automatizado por métodos e ferramentas de níveis de abstração inferiores, ou sejam, compilação para o software e sínteses de alto nível, lógica e física para o hardware. O modelo formal por sua vez, define como a descrição representará o sistema computacional para que dela seja possível extrair as informações necessárias para a realização de atividades posteriores.

O *particionamento* é o processo de dividir ou reagrupar subsistemas gerando uma *partição*, que é o sistema completo, e onde cada subsistema é denominado de *bloco*. Os blocos de uma partição são implementados por componentes de hardware e/ou software. A implementação física destes componentes na *arquitetura alvo*[1] é realizada em *tiles*[2]. O particionamento leva em consideração requisitos e restrições de projeto e características da arquitetura alvo. O particionamento é geralmente auxiliado por estimativas do sistema computacional frente a requisitos, tal como minimização do consumo de área; e restrições, tal como máximo consumo de energia de um núcleo IP. Estas estimativas permitem determinar quais partes da especificação serão direcionadas para componentes de hardware e quais serão direcionadas para componentes de software. A implementação destes componentes, por outro lado, será realizada pela síntese de hardware e/ou de software. A maioria dos sistemas de síntese utiliza um formato interno para representar o sistema particionado. O *formato interno* é uma descrição que contém informações para viabilizar etapas de projeto subseqüentes, tais como a interface de comunicação entre os blocos da partição. A grande maioria dos sistemas de apoio ao projeto possui mecanismos para guiar o projetista na escolha da melhor partição, seja automatizando as atividades de avaliação da qualidade de uma partição candidata, seja auxiliando na produção rápida de partições de qualidade a partir da especificação. Todavia, a complexidade e a influência do particionamento no

[1] Um recurso computacional representa aqui um componente de hardware e/ou software. *Arquitetura alvo* é o conjunto de recursos computacionais inter-relacionados onde o sistema é implementado. Para uma implementação com codesign, uma arquitetura alvo é geralmente composta por: (*i*) componentes de hardware, tais como ASICs (em inglês, *Application Specific Integrated Circuits*), memórias, FPGAs (em inglês, *Field Programmable Gate Arrays*) e processadores; (*ii*) componentes de software, tais como sistemas operacionais, programas concorrentes e procedimentos; e (*iii*) recursos de interconexão, ou seja, hardware utilizado para comunicação entre os componentes de hardware e software, tais como: barramentos e memória compartilhada.

[2] Neste trabalho, denomina-se *tile* uma região de uma arquitetura alvo onde são implementados componentes de hardware e/ou software, aqui chamados de *núcleos*, e cuja comunicação com o resto do sistema ocorre através de pontos de acesso a uma infra-estrutura de comunicação da arquitetura alvo.

desempenho do produto final são tais que grande parte dos sistemas existentes, até o momento, advoga a necessidade de alguma forma de interação com o projetista. Exceções à regra existem, tal como no algoritmo de particionamento automático proposto por Barros (1993) e Zou, Zhuang e Chen (2004). Abordagens intermediárias de particionamento consistem em partir de uma descrição inicial de apenas software (ERNST; HENKEL, 1992), ou hardware (GUPTA, 1994) e prover meios para automaticamente migrarem módulos para hardware ou para software, respectivamente. Entretanto, o particionamento não é apenas uma atividade que separa as tarefas que serão realizadas em componentes de hardware das tarefas que serão realizadas em componentes de software. Esta atividade também pode agrupar tarefas. Este agrupamento pode ser guiado por funções objetivo de particionamento que levam em consideração requisitos de projeto. Um exemplo é o problema de particionamento de tarefas de um sistema computacional (aqui chamado de *aplicação*) em blocos (aqui chamados de *núcleos*) objetivando reduzir o volume de comunicação entre tarefas, de forma a agrupar tarefas que se comunica muito no mesmo núcleo.

O *escalonamento* é a atividade que gera a ordem parcial de execução das tarefas da aplicação. A *alocação* é a atividade que direciona a implementação da aplicação aos recursos computacionais disponíveis, produzindo uma nova ordem parcial, compatível com a primeira. O escalonamento de tarefas e a alocação de recursos computacionais são atividades que geralmente sucedem o particionamento. Estas atividades agregam novas informações ao formato interno, tal como a ordenação parcial das tarefas e os *deadlines*[3] para a execução das mesmas. As atividades de particionamento, escalonamento e alocação estão intimamente relacionadas. Mesmo assim, devido à complexidade das mesmas, os sistemas de apoio ao projeto muitas vezes realizam estas atividades separadamente.

A *síntese* é a atividade que traduz uma descrição mais abstrata para uma descrição menos abstrata (EDWARDS et al., 1997). A síntese de software a partir do formato interno normalmente implica a transformação do conjunto de estruturas concorrentes do modelo para um código seqüencial. Assim, algoritmos de escalonamento devem ser empregados. O objetivo destes algoritmos é transformar a ordem parcial do modelo formal em uma ordem total equivalente, normalmente exigida em software. A síntese do hardware, por sua vez, procede de forma a extrair o máximo de paralelismo da descrição. Quando o modelo subjacente à descrição é de software, a síntese de hardware transforma a ordem total em uma ordem parcial equivalente. A comunicação entre a implementação de software e a implementação de hardware implica a definição e geração de interfaces de comunicação, tanto no lado do hardware, quanto no lado do software. Esta última atividade é chamada de síntese da interface de hardware/software. Estas atividades podem ser complexas, devido à necessidade de manutenção não apenas da funcionalidade original, mas também dos requisitos de desempenho do sistema.

O *mapeamento* de uma aplicação é uma função que associa os blocos da aplicação a tiles conectados a *pontos de acesso*[4] de uma infra-estrutura de comunicação. O mapeamento de uma aplicação é geralmente considerado uma das atividades da

---

[3] *Deadline* é o tempo limite para execução de uma tarefa.

[4] *Ponto de acesso* é uma posição de uma infra-estrutura de comunicação que permite que um núcleo seja conectado. Esta posição, muitas vezes é referenciada por um endereço, que habilita a comunicação com os demais núcleos da aplicação.

síntese, onde, a partir de uma linguagem de descrição, as funcionalidades da aplicação são organizadas. O mapeamento transforma uma especificação funcional em uma arquitetura e um conjunto de funções para as unidades arquiteturais. Assim, associar blocos do sistema particionado a pontos de acesso de uma infra-estrutura de comunicação é uma tarefa de mapeamento. Normalmente, as entradas do mapeamento são o *formato interno*, proveniente das etapas de particionamento e escalonamento, a *descrição da infra-estrutura de comunicação* e os *requisitos e restrições do projeto de aplicações*. Edwards et al. (1997) expõem que funções objetivo para cálculo do mapeamento devem considerar principalmente informações de tempo, custo dos componentes e consumo de energia, onde a importância relativa de cada fator depende fortemente do tipo de aplicação. A necessidade exposta por Edwards pode ser facilmente observada, por exemplo, quando sistemas projetados para operar com tolerância a falhas (e.g. sistemas espaciais) são comparados com sistemas projetados para apresentar baixo consumo de energia (e.g. sistemas de telefonia móvel). Para os primeiros, a importância relativa do número de componentes e da área ocupada é muito menor que para os últimos. Isto ocorre porque os sistemas tolerantes a falhas geralmente utilizam redundância de componentes, implicando o aumento no consumo de energia.

A *validação* fornece ao projetista garantias de que o sistema está conforme especificado para uma dada etapa do projeto. Quando realizada antes do particionamento, é uma validação geral da funcionalidade do sistema, apta a capturar erros mais grosseiros, cometidos no momento da especificação. Quando a validação é realizada após o particionamento, ela pode capturar problemas sutis da interação entre os componentes, produzidos, por exemplo, pelo particionamento manual. A validação pode ser realizada através de atividades de verificação e/ou teste. A *verificação* é um processo usado para demonstrar que a intenção do projetista é preservada em uma dada implementação (BERGERON, 2002), e normalmente é dividida em verificação formal e verificação funcional. A *verificação formal* prova que um sistema se comporta ou não conforme sua especificação. Contudo, a complexidade desta abordagem é muito grande para verificar um sistema completo, de forma que a mesma é empregada apenas na validação de módulos críticos do sistema e para guiar a síntese. A *verificação funcional* mostra se um sistema tem funcionalidade de acordo com o especificado. Este é o caso da *simulação*, que permite ao projetista avaliar a funcionalidade de um sistema, injetando estímulos neste e comparando se os sinais obtidos correspondem ao esperado. *Teste* é o processo que verifica se um projeto foi manufaturado corretamente. Normalmente, este é dividido em três (RABAEY, 1996): (*i*) *diagnóstico* que é usado durante a depuração de um circuito para identificar e localizar possíveis falhas; (*ii*) *teste funcional* que determina se o componente fabricado tem ou não a funcionalidade planejada; e (*iii*) *teste paramétrico* que verifica parâmetros como margem de ruído e atraso de propagação sobre uma variedade de condições de trabalho, tais como diferentes temperaturas e tensões.

No fluxo de projeto apresentado na Figura 1.1, o projetista parte da especificação de um sistema, analisando o conjunto de requisitos e restrições para obtenção de um ou mais modelos que representem o sistema computacional. Obtidos os modelos, devem ser extraídas informações que permitam direcionar a implementação do mesmo. Estas informações geralmente estão contidas no conjunto de requisitos e restrições do projeto. O particionamento gera novas descrições, tanto para os blocos que serão implementados em componentes de hardware, quanto para os blocos que serão implementados em componentes de software. Ambas as descrições passam pelo processo de síntese, que gera três elementos distintos: o software, o hardware e a

interface de comunicação entre o hardware e o software. Uma vez definidos os núcleos de hardware e/ou software da aplicação, o mapeamento associa estes aos pontos de acesso da infra-estrutura de comunicação. Para todas as atividades do fluxo de projeto podem existir estágios de validação, por verificação formal ou simulação, que permitem avaliar o quão correta está a descrição obtida.

1.2 Motivação

A tecnologia atual permite que sejam concebidos dispositivos com centenas de milhões de transistores. Em conseqüência, diversos sistemas computacionais, sobretudo os sistemas embarcados, que eram implementados de forma discreta podem ser implementados de forma integrada em um único circuito. Estes sistemas são conhecidos por sistemas intrachip, em inglês *Systems-on-Chip* (SoCs).

Esta evolução tecnológica traz algumas diferenças para o processo de projetar sistemas, que por sua vez forçam o estudo de novos métodos. Entre as diferenças está a execução do software, geralmente associada a um processador de uso genérico mais uma memória externa, que passa a ser efetuada em um único circuito integrado (CI). A execução do hardware, muitas vezes realizada com componentes discretos, pode ser obtida com núcleos de propriedade intelectual (em inglês, *intellectual property cores*[5]) mapeados em circuitos dedicados internos ao CI. Um grande ganho desta abordagem está na comunicação, pois reduz a perda de desempenho causada pela troca de informações entre CIs distintos. Caso os componentes de hardware e software estejam integrados em um único CI, o desempenho global do sistema será muito maior, reduzindo ou eliminando gargalos de comunicação.

O *International Technology Roadmap for Semiconductors* (ITRS) (2005) prevê que circuitos integrados fabricados na próxima década terão dezenas de bilhões de transistores. As tecnologias dos SoCs permitirão a implementação de circuitos com transistores de dimensão em torno de 50 nm e freqüência de operação acima de 10 GHz. Estas tecnologias, em inglês chamadas de *deep submicron technologies*, gerarão novos desafios de projeto, devido a fatores como dissipação de potência e atraso nas interconexões. As interconexões físicas intrachip serão fator limitante para o tempo de execução e para o consumo de energia das aplicações. O atraso de propagação dos sinais irá exceder o período de relógio. Estes efeitos são ainda mais problemáticos em sinais de sincronismo, tal como os sinais de relógio em sistemas síncronos. A sincronização dos dispositivos com um único relógio será extremamente difícil ou até impossível (IYER; MARCULESCU, 2002).

O ITRS (2005) também prevê que o consumo de energia global deve crescer nas tecnologias futuras, pois a redução da tensão de alimentação não será suficiente para balancear o crescimento da complexidade dos sistemas. Enquanto a energia consumida tanto pela computação, quanto pelo armazenamento, é reduzida pelo escalamento, a energia da comunicação global não será reduzida. Ao contrário, projetos baseados nas técnicas de otimização de atrasos atuais mostram que a comunicação global intrachip irá exigir um crescimento ainda maior no consumo de energia (DUARTE, 2002). Conseqüentemente, para as futuras tecnologias será crescente a necessidade de

[5] *Intellectual property core* é normalmente chamado de *IP core* ou apenas *core*. Neste trabalho será utilizada apenas a denominação *núcleo*, que é a tradução do termo *core*.

minimização da energia causada pela comunicação.

Com o intuito de resolver os problemas inerentes aos futuros SoCs, muitos projetistas propõem mudar o paradigma de projeto inteiramente síncrono para um paradigma globalmente assíncrono e localmente síncrono, em inglês *Globally Asynchronous, Locally Synchronous* (GALS) (IYER; MARCULESCU, 2002). Nos projetos GALS existem vários sinais de *relógio físicos e ou lógicos*[6], permitindo que a aplicação seja subdividida em domínios síncronos. Um recurso assíncrono da comunicação fornece a interface entre domínios síncronos. As redes intrachip, em inglês *Networks-on-Chip* (NoCs), são infra-estruturas de comunicação compostas por um conjunto de roteadores interconectados por canais de comunicação, que podem fornecer uma comunicação assíncrona entre domínios síncronos. Cada domínio síncrono é inserido dentro de uma região conhecida por tile. Cada tile é composto por um roteador, canais de comunicação e, *geralmente um núcleo*[7]. O roteador é responsável pela comunicação do núcleo conectado ao tile com os demais núcleos fisicamente posicionados nos demais tiles. NoCs podem ser projetadas para atender o paradigma GALS. Além de escalabilidade, NoCs apresentam alto potencial para reusabilidade, confiabilidade, e dissipação de potência eficiente (DALLY; TOWLES, 2001).

1.3 Contexto do Trabalho

O mapeamento de núcleos em infra-estruturas de comunicação se assemelha ao problema de cobertura de grafos, que reconhecidamente é um problema NP-completo. Por exemplo, supondo que uma aplicação utilize como infra-estrutura de comunicação uma NoC, e sendo n o número de tiles desta NoC e também o número de núcleos da aplicação, o problema de mapeamento permite $n!$ mapas possíveis. Em poucos anos, espera-se acomodar mais de 10×10 tiles em uma NoC (KUMAR et al., 2002). A execução eficiente de uma aplicação em uma infra-estrutura de comunicação com estas características requer estratégias eficientes de mapeamento. Este trabalho foca o processo de mapear blocos em tiles, de forma a atender os requisitos especificados e respeitar as restrições impostas pelo projeto. Esta atividade está hachurada na Figura 1.1.

Dado o exposto acima, este trabalho utilizará NoCs como estudo de caso de infra-estrutura de comunicação. Os compromissos de computação e comunicação são levados em consideração para determinar o melhor mapeamento dos núcleos em tiles. Os requisitos de projeto considerados neste trabalho são: minimização do tempo de execução da aplicação e redução da energia consumida pela infra-estrutura de comunicação.

---

[6] Uma implementação GALS não implica a necessidade de ter um sinal de relógio distinto para cada domínio síncrono, mas sim, que cada domínio trate do(s) seu(s) relógio(s) de forma independente dos demais domínios. Ou seja, os relógios podem ser fisicamente os mesmos, mas tratados logicamente como relógios distintos.

[7] A existência ou não de núcleos associados a roteadores depende da topologia da rede. Maiores detalhes sobre o assunto são abordados na Seção 3.2.1.

1.4 Objetivos Específicos do Trabalho

Fazem parte do escopo deste trabalho a proposta e a avaliação de modelos de aplicações para a atividade de mapeamento em infra-estruturas de comunicação. A essência do trabalho não está nas infra-estruturas de comunicação. Todavia, para que seja possível obter dados comparativos da modelagem da aplicação sobre a arquitetura alvo, foram definidas NoCs como estudo de caso de infra-estrutura de comunicação. Três topologias são avaliadas: malha, toro dobrado e árvore-gorda, mas a maioria dos resultados é obtido com NoCs de topologia malha, chaveamento wormhole e roteamento determinístico XY. São cinco os objetivos específicos deste trabalho:

1. Estudar, avaliar e propor métodos e *modelos computacionais*[8] para a atividade de mapeamento, enfocando a computação e a comunicação da aplicação;
2. Comparar formatos internos e seus modelos subjacentes com relação à capacidade destes em capturar a funcionalidade da aplicação e obter estimativas de qualidade para os requisitos de projeto utilizados na atividade de mapeamento;
3. Propor um metamodelo para modelos de mapeamento;
4. Avaliar e propor a evolução de modelos para estimar consumo de energia em NoCs;
5. Elaborar um *framework* para avaliar mapeamentos de aplicações em NoCs.

Para alcançar os objetivos específicos foram planejadas várias atividades. Dentre estas se destacam:

1. Estudo de trabalhos relacionados, valorizando aqueles que modelam aplicações e infra-estruturas de comunicação para a atividade de mapeamento;
2. Síntese de NoCs para o nível elétrico, com a finalidade de obter avaliações precisas sobre o consumo de energia e validar o modelo de estimativa de potência apresentado aqui. As simulações elétricas permitem avaliar a qualidade do modelo de consumo de energia em nível sistêmico e propor alterações para melhorar a sua precisão;
3. Implementação de um método para validar modelos de energia e tempo de comunicação em redes intrachip. O método implica o uso de ferramentas comerciais e ferramentas desenvolvidas pelo autor.
4. Execução de aplicações sobre as NoCs em nível de transferência de registradores (em inglês, *Register Transfer Level* ou RTL), para obter o tempo de comunicação preciso em ciclos de relógio. Estes resultados permitem avaliar o modelo de tempo de comunicação em nível sistêmico para a NoC escolhida;
5. Desenvolvimento de uma técnica para avaliar o consumo de energia de infra-estruturas de comunicação descritas em VHDL;

---

[8] Utiliza-se neste trabalho o termo *modelo computacional* para se referir aos modelos de sistemas compostos por componentes de *hardware* e *software*.

6. Desenvolvimento de um *framework* que suporte várias ferramentas, tais como simuladores e editores, para auxiliar no mapeamento da aplicação. Este *framework* deve permitir a modelagem de aplicações e de infra-estruturas de comunicação;
7. Implementação de algoritmos para manipular os formatos internos. Com a finalidade de comparar os formatos internos, serão descritas NoCs no nível sistêmico e modelados sobre estas o consumo de energia e o tempo de comunicação;
8. Determinação de métricas que permitam avaliar os compromissos de computação e comunicação, de forma a estimar em alto nível a qualidade do mapeamento de núcleos da aplicação em tiles da infra-estrutura de comunicação.

1.5 Trabalhos Relacionados

Este trabalho demanda a análise de propostas no âmbito de infra-estruturas de comunicação intrachip, e na área de modelagem e síntese de sistemas computacionais. Esta Seção foca apenas os trabalhos relevantes relacionados à modelagem de aplicações com o objetivo de mapeá-las em infra-estruturas de comunicação intrachip atendendo os requisitos de projeto. A principal infra-estrutura de comunicação abordada é a rede intrachip, dado que esta é usada como estudo de caso deste trabalho.

Raghunathany, Srivastavay e Gupta (2003) apresentam um levantamento sobre infra-estruturas de comunicação com enfoque no baixo consumo de energia. Os autores descrevem diversas infra-estruturas de comunicação, tais como *barramento segmentado*[9], arquiteturas baseadas em roteador, NoCs e até tecnologias emergentes como barramentos baseados em multiplexação por CDMA (*Code Division Multiple Access*) e interconexões sem fio. A principal motivação para o trabalho é o fato de que cerca de 50% do consumo de energia dos sistemas atuais ocorre nas interconexões, e este consumo tende a aumentar para os futuros sistemas a serem implementados com tecnologias submicrônicas.

Para muitos recursos de comunicação, tal como o *barramento monolítico*[10], não há sentido a atividade de mapeamento de núcleos, pois a posição em que estes são mapeados pouco ou nada afeta a implementação final. Por outro lado, recursos de comunicação como NoC, e barramento segmentado ou *barramento hierárquico*[11] requerem a atividade de mapeamento para uma implementação eficiente. O

---

[9] *Barramento segmentado* é uma infra-estrutura de comunicação composta por segmentos de comunicação interligados através de chaves, onde mais de um elemento computacional conectado a esta estrutura pode enviar informações ao mesmo tempo, contanto que estes elementos não compartilhem do mesmo segmento de comunicação.

[10] Implementar um sistema com um *barramento monolítico* implica ter um único segmento de comunicação que implementa toda a infra-estrutura, onde todos os elementos computacionais se comunicam através do compartilhamento temporal deste segmento.

[11] *Barramento hierárquico* é uma organização lógica de uma infra-estrutura de comunicação do tipo barramento segmentado. A segmentação da infra-estrutura é montada de forma a determinar níveis de segmentos que servem de comunicação com elementos computacionais e com outros barramentos de outros níveis. Em cada nível podem existir um ou mais segmentos.

mapeamento de núcleos em infra-estruturas de comunicação é uma atividade recente, por este motivo a quantidade de trabalhos relacionados é escassa. Este problema é ainda mais saliente quando os recursos de comunicação são barramentos, sejam estes distribuído ou hierárquico. Outro motivo para a escassez de trabalhos relacionados, é a baixa regularidade desta infra-estrutura de comunicação, que implica a análise desta ser dificilmente estendida para as demais arquiteturas de barramentos. Entre estes trabalhos destacam-se (GIUSTO; DEMMELER, 2004), (HSIEH; PEDRAM, 2002) e (LAHIRI; RAGHUNATHAN; DEY, 2004).

Giusto e Demmeler (2004) apresentam uma plataforma virtual baseada no VCC (*Virtual Component Codesign*) da Cadence para análise e validação de aplicações automotivas. Esta plataforma modela unidades eletrônicas que se comunicam através de barramentos. A modelagem parte de uma descrição da aplicação particionada em componentes de hardware e software. Estes componentes são mapeados para uma arquitetura de barramento hierárquico, onde o foco principal é a exploração do espaço de projeto pela avaliação de possíveis mapeamentos, troca de prioridade das tarefas e alteração dos protocolos de comunicação.

Hsieh e Pedram (2002) comparam sistemas que utilizam barramento monolítico com sistemas que utilizam barramento segmentado. Os autores propõem uma arquitetura com barramento segmentado para reduzir o consumo de energia e aumentar a largura de banda. Os autores mostram que se uma aplicação for implementada sobre uma arquitetura de barramento segmentado, o consumo de energia pode ser reduzido em até 50%, se comparado com uma implementação equivalente, implementada em uma arquitetura de barramento monolítico. Este percentual depende do tipo de aplicação que determina os dados que irão trafegar na infra-estrutura de comunicação e do mapeamento dos elementos computacionais nesta infra-estrutura de comunicação.

Lahiri, Raghunathan e Dey (2004) apresentam uma técnica de mapeamento de núcleos em arquiteturas que se comunicam através de arquiteturas do tipo barramento segmentado. Esta técnica permite obter até 53% de redução no tempo de comunicação da aplicação quando comparado com resultados obtidos ad hoc.

As redes intrachip são infra-estruturas de comunicação mais recentes que os barramentos e por este motivo o ferramental disponível para a síntese de sistemas através destes recursos ainda é incipiente. Este é um dos principais motivos pelo qual as NoCs compõem uma parcela ainda muito pequena das plataformas existentes. Dally e Towles (2001) analisam várias infra-estruturas de comunicação e concluem que as NoCs são adequadas para projetos modulares, devido a sua alta escalabilidade e regularidade com baixo custo extra de área se comparado com soluções *ad hoc*. A sua alta escalabilidade é um dos principais motivos para a maior quantidade de trabalhos relacionados ao mapeamento de núcleos. Entre estes trabalhos referenciam-se aqui (BOLOTIN, 2004) (HU; MARCULESCU, 2003) (HU; MARCULESCU, 2004) (HU; MARCULESCU, 2005) (MURALI; DE MICHELI, 2004a) (MURALI; DE MICHELI, 2004b) (RHEE; JEONG; HA, 2004) (YE; BENINI; DE MICHELI, 2002) (YE; BENINI; DE MICHELI, 2003) e (YE; BENINI; DE MICHELI, 2004).

Hu e Marculescu (2003) propõem uma abordagem de mapeamento dos núcleos de uma aplicação em NoCs. A aplicação é modelada por um grafo denominado *grafo de caracterização da aplicação*, em inglês, *APplication Characterization Graph* (APCG). Esta abordagem é baseada em *modelos de comunicação com pesos*, em inglês *Communication Weighted Model* (CWM), onde o peso do canal de comunicação

corresponde à quantidade de bits das mensagens transmitidas por este canal. Implementando este modelo em algoritmos de mapeamento, os autores mostram que é possível reduzir o consumo de energia em mais de 60%, quando comparado com soluções de mapeamento *ad hoc*. Os mesmos autores (2004) estendem a abordagem acima para avaliar também o desempenho da comunicação. Eles descrevem que a abordagem adotada consegue melhorar o desempenho da comunicação, considerando como restrição a largura de banda máxima de cada canal na NoC. O objetivo adotado para o mapeamento é distribuição dos núcleos em tiles da NoC de forma a não exceder o limite de largura de banda.

Murali e De Micheli (2004^{A}) implementam uma solução similar à apresentada por Hu e Marculescu (2003) e (2005). A aplicação também é representada por um modelo do tipo CWM, denominado de *grafo de núcleos* (em inglês, *core graph*). O foco do trabalho é a apresentação de um algoritmo de mapeamento de núcleos em NoCs com topologia malha, considerando a largura de banda de comunicação como principal restrição. Tanto o trabalho de Murali e De Micheli (2004^{A}), quanto o de Hu e Marculescu (2003), avaliam a largura de banda utilizando o pior caso de comunicação entre os núcleos, ou seja, consideram que todas as comunicações ocorrem simultaneamente. Esta abordagem é devido à limitação do modelo CWM, que não captura os intervalos de tempo que ocorrem as comunicações.

Murali e De Micheli estendem em (2004^{B}) o trabalho apresentado em (2004^{A}), introduzindo uma ferramenta chamada SUNMAP. Esta ferramenta constrói internamente uma biblioteca de topologias de NoCs e usa uma função multi-objetivo, que inclui média de atraso de comunicação, consumo de área e consumo de energia. O principal objetivo da ferramenta é selecionar automaticamente a melhor topologia de NoC para uma dada aplicação e gerar o mapeamento de núcleos na topologia escolhida. Uma vez que as bibliotecas de topologias estejam caracterizadas, a abordagem adotada permite aumentar a precisão das estimativas no processo de síntese de alto nível. Todavia, para estimativas precisas, as alturas e larguras dos tiles da NoC devem ser fixadas nos valores previamente caracterizados na biblioteca, o que pode implicar maior consumo de área e energia. Além do mais, a ferramenta SUNMAP modela a aplicação através de CWM, cuja abstração do modelo compromete a escolha da topologia da rede. Bertozzi et al. (2005) apresentam a construção de um fluxo de síntese para arquiteturas de rede customizadas, chamado de NetChip. Este fluxo utiliza a ferramenta SUNMAP para o mapeamento de diversas aplicações em várias topologias de NoCs.

Rhee, Jeong e Ha (2004) propõem uma abordagem de mapeamento em NoCs onde os tiles podem conter vários núcleos e a cada núcleo é associada uma interface de rede. Estas interfaces ficam fisicamente distribuídas nos tiles com o objetivo de reduzir o gargalo de comunicação dos núcleos locais ao tile com os demais núcleos da rede. Esta abordagem é chamada pelos autores de *many-to-many Cores Switching Mapping* (mCSM), em contraposição à abordagem *one-to-one Cores Switching Mapping* (oCSM), que considera apenas o mapeamento de um núcleo por tile. Os autores descrevem que o uso de mCSM permite até 81,2% de redução de energia quando comparado com oCSM.

Ye, Benini e De Micheli (2002) apresentam um *framework* para avaliar o consumo de energia em redes intrachip. A parte que mais interessa neste trabalho é a modelagem do consumo de energia dinâmica da rede em alto nível. Para obter o modelo de consumo de energia dinâmica, os autores levam em consideração a lógica de roteamento, as áreas de armazenamento temporário e os fios de interconexão entre roteadores e interno aos roteadores. Os mesmos autores, em (2003) e (2004), descrevem

diferentes esquemas de roteamento de pacotes intrachip. Neste último trabalho, os autores enfocam o problema de contenção em NoCs e associam este problema à redução de desempenho do recurso de comunicação. Eles recomendam soluções que reduzem as áreas de armazenamento temporário necessárias para a comunicação e, em conseqüência, alcançam também a redução no consumo de energia.

Bolotin, Cidon, Ginosar e Kolodny (2004) propõem um processo de personalização de NoCs para atender os requisitos de qualidade de serviço. Este processo modifica uma arquitetura de rede genérica baseada em topologia malha bidimensional e roteamento XY. O objetivo é minimizar o consumo de área e energia da NoC e ao mesmo tempo garantir que os requisitos de qualidade de serviço sejam atingidos. O grande mérito deste trabalho reside na síntese multi-objetivo com requisitos de qualidade de serviço, que permite tratar de aplicações de tempo real. Este tratamento pode ser realizado através de atividades de mapeamento e técnicas de tratamento do roteamento dos pacotes, tal como o uso de canais virtuais.

Hu e Marculescu (2004) introduzem um modelo que captura a comunicação e a computação. A comunicação é representada pela quantidade total de bits enviada de uma tarefa para outra e a computação é representada pelo escalonamento das tarefas com a correspondente dependência de execução e o deadline para executar a tarefa. Este modelo é implementado através de um *grafo de tarefas de comunicação*, em inglês *Communication Task Graph* (CTG), que é o formato interno do algoritmo de particionamento[12]. O CTG permite obter resultados mais precisos que os apresentados em (HU; MARCULESCU, 2003), pois o modelo subjacente leva em consideração além da quantidade total de bits das tarefas, a dependência de execução das tarefas. Outra questão endereçada com este modelo é o tratamento de aplicações de tempo real, donde deriva a necessidade de modelar os deadlines de cada tarefa.

O trabalho apresentado por Hu e Marculescu (2005) mostra uma preocupação em modelar a aplicação pela sua computação, além da comunicação. A exploração eficiente da computação e da comunicação implica gerar novas partições candidatas a gerarem bons mapeamentos. Os trabalhos que mais se aproximam deste são (LEI; KUMAR, 2003), (LIANG; SWAMINATHAN; TESSIER, 2000) e (MIHAL; KEUTZER, 2003).

Lei e Kumar (2003) apresentam um método de mapeamento de tarefas em elementos de processamento através de algoritmos genéticos. Os autores propõem que esta atividade seja realizada em dois passos. O primeiro passo calcula o melhor particionamento de tarefas em núcleos e o segundo passo posiciona estes núcleos sobre uma rede intrachip com topologia malha. Para tanto, implementaram uma ferramenta cuja aplicação é descrita através de um grafo de escalonamento de tarefas, denominado pelos autores de grafo de tarefas, em inglês *task graph*.

Liang, Swaminathan e Tessier (2000) descrevem o sistema aSoC, que é composto por uma infra-estrutura de comunicação intrachip que suporta o mapeamento de *recursos heterogêneos*[13]. aSoC é uma plataforma cuja comunicação é efetuada por

---

[12] Hu e Marculescu (2004) se referem à atividade de associar tarefas a elementos de processamento através do termo mapeamento. Neste trabalho, esta atividade será chamada de *particionamento*, pois a associação de tarefas a elementos de processamento gera blocos diferentes, ou seja, diferentes partições.

[13] Liang, Swaminathan e Tessier (2000) denominam de *recursos heterogêneos* o que neste trabalho é denominado de elemento computacional, tal como FPGAs, processadores ou memórias.

uma NoC. Em cada tile está fisicamente posicionado um recurso heterogêneo. O sistema dispõe de um software que dá suporte ao mapeamento da aplicação na infra-estrutura de comunicação. O usuário parte de uma especificação em C/C++ que é convertida para um formato interno. O formato interno permite realizar o particionamento das tarefas em núcleos, e o mapeamento dos núcleos nos recursos heterogêneos. Através do particionamento e do mapeamento, os autores atuam sobre os problemas de computação e comunicação da aplicação.

Mihal e Keutzer (2003) propõem a modelagem de aplicações concorrentes heterogêneas em plataformas de multi-processadores heterogêneos. A chave para a solução de tal problema é a modelagem da concorrência da aplicação que implica a captura formal da comunicação em modelos computacionais. Para tanto, os autores utilizam o sistema Ptolemy (BHATTACHARYYA, 1997) que provê uma estratégia para implementar modelos computacionais. A abordagem proposta parte de uma aplicação descrita em alto nível, a qual é particionada manualmente em tarefas. Este particionamento associa as tarefas da aplicação a elementos de processamento. Etapas posteriores realizam o mapeamento dos elementos de processamento na arquitetura alvo, que tem como infra-estrutura de comunicação uma NoC.

Wu, Al-Hashimi e Eles (2003) apresentam uma modelagem similar à apresentada em (HU; MARCULESCU, 2004), mas denominam CTG de grafo de tarefas condicional, em inglês *Conditional Task Graph* (CTG*)[14]. CTG* é utilizado para modelar tarefas com o objetivo de reduzir do consumo de energia através de técnicas de variação dinâmica da voltagem. Os autores conseguem obter uma redução de até 51% no consumo de energia quando aplicam esta técnica junto ao mapeamento de tarefas em elementos de processamento.

1.6 Originalidade

Como se depreende da Seção anterior, existem alguns trabalhos relacionados à atividade de mapeamento de núcleos para pontos de acesso de infra-estruturas de comunicação. Porém, muitas lacunas são identificadas e devem ser preenchidas para que se possa dominar esta atividade, principalmente considerando a quantidade de variáveis envolvidas.

As principais contribuições deste trabalho são a *avaliação de modelos* existentes e a *proposta de novos modelos* que representem a aplicação, objetivando verificar a adequação destes modelos para a atividade de mapeamento; e também verificar quais as vantagens e as desvantagens dos mesmos. Entre as vantagens da maioria dos modelos avaliados estão a facilidade da captura do formato interno, a exploração da computação e comunicação da aplicação, o tempo necessário para obter o mapeamento e a complexidade algorítmica associada ao tratamento do problema em vista através do modelo. Alguns modelos de aplicação aqui propostos obtêm mapeamentos de qualidade com redução média de 40% no tempo de comunicação da aplicação e 20% no consumo médio de energia, quando comparados com modelos previamente publicados.

---

[14] Aqui se utiliza * para diferenciar o símbolo CTG* dado ao *Conditional Task Graph* por Wu, Al-Hashimi e Eles (2003) do símbolo CTG dado ao *Communication Task Graph* por Hu e Marculescu (2004).

A avaliação de modelos de aplicação permite que seja proposto um metamodelo baseado em critérios ortogonais comuns a todas as aplicações, tais como comunicação e computação. Cada uma destas características pode ser modelada com diferentes graus de detalhamento. Dependendo das características e do grau de detalhamento, classes de modelos podem ser definidas. Estas classes, por sua vez, podem ser avaliadas com relação à capacidade de capturar requisitos de projeto e à complexidade em tratar estes requisitos. Desta forma, o metamodelo e a conseqüente definição das classes de modelos para a aplicação permitem, além da formalização dos modelos e melhor compreensão do problema associado, a escolha da melhor classe de modelos da aplicação para capturar os requisitos de projeto em vista.

Este trabalho também propõe novos modelos de consumo de energia para redes intrachip, visando capturar com maior precisão certas características da aplicação, tais como o número de transições causado pelo conteúdo das mensagens. Para que o mapeamento consiga obter vantagens com este novo modelo de consumo de energia em redes intrachip, os modelos da aplicação devem contemplar tais informações. Esta abordagem permite aumentar a precisão das estimativas de consumo de energia, tendo como penalização o acréscimo de tempo necessário para capturar esta característica e o tempo de computação necessário para executar o algoritmo de mapeamento sobre os modelos mais detalhados. A validação dos modelos de energia para redes intrachip força a criação de um método e de ferramentas de avaliação. O método e as ferramentas também são originais, sendo contribuições deste trabalho.

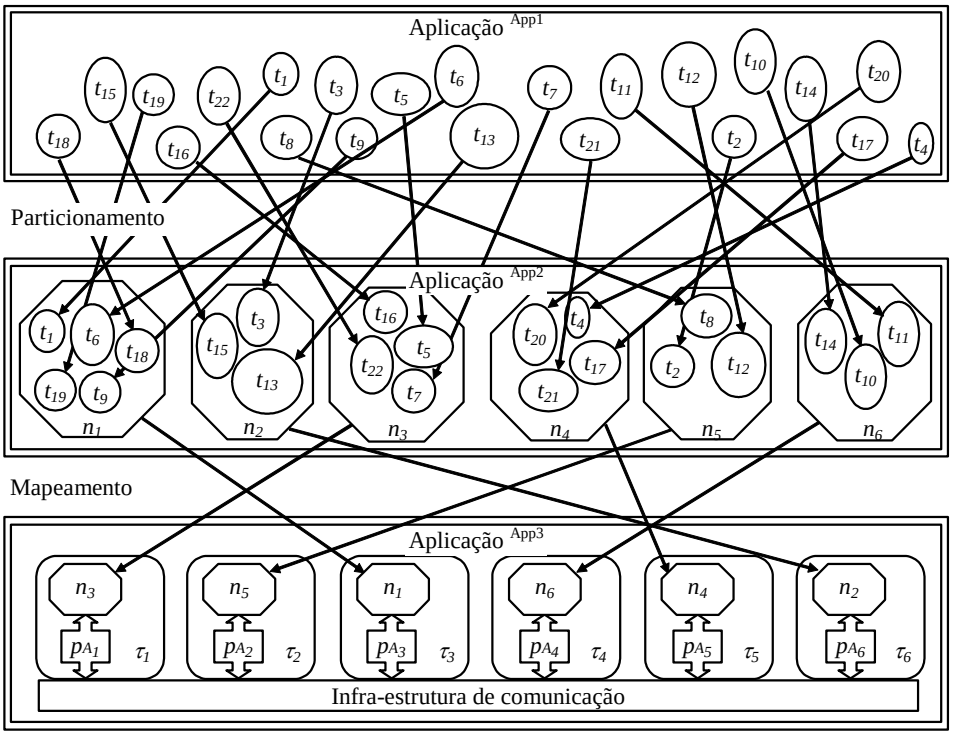
1.7 Estrutura do Trabalho

Este Capítulo introduziu o problema de mapeamento de núcleos de uma aplicação em tiles conectados a roteadores de infra-estruturas de comunicação através de pontos de acesso, além de apresentar um fluxo típico de projeto de sistemas computacionais e descrever as motivações deste trabalho. O Capítulo 2 detalha as atividades de captura do formato interno, além de definir particionamento e mapeamento com o objetivo de contextualizar o leitor no foco deste trabalho. O Capítulo 3 aborda algumas infra-estruturas de comunicação intrachip, cuja atividade de mapeamento interfere no desempenho da aplicação. Os objetivos do Capítulo 3 são enumerar as características principais de cada infra-estrutura de comunicação e descrever a importância do mapeamento de núcleos em cada infra-estrutura. O Capítulo 4 descreve formalmente modelos computacionais e elementos para a construção destes. No Capítulo 5 são descritos os modelos computacionais subjacentes aos formatos internos utilizados como entrada da atividade de mapeamento. Para ilustrar os modelos computacionais utilizados na atividade de mapeamento, o Capítulo 6 descreve uma aplicação de imagens e diferentes modelagens conforme os modelos apresentados no Capítulo 5. O Capítulo 7 descreve o metamodelo QOD, que serve como arcabouço de classificação para os modelos de mapeamento construídos. No Capítulo 8 são apresentados os modelos de consumo de energia e tempo de comunicação para NoCs. A finalidade de tais modelos formais é a possibilidade de qualificar o mapeamento através de funções objetivo. O Capítulo 9 mostra como foram validados e construídos os modelos de consumo de energia para redes intrachip. O Capítulo 10 apresenta alguns resultados obtidos com o *framework* CAFES, desenvolvido para avaliar o mapeamento de aplicações em redes intrachip. O Capítulo 11 compara qualitativamente os modelos computacionais utilizados para a modelagem da aplicação na atividade de mapeamento. O Capítulo 12 mostra as principais conclusões e alguns trabalhos futuros. O Apêndice A

apresenta descrições de várias aplicações em um dos formatos internos apresentado no Capítulo 5. Estas descrições fazem parte do conjunto de aplicações que foram utilizadas para a validação deste trabalho. O Apêndice B finaliza este trabalho apresentando heurísticas para caracterizar aplicações quanto à dependência versus concorrência e quanto à computação versus comunicação.

2 FORMATO INTERNO, PARTICIONAMENTO E MAPEAMENTO

Este Capítulo considera as atividades de particionamento das tarefas da aplicação em núcleos e o mapeamento destes a pontos de acesso de uma infra-estrutura de comunicação, bem como a captura do formato interno que representa a aplicação em cada atividade. Esquematicamente, o processo pode ser representado como na Figura 2.1. Devido à complexidade destas atividades, este trabalho irá abordar apenas a atividade de mapeamento, considerando parcialmente a captura do formato interno e o particionamento com a finalidade de melhorar a compreensão do problema.



Onde: App1, App2 e App3 representam a aplicação descrita em vários níveis.

Figura 2.1: Ilustração do particionamento de tarefas (t) da aplicação em núcleos (n), seguido do mapeamento de núcleos em tiles (τ). Estes núcleos são conectados a uma infra-estrutura de comunicação através de pontos de acesso (p_A).

Na Figura 2.1, a aplicação é representada em três níveis distintos por retângulos com bordas duplas. No primeiro nível a aplicação App1 é composta por um *conjunto de tarefas* $T = \{t_1, t_2, ..., t_n\}$, onde cada tarefa é representada por uma elipse. Sobre App1 é realizado um particionamento que agrupa as tarefas conforme um

objetivo, gerando um *conjunto de núcleos* $N = \{n_1, n_2, \ldots, n_c\}$. Para este trabalho, um bom objetivo de particionamento é o agrupamento de tarefas de forma a minimizar a quantidade de comunicação entre núcleos e atender as necessidades de computação de cada tarefa. Após o particionamento, chega-se ao segundo nível onde a aplicação App2 é composta por um conjunto de núcleos. Um *conjunto de tiles* $\tau = \{\tau_1, \tau_2, ..., \tau_p\}$ serve como meio físico para posicionar os núcleos que se conectam à infra-estrutura de comunicação através de um *conjunto de pontos de acesso* $P_A = \{p_{A1}, p_{A2}, ..., p_{Ap}\}$. Dependendo da complexidade da infra-estrutura de comunicação, os pontos de acesso podem ser fios, chaves ou até circuitos seqüenciais complexos. O mapeamento aplica funções objetivo para avaliar a qualidade da associação dos núcleos aos tiles da arquitetura alvo. Se a infra-estrutura de comunicação é um barramento, a posição dos núcleos é irrelevante. No caso de uma NoC, por outro lado, o mapeamento é tarefa fundamental. A atividade de mapeamento gera o terceiro e último nível da aplicação (App3) descrito na Figura 2.1.

As atividades de particionamento e mapeamento têm como entrada e saída formas de descrição da aplicação. Estas descrições são denominadas *formato interno*, ou *formato intermediário*, pois são usadas apenas pelas ferramentas ou etapas que fazem parte do fluxo de projeto. O formato interno deve ser capturado a partir de descrições fornecidas pelo projetista, tais como SystemC ou Esterel. A Figura 2.2 detalha as atividades de projeto que precedem e sucedem o mapeamento. Na Figura, as elipses representam descrições da aplicação, os retângulos representam transformações sobre as descrições e o retângulo de canto dobrado reúne outras informações necessárias durante o fluxo de projeto.

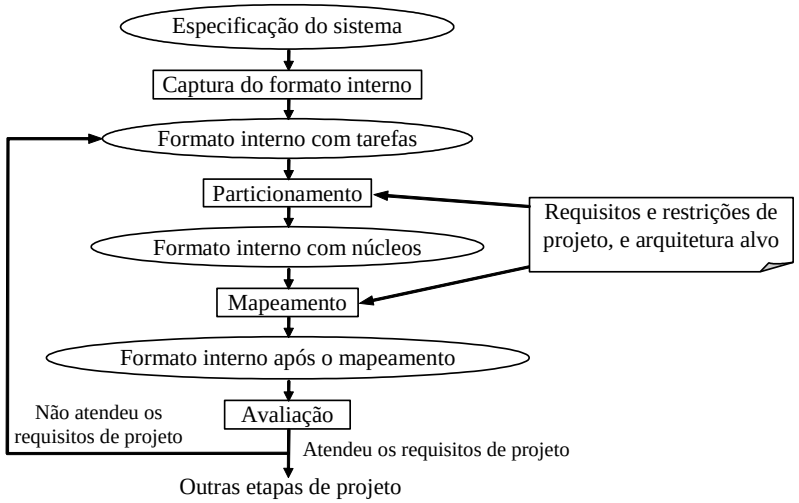


Figura 2.2: Fluxo de projeto parcial, relacionando as atividades de captura do formato interno, particionamento e mapeamento.

Considera-se aqui que a arquitetura alvo, composta pela infra-estrutura de comunicação com os núcleos da aplicação, pode ser definida após a atividade de particionamento. Ou seja, não se trata de projetos baseados em plataforma (KEUTZER, 2000), mas de projetos com graus de liberdade para geração da arquitetura alvo. Caso fossem projetos baseados em plataforma, tanto o particionamento quanto o mapeamento deveriam ter informações sobre os recursos disponíveis na plataforma, para saber a possibilidade de agrupar tarefas e/ou mapear núcleos.

2.1 Captura do Formato Interno

O *formato interno* é uma descrição da aplicação, tal como um CDFG (*Control-Data Flow Graph*) ou um ACFSM (*Abstract Codesign Finite State Machine*) (MARCON, 2000). Neste trabalho, o formato interno é utilizado como entrada das ferramentas de particionamento e mapeamento. Com freqüência, o formato interno é implementado usando representações de grafos dirigidos, dado que grafos são estruturas adequadas para representar paralelismo e dependência de mensagens.

Todo formato interno possui um modelo computacional subjacente. Certos modelos computacionais podem ser muito abstratos, não provendo informação suficiente para guiar uma atividade de projeto, enquanto outros podem ser detalhados demais, tornando complexo o tratamento computacional. As características de cada formato interno são descritas no decorrer deste trabalho para comparar a complexidade dos modelos subjacentes.

A captura do formato interno é muitas vezes efetuada de forma manual devido à complexidade de automatizar o processo. Contudo, dependendo do modelo computacional subjacente ao formato interno, a captura pode ser automatizada. Por exemplo, para modelos mais simples, tais como aqueles que apenas consideram a quantidade de comunicação da aplicação, ou para os modelos que representam a comunicação e a computação pela ordem dos eventos, mas que não levam em consideração a dependência de dados, a automatização da captura do formato interno pode ser obtida por simulação. Este é o caso da obtenção do formato interno CWG (detalhes sobre este modelo são encontrados na Seção 5.1) a partir da simulação de uma aplicação descrita em SystemC. Esta mesma facilidade não ocorre para modelos mais complexos que modelam dependência de mensagens. O motivo é que uma simulação fornece apenas uma das ordens possíveis de execução, que pode mudar de acordo com os estímulos que excitarem a aplicação. Por outro lado, a dependência não tem relação com os estímulos, apenas com a funcionalidade da aplicação. Na verdade, para obter a dependência de mensagens é necessário realizar a análise semântica da descrição, e neste ponto é que reside a complexidade de automatizar este processo. Este é o caso da obtenção do formato interno CDCM (detalhes sobre este modelo são encontrados na Seção 5.4).

O formato interno adequado para uma atividade, tal como o particionamento, pode não ser adequado para outra atividade, tal como o mapeamento. De acordo com a execução das atividades do fluxo de projeto, o formato interno pode ser alterado, ou apenas anotado com informações geradas nas etapas anteriores. Eventualmente, pode ser necessário mudar o modelo subjacente para que o formato interno passe a ter maior significado para as atividades seguintes. Por exemplo, a atividade de particionamento pode inserir informações relativas à partição e retirar informações relativas às tarefas, mantendo o modelo subjacente do formato interno.

2.2 Particionamento

Seja $\boldsymbol{O} = \{o_1, o_2, ..., o_n\}$ o conjunto de todos os objetos de um sistema, $b \subseteq \boldsymbol{O}$ um subconjunto de $\boldsymbol{O}$ denominado *bloco* e $\boldsymbol{P} = \{b_1, b_2, ..., b_m\}$ uma partição que é o conjunto de todos os blocos de $\boldsymbol{O}$, *particionamento* é o processo que gera $\boldsymbol{P}$, tal que $b_1 \cup b_2 \cup ... \cup b_m = \boldsymbol{O}$ e $b_k \cap b_i = \varnothing \ \forall \ b_k, b_i$ com $k \neq i$.

O maior bloco de uma partição ocorre quando todos os objetos que compõem o

sistema fazem parte deste bloco. Neste caso, existe apenas um bloco ($m = 1$) que é o próprio sistema. O menor bloco de uma partição ocorre quando existe apenas um objeto por bloco. Neste caso, o número de blocos é igual ao número de objetos $m = n$. Logo, o número de blocos pode variar de um até o número de objetos que compõem o sistema.

No esquema representado na Figura 2.1, as tarefas são os objetos e após aplicar o particionamento, cada bloco equivale a um núcleo e a partição equivale ao conjunto de núcleos.

Para projetos de sistemas computacionais, o particionamento mais elementar realiza a divisão da aplicação em blocos de hardware e de software. Todavia, o problema de particionamento não se restringe à escolha de componentes que implementarão cada núcleo da partição. Existem vários outros subproblemas associados, que muitas vezes implicam grandes diferenças de custos na implementação do sistema. Por exemplo, independente de escolher uma implementação em hardware ou software para um núcleo mapeado em um tile, a necessidade de computação pode determinar uma arquitetura alvo com diferentes relógios de operação, um para cada tile, implicando diferentes consumos de energia.

2.3 Mapeamento

Seja $X = \{x_1, x_2, \ldots, x_c\}$ o conjunto de objetos origem e $Y = \{y_1, y_2, \ldots, y_p\}$ o conjunto de objetos destino. *Mapeamento* é uma *função injetora completa* (CALAZANS, 1998) $\varphi: X \rightarrow Y$ que associa os objetos do conjunto origem aos objetos do conjunto destino com $| Y | \geq | X |$. Esta associação é chamada de *mapa*.

Para este trabalho os objetos do conjunto origem são os núcleos da aplicação e os objetos do conjunto destino são os tiles da NoC. Assim, o mapeamento é a atividade que associa núcleos da aplicação a tiles, sendo que cada núcleo é conectado à infra-estrutura de comunicação por um canal de comunicação. Os núcleos da aplicação são obtidos de atividades anteriores, tais como o particionamento, que pode ser efetuado manual ou automaticamente, e são elementos computacionais que realizam um conjunto de tarefas, tais como DSPs (*Digital Signal Processors*), processadores ou memórias. Os canais da infra-estrutura de comunicação são os pontos de acesso onde os núcleos podem ser conectados à infra-estrutura de comunicação. Dado que para este trabalho serão utilizadas NoCs como estudo de caso de infra-estrutura de comunicação, o mapeamento se refere à associação de núcleos a tiles de NoCs. O comportamento dos núcleos da aplicação, bem como a interação entre eles, que reflete a computação e a comunicação da aplicação, são descritos no formato interno proveniente de atividades anteriores, tal como o particionamento. Por sua vez, o mapeamento gera como saída um novo formato interno, onde cada núcleo passa a estar associado a um tile.

O problema de mapeamento é, respeitando as restrições impostas pelo projeto, encontrar mapeamentos que atendam os requisitos de projeto. Dado que pode existir mais de um requisito de projeto, o privilégio para atender estes requisitos é dado por uma função objetivo de mapeamento, com pesos para cada requisito.

Este trabalho tem como requisitos a minimização do consumo de energia da infra-estrutura de comunicação e a redução do tempo de execução da aplicação. Embora haja dois requisitos, a função objetivo trata apenas da minimização do consumo de energia. Adotando modelos onde as infra-estruturas de comunicação consomem energia mesmo na ausência de tráfego, requisitos de minimização de tempo de execução são

inseridos indiretamente no requisito de minimização do consumo de energia, simplificando o cálculo da função objetivo de mapeamento.

Ao final de um mapeamento, deve ser testado se o mapa obtido atende os requisitos de projeto. Em caso positivo, o fluxo de projeto segue e o formato interno de saída do mapeamento se torna entrada de novas atividades que estão fora do escopo deste trabalho. Caso o mapa obtido não atenda as restrições de projeto, pode ser refeita a atividade de particionamento para obter uma nova partição que será entrada de um novo mapeamento, tal como descrito na Figura 2.2. Este processo pode ser refeito enquanto não forem atendidos os requisitos de projeto, ou enquanto não for atingido um limite de iterações. A cada novo laço, que envolve mapeamento e particionamento, são explorados novos compromissos entre computação e comunicação. Embora nem sempre seja verdadeiro, grosseiramente pode-se dizer que quanto maior for o agrupamento obtido pelo particionamento, menor será a comunicação entre os núcleos. Em contrapartida, maior será a necessidade de computação de cada núcleo. Por outro lado, quanto menor for o agrupamento, maior será a comunicação e a carga de computação de cada núcleo é reduzida.

3 INFRA-ESTRUTURAS DE COMUNICAÇÃO INTRACHIP

Uma *infra-estrutura de comunicação* é um conjunto de elementos que provê meios para o tráfego de informações entre núcleos a ela conectados. Existem dois critérios de classificação básicos para infra-estruturas de comunicação: (*i*) conectividade e (*ii*) dinamicidade.

Conectividade é a medida do número de componentes ligados a cada *canal*[15] da infra-estrutura de comunicação. Segundo este critério as infra-estruturas de comunicação podem ser classificadas em ponto a ponto, multiponto ou mista. A infra-estrutura de comunicação é do tipo *ponto a ponto* quando fornece apenas canais dedicados para a comunicação de dois sistemas; *multiponto* quando fornece apenas canais para a comunicação de mais de dois sistemas; e *mista* quando implementa os dois tipos anteriores. Redes intrachip são exemplos de infra-estruturas de comunicação ponto a ponto e barramentos são exemplos de infra-estruturas de comunicação multiponto.

Dinamicidade reflete a capacidade de um mesmo canal da infra-estrutura de comunicação servir para troca de informações de diferentes núcleos. Segundo este critério, as redes podem ser classificadas em estáticas ou dinâmicas. Infra-estruturas de comunicação *estáticas* são aquelas cujos canais de comunicação são implementados para comunicação exclusiva entre pares de núcleos. Infra-estruturas de comunicação *dinâmicas* são aquelas cujos canais de comunicação são implementados para que a comunicação entre diversos núcleos seja compartilhada temporalmente.

O objetivo deste Capítulo é apresentar algumas infra-estruturas de comunicação intrachip e suas características, tendo como enfoque infra-estruturas de comunicação cuja atividade de mapeamento influi no desempenho final da aplicação. Aqui, as infra-estruturas de comunicação são classificadas segundo o critério de conectividade.

3.1 Comunicação de Núcleos Baseada em Conexões Multiponto

A comunicação baseada em conexões multiponto permite que mais de um núcleo compartilhe um mesmo canal físico em intervalos de tempo distintos. Embora as conexões multiponto permitam apenas um escritor em um dado intervalo de tempo, pode existir mais de um leitor durante este intervalo. Este é o caso típico que ocorre

[15] *Canal* é qualquer meio pelo qual podem trafegar informações. Para este trabalho considera-se que os canais sejam compostos apenas por fios.

com mensagens *multicast*[16] e *broadcast*[17]. O compartilhamento temporal é um fator limitante do paralelismo da aplicação, sendo que este limite é dado pela serialização da comunicação na infra-estrutura de comunicação. Para muitas aplicações, esta limitação pode reduzir o tempo global de execução. Para outras, sobretudo aquelas voltadas à computação e não à comunicação, o baixo paralelismo da infra-estrutura de comunicação não compromete o tempo de execução da aplicação.

3.1.1 Barramentos

Uma infra-estrutura de comunicação típica que utiliza comunicação multiponto é o barramento, ilustrado na Figura 3.1. O barramento é meio físico onde núcleos são conectados através de *chaves*[18]. Hennessy e Patterson (2003) descrevem que muitos autores classificam barramentos em:

1. *Barramentos para conectar processador e memória*, que normalmente são curtos e de alta velocidade;
2. *Barramentos de entrada e saída*, que geralmente são longos, podem ter muitos dispositivos conectados, têm uma grande variedade de largura de banda de dados e, em geral, são padronizados.

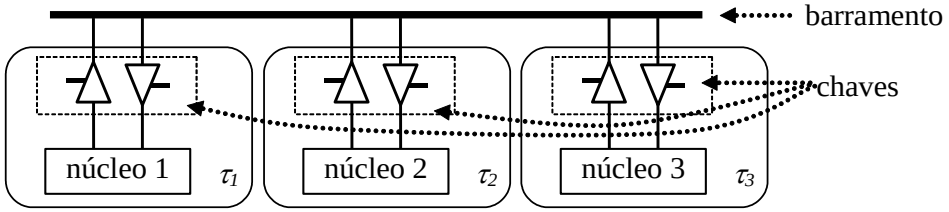


Figura 3.1: Infra-estrutura de comunicação do tipo barramento, permitindo a interligação de três núcleos através de chaves.

Um barramento intrachip normalmente consiste de um conjunto de fios que conectam diferentes núcleos da aplicação, e sobre o qual dados são transmitidos e recebidos. Estes núcleos podem ser classificados como mestres e/ou escravos do barramento. Um núcleo mestre é uma unidade que controla as transferências em um barramento, ou seja, pode solicitar a transmissão ou a recepção de dados através do barramento. Um núcleo escravo é uma unidade que apenas responde às solicitações do mestre. Exemplo é um microprocessador, atuando como mestre do barramento, e uma memória atuando como escrava. As informações são lidas ou escritas da/na memória a partir dos sinais de controle gerados pelo microprocessador.

Barramentos com diversos núcleos conectados têm métodos de arbitragem para controlar a prioridade de acesso ao barramento. Estes métodos são normalmente classificados em centralizado e distribuído. No método de arbitragem *centralizado*, um

---

[16] *Mensagem multicast* ocorre quando há um escritor e mais de um leitor em um meio de comunicação, mas não necessariamente todos os leitores do meio de comunicação devem receber a mensagem.

[17] *Mensagem broadcast* ocorre quando há um escritor para todos os leitores do meio de comunicação.

[18] *Chaves* são elementos que habilitam a conexão entre sistemas, tais como tri-states. Em geral, para este trabalho, as chaves são os pontos de acesso e cada tile do barramento pode conter um núcleo.

dispositivo denominado árbitro é responsável pela atribuição de acesso ao barramento. Neste caso, existem sinais de requisição e de permissão entre o árbitro e os núcleos. O acesso do núcleo ao barramento depende do barramento estar livre e da prioridade deste núcleo frente aos demais. No método de arbitragem *distribuído*, não há um árbitro, a prioridade de acesso ao barramento é feita pelos próprios núcleos. Uma das formas utilizadas para descentralizar a arbitragem é delegar o monitoramento das linhas de requisição aos próprios núcleos do barramento. Desta maneira, cada núcleo sabe sua prioridade na ordem das requisições e se podem ou não utilizar o barramento.

As duas principais vantagens de implementar infra-estruturas de comunicação com barramentos são: o *baixo custo* e a *extensibilidade*. O custo é baixo, pois apenas um conjunto de fios é compartilhado por vários dispositivos, enquanto a extensibilidade é dada pela facilidade em acrescentar novos dispositivos ao barramento. Em contrapartida, podem-se enumerar algumas desvantagens:

1. *Redução do paralelismo da comunicação entre os dispositivos conectados ao barramento*: O paralelismo inexiste em infra-estruturas do tipo barramento monolítico[19], pois não são permitidas comunicações simultâneas, dado que todos os núcleos compartilham um único canal de comunicação. Outras arquiteturas de barramentos, tal como o barramento segmentado, podem aumentar o número de transações simultâneas, porém, ainda assim, só permitem uma transação por segmento;
2. *Baixa escalabilidade*: A escalabilidade é limitada a poucas dezenas de núcleos (BENINI; DE MICHELI, 2002) (GUERRIER; GREINER, 2000);
3. *Alto consumo de energia*: O consumo de energia é elevado devido a fatores físicos, como o comprimento do barramento e o número de dispositivos a ele conectados. Estes fatores implicam aumento da carga capacitiva do barramento e, conseqüentemente, aumento no consumo de energia para efetuar a transição de um sinal;
4. *Limitação na velocidade de operação com o aumento do número de núcleos conectados*: Pelos mesmos motivos enumerados no item anterior, a velocidade de operação do barramento é reduzida com o aumento do número de núcleos conectados.

3.1.2 Barramento Segmentado

Outra infra-estrutura de comunicação multiponto é o barramento segmentado (em inglês *split bus*). Esta infra-estrutura é implementada com a divisão de um barramento monolítico em um barramento com múltiplos segmentos conectados por *portas de roteamento*[20] (p_R). Assim, cada segmento conecta um número menor de

[19] *Barramento monolítico* é aquele onde todos os dispositivos estão conectados a um único barramento, em contraposição ao *barramento segmentado*, onde os dispositivos estão conectados a segmentos, e estes, por sua vez, são interconectados por chaves.

[20] *Porta de roteamento* é o recurso de comunicação utilizado para conectar dois ou mais segmentos. A complexidade da porta de roteamento depende do número de segmentos conectados e dos protocolos de comunicação. Para uma topologia de barramentos com muitos segmentos, a porta de roteamento pode ser implementada por um circuito roteador, que determina o caminho da mensagem por um endereço definido pelo protocolo de comunicação. Para uma topologia mais simples, a comunicação entre segmentos pode ser implementada com tri-states.

núcleos e o comprimento de cada segmento pode ser menor que o comprimento do barramento original, conferindo aos segmentos melhores características elétricas, tal como a redução da impedância. Em conseqüência, o consumo de energia pode ser reduzido e a freqüência de operação pode ser aumentada. Além do mais, o paralelismo é aumentado, uma vez que as comunicações dentro de cada segmento independem das comunicações internas nos demais.

A Figura 3.2 mostra dois barramentos, um monolítico e outro segmentado. Quando a porta de roteamento que interliga os segmentos é habilitada, a funcionalidade do barramento segmentado se assemelha a do barramento monolítico (HSIEH; PEDRAM, 2002).

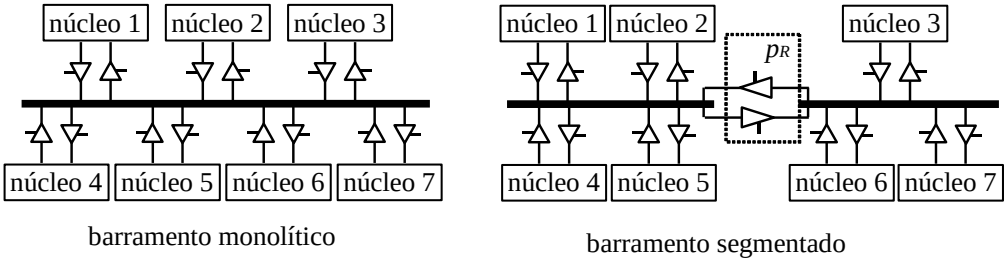


Figura 3.2: Comparação de um sistema implementado com barramento monolítico versus barramento segmentado.

Exemplos de arquiteturas de barramento segmentado encontrados na literatura são: AMBA da ARM (ARM CORPORATION, 2003) e CoreConnect da IBM (IBM CORPORATION, 2004). Geralmente, estas arquiteturas de barramentos estão vinculadas à arquitetura de um processador, tal como o AMBA vinculado ao processador ARM e o CoreConnect vinculado ao processador PowerPC.

A segmentação de um barramento oferece diversas vantagens arquiteturais (HSIEH; PEDRAM, 2002), tais como: aumento do número de transferências em paralelo e redução no consumo de energia durante as transições do barramento. A transferência de dados pode proceder em paralelo em diferentes segmentos devido ao isolamento fornecido pelas portas de roteamento (RAGHUNATHAN; SRIVASTAVA; GUPTA, 2003). Todavia, a segmentação de barramentos oferece alguns problemas de ordem física e lógica que afetam sua escalabilidade. Entre os problemas de ordem física estão as diferenças entre as freqüências de operação e a largura de banda de cada segmento. Entre os problemas de ordem lógica está a adaptação de diferentes protocolos de comunicação.

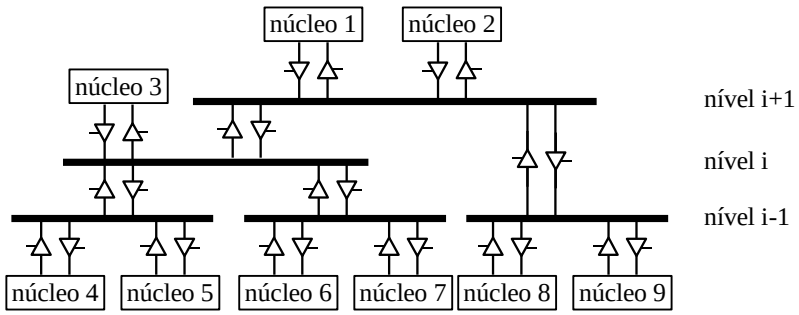


Figura 3.3: Exemplo de barramento segmentado do tipo hierárquico. Neste exemplo são ilustrados três níveis de hierarquia.

Um caso especial de barramento segmentado é o barramento hierárquico. Este é construído de forma a ter uma estrutura de segmentos hierárquica, onde cada nível da

hierarquia reflete um nível lógico de comunicação. O que determina a construção da hierarquia é a necessidade de comunicação da aplicação que irá executar sobre esta infra-estrutura. Em essência, cada segmento da hierarquia se comunica com um conjunto de núcleos e com os segmentos imediatamente acima ou abaixo da hierarquia. A Figura 3.3 ilustra um barramento com três níveis de hierarquia.

Vários são os objetivos da atividade de mapeamento em um barramento segmentado, citam-se aqui três de grande importância:

1. *Definir o número ótimo de segmentos*: Quanto maior o número de segmentos maior será o paralelismo das comunicações, porém maior será a necessidade de comunicar núcleos mapeados em segmentos distintos. Quanto menor for o número de segmentos menor será o paralelismo, mas aumenta o número de comunicações dentro de um único segmento. Nos casos extremos, tem-se de um lado uma estrutura com um segmento por núcleo, e do outro se tem a estrutura de barramento monolítico;
2. *Definir a topologia de conexão entre os segmentos*: De acordo com a aplicação, a topologia adotada pode aumentar o paralelismo das comunicações e reduzir o consumo de energia. Casos típicos são topologias em cadeia, em anel e hierárquica;
3. *Escolher quais núcleos serão mapeados em quais segmentos*: O mapeamento de núcleos afeta o desempenho global da aplicação, seja devido ao consumo de energia, seja pelo tempo de comunicação.

Para este trabalho, não são de grande interesse as infra-estruturas de comunicação tipo barramento monolítico, pois no nível lógico o posicionamento dos núcleos não interfere nos requisitos de projeto não sendo necessário realizar o mapeamento. O mesmo não pode ser dito para as arquiteturas do tipo barramento segmentado. Para estas, o mapeamento tem a granularidade do segmento, e objetivo é mapear núcleos em segmentos de forma a atender os requisitos de projeto.

3.2 Comunicação de Núcleos Baseada em Conexões Ponto a Ponto

Na comunicação baseada em conexões ponto a ponto, ilustrada na Figura 3.4, os núcleos são interligados diretamente. Esta infra-estrutura de comunicação oferece alto grau de paralelismo, pois entre cada par de núcleos a comunicação é exclusiva.

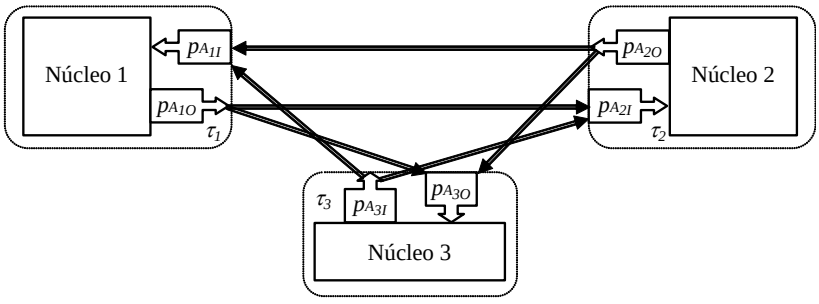


Figura 3.4: Infra-estrutura de comunicação baseada em conexões ponto a ponto.

A infra-estrutura de comunicação dedicada ponto a ponto é eficaz para a intercomunicação entre um pequeno número de núcleos. Pois, o número de conexões

dedicadas aumenta *proporcionalmente com o quadrado*[21] do número de núcleos. Este fator, e a sua irregularidade não conferem escalabilidade para esta infra-estrutura (ZEFERINO; SUSIN, 2003a). A escalabilidade é um fator limitante, à medida que *cresce o número de núcleos projetados dentro de um SoC*[22].

3.2.1 Redes Intrachip

Neste trabalho, define-se rede intrachip como uma infra-estrutura de comunicação composta por roteadores (em inglês, *routers*) interconectados por canais de comunicação (em inglês, *links*). A forma como os roteadores estão conectados entre si, e como os núcleos estão conectados aos roteadores, define a *topologia* da rede. Um *roteador* é um dispositivo que transfere informações disponíveis nos seus canais de entrada para os seus canais de saída, através de portas de comunicação (NI; MCKINLEY, 1993). O intervalo de tempo entre a entrada e a saída de uma informação do roteador é denominado de atraso de roteamento ou *latência*[23]. A estrutura de um roteador consiste de um sistema de chaveamento entre os canais de entrada e saída, um módulo de controle de chaveamento, e elementos de armazenamento temporário para os dados dos canais de entrada e/ou de saída. A estrutura genérica de um roteador é ilustrada na Figura 3.5.

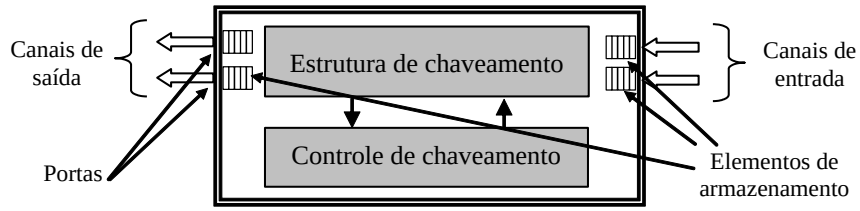


Figura 3.5: Estrutura genérica de um roteador.

O roteador é o elemento principal da rede intrachip, de forma que este deve ser projetado para não impactar a área final do SoC e ter consumo de energia e tempo de chaveamento que atendam as restrições de projeto. Um dos elementos que mais influencia no consumo de área e de energia é o armazenamento temporário (*buffer*). Assim, a estratégia de armazenamento de dados é um fator preponderante de projeto (GUERRIER; GREINER, 2000) (KUMAR, 2003).

Uma rede intrachip pode ser dividida em duas partes: os serviços e o sistema de comunicação. Rijpkema et al. (2003) descrevem alguns serviços que devem ser providos por uma rede intrachip: (*i*) garantia de integridade dos dados que trafegam pela rede; (*ii*) garantia de que todos os dados transmitidos cheguem ao seu destino; (*iii*) garantia da recepção dos dados na mesma ordem que forem enviados; (*iv*) garantia de *throughput*[24] mínimo; e (*v*) garantia de latência mínima.

---

[21] As comunicações, quando exclusivas, implicam que o número de conexões ponto a ponto entre núcleos seja C_n^2, onde *n* é o número de núcleos.

[22] O ITRS projeta para os próximos anos a existência de SoCs com dezenas a centenas de núcleos (ITRS, 2002).

[23] *Latência* é o tempo necessário para transferir uma quantidade de dados de um núcleo fonte para um núcleo destino.

[24] *Throughput* é a quantidade de dados transferida por unidade de tempo.

Uma infra-estrutura de comunicação deve permitir a transferência de informações de um núcleo origem para um núcleo destino. A transferência de informações entre núcleos se dá através da troca de mensagens efetivada através do envio de pacotes (BENINI; DE MICHELI, 2002) (DALLY; TOWLES, 2004) (DUATO, 1997) (GUERRIER; GREINER, 2000). Uma *mensagem* consiste de um conjunto de informações a ser transmitido do núcleo origem para o núcleo destino. Um *pacote* é a unidade de transmissão de informação empregada pelo meio de comunicação para transmitir mensagens. Pacotes podem conter frações de uma mensagem, uma mensagem inteira ou mesmo múltiplas mensagens. Pacotes são normalmente constituídos de um cabeçalho (em inglês, *header*), corpo (em inglês, *payload*) e finalizador (em inglês, *trailer*). Em geral, o cabeçalho contém dados úteis para o roteamento, o corpo da mensagem contém a(s) mensagem(ens) para o núcleo destino e o finalizador contém dados que garantem a coerência do pacote enviado.

Para garantir a transferência de mensagens entre núcleos, torna-se necessário impedir que venham a ocorrer fenômenos como *deadlock*, *livelock* e *starvation* (DUATO, 1997), definidos a seguir:

1. *Deadlock* é uma interdependência cíclica de comunicações que bloqueiam indefinidamente certos caminhos da infra-estrutura de comunicação;
2. *Livelock* é a situação em que a informação transmitida jamais atinge o seu destino, devido a esta percorrer continuamente caminhos cíclicos que não incluem o núcleo destino da informação. O *livelock* é um fenômeno que está normalmente associado aos algoritmos de roteamento na rede;
3. *Starvation* é a postergação indefinida de acesso a recursos de comunicação. Um exemplo de *starvation* é a solicitação de um canal de saída por um pacote armazenado em um buffer e este canal permanece bloqueado porque o canal de saída é sempre alocado para outro solicitante de mais alta prioridade. Este problema está diretamente relacionado a algoritmos de arbitragem dos roteadores.

Topologia de uma rede intrachip consiste na descrição completa do arranjo de interconexões entre elementos de roteamento desta. As topologias podem ser *regulares*, quando é possível definir um padrão deste arranjo com base na estrutura dos elementos de roteamento. Caso este padrão não possa ser identificado, a topologia é denominada *irregular*.

As topologias costumam ser descritas como grafos onde os elementos de roteamento são representados por vértices e as interconexões são representadas por arestas (NI; MCKINLEY, 1993). A Figura 3.6 ilustra o conceito, mostrando alguns tipos de topologias regulares.

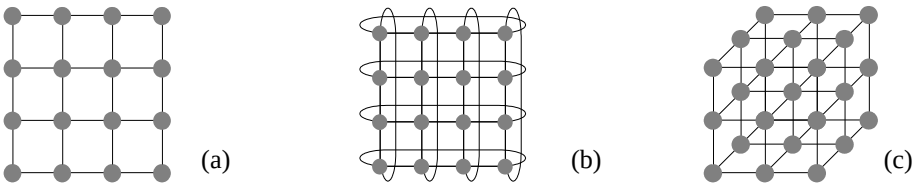


Figura 3.6: Exemplos de topologias regulares de rede intrachip: (a) malha 2D (em inglês, *mesh*), (b) toro 2D (em inglês *torus*) e (c) hipercubo 3D.

Redes *diretas* são aquelas onde cada roteador conecta-se a pelo menos um núcleo. Em contraposição, redes *indiretas* são aquelas onde alguns roteadores não têm

comunicação com qualquer núcleo (DALLY; TOWLES, 2004) (DUATO; YALAMANCHILI; NI, 2003). Exemplo da segunda é a rede com topologia árvore-gorda quaternária (ANDRIAHANTENAINA, 2003), que está ilustrada na Figura 3.7.

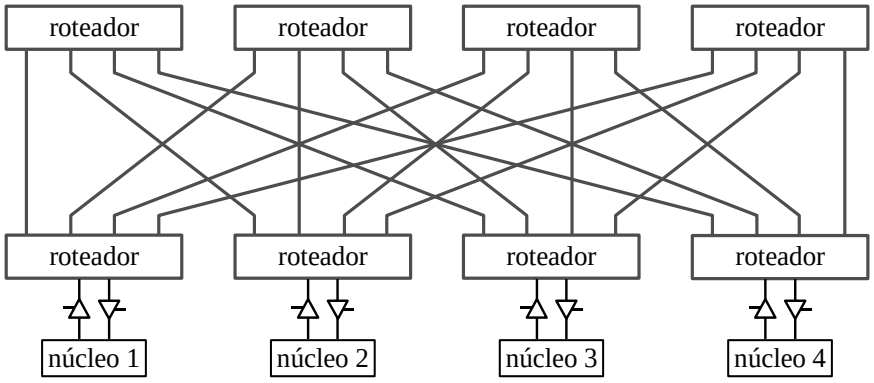


Figura 3.7: Exemplo de uma rede indireta com topologia árvore gorda quaternária.

Um detalhamento do tile de uma rede com topologia malha está ilustrado na Figura 3.8. Tanto o roteador quanto seu núcleo local estão fisicamente posicionados em tiles, que são os nodos da rede. O roteador é o ponto de acesso do núcleo à rede. O que difere esta infra-estrutura de comunicação das infra-estruturas com conexão dedicada ponto a ponto é a capacidade do ponto de acesso direcionar as informações não apenas para o núcleo local ao tile, mas também para outros pontos de acesso, sem que os núcleos locais tomem conhecimento da informação transmitida.

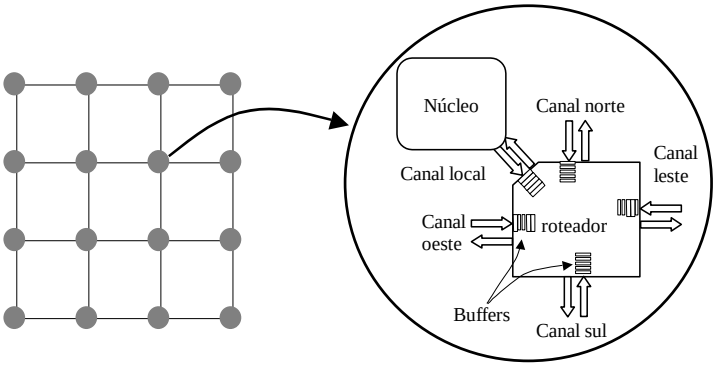


Figura 3.8: Exemplo de detalhamento de um tile em um SoC que usa como infra-estrutura de comunicação uma rede com topologia malha.

O *algoritmo de roteamento* define o caminho a ser utilizado por um pacote a partir do seu núcleo de origem até o seu núcleo destino. Os algoritmos de roteamento podem ser classificados de acordo com seu objetivo. Na literatura existem algumas propostas de taxonomia para algoritmos de roteamento (DALLY; TOWLES, 2004) (DUATO; YALAMANCHILI; NI, 2003) (NI; MCKINLEY, 1993) segundo os seguintes critérios:

1. *Quanto ao momento de realização do roteamento*: *Dinâmico*, quando realizado em tempo de execução ou *estático*, quando realizado em tempo de projeto;
2. *Quanto ao número de destinos das mensagens*: *Unicast*, quando os pacotes têm apenas um destino ou *multicast*, quando os pacotes têm mais de um destino;

3. *Quanto ao local onde a decisão de roteamento é tomada*: *Centralizado*, quando um único elemento define o caminho de todos os pacotes; *origem*, quando o caminho do pacote é definido na origem; ou *distribuído*, quando o caminho do pacote é definido durante sua transmissão até o destino;
4. *Quanto à forma de implementação*: *Baseado em tabelas*, quando o caminho é definido de acordo com uma tabela armazenada em memória ou *baseado em máquinas de estado*, quando o caminho é definido a partir de um algoritmo;
5. *Quanto ao processo de seleção do caminho*: *Determinístico*, quando dados os núcleos origem e destino, o caminho entre estes é sempre o mesmo ou *adaptativo*, quando o caminho entre a origem e o destino é determinado por fatores da rede, tais como condições de tráfego. Um exemplo de roteamento determinístico é o XY para topologias malha. Neste roteamento, o pacote percorre um caminho que passa por todas as conexões e roteadores em X até chegar ao endereço em X do roteador destino, então percorre todos os roteadores em Y até chegar ao roteador destino. Os algoritmos de roteamento adaptativos podem ser classificados de acordo com a abordagem do algoritmo:
    i. *Quanto à progressividade*: *progressivo*, se o cabeçalho sempre avança reservando canais ou *regressivo*, se o cabeçalho puder retornar liberando os canais previamente reservados;
    ii. *Quanto à minimalidade*: *mínimo*, quando o próximo roteador em direção ao destino estiver mais perto que o anterior e *não mínimo*, quando o algoritmo de roteamento permitir o afastamento de um pacote do seu destino;
    iii. *Quanto ao número de caminhos*: *completo*, quando todos os caminhos são possíveis de serem usados, ou *parcial*, quando apenas um conjunto limitado de caminhos viáveis é utilizado.

A escolha da forma na qual pacotes são transferidos da entrada de um roteador para um de seus canais de saída é o que determina o comportamento das chaves internas de cada roteador. Dois métodos de transferência de pacotes são utilizados: chaveamento de circuito e de pacotes.

1. No *chaveamento de circuitos* (em inglês, *circuit switching*), inicialmente é reservado um caminho do núcleo origem até o núcleo destino, para posteriormente serem enviadas as mensagens;
2. No *chaveamento de pacotes* (em inglês, *packet switching*), os pacotes são transmitidos sem qualquer necessidade de procedimentos que estabeleçam caminhos prévios. O emprego de chaveamento de pacotes implica o uso de três *modos de chaveamento* definidos a seguir:
    i. No modo *store-and-forward*, um pacote tem que ser completamente armazenado em um roteador antes de ser enviado para o próximo roteador;
    ii. No modo *virtual cut-through*, um roteador pode enviar um pacote a partir do momento que o próximo roteador garanta que pode receber todo o pacote;

iii. No modo *wormhole* os pacotes são quebrados e transmitidos entre os roteadores em pequenas unidades, denominadas *flits*[25]. Uma desvantagem associada a este modo é que apenas o flit de cabeçalho contém informações sobre o endereçamento destino. Logo, os demais flits que compõem o pacote devem seguir o mesmo caminho reservado para o cabeçalho. Se um cabeçalho não puder avançar na rede em função de um congestionamento, todos os flits restantes são bloqueados ao longo do caminho, até que este seja liberado.

3.3 Comparação entre Infra-estruturas de Comunicação Intrachip

Esta Seção, baseada em (OST, 2004), compara qualitativamente infra-estruturas de comunicação com relação a certas características, tais como: paralelismo, escalabilidade e reusabilidade. Estas características estão resumidas na Tabela 3.1.

1. *Paralelismo*: A infra-estrutura de comunicação que oferece maior paralelismo é a conexão dedicada ponto a ponto, pois pode ser projetada para que todas as comunicações ocorram de forma independente. Situações intermediárias ocorrem com infra-estruturas do tipo barramento segmentado e redes intrachip. Para estas infra-estruturas a topologia junto com o algoritmo de roteamento define o grau de paralelismo das comunicações. Por fim, o barramento monolítico oferece o menor grau de paralelismo dado que pode ocorrer apenas uma única comunicação por vez;
2. *Escalabilidade*: As infra-estruturas de comunicação que oferecem maior escalabilidade são as redes intrachip diretas, devido à regularidade. Independente das dimensões da rede, certas características elétricas não mudam, tais como a capacitância das conexões e a freqüência de operação dos roteadores. Uma vez projetada uma determinada topologia com um determinado tamanho, se torna fácil estimar a operação para tamanhos diferentes. A escalabilidade dos barramentos é limitada a poucas dezenas de núcleos (ZEFERINO, 2003b), pois a conexão com um novo núcleo implica variações na impedância do barramento. Para a inclusão de um novo núcleo em conexões dedicadas ponto a ponto, é necessário re-projeto, que às vezes pode implicar pouco esforço, se este estiver localizado em um subsistema isolado, ou muito esforço, caso o sistema não seja modular. Barramento segmentado podem oferecer uma maior escalabilidade que barramento monolítico, caso a segmentação seja projetada de forma a atender muitas classes de aplicações;
3. *Testabilidade*: A capacidade de testar a infra-estrutura de comunicação está diretamente associada ao número de pontos de acesso para excitar o sistema e para receber a resposta destas excitações. Neste aspecto, o teste em barramento monolítico é o mais simples, pois exige apenas dois

[25] *Phit* é uma unidade física que indica o número de fios físicos para transmissão de dados entre dois roteadores, ou entre núcleo e roteador. *Flit* é uma unidade lógica do modo de chaveamento *wormhole* que é múltipla do *phit*. Convém salientar, que neste trabalho, as redes utilizadas têm flits de tamanho exatamente igual a um phit.

núcleos de teste para verificar a sua funcionalidade. A menor testabilidade se dá com conexões dedicadas ponto a ponto, pois para testar todas as conexões o número de testes é proporcional ao quadrado do número de núcleos, supondo conectividade total. Situações intermediárias ocorrem com redes intrachip e barramento segmentado;

4. *Tolerância a falhas de comunicação*: A tolerância a falhas é encontrada tipicamente em infra-estruturas de comunicação que apresentam paralelismo. Este é o caso de redes intrachip. Nestas, caso uma das conexões falhe, dependendo do algoritmo de roteamento, outras conexões podem ser adotadas para manter a comunicação entre os núcleos. A segmentação de barramentos eventualmente pode oferecer certo grau de tolerância a falhas, mas apenas em situações específicas. Para barramento monolítico e conexões dedicadas ponto a ponto não existe tolerância a falhas, a menos que sejam inseridos mecanismos para tanto;
5. *Reusabilidade*: A única infra-estrutura que não oferece reusabilidade é a conexão dedicada ponto a ponto. Todavia, para obter alto desempenho, infra-estruturas do tipo barramento segmentado podem ser desenvolvidas para uma aplicação específica, ou uma classe de aplicações, reduzindo também a reusabilidade. O barramento tem alta reusabilidade para um pequeno conjunto de núcleos e as redes têm alta reusabilidade, independente do número de núcleos;
6. *Consumo de energia*: As conexões dedicadas ponto a ponto podem ser projetadas para consumir o mínimo de energia, dado que os componentes podem ser personalizados de forma a atender este requisito. As estruturas regulares sempre implicam penalização no quesito consumo de energia devido ao fato de serem genéricas. Para um pequeno número de núcleos, o barramento consome pouca energia. Por outro lado, quando o número de núcleos cresce, a segmentação do barramento pode ser uma alternativa que atenda o quesito consumo de energia, uma vez que esta segmentação pode ser realizada de forma personalizada para a aplicação. Um grande problema de redes intrachip é o consumo de energia dos roteadores que, dependendo da implementação e das características de comunicação da aplicação, pode ser bem elevado;
7. *Largura de banda da comunicação*: As conexões dedicadas ponto a ponto podem ser projetadas para obter o máximo de desempenho em termos de latência e tempo de comunicação global. As infra-estruturas regulares de comunicação sempre implicam penalização neste requisito, devido ao fato de serem genéricas. Para um pequeno número de núcleos, barramentos podem obter altas taxas de comunicação, principalmente para aplicações que não demandam muito paralelismo. Por outro lado, quando o número de núcleos cresce, a segmentação de barramentos pode ser uma alternativa para aumentar o desempenho global, uma vez que esta segmentação pode ser realizada de forma personalizada para a aplicação. Embora as redes intrachip geralmente impliquem aumento na latência dos pacotes, quando comparadas com barramento monolítico, devido ao alto grau de paralelismo elas obtêm altas taxas de comunicação. Um grande problema de redes intrachip é que a velocidade global do sistema é fortemente afetada pelas características do tráfego, topologia, roteamento e mapeamento dos núcleos na rede;

8. *Necessidade de técnicas de mapeamento e particionamento*: Os barramentos monolíticos são infra-estruturas de comunicação que não exigem a atividade de mapeamento de núcleos com a finalidade de aumentar o desempenho da comunicação, dado que as posições dos núcleos não afetam as características desta. As conexões dedicadas ponto a ponto, por sua vez, têm estrutura irregular, de forma que o particionamento de tarefas e o mapeamento afetam o desempenho global do sistema. Contudo, devido à enorme variedade de topologias, a complexidade destas atividades é extremamente elevada. Infra-estruturas como redes intrachip e barramentos segmentados têm certa regularidade, que limita os graus de decisão quanto às atividades de particionamento e mapeamento. Todavia, estas atividades são imprescindíveis para que as aplicações, implementadas sobre estas infra-estruturas de comunicação, consigam obter um bom desempenho. Grosseiramente, pode se dizer que o papel do particionamento para estas infra-estruturas é agrupar em núcleos as tarefas que mais se comunicam. O mapeamento, por sua vez, objetiva aproximar na infra-estrutura de comunicação os núcleos que mais se comunicam. Para barramentos segmentados, os núcleos que mais se comunicam são direcionados para o mesmo segmento, enquanto que para redes intrachip os núcleos que mais se comunicam são colocados em tiles próximos.

Tabela 3.1: Resumo da comparação qualitativa entre infra-estruturas de comunicação intrachip.

Características		Conexões ponto a ponto		Conexões multiponto	
		NoC	**Conexão dedicada**	**Barramento monolítico**	**Barramento segmentado**
Relativas à infra-estrutura de comunicação	**Paralelismo**	3	4	0	1, 2, 3 #
	Escalabilidade	4	1	2 *	2, 3 #
	Testabilidade	2, 3 #	1	4	2, 3 #
	Tolerância a falhas	4	0	0	1, 2, 3 #
	Reusabilidade	4	1	3, 4 *	2, 3 #
	Consumo de energia	4	1	2 *	2, 3, 4 #
	Taxa de comunicação	3	4	1, 2, 3, 4 *	2, 3, 4 #
Relativas aos algoritmos	**Complexidade do mapeamento e/ou particionamento**	3	4	0	2, 3

Legenda: 0 – Não existe;
1 a 4 – Mínimo a máximo.

Notas: * – Limitado a um número pequeno de núcleos;
– Fortemente dependente da topologia.

4 MODELOS COMPUTACIONAIS

Existem várias maneiras de descrever o comportamento de sistemas. No caso do comportamento ser expresso através de linguagem natural, o sistema estará descrito de maneira informal e conseqüentemente poderá conter ambigüidades, o que dificultará, ou impossibilitará o tratamento computacional da descrição. Este fato mostra ser essencial inserir modelos e linguagens formais que os capturem, para viabilizar o emprego de métodos automatizados de projeto de sistemas computacionais. Calazans (1998) expõe que modelos são abstrações de entidades do mundo real, destinadas a permitir a manipulação sistemática de parte dos conceitos destas entidades. Menciona ainda que um bom modelo deve ser capaz de representar todos e apenas os conceitos relevantes a serem manipulados.

Essencialmente, modelos computacionais servem para formalizar uma classe de sistemas e permitir a manipulação computacional de elementos desta classe. Sgroi, Lavagno e Sangiovanni-Vincentelli (2000) expõem que através de modelos computacionais é possível:

1. Capturar sem ambigüidade as funcionalidades do sistema bem como características não funcionais, tais como tempo de execução, consumo de energia e qualidade de serviço;
2. Verificar a validade das especificações funcionais com relação às propriedades desejáveis do sistema (e.g. satisfazer propriedades lógicas temporais ou comportamentos esperados);
3. Sintetizar parte da especificação sobre recursos arquiteturais e de comunicação escolhidos, ou refinar a especificação manualmente e verificar a validade deste refinamento;
4. Usar diferentes ferramentas para manipular diferentes aspectos do modelo.

Uma especificação de um sistema deve ser realizada mediante o emprego de primitivas do modelo computacional escolhido. Caso as primitivas do modelo não sejam capazes de capturar as características relevantes do sistema, o modelo é inadequado para a especificação em vista.

Em geral, existem linguagens que são direcionadas para a descrição de sistemas mediante emprego de um determinado modelo. Contudo, certas linguagens têm a capacidade de representar mais de um modelo computacional, enquanto outras são por demais abstratas para especificar o sistema desejado e outras até inadequadas por não permitirem a representação de primitivas de um modelo computacional. Como exemplo, um projetista poderia considerar uma linguagem de programação como C adequada para descrever algoritmos que podem ser implementados em *hardware*, mas poderia concluir ser esta inadequada para descrever características temporais do mesmo sistema.

4.1 Elementos de Um Modelo Computacional

Uma linguagem é um conjunto de símbolos, regras para combiná-los, formando construções válidas (a sintaxe) e regras para interpretar o significado de construções válidas (a semântica). A semântica pode ser *operacional*, quando dá significado à linguagem em termos de ações tomadas por uma máquina abstrata, ou *denotacional*, que dá significado à linguagem em termos de relações. O modelo computacional que dá suporte a uma linguagem representa o comportamento da máquina abstrata na semântica operacional. Nesta Seção são discutidos alguns conceitos sobre modelos computacionais, baseado em Edwards et al. (1997), Lee e Sangiovanni-Vincentelli (1996), (MARCON, 2000) e (MARCON et al., 2002b).

4.1.1 O Modelo de Sinal com Rótulo

O *modelo de sinal com rótulo* é um formalismo utilizado para descrever modelos computacionais para sistemas embarcados, proposto por Lee e Sangiovanni-Vincentelli (1996), onde a entidade fundamental do modelo é o evento. Estes autores representam formalmente um evento por um par (valor, rótulo). Para eles, dado um conjunto de valores V e um conjunto de rótulos T, um *evento* e é definido como um membro do produto cartesiano $T \times V$. Neste caso, um evento tem sempre um valor v_j associado a um rótulo t_k. Assim, um evento pode ser denotado pelo par $e = (v_j, t_k)$. Assim, uma transição de um valor em um dado instante de tempo é marcada por um evento. Para sistemas Booleanos, por exemplo, uma transição do valor 1 para 0 em um instante de tempo é um evento.

Rótulo é uma marca que permite ordenar elementos computacionais. Para os modelos apresentados neste trabalho, a ordenação está associada ao tempo. Ou seja, usa-se aqui ordenamento temporal. Desta forma, rótulo pode ser compreendido como um valor que representa um instante de tempo, ou outra grandeza passível de ordenamento.

Assumindo *sinal* como uma entidade sobre a qual ocorre uma seqüência de eventos, seria possível considerar um fio ou um barramento como um sinal, já que ambos são entidades que assumem diversos valores no tempo. Assim, um sinal s é definido de forma puramente comportamental como um conjunto de eventos $s = \{e_1, e_2, e_3, ..., e_n\}$ (LEE; SANGIOVANNI-VINCENTELLI, 1996). Aqui, cabe salientar que o conjunto T de rótulos é totalmente ordenado, de forma que seus valores podem ser usados para definir a ordem dos eventos.

Um *processo* P com n sinais é um subconjunto do conjunto de todas as n-tuplas de sinais S^n para algum n. Considerando que o número de sinais associado ao processo é $n = i + o$, onde i e o são o número de sinais de entrada e saída, respectivamente, é possível usar (S^i, S^o) como uma forma de separar as partes de entrada e de saída de S^n. Assim, um processo é um subconjunto do produto cartesiano $S^i \times S^o$. Um sinal $s \in S^n$ é dito satisfazer um processo se $s \in P$. Um sinal s que satisfaz um processo é chamado de *um comportamento do processo*. Assim, um processo é um conjunto de possíveis comportamentos ou relações entre sinais. Em outras palavras, um processo define relações entre sinais de entrada e sinais de saída.

Um modelo de computação é caracterizado pelos seus processos e pela ordem imposta nos rótulos. Diferentes modelos de tempo podem ser traduzidos em diferentes relações de ordem para o conjunto de rótulos T no modelo de sinal com rótulo.

4.1.2 Estado

Muitos modelos computacionais são baseados na noção de estado e representam o comportamento com seqüências de transição entre estados. Informalmente, *estado* pode ser considerado uma das possíveis situações em que o sistema se encontra. Para sistemas digitais, o estado pode ser interpretado como uma representação do sistema em um intervalo de tempo, onde qualquer transição de sinal gera um comportamento puramente combinacional para o sistema. Ou seja, durante cada estado, durante este intervalo, o sistema digital se comporta como um circuito combinacional. Assim, define-se que uma *seqüência de estados* é uma *máquina de circuitos combinacionais*, que na verdade é o sistema digital em tempos *t* distintos. Isto, intuitivamente leva ao conceito de memória, já que a máquina deve armazenar valores referentes aos circuitos combinacionais para cada tempo *t*. A máquina de circuitos combinacionais é chamada de circuito seqüencial e a ela está associada à idéia de tempo, enquanto que à implementação combinacional está associada à idéia de espaço. Para a comunidade de *hardware*, circuitos com um único estado são chamados de combinacionais, enquanto que os circuitos com mais de um estado são denominados seqüenciais.

Lee e Sangiovanni-Vincentelli (1996) formalizam a definição de estado matematicamente. Este mesmo modelo, descrito a seguir, é adotado por Edwards et al. (1997):

> Seja um sistema temporizado (um sistema no qual rótulos são totalmente ordenados) e um processo P que pertence a este sistema. Então, para qualquer tupla $\boldsymbol{s}$ de sinais, é possível definir $\boldsymbol{s} > t$ como uma tupla dos subconjuntos de eventos em $\boldsymbol{s}$ com rótulos maiores que t. Duas tuplas de sinais de entrada $\boldsymbol{r}$, $\boldsymbol{s} \in S^i$ estão em uma relação E_t^p (denotados por $(\boldsymbol{r}^i, \boldsymbol{s}^i) \in E_t^p$) se $\boldsymbol{r} > t = \boldsymbol{s} > t$ implica em $P(\boldsymbol{r}) > t = P(\boldsymbol{s}) > t$. Esta definição, intuitivamente significa que o processo P não pode distinguir entre as histórias de $\boldsymbol{r}$ e $\boldsymbol{s}$ antes do tempo t. Logo, se as entradas são idênticas após o tempo t, as saídas também serão. Dado que E_t^p é uma relação de equivalência, o particionamento do conjunto de tuplas de sinais de entrada em classes de equivalência para cada t, gera o conceito de estado como sendo cada classe de equivalência de E_t^p para um processo P.

4.1.3 Decidibilidade

A decidibilidade (COHEN, 1997) afeta a complexidade na avaliação de uma especificação. Sistemas indecidíveis não são barreiras insuperáveis, mas também, sistemas decidíveis podem ter, na prática, várias questões sem resposta (e.g. a quantidade de recursos computacionais exigidos para decidir o problema pode ser proibitiva).

O que caracteriza a decidibilidade de um sistema é a condição de previamente saber-se a quantidade de recursos computacionais necessários para a sua operação. Avaliando um sistema com relação ao número de estados, este pode ser classificado como finito ou infinito; Para certos modelos (aqueles que são Turing completos), muitas propriedades desejáveis são impossíveis de serem decididas em uma quantidade de tempo finito. Estas propriedades podem ser a quantidade de memória que o sistema

precisa, a determinação se o sistema irá parar e quão rápido o sistema irá operar. Embora nenhum algoritmo possa resolver estes problemas para todos os sistemas, existem algoritmos que podem resolver estes problemas para a maioria dos sistemas (EDWARDS et al., 1997).

4.1.4 Computação e Comunicação

A *computação* descreve o comportamento de um sistema frente aos estímulos externos recebidos. A *comunicação*, por sua vez, descreve a interação deste sistema com os demais sistemas.

A separação entre computação e comunicação permite que sejam avaliadas com maior clareza alternativas de implementação que atendam às necessidades do sistema.

De acordo com a atividade realizada no fluxo de projeto, há maior relevância em conhecer dados de computação ou de comunicação. Como exemplo, para o mapeamento de tarefas em redes intrachip, a informação mais pertinente é a relativa à necessidade comunicação dos núcleos. Por outro lado, a atividade de particionamento normalmente requer o conhecimento tanto da computação quanto da comunicação. Desta forma, a necessidade de diferentes modelos computacionais para cada atividade é evidente.

4.1.5 Dependência e Concorrência

Dependência e *concorrência* descrevem relações temporais. Utilizando o modelo de sinal com rótulo, eventos concorrentes são aqueles que têm a mesma marca de tempo. Em oposição, eventos dependentes jamais podem ter a mesma marca de tempo. A dependência gera uma ordem parcial entre os eventos. Assim, supondo dois eventos $e_1 = (v_1, t_1)$ e $e_2 = (v_2, t_2)$. Caso e_1 seja dependente de e_2, então, necessariamente $t_2 < t_1$.

As relações de dependência e concorrência podem ser aplicadas a tarefas executando em núcleos de uma aplicação. Sejam duas tarefas t_a e t_b. Normalmente, diz-se que t_a é dependente de t_b se a *computação* de t_a depende do resultado da *computação* de t_b, e a solução desta dependência geralmente é obtida por uma informação proveniente de uma *comunicação* entre t_b e t_a.

4.2 Classificação e Escolha de Modelos Computacionais

Esta Seção descreve resumidamente dois modelos computacionais clássicos: *Discrete Event* e *Petri Nets*, para que o leitor possa compará-los com os modelos estudados e/ou propostos neste trabalho. O autor descreve estes e outros modelos computacionais detalhadamente em (MARCON, 2000) (MARCON, 2001).

Lee e Sangiovanni-Vincentelli (1996) definem formalmente o modelo de *eventos discretos*, em inglês *Discrete Event* (DE), utilizando as noções de eventos e sinais (EDWARDS et al., 1997) (LEE; SANGIOVANNI-VINCENTELLI, 1996). De acordo com estes autores, um *sistema de eventos discretos Q*, é um sistema temporizado, onde para toda tupla de sinais $\boldsymbol{s} \in Q$, a relação de ordem para $T(\boldsymbol{s})$ é

discreta. O que se pode notar é que o rótulo é um componente *imprescindível* do modelo DE. Eventos usualmente transportam rótulos totalmente ordenados, indicando o tempo em que cada evento ocorre. No modelo de evento discreto, o comportamento é geralmente especificado por uma linguagem seqüencial. Cada processo é usualmente executado quando recebe um evento de entrada e produz um evento de saída, com tempo igual ou maior ao evento de entrada. A grande vantagem do modelo para a simulação é a possibilidade de modelar apenas os instantes de tempo em que existem eventos. Assim, o tempo de execução deste modelo depende do nível de atividade da aplicação e não do tempo de execução desta. Ou seja, o simulador não executa instantes de tempo que não representam eventos da aplicação, aumentando o desempenho da simulação.

As redes de Petri, em inglês *Petri Nets* (PN), modelam dependência e concorrência, não-determinismo, comunicação e sincronismo (VALDERRAMA, 2000), e são normalmente implementadas por grafos, onde existem dois tipos de vértices: os *lugares* e as *transições*. Uma aresta com peso associado pode conectar um lugar a uma transição e vice-versa. Lugares podem conter *tokens* e a disposição destes *tokens* representa o estado da rede. A liberação de uma transição equivale à liberação de uma dependência, tornando ativo um novo lugar. O modelo tem sido extensamente explorado para descrever sistemas complexos. Todavia, igualmente como o modelo DE, PNs se mantém utilizadas principalmente para análise. Elas são modelos para os quais outros modelos computacionais são traduzidos na busca de um comportamento mais eficiente.

Tanto DE quanto PN são chamados de modelos homogêneos por não serem compostos por outros modelos computacionais. Muitas aplicações embarcadas possuem características que um único modelo homogêneo não é adequado para expressá-las. Nestes casos, pode ser interessante a composição de *modelos heterogêneos* - aqueles formados por pelo menos dois modelos. Um exemplo é o modelo *Control Data Flow* (CDF), que é obtido pela composição dos modelos *Data Flow* (DF) e *Control Flow* (CF). Normalmente, a parte de controle da aplicação é descrita pelo modelo CF, enquanto a parte operativa é descrita pelo modelo DF.

A eficiência de muitas atividades do fluxo de projeto depende do modelo subjacente à especificação da aplicação. Modelos muito complexos implicam tratamentos igualmente complexos que normalmente consomem muito tempo de CPU e conduzem a soluções distantes do ótimo. Por outro lado, modelos muito simples podem abstrair informações relevantes para a atividade em vista, não permitindo que o problema seja avaliado corretamente. Neste trabalho conclui-se que três elementos são muito importantes na escolha do modelo computacional: (*i*) o tipo de atividade do fluxo de projeto; (*ii*) a função objetivo aplicada a atividade em questão; e (*iii*) as características da aplicação. De acordo com estes elementos, informações das aplicações têm diferentes relevâncias, podendo ser detalhadas ou abstraídas (MARCON et al., 2005c). A seguir são descritos exemplos da importância destes elementos:

1. **Tipo de atividade do fluxo de projeto** - Utilizando-se como exemplo, modelos que capturam informações da comunicação e da computação, para a atividade de particionamento de tarefas em núcleos, o detalhamento da computação de tarefas (e.g. prioridades, deadlines e volume de computação) é normalmente mais relevante que o detalhamento da comunicação entre tarefas (e.g. volume de bits, instante de transmissão da comunicação). Assim, saber os instantes da comunicação entre núcleos é normalmente irrelevante, em oposição aos instantes de computação das

tarefas que podem determinar o número máximo de tarefas que podem ser agrupadas em um determinado núcleo. Por outro lado, a atividade de mapeamento de núcleos em infra-estruturas de comunicação requer maior detalhamento da comunicação entre núcleos, se comparado com a computação de cada núcleo. Isto porque é irrelevante para a infra-estrutura de comunicação a informação de quando uma tarefa será escalonada em um núcleo, mas pode ser de grande importância a informação dos instantes que esta tarefa irá se comunicar com outras tarefas que estão mapeadas em núcleos distintos;

2. **Função objetivo aplicada na atividade do fluxo de projeto** - Considera-se como funções objetivo a latência e o consumo de energia dinâmica. Modelos que capturam informações de tempo, explícitas (com uma marca de tempo) ou implícitas (pela dependência de eventos), são imprescindíveis para estimativas adequadas de latência em infra-estruturas de comunicação. Por outro lado, modelos que capturam a quantidade de transição de bits são adequados para estimar o consumo de energia dinâmica;
3. **Características da aplicação** - Diferentes aplicações têm diferentes graus de concorrência-dependência, computação-comunicação e/ou outras características. Dependendo do modelo computacional, estas características podem ser melhor expressas de forma a atender os objetivos aplicados na atividade do fluxo de projeto.

Além do mais, a escolha adequada do modelo computacional implica menor tempo de projeto, seja pela obtenção da descrição da aplicação, seja pelo tempo de CPU necessário para alcançar uma solução do problema.

Este trabalho tem como atividade principal o mapeamento de núcleos em infra-estruturas de comunicação e para esta atividade os modelos descritos acima podem gerar mapeamentos ineficazes devido a complexidades ou abstrações excessivas. Desta forma, o Capítulo seguinte analisa e/ou propõe modelos computacionais que visam melhorar a qualidade dos mapeamentos frente a requisitos como minimização do consumo de energia e redução do tempo de execução da aplicação.

5 MODELOS COMPUTACIONAIS UTILIZADOS PARA A ATIVIDADE DE MAPEAMENTO DE NÚCLEOS EM INFRA-ESTRUTURAS DE COMUNICAÇÃO

Descrever aplicações através de linguagens que tenham modelo subjacente projetado especificamente para a atividade de mapeamento favorece a obtenção de resultados que atendam os objetivos desejados, uma vez que as informações contidas nestes podem ser exatamente as necessárias e suficientes para esta atividade. Como conseqüência da especificidade destes modelos decorrem vantagens, entre estas: (*i*) menor área necessária para armazenar as informações que representam a aplicação; (*ii*) maior simplicidade nas ferramentas ou métodos para a extração do formato interno que descreve a aplicação; e (*iii*) menor complexidade e maior velocidade dos algoritmos que tratam estes modelos. Por outro lado, algumas abstrações podem remover dos modelos computacionais a capacidade de expressar certas características da aplicação, reduzindo a qualidade das estimativas. Exemplos são os modelos que omitem informações temporais, dificultando estimar o consumo de energia estática e o consumo de energia durante os intervalos de *ociosidade do sistema*[26], dado que estes consumos de energia dependem do tempo de execução da aplicação. Assim, a questão é: "qual o melhor modelo para atender uma determinada atividade". Este Capítulo objetiva preencher esta lacuna, descrevendo modelos computacionais utilizados para a atividade de mapeamento. Alguns destes modelos foram propostos por outros autores, e outros são originais.

Os modelos computacionais tratados neste trabalho são subjacentes ao formato interno das descrições de entrada da atividade de mapeamento. Eles foram elaborados de forma independente da infra-estrutura de comunicação. Para tanto, o problema de mapeamento é colocado de forma que esta seja vista apenas como um objeto sobre o qual núcleos podem ser mapeados. A diferença de cada infra-estrutura de comunicação reside no cálculo da função objetivo, que permite avaliar o custo de cada mapeamento. Desta forma, apenas a função objetivo é dependente da infra-estrutura de comunicação. A obtenção da função objetivo para redes intrachip é abordada no Capítulo 8. A não

[26] A maioria das infra-estruturas de comunicação é composta por circuitos que consomem energia dinâmica mesmo na ausência de tráfego. Este é o caso de circuitos que ficam em laços verificando se há novos dados para serem transmitidos ou recebidos. Nestes casos, o consumo de energia durante os *períodos de ausência de tráfego* (períodos de ociosidade da infra-estrutura de comunicação) não podem ser negligenciados. Algumas infra-estruturas de comunicação conseguem reduzir ou eliminar o consumo de energia dinâmica nestes períodos pela mudança do paradigma de implementação. É o caso das implementações assíncronas, onde a ativação de parte da infra-estrutura de comunicação pode ser realizada pelo próprio tráfego, fazendo com que os períodos de ociosidade apenas impliquem o consumo de energia estática. Todavia, o paradigma assíncrono não é tratado neste trabalho.

dependência da infra-estrutura de comunicação permite que o projetista utilize todos os modelos computacionais descritos neste Capítulo e apenas altere as funções objetivo, caso queira avaliar mapeamentos para outras infra-estruturas de comunicação, tal como barramento segmentado.

5.1 Modelo de Comunicação com Pesos (CWM)

Hu e Marculescu propuseram (2003) a descrição de uma aplicação, para a realização da atividade de mapeamento, através de um *grafo de caracterização da aplicação*, em inglês, *APplication Characterization Graph* (APCG). Murali e De Micheli (2004[A]) propuseram que uma aplicação seja descrita através de um *grafo dos núcleos* (em inglês, *core graph*). Analisando estas descrições, conclui-se que os modelos subjacentes são equivalentes. Para representar genericamente tais descrições, introduz-se o *modelo de comunicação com pesos* (em inglês, *Communication Weighted Model* ou CWM) (MARCON et al., 2005a).

O CWM modela uma aplicação apenas pela quantidade de comunicação, mais precisamente, por todas as comunicações que ocorrem entre pares de núcleo. Neste caso, a comunicação entre núcleos é dada pela soma de todos os bits de todos os pacotes transmitidos de um núcleo para outro. Com este modelo, os autores avaliam a qualidade de mapeamentos frente ao requisito de minimização do consumo de energia dinâmica.

O modelo pode ser capturado por um grafo que descreve a aplicação através de seus núcleos e da quantidade de comunicação entre estes. Este grafo, denominado grafo de comunicação com pesos (em inglês, *Communication Weighted Graph*, CWG), é utilizado como formato interno para o mapeamento de núcleos da aplicação em tiles da arquitetura alvo. A definição formal do CWG é dada a seguir.

> ***Definição 1***: Um *grafo de comunicação com pesos* é um grafo dirigido representado pelo par CWG = $<N, W>$, onde $N = \{n_1, n_2, \ldots, n_c\}$ é o conjunto de vértices e $W = \{w_{12}, w_{13}, ..., w_{1j}, w_{21}, w_{23}, ..., w_{2k}, ..., w_{l1}, w_{l2}, ..., w_{ln-1}\}$ é o conjunto de arestas. O conjunto de vértices representa os núcleos da aplicação, e o conjunto de arestas representa as comunicações entre cada par de núcleos. Seja w_{ijq} o número de bits da mensagem de índice q (m_q) que trafega no meio físico do núcleo n_i para o núcleo n_j, e seja k_{ij} o total de mensagens enviadas de n_i para n_j, então $\omega_{ij} = \sum_{q=1}^{k_{ij}} w_{ijq}$ é o total de bits enviados de enviados de n_i para n_j. Cada aresta $w_{ij} \in W$ é uma tripla $w_{ij} = <n_i, n_j, \omega_{ij}>$, tal que $n_i \neq n_j$ e $\omega_{ij} > 0$. No limite máximo do número de comunicações, que ocorre quando todos os núcleos estão conectados entre si, temos $|W| = 2 * C^2_{|N|}$[27]

A Figura 5.1(a) descreve uma aplicação sintética com 6 conjuntos de mensagens trocadas entre quatro núcleos $N = \{n_1, n_2, n_3, n_4\}$, com o número total de bits

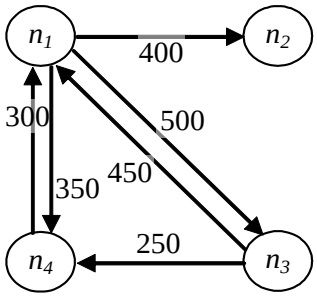
---

[27] $|W| = 2 * C^2_c$ significa que o número de arestas de um CDG é igual a duas vezes a combinação de todos os núcleos dois a dois.

enviado no meio físico dado por $\omega_{12} = 400$, $\omega_{13} = 500$, $\omega_{14} = 350$, $\omega_{31} = 450$, $\omega_{34} = 250$, $\omega_{41} = 300$. Esta aplicação é representada pelo CWG da Figura 5.1(b).

$w_{14} = <n_1, n_4, 350>$
$w_{41} = <n_4, n_1, 300>$
$w_{34} = <n_3, n_4, 250>$
$w_{12} = <n_1, n_2, 400>$
$w_{31} = <n_3, n_1, 450>$
$w_{13} = <n_1, n_3, 500>$

(a)



(b)

Figura 5.1: Exemplo de CWM para uma aplicação sintética, onde: (a) apresenta a estrutura de *W*, e (b) mostra o CWG e a aplicação.

5.2 Modelo estendido de Comunicação com Pesos (ECWM)

O *modelo estendido de comunicação com pesos*, em inglês, *Extended Communication Weighted Model* (ECWM) foi introduzido em (MARCON et al., 2005d) e (PALMA; MARCON; MORAES; CALAZANS; REIS; SUSIN, 2005). Este modelo é um refinamento do CWM. O ECWM faz parte da mesma classe de modelos do tipo CWM, acrescentando, porém, informação do número de transições de cada comunicação. Em (MARCON et al., 2005d) e (PALMA; MARCON; MORAES; CALAZANS; REIS; SUSIN, 2005) mostra-se que a ausência da informação do número de bits que variam durante uma transmissão pode conduzir a erros de estimativa do consumo de energia dinâmica que não podem ser ignorados. Nos estudos de caso apresentados, este erro está da ordem de 45%.

O modelo pode ser capturado por um grafo que descreve a aplicação através de seus núcleos e da comunicação entre estes. A comunicação é composta pelo número de bits transmitidos no meio físico, junto com o número de transições entre bits consecutivos. Este grafo, denominado grafo estendido de comunicação com pesos (em inglês, *Extended Communication Weighted Graph*, ECWG), é utilizado como formato interno para o mapeamento de núcleos da aplicação em tiles da arquitetura alvo. A definição formal do ECWG é dada a seguir.

> ***Definição 2***: Um *grafo estendido de comunicação com pesos* é um grafo dirigido representado pelo par ECWG = <*N*, *H*>, onde $N = \{n_1, n_2, \ldots, n_c\}$ é o conjunto de vértices e $H = \{H_{12}, H_{13}, ..., H_{1j}, H_{21}, H_{23}, ..., H_{2k}, ..., H_{l1}, H_{l2}, ..., H_{lm-1}\}$ é o conjunto de arestas. O conjunto de vértices representa os núcleos da aplicação, e o conjunto de arestas representa as comunicações entre cada par de núcleos. Seja w_{ijq} o número de bits da mensagem de índice q (m_q), que trafega no meio físico do núcleo n_i para o núcleo n_j, s_{ijq} o número de transição de bits consecutivos desta comunicação, e seja k_{ij} o total de mensagens enviadas de n_i para n_j, então $\omega_{ij} = \sum_{q=1}^{k_{ij}} w_{ijq}$ e $\sigma_{ij} = \sum_{q=1}^{k_{ij}} s_{ijq}$ são o número total de bits enviados e o número de transições que ocorrem em uma comunicação do núcleo n_i para o núcleo n_j, respectivamente.

Cada aresta $h_{ij} \in H$ é uma quádrupla $h_{ij} = <n_i, n_j, \omega_{ij}, \sigma_{ij}>$, tal que $n_i \neq n_j$, $\omega_{ij} > 0$ e $0 \leq \sigma_{ijq} < \omega_{ijq}$.

A Figura 5.2(a) descreve uma aplicação sintética com 6 conjuntos de mensagens trocadas entre quatro núcleos $N = \{n_1, n_2, n_3, n_4\}$, com o respectivo número total de bits e transições dado por $\omega_{12} = 400$, $\sigma_{12} = 200$, $\omega_{13} = 500$, $\sigma_{13} = 30$, $\omega_{14} = 350$, $\sigma_{14} = 50$, $\omega_{31} = 450$, $\sigma_{31} = 35$, $\omega_{34} = 250$, $\sigma_{34} = 30$, $\omega_{41} = 300$, $\sigma_{41} = 100$. Esta aplicação é representada pelo ECWG da Figura 5.2(b).

$h_{14} = <n_1, n_4, 350, 50>$

$h_{41} = <n_4, n_1, 300, 100>$

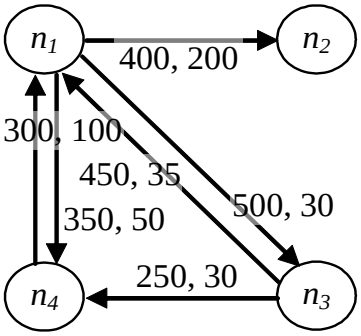
$h_{34} = <n_3, n_4, 250, 30>$

$h_{12} = <n_1, n_2, 400, 200>$

$h_{31} = <n_3, n_1, 450, 35>$

$h_{13} = <n_1, n_3, 500, 30>$

(a)



(b)

Figura 5.2: Exemplo de ECWM para uma aplicação sintética, onde: (a) apresenta a estrutura *H*, e (b) mostra o ECWG equivalente para esta aplicação.

5.3 Modelo de Dependência das Comunicações (CDM)

O *modelo de dependência das comunicações* (em inglês, *Communication Dependence Model* ou CDM), foi introduzido em (MARCON et al., 2005a). O CDM modela uma aplicação através da quantidade e dependência das comunicações. Todas as mensagens transmitidas entre núcleos são expressas no modelo, que tem como ponto forte a capacidade de modelar a dependência entre mensagens. Cada mensagem contém exatamente o número de bits que devem ser transmitidos do núcleo fonte para o núcleo destino. O CDM é similar a uma versão simplificada das redes de Petri. A vantagem deste modelo frente ao CWM é que a informação de dependência entre mensagens permite prever quais mensagens podem concorrer pelo mesmo *recurso de comunicação*[28], e com esta informação reduzir as contenções de mensagens na infra-estrutura de comunicação. O objetivo do CDM é permitir que sejam construídos algoritmos que mapeiem núcleos de forma a evitar ou reduzir a competição de mensagens pelos mesmos recursos de comunicação. Isto é feito de forma estática e pessimista, pois os algoritmos baseados neste modelo assumem que, caso duas mensagens **possam** concorrer pelo mesmo recurso durante o mesmo intervalo de tempo elas **irão** concorrer pelo mesmo recurso. O algoritmo pesquisa então novos mapeamentos que evitem ou pelo menos reduzam esta competição. Este comportamento pessimista objetiva eliminar contenções de pacotes na infra-estrutura de comunicação. A conseqüência é a diminuição do tempo de execução da aplicação e a redução dos consumos de energia estática de todo o circuito e dinâmica dos circuitos que operam mesmo frente à ausência de tráfego.

[28] *Recurso de comunicação* no contexto deste trabalho é qualquer elemento que faz parte da infra-estrutura de comunicação, tais como roteadores e conexões para redes intrachip, ou segmentos e portas de conexão para barramento segmentado.

O modelo pode ser capturado por um grafo que descreve a aplicação através da dependência das comunicações entre os núcleos. Este grafo, denominado grafo de dependência das comunicações (em inglês, *Communication Dependence Graph*, CDG), é utilizado como formato interno para o mapeamento de núcleos da aplicação em tiles da infra-estrutura de comunicação. A definição formal do CDG é dada a seguir.

> ***Definição 3***: Um *grafo de dependência das comunicações* é um grafo dirigido acíclico CDG = $<M, D>$. O conjunto de vértices M caracteriza as mensagens trocadas entre pares de núcleos da aplicação. O conjunto M também contém dois vértices especiais, denominados INÍCIO e FIM. O conjunto de arestas D representa as dependências de comunicação da aplicação. Seja $N = \{n_1, n_2, \ldots, n_c\}$ o conjunto de núcleos de uma dada aplicação. Então, o conjunto de vértices de CDG é $M = \{m_q = (q,\ n_a, n_b, w_{abq}) \mid q \in \mathbb{N},\ n_a, n_b \in N$ e $w_{abq} \in \mathbb{N}^*\} \cup \{$INÍCIO**,** FIM$\}$, tal que m_q é a mensagem de índice q, sendo transmitida do núcleo n_a para o núcleo n_b com a quantidade de bits que trafegam pelo meio físico igual a w_{abq} e $w_{abq} > 0$. q é um identificador que permite distinguir diferentes mensagens entre mesmos pares de núcleos. O conjunto de arestas é $D = \{(m_r, m_q) \mid m_r, m_q \in M\}$, onde $(m_r, m_q) \in D$ se e somente se m_r depende de m_q, ou seja, se a transmissão de m_r só puder ocorrer após a transmissão de m_q. Adicionalmente, toda mensagem m_r que não depende de nenhuma outra mensagem implica a existência da aresta (m_r, INÍCIO) em D, e toda a mensagem m_r da qual nenhuma outra mensagem depende implica a existência da aresta (FIM, m_r) em D.

INÍCIO e FIM marcam respectivamente o início e o fim do grafo que descreve a aplicação. Nenhuma aresta chega a INÍCIO e nenhuma aresta sai de FIM.

O CDG representa a comunicação de uma aplicação composta por um número arbitrário de núcleos. A direção dos vértices no grafo denota que a comunicação descrita no vértice destino depende da comunicação descrita no vértice origem. Em outras palavras, o vértice destino apresenta uma *dependência de comunicação* com relação ao vértice origem.

Existem dois tipos de dependência de comunicação: (*i*) a originada por mensagens subseqüentes provenientes do mesmo núcleo, denominada *dependência intranúcleo*; e (*ii*) a originada por mensagens provenientes de núcleos distintos, denominada *dependência internúcleos*. A primeira é extraída diretamente do algoritmo que descreve o comportamento do núcleo. A segunda é extraída da interação entre núcleos da aplicação. O motivo de diferenciar estas dependências está na estimativa de tempo de comunicação. Uma dependência intranúcleo é resolvida quando a mensagem é lançada na infra-estrutura de comunicação. Uma dependência internúcleos, por sua vez, é resolvida somente quando a mensagem for entregue ao núcleo destino. Embora haja esta distinção de dependências da comunicação, não há necessidade de torná-la explícita na descrição da aplicação, pois os algoritmos de mapeamento conseguem detectá-la automaticamente, bastando para isto saber qual a origem da mensagem dependente.

A Figura 5.3(a) apresenta uma aplicação sintética com troca de 8 mensagens entre quatro núcleos $N = \{n_1, n_2, n_3, n_4\}$. O conjunto de mensagens é dado por $M = \{(1, n_1, n_4, 350),\ (2, n_4, n_1, 300),\ (3, n_3, n_4, 250),\ (4, n_1, n_2, 400),\ (5, n_3, n_1, 450),\ (6, n_1, n_3, 500),\ (7, n_2, n_3, 450),\ (8, n_1, n_4, 350)\}$. A dependência de mensagens (comunicação) é dada por $D = \{(m_1,$ INÍCIO$),\ (m_2,$ INÍCIO$),\ (m_3,$ INÍCIO$),\ (m_7,$ INÍCIO$),$

(m_4, m_1), (m_4, m_2), (m_5, m_3), (m_8, m_1), (m_6, m_4), (m_6, m_5), (FIM, m_6), (FIM, m_7), (FIM, m_8)}, onde:

1. As mensagens m_1, m_2, m_3 e m_7 dependem apenas do vértice especial INÍCIO;
2. m_4 é dependente das mensagens m_1 e m_2. Isto significa que m_4, representando a comunicação do núcleo n_1 para o núcleo n_2, somente pode ocorrer após n_1 ter enviado m_1 para n_4 e ter recebido m_2 de n_1;
3. m_5 é dependente de m_3;
4. m_8 é dependente de m_1;
5. m_6 é dependente das mensagens m_4 e m_5;
6. As trocas de mensagens entre núcleos da aplicação somente encerram após o término das mensagens m_6, m_7 e m_8;
7. As mensagens m_1 e m_8, embora ocorram entre os mesmos núcleos e tenham o mesmo número de bits transmitidos, são diferentes, pois possuem identificadores distintos.

Supondo este ser o comportamento da comunicação da aplicação sintética, ela seria representada pelo CDG da Figura 5.3(b).

m_1 = (1, n_1, n_4, 350)

m_2 = (2, n_4, n_1, 300)

m_3 = (3, n_3, n_4, 250)

m_4 = (4, n_1, n_2, 400)

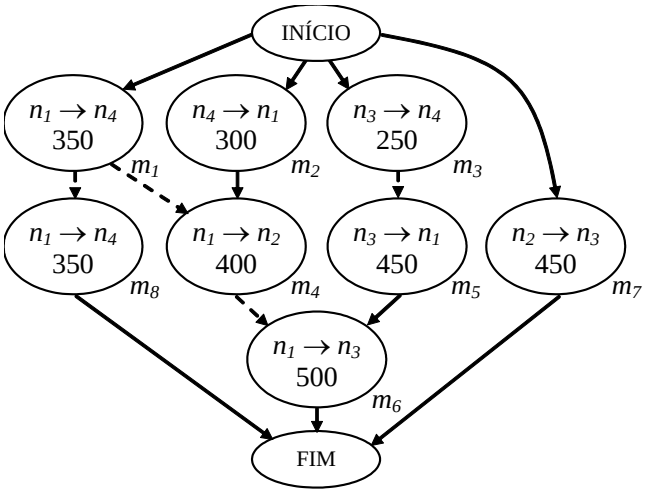
m_5 = (5, n_3, n_1, 450)

m_6 = (6, n_1, n_3, 500)

m_7 = (7, n_2, n_3, 450)

m_8 = (8, n_1, n_4, 350)

(a)



(b)

Figura 5.3: Exemplo de CDM para uma aplicação sintética, onde: (a) apresenta o conjunto *M* e (b) mostra o CDG equivalente para esta aplicação, considerando as dependências descritas em (a). As flechas contínuas apresentam dependências internúcleos, enquanto que as flechas tracejadas apresentam dependência intranúcleo.

5.4 Modelo de Computação e Dependência das Comunicações (CDCM)

O *modelo de computação e dependência das comunicações* (em inglês, *Communication Dependence and Computation Model* ou CDCM), que representa uma aplicação pela sua comunicação e computação, foi introduzido em (MARCON et al., 2005b). O CDCM é obtido acrescendo ao CDM a quantidade de computação de núcleos da aplicação. Esta informação permite que o tempo de execução da aplicação seja

estimado de forma mais precisa, pois a cada mensagem está associado o tempo de computação do núcleo que origina a comunicação. Este tempo é determinado pelo intervalo compreendido entre o momento que uma mensagem foi ativada, pela solução de suas dependências, e o momento de envio desta mensagem. Associando o tempo de computação dos núcleos, algoritmos que implementam o CDCM podem ser usados para estimar de forma mais precisa o consumo de energia durante o período de ociosidade da infra-estrutura de comunicação, que é obtido apenas de forma "pessimista" pelo CDM, devido a este último negligenciar o tempo de computação.

O CDCM é aqui representado por um grafo de computação e dependência das comunicações (em inglês, *Communication Dependence and Computation Graph* ou CDCG), que é utilizado como formato interno para o mapeamento de núcleos da aplicação em tiles da arquitetura alvo. A definição formal do CDCG é dada a seguir.

> ***Definição 4***: Um *grafo de computação e dependência das comunicações* é um grafo dirigido acíclico CDCG = $<M, D>$. O conjunto de vértices M representa as mensagens trocadas entre pares de núcleos da aplicação e as computações que precedem o envio das mensagens. M contém também dois vértices especiais denominados INÍCIO e FIM. O conjunto de arestas D representa as dependências de comunicação. Seja $N = \{n_1, n_2, \ldots, n_c\}$ o conjunto de núcleos de uma aplicação. Então, o conjunto de vértices de CDCG é $M = \{m_q = (q, n_a, n_b, t_{aq}, w_{abq}) \mid n_a, n_b \in N, q, t_{aq} \in \mathbb{N}$ e $w_{abq} \in \mathbb{N}^*\} \cup$ {INÍCIO**,** FIM}, onde m_q é a mensagem de índice q, sendo transmitida do núcleo n_a para o núcleo n_b. w_{abq} representa a quantidade de bits que trafegam pelo meio físico, t_{aq} representa o tempo de computação que, após terem sido satisfeitas todas as dependências do vértice, precede a transmissão da q-ésima mensagem, e q é um identificador que permite distinguir diferentes mensagens entre mesmos pares de núcleos. O conjunto de arestas é $D = \{(m_r, m_q) \mid m_r, m_q \in M\}$, onde $(m_r, m_q) \in D$ se e somente se m_r depende de m_q, ou seja, se a transmissão de m_r só puder ocorrer após a transmissão de m_q. Adicionalmente, toda mensagem m_r que não depende de nenhuma outra mensagem implica a existência da aresta (m_r, INÍCIO) em D, e toda a mensagem m_r da qual nenhuma outra mensagem depende implica a existência da aresta (FIM, m_r) em D.

INÍCIO e FIM marcam respectivamente o início e o fim do grafo que descreve a aplicação. Nenhuma aresta chega a INÍCIO e nenhuma aresta sai de FIM.

O CDCG representa a comunicação e a computação de uma aplicação composta por $|N|$ núcleos. A direção dos vértices no grafo denota que a computação do núcleo no vértice destino depende da transmissão da mensagem no vértice origem. Em outras palavras, o vértice destino apresenta uma *dependência de comunicação* com relação ao vértice origem.

De forma análoga ao CDM, o CDCM também modela as dependências intranúcleo e internúcleos. Estas dependências podem ser observadas na Figura 5.4, que apresenta uma aplicação sintética com troca de 6 mensagens entre quatro núcleos ($N = \{n_1, n_2, n_3, n_4\}$). O conjunto de mensagens é dado por $M = \{(1, n_1, n_2, 6, 15), (2, n_3, n_1, 10, 20), (3, n_2, n_4, 10, 40), (4, n_1, n_4, 6, 15), (5, n_3, n_1, 20, 15), (6, n_4, n_2, 6, 15)\}$. A dependência de mensagens é dada por $D = \{(m_1$, INÍCIO), (m_2, INÍCIO), (m_3, INÍCIO), (m_4, m_1), (m_4, m_2), (m_5, m_2), (m_6, m_4), (FIM, m_3), (FIM, m_5),

(FIM, m_6)}, implicando:

1. As mensagens m1, m2 e m3 não dependem de nenhuma mensagem;
2. m4 depende das mensagens m1 e m2;
3. m5 depende de m2, e;
4. m6 depende de m4.
5. As trocas de mensagens entre núcleos da aplicação somente encerram após o envio das mensagens m_3, m_5 e m_6;

A Definição 4 pode ser entendida a partir do exemplo mostrado na Figura 5.4(a), cuja aplicação seria descrita pelo CDCG da Figura 5.4(b).

$m_1 = (1, n_1, n_2, 6, 15)$

$m_2 = (2, n_3, n_1, 10, 20)$

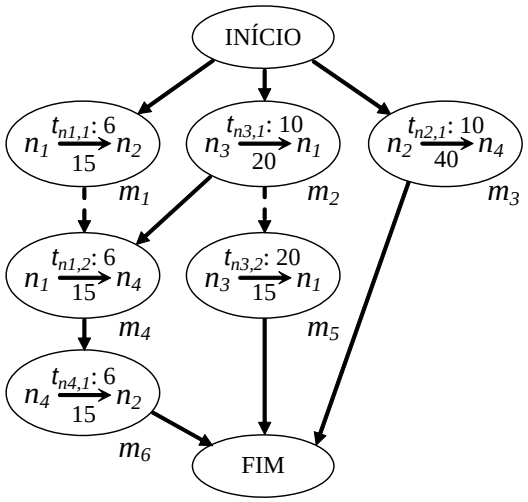
$m_3 = (3, n_2, n_4, 10, 40)$

$m_4 = (4, n_1, n_4, 6, 15)$

$m_5 = (5, n_3, n_1, 20, 15)$

$m_6 = (6, n_4, n_2, 6, 15)$

(a)



(b)

Figura 5.4: Exemplo de CDCM para uma aplicação sintética, onde: (a) apresenta o conjunto *M* e (b) mostra um CDCG equivalente para esta aplicação, considerando as dependências citadas no texto. As flechas contínuas apresentam dependências internúcleos, enquanto que as flechas tracejadas apresentam dependência intranúcleo.

5.5 Modelo do Padrão de Comunicação da Aplicação (ACPM)

O *modelo do padrão de comunicações da aplicação*, em inglês, *Application Communication Pattern Model* (ACPM), foi introduzido em (KREUTZ; MARCON; CALAZANS; CARRO; SUSIN, 2005a) e (KREUTZ; MARCON; CARRO; WAGNER; SUSIN, 2005b). ACPM faz parte da classe de modelos DE. Este modela uma aplicação através de eventos totalmente ordenados, de tal forma que todas as mensagens têm uma marca de tempo associada. A marca de tempo indica o instante que uma mensagem é enviada de um núcleo para outro.

O ACPM, como definido em (KREUTZ; MARCON; CALAZANS; CARRO; SUSIN, 2005a), não descreve as necessidades de computação dos núcleos da aplicação. Por este motivo, não consegue definir com maior precisão o tempo de execução da aplicação, apenas da mesma forma otimista que CDM.

O ACPM pode ser realizado por uma estrutura chamada de *padrão de comunicação da aplicação*, em inglês, *Application Communication Pattern* (ACP), que define conjuntos de mensagens que são disparadas a cada marca de tempo. O par

(conjunto de mensagens, tempo) define um evento do modelo. A definição formal de ACP é dada a seguir.

> ***Definição 5***: *Padrão de comunicação da aplicação* é um conjunto totalmente ordenado ACP. Seja $N = \{n_1, n_2, \ldots, n_c\}$ o conjunto de núcleos da aplicação. Seja $M = \{m_1, m_2, \ldots, m_k\}$ o conjunto de todas as mensagens trocadas entre os núcleos da aplicação, tal que cada mensagem m_q possui estrutura $m_q = (n_a, n_b, w_{abq})$, com $n_a, n_b, \in N$ e $w_{abq} \in \mathbb{N}^*$. Para conjuntos de mensagens associa-se t como uma marca de tempo que determina o instante de início de transmissão das mensagens do conjunto. Então, ACP = $\{T_i = (i, \boldsymbol{m}_i) \mid \boldsymbol{m}_i \subseteq M, \boldsymbol{m}_i \neq \varnothing$ e $i \in \mathbb{N}\} \cup \{$INÍCIO $= (0, \varnothing)$, FIM $= (\infty, \varnothing)\}$. INÍCIO e FIM são pares especiais de ACP, tal que existe uma relação de ordem total definida sobre os elementos de ACP, com estrutura dada por:
>
> $$\{(T_i, T_j) \mid T_i = (i, \boldsymbol{m}_i), T_j = (j, \boldsymbol{m}_j), i < j \text{ e } \nexists k \mid (k, \boldsymbol{m}_k) \in \text{ACP e } i < k < j\}$$

i é a marca de tempo que define o início da transmissão de todas as mensagens que fazem parte do conjunto $\boldsymbol{m}_i$. INÍCIO e FIM são pares especiais, onde INÍCIO contém a marca de tempo 0 e FIM contém a marca de tempo ∞. Estes pares determinam o início e fim do ordenamento, respectivamente. Ambos estão associados a um conjunto vazio de mensagens.

A Definição 5 pode ser melhor compreendida a partir da Figura 5.5, que apresenta uma aplicação sintética com a troca de 6 mensagens entre quatro núcleos $N = \{n_1, n_2, n_3, n_4\}$, com o respectivo número total de bits enviado por mensagem e o tempo de início da transmissão de cada mensagem. O conjunto de mensagens é dado por $M = \{(n_1, n_2, 15), (n_3, n_1, 20), (n_3, n_1, 15), (n_1, n_4, 15), (n_2, n_4, 40), (n_4, n_2, 15)\}$. Os subconjuntos de mensagens são: $\boldsymbol{m}_1 = \{m_1, m_2\}$, $\boldsymbol{m}_2 = \{m_3, m_4, m_5\}$ e $\boldsymbol{m}_3 = \{m_6\}$. Assim, ACP = $\{(1, \boldsymbol{m}_1), (2, \boldsymbol{m}_2), (3, \boldsymbol{m}_3)\}$. A representação do ACP da aplicação sintética descrita é dada pela Figura 5.5(b).

$T_0 = (0, \varnothing)$

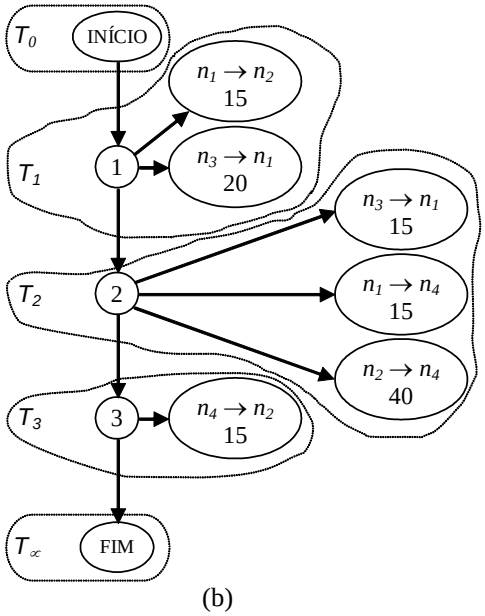
$T_1 = (1, \{(n_1, n_2, 15), (n_3, n_1, 20)\})$

$T_2 = (2, \{(n_3, n_1, 15), (n_1, n_4, 15), (n_2, n_4, 40)\})$

$T_3 = (3, \{(n_4, n_2, 15)\})$

$T_\infty = (\infty, \varnothing)$

(a)



(b)

Figura 5.5: Exemplo de ACPM para uma aplicação sintética, onde: (a) apresenta a estrutura geral do ACP, e (b) mostra o ACP sob forma de um grafo que

descreve esta aplicação, ilustrando a ordenação total dos tempos de início de transmissão das mensagens.

5.6 Modelo de Tarefas da Comunicação (CTM)

O *modelo de tarefas da comunicação* (em inglês, *Communication Task Model* ou CTM), foi introduzido por Hu e Marculescu (2004)[29]. O CTM é um modelo que trata da computação e da comunicação da aplicação pela modelagem do escalonamento de tarefas e pela taxa de comunicação entre estas. Enquanto a computação é representada pela sua dependência, a comunicação é expressa pela quantidade de bits transmitida entre duas tarefas. O modelo foi planejado para que aplicações de tempo real possam ser modeladas, inserindo restrições de tempo para as execuções das tarefas da aplicação.

O CTM modela uma aplicação como um conjunto de tarefas a serem mapeadas em um conjunto de elementos de processamento, em inglês, *Processing Elements* (PE), cada um destes PEs posicionados em um respectivo tile. Múltiplas tarefas podem ser agrupadas em um mesmo PE, e neste caso a comunicação destas com a infra-estrutura de comunicação não será concorrente. Esta característica permite que o modelo agregue informações sobre as atividades de particionamento e mapeamento da aplicação. O agrupamento de tarefas define conjuntos de blocos da aplicação, ou seja, uma partição. O mapeamento, por outro lado, é obtido pela associação destes blocos aos PEs.

Analogamente a determinação de dependência das mensagens contida nos modelos CDCM e CDM, o CTM determina a dependência das tarefas pelo escalonamento.

O CTM pode ser implementado por um grafo denominado *grafo de tarefas da comunicação* (em inglês, *Communication Task Graph*, CTG). Este é utilizado como formato interno para o particionamento e mapeamento de tarefas em PEs. A definição formal de CTG é dada a seguir.

> ***Definição 6***: Um *grafo de tarefas da comunicação* é um grafo dirigido acíclico CTG = <*T*, *C*>, onde *T* é o conjunto de vértices e *C* é o conjunto de arestas. Cada vértice t_i representa uma tarefa da aplicação definida pela tripla $t_i = (d_i, T_i, E_i)$. O elemento d_i é o requisito de tempo limite máximo (deadline) para conclusão de t_i. Caso este não seja fornecido, assume-se $d_i = \infty$, ou seja, não há limite de tempo para concluir a tarefa. T_i é um vetor cujos elementos t_{Eij} representam o tempo de execução da tarefa t_i no j-ésimo PE da arquitetura. E_i é um vetor cujos elementos e_{ij} fornecem o consumo de energia da tarefa t_i ao ser executada sobre o j-ésimo PE na arquitetura. Cada aresta $c_{ij} \in C$ caracteriza a dependência de controle e comunicação entre pares (t_i, t_j). A cada c_{ij} está associado a quantidade de comunicação ω_{ij}, que representa o número de bits transmitidos de t_i para t_j. Se $\omega_{ij} > 0$, então t_j pode iniciar apenas depois de t_i ter terminado e ter transferido ω_{ij} bits para a tarefa t_j. O grafo contém dois vértices especiais: INÍCIO e FIM, que marcam o início e o fim da aplicação, respectivamente.

[29] Hu e Marculescu (2004) chamam o modelo de comunicação de *Communication Task Graph* (CTG). Neste trabalho usa-se a denominação alternativa *Communication Task Model* (CTM), para diferenciar o modelo de sua representação.

INÍCIO não depende de nenhum vértice e nenhum vértice depende de FIM.

A Figura 5.6 apresenta um CTG com 6 tarefas $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, que devem ser mapeadas em 2 PEs. Ainda, $T = \{t_1 = (10, [5, 14], [12, 40])$, $t_2 = (20, [8, 12], [7, 22])$, $t_3 = (15, [9, 9], [6, 11])$, $t_4 = (32, [13, 12], [5, 21])$, $t_5 = (17, [8, 16], [32, 32])$ e $t_6 = (12, [12, 23], [10, 16])\}$. As arestas do CTG representam a dependência das tarefas e a quantidade de comunicação: $C = \{(\text{INÍCIO} \rightarrow t_1, 0)$, $(t_1 \rightarrow t_2, 40)$, $(t_1 \rightarrow t_3, 35)$, $(t_1 \rightarrow t_5, 20)$, $(t_2 \rightarrow t_4, 43)$, $(t_3 \rightarrow t_4, 60)$, $(t_4 \rightarrow t_6, 10)$, $(t_5 \rightarrow t_6, 80)$, $(t_6 \rightarrow \text{FIM}, 0)\}$.

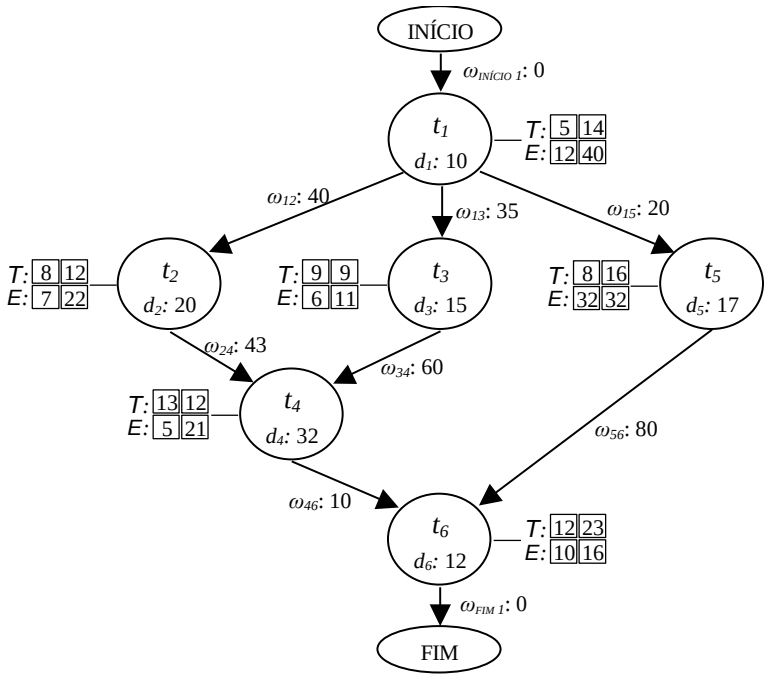


Figura 5.6: Exemplo de CTM para uma aplicação sintética composta por 6 tarefas que devem ser mapeadas em 2 elementos de processamento.

6 EXEMPLO DE APLICAÇÃO DE SEGMENTAÇÃO DE IMAGEM MODELADA PARA A ATIVIDADE DE MAPEAMENTO

Este Capítulo apresenta uma aplicação de *segmentação de imagem para reconhecimento de objetos* desenvolvida por Borin (2004), denominada SegImag. SegImag foi completamente descrita e verificada em C++. Esta aplicação objetiva acelerar o processo de identificar o número de objetos de uma imagem. Para tanto, a imagem original deve ser particionada em segmentos, onde cada segmento é tratado concorrentemente por um processador auxiliar (PA). Além do PA, SegImag contém dois outros elementos de processamento (PEs): um processador central (PC) e uma memória externa (ME). O padrão de comunicação da aplicação com PAs está ilustrado na Figura 6.1. A finalidade de descrever esta aplicação é ilustrar o uso dos modelos apresentados no Capítulo 5.

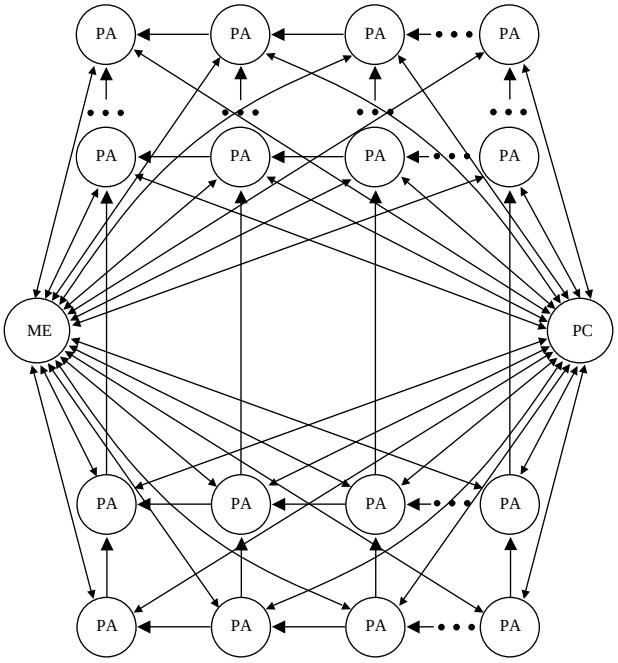


Figura 6.1: Padrão de comunicação para a aplicação SegImag.

Na aplicação SegImag o número de PAs é parametrizável. SegImag é uma aplicação onde podem ser comparadas implementações com diferentes necessidades de computação e comunicação, ou seja, a relação entre a computação de cada PA com o número de PAs e a sobrecarga causada pela comunicação entre estes. Dois PAs, um PC e uma ME são suficientes para implementar a aplicação SegImag. A comunicação está relacionada com o número de segmentos e tamanho da vizinhança de um PA.

SegImag permite avaliar diversos aspectos, tais como o melhor mapeamento para reduzir o consumo de energia e o tempo de comunicação. Outro aspecto é a melhor relação de particionamento de imagens para gerar o melhor número de PAs, ou seja, aquele que atende o requisito de projeto (a avaliação de um número de imagens por segundo) e tem o menor consumo de energia. Contudo, este Capítulo restringe-se a apresentar a modelagem da aplicação.

Assume-se que a imagem a ser segmentada encontra-se inicialmente na ME. Esta imagem é transferida para os PAs (um segmento para cada PA), que contabilizam em paralelo a quantidade de objetos de seu segmento. Nesta etapa inicial, cada PA calcula os objetos de seu segmento e atribui para os pixels da imagem uma numeração, gerando uma associação de pixel com objeto. A seguir, cada PA comunica-se com seu vizinho esquerdo, enviando a coluna de pixels esquerda (já numerada) para este verificar se existem objetos adjacentes nas fronteiras. O mesmo é feito com o PA vizinho acima. Esta etapa é necessária para que os objetos vizinhos não sejam contabilizados mais de uma vez. Após, todos os PAs enviam uma mensagem para o PC com o número de objetos encontrados e os pares de segmentos adjacentes. O PC pega todas as numerações individuais, gera uma numeração global, compacta a numeração global tirando as redundâncias geradas pelos segmentos adjacentes e devolve para os PAs a nova numeração. Os PAs substituem a nova numeração no seu segmento de imagem e retransmitem a imagem com o número de objetos contabilizado para a ME.

A aplicação SegImag está exemplificada aqui com uma imagem (quadro) de 640 × 480 pixels, com cada pixel ocupando um byte, uma taxa de processamento (t_P) de 15 quadros/seg, e 4 PAs (n_{PA}). Assim, o número de bytes em X (n_{BX}) é 640 e o número de bytes em Y (n_{BY}) é 480. Como existem 2 PAs para cada coordenada, o número de bytes que cada PA deve processar em X (n_{BPX}) é 320 e o número de bytes que cada PA deve processar em Y (n_{BPY}) é 240. As características da imagem e de cada segmento são apresentadas na Figura 6.2 e detalhadas na Tabela 6.1.

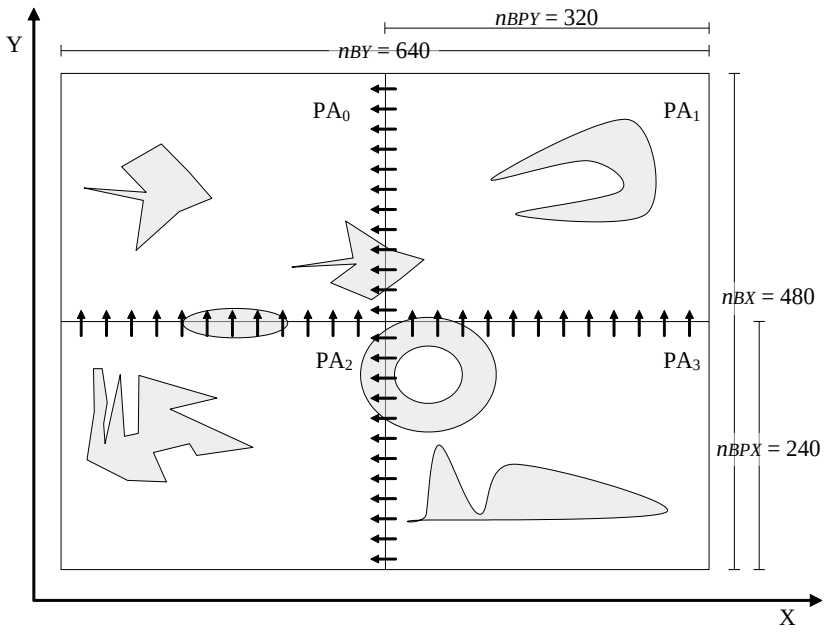


Figura 6.2: Exemplo de imagem com quatro segmentos e os correspondentes PAs para cada segmento.

Tabela 6.1: SegImag com uma imagem de 640 × 480 bytes e apenas um quadro.

PA ↔ ME		Número de bytes total da imagem (n_{BI}) = 640 × 480 = 307200 bytes
		Número de bytes de cada segmento (n_{BS}) = n_{BI} / n_{PE} = 307200 / 4 = 76800 bytes
PA ↔ PA	Em X	Número de bytes da imagem em Y (n_{BY}) = 480 bytes
		Número de PAs em Y (n_{PAY}) = 2
		Número de bytes na fronteira de cada PA para Y (n_{BPY}) = n_{BY} / n_{PAY} = 480 / 2 = 240 bytes
	Em Y	Número de bytes da imagem em X (n_{BX}) = 640 bytes
		Número de PAs em X (n_{PAX}) = 2
		Número de bytes na fronteira de cada PA para X (n_{BPX}) = n_{BX} / n_{PAX} = 640 / 2 = 320 bytes
PA ↔ PC		Número de bytes de controle de vizinhança (n_{CV}) = 128 bytes (estimado com base no tamanho dos objetos em imagens típicas)

Para o exemplo ilustrado, as comunicações entre PAs, entre PAs e ME, e entre PAs e PC apresentam as taxas descritas na Tabela 6.2.

Tabela 6.2: Taxas de comunicação para a aplicação de segmentação de imagens, considerando uma imagem de 640 × 480 bytes e 15 quadros por segundo.

		Taxas de comunicação (bytes/s)
PA ↔ ME		PA com MEM (t_{CPM}) = n_{BS} * t_P = 76800 * 15 * 2 = 2304000 bytes/s
PA ↔ PA	Em X	PA com seu vizinho em X (t_{CPPX}) = n_{BPY} * t_P = 320 * 15 = 4800 bytes/s
	Em Y	PA com seu vizinho em Y (t_{CPPY}) = n_{BPX} * t_P = 240 * 15 = 3600 bytes/s
PA ↔ PC		PA com o PC (t_{CPPC}) = n_{CV} * t_P = 128 * 15 * 2 = 3840 bytes/s

As operações de cada PA, do PC e da ME são um fluxo de dados, tal como descrito nos fluxos de 16 tarefas da Figura 6.3. Estes fluxos de tarefas foram extraídos da descrição C++ (BORIN, 2004).

t_1: Recebe imagem da ME t_2: Processa imagem Se (tem PA vizinho esquerdo) t_3: Envia fronteira p/ PA esquerdo Se (tem PA vizinho acima) t_4: Envia fronteira para PA acima Se (tem PA vizinho direito) t_5: Recebe fronteira do PA direito Se (tem PA vizinho abaixo) t_6: Recebe fronteira do PA abaixo Se (recebeu fronteira) t_7: Processa fronteira t_8: Envia cv para PC t_9: Recebe cv do PC t_{10}: Processa imagem t_{11}: Envia imagem para ME	t_{12}: Enquanto (houverem PAs) Recebe cv de um PA t_{13}: Processa todos cvs t_{14}: Enquanto (houverem PAs) Envia cv totalizado para PA	t_{15}: Enquanto (houverem PAs) Envia imagem para um PA t_{16}: Enquanto (houverem PAs) Recebe imagem de cada PA
(a)	(b)	(c)
Legenda: cv – controle de vizinhança		

Figura 6.3: Fluxo de tarefas dos PAs (a), do PC (b) e da ME (c) em SegImag.

Associado a cada tarefa t_i da Figura 6.3 está o seu instante de início t_i. Estes instantes representam as necessidades de comunicação e computação de cada PE:

1. t_1, t_{12} e t_{15} são os tempos iniciais de PA, PC e ME, respectivamente;
2. Para os PAs, t_2 é determinado pelo tempo de comunicação com a ME. t_3 depende da computação de cada PA. t_4 é igual a t_3 quando não há PA vizinho esquerdo ou dependente do tempo para enviar dados para o PA esquerdo. De forma semelhante a t_4, t_5 é igual a t_4 quando não há PA vizinho acima ou dependente do tempo para enviar dados para o PA acima. t_6, t_7 e t_8 dependem da posição da PA, que pode ou não receber

vizinhanças. t_9 é determinado pelo tempo de envio de todas as comunicações de cada PA para o PC e do tempo de processamento do PC. t_{10} é determinado pelo tempo de comunicação com o PC. Por último, t_{11} depende do tempo de computação de cada PA;

3. Para o PC, t_{13} é dependente do tempo que todas as PAs levam para enviar seus dados. Enquanto que t_{14} depende apenas da computação do PC;
4. Para a ME, t_{16} é dependente do tempo que todas as PAs levam para enviar seus dados.

Para todos os formatos internos apresentados no Capítulo 5, os PAs, a ME e o PC da aplicação SegImag foram associados a núcleos $N = \{PA_0, PA_1, PA_2, PA_3, PC, ME\}$. Para aumentar a clareza na comparação entre os modelos, nos formatos internos está sendo representado apenas o número de bytes das mensagens.

6.1 Aplicação de Segmentação de Imagens Modelada com CWM

Para o modelo CWM, as comunicações são modeladas pela quantidade total de bytes enviada de um núcleo para outro:

1. Entre a ME e um PA e vice-versa (n_{BS}): 76800;
2. Entre um PA e seu vizinho esquerdo (n_{BPY}): 240;
3. Entre um PA e seu vizinho acima (n_{BPX}): 320; e
4. Entre um PA e o PC e vice-versa (n_{CV}): 128.

Estas comunicações, representadas na Figura 6.4, geram um CWG = <*N*, *W*>, onde:

$N = \{PA_0, PA_1, PA_2, PA_3, PC, ME\}$

$W = \{$
$W_{PA0,ME} = 76800$ $W_{PA0,PC} = 128$ $W_{PA1,ME} = 76800$ $W_{PA1,PC} = 128$
$W_{PA1,PA0} = 320$ $W_{PA2,ME} = 76800$ $W_{PA2,PC} = 128$ $W_{PA2,PA0} = 320$
$W_{PA3,ME} = 76800$ $W_{PA3,PC} = 128$ $W_{PA3,PA1} = 320$ $W_{PA3,PA2} = 240$
$W_{PC,PA0} = 128$ $W_{PC,PA1} = 128$ $W_{PC,PA2} = 128$ $W_{PC,PA3} = 128$
$W_{ME,PA0} = 76800$ $W_{ME,PA1} = 76800$ $W_{ME,PA2} = 76800$ $W_{ME,PA3} = 76800$
$\}$

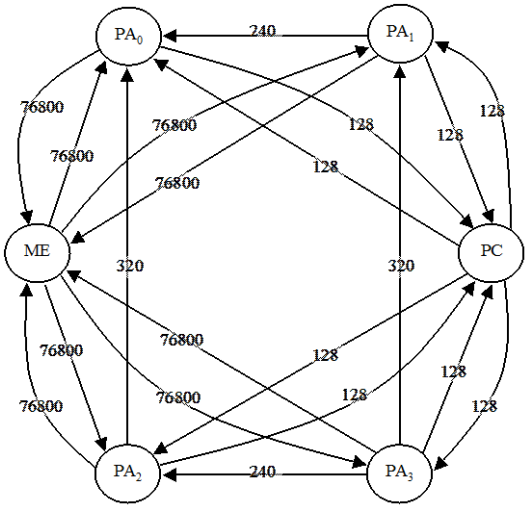


Figura 6.4: CWG da aplicação SegImag.

6.2 Aplicação de Segmentação de Imagens Modelada com ECWM

O modelo ECWM acrescenta ao CWM as transições das comunicações. Como o objetivo deste Capítulo é apenas apresentar a modelagem da aplicação de segmentação de imagens descrita na Figura 6.2, utilizam-se aqui valores hipotéticos para cada conjunto de transições. No caso, para os bits de controle usa-se 80% de transições, enquanto que para os bits de dados usa-se 20% de transições. Com estes percentuais, obtêm-se os seguintes valores de transições:

1. Entre a ME e um PA e vice-versa: 15360;
2. Entre um PA e seu vizinho esquerdo: 192;
3. Entre um PA e seu vizinho acima: 256; e
4. Entre um PA e o PC e vice-versa: 103.

A Figura 6.5 apresenta o grafo ECWG = <*N*, *H*> que contém as comunicações com as respectivas quantidades de bytes transmitidos e a quantidade de transições. Onde:

$N = \{PA_0, PA_1, PA_2, PA_3, PC, ME\}$

$H = \{$
$H_{PA0,ME} = 76800, 15360$ $H_{PA0,PC} = 128, 103$ $H_{PA1,ME} = 76800, 15360$
$H_{PA1,PC} = 128, 103$ $H_{PA1,PA0} = 240, 192$ $H_{PA2,ME} = 76800, 15360$
$H_{PA2,PC} = 128, 103$ $H_{PA2,PA0} = 320, 256$ $H_{PA3,ME} = 76800, 15360$
$H_{PA3,PC} = 128, 103$ $H_{PA3,PA1} = 320, 256$ $H_{PA3,PA2} = 240, 192$
$H_{PC,PA0} = 128, 103$ $H_{PC,PA1} = 128, 103$ $H_{PC,PA2} = 128, 103$
$H_{PC,PA3} = 128, 103$ $H_{ME,PA0} = 76800, 15360$ $H_{ME,PA1} = 76800, 15360$
$H_{ME,PA2} = 76800, 15360$ $H_{ME,PA3} = 76800, 15360$
$\}$

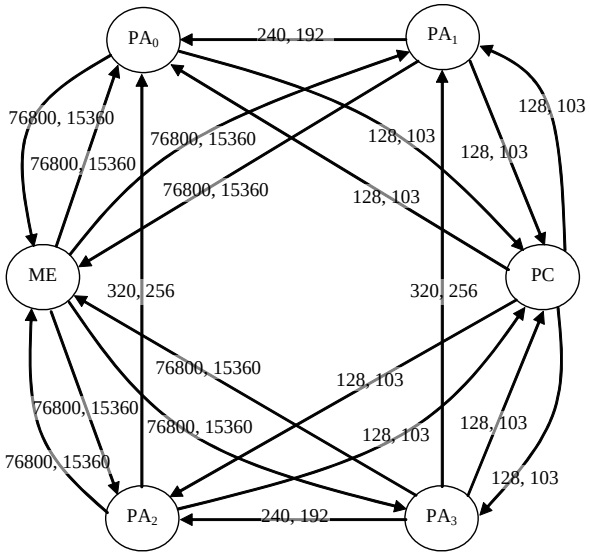


Figura 6.5: ECWG da aplicação SegImag.

6.3 Aplicação de Segmentação de Imagens Modelada com CDM

O modelo CDM, por sua vez, requer a descrição de todas as mensagens e a dependência entre estas. Embora a descrição da aplicação seja seqüencial, muitas mensagens podem ser transmitidas concorrentemente, tal como descrito na Figura 6.6.

O aumento da complexidade dos formatos internos CWG e ECWG para o CDG é notado com o aumento do número de vértices de 6 para 22, e o aumento do número de arestas de 20 para 37. Isto ocorre porque os vértices deixam de representar núcleos para representar mensagens, e as arestas deixam de representar a comunicação entre núcleos para representar a dependência entre mensagens. A Figura 6.6 ilustra um CDG = <*M*, *D*>, onde:

M = {m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}, m_{16}, m_{17}, m_{18}, m_{19}, m_{20}}

m_1 = (1, ME, PA_0, 76800) m_2 = (2, ME, PA_1, 76800) m_3 = (3, ME, PA_2, 76800)
m_4 = (4, ME, PA_3, 76800) m_5 = (5, PA_1, PA_0, 240) m_6 = (6, PA_1, PA_0, 320)
m_7 = (7, PA_3, PA_2, 240) m_8 = (8, PA_3, PA_1, 320) m_9 = (9, PA_0, PC, 128)
m_{10} = (10, PA_1, PC, 128) m_{11} = (11, PA_2, PC, 128) m_{12} = (12, PA_3, PC, 128)
m_{13} = (13, PC, PA_0, 128) m_{14} = (14, PC, PA_1, 128) m_{15} = (15, PC, PA_2, 128)
m_{16} = (16, PC, PA_3, 128) m_{17} = (17, PA_0, ME, 76800) m_{18} = (18, PA_1, ME, 76800)
m_{19} = (19, PA_2, ME, 76800) m_{20} = (20, PA_3, ME, 76800)

D = { (m_1, INÍCIO), (m_2, INÍCIO), (m_3, INÍCIO), (m_4, INÍCIO), (m_5, m_2), (m_6, m_3), (m_7, m_4),
(m_8, m_7), (m_9, m_1), (m_9, m_5), (m_{10}, m_6), (m_{11}, m_7), (m_{12}, m_8), (m_{13}, m_9),
(m_{13}, m_{10}), (m_{13}, m_{11}), (m_{13}, m_{12}), (m_{14}, m_9), (m_{14}, m_{10}), (m_{14}, m_{11}), (m_{14}, m_{12}),
(m_{15}, m_9), (m_{15}, m_{10}), (m_{15}, m_{11}), (m_{15}, m_{12}), (m_{16}, m_9), (m_{16}, m_{10}), (m_{16}, m_{11}),
(m_{16}, m_{12}), (m_{17}, m_{13}), (m_{18}, m_{14}), (m_{19}, m_{15}), (m_{20}, m_{16}), (FIM, m_{17}), (FIM, m_{18}),
(FIM, m_{19}), (FIM, m_{20})
}

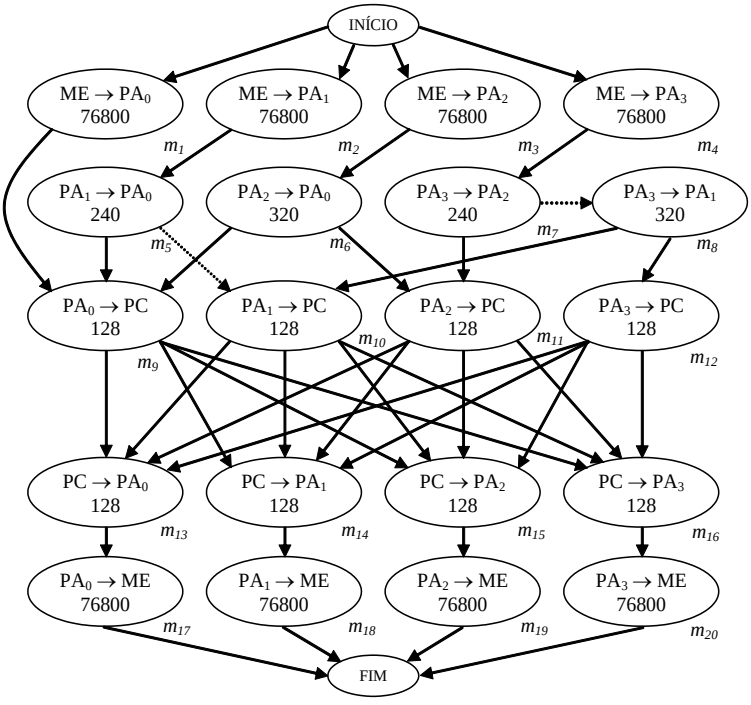


Figura 6.6: CDG da aplicação SegImag.

As arestas (m_{10}, m_5) e (m_8, m_7), representadas na Figura 6.6 por flechas tracejadas, representam dependências intranúcleo. A dependência (m_8, m_7), por exemplo, reflete a precedência da tarefa t_3 sobre a tarefa t_4, ilustrada na Figura 6.3(a). Neste caso, m_8 é liberada para ser enviada na infra-estrutura de comunicação tão logo m_7 for transmitida, não necessitando esta última chegar ao seu destino.

6.4 Aplicação de Segmentação de Imagens Modelada com CDCM

O CDCG da aplicação SegImag tem o mesmo número de vértices que o CDG. A diferença está no acréscimo do tempo de computação anterior à geração de cada mensagem. Para estimar o tempo de computação, supõe-se que as memórias sempre estejam preenchidas antes de começar a execução da aplicação e que os PAs são mais rápidos que o PC.

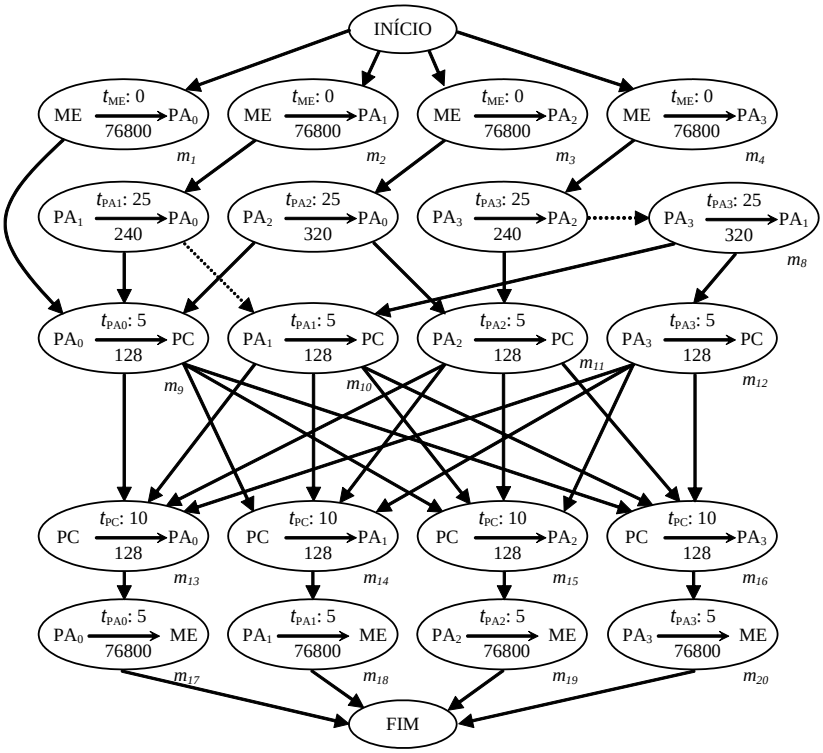


Figura 6.7: CDCG da aplicação SegImag.

A Figura 6.7 ilustra o CDCG = <*M*, *D*>, onde *D* é igual ao apresentado no CDG e *M* é tem os seguintes elementos:

m_1 = (1, ME, PA_0, 0, 76800) m_2 = (2, ME, PA_1, 0, 76800) m_3 = (3, ME, PA_2, 0, 76800)
m_4 = (4, ME, PA_3, 0, 76800) m_5 = (5, PA_1, PA_0, 25, 240) m_6 = (6, PA_1, PA_0, 25, 320)
m_7 = (7, PA_3, PA_2, 25, 240) m_8 = (8, PA_3, PA_1, 25, 320) m_9 = (9, PA_0, PC, 5, 128)
m_{10} = (10, PA_1, PC, 5, 128) m_{11} = (11, PA_2, PC, 5, 128) m_{12} = (12, PA_3, PC, 5, 128)
m_{13} = (13, PC, PA_0, 10 128) m_{14} = (14, PC, PA_1, 10 128) m_{15} = (15, PC, PA_2, 10 128)
m_{16} = (16, PC, PA_3, 10 128) m_{17} = (17, PA_0, ME, 5, 76800) m_{18} = (18, PA_1, ME, 5, 76800)
m_{19} = (19, PA_2, ME, 5, 76800) m_{20} = (20, PA_3, ME, 5, 76800)

6.5 Aplicação de Segmentação de Imagens Modelada com ACPM

A estrutura ACP, que implementa o ACPM da aplicação, descreve um conjunto de mensagens totalmente ordenado. O tempo descrito no ACP não representa necessariamente o tempo físico, apenas marca uma ordem de envio de mensagens. A Figura 6.8 ilustra um ACP, onde o conjunto de mensagens *M* é igual ao conjunto de mensagens do CDM, e a lista é dada por:

ACP = { (1, {m_1}), (2, {m_2}), (3, {m_3}), (4, {m_4}),
(5, {m_5, m_6, m_7}), (6, {m_8}), (7, {m_9, m_{10}, m_{11}, m_{12}}), (8, {m_{13}}),
(9, {m_{14}}), (10, {m_{15}}), (11, {m_{16}}), (12, {m_{17}, m_{18}, m_{19}, m_{20}})
}

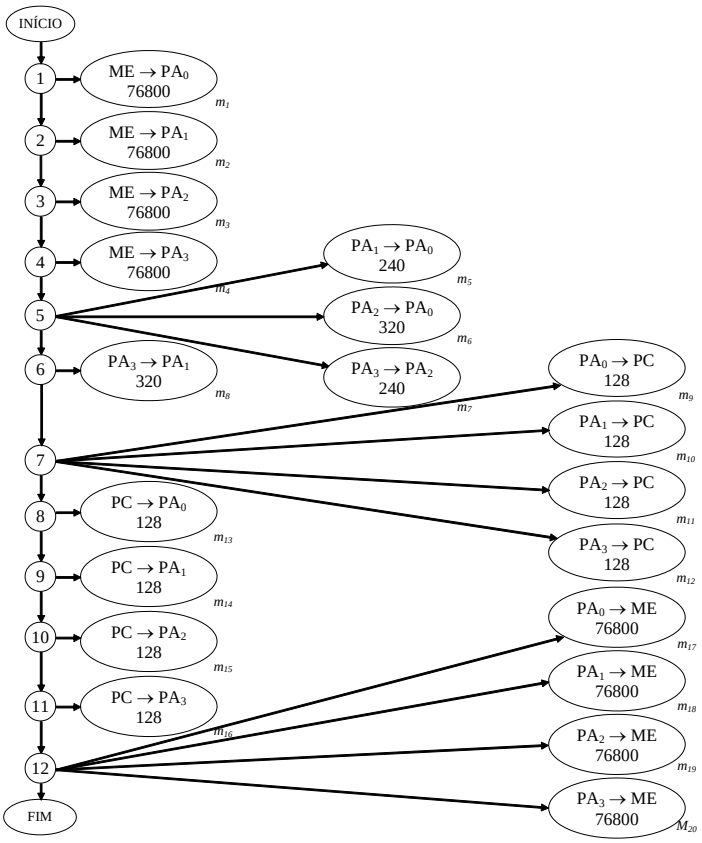


Figura 6.8: ACP da aplicação SegImag.

6.6 Aplicação de Segmentação de Imagens Modelada com CTM

O CTM é o modelo mais complexo dos apresentados neste Capítulo. Este modelo envolve muitas variáveis que devem ser estimadas. Entre elas, os deadlines das tarefas, o consumo de energia e tempo de execução das tarefas quando executadas sobre um determinado PE. Além do mais, este modelo também é adequado para avaliar a possibilidade de particionar as tarefas em PEs, e na aplicação SegImag as partições já estão definidas.

Para implementar os PAs ou o PC supõe-se a existência de dois processadores distintos. Supõe-se também a existência de dois tipos de memórias para implementar a ME. Para estimar o deadline das tarefas, o problema parte da especificação que determina a análise de 15 quadros de 640×480 bytes por segundo, implicando que o deadline da aplicação deve ser 66.66 milisegundos. Este deadline deve ser distribuído eqüitativamente dentro do tempo de execução de cada tarefa. O Conjunto de tarefas e os conjuntos de dependências para a implementação dos PAs, PC e ME estão descritos a seguir:

$T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}\}$

t_1 = (7, [4, 9], [48, 36]) t_2 = (25, [19, 41], [231, 173]) t_3 = (5, [3, 5], [34, 22])
t_4 = (5, [3, 6], [33, 25]) t_5 = (4, [4, 8], [36, 26]) t_6 = (4, [4, 8], [36, 26])
t_7 = (5, [3, 6], [32, 24]) t_8 = (3, [1, 2], [12, 8]) t_9 = (3, [1, 2], [12, 8])
t_{10} = (3, [8, 17], [96, 64]) t_{11} = (10, [4, 8], [48, 37]) t_{12} = (15, [12, 23], [144, 91])
t_{13} = (8, [5, 11], [62, 46]) t_{14} = (15, [12, 23], [144, 91]) t_{15} = (8, [5, 11], [40, 36])
t_{16} = (8, [5, 10], [41, 30])

C_{PA} = { (INÍCIO → t_1, 0), (t_1 → t_2, 0), (t_2 → t_3, 0), (t_2 → t_4, 0), (t_2 → t_5, 0),
(t_2 → t_6, 0), (t_2 → t_7, 0), (t_5 → t_7, 0), (t_6 → t_7, 0), (t_3 → t_4, 240),
(t_3 → t_7, 240), (t_4 → t_7, 320), (t_7 → t_8, 128), (t_8 → t_9, 0), (t_9 → t_{10}, 0),
(t_{10} → t_{11}, 0) (t_{11} → FIM, 76800)
}

C_{PC} = { (INÍCIO → t_{12}, 0), (t_{12} → t_{13}, 0), (t_{13} → t_{14}, 128 × 4), (t_{14} → FIM, 0) }

C_{ME} = { (INÍCIO → t_{15}, 0), (t_{15} → t_{16}, 76800 × 4), (t_{16} → FIM, 0) }

A Figura 6.9 mostra os CTGs para PAs (a), PC (b) e ME (c).

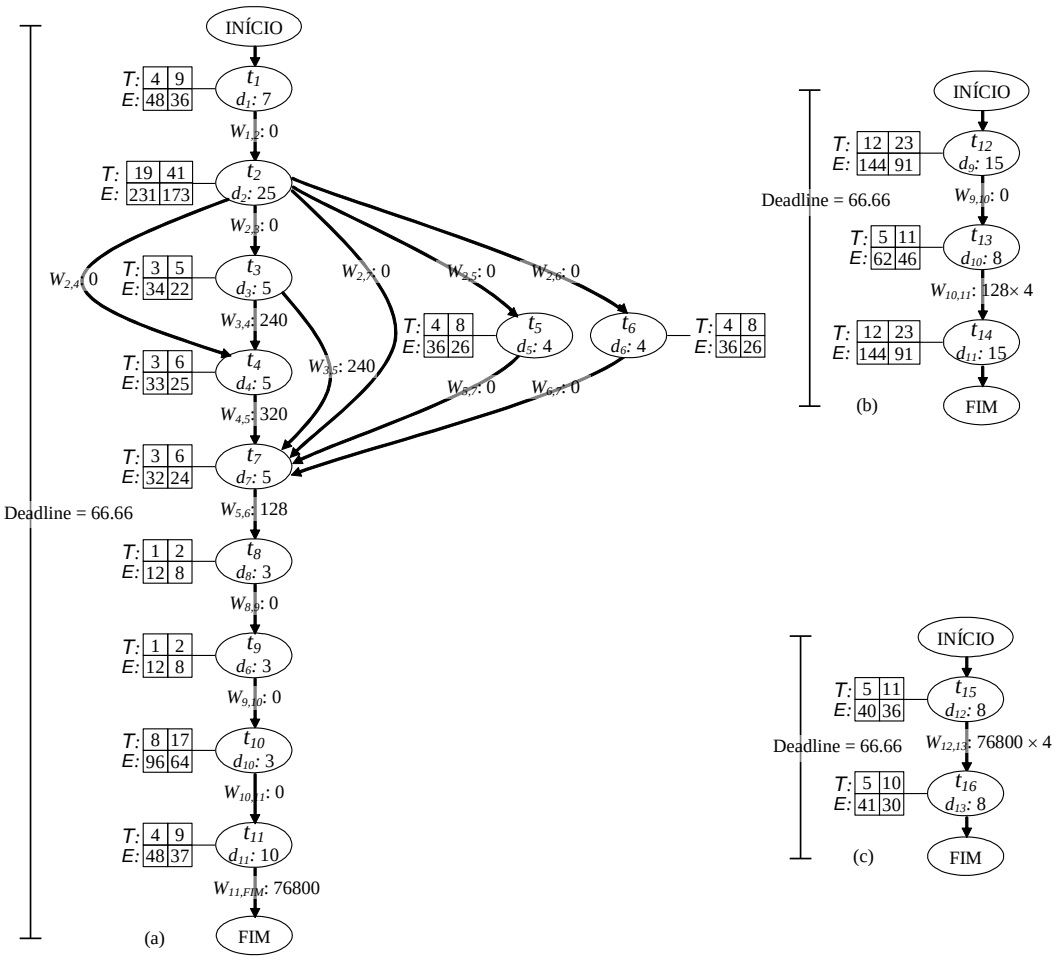


Figura 6.9: CTG da aplicação SegImag.

7 METAMODELO QUANTIDADE - ORDEM - DEPENDÊNCIA (QOD)

Keutzer et al. (2000) descrevem que a separação de certos aspectos da aplicação permite a exploração mais eficiente do espaço de soluções de projeto. O sucesso deste processo depende das características da aplicação, dos requisitos de projeto e da escolha adequada dos aspectos a avaliar. Analisando corretamente estes elementos, é possível abstrair detalhes de um aspecto sem interferir na avaliação eficiente dos efeitos que os demais aspectos causam no projeto. Em conseqüência desta abstração, os projetos podem ser realizados com menor complexidade ampliando a exploração de soluções de qualidade que atendam os requisitos de projeto. Entres estes aspectos dois que se destacam no projeto de sistemas computacionais são a computação e a comunicação, pois permitem especificar diversas informações, presentes em várias etapas do projeto destes sistemas. Ambos são aspectos que refletem formas distintas da aplicação processar informações. Enquanto a comunicação refere-se à troca de informação entre elementos de um sistema ou entre o sistema e o meio, a computação refere-se à transformação da informação internamente a um elemento. O estudo realizado neste trabalho, propõe modelos que partem dos aspectos comunicação e/ou computação. Além destes aspectos, outros são considerados, gerando novas propostas de modelos mais abrangentes A composição de modelos propostos e sua relação com os aspectos da aplicação estão ilustradas na Figura 7.1.

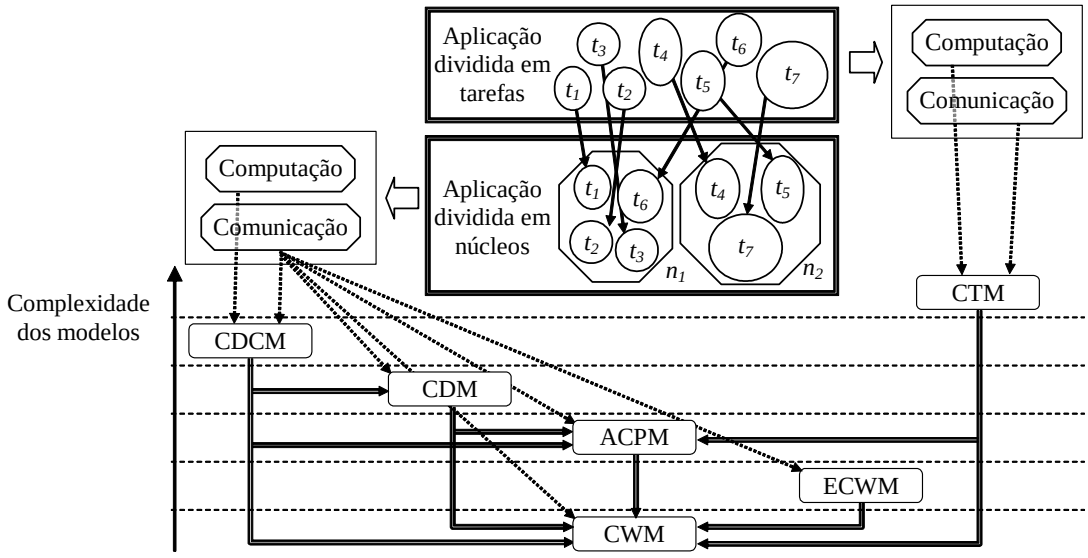


Figura 7.1: Composição de modelos a partir da computação e/ou comunicação da aplicação e a partir de outros modelos de maior complexidade. A computação e a comunicação são extraídas da aplicação descrita por tarefas ou por núcleos. O CTM é extraído das tarefas da aplicação,

enquanto que os demais modelos são extraídos dos núcleos da aplicação. Flechas simples contínuas mostram o mapeamento de tarefas em núcleos, flechas vazadas mostram a extração dos aspectos computação e comunicação, flechas tracejadas indicam a composição da computação ou comunicação e flechas duplas contínuas ilustram a extração de um modelo a partir de outro mais complexo.

A Figura 7.1 apresenta também a ilustração da complexidade relativa entre os modelos. Como podem ser observados, cinco modelos são extraídos da aplicação descrita com núcleos (CDCM, CDM, ECWM, CWM e ACPM) enquanto que apenas um é extraído da aplicação descrita em tarefas (CTM). O CWM pode ser obtido a partir de qualquer modelo mais complexo aqui descrito, o CDM pode ser extraído do CDCM, pela abstração da comunicação. O ACPM pode ser extraído dos modelos CDM, CDCM e CTM. Cabe aqui salientar que o resultado desta extração não é unívoco, pois como o ACPM modela a ordem total da comunicação, que é *um* caso da ordem parcial das comunicações, mais de um ACPM pode ser capturado de qualquer um dos modelos CDM, CDCM e CTM. Os demais modelos não podem ser obtidos de outros.

Para a atividade de mapeamento, a abstração da computação pode auxiliar na redução da complexidade, permitindo que o projetista utilize modelos e algoritmos mais simples. Por outro lado, para certos requisitos de projeto, esta abstração pode levar a mapeamentos de pior qualidade, quando comparados com mapeamentos obtidos de modelos que consideram ambos os aspectos. Um exemplo é o requisito minimização do consumo de energia. Caso se considere apenas a energia dinâmica devido ao tráfego na infra-estrutura de comunicação, o tempo em que ocorrem as comunicações pode ser abstraído, pois o que importa é o número de transistores chaveando e não os momentos que ocorrem estes chaveamentos. Assim, o tempo de computação de cada núcleo, que permite estimar o tempo de execução da aplicação, pode ser desconsiderado. Por outro lado, se o consumo de energia estática de todo o circuito, ou o consumo de energia dinâmica dos circuitos que permanecem operando mesmo na ausência de tráfego fizerem parte dos requisitos de projeto, então as informações de tempo se tornam imprescindíveis, pois estes consumos são diretamente proporcionais ao tempo de execução da aplicação. Neste caso, a computação da aplicação não pode ser abstraída.

Definem-se aqui quantidade e ordem como critérios adicionais para avaliar a aplicação. A *quantidade* é um critério expresso por um valor, tal como o número de bits de uma mensagem e o número de segundos necessário para executar uma tarefa; A *ordem* é um critério expresso por relações entre elementos de conjuntos. A ordem é total, quando são associadas informações que determinem a seqüência precisa de elementos, tal como a associação de marcas de tempo aos elementos de um conjunto, ou parcial, quando existir uma ordem relativa entre estes, que admite várias seqüências distintas dos elementos.

Os critérios de avaliação ordem e quantidade são ortogonais, enquanto que os critérios ordem parcial e ordem total não o são. Contudo, devido à influência que estes dois últimos critérios têm na modelagem de aplicações, são aqui tratados distintamente. Aqui, denomina-se *ordem total* apenas *ordem* e *ordem parcial*, *dependência*. Assim, a ordem é um critério que pode ser modelado por eventos (MARCON, 2001), presentes nos modelos DE. A dependência, por sua vez, é um critério que determina uma ordem parcial pela associação de compromissos entre elementos, tal como um grafo de escalonamento que descreve relações de ordem parciais entre tarefas. Para este exemplo, cada tarefa é um vértice do grafo e as arestas dirigidas determinam a relação entre os vértices, ou seja, os compromissos de ordem, tal como uma tarefa t_1 somente

pode ser executada quando uma informação for processada na tarefa t_2, da qual t_1 é dependente. Sabe-se qual tarefa precede a outra, mas não o instante preciso de sua execução. Várias tarefas independentes podem ser executadas de forma concorrente ou em diferentes seqüências.

A Figura 7.2 apresenta os critérios quantidade, ordem e dependência que permitem avaliar aplicações juntamente com aos aspectos comunicação e computação.

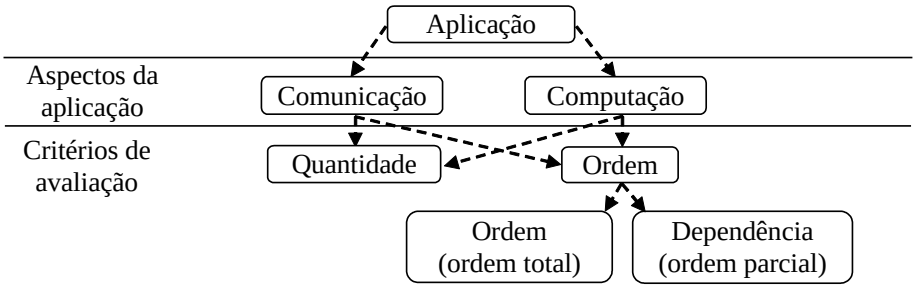


Figura 7.2: Critérios de avaliação (quantidade, ordem e dependência) e suas relações com os aspectos da aplicação (comunicação e computação).

Os critérios de avaliação da computação e da comunicação compõem bases para criação de modelos computacionais utilizados na atividade de mapeamento. Estes modelos são aqui construídos a partir do metamodelo denominado QOD (Quantidade, Ordem e Dependência). Este metamodelo relaciona critérios com aspectos, permitindo relacionar a complexidade dos modelos gerados. Considera-se que os modelos mais simples são aqueles que contêm apenas informações de quantidade, enquanto que os mais complexos são aqueles que contêm informações de dependência. Casos intermediários são os modelos que contêm informações de ordem. A relação de complexidade entre os critérios é assim dada, pois enquanto a quantidade é expressa por um valor, a ordem e a dependência descrevem relações; e a dependência é um critério que engloba todas as possíveis ordens.

O metamodelo QOD é construído basicamente sobre o eixo estrutural do diagrama Y (GAJSKI; KUHN, 1983). Assim, pouca informação comportamental é inserida nos modelos criados a partir deste, e em conseqüência desta abstração, a complexidade para o tratamento dos formatos internos construídos a partir dos modelos é bastante simplificada e suficiente para as estimativas de tempo de comunicação e consumo de energia propostas neste trabalho.

A elaboração do metamodelo é útil para que o projetista escolha adequadamente o modelo que melhor atende o projeto em vista. Entre as principais vantagens do entendimento do modelo QOD estão:

1. Classificação de modelos existentes e de novos a serem elaborados - Esta classificação é útil para compreensão dos fundamentos de cada modelo;
2. Avaliação da complexidade de uma classe de modelos - A noção da complexidade de cada classe de modelos permite ao projetista estimar o esforço gasto para modelar a aplicação de acordo com o modelo escolhido;
3. Avaliar a possibilidade de estender novos modelos - O metamodelo serve como base para a criação de novos modelos e também para estender os mesmos, inserindo novos aspectos e/ou critérios que aumentem a eficácia destes. Entre as possibilidades de extensão está a criação de modelos hierárquicos versus não hierárquicos. A hierarquia neste caso poderia ser

um quarto critério ortogonal aos demais, estendendo assim o próprio metamodelo;

4. Avaliação das potencialidades de cada classe de modelos - De acordo com a classe de modelos, certos requisitos são melhor atendidos. Exemplos são as classes que quantificam a comunicação, adequadas para o cálculo da energia dinâmica da infra-estrutura de comunicação; as classes que modelam a ordem da comunicação, adequadas para o cálculo do tempo de comunicação; e as classes que modelam a ordem da comunicação e quantificam a computação, adequadas para o cálculo do tempo de execução de uma aplicação. Trabalhos que visam este objetivo foram apresentados em (MARCON et al., 2005c).

A construção de modelos a partir do metamodelo QOD está ilustrada nas cinco Figuras que seguem. A Figura 7.3 mostra que frente aos critérios e aspectos escolhidos são definidas seis *classes simples*[30] *e homogêneas*[31] que são: quantidade de comunicação, ordem da comunicação, dependência da comunicação, quantidade da computação, ordem da computação e dependência da comunicação.

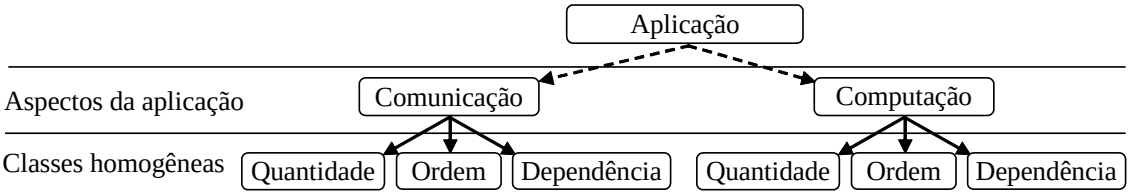


Figura 7.3: Classes de modelos simples e homogêneos derivados dos aspectos comunicação e computação.

A Figura 7.4 ilustra como os modelos homogêneos e simples apresentados neste trabalho são classificados. Note que os modelos *core graph* (MURALI; DE MICHELI, 2004a) e *Application Characterization Graph* (HU; MARCULESCU, 2003) estão agrupados, pois fazem parte da classe de modelos tipo CWM.

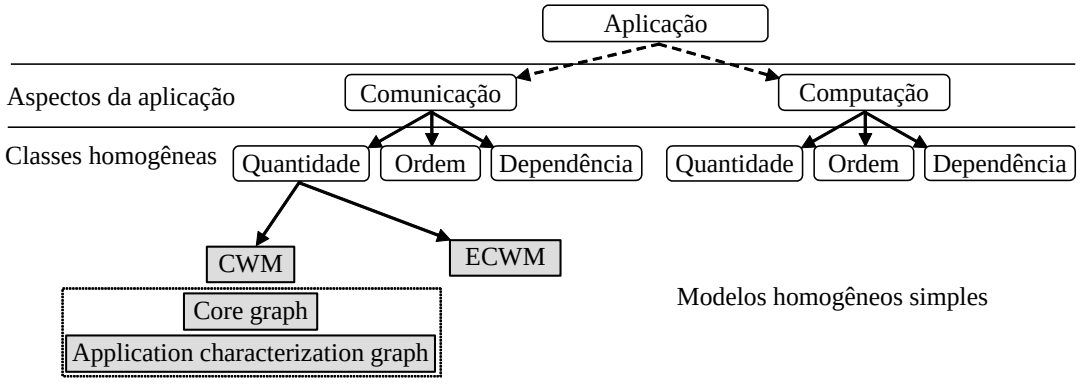


Figura 7.4: Classificação de modelos simples e homogêneos, descritos neste trabalho, frente aos critérios quantidade, ordem e dependência e frente

[30] Aqui se denomina *classe simples* aquelas derivadas de apenas um critério de avaliação, em oposição às *classes compostas*, derivadas de mais de um critério, tal como quantidade e ordem.

[31] *Classes homogêneas* são aquelas derivadas de um único aspecto da aplicação, podendo ser simples ou compostas, em oposição às *classes heterogêneas*, derivadas de mais de um aspecto da aplicação.

aos aspectos comunicação e computação.

A Figura 7.4 ilustra que este trabalho não apresenta modelos homogêneos derivados da computação. Isto ocorre porque estes modelos não são adequados para a atividade de mapeamento de aplicações em infra-estruturas de comunicação, muito embora possam ser objeto de pesquisa em atividades de particionamento de tarefas em núcleos. Como um dos principais objetivos deste trabalho é a avaliação do consumo de energia em infra-estruturas de comunicação, e este está diretamente relacionado com a quantidade de bits que trafega no meio físico, a quantificação da comunicação está presente em todos os modelos aqui estudados.

Certas classes de modelos simples podem não ser suficientes para modelar uma aplicação frente ao problema em vista. A associação de mais de um critério a um mesmo aspecto aumenta a capacidade de expressão do modelo e conseqüentemente a sua complexidade. Como a dependência e ordem são critérios mutuamente exclusivos, não existem modelos compostos pela combinação destes, restando apenas a composição de quantidade com ordem ou dependência para ambos os aspectos. Esta composição está ilustrada na Figura 7.5, que mostra a criação de quatro novas classes de modelos, as classes homogêneas compostas.

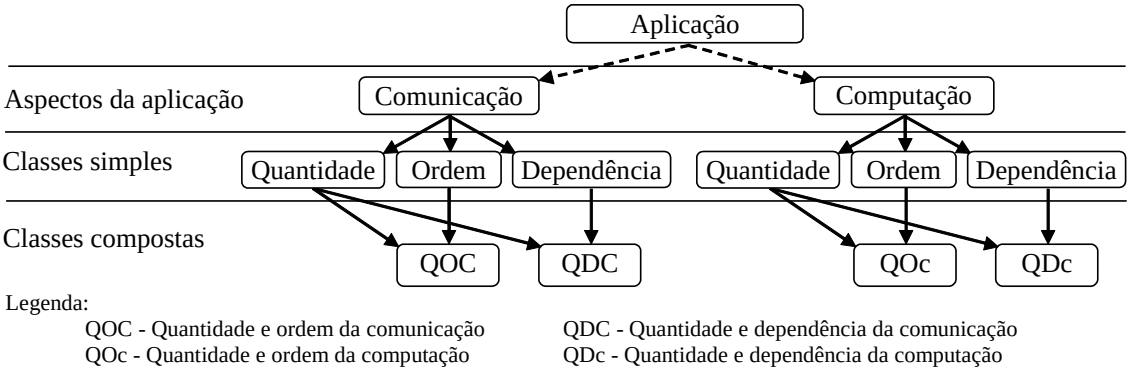


Figura 7.5: Representação de todas as classes puras e compostas geradas a partir da combinação dos critérios quantidade e ordem para cada aspecto da aplicação.

A Figura 7.6 ilustra os modelos ACPM e CDM, apresentados neste trabalho, que fazem parte das classes homogêneas compostas QOC e QDC, respectivamente.

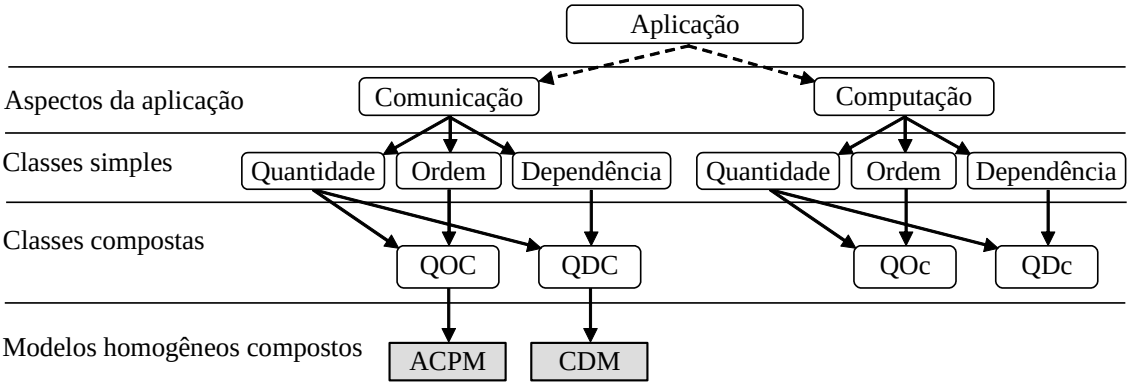
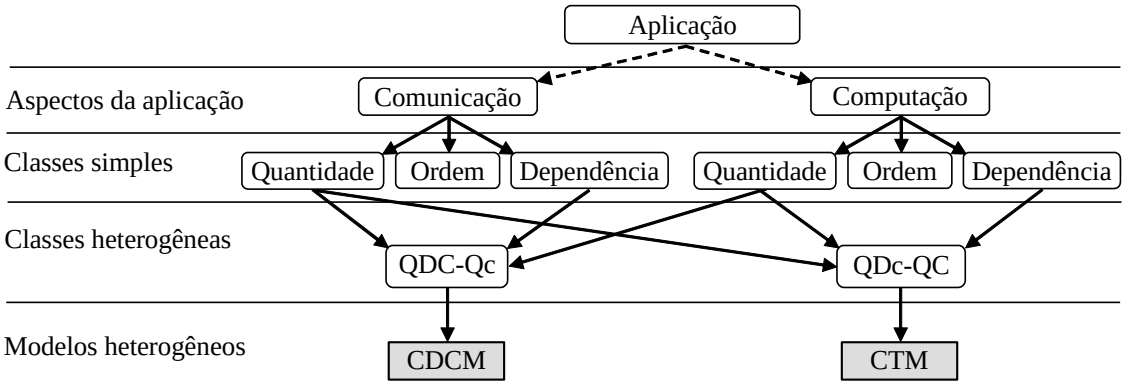


Figura 7.6: Classificação de modelos homogêneos compostos frente aos critérios quantidade, ordem e dependência e os aspectos comunicação e computação.

Modelos mais complexos são criados a partir da combinação de critérios com

mais de um aspecto. Estes modelos são chamados aqui de heterogêneos. Em teoria existem diversos modelos heterogêneos que podem ser obtidos pelas combinações de critérios e aspectos, muito embora boa parte destes possa não ser adequada para uma dada atividade, como o mapeamento de núcleos em infra-estruturas de comunicação. Este trabalho apresenta dois modelos heterogêneos: CDCM e CTM, que são classificados no metamodelo QOD de acordo com a Figura 7.7.



Legenda:
QDC-Qc - Quantidade e dependência da comunicação com quantidade da computação
QDc-QC - Quantidade e dependência da computação com quantidade da comunicação

Figura 7.7: Representação de modelos heterogêneos.

8 MODELOS DE CONSUMO DE ENERGIA E TEMPO DE COMUNICAÇÃO PARA REDES INTRACHIP

A atividade de mapeamento tem sua função objetivo fortemente dependente da infra-estrutura de comunicação, onde é necessário modelar tanto informações tecnológicas quanto topológicas. Como estudo de caso para avaliação da qualidade dos mapeamentos, este trabalho emprega algumas classes de NoCs como infra-estrutura de comunicação. Outras infra-estruturas podem ser utilizadas, tais como barramento segmentado e conexões dedicadas ponto a ponto. A escolha de NoCs dá-se pela sua regularidade e escalabilidade, requisitos muito importantes para lidar com a crescente complexidade de projetos de sistemas computacionais.

Este Capítulo propõe modelos para estimar latência e consumo de energia em NoCs síncronas. Primeiro, a Seção 8.1 apresenta um modelo para estimar o tempo da comunicação para uma classe específica de NoCs, a qual pertence o estudo de caso usado neste trabalho. Esta classe compreende NoCs com topologia malha, roteamento XY e chaveamento wormhole. Escolheu-se esta topologia, por ser a mais utilizada em redes intrachip. A seguir, a Seção 8.2 apresenta modelos para estimar o consumo de energia de NoC malha, e estende o trabalho para mais duas topologias de NoC referenciadas em diversos trabalhos (toro dobrado e árvore gorda).

8.1 Modelo de Tempo de Comunicação para NoCs Malha

O modo de chaveamento wormhole, tal como descrito na Seção 3.2.1, divide os pacotes de cada mensagem em flits. O flit de cabeçalho possui informação para permitir determinar o caminho até o roteador destino e, uma vez criado o caminho, todos os demais flits do pacote seguem por ele. Neste trabalho, assume-se NoCs com flits e phits de mesmo tamanho. Como os últimos são uma informação física da rede, serão doravante preferencialmente utilizados para a formulação dos modelos de tempo de comunicação e consumo de energia.

Para construir um modelo de tempo de comunicação, o *atraso total de um pacote* para o modo de chaveamento wormhole é decomposto em atraso de roteamento e atraso de carga útil. O *atraso de roteamento* é definido como o tempo necessário para que o phit de cabeçalho vá da origem ao destino. O *atraso de carga útil* depende do número de phits restante, e é definido como o tempo transcorrido entre a chegada do phit de cabeçalho ao destino e a chegada do último phit do pacote ao destino. Como XY é um algoritmo determinístico mínimo, que permite definir com precisão qual o caminho por onde o pacote irá passar, é possível definir η como o número de roteadores e η+1 o número de conexões (entre núcleos e roteadores e entre roteadores) por onde o pacote irá passar. Seja λ o período do relógio, t_R o número de ciclos necessários para a

decisão de chaveamento em um roteador, t_L o número de ciclos necessários para transmitir um phit entre roteadores e t_l o número de ciclos necessários para transmitir um phit entre roteador e núcleo local. Considerando-se ainda uma rede com layout regular, onde todos os tiles utilizem relógios com o mesmo período e fase. O atraso de roteamento (d_{Rij}) de um pacote que parte do tile τ_i indo até o tile τ_j, considerando que o caminho envolve η roteadores e que não há contenção de pacotes é representado pela Equação 8.1 (MARCON et al., 2005a).

Equação 8.1: $d_{Rij} = (\eta \times t_R + 2 \times t_l + (\eta - 1) \times t_L) \times \lambda$

Contenções não são expressas na Equação 8.1, pois elas somente podem ser determinadas durante a execução da aplicação, ou estimadas por modelagem probabilística não considerada aqui. Embora os tempos de propagação entre roteadores e entre roteador e núcleo local possam ser diferentes, para todas as implementações de NoCs conhecidas, o número de ciclos é igual. Assim, assume-se t_l igual à t_L, de forma que a Equação 8.1 pode ser reescrita na Equação 8.2.

Equação 8.2: $d_{Rij} = (\eta \times t_R + (\eta + 1) \times t_L) \times \lambda$

Seja n_{Fabq} o número de phits do pacote de índice q, que parte do núcleo n_a para o núcleo n_b, obtido a partir do número de bits de cada mensagem (w_{abq}) e das características da infra-estrutura de comunicação (número de pacotes por mensagem e o número de bits extra de controle dos pacotes). De forma análoga ao atraso de roteamento, define-se ($d_{Pij,abq}$) como o atraso de carga útil do pacote de índice q, que parte do núcleo n_a mapeado no tile τ_i indo até núcleo n_b mapeado no tile τ_j. Considerando-se que o caminho envolve η roteadores e que não há contenção de pacotes, $d_{Pij,abq}$ é dado pela Equação 8.3.

Equação 8.3: $d_{Pij,abq} = ((n_{Fabq} - 1) \times t_L) \times \lambda$

Compondo a Equação 8.2 com a Equação 8.3, o atraso total do pacote de índice q, que parte do núcleo n_a, mapeado no tile τ_i, indo até o núcleo n_b, mapeado no tile τ_j, ($d_{ij,abq}$) é obtido pela soma de d_{Rij} com $d_{Pij,abq}$. Este é dado pela Equação 8.4.

Equação 8.4: $d_{ij,abq} = (\eta \times (t_R + t_L) + n_{Fabq} \times t_L) \times \lambda$

Para exemplificar o uso da Equação 8.4, considere a NoC 2 × 2 ilustrada na Figura 8.1. Para esta NoC, o atraso total do quinto pacote que parte do núcleo n_2 mapeado no tile τ_1, e vai até o núcleo n_3 mapeado no tile τ_4, considerando $t_R = 2$, $t_L = 1$, $\lambda = 10$ ns e $n_{235} = 20$ phits, é: $d_{14,235} = (3 \times (2 + 1) + 20 \times 1) \times 10$ ns = 290 ns.

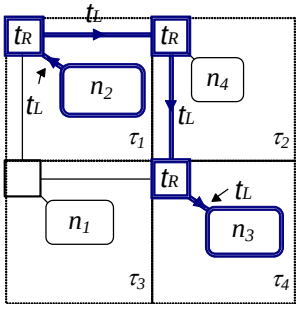


Figura 8.1: Modelagem do tempo de comunicação para enviar um pacote através de

uma NoC 2×2 com topologia malha, roteamento XY e chaveamento wormhole. São destacados os caminhos e tempos associados (t_R, t_L) para um pacote ser enviado do núcleo n_2 para o núcleo n_3, mapeados nos tiles τ_1 e τ_4, respectivamente.

8.2 Modelo de Energia

O consumo de energia de uma aplicação ocorre tanto nos núcleos quanto nas comunicações entre estes. Este trabalho foca apenas o consumo de energia na infra-estrutura de comunicação, e apresenta modelos para estimar os consumos de energia estática e dinâmica. Os modelos de consumo de energia e tempo de comunicação são usados pelos algoritmos de mapeamento na construção da função objetivo para avaliar a qualidade de cada mapa. Este trabalho utiliza modelos similares aos propostos em (HU; MARCULESCU, 2003) (HU; MARCULESCU, 2004) (MURALI; DE MICHELI, 2004a) (OGRAS; MARCULESCU, 2005) e (WU et al., 2003), e estende estes para estimar o consumo de energia estática da rede intrachip e energia dinâmica dos circuitos que operam mesmo na ausência de tráfego (MARCON et al., 2005a) (MARCON et al., 2005b). Detalha-se aqui o consumo de energia na topologia malha, mas mostra-se que a modelagem para outras topologias como toro dobrado e árvore-gorda também pode ser obtida de forma análoga (KREUTZ; MARCON; CALAZANS; CARRO; SUSIN, 2005a) (KREUTZ; MARCON; CARRO; WAGNER; SUSIN, 2005b).

A energia dinâmica da infra-estrutura de comunicação é proporcional à atividade de chaveamento que acontece quando os pacotes se movem através da NoC, consumindo energia nos fios de interconexão e internamente aos roteadores. Para estimar o consumo de energia, três aspectos foram estudados: (*i*) a dependência da topologia e geometria da infra-estrutura de comunicação; (*ii*) a dependência do tráfego; e (*iii*) a dependência do modelo computacional subjacente ao formato interno que descreve a aplicação.

8.2.1 Dependências Topológicas e Geométricas para Cálculo do Consumo de Energia Dinâmica em Redes Intrachip

O conceito de energia de bit *Ebit* (MURALI; DE MICHELI, 2004a) é usado para estimar o consumo de energia dinâmica para cada bit quando este altera o seu valor. *Ebit* é dividido em:

1. Energia dinâmica do bit consumida no roteador (*ERbit*), que ocorre no controle de chaveamento, na estrutura de chaveamento e nos elementos de armazenamento temporário do roteador (ver Seção 3.2.1);
2. Energia dinâmica do bit consumida nas conexões entre roteadores (*ELbit*); e,
3. Energia dinâmica do bit consumida nas conexões entre roteador e núcleo local (*ECbit*).

A relação entre estas quantidades é expressa pela Equação 8.5, que representa o consumo de energia dinâmica de um bit proveniente de um núcleo, passando por um roteador e uma conexão vertical ou horizontal.

Equação 8.5: $Ebit = ERbit + ELbit + ECbit$

Para estimar o consumo de energia dinâmica das topologias malha, toro dobrado e árvore-gorda foram assumidas considerações topológicas e físicas: (*i*) assume-se que a área do roteador não é significativa em comparação com a área restante do tile; e (*ii*) todas as topologias são implementadas com a mesma tecnologia. A consideração (*i*) permite negligenciar o custo de área do roteador e suas conseqüências no consumo de energia. Esta consideração é razoável, pois uma vez assim implementadas as redes, não implica grande erro nas estimativas. A consideração (*ii*), por sua vez, permite que equações obtidas para diferentes topologias sejam comparáveis.

Para topologias regulares de NoCs com tiles quadrados, onde cada uma das conexões entre roteadores pode ser implementada apenas com um segmento de reta vertical ou horizontal, tal como as topologias malha e toro dobrado, *ELbit* deve ser transformado em *ELHbit* ou *ELVbit*, dependendo do consumo de energia ser decorrente de um bit que trafega em uma conexão horizontal ou vertical, respectivamente. Assim, o uso de apenas *ELbit* ou de *ELHbit* com *ELVbit* depende das dimensões dos tiles da rede intrachip.

Para tiles grandes em relação ao roteador, *ECbit* pode ser negligenciado frente a *ELbit*, pois as distâncias entre núcleos e roteadores são muito menores que os caminhos de comunicação que conectam roteadores, implicando uma impedância muito menor e conseqüentemente um consumo de energia muito menor nos enlaces locais.

ERbit depende da estrutura do roteador e da tecnologia para estimar quantos bits podem alterar seu valor para escrever, ler ou preservar uma informação. Para certas topologias, tais como aquelas implementadas com roteadores irregulares, uma melhor estimativa do consumo de energia do roteador pode ser obtida dividindo *ERbit* em *EBbit* e *ESbit*. Ou seja, a energia dinâmica consumida pelo roteador é decomposta na energia dinâmica consumida pelo armazenamento e pelo controle e chaveamento do pacote, respectivamente. Os elementos *EBbit*, *ESbit*, *ELbit* (*ELHbit* e *ELVbit*) e *ECbit* que compõem *Ebit* estão ilustrados na Figura 8.2.

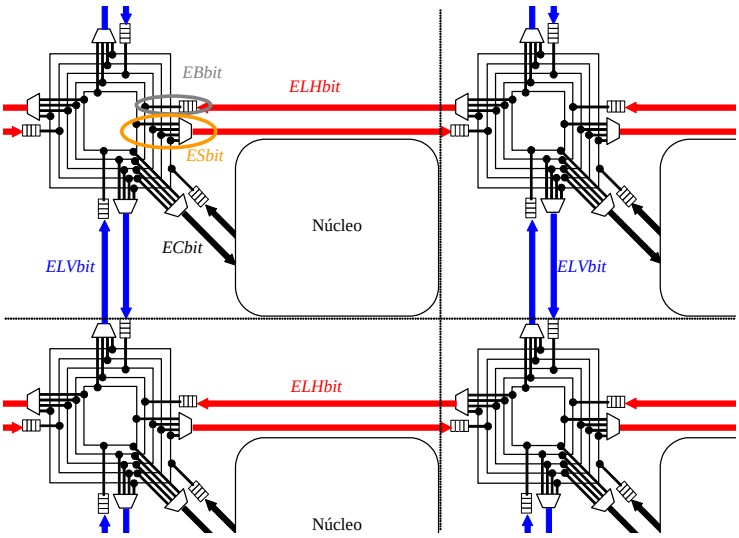


Figura 8.2: Ilustração dos elementos que modelam o consumo de energia de um bit. Estes são ilustrados em vista parcial de um esquemático de uma NoC com topologia malha. As linhas claras separam tiles, compostos por roteador, núcleo e conexões. Nesta ilustração, os tiles são retangulares e por este motivo, salienta-se a diferença entre *ELHbit* e *ELVbit*.

Devido às dependências topológica e geométrica das Equações de consumo de energia, define-se α como uma função para estimar a soma de *EBbit* e *ESbit* e φ como uma função para estimar a energia de cada conexão a partir de *ELHbit* e *ELVbit*. Desta forma, a Equação 8.6 é uma estimativa para o consumo de energia dinâmica de um único bit passando através de uma NoC, do núcleo mapeado no tile τ_i até o núcleo mapeado no tile τ_j, onde *TOP* representa a topologia da NoC selecionada para a estimativa.

Equação 8.6: $Ebit_{ij\ TOP} = \alpha_{TOP}(i, j, EBbit, ESbit) + \varphi_{TOP}(i, j, ELHbit \text{ e } ELVbit)$

A Figura 8.3 ilustra um layout de uma NoC 4 × 4 regular com topologia malha e com tiles de dimensão quadrada.

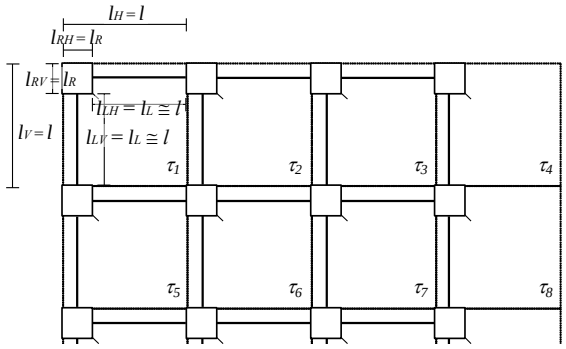


Figura 8.3: Exemplo parcial de layout de uma NoC 4 × 4 com topologia malha e tiles quadrados.

No exemplo de layout da Figura 8.3, as alturas e larguras dos roteadores, tiles e conexões entre roteadores são representados por l_{RV} e l_{RH}, l_{LV} e l_{LH}, e l_V e l_H, respectivamente. Assumindo que o tile seja quadrado, a largura e altura de cada conexão têm o mesmo tamanho. Assim, define-se uma medida única para todas as direções dos roteadores, tiles e conexões entre roteadores como: l_R, l_L e l, respectivamente, de tal forma que $l_{RV} = l_{RH} = l_R$, $l_{LV} = l_{LH} = l_L$ e $l_V = l_H = l$.

Para o layout de NoC exemplificado, é razoável estimar que $l_L = l - l_R$. Assumindo (*i*), a medida de um lado de um roteador se torna insignificante frente à medida de um lado de um tile, i.e. $l_R << l$, implicando $l_L \cong l$. Como *ELHbit* é diretamente proporcional a l_{LH} e *ELVbit* é diretamente proporcional à l_{LV}, consideramos aqui apenas *ELbit* que é diretamente proporcional à l, representando uma conexão entre tiles de uma NoC com topologia malha independente de direção do pacote. Assim, para topologias do tipo malha, a função φ da Equação 8.6 pode ser aproximada pela Equação 8.7, onde η corresponde ao número de roteadores por onde o bit deve passar para que um pacote parta de um núcleo posicionado em um tile τ_i e vá até um núcleo posicionado no tile τ_j.

Equação 8.7: $\varphi_{MALHA}(i, j, ELbit) = (\eta - 1) \times ELbit$

Para a topologia do tipo malha, o consumo de energia do roteador pode ser dado pela soma de *EBbit* com *ESbit*, pois, devido à simetria do roteador, a energia consumida nas áreas de armazenamento temporário e nas chaves é igual para qualquer caminho. Assim, a função α da Equação 8.6 pode ser estimada pela Equação 8.8.

Equação 8.8: $\alpha_{MALHA}(i, j, EBbit, ESbit) = \eta \times (EBbit + ESbit)$

A composição da Equação 8.7 com a Equação 8.8 gera a Equação 8.9, que é uma aproximação para o consumo de energia dinâmica de um único bit passando através de uma NoC com topologia malha, partindo do núcleo mapeado no tile τ_i até o núcleo mapeado no tile τ_j. Na Equação 8.9, η corresponde ao número de roteadores por onde o bit deve passar.

Equação 8.9: $Ebit_{ij\ MALHA} = \eta \times (EBbit + ESbit) + (\eta - 1) \times ELbit$

Comparando-se a Figura 8.3 com a Figura 8.4 observa-se que o layout da topologia malha é semelhante ao layout da topologia toro dobrado. Todavia, para a topologia toro dobrado existem conexões entre roteadores de tiles não adjacentes, nas direções vertical e horizontal. As medidas destas conexões são representadas por L_{LV} e L_{LH}.

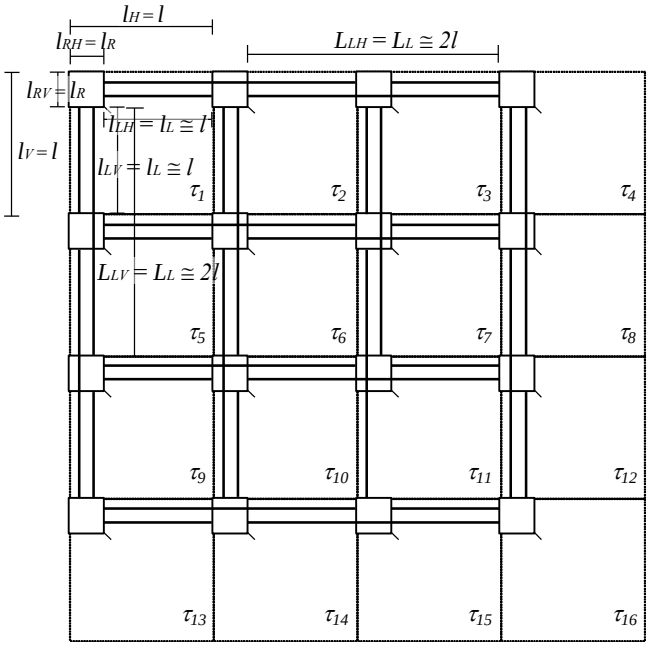


Figura 8.4: Exemplo de layout de uma NoC 4 × 4 com topologia toro dobrado e tiles quadrados.

Assumindo que os tiles sejam quadrados, define-se L_L como uma medida única para a altura e largura das conexões longas, de forma que $L_{LV} = L_{LH} = L_L$. Assim, $L_L = 2 \times l - l_R$. Assumindo (*i*), se $l_R << l$, então $L_L \cong 2 \times l$. Desta forma, para topologias toro dobrado, a função φ da Equação 8.6 pode ser aproximada pela Equação 8.10, onde linkSize depende do caminho ser uma conexão curta ou longa, com valores 1 ou 2, respectivamente.

Equação 8.10: $\varphi_{TORO\ DOBRADO}(i, j, ELbit) = \eta \times \text{linkSize} \times ELbit$

A topologia toro dobrado tem a função α da Equação 8.6 estimada de forma análoga à topologia malha, tal como pode ser observado na Equação 8.11. Contudo, devido à melhor diversidade de caminhos, o número de roteadores por onde o bit deve passar (η) na topologia toro dobrado tende a ser menor que o da topologia malha.

Equação 8.11: $\alpha_{TORO\ DOBRADO}(i, j, EBbit, ESbit) = \eta \times (EBbit + ESbit)$

A Equação 8.12 é a composição da Equação 8.10 com a Equação 8.11. Esta, é uma aproximação para o consumo de energia dinâmica de um único bit passando através de uma NoC com topologia toro dobrado, considerando que este parte do núcleo mapeado no tile τ_i e vai até o núcleo mapeado no tile τ_j.

Equação 8.12: $Ebit_{ij\ TORO\ DOBRADO} = \eta \times (EBbit + ESbit) + (\eta - 1) \times \text{linkSize} \times ELbit$

A irregularidade da topologia árvore-gorda implica caminhos com comprimentos diferentes e conseqüentemente diferentes *ELbit* (KREUTZ; MARCON; CALAZANS; CARRO; SUSIN, 2005a). Por exemplo, a Figura 8.5(a) mostra um possível layout de uma árvore-gorda com 16 tiles. Nele, são destacados caminhos entre roteadores de nível 1 com roteadores de nível 2. A Figura 8.5(b) correspondente à representação lógica de conexões entre roteadores da Figura 8.5(a).

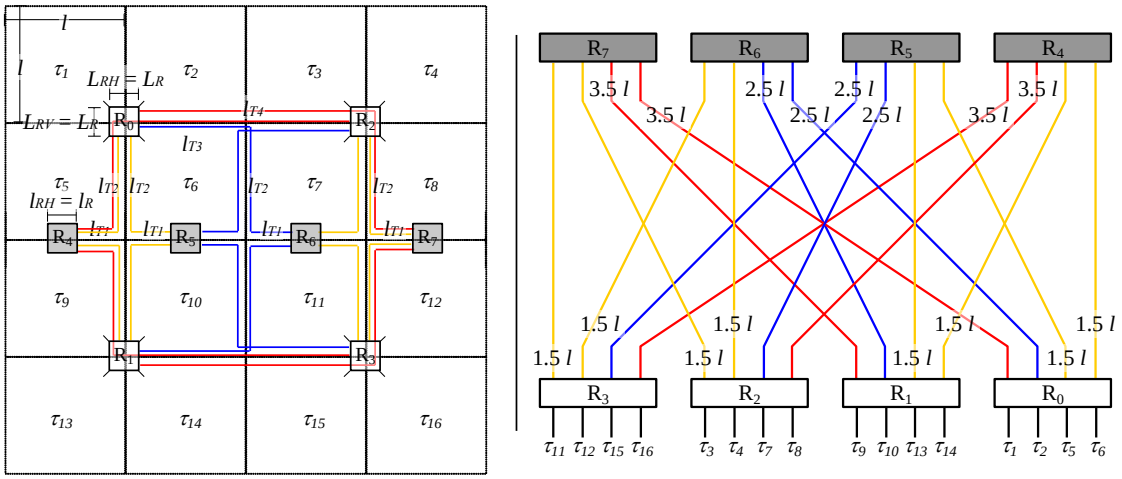


Figura 8.5: NoC com topologia de árvore-gorda quaternária. (a) apresenta um exemplo de layout, enquanto que (b) ilustra a representação lógica deste. Tanto em (a), quanto em (b), os 4 roteadores de nível 2 estão representados por quadrados sombreados, enquanto os 4 roteadores de nível 1 estão representados por quadrados transparentes. Em (b) estão ilustrados os tamanhos relativos dos fios de conexão entre roteadores.

Os roteadores de nível 2 da topologia de árvore-gorda têm medidas de lados semelhantes aos roteadores das topologias malha e toro dobrado (l_R). Os roteadores de nível 1, contudo, têm dimensão maior que os roteadores de nível 2 devido ao tamanho dos multiplexadores. Assim, define-se L_{RV} e L_{RH} para a altura e largura de roteadores de nível 1, respectivamente.

Assumindo que os tiles sejam quadrados, define-se L_R como medida de lado, independente deste ser vertical ou horizontal, de tal forma que $L_{RV} = L_{RH} = L_R$. Para compor os caminhos são ilustrados quatro segmentos: l_{T1}, l_{T2}, l_{T3} e l_{T4}. Tal que: $l_{T1} \cong (l - l_R) / 2$, $l_{T2} \cong l - (L_R + l_R) / 4$, $l_{T3} \cong l - L_R / 2$, e $l_{T4} \cong 2 \times l - L_R / 2$. Considerando o caminho de roteamento físico ilustrado na Figura 8.5(a), e assumindo (*i*), então $l_{T1} \cong l / 2$, $l_{T2} \cong l_{T3} \cong l$ e $l_{T4} \cong 2 \times l$. Para exemplificar, *ELbit* é diretamente proporcional a $3 \times l$ quando o caminho passa por um roteador de nível 2 que está parcialmente implementado nos mesmos tiles do roteador de nível 1 origem e destino. Isto ocorre porque o caminho passa por dois fios de comprimento l_{T1} e dois fios de comprimento l_{T2} (este é o caso de um caminho estabelecido entre os roteadores R_0 e R_1 através do

roteador R_5). Por outro lado, *ELbit* pode ser negligenciado se o caminho não passa por roteadores de nível 2.

A Equação 8.13 representa o consumo de energia de uma conexão para a topologia de árvore-gorda, com linkTreeSize sendo uma constante que define o comprimento da conexão.

Equação 8.13: $\varphi_{TREE}(i, j, ELbit) = \text{linkTreeSize} \times ELbit$

Nesta Equação, linkTreeSize indica a proporcionalidade da conexão com relação a *ELbit*, considerando um roteamento determinístico mínimo. Este pode valer 0, 3, 4 ou 5. O valor 0 indica que a comunicação é local ao roteador de nível 1, não necessitando gerar caminhos que passem por roteadores de nível 2. Os valores 3, 4 ou 5 dependem do comprimento da conexão, tal como ilustrado na Figura 8.5(a).

O controle interno de um roteador de nível 1 de uma árvore-gorda, tal como ilustrado na Figura 8.6, implica o uso de áreas de armazenamento temporário e de multiplexadores que consomem em média o dobro da energia dos multiplexadores das topologias malha e toro dobrado.

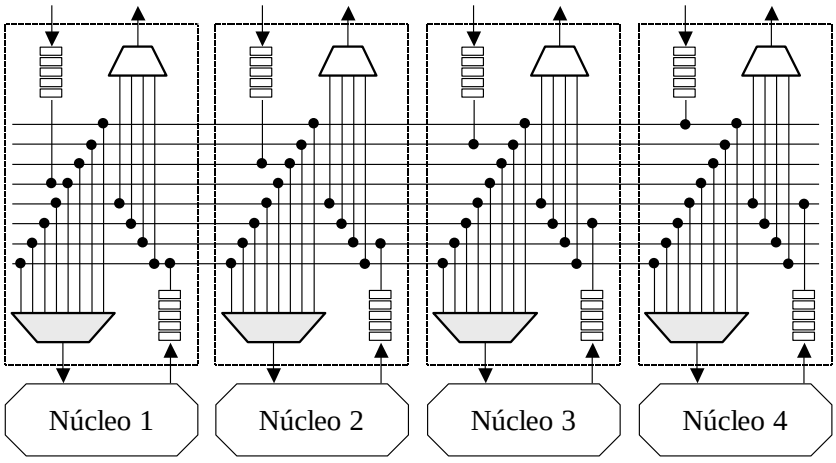


Figura 8.6: Diagrama esquemático de um roteador nível 1 de uma árvore-gorda conectado a quatro núcleos locais. Os multiplexadores que se conectam aos núcleos são maiores, pois têm que selecionar os sinais provenientes do roteador local, ou provenientes dos roteadores de nível 2.

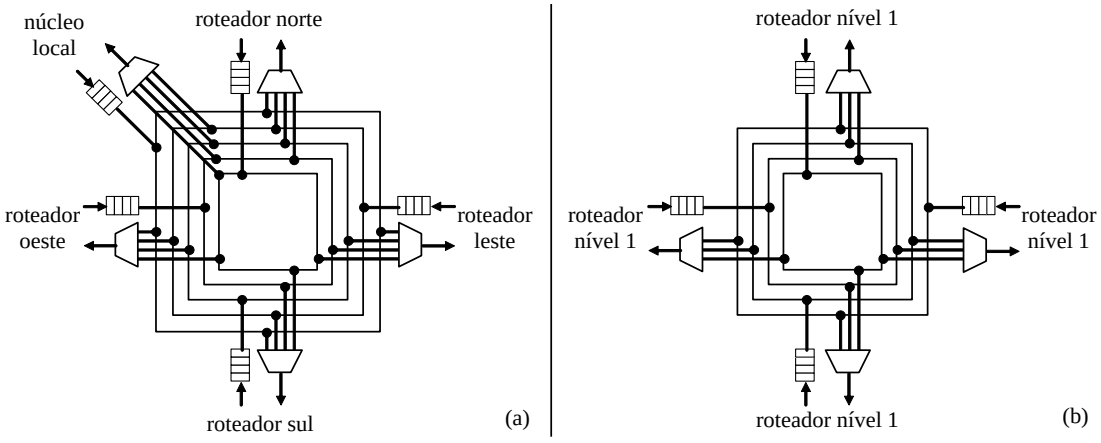


Figura 8.7: Diagrama esquemático de um roteador para topologias malha e toro dobrado (a), e de um roteador de nível 2 para topologia árvore-gorda (b).

Implementações de roteadores de nível 2 da árvore-gorda, quando comparadas com implementações dos roteadores das topologias malha e toro dobrado, mostram a redução de um buffer e uma correspondente redução no circuito de controle, pois estes não têm canal de comunicação com núcleo local. A Figura 8.7 ilustra estes esquemas de implementação, para que possam ser comparados.

Quando o roteamento implica o uso de um roteador de nível 2, existem três níveis de armazenamento temporário e dois estágios de multiplexação, um para cada nível. Além disto, os multiplexadores que se conectam com o núcleo destino (representado na Figura 8.6 em hachurado) consomem mais energia que os demais multiplexadores. Aqui se estima que estes multiplexadores consomem em média o dobro da energia, por terem praticamente o dobro de transistores.

Assume-se bufferTreeSize como uma constante que indica a proporcionalidade do consumo de energia dos *buffers* com relação a E_{Bbit}, e que pode valer 1 ou 3. O valor 1 refere-se ao uso de apenas um buffer de um roteador de nível 1, e o valor 3 refere-se ao uso de um buffer de cada um dos dois roteadores de nível 1 e um buffer de um roteador de nível 2. Assume-se, também, switchTreeSize como uma constante que indica a proporcionalidade do consumo de energia do controle com relação a E_{Sbit}, que pode valer 2 ou 3.6. O valor 2 é obtido quando é usada apenas uma chave de um roteador de nível 1 e o valor 3.6 refere-se ao uso de uma chave grande (consumo proporcional a 2) e uma pequena (consumo proporcional a 1) de dois roteadores de nível 1 e uma chave do roteador de nível 2 (consumo proporcional a 0.6). Dadas estas constantes, a Equação 8.14 representa o consumo de E_{Bbit} e E_{Sbit} para a topologia de árvore-gorda.

Equação 8.14: $\alpha_{TREE}(i, j, E_{Bbit}, E_{Sbit}) = \text{bufferTreeSize} \times E_{Bbit} + \text{switchTreeType} \times E_{Sbit}$

A Equação 8.15 é uma aproximação para o consumo de energia dinâmica de um único bit passando através de uma NoC com topologia árvore-gorda, sendo este proveniente do núcleo mapeado no tile τ_i e tendo como destino o núcleo mapeado no tile τ_j.

Equação 8.15: $E_{bit_{ij\ TREE}} = (1 \text{ ou } 3)\ E_{Bbit} + (2 \text{ ou } 3.6)\ E_{Sbit} + (0 \text{ ou } 3 \text{ ou } 4 \text{ ou } 5) \times E_{Lbit}$

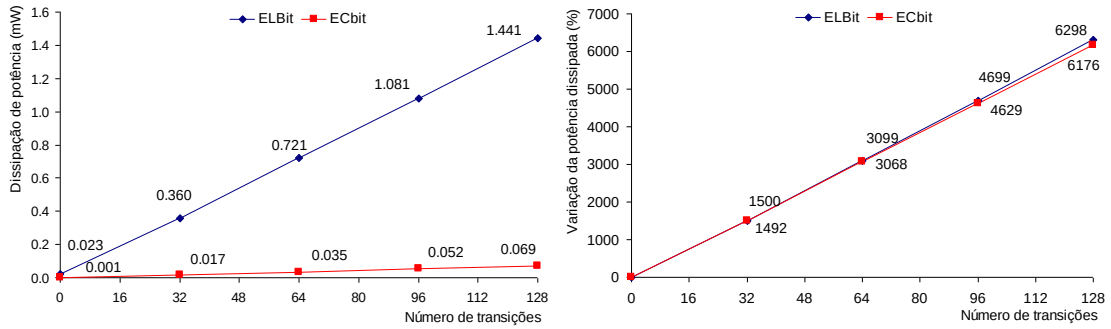
8.2.2 Dependência do Tráfego para Cálculo da Energia Dinâmica em Redes Intrachip

Para modelos de energia que consideram apenas a quantidade de bits de cada pacote e abstrai as transições, as Equações 8.9, 8.12 e 8.15 representam estimativas adequadas para o consumo de energia dinâmica de redes intrachip com topologias malha, toro dobrado e árvore-gorda, respectivamente. Estes modelos são aceitos e adotados por vários autores (HU; MARCULESCU, 2003) (HU; MARCULESCU, 2004) (MURALI; DE MICHELI, 2004a) (OGRAS; MARCULESCU, 2005). Todavia, trabalhos apresentados em (MARCON et al., 2005d) e (PALMA; MARCON; MORAES; CALAZANS; REIS; SUSIN, 2005) mostram que esta consideração pode inserir um erro significativo nas estimativas de consumo de energia dinâmica. Os casos analisados nestes trabalhos mostram que esta abstração pode levar a mapeamentos com mais de 45% de consumo de energia dinâmica. Este fato motivou a proposta do modelo

ECWM, descrito na Seção 5.2 e uma correspondente modificação no modelo de consumo de energia dinâmica na NoC, que permita capturar não apenas da quantidade de bits dos pacotes que trafegam na rede, mas também parte do conteúdo destes.

Para que as informações sobre o conteúdo do tráfego sejam utilizadas no cálculo do consumo de energia dinâmica, os parâmetros utilizados nas Equações que estimam esta energia devem ser decompostos em duas parcelas: (*i*) uma que permite estimar o consumo de energia devido ao número de bits dos pacotes, com fatores semelhantes aos inseridos nas Equações até aqui apresentadas; e (*ii*) outra que representa o acréscimo de energia causado pelas transições de bits consecutivos em um mesmo recurso de comunicação. A modelagem deste acréscimo é descrita a seguir.

Experimentos apresentados em (MARCON et al., 2005d) e (PALMA; MARCON; MORAES; CALAZANS; REIS; SUSIN, 2005) mostram que os parâmetros *ELbit* e *ECbit* somente influem no cálculo total de energia dinâmica quando existe transição de bits entre phits consecutivos. Estes experimentos podem ser observados na Figura 8.8 (a) e (b). Estas Figuras ilustram o efeito da variação de bits em um único fio, para um pacote com 128 bits, considerando todo o intervalo de transições. As Figuras mostram que é praticamente nulo o consumo de energia de um pacote quando não há transições (0.023 mW e 0.001 mW para conexão entre roteadores e entre núcleo e roteador, respectivamente). Por outro lado, considerando a variação do mínimo para o máximo de transições, o consumo de energia sofre variações que alcançam 6100%. Pode se notar, também, que o efeito da variação do número de transições sobre o consumo de energia é linear. Esta linearidade permite simplificar o modelo de energia, que pode ser construído com uma constante que representa a inclinação da reta.



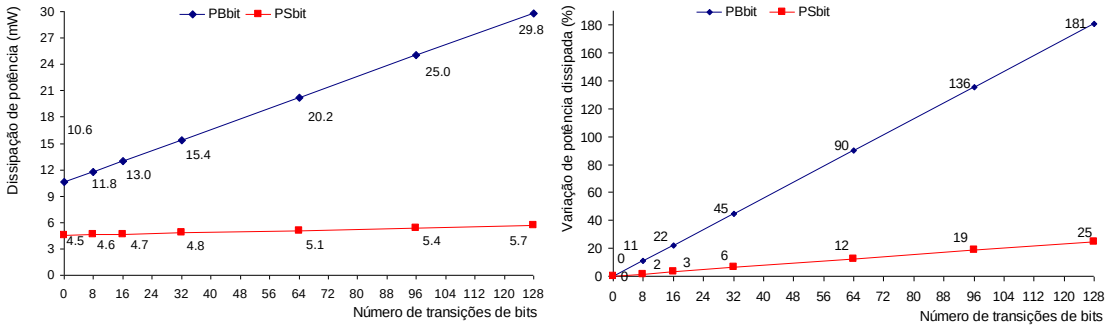
(a) (b)

Figura 8.8: Efeito das transições na dissipação de potência das conexões da NoC Hermes (MORAES, 2004). Utiliza-se um pacote de 128 bits trafegando entre núcleo e roteador local (curva *ECbit*) e entre roteadores (curva *ELbit*). Resultados obtidos a partir de simulações SPICE (BERKELEY UNIVERSITY, 2005) para uma tecnologia CMOS TSMC 0.35 μm, com 4 níveis de metal. (a) mostra os valores absolutos de dissipação de potência, enquanto (b) mostra a variação percentual em todo intervalo de transições. A conexão local ao tile foi avaliada considerando fios de comprimento 0.25 mm e parâmetros tecnológicos do metal de nível 1, enquanto que as conexões entre tiles foram avaliadas considerando fios de comprimento 5 mm e parâmetros tecnológicos do metal de nível 2.

A Figura 8.8 mostra também a comparação entre os parâmetros *ECbit* e *ELbit*. Daí, se conclui que para as considerações de tecnologia e dimensão adotadas, e que são válidas para abordagens de grão grosso, a abstração do parâmetro *ECbit* não gera erro significativo nas estimativas de consumo de energia dinâmica. Outro aspecto importante

a ser considerado, é o fato de *ELbit* não ter efeito significativo na estimativa do consumo de energia quando não há transições. Assim, não é necessário decompor este parâmetro, mas apenas usá-lo, considerando apenas o número de transições.

Para os parâmetros *EBbit* e *ESbit*, o efeito do tráfego no consumo de energia dinâmica é comparável ao efeito causado pela quantidade de bits, sendo necessária à decomposição destes parâmetros. Este efeito pode ser observado na Figura 8.9. Nesta Figura, estão representadas as potências dissipadas devido à transição e não às energias.



(a) (b)

Figura 8.9: Efeito das transições na dissipação de potência para buffer (curva *PBbit*) e controle (curva *PSbit*) de um roteador da NoC Hermes (MORAES, 2004). (a) mostra os valores absolutos de dissipação de potência, enquanto (b) mostra a variação percentual em todo intervalo de transições.

Da Figura 8.9 extrai-se os seguintes dados:

1. O efeito das transições é linear para ambos os parâmetros, simplificando o modelo de consumo de energia, de forma análoga ao parâmetro *ELbit*;
2. O efeito das transições no circuito de controle e de chaveamento (curva *PSbit*) é pequeno. Para o caso máximo apresentado, chega a 25%. Este fato permitiria a abstração da variação da parcela que considera as transições com um pequeno erro nas estimativas de consumo de energia;
3. O efeito das transições é muito significativo nos *buffers* (curva *PBbit*). Para o caso apresentado chega a 181%, mas não pode ser abstraído o consumo de energia sem transição, pois este é da mesma ordem de grandeza que a máxima variação. Assim, é necessário considerar as duas parcelas no modelo de energia.

Considerando os dados extraídos acima e adotando o sufixo "*_Q*" para os parâmetros que consideram a quantidade de bits e "*_T*" para os parâmetros que consideram as transições, a Equação 8.16 é um refinamento da estimativa de consumo de energia dinâmica apresentada na Equação 8.5.

Equação 8.16: $Ebit = EBbit\_Q + EBbit\_T + ESbit\_Q + ESbit\_T + ELbit\_T$

8.2.3 Dependência do Modelo Computacional para Cálculo da Energia Dinâmica em Redes Intrachip

Esta Seção mostra como as Equações de consumo de energia dinâmica de um bit trafegando por uma rede são dependentes do modelo computacional da aplicação.

Para o CWM, como descrito na Seção 5.1, uma aplicação é representada

apenas pela quantidade de comunicação que ela induz na infra-estrutura de comunicação. Este modelo permite representar a quantidade total de bits que são transmitidos de um núcleo para outro durante a execução de uma aplicação. Sejam τ_i e τ_j os tiles onde os núcleos n_a e n_b estão mapeados, respectivamente. Então, a energia dinâmica consumida por todos os pacotes transmitidos do núcleo n_a para o núcleo n_b é dada por $E_{ab} = \omega_{ab} \times Ebit_{ij}$, onde ω_{ab} é o número total de bits transmitidos e $Ebit_{ij}$ pode ser obtido das Equações 8.9, 8.12 ou 8.15, dependendo da topologia de rede. Assim, para o modelo CWM, a Equação 8.17 representa a energia dinâmica consumida em uma NoC devido a todas as comunicações entre núcleos de uma aplicação.

Equação 8.17: $$EDyNoC_{(CWM)} = \sum_{n_a, n_b} E_{ab} \quad \forall\, n_a, n_b \in N, \text{ com } \omega_{ab} > 0$$

O ECWM (MARCON et al., 2005d) (PALMA; MARCON; MORAES; CALAZANS; REIS; SUSIN, 2005), como descrito na Seção 5.2, se diferencia de CWM por considerar também as transições das comunicações. Dada as mesmas considerações de CWM e acrescentando σ_{ab}, como o número total de transições de todos os pacotes transmitidos do núcleo n_a para o núcleo n_b, então $E_{ab} = Ebit_{ij}\,(\omega_{ab}, \sigma_{ab})$ é a energia dinâmica consumida na NoC devido ao tráfego destes pacotes. Para exemplificar o cálculo de E_{ab} a Equação 8.18 relaciona ω_{ab} e σ_{ab} com os parâmetros apresentados na Equação 8.16 e com o cálculo do consumo de energia dinâmica por um bit na topologia malha apresentada na Equação 8.9.

Equação 8.18: $$E_{ab\ MALHA} = \eta \times ((EBbit\_Q + ESbit\_Q) \times \omega_{ab} + (EBbit\_T + ESbit\_T) \times \sigma_{ab}) + (\eta - 1) \times ELbit \times \sigma_{ab}$$

A Equação 8.19 representa a energia dinâmica consumida por todas as comunicações de todos os núcleos de uma aplicação, quando a mesma está representada por ECWM.

Equação 8.19: $$EDyNoC_{(ECWM)} = \sum_{n_a, n_b} E_{ab} \quad \forall\, n_a, n_b \in N, \text{ com } \omega_{ab} > 0 \text{ e } \sigma_{ab} \geq 0$$

Os modelos CDM, CDCM e ACPM, descritos nas Seções 5.3, 5.4 e 5.5, permitem representar a quantidade total de bits de cada mensagem transmitida entre dois núcleos no meio físico. Sejam τ_i e τ_j os tiles onde os núcleos n_a e n_b estão mapeados, respectivamente. Seja w_{abq} a quantidade total de bits da mensagem de índice q, que parte do núcleo n_a para o núcleo n_b. Então, a energia dinâmica consumida pela mensagem de índice q da comunicação $n_a \rightarrow n_b$ é dada por $E_{abq} = w_{abq} \times Ebit_{ij}$. Assim, a Equação 8.20 representa para os três modelos a quantidade total de energia dinâmica consumida por todas as mensagens de todos os núcleos, e k_{ab} representa o número de mensagens enviadas do núcleo n_a para o núcleo n_b (MARCON et al., 2005a).

Equação 8.20: $$EDyNoC_{(CDM,\ CDCM \text{ ou } ACPM)} = \sum_{n_a, n_b} \sum_{q=1}^{k_{ab}} E_{abq} \quad \forall\, n_a, n_b \in N, \text{ com } w_{abq} > 0$$

Sejam τ_i e τ_j os tiles onde os núcleos n_x e n_y estão mapeados, respectivamente. Seja ϕ uma função que associa tarefas a núcleos, tal que $n_x = \phi\,(t_a)$ e $n_y = \phi\,(t_b)$. Seja w_{ab} a quantidade total de bits transmitido da tarefa t_a para a tarefa t_b. Então, a energia

dinâmica consumida por todos os bits da comunicação $t_a \rightarrow t_b$ é dada por $E_{ab} = w_{ab} \times Ebit_{ij}$. Para o modelo CTM, o cálculo da energia dinâmica consumida na rede intrachip, descrito na Equação 8.21, é dado pelo conjunto de comunicações entre tarefas que não compartilham o mesmo PE.

Equação 8.21: $EDyNoC_{(CTM)} = \sum_{t_a, t_b} E_{ab} \quad \forall\ t_a, t_b \in T$, com $n_x = \phi(t_a)$, $n_y = \phi(t_b)$, $w_{ab} > 0$ e $n_x \neq n_y$

O CTM ainda fornece o consumo de energia dinâmica da tarefa executando sobre uma aplicação. Esta última estimativa não é dada por nenhum outro modelo aqui apresentado e tão pouco faz parte do escopo deste trabalho. Para melhor entender o modelo, o cálculo do consumo de energia dinâmica das tarefas da aplicação (*EDyTask*) é dado pela soma de todos os e_{ij}, onde cada e_{ij} representa o consumo de energia dinâmica da tarefa t_i executando sobre o j-ésimo PE, tal como descrito na Seção 5.6 e ilustrado na Equação 8.22.

Equação 8.22: $EDyTask_{(CTM)} = \sum_{t_i} e_{ij} \quad \forall\ t_i \in T$ executando sobre o j-ésimo PE

8.2.4 Dependência do Modelo Computacional para Cálculo das Energias Estática e Dinâmica Consumidas nos Circuitos que Operam mesmo na Ausência de Tráfego

A potência estática dissipada em cada roteador (*PStRouter*) é proporcional ao número de elementos ativos que o compõem. A energia dinâmica consumida em cada roteador durante o período de ociosidade (*EDynRouter*) ocorre apenas em alguns subcircuitos deste. As conexões entre tiles e entre tile e núcleo, por sua vez, têm um consumo de energia desprezível quando não há tráfego na rede, sendo aqui este consumo ignorado.

PStRouter é uma parcela que deve ser computada durante todo o período de operação da rede intrachip, pois independente de estar ociosa ou com tráfego, o consumo de energia estática está sempre ocorrendo. Por outro lado, *EDynRouter* é uma parcela difícil de ser estimada, pois enquanto alguns roteadores estão ociosos, outros estão com tráfego e, mesmo estes com tráfego não necessitam ter todos os canais ocupados. Para simplificar os modelos de energia, o circuito de controle de cada roteador foi dividido em circuitos que realizam o roteamento e circuitos que estão sempre ativos detectando a necessidade de comunicação entre roteadores. O consumo de energia dinâmica dos circuitos que realizam o roteamento é capturado por *ESbit*. Já, o consumo de energia dinâmica dos circuitos que estão sempre em atividade é capturado por *EDynRouter*. Ou seja, *EDynRouter* deixa de ser uma parcela computada apenas em períodos de ociosidade e passa a ser computada durante toda a operação da rede intrachip, da mesma forma que *PStRouter*. Além disto, calcular *PDynRouter* (potência dinâmica dissipada pelos circuitos que operam mesmo na ausência de tráfego) a partir de *EDynRouter*, permite definir *PRouter* como a soma de *PStRouter* e *PDynRouter* (*PRouter* = *PStRouter* + *PDynRouter*). Finalmente, sendo $|\tau|$ o número de tiles da rede intrachip, a Equação 8.23 computa a dissipação de potência de uma NoC (*PStDynNoC*), composta pela potência estática consumida por todos os elementos ativos desta e

composta pela potência dinâmica dos circuitos que operam mesmo na ausência de tráfego.

Equação 8.23: $P_{StDynNoC} = |\tau| \times P_{Router}$

Para os modelos CWM e ECWM, o *tempo de execução da aplicação* (t_{exec}) somente pode ser estimado a partir do tempo necessário para transmitir todas as mensagens, sem considerar a possibilidade de contenção e sem considerar o tempo de computação que precede as mensagens. Além do mais, como não existem informações de ordem total ou parcial, todas as mensagens são transmitidas concorrentemente. Estas considerações mostram que o t_{exec} para estes modelos é tipicamente muito menor que o tempo de execução de uma aplicação real e serve apenas para estimar o tempo mínimo necessário para transmitir todas as mensagens em uma infra-estrutura ideal, aquela que permite a comunicação entre quaisquer núcleos ao mesmo tempo. Em conseqüência, estes modelos não são suficientes para computar o consumo de energia estática de toda a rede intrachip de forma precisa. Também são inadequados para calcular o consumo de energia dinâmica dos circuitos que operam mesmo na ausência de tráfego.

Para o modelo CDM, t_{exec} é obtido pela soma de todos $d_{ij,abq}$, e todos os tempos de contenção. Ou seja, o CDM considera que o gargalo de operação está na comunicação e não no processamento. Uma vez transmitida uma mensagem, a próxima mensagem pode ser transmitida, sem considerar o *tempo necessário para a computação do núcleo*[32] que irá transmitir a próxima mensagem.

O modelo CDCM é semelhante ao CDM. Porém, como este modelo permite representar o tempo de computação de um núcleo n_i antes do envio da q-ésima mensagem (t_{iq}), o cálculo do t_{exec} é mais preciso, pois é obtido pela soma de todos os t_{iq} e $d_{ij,abq}$, e de todos os tempos de contenção.

O modelo ACPM fornece os instantes de tempo do envio das mensagens, mas estes instantes não refletem necessariamente o tempo real de execução da aplicação, pois para tanto, os tempos teriam que ser extraídos de uma aplicação sendo executada sobre os elementos de processamento da arquitetura real. Assim, o t_{exec} do ACPM é uma estimativa do tempo de execução para a arquitetura simulada.

O modelo CTM representa o tempo total de execução da aplicação pela soma de todos t_{Eij}, onde t_{Eij} é o tempo de execução da tarefa t_i sobre o j-ésimo PE. Todavia, o tempo gasto pelas mensagens trafegando pela infra-estrutura de comunicação é computado de forma semelhante ao CWM, pois a quantidade total de bits de todas as mensagens entre duas tarefas é totalizada em uma única variável. Para aplicações computacionalmente intensivas, este tempo pode ser negligenciado e assim o t_{exec} se torna uma estimativa razoável.

O consumo de energia estática de toda a NoC e dinâmica dos circuitos que operam mesmo na ausência de tráfego é proporcional a $P_{StDynNoC}$ e t_{exec}. Assim, a Equação 8.24 ilustra o cálculo deste consumo de energia ($E_{StDynNoC}$), considerando que para cada modelo pode ser computado um t_{exec} diferente.

---

[32] O modelo CDM tem por definição que o tempo de computação do núcleo seja 0. Todavia, a ferramenta CAFES (Capítulo 9) expande a capacidade do modelo permitindo que o projetista atribua um tempo de computação diferente de 0, embora este tenha que ser igual para todos os núcleos da aplicação.

Equação 8.24: $E_{StDynNoC} = P_{StDynNoC} \times t_{exec}$

8.2.5 Cálculo Total da Energia Consumida

A Equação 8.25 traz o consumo de energia total na NoC (E_{NoC}) para todos os modelos da aplicação, onde são computadas as energias dinâmica e estática da infra-estrutura de comunicação.

Equação 8.25: $E_{NoC} = E_{StDynNoC} + E_{DyNoC}$

Finalmente, a Equação 8.26 informa o consumo de energia total da aplicação (E_{DyApp}) para o CTM. Este modelo permite computar as energias dinâmica e estática da infra-estrutura de comunicação e, também, a energia dinâmica consumida pelos núcleos da aplicação (E_{DyTask}).

Equação 8.26: $E_{DyApp} = E_{NoC} + E_{DyTask}$

9 VALIDAÇÃO DO MODELO DE CONSUMO DE ENERGIA PARA NOCS

A avaliação da qualidade dos mapeamentos foi realizada através de métodos que habilitam estimar parâmetros de aplicações implementadas sobre diversas arquiteturas alvo. Para o escopo deste trabalho, os parâmetros relevantes são: (*i*) tempo de execução da aplicação, focando principalmente o atraso causado pela infra-estrutura de comunicação, e (*ii*) consumo de energia da infra-estrutura durante a execução da aplicação. Quanto mais alto o nível de abstração dos modelos, maior é a imprecisão das estimativas. Com o objetivo de reduzir esta imprecisão, propõe-se um fluxo de atividades que permite avaliar e validar os modelos que capturam aplicações e implementações físicas das infra-estruturas de comunicação.

Entre as atividades descritas no fluxo proposto está a síntese da rede intrachip e dos modelos de alto nível que descrevem a aplicação em descrições VHDL. Devido ao foco deste trabalho ser infra-estruturas de comunicação e não aplicações, as descrições em alto nível da aplicação são sintetizadas para VHDL comportamental. A rede intrachip, por sua vez, é descrita em VHDL estrutural. Como exposto anteriormente, a infra-estrutura de comunicação escolhida para estudo de caso deste trabalho é a NoC Hermes[33] (MORAES, 2004), devido à disponibilidade da descrição em VHDL estrutural e ao ferramental de apoio dado pelo ambiente Maia[34] (OST et al., 2005). As estimativas de mapeamentos são computadas com o auxílio de simulador VHDL, onde a descrição da aplicação serve como *testbench* para avaliar os efeitos da computação e comunicação desta na rede intrachip.

9.1 Método para Síntese e Validação da Rede Intrachip

A validação do modelo de atrasos na NoC foi efetuada com o auxílio de simulador VHDL executando uma descrição estrutural da NoC Hermes. Este nível de abstração de descrições é suficiente para estimar tempo de execução, pois a medida de tempo que interessa para as estimativas de alto nível é o número de ciclos de relógio necessário para tráfego dos phits na rede. Esta informação é obtida de forma precisa neste nível, permitindo que as Equações 8.2, 8.3 e 8.4 descritas no Capítulo 8 sejam validadas para serem inseridas no modelo de tempo do *framework* CAFES (Capítulo 10) habilitando estimar o tempo das comunicações. Salienta-se que os modelos da infra-

[33] Hermes é uma rede intrachip de dimensão parametrizável, construída com topologia malha, chaveamento wormhole e roteamento XY. Nesta rede flit e phit têm o mesmo tamanho.

[34] Maia é um ambiente para geração de descrições VHDL da NoC Hermes. Este permite, também, a geração de tráfego para simular comunicações entre núcleos e assim validar a NoC gerada.

estrutura de comunicação são independentes dos modelos das aplicações. Assim, as Equações 8.2, 8.3 e 8.4 valem para qualquer modelo de aplicação que forneça informações sobre tile origem e destino, e quantidade de phits que irá trafegar de um tile a outro em uma NoC com roteamento wormhole.

Validar o modelo de energia é tarefa bem mais complexa que validar o modelo de tempo. Normalmente, para obter um bom grau de precisão, as estimativas de consumo de energia são realizadas no nível elétrico. O grande problema neste nível de abstração é o tempo necessário para simular circuitos com uma grande quantidade de transistores, como é o caso de NoCs com diversos roteadores[35]. Para minimizar este problema, propõe-se uma técnica para estimar o consumo de energia de circuitos descritos em VHDL estrutural. Ou seja, no mesmo nível de abstração que os modelos de tempo de comunicação foram validados. Esta técnica tem como ponto chave a caracterização de uma biblioteca de portas lógicas – as mesmas portas lógicas utilizadas na implementação física do circuito. A idéia é que se a implementação física for realizada com as mesmas portas que foram caracterizadas, então uma simulação da aplicação com estimativas de consumo de energia das portas caracterizadas conduzirá a uma estimativa razoável do consumo de energia do circuito. A técnica foi planejada para estimar o consumo de energia da NoC Hermes, contudo ela é suficientemente genérica para estimar o consumo de energia de qualquer circuito descrito em VHDL estrutural. A Figura 9.1 apresenta o fluxo de atividades da técnica proposta. O fluxo e os componentes nele ilustrados são descritos no decorrer deste Capítulo.

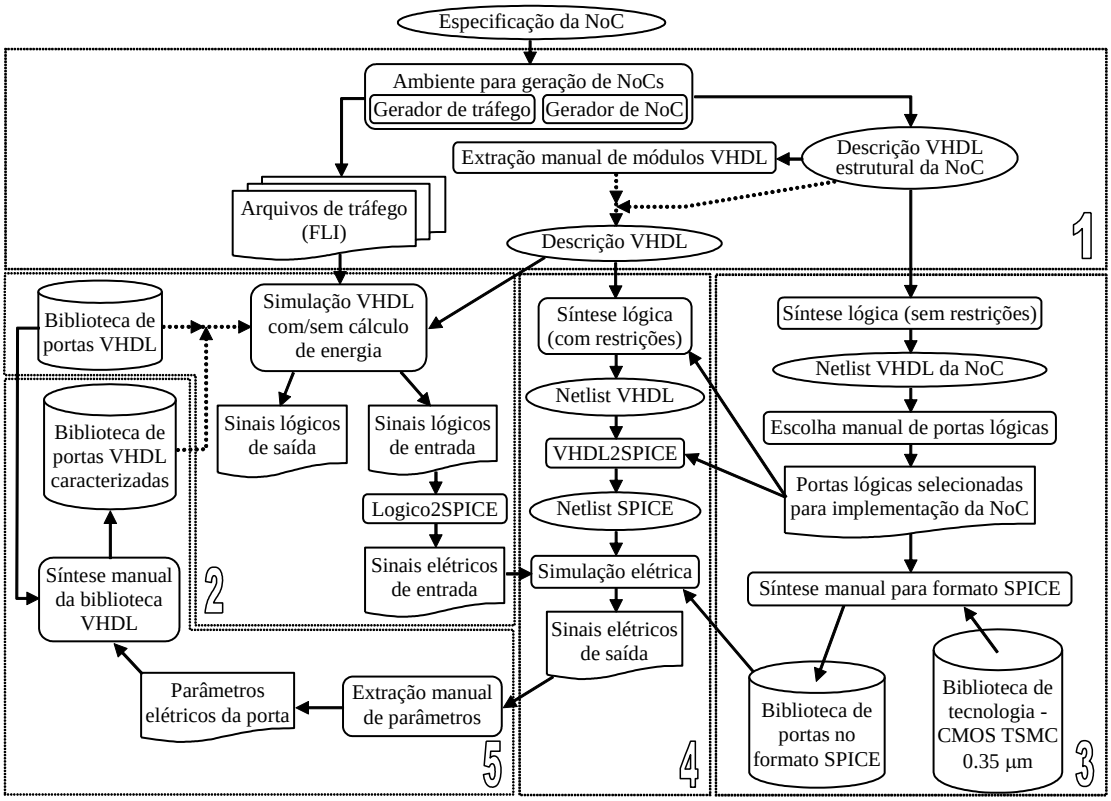


Figura 9.1: Fluxo de atividades para estimar o consumo de energia de circuitos descritos em VHDL. O fluxo está dividido em 5 conjuntos de atividades,

[35] Para exemplificar a quantidade de transistores, uma implementação de um roteador da rede Hermes com phits de 16 bit e *buffers* de profundidade 4 tem em torno de 15.000 transistores.

numerados e envolvidos por retângulos pontilhados. Retângulos com cantos arredondados correspondem a ferramentas ou ações manuais, elipses representam descrições, cilindros são bibliotecas, retângulos com borda inferior ondulada são arquivos. Flechas contínuas indicam direção do fluxo e flechas pontilhadas indicam direções mutuamente exclusivas.

9.1.1 Geração de NoC e Tráfego

O primeiro conjunto de atividades objetiva a criação da descrição da NoC que será usada em todos os demais conjuntos de atividades. Este conjunto também gera o tráfego utilizado como estímulo para verificar o funcionamento da NoC e/ou de módulos extraídos desta. Esta etapa parte de uma especificação, onde o projetista informa parâmetros[36] que serão entradas para o ambiente Maia (OST et al., 2005). Através destes parâmetros o ambiente fornece a descrição estrutural VHDL de uma NoC Hermes correspondente. De acordo com os parâmetros de tráfego, o ambiente fornece vários conjuntos de sinais, cada um representando o comportamento da comunicação de um módulo com os demais. Este comportamento é simulado por sinais conectados à NoC através de canais de entrada dos roteadores. Neste primeiro conjunto de atividades, o roteador da NoC gerada pode ser particionado manualmente, para que sejam analisados os efeitos da comunicação nos diferentes módulos deste, facilitando a captura de um modelo de energia representativo. Para os roteadores da NoC foram extraídos os circuitos contendo os *buffers* e os circuitos de controle de roteamento. Com estes circuitos é possível estimar os parâmetros E_{Bbit} e E_{Sbit}, descritos no Capítulo 8.

9.1.2 Simulação Lógica

O segundo conjunto de atividades permite realizar a validação estrutural da NoC ou de um de seus módulos, através de estímulos (o tráfego) gerados no primeiro conjunto de atividades. Estes estímulos são fornecidos para o simulador VHDL com auxílio de um código em linguagem C. O simulador VHDL ModelSim (MENTOR GRAPHICS, 2005b) permite interfacear o código escrito em C com a linguagem VHDL através da linguagem proprietária FLI (do inglês, *Foreign Language Interface*). Com esta linguagem, os sinais contidos em arquivos C são convertidos para sinais VHDL que, por sua vez, são as entradas dos módulos que serão simulados. Desta forma, obtém-se resultados de consumo de energia de diversos componentes da NoC, tais como áreas de armazenamento temporário, circuito de controle e conexão entre roteadores

Além do grande tempo de processamento, um dos problemas para a simulação em nível elétrico é a geração de estímulos elétricos que permitam verificar o funcionamento de um sistema. Para resolver este problema, os estímulos utilizados para validar a descrição VHDL da NoC, que foram obtidos da geração de tráfego do ambiente Maia, foram utilizados para validar a descrição SPICE (BERKELEY UNIVERSITY, 2005). A simulação VHDL resulta na geração de 2 arquivos: (*i*) um

[36] O ambiente Maia permite a parametrização de informações para a geração da NoC, tais como: (*i*) profundidade dos *buffers*, (*ii*) número de linhas e colunas, (*iii*) tipo de controle de fluxo (*credit-based* ou *handshake*), (*iv*) tamanho do flit/phit, e (*v*) algoritmo de roteamento (*west-first* ou XY). O ambiente permite também especificar algumas características do tráfego entre os núcleos da aplicação, tais como: (*i*) a origem e o destino de cada pacote e (*ii*) as taxas de inserção de pacotes na rede.

contendo os sinais que representam o resultado da simulação – composto de formas de onda, que serve para validação do circuito pela comparação visual com resultados de simulação elétrica equivalente; (*ii*) outro contendo os valores e os instantes precisos de cada estímulo de entrada para o circuito simulado. Aqui, a ferramenta Lógico2Spice foi desenvolvida com o objetivo de converter arquivos de estímulos para arquivos equivalentes em formato SPICE, tal como ilustrado na Figura 9.2. Este último é utilizado como descrição de estímulos de entrada na simulação elétrica.

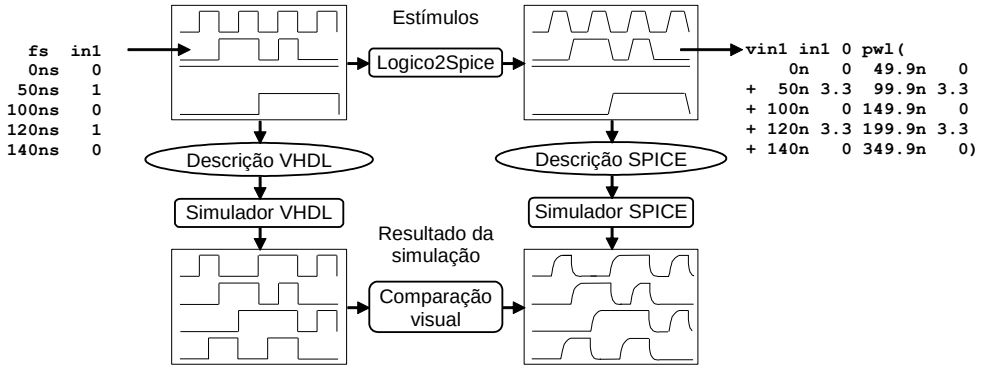


Figura 9.2: Comparação de resultados de simulações VHDL e SPICE, e extração de informações elétricas para avaliação dos modelos de consumo de energia.

9.1.3 Caracterização da biblioteca de portas lógicas

O terceiro conjunto de atividades compreende a caracterização da biblioteca de portas lógicas utilizada na simulação elétrica. A primeira atividade desenvolvida é a síntese lógica, sem restrições da descrição da NoC gerada pelo ambiente Maia. Esta síntese, utiliza a ferramenta Leonardo Spectrum (MENTOR GRAPHICS, 2005a), que transforma a descrição VHDL RTL hierárquica da NoC para um diagrama não hierárquico de portas lógicas - um *netlist*. A partir deste *netlist*, as portas consideradas mais importantes (seja pelo número de vezes que são utilizadas, seja por suas especificidades ou complexidades) são anotadas para serem caracterizadas e posteriormente utilizadas no quarto conjunto de atividades. A caracterização inicia com a síntese manual da porta para o formato SPICE, que deve considerar a *biblioteca de tecnologia* (CMOS TSMC 0.35 μm) adotada. A Figura 9.3 ilustra uma porta tri-state no formato SPICE que fará parte da biblioteca VHDL.

```
.subckt Inversor out in Vcc 0
MP1  out in Vcc Vcc MODP L=0.35U W=5.0U AD=10.0P AS=10.0P PD=9.0U PS=9.0U
MN2  out in   0   0 MODN L=0.35U W=2.0U AD= 4.0P AS= 4.0P PD=6.0U PS=6.0U
.ends Inversor

.subckt TransmissionGate out in control notControl Vcc 0
M1 in  control    out   0 MODN L=0.35U W=2.0U AD= 4.0P AS= 4.0P PD=6.0U PS=6.0U
M2 in  notControl out Vcc MODP L=0.35U W=5.0U AD=10.0P AS=10.0P PD=9.0U PS=9.0U
.ends TransmissionGate

.subckt TriState out in control Vcc   0
X1 notControl control                    Vcc 0 Inversor
X2 out_1           in control notControl Vcc 0 TransmissionGate
R0 out          out_1                    50
C0 out              0                    50fF
.ends TriState
```

Figura 9.3: Porta lógica tri-state no formato SPICE. A porta é composta pelos subcircuitos: Inversor e TransmissionGate e tem modelado na saída uma

capacitância de 50 fF e uma resistência de 50 ohms, representando o *fan-out* de 3 portas lógicas. O *fan-out* descrito é apenas ilustrativo, pois para a estimativa do consumo de energia na simulação VHDL o valor do *fan-out* é estimado considerando o número de portas efetivamente conectadas a cada saída.

9.1.4 Síntese lógica

O quarto conjunto de atividades refere-se à avaliação elétrica de módulos da NoC. Para tanto, sintetiza-se logicamente um módulo VHDL, utilizando novamente a ferramenta Leonardo Spectrum, agora usando como restrição de síntese as portas caracterizadas manualmente durante o terceiro conjunto de atividades. Assim, o *netlist* VHDL gerado conterá apenas portas lógicas já caracterizadas. Estas portas são entradas da ferramenta VHDL2SPICE que converte o *netlist* de portas lógicas VHDL em um *netlist* no formato SPICE. Com este último e mais os sinais de entrada provenientes do segundo conjunto de atividades, a simulação do módulo é realizada e são extraídos valores do consumo de energia do módulo frente ao tráfego de entrada. Estes valores são usados para extração de parâmetros usados nas avaliações de níveis de abstração superior, tal como E_{BPhit} e E_{SPhit}, descritos na Seção 10.1.

9.1.5 Geração da Biblioteca VHDL Caracterizada

O quinto conjunto de atividades compreende a caracterização das bibliotecas de portas utilizadas na simulação VHDL. Todas as portas lógicas selecionadas manualmente no terceiro conjunto de atividades são agora descritas em VHDL de acordo com o comportamento de suas entradas. Nesta descrição, é anotado o consumo de energia para cada transição e a dissipação de potência estática durante o período que as entradas permanecem inalteradas.

Para exemplificar, os valores referentes ao consumo de energia dinâmica de uma porta inversora estão ilustrados na Tabela 9.1. Os parâmetros elétricos foram extraídos para a tecnologia CMOS TSMC 0.35 μm, considerando um *fan-out* típico de 3 portas.

Tabela 9.1: Cálculo do consumo de energia dinâmica para um inversor. As duas primeiras colunas ilustram as transições de entrada e as correspondentes variações na saída. As colunas “Início”, “Fim” e “Potência média dissipada” mostram um intervalo de tempo e a dissipação de potência neste intervalo. Por fim, são transportados para a biblioteca VHDL os valores de energia consumida pela porta lógica devido à transição. Ou seja, a coluna “Energia”, que é a integral da potência no intervalo descrito.

Entrada	Saída	Início (ns)	Fim (ns)	Potência média dissipada (W)	Energia (J)
0->1	**1->0**	**45**	**55**	**4.5964E-05**	**4.5964E-13**
1->0	**0->1**	**95**	**105**	**4.9066E-05**	**4.9066E-13**

A Tabela 9.2 é semelhante à Tabela 9.1. Todavia, esta apresenta a dissipação de potência estática da mesma porta inversora durante o intervalo de tempo em que as entradas *permanecem inalteradas*. Os valores obtidos nesta Tabela são multiplicados pelo tempo de operação do circuito, e somados aos valores do consumo de energia

gerada pelo chaveamento para compor a estimativa de consumo total de energia do inversor.

Tabela 9.2: Cálculo da dissipação de potência estática para uma porta inversora, durante o período em que as entradas permanecem inalteradas.

Entrada	Saída	Potência média dissipada (W)
0->0	**1->1**	**3.6825E-10**
1->1	**0->0**	**7.1430E-14**

Para usar estas informações na estimativa do consumo de energia de um circuito, os valores obtidos das duas tabelas apresentadas acima são transportados para um arquivo que contém a descrição VHDL da porta inversora.

```
entity Inversor is
	port(  saida: out STD_LOGIC;
		entrada: in STD_LOGIC);
end Inversor;

architecture Inversor of Inversor is
begin
	process(entrada)
		variable entradaAnterior: STD_LOGIC := '0';
		variable energiaDaPortaComTransicao: REAL := 0.0;
		variable energiaTransicao: REAL;
		variable potenciaSemTransicao: REAL:= 0.0;
		variable transition: STD_LOGIC_VECTOR(1 downto 0);
	begin
		transition := entradaAnterior & entrada;
		energiaTransicao := 0.0;
		case transition is
			when "01" => energiaTransicao := 4.5964E-13;
			when "10" => energiaTransicao := 4.9066E-13;
			when others =>
		end case;
		energiaDaPortaComTransicao := energiaDaPortaComTransicao + energiaTransicao;
		energiaTotalComTransicao := energiaTotalComTransicao + energiaTransicao;
		potenciaTotalSemTransicao := potenciaTotalSemTransicao - potenciaSemTransicao;
		transition := (entrada & entrada);
		case transition is
			when "00" =>  potenciaSemTransicao:= 3.6825E-10;
			when "11" =>  potenciaSemTransicao:= 7.1430E-14;
			when others =>
		end case;
		potenciaTotalSemTransicao := potenciaTotalSemTransicao + potenciaSemTransicao;
		saida <= not entrada;
		entradaAnterior := entrada;
	end process;
end Inversor;
```

Figura 9.4: Caracterização do comportamento e consumo de energia de uma porta inversora. As variáveis **energiaDinamicaDaPorta** e **energiaDinamicaTotal** armazenam o consumo de energia dinâmica da porta e de todo o circuito, respectivamente. A variável **potenciaTotalSemTransicao** armazena a potência dissipada por todo o circuito durante o tempo de simulação. **energiaDinamicaTransicao** e **potenciaTotalSemTransicao** são variáveis que armazenam os valores apresentados na Tabela 9.1 e na Tabela 9.2.

Este arquivo, ilustrado na Figura 9.4, contém informações de comportamento da porta e uma lógica que permite ao simulador saber qual valor de energia deve ser computado frente à transição ocorrida. Este arquivo faz parte de uma biblioteca de portas lógicas, todas estas caracterizadas de acordo com a tecnologia alvo.

Dado uma porta anotada, tal como descrito na Figura 9.4, esta fará parte de uma biblioteca VHDL de portas lógicas. Na Figura 9.5 estão apresentadas as variáveis globais e o corpo da arquitetura que calcula o consumo de energia total do circuito

simulado.

```
package POWER_PCK is
	shared variable energiaTotalComTransicao: REAL := 0.0;
	shared variable potenciaTotalSemTransicao: REAL := 0.0;
	shared variable energiaTotalSemTransicao: REAL := 0.0;
	constant BASE_TEMPO: TIME := 100ps;
	constant BASE_TEMPO_REAL: REAL := 1.0E-10;
end POWER_PCK;

library IEEE;
use IEEE.std_logic_1164.all;
use work.POWER_PCK.all;

entity PowerEstimation is
end PowerEstimation;

architecture PowerEstimation of PowerEstimation is
	signal step: STD_LOGIC;
begin
	process
	begin
		step <= '0', '1' after BASE_TEMPO / 2;
		wait for BASE_TEMPO;
	end process;
	process(step)
		variable ciclos: NATURAL := 0;
		variable energiaTotal: REAL;
		variable energiaTotalST: REAL;
	begin
		ciclos := ciclos + 1;
		energiaTotalST = potenciaTotalSemTransicao * BASE_TEMPO_REAL;
		energiaTotalSemTransicao := energiaTotalSemTransicao + energiaTotalST;
		energiaTotal := energiaTotalComTransicao + energiaTotalSemTransicao;
	end process;
end PowerEstimation;
```

Figura 9.5: Par entidade-arquitetura principal para cálculo do consumo de energia (**PowerEstimation**) e pacote com variáveis globais (**POWER_PCK**). As variáveis declaradas como *shared* têm escopo global a todas as arquiteturas simuláveis, ou seja, são variáveis globais do simulador. O consumo de energia total do circuito é computado na variável **energiaTotal**.

A cada transição de um sinal de entrada de uma porta lógica, o valor correspondente de consumo de energia é contabilizado em variáveis globais que armazenam o consumo de energia de todo o circuito devido a transições. Também é contabilizado o período de ausência de transição nas entradas das portas, para que a dissipação de potência seja considerada durante todo o período de simulação. Ao final da simulação, têm-se como resultados o tempo de simulação e a energia total consumida em cada porta lógica e em todo o circuito.

Para exemplificar o uso da técnica, a Figura 9.6 apresenta o cálculo do consumo de energia de um inversor, cuja entrada é um relógio com freqüência de 50 MHz. Nota-se nesta Figura uma característica importante: o uso da técnica é praticamente transparente para a descrição VHDL, pois o projetista necessita apenas incluir o pacote de potência **POWER_PCK** e a entidade **PowerEstimation**. O resto da descrição se mantém inalterada. Em caso de ausência desta entidade, ou caso esta esteja incompleta (não contenha todas as portas usadas no circuito), o comportamento do circuito não será alterado, pois será suprido pelas portas da biblioteca default. A única conseqüência desta omissão será a não contabilização do consumo de energia das portas não caracterizadas.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.POWER_PCK.all;

entity CircuitoInversor is
end;

architecture CircuitoInversor of CircuitoInversor is
	component Inversor is
		port(  saida: out STD_LOGIC;
			entrada: in STD_LOGIC);
	end component;
	signal excitacao, resposta: STD_LOGIC;
begin
	process
	begin
		excitacao <= '1', '0' after 10ns;
		wait for 20ns;
	end process;
	P: entity work.PowerEstimation;
	I: Inversor port map(saida=>resposta, entrada=>excitacao);
end CircuitoInversor;
```

Figura 9.6: Exemplo de simulação do comportamento e consumo de energia do circuito **CircuitoInversor**. Este circuito tem apenas uma porta lógica - um inversor - que faz parte do pacote de potência e está descrita no par entidade-arquitetura da Figura 9.4.

A Tabela 9.3 apresenta o resultado da simulação VHDL comparado com o resultado da simulação SPICE. Pode se notar que para um intervalo de 2000 ns, correspondendo a 100 transições do circuito descrito na Figura 9.6, o erro encontrado entre as duas simulações foi menor que 0.1%.

Tabela 9.3: Comparação do cálculo de consumo de energia para uma porta inversora entre a simulação SPICE e a simulação VHDL. A Tabela mostra um erro menor que 0.1% entre as simulações que ocorreram durante um intervalo de 2000 ns. Neste intervalo, a porta inversora foi estimulada por um relógio que gerou 100 variações na entrada.

Intervalo (ns)		Consumo de energia (J)		ERRO (%)
Início	Fim	Simulador HSPICE	Simulador VHDL	
0	**2000**	**8.2274E-11**	**8.2206E-11**	**0.0827%**

A Tabela 9.3 mostra que o erro de estimativa para uma única porta é insignificante. Este erro permanece praticamente constante para qualquer intervalo de simulação. Contudo, para circuitos com muitas portas lógicas, o erro aumenta, aumentando o grau de incerteza das estimativas. Por exemplo, para o circuito de controle de um roteador, o erro observado em 10 μs de uma simulação é em torno de 25%.

9.2 Método para Síntese e Validação da Aplicação

A Seção anterior mostra como é avaliado o consumo de energia de uma rede intrachip, mas não entra em detalhes sobre o tráfego ao qual a rede será submetida. Para tanto é necessário avaliar o consumo de energia da rede tendo como estímulo as mensagens provenientes dos núcleos da aplicação. A associação das mensagens com o mapeamento dos núcleos é que representa o tráfego na rede intrachip. Para obter este tráfego, o formato interno que descreve a aplicação, tipicamente um grafo CDCG, é sintetizado pelo ambiente CAFES em uma descrição VHDL comportamental. Esta

descrição, mais a descrição VHDL estrutural da NoC e a biblioteca de potência com portas caracterizadas compõem a descrição VHDL completa para estimar o consumo de energia e o tempo de execução da aplicação, tal como está ilustrado na Figura 9.7.

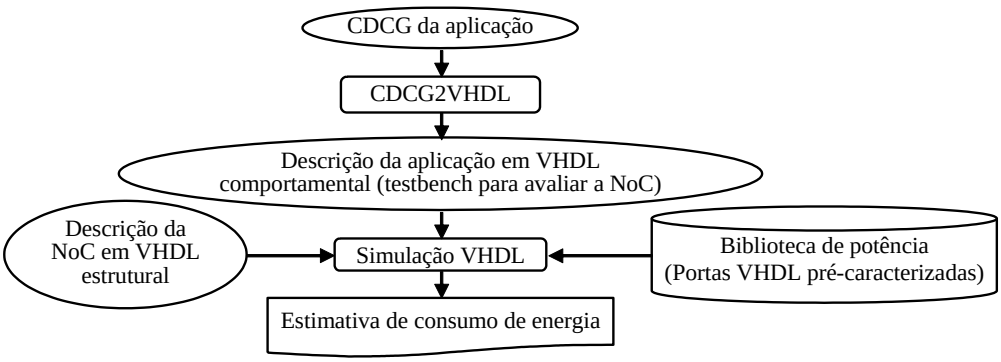
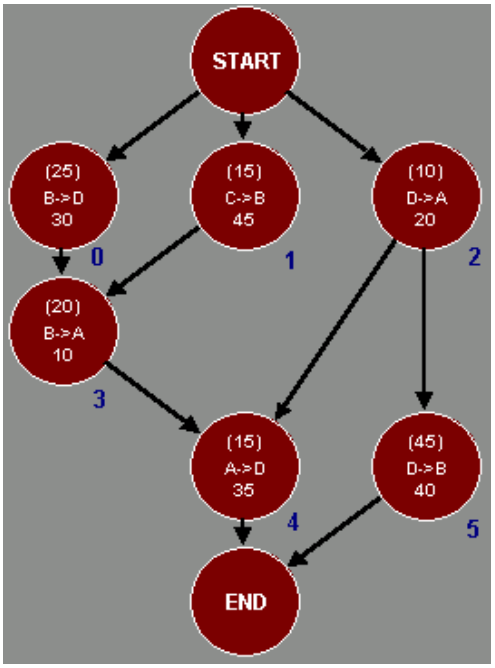


Figura 9.7: Fluxo para estimativa do consumo de energia de NoCs, tendo como estímulo uma descrição comportamental da aplicação. As portas da biblioteca estão caracterizadas com valores referentes ao consumo de energia para todas as combinações de suas entradas.

A síntese da aplicação para um VHDL comportamental é realizada de forma semi-automática com o auxílio da ferramenta CDCG2VHDL, que tem como entrada a descrição textual do CDCG da aplicação. A Figura 9.8(a) ilustra o CDCG de uma aplicação sintética. Esta mesma aplicação está descrita textualmente na Figura 9.8(b). A descrição textual, quando obtida após serem executados os algoritmos de mapeamento, pode estar associada aos núcleos com os tiles da rede intrachip, indicado pelo comentário (**#_CDCG2NoC_Mapping**). Para exemplificar, no mapeamento da Figura 9.8(b), que foi obtido com uma rede malha com 2 linhas e 2 colunas, o núcleo C está localizado no tile [0, 1] (linha 0 e coluna 1).




```
#_CDCG_Graphic (Posição XY de cada vértice)
START   200     100
END     200     400
0       100     175
1       200     175
2       300     175
3       100     250
4       200     325
5       300     325

#_CDCG_Vertices (Conteúdo de cada vértice)
0       B - D   30 : 25
1       C - B   45 : 15
2       D - A   20 : 10
3       B - A   10 : 20
4       A - D   35 : 15
5       D - B   40 : 45

#_CDCG_Edges (lista de dependências)
START 1 0 2
END
0 3
1 3
2 4 5
3 4
4 END
5 END

#_CDCG2NoC_Mapping (linhas e colunas)
B C
D A
```

(a) (b)

Figura 9.8: (a) apresenta o CDCG de uma aplicação sintética com 6 vértices, representando as comunicações e 10 arestas, representando as dependências. (b) apresenta o mesmo CDCG descrito de forma textual e mais um mapeamento em uma NoC 2 × 2.

Aplicando a ferramenta CDCG2VHDL sobre o circuito descrito na Figura 9.8 gera-se o VHDL comportamental ilustrado na Figura 9.9. Nele é possível notar que cada núcleo do CDCG é convertido em um processo para que todas as mensagens, provenientes de núcleos distintos possam ser concorrentes. Por outro lado, a dependência é fornecida pelas variáveis que indicam se uma mensagem foi ou não enviada.

```
B_00: process
       variable numPhits : integer;
begin
       loop
              wait until reset = '0';
              enviou_B_D_0 <= false;
              Computacao(25ns);
              Comunicacao(LinhaDestino:=1, ColunaDestino:=0, numPhits:=30);
              enviou_B_D_0 <= true;

              wait until enviou_B_D_1 = true and enviou_C_B_2 = true;
              enviou_B_A_4 <= false;
              Computacao(20ns);
              Comunicacao(LinhaDestino:=1, ColunaDestino:=1, numPhits:=10);
              enviou_B_A_4 <= true;
       end loop;
end process;

C_01: process
       variable numPhits : integer;
begin
       loop
              wait until reset ='0';
              enviou_C_B_2 <= false;
              Computacao(15ns);
              Comunicacao(LinhaDestino:=0, ColunaDestino:=0, numPhits:=45);
              enviou_C_B_2 <= true;
       end loop;
end process;

D_10: process
       variable numPhits : integer;
begin
       loop
              wait until reset ='0';
              enviou_D_A_3 <= false;
              Computacao(10ns);
              Comunicacao(LinhaDestino:=1, ColunaDestino:=1, numPhits:=20);
              enviou_D_A_3 <= true;

              wait until enviou_D_A_3 = false;
              enviou_D_B_6 <= false;
              Computacao(45ns);
              Comunicacao(LinhaDestino:=0, ColunaDestino:=0, numPhits:=40);
              enviou_D_B_6 <= true;
       end loop;
end process;

A_11: process
       variable numPhits : integer;
begin
       loop
              wait until reset ='0';
              wait until enviou_BD_1 = false and enviou_CB_2 = false;
              enviou_AD_5 <= false;
              Computacao(15ns);
              Comunicacao(LinhaDestino:=1, ColunaDestino:=0, numPhits:=35);
              enviou_AD_5 <= true;
       end loop;
end process;
```

Figura 9.9: Descrição parcial do VHDL comportamental que implementa um CDCG. Cada processo representa um núcleo do CDCG e as dependências entre vértices do CDCG são implementadas pelos sinais que informam o envio de mensagens.

Para compreender melhor a relação entre a Figura 9.8 e a Figura 9.9, exemplifica-se aqui a conversão do vértice **3** da Figura 9.8. A mensagem contida nele depende das mensagens contidas nos vértices **0** e **1**. O processo que implementa o envio da mensagem contida no vértice **3** é representado na Figura 9.9 por **B_00**, indicando que o núcleo B está mapeado no tile de linha e coluna 0, e as dependências do vértice **3** são dadas pelos sinais **enviou_BD_1** e **enviou_CB_2**. Quando estas dependências são resolvidas, é executada a computação de 20 ns descrita no vértice **3**, para posteriormente efetuar-se a comunicação. Esta, por sua vez, tem parâmetros (LinhaDestino:=1, ColunaDestino:=1, numPhits:=10), indicando que tem uma comunicação de 10 phits do núcleo **B**, que está mapeado no tile [0, 0], para o núcleo **A**, que está mapeado no tile [1, 1]. Uma vez enviada a mensagem contida no vértice **3** a variável **enviou_BD_1** é validada (recebe *true*), indicando que a mensagem foi enviada e liberando as dependências desta.

Todas as variáveis que informam o envio de mensagens (e.g. **enviou_BD_1** e **enviou_CB_2**) são implementadas como variáveis compartilhadas (declaradas como *shared*) para terem escopo global a todos os processos que implementam os núcleos do CDCG.

Computacao, descrita na Figura 9.9 é um procedimento comportamental descrito em VHDL, que é implementado com uma simples instrução de espera. Assim, **Computacao**(10 ns) gera código para o processo simulado aguardar por 10 ns. **Comunicacao**, por sua vez, é um pseudo-código, inserido na Figura 9.9 para facilitar a compreensão. A implementação da **Comunicacao**(LinhaDestino:=1, ColunaDestino:=1, numPhits:=10) pode ser observada na Figura 9.10. O pacote de comunicação é gerado conforme o protocolo implementado na NoC Hermes, que é composto por três flits iniciais com o endereço destino, tamanho e endereço fonte, seguidos dos flits de dados.

```
tx(N00) <= '1';
data_out(N00) <= x"01";  -- Destino
wait until credit_i(N00) = '1' and clock_tx(N00) = '1';
wait until clock_tx(N00) = '0';

data_out(N00) <= x"1c";  -- Tamanho
wait until credit_i(N00) = '1' and clock_tx(N00) = '1';
wait until clock_tx(N00) ='0';

data_out(N00) <= x"01";  -- Origem
wait until credit_i(N00) = '1' and clock_tx(N00) = '1';
wait until clock_tx(N00) = '0';

numPhits := 27;
while numPhits>0 loop
       numPhits := numPhits - 1;
       data_out(N00) <= CONV_STD_LOGIC_VECTOR(numPhits, 8);
       wait until credit_i(N00) = '1' and clock_tx(N00) = '1';
       wait until clock_tx(N00) = '0';
end loop;
tx(N00) <= '0';
```

Figura 9.10: Implementação do procedimento **Comunicacao**. Representado um pacote com 30 flits enviado do núcleo mapeado no tile [0, 0] para o núcleo mapeado no tile [0, 1].

Uma vez gerado o comportamento da comunicação da aplicação, tal como exemplificado na Figura 9.9, este é inserido manualmente no arquivo VHDL de teste da NoC, servindo como padrão de tráfego para estimar o consumo de energia da NoC.

9.3 Avaliação e Validação do Consumo de Energia em Alto Nível

As seções anteriores mostram como é estimado o consumo de energia para descrições em VHDL. O método utilizado apresenta bons resultados quando comparado com simulações em nível elétrico, devido aos modelos utilizados e devido às portas VHDL terem sido pré-caracterizadas com as estimativas de energia obtidas pelos simuladores elétricos. Todavia, o tempo necessário para executar todo este processo torna proibitivo utilizá-lo como estimativa de consumo de energia durante os *algoritmos de mapeamento*[37], devido ao grande número de iterações necessárias.

A abordagem adotada utiliza os modelos de consumo de energia descritos no Capítulo 8 para estimar o consumo de energia de cada mapeamento durante a execução algorítmica. Esta estimativa é aqui chamada de estimativa de consumo de energia em alto nível. Nesta, o consumo de energia de uma rede intrachip é estimado pelo tráfego de bits relacionado com os parâmetros de energia E_{Bbit}, E_{Sbit}, E_{Lbit} e E_{Cbit}. Para que as estimativas de alto nível sejam próximas às estimativas de mais baixo nível é imprescindível que estes parâmetros tenham valores coerentes com o efeito da comunicação sobre o modelo de consumo de energia da rede intrachip. Assim, o problema ilustrado nesta Seção é a extração e refinamento dos parâmetros de energia.

A técnica adotada aqui, e ilustrada na Figura 9.11, possui três conjuntos de atividades. Ela é na verdade um processo de refinamento manual, sendo que os parâmetros partem com uma estimativa inicial, seguem-se refinamentos sucessivos (laços de tentativa e erro) até que sejam alcançados resultados satisfatórios (erro aceitável).

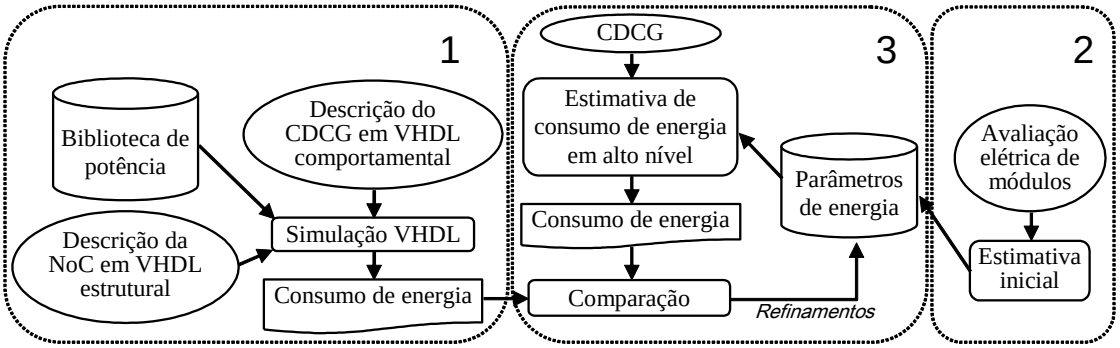


Figura 9.11: Fluxo de atividades para extração e refinamento de parâmetros dos modelos de consumo de energia utilizados nas avaliações de alto nível do *framework* CAFES.

O primeiro conjunto de atividades descrito na Figura 9.11 refere-se à simulação VHDL da rede intrachip, cujas portas estão caracterizadas. Esta atividade serve para obter o valor de consumo de energia considerado padrão, que será utilizado para refinar as estimativas de alto nível através de sucessivas comparações. Este conjunto de atividades está descrito na Seção 9.2.

O segundo conjunto de atividades tem por objetivo a geração da primeira

[37] A solução exaustiva do problema de mapeamento implica *n!* estimativas de consumo de energia, sendo *n* o número de núcleos da aplicação. Soluções heurísticas reduzem em muito o número de estimativas, mas não o suficiente para utilizar técnicas de estimativa de consumo de energia que gastem muito tempo de processamento.

estimativa de valores para os parâmetros elétricos (*EBbit*, *ESbit*, *ELbit* e *ECbit*). É uma estimativa grosseira obtida pela relação de quanto cada parâmetro influi no cálculo do consumo de energia total. Ou seja, o peso de cada parâmetro no cômputo total do consumo de energia. Para verificar o peso de cada parâmetro, estes são inicialmente associados a módulos da NoC. Por exemplo, o parâmetro *EBbit* é associado aos circuitos de armazenamento temporário, enquanto que o *ESbit* é associado aos circuitos de controle de roteamento. Posteriormente, no simulador VHDL, configurações de NoCs são estimuladas com diversos tráfegos e as medidas de consumo de energia de cada módulo e total da NoC são armazenadas. Estas medidas permitem extrair uma estimativa do consumo de energia de um bit trafegando através de cada um destes módulos, definindo assim as primeiras estimativas de *EBbit*, *ESbit*, *ELbit* e *ECbit*. O gráfico da Figura 9.12 ilustra estas medidas obtidas para diversas simulações de tráfego e topologias para a NoC Hermes. Como pode ser observado, quando é considerado o número de transições, existe uma significativa variação para os parâmetros *EBbit*, *ELbit* e *ECbit*. Todavia, como *certos modelos da aplicação*[38] abstraem o efeito da transição de bits, é interessante ter mais de um conjunto de parâmetros iniciais para que estes permitam estimar o consumo de energia da aplicação com maior precisão.

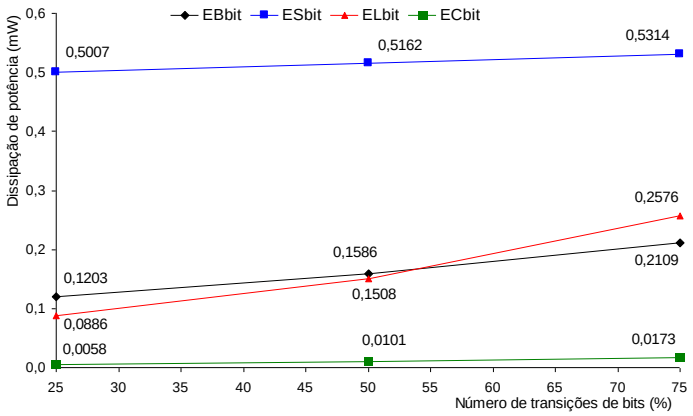


Figura 9.12: Gráfico de dissipação de potência frente o número de chaveamentos, extraído de simulações de módulos da NoC Hermes. Os valores foram obtidos a partir de aplicações sintéticas com tráfegos cujas transições entre bits consecutivos variavam de 25%, 50% e 75% do volume total de bits transmitido. Foram utilizadas NoCs 2 × 3 com tamanhos de phits e *buffers* variando entre 8 e 16. Cada ponto do gráfico representa uma média destas simulações.

O gráfico da Figura 9.12 requer uma breve análise, para que seja melhor compreendida a forma como os parâmetros são utilizados nos algoritmos de estimativa de consumo de energia. Considerando uma aplicação com uma média de 25% transições, a estimativa inicial para os parâmetros seria: *ESbit* = 500 μW, *EBbit* = 120 μW, *ELbit* = 89 μW e *ECbit* = 6 μW. Estes valores dão a noção que o consumo de energia do circuito de controle, modelado por *ESbit*, é o maior de todos. Todavia, existe apenas um circuito de controle para cinco canais de entrada e saída, e o consumo deste circuito pouco altera com a quantidade de canais simultâneos que devem ser roteados. Ou seja, o consumo de energia do controle não é linear com relação ao

[38] A maior parte dos modelos da aplicação apresentados neste trabalho abstraem o efeito das transições na comunicação. Na verdade, o único modelo que considera este efeito é o ECWM.

número de bits que chega ao roteador, a menos que estes bits não sejam simultâneos. Este fato faz com que os demais parâmetros possam afetar o consumo de energia até 5 vezes mais, no caso do circuito de armazenamento temporário e até 10 vezes mais no caso das conexões entre roteadores e entre roteador e núcleo local. Além disto, para cada canal de entrada existe uma fila de elementos de memorização, fazendo que o consumo de energia do circuito de armazenamento temporário supere em muito o consumo de energia do circuito de controle. Para modelar este efeito, independente do número de comunicações concorrentes, o algoritmo que implementa a NoC contabiliza apenas um tráfego para o circuito de controle, enquanto que para os demais circuitos todos os bits são contabilizados.

O terceiro e último conjunto de atividades consiste na etapa de refinamento dos parâmetros iniciais. Para tanto, aplicações sintéticas foram executadas sobre diferentes topologias de NoC, mais especificamente, NoCs 2×3 e 3×3. O refinamento é obtido pela comparação da estimativa de consumo de energia obtida com o CAFES frente à estimativa obtida com a simulação VHDL. O processo é realizado através de sucessivas tentativas e erros, com a intenção de alterar os parâmetros para que estes conduzam as estimativas de alto nível que levem a resultados semelhantes aos obtidos com as simulações VHDL.

Nesta etapa, chegou-se a duas conclusões importantes: (*i*) o segundo conjunto de atividades conduz a estimativas iniciais pouco razoáveis para os parâmetros de energia, necessitando de algumas etapas de refinamento para que se obtivessem parâmetros que conduzissem a melhores resultados; e (*ii*) o refinamento para uma determinada aplicação não leva necessariamente ao melhor refinamento para outra aplicação. Assim, para obter estimativas precisas, para cada aplicação é necessário um novo refinamento.

Enquanto a primeira conclusão apenas implica um tempo maior de refinamento, não sendo um problema de projeto, a segunda é problemática, pois não permite parametrizar a NoC, sendo necessário levar em consideração a aplicação e o modelo subjacente a sua descrição. Assim, chega-se a uma terceira conclusão, os modelos de consumo de energia aqui adotados, e que são adotados por diversos autores, devem ser refinados. Esta atividade, contudo não será foco deste trabalho, mas de trabalhos futuros.

10 O FRAMEWORK CAFES

Este Capítulo tem por objetivo apresentar um *framework* proposto e desenvolvido pelo Autor (MARCON et al., 2005d) para avaliar os formatos internos apresentados no Capítulo 5 e os modelos computacionais subjacentes a estes. O *framework* é denominado CAFES (em inglês, *Communication Analysis For Embedded Systems*), devido a este ser originalmente uma ferramenta planejada para analisar a comunicação de aplicações embarcadas. Mais especificamente, na sua concepção, CAFES era uma ferramenta planejada para comparar a qualidade de estimativas de consumo de energia de aplicações executando sobre uma infra-estrutura de comunicação. Os primeiros modelos implementados foram o CWM e o CDM, e a infra-estrutura de comunicação uma NoC com topologia malha, roteamento XY e chaveamento wormhole. Posteriormente, a ferramenta passou a permitir a inserção de novos modelos computacionais, tal como o CDCM e o ECWM, e novas topologias de rede, tal como toro dobrado, ampliando o espectro de modelagem.

10.1 Interfaces e Recursos do Framework CAFES

Na atual versão (3.7), CAFES permite comparar e inserir novos modelos, analisar várias estratégias de mapeamento de aplicações em diferentes topologias de rede intrachip, simular aplicações para avaliar o seu comportamento e estimar o tempo e o consumo de energia devido à comunicação. Além do mais, o *framework* assume como infra-estrutura de comunicação uma NoC parametrizável, permitindo definir várias características, tais como a largura do canal de comunicação entre roteadores, e entre roteador e núcleo local, e a dimensão das áreas de armazenamento temporário de cada roteador. Os parâmetros da infra-estrutura de comunicação são utilizados nas Equações descritas no Capítulo 8 para cálculo do custo de cada mapeamento. Estes parâmetros, mostrados na Figura 10.1, são divididos em três grupos: (*i*) parâmetros de topologia da NoC, (*iii*) parâmetros de tempo, e (*ii*) parâmetros de consumo de energia da NoC.

A Figura 10.1 ilustra a interface de abertura do *framework* CAFES. Esta interface mostra que o projetista pode avaliar mapeamentos através de um dos modelos básicos descritos no Capítulo 5: CWM, ECWM, CDM, CDCM, ACPM e CTM.

Os parâmetros implementados na versão atual do *framework* estão descritos a seguir. O primeiro conjunto de itens descreve os parâmetros topológicos:

1. ***NoC size (lines . columns)*** – Número de tiles da NoC em linhas e colunas;
2. ***Tile size (width . height)*** – Largura versus altura do tile em mm^2. Permite estimar o comprimento das conexões entre roteadores;
3. ***Buffer length*** - Número de elementos que compõem as áreas de

armazenamento temporário das entradas de cada roteador. A largura de cada elemento da área de armazenamento temporário é de um phit;

4. ***Topology*** - Topologias cujos modelos estão disponíveis no *framework*:
   i. ***Mesh*** - Topologia malha, roteamento XY e chaveamento wormhole;
   ii. ***Torus*** - Topologia toro, roteamento XY e chaveamento wormhole.

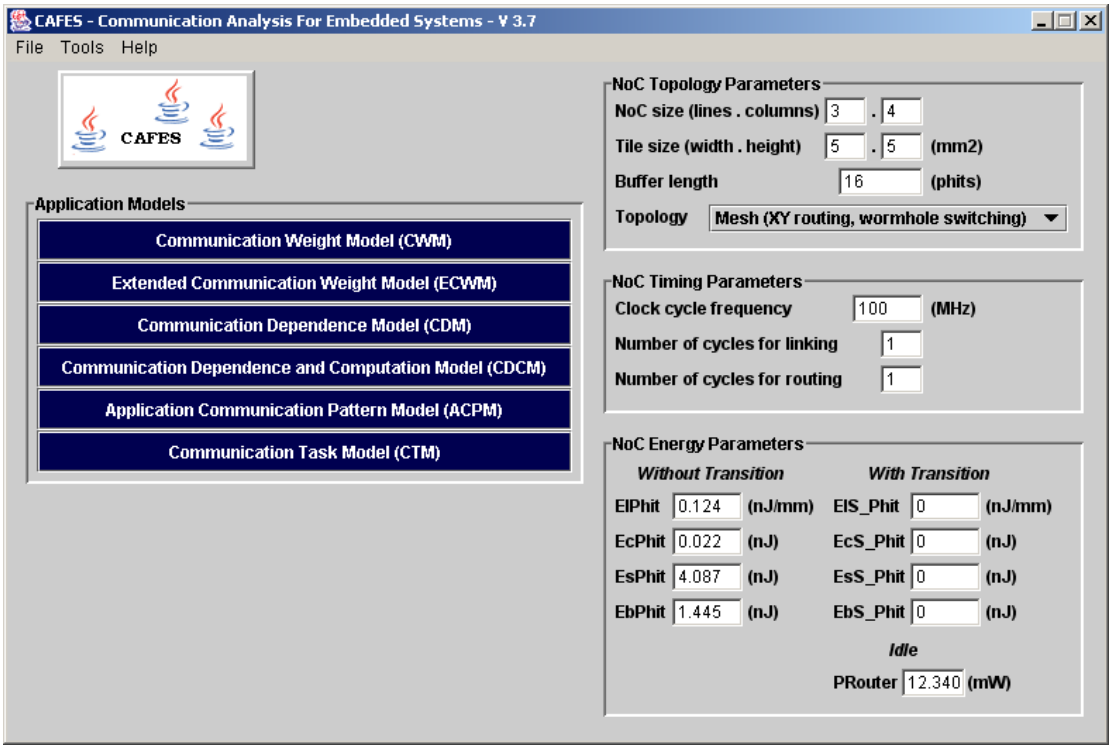


Figura 10.1: Interface de abertura do *framework* CAFES. Esta é dividida em quatro áreas: (*i*) *Application Models* – local onde o usuário pode selecionar um dentre diferentes modelos para descrever e avaliar aplicações; (*ii*) *NoC Topology Parameters* – local onde estão as parametrizações aceitas para modelar diferentes topologias de NoC; (*iii*) *NoC Timing Parameters* – permite que sejam inseridos os parâmetros para estimar o tempo de execução de aplicações; e (*iv*) *NoC Energy Parameters* – informa os parâmetros de energia utilizados pelos modelos da rede intrachip.

O segundo conjunto de itens descreve os parâmetros que permitem modelar o tempo de execução de aplicações:

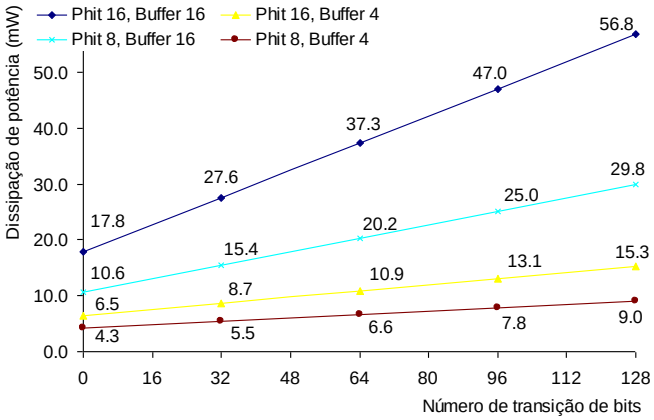
1. ***Clock cycle frequency*** – Freqüência de operação do relógio da NoC, utilizado para determinar o tempo de execução da aplicação e o consumo de energia estática de toda rede e dinâmica dos circuitos que operam mesmo frente a ausência de tráfego;
2. ***Number of cycles for linking*** - Número de ciclos de relógio necessários para realizar a transmissão de um phit de uma conexão entre roteadores ou entre roteador e núcleo local;
3. ***Number of cycles for routing*** - Número de ciclos de relógio para rotear um pacote de um canal de entrada para um canal de saída de um roteador.

Por fim, estão descritos os elementos do conjunto de parâmetros que permitem

estimar o consumo de energia de redes intrachip (*Cabe aqui salientar que embora os modelos de energia sejam descritos com relação à transmissão de bits, o framework CAFES adota como unidade mínima o phit, para simplificar as operações internas. Por este motivo, todos os modelos têm como entrada o número de phits da comunicação e não o número de bits*):

1. ***Without Transitions*** – Parâmetros de consumo de energia de um phit. Para modelos que consideram transição de bits, estes campos representam os consumos de energia na ausência de transições entre os phits consecutivos. Para os demais modelos, estes campos representam consumos médios de energia para phits:
   i. ***ELPhit*** - Energia dinâmica de um phit consumida em uma conexão entre roteadores. Equivale ao ***ELbit*** multiplicado pelo número de bits de um phit;
   ii. ***ECPhit*** - Energia dinâmica de um phit consumida em uma conexão entre um roteador e um núcleo local ao tile. Equivale ao ***ECbit*** multiplicado pelo número de bits de um phit;
   iii. ***ESPhit*** - Energia dinâmica de um phit consumida no circuito de controle de um roteador. Equivale ao ***ESbit*** multiplicado pelo número de bits de um phit;
   iv. ***EBPhit*** - Energia dinâmica de um phit consumida por um elemento do circuito de armazenamento. Equivale ao ***EBbit*** multiplicado pelo número de bits de um phit.
2. ***With Transitions*** – Parâmetros de consumo de energia para transições. Usado apenas nos modelos que consideram transição de bits. Representa os consumos de energia quando ocorrer *transição em todos os bits*[39] de phits consecutivos:

[39] Os valores de consumo de energia são extraídos considerando a transição de todos os bits entre phits consecutivos. Este não é o caso prático, mas simulações SPICE para diversos tamanhos de phit mostraram que o consumo de energia é direta e linearmente proporcional ao número de transição de bits, tal como ilustrado abaixo. Assim, o algoritmo de energia aplica uma regra linear para chaveamentos de bits não múltiplos do tamanho do phit.



A Figura acima descreve a variação da dissipação de potência frente ao número de transições de bits. Para avaliação são utilizados circuitos de armazenamento temporário, com profundidade 4 ou 16 e largura de phit 8 ou 16. Estes circuitos foram extraídos do VHDL da NoC Hermes (MORAES, 2004).

i. ***ELS_Phit*** - Equivale a ***ELPhit***, considerando agora transições;

ii. ***ECS_Phit*** - Equivale a ***ECPhit***, considerando agora transições;

iii. ***ESS_Phit*** - Equivale a ***ESPhit***, considerando agora transições;

iv. ***EBS_Phit*** - Equivale a ***EBPhit***, considerando agora transições.

3. ***PRouter*** – Potência estática dissipada por todos os elementos ativos de um roteador, somado com a potência dinâmica dissipada pelos circuitos do roteador que operam mesmo frente à ausência de tráfego.

Uma vez escolhido o formato interno para descrever a aplicação e avaliar o mapeamento, o *framework* fornece novas interfaces onde o projetista pode descrever graficamente a aplicação através do formato interno de cada modelo. A Figura 10.2 ilustra uma aplicação sintética descrita pela sua computação e comunicação com o formato interno CDCG (ver Definição 4 no Capítulo 5). Esta aplicação tem quatro núcleos (N), seis mensagens (M) e nove dependências (D). Devido ao CAFES estar escrito em inglês, os vértices especiais INÍCIO e FIM, das definições 3, 4, 5 e 6 do capítulo 5, são referenciados como START e END, respectivamente:

$N = \{A, B, C, D\}$

$M = \{(1, A, B, 10, 250), (2, C, D, 12, 130), (3, B, D, 25, 145), (4, B, C, 20, 450), (5, D, A, 15, 180), (6, D, B, 5, 200)\}$

$D = \{(m_1, \text{START}), (m_2, \text{START}), (m_3, m_1), (m_4, m_1), (m_4, m_2), (m_5, m_2), (m_6, m_4), (m_6, m_5), (\text{END}, m_3), (\text{END}, m_6)\}$

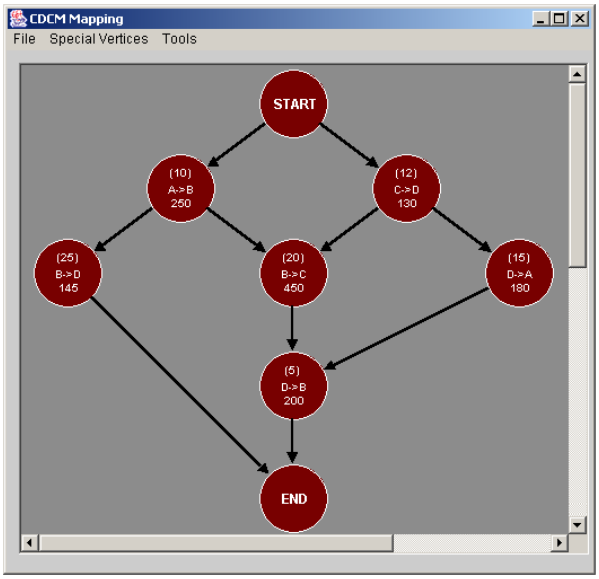
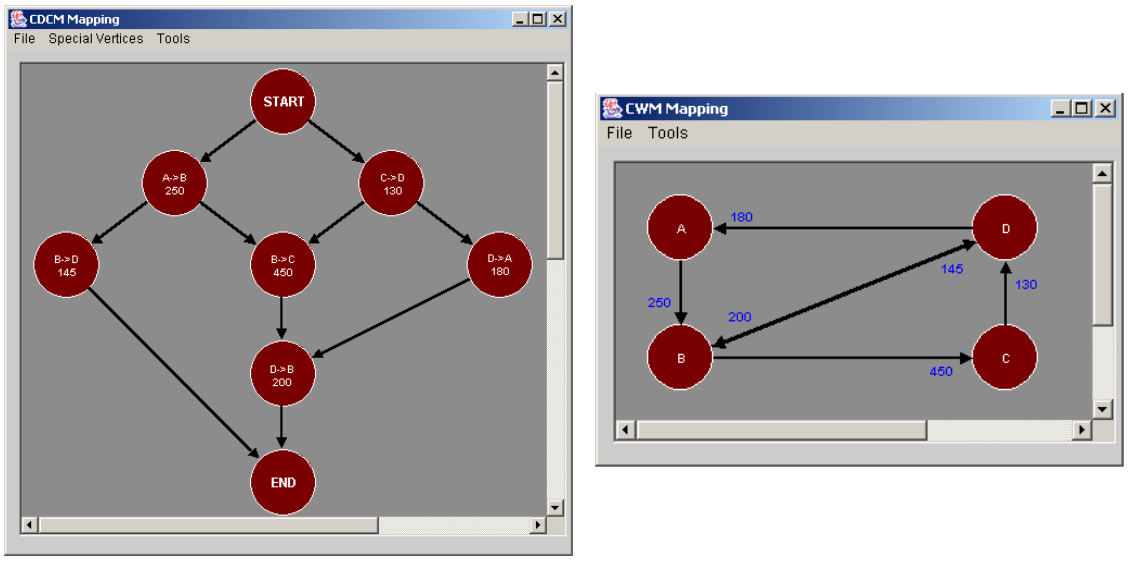


Figura 10.2: CDCG de uma aplicação sintética com quatro núcleos e seis mensagens.

O projetista pode descrever a mesma aplicação com os demais modelos no *framework* CAFES. Dado que certos modelos podem ser extraídos de outros modelos mais complexos, algumas descrições podem ser obtidas automaticamente a partir de outras pré-existentes. Este é o caso dos formatos internos CDG e CWG. O CDG pode ser obtido automaticamente do formato interno CDCG, retirando apenas a computação de cada vértice, tal como ilustrado na Figura 10.3(a). Este foi extraído automaticamente do CDCG descrito na Figura 10.2. Devido à simplicidade do modelo, o CWG pode ser obtido de qualquer um dos outros formatos internos apresentados no Capítulo 5, pela soma de todos os bits que são enviados de um núcleo para outro. O CWG extraído a

partir do CDCG da Figura 10.2, ou do CDG da Figura 10.3(a) está ilustrado na Figura 10.3(b).



(a) (b)

Figura 10.3: CDG (a) e CWG (b) extraídos automaticamente do CDCG da Figura 10.2.

Para o formato interno CDCG, CAFES permite analisar caminhos críticos e estimar o tempo de execução da aplicação, desconsiderando os atrasos causados pela infra-estrutura de comunicação. Esta estimativa é interessante para avaliar o quanto a infra-estrutura de comunicação influi no tempo total de execução da aplicação. A Figura 10.4 apresenta três caminhos críticos computados com base na comunicação e na computação de um CDCG: um caminho calcula apenas o caminho crítico devido à computação, outro apresenta o caminho crítico devido à comunicação e o terceiro mostra o efeito conjunto da comunicação e da computação.

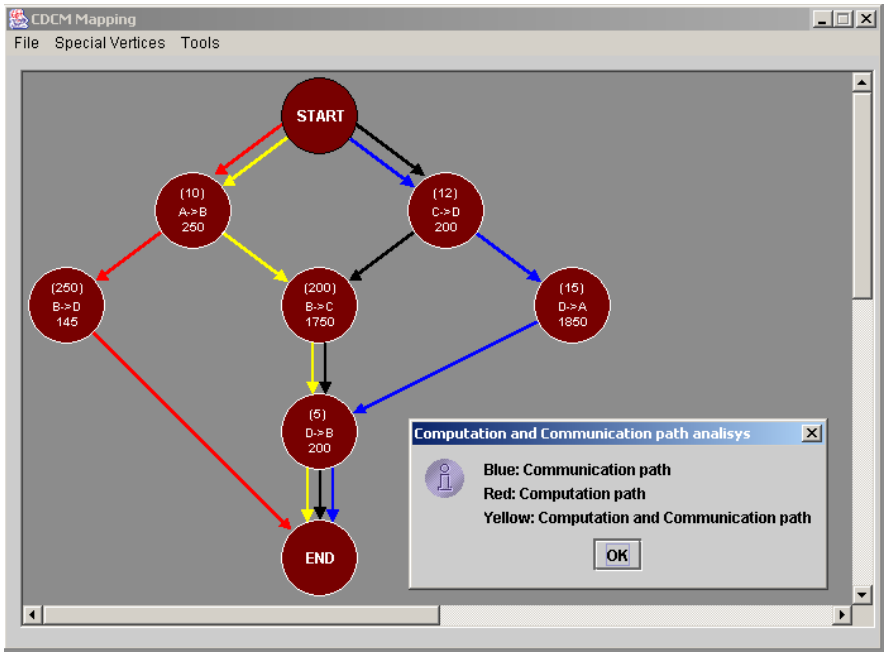


Figura 10.4: Avaliação dos caminhos críticos da computação e da comunicação de uma aplicação modelada com CDCM. Esta Figura é semelhante à Figura

10.2, porém com valores de computação e comunicação alterados para ilustrar diferentes caminhos críticos.

Uma vez descrita a aplicação a ser avaliada conforme o modelo escolhido, o projetista pode requisitar que seja avaliado o mapeamento de menor custo. CAFES encontra mapeamentos que reduzem o consumo de energia e o tempo total de execução da aplicação considerando os parâmetros de energia e tempo fornecidos pelo projetista. A Figura 10.5 apresenta o resultado de um mapeamento com o cálculo da energia consumida nos canais e roteadores de uma NoC 2 × 2. A aplicação que resultou a Figura 10.5 está descrita na Figura 10.2.

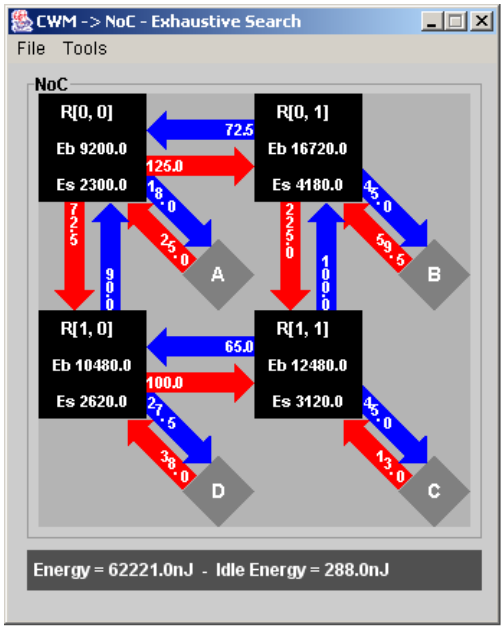


Figura 10.5: Descrição de uma NoC 2 × 2 de topologia malha, com a especificação da energia consumida em cada canal e cada roteador. A Figura ilustra também o consumo de energia total (Energy = 62221.0 nJ) e a parcela que corresponde ao consumo de energia estática ou dinâmica nos circuitos que permanecem operando mesmo na ausência de tráfego (Idle Energy = 288.0 nJ).

CAFES dispõe de ferramentas de avaliação para todos os modelos que permitem estimar o tempo de comunicação da aplicação de forma *simplista*[40], considerando apenas a quantidade de comunicação; *otimista*[41], através da dependência de comunicação; ou de forma *realista*[42], através da quantidade de computação com a dependência da comunicação.

A análise temporal da aplicação descrita através do CWG da Figura 10.3(b)

[40] Os modelos CWM e ECWM calculam o tempo de execução de forma *simplista*, pois não consideram a contenção de mensagem nos recursos da infra-estrutura de comunicação e tampouco o tempo entre o envio de mensagens.

[41] O modelo CDM calcula o tempo de execução de forma *otimista*, pois considera que depois de resolvidas as dependências das mensagens, os núcleos podem enviar a mensagem que foi liberada, sem que seja considerado o tempo de computação do núcleo.

[42] O modelo CDCM pode estimar o tempo de execução da aplicação de forma *realista*, pois captura o tempo de comunicação e o tempo de computação.

com o mapeamento da Figura 10.5 está ilustrada na Figura 10.6. Esta é uma análise simplista, que considera todos os núcleos podendo enviar e receber informações concorrentemente. Os atrasos são computados frente aos seguintes elementos: (*i*) posicionamento dos núcleos na rede e (*ii*) número de phits transmitido; e frente as seguintes restrições: (*iii*) um núcleo não pode transmitir dados concorrentemente para mais de um núcleo; e (*iv*) um núcleo não pode receber dados concorrentemente de mais de um núcleo.

Origem -> Destino	Phits	Inicio	Fim	Diferença
A -> B	250	0	253	253
B -> C	450	0	453	453
B -> D	145	450	600	150
C -> D	130	0	133	133
D -> B	200	0	453	453
D -> A	180	200	383	183

Tempo Máximo: 600

Figura 10.6: Análise temporal da aplicação descrita na Figura 10.3(b) sobre a NoC com o mapeamento da Figura 10.5, considerando roteamento XY e chaveamento wormhole. Nota-se aqui, que as comunicações (A, B), (B, C) e (C, D) iniciam no instante 0 e levam exatamente o tempo necessário para transmitir todos os phits da mensagem passando por todos os nodos da rede até chegarem ao seu destino. A mensagem (D, B), embora inicie no instante 0 é postergada, pois o núcleo B estará ocupado durante toda a comunicação (A, B). A mensagem (B, D) tem transmissão iniciada assim que o núcleo B terminou de enviar a mensagem (B, C), ou seja, no instante 450. A mensagem (D, A), por sua vez, somente pode iniciar após o núcleo D ter transmitido toda a mensagem (D, B). Neste caso, a análise apenas espera que toda a mensagem saia do núcleo D, mas não é necessário que a mesma chegue ao destino.

A avaliação simplista pode ainda ser melhorada, mesmo para o modelo CWM, considerando a limitação de phits que podem ser armazenados na infra-estrutura de comunicação durante o envio de uma mensagem. Neste caso, a mensagem (D, A) levaria mais tempo para iniciar.

A Figura 10.7, por sua vez, ilustra a análise da comunicação de forma otimista, através da aplicação descrita pelo CDG da Figura 10.3(a) com o mapeamento da Figura 10.5. A análise é otimista, pois considera que uma vez resolvidas as dependências de uma mensagem, esta pode ser lançada na rede, o que implica desconsiderar o tempo de computação que antecede o envio da mesma.

ID	Origem -> Destino	Phits	Inicio	Fim	Diferença
0	A -> B	250	1	254	253
1	C -> D	130	1	134	133
2	B -> D	145	255	405	150
3	B -> C	450	255	853	598
4	D -> A	180	135	318	183
5	D -> B	200	854	1059	205

Tempo Máximo: 1059

Figura 10.7: Análise temporal da aplicação descrita na Figura 10.3(a) sobre a NoC com o mapeamento da Figura 10.5, considerando roteamento XY e chaveamento wormhole. A coluna ID é um identificador interno ao

programa apenas para distinguir entre diferentes mensagens com mesmos núcleos origem e destino.

Nesta análise, nota-se que as mensagens (A, B) e (C, D) iniciam no tempo 1, pois dependem apenas do vértice START. A mensagem (B, D) depende da mensagem (A, B), e por este motivo inicia apenas quando a mensagem (A, B) for completamente recebida pelo núcleo B. O mesmo acontece para a mensagem (D, A), que depende da mensagem (C, D). A mensagem (B, C) inicia ao mesmo tempo em que a mensagem (B, D), pois a última dependência a ser resolvida é a mesma dependência que (B, D) tem. Por fim, a última mensagem a ser lançada na rede é (D, B), cujas dependências são (B, C) e (D, A).

Comparando a análise simplista do modelo CWM com a otimista do modelo CDM nota-se uma diferença entre os modelos para estimar o tempo de comunicação.

A estimativa de tempo de execução da aplicação mais precisa é extraída a partir do modelo CDCM, pois nela o envio de mensagens depende não apenas da comunicação, mas também da computação dos núcleos. A Figura 10.8 ilustra a análise temporal obtida do CDCG descrito na Figura 10.2. Como pode ser observado, o tempo de execução obtido (946 ciclos) é inferior ao tempo de execução estimado pelo modelo CDM (1059 ciclos). Embora este resultado pareça um contra-senso, já que o modelo CDCM acrescenta o tempo de computação ao modelo CDM, ele serve para mostrar que a consideração do modelo CDM de que "toda mensagem independente vai causar contenção" pode gerar uma estimativa superior ao tempo de execução real.

ID	Origem -> Destino	Computação	Phits	Inicio	Fim	Diferença
0	A -> B	10	250	10	263	253
1	C -> D	12	130	12	145	133
4	D -> A	15	180	160	343	183
3	B -> C	20	450	283	736	453
2	B -> D	25	145	288	438	150
5	D -> B	5	200	741	946	205

Tempo Máximo: 946

Figura 10.8: Análise temporal da aplicação descrita na Figura 10.3(a) sobre a NoC com o mapeamento da Figura 10.5, considerando roteamento XY e chaveamento wormhole.

Para o CDCM, a ferramenta de análise temporal do *framework* CAFES mapeia o grafo da aplicação sobre o grafo da NoC, considerando as contenções e a computação. Caso duas ou mais mensagens concorram pelo mesmo recurso de comunicação ao mesmo tempo, o algoritmo de simulação privilegia a primeira mensagem avaliada, deixando as demais mensagens contidas nas áreas de armazenamento temporário.

Para todas as análises temporais (Figura 10.6, Figura 10.7 e Figura 10.8) foi considerado que o tempo de roteamento, transmissão de um phit entre roteadores e transmissão de um phit entre roteador e núcleo local é igual a um ciclo de relógio, tal como ilustrado na Figura 10.1.

10.2 Algoritmos Implementados no Framework CAFES

Para avaliar a influência do modelo da aplicação no mapeamento, os algoritmos de mapeamento foram divididos em duas partes aninhadas: (*i*) uma parte interna que é dependente da infra-estrutura de comunicação e do modelo da aplicação,

denominada *ObjectiveFunction*; e (*ii*) uma parte externa que envolve a *ObjectiveFunction*. A parte externa é um algoritmo genérico de pesquisa por soluções que alcancem mapeamentos de menor custo, sendo independente dos modelos da aplicação e da arquitetura alvo. A parte externa é chamada *IndependentMappingAlgorithm* e é implementada por um algoritmo de pesquisa exaustiva ou por uma variação do algoritmo heurístico *simulated annealing* (SA) (SHERWANI, 1999). A escolha do algoritmo de mapeamento pode ser determinada pelo projetista. Para NoCs de dimensão menor ou igual a 3×4, os algoritmos de pesquisa exaustiva conseguem obter resultados ótimos em tempo de computação aceitável. Para avaliar NoCs maiores em um tempo de computação aceitável, o projetista deve escolher mapeamentos através de SA.

10.2.1 Algoritmo de Mapeamento Independente dos Modelos da Aplicação e da Infra-Estrutura de Comunicação

Os algoritmos de pesquisa exaustiva calculam todas as possibilidades de mapeamento[43] para escolher o mapeamento que obtém menor custo. O SA utiliza heurísticas que permitem obter bons resultados sem a exploração de todo o espaço de soluções. O pseudo-código *IndependentMappingAlgorithm* está descrito na Figura 10.9.

```
IndependentMappingAlgorithm:

1   iteration ← initialInteractionValue
2   globalMinimumMappingCost ← MAXIMUM_VALUE
3   while iteration > 0
4   {
5         temperature ← initialTemperatureValue
6         minimumMappingCost ← MAXIMUM_VALUE
7         acceptedMappingCost ← MAXIMUM_VALUE
8         actualMapping ← RandomMapping(null, big_move)
9         while temperature > 0
10        {
11              actualMappingCost ← ObjectiveFunction(actualMapping)
12              if minimumMappingCost > actualMappingCost
13              {
14                    minimumMappingCost ← actualMappingCost
15                    minimumMapping ← actualMapping
16              }
17              if |actualMappingCost - acceptedMappingCost| < Threshold(temperature)
18              {
19                    acceptedMappingCost ← actualMappingCost
20                    acceptedMapping ← actualMapping
21              }
22              actualMapping ← RandomMapping(acceptedMapping, small_move)
23              decrease temperature
24        }
25        if globalMinimumMappingCost > minimumMappingCost
26        {
27              globalMinimumMappingCost ← minimumMappingCost
28              globalMinimumMapping ← minimumMapping
29        }
30        decrement iteration
31  }
```

Figura 10.9: Pseudo-código do *IndependentMappingAlgorithm* para o algoritmo simulated annealing.

A Figura 10.9 ilustra dois laços aninhados que constituem o algoritmo

[43] Existem *n!* possíveis mapeamentos distintos para *n* núcleos serem mapeados em *n* tiles de uma NoC.

IndependentMappingAlgorithm. O laço externo (linhas 3-31) representa uma técnica de pesquisa esparsa, enquanto o laço interno (linhas 9-21) implementa um SA elementar. O efeito de ambos os laços é a pesquisa aleatória por um largo espectro de possibilidades, permitindo encontrar um mapeamento que minimize a função objetivo, implementada no algoritmo interno (*ObjectiveFunction*).

Seja $N = \{n_1, n_2, \ldots, n_c\}$ o conjunto de núcleos da aplicação e $\tau = \{\tau_1, \tau_2, \ldots, \tau_p\}$ o conjunto de tiles da NoC, tal que $|\tau| \geq |N|$. Seja $\varphi_1, \varphi_2: N \rightarrow \tau$ duas funções de mapeamento de núcleos em tiles, chamadas de *mapeamento inicial* (φ_1) e *mapeamento final* (φ_2). Então, um *move* é uma função possivelmente incompleta *move*: $\tau \rightarrow \tau$, onde para dois tiles τ_i, τ_j, $move(\tau_i) = \tau_j$ se e somente se existir um core n em N, tal que $\varphi_1(n) = \tau_i$ e $\varphi_2(n) = \tau_j$.

A função *move* associa dois mapeamentos de núcleos, ou seja, relaciona tiles da NoC com núcleos da aplicação. É possível definir *null_move* como um *move* onde para todo τ_i $null\_move(\tau_i) = \tau_i$.

Para o algoritmo acima, é interessante definir *small_move*[44] como um movimento que troca a posição de dois núcleos ou, quando $|N| < |\tau|$, um movimento que troca de lugar um único núcleo. Formalmente, *small_move* é um *move* onde, para dois tiles τ_a, τ_b, $move(\tau_a) = \tau_b$ com $\tau_a \neq \tau_b$ para no máximo dois tiles distintos em τ. Para todos os outro tiles τ_i em τ, ou $move(\tau_i) = \tau_i$ ou $move(\tau_i)$ é indefinido. Um movimento não classificado como *small_move* é chamado de *big_move*.

O laço interno é baseado em *small_moves*, enquanto que o laço externo é baseado em *big_moves*, onde tipicamente todos os núcleos trocam de posição.

O laço interno inicia com um mapeamento aleatório, provido pelo laço externo, que é denominado **actualMapping**. Para pesquisar por um mínimo local, o laço interno implementa um algoritmo SA. A variável **temperature**[45] é usada para controlar o número de iterações do laço interno e a entropia do sistema. Quando **temperature** aumenta, a entropia também aumenta. Isto significa que aumenta a probabilidade do algoritmo aceitar um mapeamento com custo pior que os previamente computados. O oposto também é verdade - menor **temperature**, menor entropia - significando que a probabilidade de aceitação é reduzida. A cada nova execução do laço interno a variável **temperature** é reinicializada.

As variáveis **minimumMappingCost** e **acceptedMappingCost** são usadas para armazenar o custo do melhor mapeamento computado pelo laço interno e o custo de um mapeamento aceito pelo SA, respectivamente. São variáveis inicializadas com o maior valor real possível. A variável **minimumMappingCost** sempre mantém o menor custo dos mapeamentos obtidos com o laço interno, e é associada à estrutura

---

[44] Em geral, *small_moves* são obtidos pela mudança da associação de uns poucos núcleos a tiles.

[45] A *temperatura* é um parâmetro do algoritmo simulated annealing que determina o grau de aceitação de uma amostra, com função objetivo atingindo valor superior ao mínimo global até o momento. Quanto maior a temperatura, maior é a liberdade de aceitar amostras com um custo maior, permitindo que o algoritmo fuja de mínimos locais. Quanto menor a temperatura, menor a probabilidade de aceitar amostras com custo maior, fazendo com que o algoritmo se dirija para o mínimo local.

minimumMapping - uma estrutura que armazena o mapeamento de menor custo. Por outro lado, a variável **acceptedMappingCost** pode ter valores maiores que os previamente calculados, já que a mesma depende de **temperature**, e conseqüentemente, da probabilidade de aceitação do mapeamento. A estrutura **acceptedMapping**, que armazena o último mapeamento aceito pelo SA, está associada à variável **acceptedMappingCost**.

De acordo com os modelos da aplicação e da arquitetura alvo, a função objetivo de mapeamento (*ObjectiveFunction*) computa o custo de um mapeamento. O mapeamento é armazenado na estrutura **actualMapping** e o custo é armazenado na variável **actualMappingCost**. O custo é comparado duas vezes: (*i*) primeiramente com o **minimumMappingCost** para armazenar um novo custo mínimo, se for o caso, e (*ii*) posteriormente com o **acceptedMappingCost**. Neste último caso, se o **actualMappingCost** for menor que o **acceptedMappingCost** o mapeamento atual é aceito. Entretanto, conforme o valor de **temperature**, mesmo mapeamentos com custos mais altos podem ser aceitos.

Um novo mapeamento aleatório é calculado com *small_move* sobre o mapeamento armazenado em **acceptedMapping**, gerando um novo mapeamento atual (**actualMapping**), que será avaliado no laço seguinte. A última etapa do laço interno é o decremento de **temperature** e uma nova iteração do laço interno é executada outra vez, enquanto **temperature** for maior que zero. Após terminar um laço interno, **minimumMappingCost** é comparado com **globalMinimumMappingCost**, e caso seja menor, então **minimumMapping** e **minimumMappingCost** são armazenados em **globalMinimumMapping** e **globalMinimumMappingCost**, respectivamente. Por fim, a variável **iteration**, que controla o laço externo é decrementada e é executada uma nova iteração do laço externo.

O laço externo usa *big_moves* para pesquisar mapeamentos substancialmente diferentes dos atuais. Este laço é controlado pela variável **iteration**. O número de iterações indica o número de diferentes sementes de mapeamentos exploradas. Quando a variável **iteration** chegar a zero, o algoritmo pára e o mapeamento com custo mínimo global é armazenado na estrutura **globalMinimumMapping**.

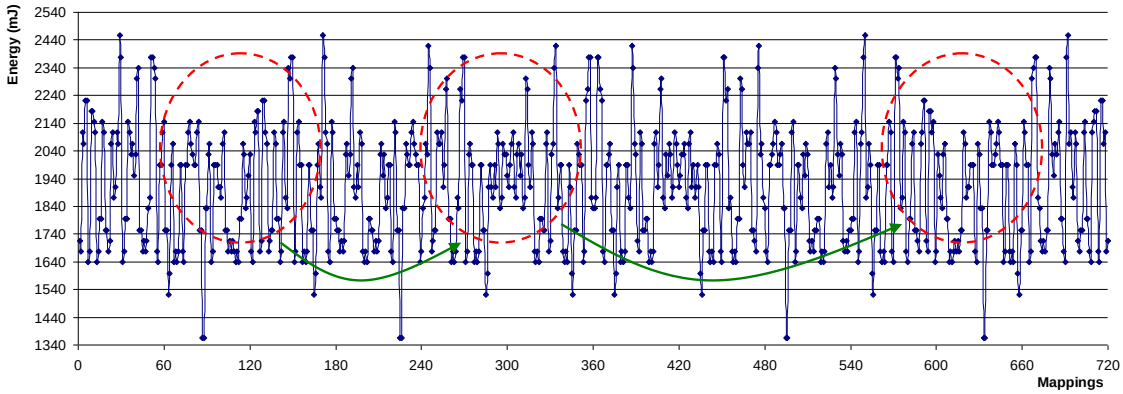


Figura 10.10: Ilustração do consumo de energia encontrado em 720 mapeamentos (*n!*) de uma aplicação sintética com seis núcleos. Os círculos pontilhados representam a exploração dos mínimos locais e as flechas curvas representam a pesquisa por mapeamentos que fogem dos mínimos locais. Existem 4 mínimos globais, com valor igual a 1343 mJ. Estes se encontram nos intervalos de mapeamento [60, 120], [180, 240], [480, 540] e [600, 660].

A razão para dividir o algoritmo de mapeamento externo em dois laços aninhados está no aumento da probabilidade de encontrar mínimos globais em vez de mínimos locais. Este efeito é ilustrado na Figura 10.10, que foi obtida pela exploração exaustiva de mapeamentos de uma aplicação sintética composta por seis núcleos. Nesta Figura, os círculos pontilhados representam uma pesquisa realizada com o laço interno (*small_move*) e as flechas curvas representam um novo local de pesquisa realizada com o laço externo (*big_moves*).

Para aplicações com até 12 núcleos, as implementações algorítmicas com SA e pesquisa exaustiva foram comparadas mostraram que ambos os algoritmos obtém resultados similares, o que comprova a qualidade do SA para pequenas aplicações. Para aplicações maiores, a pesquisa exaustiva é intratável, impossibilitando avaliar a qualidade das soluções obtidas com o SA.

10.2.2 Função Objetivo Utilizada para o Cálculo do Custo de Mapeamento

A função objetivo para o cálculo do custo de mapeamento, denominada *ObjectiveFunction*, é dependente tanto do modelo que descreve a infra-estrutura de comunicação, quanto do modelo que descreve a aplicação. O trabalho realizado envolve a implementação e análise de seis funções objetivo, uma para cada modelo descrito no Capítulo 5. Todavia, para não estender por demais o texto, serão descritos apenas os algoritmos que implementam os modelos CWM e CDM. Os demais podem ser inferidos a partir destes e dos modelos descritos no Capítulo 5.

Para todos os modelos, a *ObjectiveFunction* é implementada através de dois algoritmos aninhados: (*i*) um algoritmo que percorre o grafo da aplicação; e (*ii*) um algoritmo que percorre o grafo da infra-estrutura de comunicação. Estes algoritmos e suas interações são descritos no decorrer desta Seção.

10.2.2.1 Função Objetivo Utilizada para o Cálculo de Mapeamento com CWM

Para uma aplicação modelada por CWM, o ponto de partida da *ObjectiveFunction* é denominado *Communication Weighted Algorithm* (CWA). O CWA executa o caminhamento no CWG, o grafo que descreve a aplicação. O CWG é implementado por um conjunto de conjuntos, onde cada conjunto é implementado com uma lista. A lista externa contém os vértices fonte (origens das mensagens), para cada vértice fonte há uma lista de vértices destino (destinos das mensagens) e um peso da comunicação, que é o número de phits transmitidos do vértice fonte para o vértice destino (tamanho das mensagens).

O CWA, ilustrado na Figura 10.11, é implementado com dois laços aninhados. O laço mais externo (linhas 3-12) percorre os vértices fonte. Para cada vértice fonte, o laço mais interno (linhas 6-10) percorre todas as comunicações deste com os seus vértices destino. Cada comunicação é representada por uma aresta do CWG. A cada iteração, as posições dos vértices fonte e destino, e a quantidade de phits transmitido na comunicação são parâmetros para o algoritmo que percorre o grafo da infra-estrutura de comunicação (*CWM_NoC_Algorithm*). A variável **mappingCost**, que contém o custo total de mapeamento, é zerada toda a vez que o CWA for executado, ou seja, a cada novo cálculo de custo de mapeamento.

***Communication Weighted Algorithm*:**

```
1   mappingCost ← 0
2   s ← firstSourceVertex
3   while s != null
4   {
5         t ← s.firstTargetVertex
6         while t != null
7         {
8               mappingCost ← mappingCost + CWM_NoC_Algorithm(s.x, s.y, t.x, t.y, t.phits)
9               t ← t.nextTargetVertex
10        }
11        s ← s.nextSourceVertex
12  }
13  return mappingCost
```

Figura 10.11: Algoritmo que implementa a função objetivo (*ObjectiveFunction*) para o modelo CWM.

Sendo n o número de núcleos da aplicação, o CWA tem complexidade O(n^2). Todavia, exemplos de aplicações práticas mostram que raramente muitos núcleos compõem o laço mais interno, justificando o baixo tempo de processamento do mesmo, quando comparado com os demais algoritmos.

O *CWM_NoC_Algorithm* implementa a parte da função objetivo que é dependente da infra-estrutura de comunicação. Este algoritmo retorna para o CWA o custo de um determinado mapeamento, respeitando a estimativa de consumo de energia de cada componente e as características topológicas da infra-estrutura de comunicação (e.g. algoritmo de roteamento).

A Figura 10.12 ilustra o *CWM_NoC_Algorithm* considerando uma NoC malha, com roteamento XY e chaveamento wormhole. O *CWM_NoC_Algorithm* é composto por dois laços consecutivos. O primeiro laço (linhas 4-8) representa o tráfego de um pacote que parte da coluna (X) onde está mapeado o núcleo origem (**xSource**) e vai até a coluna onde está mapeado o núcleo destino (**xTarget**). O segundo laço (linhas 9-13) implementa o restante do caminho de comunicação, que é o deslocamento do pacote da linha (Y) onde está mapeado o núcleo origem (**ySource**) até a linha onde está mapeado o núcleo destino (**yTarget**).

CWM_NoC Algorithm:

```
1   commCost ← 0
2   x ← xSource
3   y ← ySource
4   while xTarget != x
5   {
6         commCost ← commCost + ResourceCost(x, y, phits)
7         x ← x + 1
8   }
9   while yTarget != y
10  {
11        commCost ← commCost + ResourceCost(x, y, phits)
12        y ← y + 1
13  }
14  return commCost
```

Figura 10.12: Descrição do algoritmo *CWM_NoC_Algorithm*. O algoritmo computa o custo da comunicação em termos de energia total, para uma NoC malha com roteamento XY e chaveamento wormhole.

Em cada novo recurso de comunicação que o pacote emprega, há consumo de energia associado. Este consumo é computado para cada recurso pela função

ResourceCost. Esta recebe como parâmetros a posição onde está o recurso de comunicação e a quantidade de phits que trafega por este.

O consumo de energia de cada recurso é somado à variável de custo da comunicação (**commCost**) para compor o consumo de energia total de uma comunicação entre o núcleo origem e o núcleo destino. O **commCost** é retornado para o CWA de forma a este contabilizar o consumo total de energia de uma comunicação.

Para exemplificar o funcionamento do CWA, utiliza-se aqui a aplicação da Figura 10.3(b), onde as 6 comunicações são mapeadas sobre grafos que representam a NoC, com as comunicações ilustradas da Figura 10.13(a1) até a Figura 10.13(f2).

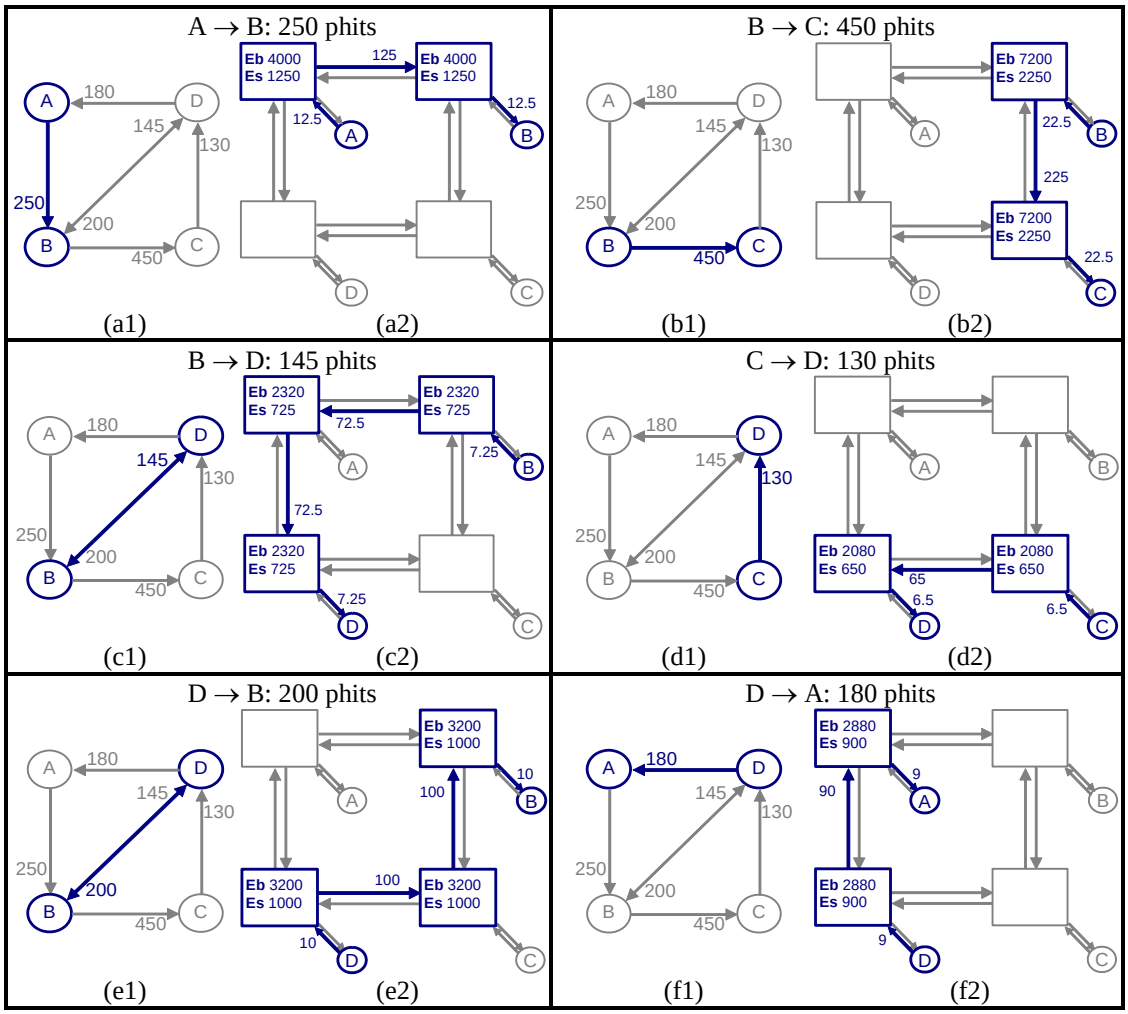


Figura 10.13: Ilustrações dos passos do algoritmo que implementa a função objetivo do modelo CWM, tendo como infra-estrutura de comunicação uma NoC 2×2 com topologia malha, roteamento XY e chaveamento wormhole. Cada par de figuras ilustra uma comunicação. Figuras à esquerda representam o grafo da aplicação com uma mensagem em destaque. Figuras à direita representam o consumo de energia desta mensagem em cada recurso de comunicação.

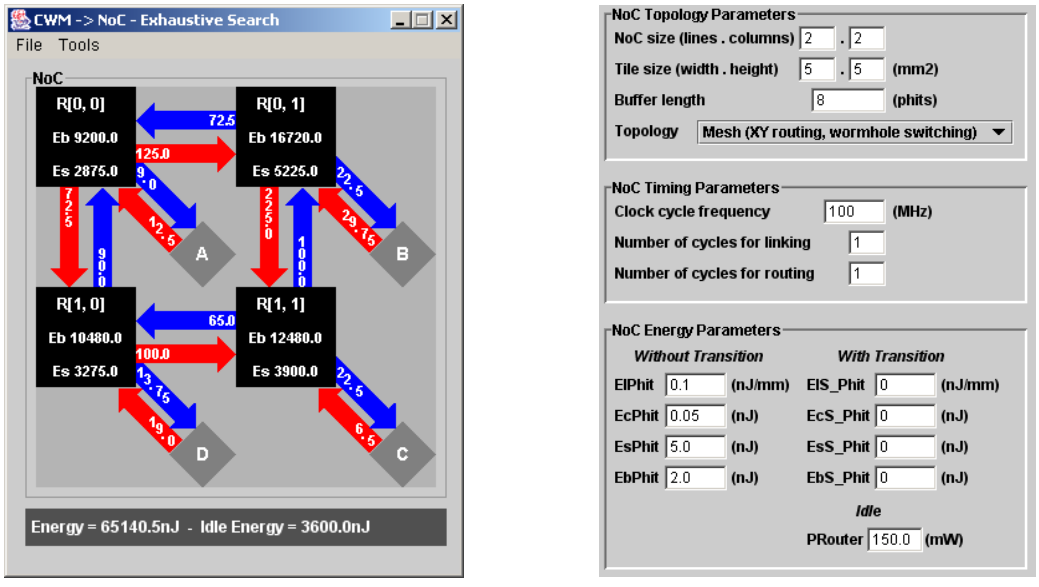
A Figura 10.13(a1) mostra em negrito uma comunicação de 250 phits do núcleo **A** para o núcleo **B**, e a Figura 10.13(a2) mostra o consumo de energia nos recursos por onde esta comunicação passa, considerando o mapeamento ilustrado e os parâmetros de energia apresentados na Figura 10.14(b). Cada conexão entre núcleo e roteador consome 12.5 nJ (250×0.05 nJ), a conexão entre roteadores consome 125 nJ

(250 × 5 mm×0,1 nJ/mm), o circuito de controle de cada roteador consome 1250 nJ (250 × 5 nJ) e o circuito de armazenamento temporário consome 4000 nJ (250 ×8 ×2 nJ). Assim, esta mensagem totaliza o consumo de 10650 nJ. Este mesmo resultado pode ser obtido com uma pequena adaptação da Equação 8.9 dado que a conexão entre núcleo local e roteador não é aqui negligenciada, os *buffers* têm comprimento 8 e a informação de consumo de energia para cada phit é dada por:

$$EPhit_{ij} = \eta \times (ESPhit + \text{tamanhoBuffer} \times EBPhit + ECPhit) + (\eta -) \times \text{comprimentoLink} \times ELPhit$$
$$= 2 \times (5\text{ nJ} + 8 \times 2\text{ nJ} + 0.05\text{ nJ}) + (2 - 1) \times 5\text{ mm} \times 0.1\text{ nJ/mm}$$
$$= 42.6\text{ nJ}$$

O consumo total de energia é obtido multiplicando-se o número de phits por *EPhit*: 250 × 42.6nJ = 10650 nJ, que é o mesmo resultado descrito acima. As demais mensagens representadas da Figura 10.13(b1) até a Figura 10.13 (f2) têm consumo de energia computado de forma análoga.

A totalização do exemplo ilustrado da Figura 10.13(a1) até a Figura 10.13(f2) pode ser observada na Figura 10.14(a), obtida com a ferramenta CAFES, a partir dos parâmetros de entrada ilustrados na Figura 10.14(b). Nesta Figura, em todos os recursos de comunicação está anotada a soma de toda energia consumida por todas as comunicações que passam pelo recurso. Na parte inferior da Figura 10.14(a) está apresentado o somatório de todas as energias consumidas em todos os recursos (65140.5 nJ), que é calculado pelo CAFES através da Equação 8.17. A Figura 10.14(a) também ilustra o consumo de energia estática e dinâmica nos circuitos que operam mesmo na ausência de tráfego (3600 nJ). Esta última estimativa é obtida pela multiplicação da estimativa de tempo de execução da aplicação (600 ciclos), tal como ilustrado na Figura 10.6, com o parâmetro *PRouter* (150 mW), ilustrado na Figura 10.14(b), e com o número de tiles da aplicação (4).



(a) (b)

Figura 10.14: Em (a) apresenta-se o mapeamento de uma aplicação sintética em uma NoC 2 × 2 com consumo de energia anotado em todos os recursos de comunicação; em (b) estão ilustrados parâmetros da NoC.

10.2.2.2 Função Objetivo Utilizada para o Cálculo de Mapeamentos com CDM

Para uma aplicação modelada por CDM, o ponto de partida da

ObjectiveFunction é denominado *Communication Dependence Algorithm* (CDA). A principal diferença entre os algoritmos CWA e CDA é a informação de dependência provida pelo CDM, que permite estimar contenção de mensagens independentes que compartilham os mesmos recursos de comunicação.

Embora o CDA tenha sido desenvolvido para calcular mapeamentos de aplicações modeladas por CDM, a entrada do algoritmo não é um CDG. Para diminuir o tempo de processamento, é criada, a partir do CDG, uma lista de listas que representa níveis de dependência entre as mensagens. Esta lista é denominada CDL (*Communication Dependence List*), e sua estrutura está ilustrada através do exemplo da Figura 10.15. O tempo de processamento é reduzido porque é necessário criar a CDL uma única vez antes de iniciar a pesquisa por mapeamentos, enquanto que devem ser executadas diversas pesquisas por dependências a cada novo mapeamento. A contrapartida é o gasto a mais em memória, dado que cada mensagem tem associada uma lista contendo todas as suas dependências.

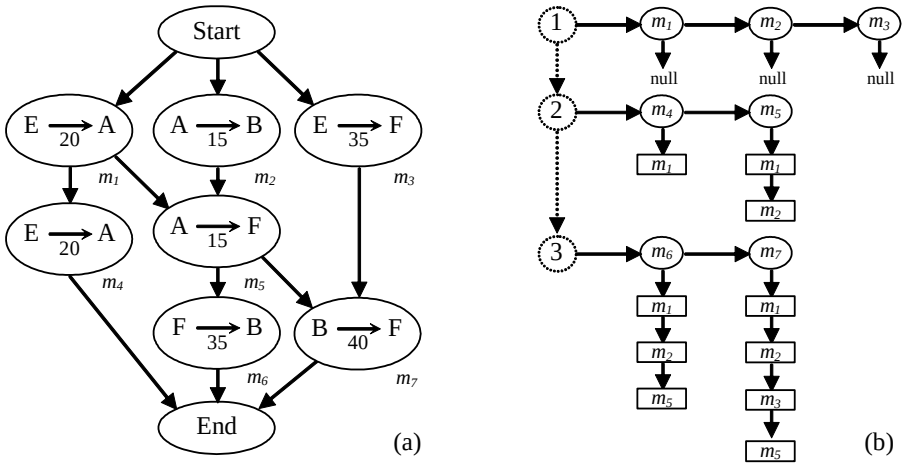


Figura 10.15: Descrição de uma aplicação através do CDG (a) e a correspondente lista de listas CDL (b). Nesta última, círculos pontilhados ligados por linhas pontilhadas representam a lista dos níveis; círculos contínuos ligados por linhas contínuas representam listas de mensagens do mesmo nível. Para cada mensagem está associada uma lista das dependências, onde cada dependência é representada por um retângulo.

Cada elemento de cada nível da CDL é composto por um vértice *v* e uma lista de vértices do qual o vértice *v* depende. Vértices do mesmo nível são necessariamente independentes e podem apenas depender de vértices de níveis inferiores.

A Figura 10.15(a) ilustra um CDG de uma aplicação sintética e a Figura 10.15(b) apresenta a CDL correspondente. A construção de CDL é obtida percorrendo todos os caminhos que levam a cada vértice do CDG e anotando todos os vértices percorridos como dependentes deste. Para exemplificar, a Figura 10.15(a) mostra que enquanto a mensagem m_1 depende apenas do vértice START e por este motivo tem na CDL uma lista nula, a mensagem m_6 depende diretamente da mensagem m_5, e indiretamente das mensagens m_1 e m_2, e por este motivo m_1, m_2 e m_5 fazem parte da lista de vértices dos quais m_6 depende. Ou seja, a lista de dependências de uma mensagem armazena todas as mensagens de que esta depende desde o vértice START.

O CDA, ilustrado na Figura 10.16, é implementado com dois laços aninhados de forma semelhante ao CWA. A principal diferença aqui é a ordem natural dos laços suprida pelo CDL e os parâmetros fornecidos para o algoritmo que percorre o grafo da

NoC (*CDM_NoC_Algorithm*). Enquanto que para o *CWM_NoC_Algorithm* são necessários apenas as posições dos núcleos origem e destino e a quantidade de bits transmitida, no *CDM_NoC_Algorithm* é necessário passar a informação de qual é a mensagem que utilizará o recurso e a lista de dependências desta mensagem.

***Communication Dependence Algorithm*:**

```
1   mappingCost ← 0
2   level ← firstLevel
3   while level != null
4   {
5         message ← level.firstElement
6         while message != null
7         {
8               mappingCost ← mappingCost + CDM_NoCAlgorithm(message)
9               message ← message.nextMessage
10        }
11        level ← level.nextLevel
12  }
13  return mappingCost
```

Figura 10.16: Algoritmo utilizado para o cálculo da função objetivo de mapeamento para aplicações modeladas com CDM.

O *CDM_NoC_Algorithm*, ilustrado na Figura 10.17, calcula o caminho de uma mensagem percorrendo todos os recursos pertinentes da NoC para o roteamento XY, partindo da posição em que está mapeado o núcleo origem e indo até a posição onde está mapeado o núcleo destino.

CDM_NoC Algorithm:

```
1   commCost ← 0
2   x ← message.sourceCore.x
3   y ← message.sourceCore.y
4   while message.targetCore.x != x
5   {
6         commCost ← commCost + ResourceCost(x, y, message)
7         x ← x + 1
8   }
9   while message.targetCore.y != y
10  {
11        commCost ← commCost + ResourceCost(x, y, message)
12        y ← y + 1
13  }
14  return commCost
```

Figura 10.17: Ilustração do algoritmo *CDM_NoC_Algorithm*. Este, descreve o tráfego de uma mensagem dentro de uma NoC, considerando roteamento XY.

A função *ResourceCost* recebe como parâmetro a posição **x** e **y** do recurso e a estrutura **message**. Esta última é um nodo de CDL, contendo os núcleos origem e destino, a quantidade de phits e a lista de mensagens de que esta depende. Em cada recurso de comunicação são armazenados referências de todas as mensagens e o intervalo de tempo que esta mensagem permanece ocupando o recurso. Assim, o algoritmo tem condições de avaliar se as próximas mensagens que irão ocupar este recurso devem ser atrasadas ou não. O algoritmo não se preocupa com as mensagens dependentes, pois estas não podem estar na infra-estrutura de comunicação no mesmo intervalo de tempo. Todavia, a característica pessimista deste algoritmo considera que se duas mensagens concorrem pelo mesmo recurso e não são dependentes, então elas irão causar contenção. Em decorrência, esta última mensagem será postergada pelo tempo que as anteriores ocuparem o recurso. A conseqüência desta abordagem é que o

algoritmo de mapeamento pesquisará por mapeamentos de núcleos capazes de evitar que mensagens não dependentes compartilhem o mesmo recurso.

Para exemplificar o CDA, considera-se a aplicação da Figura 10.3(a), semelhante à Figura 10.18(a). Nesta última, porém, é acrescida a informação do número de cada mensagem. Com um processamento inicial, o CDG da Figura 10.18(a) é transformado na CDL da Figura 10.18(b). O algoritmo que transforma o CDG em CDL e que não está descrito neste trabalho assemelha-se ao algoritmo de escalonamento ASAP (*as soon as possible*) (MARCON, 2002a).

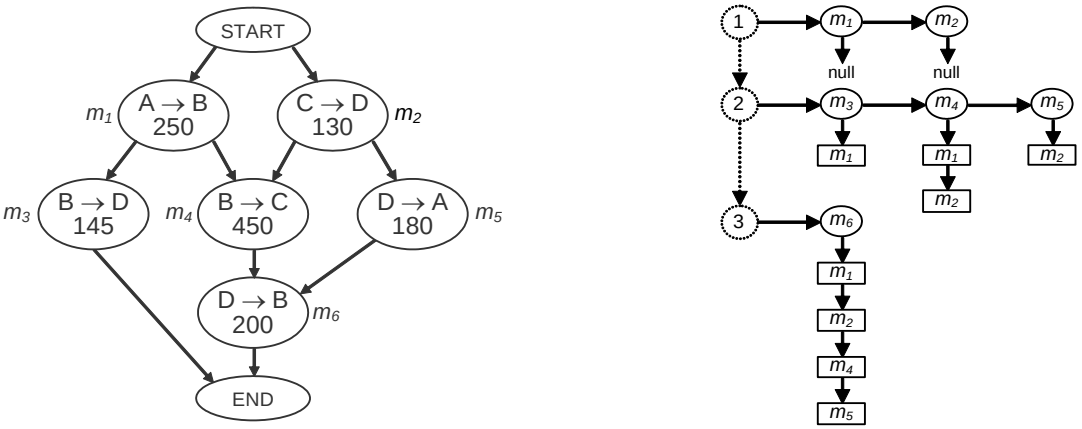


Figura 10.18: (a) CDG de uma aplicação sintética e (b) a CDL correspondente.

O CDA assume que os recursos de comunicação anotam as mensagens e intervalo de tempo que estas estiverem ocupando. Os caminhos percorridos por todas as mensagem na NoC estão ilustrados da Figura 10.19 a Figura 10.24 com a simbologia $F_{ai}\,[t_{ini}, t_{fim}]$, onde:

1. F – é o número de phits da mensagem que é transmitido no meio físico;
2. a – é um dos canais de comunicação do roteador, para onde o algoritmo de roteamento está direcionando a mensagem. Estes canais podem ser: norte (N), sul (S), leste (E), oeste (W), e local (L);
3. i – é o identificador da mensagem que está sendo transmitida;
4. $[t_{ini}, t_{fim}]$ – é o intervalo de tempo que a mensagem utiliza um recurso de comunicação.

A Figura 10.19 descreve a transmissão de m_1 que parte do núcleo **A**, mapeado no tile τ_1 e segue até o núcleo **B**, mapeado em τ_2.

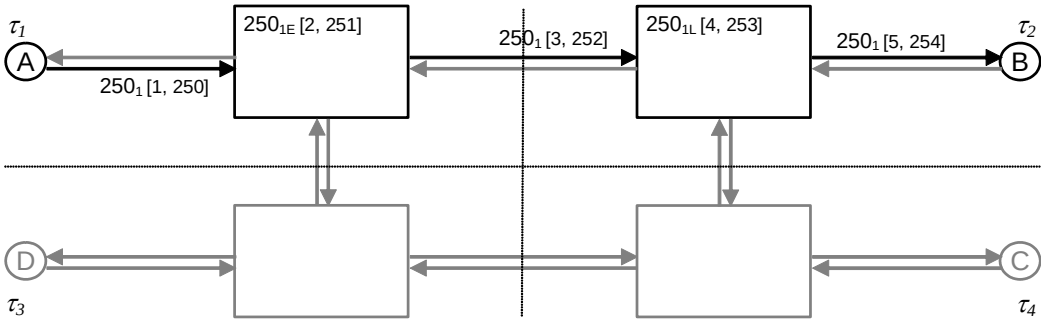


Figura 10.19: Transmissão de m_1, ilustrada para a CDL da Figura 10.18(b), em uma NoC malha 2 × 2 com roteamento XY e chaveamento wormhole.

A ausência de mensagens de que m_1 depende faz que esta comunicação possa iniciar no tempo 1, partindo de τ_1 e percorrendo a NoC através do roteamento XY até τ_2. A passagem por apenas 2 roteadores e 3 conexões faz com que esta mensagem tenha um intervalo total de transmissão pela NoC que inicia no ciclo 1 e termina no ciclo 254. Em cada recurso de comunicação as características da mensagem são anotadas. Como exemplo, o canal de comunicação entre o roteador implementado em τ_1 e o roteador implementado em τ_2 está anotado com 250_1 [3, 252], significando que m_1 tem 250 phits e o intervalo que a mensagem ocupa o recurso é compreendido entre 3 e 252, inclusive.

Aplicando a Equação 8.4, com os parâmetros da Figura 10.14(b) (1 ciclo para trafegar um phit entre um núcleo e um roteador ou entre roteadores (t_L), 1 ciclo para efetuar o roteamento do primeiro phit de uma mensagem (t_R), e o período de relógio λ é igual a 10 ns (100 MHz)), obtém-se que o tempo necessário para transmitir m_1 de τ_1 para τ_2 ($d_{12,1}$) é:

$$d_{12,1} = (\eta \times (t_R + t_L) + t_L \times n_{141}) \times \lambda = (2 \times (1 + 1) + 1 \times 250) \times 10 \text{ ns} = 2.54 \text{ µs}.$$

A Figura 10.20 descreve a transmissão de m_2, que parte do núcleo **C**, mapeado em τ_4, e segue até o núcleo **D**, mapeado em τ_3. Da mesma forma que m_1, esta mensagem não depende de nenhuma outra. Logo, para o algoritmo CDA, m_1 e m_2 são consideradas mensagens concorrentes.

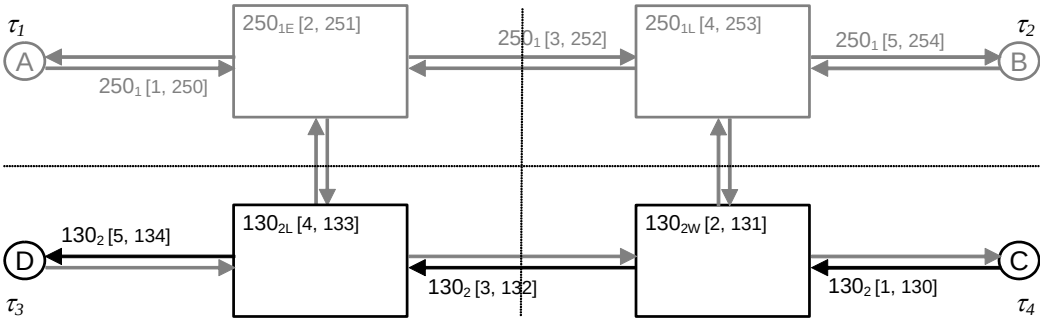


Figura 10.20: Envio de m_2 (seqüência da Figura 10.19 com m_1 em cor mais clara).

A Figura 10.21 descreve o tráfego de m_3, que parte do núcleo **B**, mapeado em τ_2, seguindo até o núcleo **D**, mapeado em τ_3. A mensagem m_3 é dependente de m_1, ou seja, somente pode ser lançada na rede quando m_1 for completamente recebida pelo núcleo **A**. Como o CDM não considera o tempo de computação dos núcleos, m_3 é lançada no ciclo subseqüente ao término de m_1.

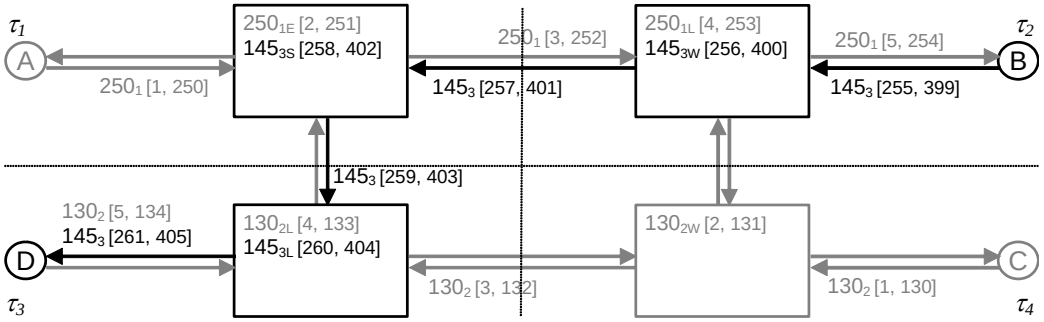


Figura 10.21: Transmissão de m_3, que ocorre após m_1 chegar ao seu destino (seqüência da Figura 10.20 com m_2 ilustrada em cor mais clara).

A Figura 10.22 descreve o tráfego de m_4, que parte do núcleo **B**, mapeado em τ_2 seguindo até o núcleo **C**, mapeado em τ_4. A Figura mostra que ao lançar m_4, o

algoritmo detecta que o canal entre o núcleo **B** e roteador de τ_2 está ocupado por m_3, pois m_3 e m_4 são independentes. Desta forma, o tempo que m_4 leva para ser entregue ao núcleo destino é acrescido do tempo de contenção no recurso de comunicação. A contenção faz com que m_4 seja atrasada de 145 ciclos de relógio, que é o tempo necessário para m_3 liberar o recurso. Aqui vale uma ressalva: "em uma aplicação não sintética existiria uma dependência algorítmica natural entre as mensagens m_3 e m_4, pois ambas são provenientes do mesmo núcleo". Todavia, o CDA não faz consistências com relação a dependências da aplicação, pois está voltado para avaliação apenas da infra-estrutura de comunicação.

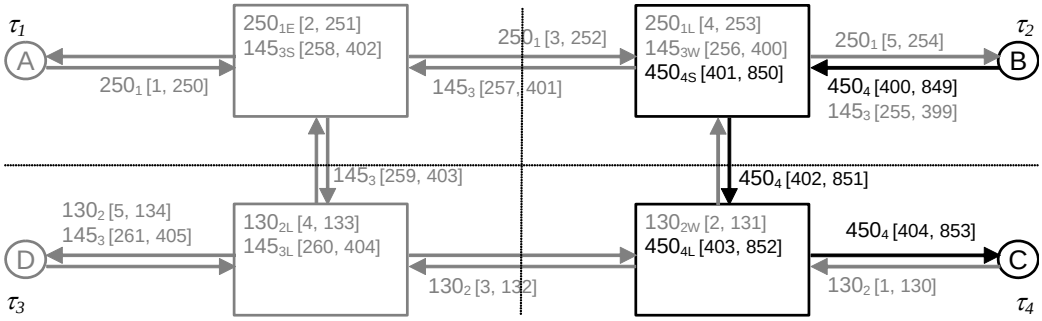


Figura 10.22: Tráfego de m_4 que fica contida no núcleo **B** enquanto a conexão entre este núcleo e o roteador estiver sendo ocupada por m_3. O comportamento descrito sucede o da Figura 10.21, com m_3 ilustrada em cor mais clara.

A Figura 10.24 descreve o caminhamento de m_6, que parte do núcleo **D**, mapeado em τ_3 e segue até o núcleo **B**, mapeado em τ_2. A mensagem m_6 inicia apenas no ciclo 854, pois esta é diretamente dependente de m_4, que encerra no ciclo anterior.

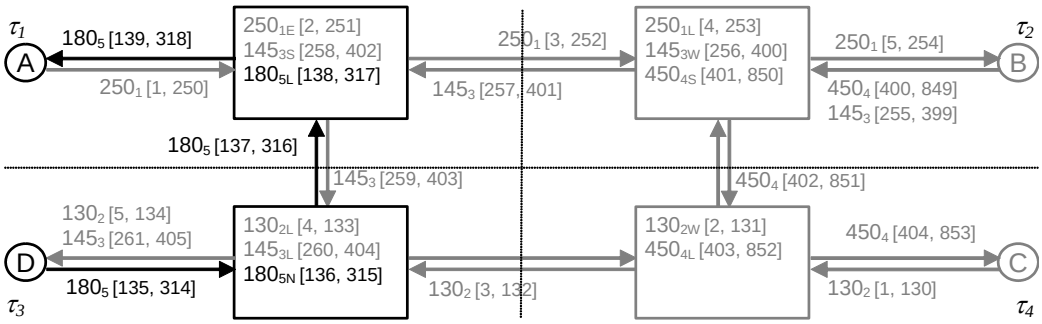


Figura 10.23: Tráfego de m_5, que é apenas dependente de m_2 (seqüência da Figura 10.22 com m_4 ilustrada em cor mais clara).

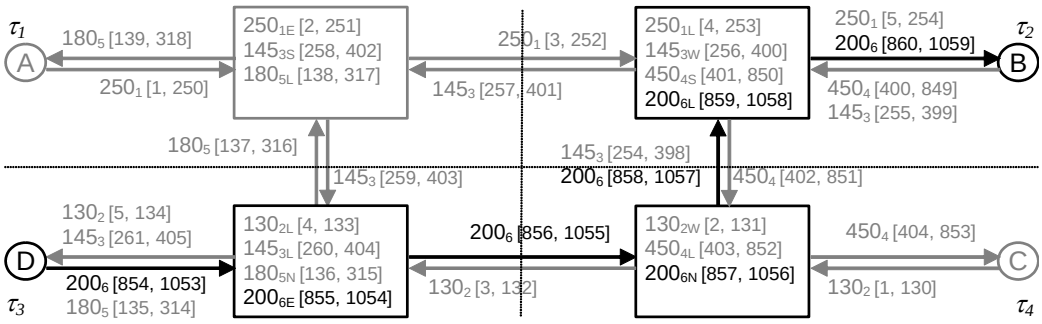


Figura 10.24: Transmissão de m_6, cujo início depende de m_4. O comportamento descrito aqui sucede o da Figura 10.23, com m_5 ilustrada em cor mais clara.

O algoritmo que implementa o CDM trata todas as mensagens que podem ser concorrentes como sendo concorrentes. De acordo com o CDG descrito na Figura 10.18, podem ocorrer as seguintes concorrências: (*i*) m_1 pode concorrer com as mensagens m_2 e m_5; (*ii*) m_2 pode concorrer com as mensagens m_1 e m_3; (*iii*) m_3 pode concorrer com as mensagens m_2, m_4, m_5, e m_6; (*iv*) m_4 pode concorrer com as mensagens m_3 e m_5; (*v*) m_5 pode concorrer com as mensagens m_1, m_3, e m_4; e (*vi*) m_6 pode concorrer com a mensagem m_3.

A execução do CDA tem como resultado a ilustração da Figura 10.25. Para este exemplo, o cálculo do consumo de energia dinâmica é igual ao encontrado pelo algoritmo CWA (Figura 10.14(a)). Contudo o consumo de energia devido ao tempo de execução do sistema é muito maior.

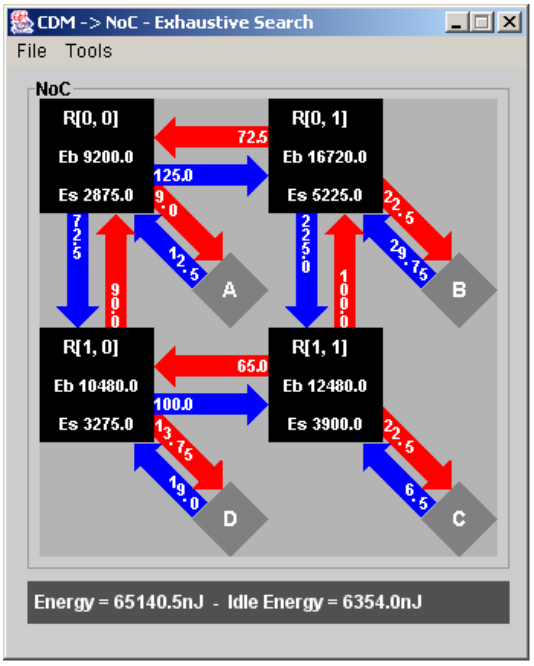


Figura 10.25: Cálculo total da energia consumida por uma aplicação descrita via CDG sobre o grafo que representa a NoC.

O tempo de execução da aplicação (*texec*) é dado pelo número de ciclos de relógio necessários para transmitir todas as mensagens. Para a aplicação acima, *texec* é igual a 1059 ciclos de relógio. Utilizando os parâmetros da Figura 10.14(b), o relógio tem período igual a 10 ns. Assim, o tempo total de execução da aplicação é igual a 10.59 μs. A energia estática consumida na NoC e a energia dinâmica consumida nos circuitos que operam mesmo na ausência de tráfego (*EStDynNoC*) é obtida pela Equação 8.24, onde o número de tiles ($|\tau|$) é 4 e *PRouter* é 150 mW. Assim,

$$EStDynNoC = PStDynNoC \times texec = |\tau| \times PRouter \times texec = 4 \times 150\ \text{mW} \times 10.59\ \mu\text{s} = 6354\ \text{nJ}.$$

O *EStDynNoC* estimado para o modelo CDM é otimista, pois não considera o tempo de computação. Mesmo assim, é mais adequado que o modelo utilizado para estimar tempo de execução pelo algoritmo CWA.

10.3 Resultados

Esta Seção descreve resultados obtidos com o *framework* CAFES.

Inicialmente, os algoritmos de mapeamento são comparados com relação ao tempo médio de execução. O objetivo é comparar quantitativamente os algoritmos e avaliar quais fatores afetam o tempo de execução. Posteriormente, os algoritmos são avaliados com relação à capacidade de obter mapeamentos que atendam os requisitos de projeto, que para este trabalho são a redução do tempo de comunicação e a redução do consumo de energia da infra-estrutura de comunicação.

10.3.1 Análise do Tempo de Execução dos Algoritmos de Mapeamento

O *framework* CAFES tem implementado algoritmos referentes aos modelos CWM, ECWM, CDM, CDCM e ACPM. Os tempos de execução destes algoritmos são avaliados em quatro experimentos: (*i*) o primeiro salienta a complexidade do problema de mapeamento comparando algoritmos de pesquisa exaustiva com *simulated annealing*; (*ii*) o segundo experimento utiliza a aplicação SegImag, descrita no Capítulo 6, para ilustrar valores absolutos de tempo de execução dos algoritmos, separando a parcela do algoritmo que é independente dos modelos da aplicação e infra-estrutura de comunicação da parcela que é dependente; (*iii*) o terceiro ilustra a influência da infra-estrutura de comunicação e da aplicação no tempo de execução dos algoritmos de mapeamento; e (*iv*) o quarto e último experimento analisa a influência de características da aplicação no tempo de execução de cada um dos algoritmos.

Para todos os experimentos, os resultados foram obtidos a partir de um computador pessoal com processador Pentium IV, freqüência de operação 2.4GHz, 512 MBytes de memória principal e sistema operacional Windows XP SP#2.

Sendo $|\tau|$ o número de tiles de uma NoC, o problema de mapeamento implica $|\tau|!$ mapas distintos para que todas as possibilidades sejam avaliadas. Assim, mesmo para NoCs de poucos tiles fica evidente que algoritmos de pesquisa exaustiva consomem muito tempo de execução, sendo impraticáveis. A natureza fatorial do problema de mapeamento pode ser observada na Figura 10.26. Este fato acontece para todos os algoritmos de mapeamento, independente de qual é o modelo da aplicação implementado. Por exemplo, o CWA, que é o algoritmo de menor complexidade, leva quase 5 horas (17460251 ms) para encontrar exaustivamente o mapeamento ótimo em uma NoC 3×4. Reduzindo apenas dois tiles, que é o caso exemplificado de uma NoC 2×5, o mesmo algoritmo consegue encontrar um mapeamento ótimo em cerca de 2 minutos (120004 ms) de execução. Esta pequena diferença do número de tiles, implicando a grande diferença no tempo de execução do algoritmo, ilustra o crescimento do tempo de execução com característica tipicamente exponencial do problema de mapeamento. Para o CDA, cuja complexidade é maior que o CWA, o tempo de execução é ainda maior. Por exemplo, em uma NoC 3×4 a pesquisa exaustiva consumiu mais de 34 horas de CPU (124672955 ms). Para lidar com a complexidade do problema de mapeamento, CAFES possui uma versão de *simulated annealing*, descrita na Figura 10.9. Esta versão utiliza heurísticas que permitem obter bons resultados em tempo de execução aceitável. Para NoCs pequenas (menos de seis tiles – 2×3), o algoritmo *simulated annealing* chega a obter tempos de execução pouco superiores ao da pesquisa exaustiva. Para NoCs um pouco maiores (mais que 8 tiles – 2×4) contudo, o tempo de execução de algoritmos de pesquisa exaustiva é pelo menos uma ordem de grandeza superior.

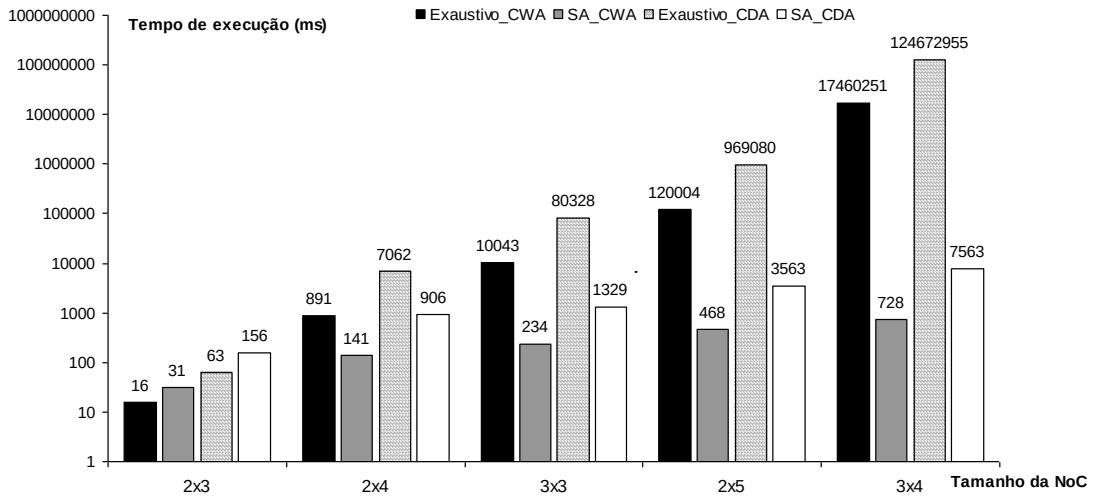


Figura 10.26: Ilustração em escala logarítmica do tempo de execução de algoritmos de mapeamentos obtido para diferentes tamanhos de NoCs. Os algoritmos comparados são a pesquisa exaustiva (prefixo Exaustivo_) e o simulated annealing (prefixo SA_) implementados para a avaliação de aplicações modeladas por CWM e CDM.

Como foi descrito na Seção 10.2, os algoritmos aqui implementados são divididos em duas partes: uma que é independente dos modelos da aplicação e da infra-estrutura de comunicação (*IndependentMappingAlgorithm*) e outra dependente, que é a função objetivo de mapeamento (*ObjectiveFunction*). A primeira parte é implementada com pesquisa exaustiva ou *simulated annealing*, a segunda parte é implementada por laços que percorrem os grafos que descrevem a aplicação e a NoC. Para avaliar a influência dos modelos da aplicação e da infra-estrutura de comunicação no tempo de execução é importante separar o tempo de cada parte do algoritmo. A Tabela 10.1 apresenta, para uma mesma aplicação (SegImag, descrita no Capítulo 6), o tempo de execução do *IndependentMappingAlgorithm* e *ObjectiveFunction* para diferentes modelos da aplicação. Nesta Tabela, observa-se que o *IndependentMappingAlgorithm*, implementado neste experimento com pesquisa exaustiva, obteve praticamente o mesmo tempo de execução para qualquer modelo. Contudo, a *ObjectiveFunction* mostra que a baixa complexidade dos algoritmos CWA e ECWA conduz a baixos tempos de execução, enquanto que algoritmos mais complexos como o CDCA e o CDA acarretam maiores tempos de execução. Para esta aplicação, o ACPA obtém tempo de execução intermediário.

Tabela 10.1: Tempo de execução, em milisegundos, de cada parte do algoritmo de mapeamento obtidos para a aplicação SegImag descrita no Capítulo 6.

Tempo de execução (ms)	CWA	ECWA	ACPA	CDA	CDCA
Total	31	47	78	109	185
IndependentMappingAlgorithm	15	16	15	15	16
ObjectiveFunction	16	31	63	94	169

Com o objetivo de analisar a influência do número de núcleos da aplicação e do número de tiles da rede intrachip no tempo de execução dos algoritmos de mapeamentos, três aplicações sintéticas, cada qual com 9 núcleos, foram sendo alteradas de forma a reduzir o número de núcleos, mas manter o número de tiles. Com propósito de reduzir a influência do número de vértices dos grafos CDCG, CDG e ACPG no tempo de execução, ao invés de simplesmente eliminar o vértice que continha o núcleo,

o vértice foi alterado trocando o núcleo a ser removido por outro núcleo existente. A mesma técnica não pode ser adotada para os grafos CWG e ECWG, pois para estes o núcleo é o próprio vértice da aplicação.

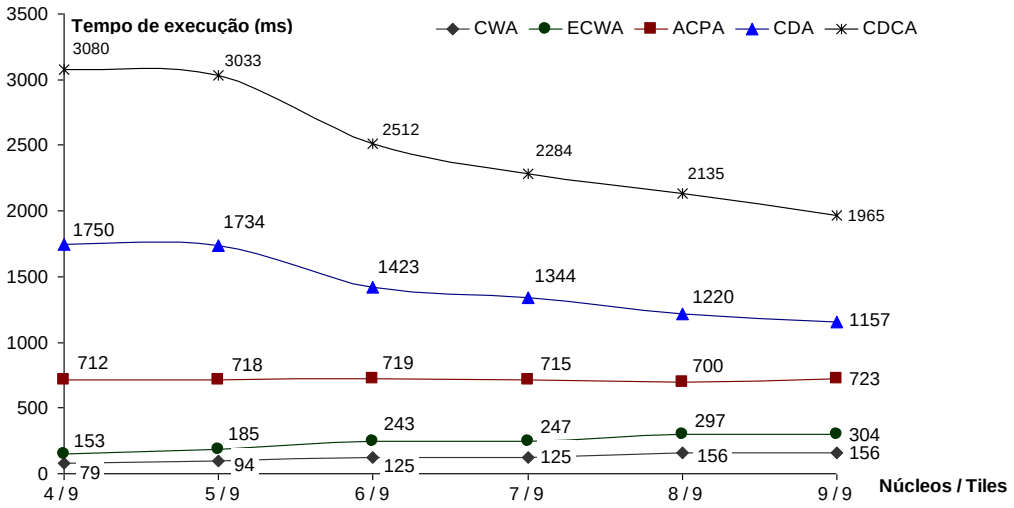


Figura 10.27: Efeito do número de núcleos e tiles no tempo de execução dos algoritmos CWA, ECWA, ACPA, CDA e CDCA. O eixo "Núcleos / Tiles" apresenta 6 combinações de núcleos e tiles. O eixo "Tempo de Execução (ms)" ilustra, para cada modelo de aplicação, o tempo de execução médio (em milisegundos) de 3 aplicações sintéticas.

O resultado deste experimento está ilustrado na Figura 10.27, que contém, para as três aplicações sintéticas, o tempo médio da função objetivo de cada algoritmo, donde conclui-se que:

1. Para algoritmos CWA e ECWA, a variação do número de núcleos, mantendo-se o número de tiles provoca uma variação linear no tempo de execução. Dada a natureza do problema de mapeamento, conclui-se que os algoritmos de mapeamento, se implementados com pesquisa exaustiva, terão um tempo de execução inaceitável, para mapear poucos núcleos (ou até mesmo um único núcleo) em redes intrachip com muitos tiles;

2. Para os algoritmos CDCA e CDA, a redução do número de núcleos, mantendo o número de tiles, aumentou linearmente o tempo de execução. A justificativa para tanto está na característica de dependência dos modelos subjacentes. A redução do número de núcleos sem reduzir o número de vértices aumenta o número de dependências de cada núcleo. Em conseqüência, os algoritmos gastam mais tempo pesquisando pelas dependências de cada vértice.

3. Por fim, uma vez mantido o número de tiles e vértices da aplicação o tempo de execução do algoritmo ACPA não é alterado pela variação do número de núcleos.

O último experimento avalia a influência de características da aplicação no tempo de execução da função objetivo de cada modelo. Neste experimento, foram utilizadas aplicações sintéticas com o mesmo número de vértices, alterando apenas as arestas de cada grafo. Estas alterações permitem que sejam elaboradas aplicações com maior ou menor grau de dependência versus concorrência (**gd_gc**). As aplicações sintéticas foram construídas de forma a terem o **gd_gc** variando de 0% a 100%

(detalhamento de como são avaliados os graus de concorrência e dependência podem ser vistos no Apêndice B).

A Figura 10.28 mostra que os algoritmos CWA e ECWA não são afetados pelas características de dependência e concorrência, enquanto que os algoritmos CDA e CDCA podem ocasionar uma multiplicação do tempo dos tempos de execução por um fator que chega a 3, quando comparado uma aplicação puramente concorrente com uma aplicação puramente dependente. Os resultado obtidos com ACPA, por sua vez, mostraram que estas características afetam parcialmente o tempo de execução do algoritmo, devido a mensagens dependentes serem lançadas na infra-estrutura de comunicação em tempos distintos, enquanto que mensagens concorrentes são lançadas no mesmo instante de tempo.

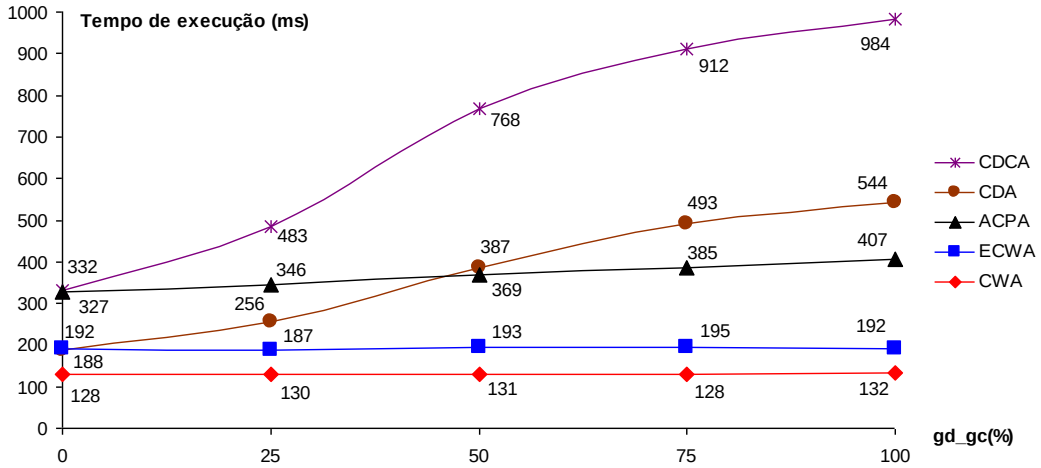


Figura 10.28: A influência da concorrência e da dependência da aplicação no tempo de computação da função objetivo de cada algoritmo. Tempos obtidos com aplicações sintéticas puramente concorrentes, puramente dependentes e mistas.

10.3.2 Consumo de Energia e Tempo de Comunicação

Esta Seção mostra estimativas de consumo de energia e tempo de execução para mapeamentos de aplicações embarcadas e sintéticas. Alguns mapas são obtidos pela aplicação dos algoritmos de mapeamento CWA, CDA, ACPA e CDCA, enquanto outros são obtidos aleatoriamente. Os mapas são comparados de forma a avaliar a necessidade de existirem algoritmos de mapeamento e verificar qual modelo obtém mapas de melhor qualidade.

Com o objetivo de quantificar a influência da atividade de mapeamento na redução do consumo de energia de infra-estruturas de comunicação, foi realizado um experimento contendo 8 aplicações embarcadas, todas descritas em CWG. Estas descrições são entradas do algoritmo CWA que, para cada descrição, estimou o consumo de energia de todos os mapas gerados. Para cada aplicação, são armazenados o menor e o maior consumo de energia, bem como o consumo de energia médio. A diferença entre o maior e o menor consumo de energia indica a penalização máxima de consumo de energia que um projeto teria se não considerar cuidadosamente a atividade de mapeamento. De forma análoga, a diferença entre a média e o menor consumo de energia indica a penalização média de consumo de energia. Os resultados obtidos destes

experimentos estão ilustrados na Tabela 10.2. A análise desta mostra que, utilizando o modelo CWM, é possível reduzir em muito o consumo de energia da rede intrachip. Por exemplo, para o pior caso de uma aplicação, chegou-se à redução de quase 70%, e a média de redução do consumo de energia é em torno de 50%. Estes resultados salientam a importância que a atividade de mapeamento tem na redução do consumo de energia para redes intrachip.

Tabela 10.2: Avaliação do consumo de energia de NoCs do tipo malha, com roteamento XY e chaveamento wormhole, sendo estimuladas por tráfegos diversas aplicações embarcadas. Na avaliação, realizada com o algoritmo CWA, são ilustrados os consumos de energia mínimo, máximo e médio, em μJ, obtidos durante os mapeamentos. Para cada aplicação, são calculados os percentuais de redução dos valores máximo e médio para o mínimo, bem como os valores médios de todas as reduções percentuais.

Aplicação[46]	Consumo de energia (μJ)			Redução para o mínimo (%)	
	Mínimo	Máximo	Médio	Máximo	Médio
FFT (2 × 3)	170.64	217.94	193.57	21.70%	11.85%
DropInsert (3 × 3)	122.34	233.33	168.43	47.57%	27.36%
VOPD (3 × 4)	135.97	268.75	183.66	49.41%	25.97%
SDH (3 × 4)	738.12	1497.45	1022.11	50.71%	27.78%
MPEG4 (4 × 3)	274.55	654.49	374.61	58.05%	26.71%
MWD (4 × 3)	42.98	86.37	59.94	50.24%	28.30%
MultiMidia System (4 × 4)	30654.77	70360.90	42328.76	56.43%	27.58%
PABX (5 × 5)	54011.18	179536.50	81013.89	69.92%	33.33%
Romberg (7 × 7)	1068.60	2020.20	1426.10	47.10%	25.07%
Média				50.13%	25.99%

De forma análoga ao experimento anterior, foi realizado um experimento com 8 aplicações (3 embarcadas e 5 sintéticas), onde o objetivo era quantificar a influência da atividade de mapeamento na redução do tempo de execução de aplicações que têm redes intrachip como infra-estrutura de comunicação. Neste segundo experimento, as aplicações foram descritas em CDG, pois o modelo subjacente consegue capturar melhor a noção de tempo que o modelo CWM, utilizado no experimento anterior. Novamente, para todas as aplicações foram obtidos os maiores e menores tempos de execução e a média. Os resultados obtidos estão ilustrados na Tabela 10.3.

Tabela 10.3: Avaliação do tempo de execução de aplicações embarcadas e sintéticas, tendo NoCs do tipo malha, com roteamento XY e chaveamento wormhole como infra-estruturas de comunicação. Na avaliação, realizada com o algoritmo CDA, são ilustrados os tempos de execução mínimo, máximo e médio, em milisegundos, obtidos durante os mapeamentos. Para cada aplicação, são calculados os percentuais de redução dos valores máximo e médio para o mínimo, bem como os valores médios de todas

[46] DropInsert (MORAES; CALAZANS; MARCON; MESQUITA; PALMA; BLAUTH, 2003) e SDH (MARCON et al, 2005e) (CALAZANS; MORAES; MARCON; PALMA, 2005) são aplicações *dataflow* de telecomunicações, VOPD e MWD (BERTOZZI et al., 2005) são aplicações de imagem. As demais aplicações (PABX, Romberg, MPEG4, MultiMidia System e FFT) estão descritas no Apêndice A.

as reduções percentuais.

Aplicação	Tempo de execução da aplicação (milisegundos)			Redução para o mínimo (%)	
	Mínimo	Máximo	Médio	Máximo	Médio
FFT (2 × 3)	1377	2395	1893	42.51%	27.26%
PABX (5 × 5)	23238	77691	49097	70.09%	52.67%
App1 (4 × 4)	7456	16597	9897	55.08%	24.66%
App2 (6 × 6)	14323	31756	22130	54.90%	35.28%
Romberg (7 × 7)	4994	12125	8014	58.81%	37.68%
App3 (8 × 8)	35089	99920	67390	64.88%	47.93%
App4 (10 × 10)	79844	217577	143454	63.30%	44.34%
App5 (12 × 10)	132360	327008	217908	59.52%	39.26%
Média				58.64%	38.64%

A análise da Tabela 10.3 mostra uma redução média no tempo de execução da aplicação de aproximadamente 58% e 38% dos valores máximo e médio para o mínimo, respectivamente. Isto evidencia a importância que o mapeamento tem na redução do tempo de execução da aplicação.

O terceiro experimento objetiva avaliar qual o modelo de descrição de aplicações mais adequado para obter mapeamentos que reduzam o consumo de energia. Para tanto, aplicações sintéticas e embarcadas foram modeladas com CWM, ACPM, CDM e CDCM. As aplicações foram agrupadas conforme o tamanho da NoC sobre a qual foram executadas. Os mapeamentos obtidos com todas as modelagens foram comparados contra mapeamentos aleatórios, de forma a verificar a influência de cada modelo. Os resultados estão ilustrados na Figura 10.29.

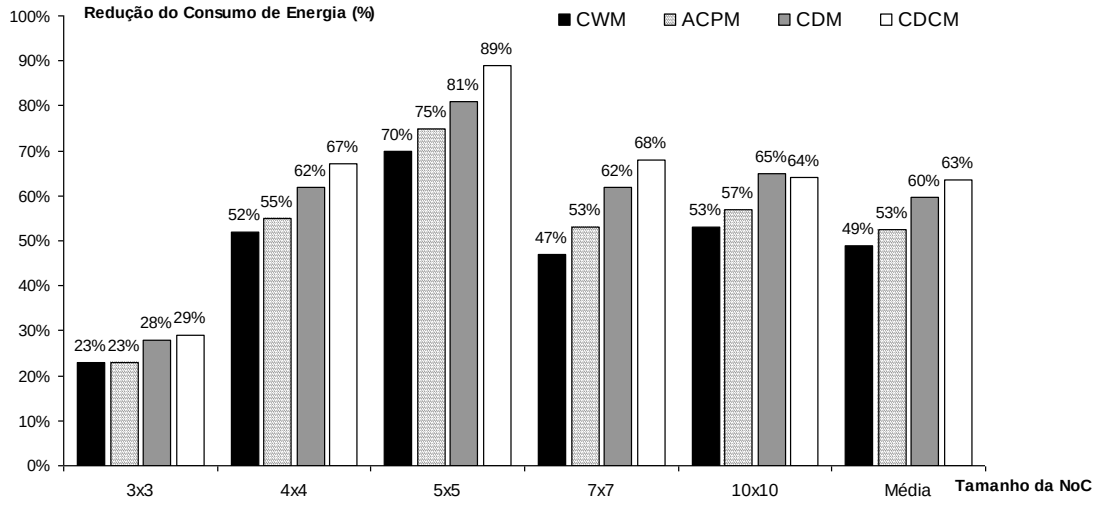


Figura 10.29: Redução percentual do consumo de energia, quando comparados mapeamentos obtidos com auxílio do CAFES com mapeamentos aleatórios. Os consumos de energia são calculados com o tráfego gerado por diversas aplicações (modeladas por CWM, ACPM, CDM e CDCM) sobre diversos tamanhos de NoC do tipo malha, com roteamento XY e chaveamento wormhole. A Figura ilustra também a média do consumo de energia para cada modelo.

Analisando a Figura 10.29, conclui-se que os modelos CDM e CDCM sempre obtêm a maior redução do consumo de energia, o que se deve a melhor estimativa das contenções. O ACPM vem em seguida, sendo ligeiramente melhor que o modelo CWM.

A vantagem do ACPM está na estimativa mais precisa do tempo de execução da aplicação. A obtenção destes valores, contudo, depende muito da implementação da NoC, pois quanto maior é o consumo de energia durante os períodos de ociosidade, maior será a diferença entre os modelos.

O último experimento é complementar ao terceiro experimento. Neste, o intuito é avaliar a redução do tempo de execução da aplicação. Estes resultados estão ilustrados na Figura 10.30.

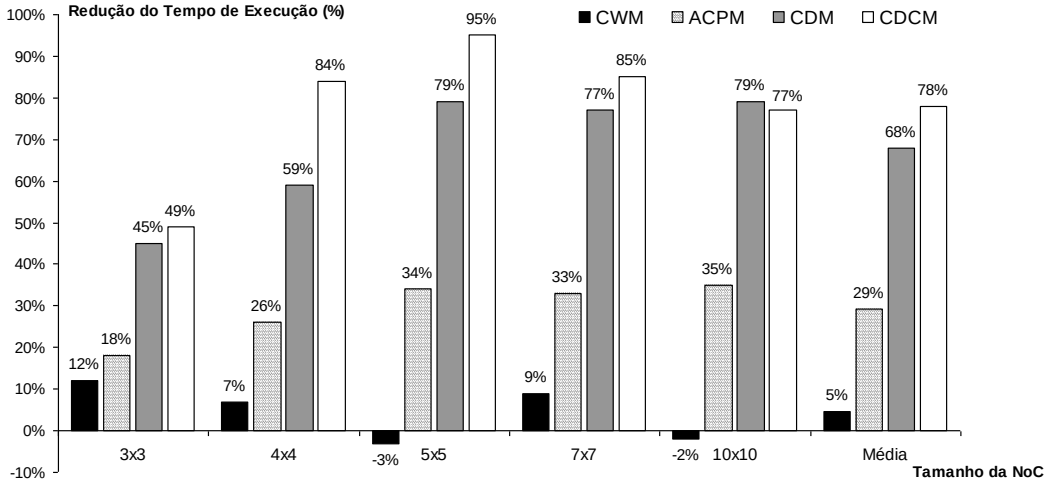


Figura 10.30: Redução percentual do tempo de execução, ao comparar mapeamentos obtidos com auxílio do CAFES com mapeamentos aleatórios. Os tempos de execução são calculados com diversas aplicações (modeladas por CWM, ACPM, CDM e CDCM) executando sobre diversos tamanhos de NoC do tipo malha, com roteamento XY e chaveamento wormhole. A Figura ilustra também o tempo de execução médio para cada modelo.

A Figura 10.30 mostra que a maior diferença entre os modelos está na capacidade destes em capturar a noção de tempo. Como o modelo CWM não é adequado para este requisito, ele dificilmente consegue obter mapas que levam a um tempo de execução menor. Por outro lado a informação de dependência, aliada ao tempo de computação, ambos presentes no modelos CDCM, fornecem a este os melhores mapeamentos com a finalidade de redução do tempo de execução.

10.4 Estado Atual do Framework CAFES e Atividades Futuras

Atualmente o *framework* CAFES implementa os algoritmos que descrevem os modelos CWM, ECWM, ACPM, CDM e CDCM para a atividade de mapeamento. O CTM é um pouco mais complexo e por este motivo projeta-se a implementação do mesmo junto com a implementação da atividade de particionamento, que não faz parte do escopo deste trabalho.

Uma modificação importante pode ser efetuada no CAFES, colocando o tamanho do pacote e o número de bits de controle que deve ser inserido em cada pacote. Com estas informações, as descrições das aplicações se tornam completamente independentes da infra-estrutura de comunicação, pois necessitam informar apenas o tamanho das mensagens e não o número de bits que trafega no meio físico.

11 COMPARAÇÃO QUALITATIVA DE MODELOS DE MAPEAMENTO

Este Capítulo tem por objetivo comparar os modelos computacionais existentes e propostos para auxiliar a solução de problemas de mapeamento, servindo como guia para avaliar quando um modelo pode ou deve ser utilizado, e quando seu uso acarreta mapeamentos de baixa qualidade. Ao final deste Capítulo, é apresentado um quadro comparativo resumido dos modelos introduzidos no Capítulo 5.

11.1 Modelo de Comunicação com Pesos (CWM)

O CWM se caracteriza pela simplicidade de modelar a aplicação, pela facilidade de obtenção do formato interno, e pela capacidade de modelar o consumo de energia dinâmica. Estas características trazem consigo as vantagens e desvantagens descritas a seguir.

11.1.1 Pontos Fortes

1. **Facilidade de obtenção do formato interno** – Para capturar a comunicação entre núcleos da aplicação, basta realizar uma simulação lógica da aplicação e a cada transmissão de um pacote por um núcleo, reconhecer o endereço do núcleo destino e contabilizar o número de bits transmitidos. Ao final de uma simulação todas as comunicações podem ser capturadas e o CWG pode ser gerado diretamente destas;
2. **Baixa complexidade computacional** – Normalmente, o número de núcleos da aplicação é pequeno se comparado com a quantidade de mensagens trocadas entre os núcleos. Desta forma, a necessidade de memória para armazenamento de CWG é relativamente pequena. O tempo de processamento gasto para executar um algoritmo que implementa CWM é pequeno, pois este algoritmo depende do caminhamento no CWG, que normalmente tem poucos vértices e arestas. A ausência de informações temporais é outro fator que reduz a complexidade e o tempo de processamento;
3. **Precisão nas estimativas de consumo de energia dinâmica** – O modelo, embora simples, oferece boa estimativa para o consumo de energia dinâmica devido ao tráfego de bits na infra-estrutura de comunicação, pois o principal elemento para o cálculo da energia dinâmica é o número de bits da comunicação. Por outro lado, outra parcela da energia dinâmica é consumida nas máquinas presentes nos roteadores, que permanecem

operando mesmo na ausência de tráfego. Esta parcela não é facilmente capturada, pois requer o conhecimento do tempo de execução da aplicação.

11.1.2 Pontos Fracos

1. **Limitado à avaliação do consumo de energia dinâmica na infra-estrutura de comunicação devido ao tráfego de bits** – A ausência de informações temporais impossibilita que seja estimado o tempo de execução da aplicação. Conseqüentemente o consumo de energia dinâmica dos circuitos durante os períodos de ausência da comunicação e o consumo de energia estática durante toda a execução da aplicação, não podem ser estimados com base apenas nos dados constantes no modelo. Para as tecnologias atuais, a parcela de consumo de energia estática é ainda pequena frente ao consumo de energia dinâmica. Contudo, para as novas tecnologias submicrônicas, é estimado que o consumo de energia estática atinja a mesma ordem de grandeza (DUARTE, 2002), não podendo mais ser negligenciado. O consumo de energia dinâmica da infra-estrutura de comunicação nos períodos de ausência de tráfego depende do tipo, tecnologia e topologia adotados. Para NoCs, por exemplo, o circuito de controle tem consumo de energia comparável com o tráfego, não podendo ser negligenciado, tal como foi ilustrado no Capítulo 9;
2. **Não habilita estimar o consumo de energia devido ao chaveamento dos bits que trafegam na infra-estrutura de comunicação** - O Capítulo 9 mostra que a omissão do número de transições pode gerar estimativas de consumo de energia com erros de até 50%;
3. **Imprecisão na avaliação da largura de banda dos canais da infra-estrutura de comunicação** - Embora alguns autores, como Hu e Marculescu (2005) e Murali e De Micheli (2004[A]), utilizem este modelo para avaliar a largura de banda da infra-estrutura de comunicação, a imprecisão obtida com esta abordagem pode ser grande. Esta imprecisão ocorre devido ao modelo não permitir estimar os instantes de tempo da comunicação levando ao uso do pior caso (todos os núcleos se comunicando ao mesmo tempo com a quantidade máxima de bits transmitidos), que em aplicações práticas provavelmente nunca irá acontecer;
4. **Redução na capacidade de estimar características do tráfego na infra-estrutura de comunicação** – A ausência de informações temporais impossibilita que este modelo estime contenções entre pacotes. O modelo é essencialmente otimista, pois assume que nenhum pacote sofre contenção em um caminho de comunicação. Para o caso de infra-estruturas de comunicação do tipo NoC, esta deficiência impossibilita que sejam dimensionadas as áreas de armazenamento temporário de entrada ou saída de cada roteador. Por este mesmo motivo, CWM não permite estimar qual é o melhor tamanho de phit para a transmissão dos pacotes, ou seja o número de linhas transportando dados entre roteadores;
5. **Inadequado para estimar topologias e roteamentos de infra-estruturas de comunicação** – A ausência de informações temporais, mais uma vez, é um fator limitante para este modelo, pois impossibilita estimar topologias

e algoritmos de roteamento que reduzam a contenção de pacotes e, conseqüentemente, o tempo de execução da aplicação.

11.2 Modelo Estendido de Comunicação com Pesos (ECWM)

ECWM é um aperfeiçoamento de CWM. Ele foi projetado para suprir a deficiência dos modelos em assumir apenas a quantidade de bits que trafega na infra-estrutura de comunicação, sem considerar o conteúdo e o efeito deste no consumo de energia. Essencialmente, todos os pontos fortes e fracos de CWM são aqui repetidos. Exceções ocorrem apenas na facilidade de captura do formato interno e na capacidade de estimar o consumo de energia dinâmica. Este último, sendo o motivo de seu projeto.

11.2.1 Pontos Fortes

Este modelo é o único dos apresentados neste trabalho que considera o efeito do conteúdo dos pacotes no consumo de energia. Essencialmente, para as conexões entre roteadores e para as áreas de armazenamento temporário dos roteadores, a variação da informação que trafega na rede influi consideravelmente no consumo de energia dinâmica, devido às cargas armazenadas em elementos capacitivos destes componentes.

11.2.2 Pontos Fracos

Parte do preço pela capacidade do modelo capturar informações sobre o conteúdo da comunicação está no aumento da complexidade de obtenção do formato interno. Para a grande maioria das aplicações, variações nas entradas podem influenciar pouco o tamanho dos pacotes, mas o conteúdo varia muito mais. Assim, para capturar o formato interno que represente o comportamento típico da comunicação, pode ser necessário trabalhar com amostragens estatísticas. A obtenção destas amostras aumenta a complexidade da captura do formato interno, quando comparada com o modelo CWM. Porém, se for possível considerar o comportamento das entradas da aplicação, podem ser obtidas boas estimativas em pouco tempo.

11.3 Modelo de Dependência das Comunicações (CDM)

CDM se caracteriza por capturar a dependência das mensagens. Se comparado com CWM ou mesmo com ECWM, esta informação torna o modelo mais complexo para capturar e tratar. Todavia, permite modelar com mais precisão características temporais da comunicação.

11.3.1 Pontos Fortes

1. **Maior precisão nas estimativas de consumo de energia e de tempo de execução da aplicação** – Para o cálculo do consumo de energia dinâmica, este modelo apresenta poucas melhoras sobre o CWM. Todavia, a informação de dependência permite estimar as mensagens que podem ser concorrentes. Com esta estimativa, algoritmos que implementam o CDM

podem elaborar mapeamentos que evitem ou pelo menos reduzam o número de mensagens que concorrem pelos mesmos recursos da infra-estrutura de comunicação. Este procedimento reduz contenções, e conseqüentemente, a execução da aplicação é potencialmente mais rápida, se comparada com a execução de aplicações cujos mapeamentos tenham sido obtidos pelos algoritmos que implementam o CWM;

2. **Estimativa do consumo mínimo de energia estática de toda a rede e dinâmica dos circuitos que operam mesmo na ausência de tráfego** – Pelo mesmo motivo descrito acima, os algoritmos que implementam o CDM estimam de forma otimista os consumos de energia estática de toda a rede intrachip e dinâmica dos circuitos que operam durante todo o tempo de execução da aplicação. Isto ocorre, porque o CDM não é capaz de informar o tempo de computação dos núcleos e conseqüentemente não pode estimar com precisão o tempo de execução da aplicação. A estimativa de tempo de execução da aplicação é dada apenas pela comunicação, sendo que o *tempo de computação é desconsiderado*[47]. Esta estimativa é muito otimista, porém, é interessante para obter o limite mínimo do tempo de execução da aplicação, e conseqüentemente de consumo mínimo de energia estática e dinâmica dos circuitos que operam mesmo na ausência de tráfego;

3. **Menor sensibilidade aos dados de entrada** – A modelagem da aplicação, quando considerada a dependência das comunicações, se torna menos sensível à variação dos dados de entrada da aplicação. Normalmente, existe uma pequena quantidade de aplicações cuja quantidade exata de bits de comunicação é conhecida a priori. Em geral, o número de bits transmitidos pode variar de acordo com os dados de entrada da aplicação. Contudo, a informação de dependência das comunicações contida no CDG não varia, pois esta reflete o algoritmo da aplicação. Desta forma, mesmo que a quantidade de bits a serem transmitidos varie, continua a existir informação de quais mensagens podem ou não ser concorrentes, habilitando algoritmos que manipulam CDMs a gerar mapeamentos de melhor qualidade, se comparado com algoritmos que manipulam CWMs. Ou seja, CDM é válido para estimar a redução das contenções mesmo quando não se sabe a priori a quantidade de comunicação e computação da aplicação;

4. **Possibilidade de avaliar topologias de infra-estruturas de comunicação** – A capacidade de avaliar topologias é obtida com o conhecimento da dependência das mensagens, que pode determinar qual o melhor caminho para cada mensagem. Ao contrário do CWM e ECWM, o CDM não supõe que todos os caminhos estão sempre livres. Os algoritmos que manipulam CDMs mapeiam as mensagens na infra-estrutura de comunicação à medida que estas são transmitidas de um núcleo para outro. Este procedimento permite avaliar a eficácia das topologias de infra-estruturas de comunicação e dos algoritmos utilizados para o roteamento de mensagens.

[47] O modelo CDM não considera o tempo de computação, mas o framework CAFES permite se este seja dado por uma constante (tipicamente um ciclo de relógio) igual para todos os núcleos.

11.3.2 Pontos Fracos

1. **Maior complexidade algorítmica** – Comparado com o CWM, este modelo implica em maior complexidade algorítmica devido a alguns fatores: (*i*) os vértices do CDG contêm mais informação que os vértices do CWG; (*ii*) o número de vértices do CDG pode ser ordens de grandeza maior que o número de vértices do CWG, porque aplicações costumam possuir muito mais mensagens do que núcleos; (*iii*) o maior problema da complexidade algorítmica está associado à forma como CDM pode ser implementado. Para obter vantagem da informação de dependência das mensagens, a cada vez que uma mensagem é transmitida esta deve ser anotada em todos os recursos de comunicação, informando suas dependências, para que o algoritmo possa prever se outras mensagens sofrerão ou não contenção. Esta implementação implica maior consumo de memória para armazenar as dependências em todos os recursos. Outras soluções podem ser adotadas para reduzir o consumo de memória, tal como investigar o grafo toda a vez que uma mensagem é mapeada sobre a NoC para verificar suas dependências, porém esta abordagem tende a consumir muito tempo de execução;
2. **Imprecisão do modelo de consumo de energia estática** – O cálculo da energia estática em CDMs é mais preciso que em CWMs. Por outro lado, a ausência do tempo de computação não permite estimar com exatidão o consumo de energia estática da maioria das aplicações práticas, mormente aquelas que são computacionalmente intensivas;
3. **Dificuldade de extração do formato interno** – Para obter a dependência das comunicações é necessário entender a semântica da aplicação, e esta informação não pode ser obtida por simulação, já que esta fornece apenas uma das ordens possíveis de envio de mensagens, mas não as dependências. A análise semântica da aplicação é uma tarefa por demais complexa. Por este motivo, o CDG normalmente é extraído manualmente pelo projetista, o que o torna mais suscetível a erros de descrição e implica maior tempo para extração do formato interno. Assim, a dificuldade de extração do formato interno é provavelmente um dos pontos mais fracos deste modelo.

11.4 Modelo de Dependência das Comunicações e Computação (CDCM)

CDCM é um super-conjunto do CDM, pois acrescenta ao primeiro a informação de computação de cada núcleo da aplicação.

11.4.1 Pontos Fortes

1. **Aumenta a precisão na estimativa de tempo de execução da aplicação** – CDCM insere o tempo de computação de cada núcleo da aplicação. Esta informação permite alcançar estimativas mais precisas que as obtidas por CDM. Além do mais, o critério adotado para representar o tempo de computação que precede o envio de uma mensagem torna a modelagem do

problema de mapeamento totalmente independente do atraso causado pelo meio de comunicação, pois o tempo é contado a partir da liberação de todas as dependências da mensagem. Assim, ao descrever a aplicação, o projetista necessita apenas obter a computação de cada núcleo, e esta independe da comunicação da aplicação;

2. **Precisão no cálculo do consumo de energia dinâmica** – O modelo permite estimar o consumo de energia dinâmica com precisão superior aos modelos CWM e CDM. Isto ocorre, pois além de representar de forma equivalente o número de bits que trafega na infra-estrutura de comunicação, habilita capturar o tempo de execução da aplicação e conseqüentemente o consumo de energia dinâmica dos circuitos que operam mesmo na ausência de tráfego;
3. **Maior precisão no consumo de energia estática** – Para aplicações onde se conhece a priori a necessidade de computação e comunicação, este modelo consegue avaliar de forma mais precisa o tempo de execução da aplicação e conseqüentemente o consumo de energia estática de toda a aplicação;
4. **Maior capacidade para estimar a necessidade de recursos na infra-estrutura de comunicação** – O conhecimento preciso do tempo que cada mensagem despende trafegando na infra-estrutura de comunicação permite que sejam estimadas com precisão certas características da infra-estrutura. Por exemplo, para NoCs, é possível estimar o tamanho das áreas de armazenamento temporário, qual a melhor topologia, roteamento e chaveamento que melhor atendem os requisitos de redução de consumo de energia e tempo de execução da aplicação.

11.4.2 Pontos Fracos

1. **Inadequado para aplicações com imprevisibilidade do tempo de computação** – CDCM não é adequado quando não se pode determinar a priori o tempo de computação dos núcleos da aplicação com algum nível de precisão, ou quando a aplicação tem um comportamento imprevisível, de forma que não seja possível representar o tempo de computação dos núcleos por constantes;
2. **Dificuldade na extração do formato interno** – Semelhante ao CDG, o processo de extração do formato interno é normalmente manual. Todavia, o tempo de computação dos núcleos pode ser obtido de forma automática pela execução da aplicação sem levar em consideração a estrutura de comunicação, bastando para isto registrar o tempo entre a liberação das dependências da mensagem e o envio da mesma. Assume-se para tanto que o tempo de computação de cada núcleo da aplicação seja estimado sobre os mesmos elementos de processamento que serão utilizados na arquitetura alvo;
3. **Complexidade algorítmica** – Este modelo tem complexidade algorítmica semelhante ao CDM. Se por um lado a inserção da computação aumenta a complexidade do modelo, por outro lado esta mesma informação reduz a complexidade. O aumento de complexidade é notado pela forma que o

algoritmo é executado e pelo tratamento do tempo de execução da aplicação, pois no cálculo do tempo total deve ser contabilizada a computação de cada núcleo da aplicação. Todos os caminhos do CDCG são executados concorrentemente sobre a infra-estrutura de comunicação, fazendo com que o tempo de execução da aplicação seja naturalmente computado na infra-estrutura de comunicação, a cada liberação de um vértice do CDCG (uma mensagem). A redução da complexidade algorítmica ocorre devido à ausência da necessidade de armazenar as dependências das mensagens em cada recurso da infra-estrutura de comunicação, já que o algoritmo determina exatamente o tempo que uma mensagem irá ocupar um recurso. Ou seja, a *abordagem pessimista*[48] de CDM não ocorre aqui.

11.5 Modelo do Padrão de Comunicações da Aplicação (ACPM)

A maneira como o formato interno é capturado permite que o ACPM seja utilizado para modelar aplicações grandes, semelhante ao modelo CWM. Ao mesmo tempo, ele consegue tratar em mais detalhe as informações temporais, que o CWM não aborda. Estas características trazem consigo as vantagens e desvantagens descritas a seguir.

11.5.1 Pontos Fortes

1. **Facilidade de obtenção do formato interno** – Para capturar a comunicação entre núcleos, basta simular a aplicação. A cada transmissão de uma mensagem é preciso associá-la com o tempo de execução atual, que depende do passo da simulação. Para cada mensagem transmitida é necessário obter os núcleos origem e destino, e o número de bits. Além disto, é necessário avaliar o tamanho do pacote na infra-estrutura de comunicação para saber quantos bits devem ser acrescidos ao tamanho da mensagem. Não existe dependência entre as mensagens, mas a informação de tempo força uma ordenação natural. A complexidade de obtenção do formato interno é muito menor se comparada com CDM e CDCM, pois este modelo captura uma das ordens possíveis que as mensagens são transmitidas, mas não a dependência entre estas;
2. **Média complexidade computacional** – Dado que o modelo é voltado a eventos, sistemas assim modelados são facilmente simulados, bastando que as mensagens sejam lançadas na infra-estrutura de comunicação nos instantes definidos. A complexidade computacional é considerada média, pois é superior a do CWM e inferior a dos modelos CDM e CDCM;
3. **Precisão nas estimativas de consumo de energia dinâmica devido ao tráfego** – O modelo tem boa precisão para estimar o consumo de energia

[48] A abordagem pessimista do CDG considera que todas as mensagens que **podem** causar contenção **irão** causar contenção, e por este motivo são pesquisados mapeamentos que evitem que mensagens concorrentes utilizem os mesmos recursos da infra-estrutura de comunicação.

dinâmica, dado que o mesmo considera o número de bits que trafega na infra-estrutura de comunicação;

4. **Possibilidade de avaliar contenções** – Este modelo, assim como CDM e CDCM, possibilita a avaliação de contenções de pacotes na infra-estrutura de comunicação, devido à capacidade de modelar mensagens concorrentes. Todavia, para que isto seja possível, a granularidade do passo de simulação deve ser pequena. Quanto mais próximo do ciclo de relógio, maior será a precisão na avaliação da concorrência de mensagens;
5. **Possibilidade de avaliar topologias de infra-estruturas de comunicação** – Por simulação, este modelo permite avaliar diversas topologias e algoritmos de roteamento de infra-estruturas de comunicação. Para tanto, como no item anterior, o passo de simulação deve ser próximo ao ciclo de relógio.

11.5.2 Pontos Fracos

1. **Imprecisão no cálculo do tempo de execução da aplicação** – Os instantes de tempo de início de transmissão das mensagens são obtidos por simulação da aplicação em uma arquitetura que não é necessariamente aquela na qual a aplicação será implementada. Além do mais, passos de simulação não refletem necessariamente o tempo real. Desta forma, os tempos obtidos não são precisos para estimar o tempo de execução da aplicação na arquitetura alvo;
2. **Dificuldade de extração do paralelismo da aplicação** – ACPM não captura as dependências entre mensagens, e conseqüentemente impossibilita a avaliação de quais mensagens podem ser concorrentes e quais não podem;
3. **Imprecisão na estimativa de consumo de energia estática** – Semelhante a CDM, a ausência do tempo de computação não permite estimar com exatidão o consumo de energia estática da maioria das aplicações práticas, mormente aquelas que são computacionalmente intensivas. Esta característica o torna menos adequado que o CDCM para estimar o consumo de energia estática.

11.6 Modelo de Tarefas da Comunicação (CTM)

O CTM é o mais complexo dos modelos aqui descritos. A sua grande vantagem está no detalhamento de aplicações, pois o mesmo permite tratar até deadlines de tarefas, imprescindível para aplicações de tempo real.

11.6.1 Pontos Fortes

1. **Aumenta o detalhamento da aplicação** – CTM permite representar aplicações através de suas tarefas. Este representação tem grão menor que os modelos que representam aplicações através de núcleos. Com o detalhamento em nível de tarefas, é possível atender melhor alguns

requisitos ou restrições de projeto, tal como a limitação do consumo de energia por parte de um núcleo;

2. **Possibilidade de tratar sistemas de tempo real** – A inserção dos deadlines das tarefas permite estimar escalonamentos, particionamentos e mapeamentos de tarefas que atendam a requisitos de tempo real;
3. **Capacidade de descrever o consumo de energia dinâmica de toda a aplicação** – CTM associa a cada tarefa um consumo de energia dinâmica no PE no qual será executada. Desta forma, é possível estimar o consumo de energia dinâmica de toda a aplicação, e não apenas da infra-estrutura de comunicação.

11.6.2 Pontos Fracos

1. **Necessidade de descrição dos elementos de processamento em baixo nível** – O modelo associa a cada tarefa a computação e o consumo de energia. Estas informações devem ser passadas pelo projetista durante a extração do formato interno. Para que seja possível estimar o tempo de execução médio de cada tarefa, é necessário executá-la no PE, com precisão de ciclo de relógio, obrigando uma descrição de baixo nível de abstração. Esta obrigação é ainda maior quando o projetista deseja estimar o consumo de energia. Neste último caso, estimativas precisas são obtidas apenas no nível elétrico;
2. **Dificuldade na extração do formato interno** – A extração do formato interno depende essencialmente do número de tarefas da aplicação, do número de PEs e do número de estimativas consideradas. O número de tarefas implica dois problemas: (*i*) a dificuldade na construção do grafo de dependências das tarefas que, semelhante ao CDCM, dificilmente pode ser automatizada; e (*ii*) a informação de deadline das tarefas, que dificilmente pode ser extraída automaticamente. Por sua vez, quanto maior o número de PEs, maior a complexidade para extrair informações de consumo de energia e tempo de computação, pois estas informações são relativas à associação de tarefas com PEs. Ou seja, o número de caracterizações necessárias para modelar a aplicação é dado pelo produto $tarefas \times PEs \times estimativas$. Supondo que seja necessário estimar o tempo de execução e o consumo de energia de uma aplicação composta por 32 tarefas, considerando que estão poderão ser executadas em 8 diferentes PEs, implicaria em 512 caracterizações ($32 \times 8 \times 2$). Este exemplo mostra que, mesmo para uma aplicação de médio porte, e tendo ferramentas de estimativa de alto nível que permitam caracterizar rapidamente cada tarefa sobre cada PE, a extração do formato interno se torna complexa devido ao número de combinações possíveis;
3. **Modelo de comunicação resumido** – CTM modela apenas a quantidade total de comunicação entre tarefas. Este modelo, de forma análoga ao CWM, impossibilita que seja estimado o tempo devido às contenções na infra-estrutura de comunicação, bem como o consumo de energia estática de todo o sistema, e a energia dinâmica dos circuitos que chaveiam mesmo na ausência de tráfego;

4. **Alta complexidade algorítmica** – Dada a grande quantidade de variáveis, as soluções algorítmicas têm alta complexidade, implicando muito tempo de processamento e consumo de memória.

11.7 Quadro Resumo Comparativo dos Modelos

O quadro comparativo geral de todos os modelos estudados no presente trabalho é mostrado na Tabela 11.1. Com o objetivo de *comparar qualitativamente* os modelos frente às características relevantes, são atribuídas notas entre 0 a 6 para cada modelo, de acordo com as ponderações descritas neste Capítulo. A nota 0 indica que o modelo não tem capacidade de tratar a característica em questão, enquanto que as notas de 1 a 6 mostram a capacidade relativa do modelo de tratar a característica, sendo 1 a capacidade mínima e 6 a capacidade máxima.

Tabela 11.1: Quadro comparativo entre os modelos para a atividade de mapeamento.

Características	Modelos					
	CWM	ECWM	CDM	CDCM	ACPM	CTM
Capacidade de estimar consumo de energia dinâmica devido ao tráfego	5	6	5	5	5	5
Capacidade de estimar consumo de energia dinâmica dos circuitos que operam mesmo na ausência de tráfego	0	0	3	6	4^{*}	3
Capacidade de estimar consumo de energia estática	0	0	3	6	$1^{\#}$, 4^{*}	2
Qualidade da estimativa do tempo de comunicação e/ou execução da aplicação	0	0	4	6	$1^{\#}$, 5^{*}	4
Facilidade de extração do formato interno	6	5	3	2	4	1
Complexidade do formato interno (modelo e área de armazenamento)	1	2	4	5	$2^{\#}$, 4^{*}	6
Complexidade algorítmica associada ao formato interno	1	1	5	5	2	6
Capacidade de estimar ou tratar comportamentos não determinísticos	1	1	4	2	2	2
Capacidade estimar o tráfego em um determinado instante na infra-estrutura de comunicação	0	0	4	6	$2^{\#}$, 5^{*}	0
Capacidade estimar topologias de infra-estruturas de comunicação	1	1	4	5	$2^{\#}$, 4^{*}	4
Capacidade de avaliar com precisão mensagens concorrentes e não concorrentes	0	1	6	6	$2^{\#}$, 5^{*}	4
Capacidade modelar aplicações de tempo real	0	0	0	0	0	6
Viabilidade para descrever aplicações complexas	6	5	3	1	4^{*}, $5^{\#}$	1
Capacidade de detalhar a comunicação da aplicação	1	2	3	5	3	1
Capacidade de detalhar a computação da aplicação	0	0	0	0	0	5

Notas: # – considerando um passo de simulação muito grande;
* – considerando um passo de simulação próximo ao ciclo de relógio.

12 CONCLUSÕES E TRABALHOS FUTUROS

O mapeamento de núcleos IP em infra-estruturas de comunicação influencia o atendimento de requisitos e/ou restrições de projeto de SoCs. Esta tarefa é de tal complexidade que se não for auxiliada por ferramentas computacionais, dificilmente gerará mapeamentos de qualidade, o que pode comprometer o desempenho do SoC com um todo, seja em termos de velocidade de operação ou consumo de energia, ou ambos. Este trabalho procurou contribuir para a solução deste problema.

As ferramentas computacionais utilizadas neste trabalho têm como entrada modelos computacionais que modelam aplicações e infra-estruturas de comunicação. Os modelos aqui explorados habilitam estimar diversas características de implementação de projetos, tais como o consumo de energia da infra-estrutura de comunicação e a latência esperada do sistema.

A complexidade para capturar as características relevantes de uma aplicação depende do modelo computacional subjacente. Se por um lado a complexidade aumenta devido ao uso de um modelo mais detalhado, também aumenta a capacidade deste para capturar informações que permitam estimar melhor características do projeto em vista.

Quando se iniciou este trabalho, somente existiam na literatura modelos voltados para a solução do problema de mapeamento que consideravam apenas a quantidade da comunicação (CWM), e existiam poucas ferramentas de apoio à atividade de mapeamento. Hoje, existem pelo menos seis classes de modelos e mais um razoável ferramental de apoio. Esta diversidade faz com que a solução do problema de mapeamento possa atender um maior espectro de requisitos de projeto. Por exemplo, neste trabalho mostrou-se que considerar apenas a quantidade bits que trafega na infra-estrutura de comunicação não é suficiente para estimar com precisão a energia consumida. Para certas partes da infra-estrutura de comunicação, como *buffers* e canais de comunicação, a influência da transição entre bits consecutivos no consumo de energia é da mesma ordem de grandeza que a quantidade de bits. Sendo assim, negligenciar uma destas informações leva a estimativas de consumo de energia de baixa confiabilidade. Este é o acréscimo que o modelo ECWM traz frente ao modelo CWM. Outro exemplo do aumento do espectro de requisitos está na avaliação do tempo de comunicação. Este requisito é alcançado, por exemplo, com modelos como CDM e CDCM graças à modelagem da dependência entre mensagens.

A escolha do modelo da aplicação depende basicamente de três fatores: (*i*) requisitos e restrições de projeto; (*ii*) características da aplicação; e (*iii*) *time-to-market*. Os requisitos e restrições de projeto influenciam a escolha dos modelos porque dependem de informações que apenas alguns modelos habilitam capturar. Por exemplo, o requisito de redução de tempo de execução de uma aplicação, não é bem atendido pelo modelo CWM, pois este não captura tempo. A influência das características de uma aplicação na escolha do modelo é facilmente observada, quando a informação que

caracteriza um modelo não é expressa na aplicação por este modelada. Por exemplo, modelar uma aplicação essencialmente concorrente com CDM ao invés de CWM trará pouco ou nenhum benefício, pois a informação de dependência do modelo não será explorada. A influência do *time-to-market* na escolha do modelo ocorre principalmente na extração da descrição de uma aplicação. Por exemplo, descrições, cujos modelos subjacentes são o CDM ou o CDCM implicam hoje em extração manual, o que consome muito tempo além de ser sensível a erros. Por outro lado, caso o modelo subjacente à descrição seja, por exemplo, ECWM ou ACPM, a extração pode ser automatizada, reduzindo o tempo de projeto e contribuindo para reduzir o *time-to-market.* O metamodelo QOD vem no sentido de auxiliar a escolha de modelos para o mapeamento. A proposta do metamodelo QOD sistematiza a influência de características da aplicação, permitindo a seleção adequada de um subconjunto de modelos para cada aplicação especificada, considerando a simplicidade, aplicabilidade e grau de precisão dos modelos.

Estimativas do tempo de execução de aplicações em alto nível têm precisão igual às estimativas obtidas com simulações em nível de ciclo relógio. Isto mostra a baixa complexidade do modelo de tempo, que permite estimar com qualidade a latência de uma aplicação e o consumo de energia estática. Por outro lado, as estimativas de consumo de energia de infra-estruturas de comunicação modeladas em alto nível são bastante diferentes das estimativas obtidas com simulações RTL. Estas diferenças passam a ser ainda mais significativas quando as estimativas de alto nível são comparadas com estimativas elétricas. Este fato aponta para a complexidade de modelar o consumo de energia de infra-estruturas de comunicação em alto nível. Neste sentido, salienta-se que neste trabalho a modelagem do consumo de energia é essencialmente linear com o tráfego de bits, e esta estimativa mostra-se imprecisa, pois parte dos circuitos de uma rede intrachip consomem energia independente do tráfego.

O ferramental apresentado aqui, tendo como ponto central o *framework* CAFES, permitiu comparar os modelos de aplicação com relação a diversos elementos como complexidade de implementação algorítmica, qualidade dos mapeamentos obtidos e facilidade de captura do formato interno.

Este trabalho abriu caminhos que poderão ser empregados para desenvolver diversos trabalhos futuros, incluindo:

1. *Extração automática de dependências* de uma aplicação, para que modelos que capturam a informação de dependência possam ser rapidamente gerados e com menor probabilidade de erro;
2. Elaboração de *algoritmos heurísticos*, que consigam pesquisar por soluções com maior eficiência que as existentes;
3. Avaliação do *particionamento de tarefas concomitante com o mapeamento de núcleos*, de forma a atender melhor requisitos de projeto que não são completamente atendidos apenas com a atividade de mapeamento;
4. *Refinamento dos modelos de consumo de energia* de redes intrachip em alto nível, para que estes habilitem estimativas mais precisas;
5. Estudo e proposta de *modelos de aplicação que aumentem a qualidade das estimativas de mapeamento*, tal como um modelo que seja construído a partir da inserção da informação de transição de bits no modelo CDCM;

6. Estudo e proposta de *modelos de aplicação e infra-estrutura de comunicação que permitam estimar requisitos de projeto não abordados neste trabalho*, tais como a redução ou eliminação de *hot-spots*, ampliando o espectro de requisitos de projeto que podem ser atendidos;
7. Proposta de *infra-estruturas de comunicação de baixo consumo de energia*, tais como redes intrachip implementadas com roteadores assíncronos, de forma a reduzir, ou até mesmo eliminar, o consumo de energia dinâmica existente nos intervalos de ociosidade (ausência de tráfego).

REFERÊNCIAS

ACTON, F. S. **Numerical Methods that Usually Work**. 2nd ed. Washington DC: Mathematical Association of America, 1990.

ANDRIAHANTENAINA, A. et al. SPIN: A Scalable, Packet Switched On-Chip Micro-Network. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE 2003, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2003. p. 70-73.

ARM CORPORATION. **AMBA 2.0 Specification**. Disponível em: <http://www.arm.com/armtech/AMBA_Spec>. Acesso em Mar. 2003.

BARROS, E. **Hardware/Software Partitioning Using UNITY**. 1993. Ph.D. Thesis (Computer Science) - Tübingen University, Tübingen.

BERKELEY UNIVERSITY. **SPICE User Manual**. Disponível em: <http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/>. Acesso em Out. 2005.

BENINI, L.; DE MICHELI, G. Networks on Chips: A New SoC Paradigm. **Computer**, Los Alamitos, v. 35, n.1, p. 70-78, Jan. 2002.

BENVENISTE, A.; BERRY, G. The Synchronous Approach to Reactive and Real-Time Systems. **Proceedings of the IEEE**, Los Alamitos, v. 79, n. 9, p. 1270-1282, Sep. 1991.

BERGERON, J. **Writing Testbenches**: Functional Verification of HDL Models. 2nd ed. Boston: Kluwer Academic Publishers, 2002.

BERTOZZI, D. et al. NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip. **IEEE Transaction on Parallel Distributed Systems**, New York, v. 16, n. 2, Feb. 2005.

BHATTACHARYYA, S. et al. **PTOLEMY 0.7 User's Manual**. Berkeley: University of California, 1997. v. 1.

BOLOTIN, E. et al. QNoC: QoS Architecture and Design Process for Network on Chip. **Journal of Systems Architecture**, North-Holland, v. 50, n. 2-3, p. 105-128, Feb. 2004.

BORIN, A. **SegImag:** Aplicação de Segmentação de Imagens. Comunicação pessoal. UFRGS, 2004.

BRIGHAM, E. O. **The Fast Fourier Transform and Its Applications**. Englewood Cliffs, NJ: Prentice Hall, 1988.

CALAZANS, N. L. V. **Projeto Lógico Automatizado de Sistemas Digitais Seqüenciais**. Rio de Janeiro: DCC/IM, 1998. Trabalho apresentado na 11ª Escola de Computação.

CALAZANS, N. L. V; MORAES, F. G.; MARCON, C. A. M.; PALMA, J. C. S. Design, Validation and Prototyping of the EMS SDH STM-1 Mapper Soft-core. In:

IEEE LATIN-AMERICAN TEST WORKSHOP, 6th, 2005. Digest of Papers. [S.l.:s.n.], 2005. p. 313-318.

COHEN, D. I. A. **Introduction to Computer Theory.** 2nd ed. New York: John Wiley & Sons, 1997.

DALLY, W. J.; TOWLES, B. Route Packets, Not Wires: On-Chip Interconnection Networks. In: DESIGN AUTOMATION CONFERENCE, DAC, 38th, 2001, Las Vegas. **Proceedings…** New York: ACM, 2001, p. 684-689.

DALLY, W. J.; TOWLES, B. **Principles and Practices of Interconnection Networks**. San Francisco: Morgan Kaufmann Publishers, 2004. 550p.

DE MICHELI, G. Hardware/Software Codesign: Application Domains and Design Technologies. In: NATO ADVANCED STUDY INSTITUTE IN HARDWARE/SOFTWARE CO-DESIGN, 1995, LOCAL. **Proceedings…**, Kluwer Academic Publishers, 1995.

DUARTE, D. et al. Impact of Scaling on the Effectiveness of Dynamic Power Reduction Schemes. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 20th, 2002, Freiburg. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2002. p. 382-387.

DUATO, J. et al. **Interconnection Networks**. Los Alamitos: IEEE Computer Society Press, 1997. 515p.

DUATO, J; YALAMANCHILI, S; NI, L. **Interconnection Networks an Engineering Approach**. San Francisco: Morgan Kaufmann Publishers, 2003. 600p.

EDWARDS, S. et al. Design of Embedded Systems: Formal Models, Validation and Synthesis. **Proceedings of the IEEE**, New York, v. 85, n. 3, p. 336-390, Mar. 1997.

ERNST, R.; HENKEL, J. Hardware/Software Codesign of Embedded Controllers based on Hardware Extraction. In: INTERNATIONAL WORKSHOP ON HARDWARE/SOFTWARE CODESIGN, 1992, Estes Park. **Proceedings…** New York: ACM, 1992.

GAJSKI, D. D.; KUHN, R. H. Guest Editor's Introduction: New VLSI Tools. **IEEE Computer**, New York, v. 16, n. 12, Dec. 1983.

GIUSTO, P.; DEMMELER, T. Rapid Design Exploration of Safety-Critical Distributed Automotive Applications via Virtual Integration Platforms. **Journal of Systems and Software**, [S.l.], v. 70, n. 3, p. 245-262, Mar. 2004.

GUERRIER, P.; GREINER, A. A Generic Architecture for On-chip Packet-switched Interconnections. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE 2000, Paris. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2000. p. 250-256.

GUPTA, R. et al. Program Implementation Schemes for Hardware-Software Systems. **IEEE Computer**, New York, p. 48-55, Jan. 1994.

HENNESSY, J. L.; PATTERSON, D. A. **Arquitetura de Computadores:** Uma Abordagem Quantitativa. 3ª ed. Rio de Janeiro: Campus, 2003.

HESSEL, F.; ROSA, V.; REIS, I.; PLANER, R.; MARCON, C. A. M.; SUSIN, A. Abstract RTOS Modeling for Embedded Systems. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, RSP'04, 15th, 2004, Geneva. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2004. p. 210-216.

HESSEL, F.; ROSA, V.; MARCON, C. A. M.; SANTOS, T. G. S. Scheduling Refinement in Abstract RTOS Models. **ACM Transactions on Embedded Computing Systems**, New York, v. 4, n. 4, 2005.

HSIEH, C.; PEDRAM, M. Architectural Energy Optimization by Bus Splitting. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 21, n. 4, Apr. 2002.

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, 8th, 2003, Kitakyushu. **Proceedings…** New York: ACM, 2003. p. 233-239.

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, 2004, Paris. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2004, p. 234-239.

HU, J.; MARCULESCU, R. Energy- and Performance-Aware Mapping for Regular NoC Architectures. **IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 24, n. 4, Apr. 2005.

IBM CORPORATION. **The CoreConnect™ Bus Architecture**. Disponível em: <http://www.3ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture>. Acesso em Dez. 2004.

ITRS. **2002 Technology Roadmap for Semiconductors**. Disponível em <http://public.itrs.net>. Acesso em: Dez. 2002.

ITRS. **2005 Technology Roadmap for Semiconductors**. Disponível em <http://public.itrs.net>. Acesso em: Out. 2005.

IYER, A.; MARCULESCU, D. Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, ISCA, 29th, 2002, Anchorage. **Proceedings…** New York: ACM, 2002. p. 158-168.

KEUTZER, K. F. et al. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. **IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems**, New York, v. 19, n. 12, 2000.

KREUTZ, M.; MARCON, C. A. M.; CALAZANS, N.; CARRO, L.; SUSIN, A. Energy and Latency Evaluation of NoC Topologies. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005, Kobe, **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2005a. p. 5866-5869.

KREUTZ, M.; MARCON, C. A. M.; CARRO, L.; WAGNER, F.; SUSIN, A. Design Space Exploration Comparing Homogeneous and Heterogeneous Network-on-Chip Architectures. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2005, 18th, Florianópolis. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2005.

KUMAR, S. et al. A Network on Chip Architecture and Design Methodology. In: IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI, 2002. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2002. p. 105-112.

KUMAR, S. On Packet Switched Network for Chip Communication. In: JANTSCH, A.; TENHUNEN, H. (Ed.), **Networks on Chip**. Boston: Kluwer Academic Publishers, 2003. p. 85-106.

LAHIRI, K.; RAGHUNATHAN, A.; DEY, S. Design Space Exploration for Optimizing On-Chip Communication Architectures. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, v. 23, n. 6, Jun. 2004.

LAVAGNO, L.; SANGIOVANNI-VINCENTELLI, A.; HSIEH, H. Embedded System Codesign: Synthesis and Verification. In: NATO ADVANCED STUDY INSTITUTE IN HARDWARE/SOFTWARE CO-DESIGN, 1995, LOCAL. **Proceedings…**, Kluwer Academic Publishers, 1995.

LEE, E.; SANGIOVANNI-VINCENTELLI, A. **The Tagged Signal Model**: A Preliminary Version of a Denotational Framework for Comparing Models of Computation. Berkeley, University of California: 1996. (Memorandum UCB/ERL M96/33).

LEI, T.; KUMAR, S. A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture. In: EUROMICRO SYMPOSIUM ON DIGITAL SYSTEM DESIGN, 2003, DSD, Belek-Antalya. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2003. p. 180–187.

LIANG, J.; SWAMINATHAN, S.; TESSIER, R. aSoC: A Scalable, Single-Chip communications Architecture. In: IEEE INTERNATIONAL CONFERENCE ON PARALLEL ARCHITECTURES AND COMPILATION TECHNIQUES, 2000, Philadelphia. **Proceedings…** New York: ACM Press, 2000. p. 37-46.

MARCON, C. A. M. **Modelos, Ferramentas e Métodos para o Projeto Integrado de Hardware e Software**. 2000. Trabalho Individual I (Doutorado em Ciência da Computação) Faculdade de Informática, PUCRS, Porto Alegre.

MARCON, C. A. M. **Modelagem de Sistemas de Telecomunicação para o Projeto Integrado de Hardware e Software**. 2001. Trabalho Individual II (Doutorado em Ciência da Computação) Faculdade de Informática, PUCRS, Porto Alegre.

MARCON, C. A. M. **Análise do Particionamento e Escalonamento de Recursos para o Projeto Integrado de Sistemas de Hardware/Software**. 2002a. Exame de Qualificação (Doutorado em Ciência da Computação) Faculdade de Informática, PUCRS, Porto Alegre.

MARCON, C. A. M.; CALAZANS, N. L. V.; MORAES, F. G. Requirements, Primitives and Models for Systems Specification. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2002, 15th, Porto Alegre. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2002b. p. 323-328.

MARCON, C. A. M.; HESSEL, F.; AMORY, A.; RIES, L.; MORAES, F.; CALAZANS, N. Prototyping of Embedded Digital Systems from SDL Language: A Case Study. In: IEEE International High-Level Design Validation and Test Workshop, HLDVT, 17th, 2002, Cannes. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2002c. p. 133-138.

MARCON, C. A. M.; 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, 10th, 2005, Shanghai. **Proceedings…** New York: ACM, 2005a. p. 33-38.

MARCON, C. A. M.; CALAZANS, N.; MORAES, F.; HESSEL, F.; REIS, I.; SUSIN, A. Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE 2005, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2005b. p. 502-507.

MARCON, C. A. M.; KREUTZ, M.; SUSIN, A.; CALAZANS, N. Models for Embedded Application Mapping onto NoCs: Timing Analysis. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, RSP'05, 16th, 2005, Montreal. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2005c. p. 17-23.

MARCON, C. A. M.; PALMA, J.; CALAZANS, N.; MORAES, F.; SUSIN, A.; REIS, R. Modeling the Traffic Effect for the Application Cores Mapping Problem onto NoCs. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, IFIP VLSI-SoC, 2005, Perth. **Proceedings…** Norwell: IFIP, 2005d.

MARCON, C. A. M.; PALMA, J.; CALAZANS, N.; MORAES, F. Design and Prototyping of an SDH-E1 Mapper Soft-Core. **Revista da Sociedade Brasileira de Telecomunicações**, Rio de Janeiro, v.20, n. 2, Ago. 2005e.

MENTOR GRAPHICS. **Leonardo Spectrum DataSheet**. Disponível em: <http://www.mentor.com/leonardospectrum/>. Acesso em Out. 2005.

MENTOR GRAPHICS. **Modelsim User Manual**. Disponível em: <http://www.mentor.com/leonardospectrum/>. Acesso em Out. 2005.

MIHAL, A.; KEUTZER, K. Mapping Concurrent Applications onto Architectural Platforms. In: JANTSCH, A.; TENHUNEN, H. (Ed.), **Networks on Chip**. Boston: Kluwer Academic Publishers, 2003. p. 39-59.

MORAES, F.; CALAZANS, N.; MARCON, C. A. M.; MESQUITA, D.; PALMA, J.; BLAUTH, V. Design and Prototyping of an E1 Drop_Insert soft core. **IEE Proceedings Communications**, London, v. 150, n. 4, Aug. 2003.

MORAES, F. et al. HERMES: an infrastructure for low area overhead packet-switching networks on chip. **Integration, the VLSI Journal**, London, v. 38, n. 1, Oct. 2004.

MURALI, S.; DE MICHELI, G. Bandwidth-Constrained Mapping of Cores onto NoC Architectures. In: DESIGN AUTOMATION AND TEST IN EUROPE, DATE, 2004a, Paris. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2004, p. 896-901.

MURALI, S.; DE MICHELI, G. SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs. In: DESIGN AUTOMATION CONFERENCE, DAC, 41th, 2004, San Diego. **Proceedings…** New York: ACM, 2004b. p. 914-919.

NI, L. M.; MCKINLEY, P. K. A Survey of Wormhole Routing Techniques in Direct Networks. **IEEE Computer Magazine**, Los Alamitos, v. 26, n. 2, Feb. 1993.

OGRAS, U.; MARCULESCU, R. Energy- and Performance-Driven NoC Communication Architecture Synthesis Using a Decomposition Approach. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE 2005, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2005. p. 352-357.

OST, L. C. **Redes Intrachip Parametrizáveis com Interface Padrão para Síntese em Hardware**. 2004. 116p. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Informática, PUCRS, Porto Alegre. 116 p.

OST, L. C et al. MAIA - A Framework for Networks on Chip Generation and Verification. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 10th, 2005, Shanghai. **Proceedings…** New York: ACM, 2005. p. 49-52.

PALMA, J.; MARCON, C. A. M.; MORAES, F.; CALAZANS, N.; REIS, R.; SUSIN, A. Mapping Embedded Systems onto NoCs - The Traffic Effect on Dynamic Energy Estimation. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2005, 18th, Florianopolis. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2005.

RABAEY, J. M. **Digital Integrated Circuits** - A Design Perspective. New Jersey: Prentice Hall, 1996.

RAGHUNATHAN, V.; SRIVASTAVA, M. B.; GUPTA, R. K. A Survey of Techniques for Energy Efficient On-Chip Communication. In: DESIGN AUTOMATION CONFERENCE, DAC, 40th**,** 2003, Anaheim. **Proceedings…** New York: ACM, 2003. p. 900-905.

RHEE, C.; JEONG, H.; HA, S. Many-to-many Core-Switch Mapping in 2-D Mesh NoC Architectures. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS, ICCD, 22th, 2004, San Jose. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2004. p. 438–443.

RIJPKEMA, E. et al. Trade Offs in the Design of a Router with both Guaranteed and Best-Effort Services for Networks On Chip. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE 2003, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2003. p. 350-355.

SCIUTO, D. Guest Editor's Introduction: Design Tools for Embedded Systems. **IEEE Design & Test of Computers**, New York, v. 17, n. 2, Apr. 2000.

SGROI, M.; LAVAGNO, L.; SANGIOVANNI-VINCENTELLI, A. Formal Models for Embedded System Design. **IEEE Design & Test of Computers**, New York, v. 17, n. 2, Apr. 2000.

SHERWANI, N. A. **Algorithms for VLSI Physical Design Automation**, 2nd ed. Boston: Kluwer Academic Publisher, 1999.

SYSTEMC COMMUNITY. **SystemC 2.1 Language Reference Manual**. Disponível em: <http://www.systemc.org>. Acesso em Out. 2005.

VALDERRAMA, C. A. et. al. **Hardware/Software Co-Design: Projetando Hardware e Software Concorrentemente**. Rio de Janeiro: DCC/IM, 2000. Trabalho apresentado na 13ª Escola de Computação.

WU, D.; AL-HASHIMI, B. M.; ELES, P. Scheduling and Mapping of Conditional Task Graph for the Synthesis of Low Power Embedded Systems. **IEE Proceedings on Computers and Digital Techniques**, London, v. 150, n. 5, Sep, 2003.

YE, T. T.; BENINI, L.; DE MICHELI, G. Analysis of Power Consumption on Switch Fabrics in Network Routers. In: DESIGN AUTOMATION CONFERENCE, DAC, 39th, 2002, New Orleans **Proceedings…** New York: ACM, 2002. p. 524-529.

YE, T. T.; BENINI, L.; DE MICHELI, G. Packetized On-Chip Interconnection Communication Analysis for MPSoC. In: DESIGN AUTOMATION AND TEST ON EUROPE, DATE, 2003, Munich. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2003. p. 344-349.

YE, T. T.; BENINI, L.; DE MICHELI, G. Packetization and Routing Analysis of On-Chip Multiprocessor Networks. **Journal of Systems Architecture**, North-Holland, v. 50, n. 2-3, Feb. 2004.

ZEFERINO, C. A.; SUSIN, A. A. SoCIN: a Parametric and Scalable Network-on-Chip. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2003, 16th, São Paulo. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 2003. p. 169-174.

ZEFERINO, C. A. **Redes-em-chip**: Arquiteturas e Modelos para Avaliação de Área e Desempenho. 2003. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ZOU, Y.; ZHUANG, Z.; CHEN, H. HW-SW Partitioning Based on Genetic Algorithm. In: CONGRESS ON EVOLUTIONARY COMPUTATION, CEC, 2004, Portland**. Proceedings…** Los Alamitos: IEEE Computer Society Press, 2004. p. 628-633.

APÊNDICE A: APLICAÇÕES UTILIZADAS PARA VALIDAÇÃO DE MODELOS DE MAPEAMENTO

Este Apêndice apresenta um subconjunto das aplicações utilizadas para validação dos modelos e métodos apresentados ao longo deste trabalho. Para cada aplicação existe uma breve especificação e uma correspondente descrição no modelo CWM, que serve para facilitar a compreensão e a comparação entre as aplicações aqui descritas.

I. INTEGRAL DE ROMBERG

Romberg é um método para calcular a integral de uma função através de aproximações sucessivas (ACTON, 1990). O método insere sucessivas extrapolações de Richardson sobre sucessivas estimativas da regra trapezoidal, de forma a obter com poucas iterações um valor numérico que se aproxima do valor da integral de uma função em um intervalo.

O método de integração de Romberg, também chamado de Integral de Romberg, pode ser implementado por fluxo de dados com formato de um triângulo retângulo, tal como apresentado na Fig. 1. A primeira coluna ilustra a aplicação da regra trapezoidal, enquanto o restante do triângulo implementa a extrapolação de Richardson.

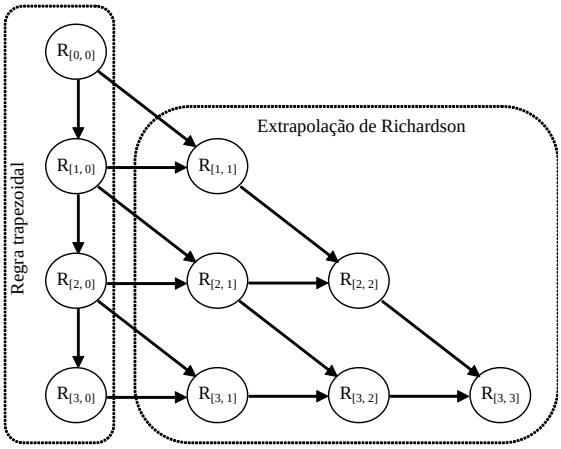


Fig. 1: Fluxo de dados para a integral de Romberg. Os círculos representam elementos de processamento (PE) e as flechas indicam o fluxo de dados. Os PEs estão ilustrados com a notação R[linha, coluna], que informa a linha e a coluna que o PE está situado dentro do triângulo retângulo.

Este método pode ser implementado por um fluxo de dados distribuído, onde cada PE recebe dados de PEs anteriores, processa estes e gera novos dados para PEs

posteriores. Quanto maior o número de linhas, maior será a precisão da estimativa de integração, em contrapartida *cresce o número de elementos de processamento*[49] e a necessidade de comunicação entre estes.

Esta aplicação tem algumas características interessantes para este trabalho. A primeira é sua alta escalabilidade que permite avaliar infra-estruturas de comunicação de diversos tamanhos. A segunda é o perfil da comunicação ser essencialmente um fluxo de dados com direção e sentido bem determinado, sendo esta características comum a várias outras aplicações. Idealmente, para a implementação do método de integração de Romberg seria interessante que a infra-estrutura de comunicação tivesse uma topologia semelhante ao formato descrito pela Fig. 1. Todavia, esta não é uma topologia comum. Assim, este exemplo permite avaliar a eficiência das ferramentas aqui propostas, para a exploração de mapeamentos que atendam os requisitos de projeto, frente a topologias genéricas.

Para modelar a Integral de Romberg com CWM, a Fig. 2 mostra que foram utilizados alguns valores de integração hipotéticos. Considera-se que a comunicação da primeira coluna é maior que as demais. Assim, para comunicação vertical foi atribuído um fluxo de 300 phits e 200 phits para as demais comunicações.

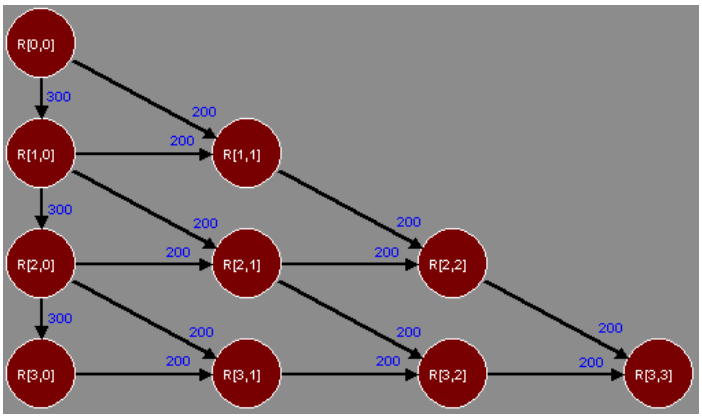


Fig. 2: Modelagem do método de integração de Romberg com CWM. Cada aresta representa um elemento de processamento indicando a sua linha e coluna. Cada vértice é anotado com o volume de comunicação.

II. TRANSFORMADA RÁPIDA DE FOURIER

A Transformada Rápida de Fourier (BRIGHAM, 1988), em inglês *Fast Fourier Transform* (FFT), é um método matemático para expressar uma função como soma de funções sinusoidais multiplicadas por coeficientes. A transformada é bastante utilizada no tratamento de imagens, tal como a remoção de ruídos.

Este método pode ser implementado concorrentemente utilizando threads. A maioria destas, chamadas de processadoras, realizando o processamento da FFT em paralelo e mais uma thread adicional, chamada de sincronizadora, responsável pelo envio de dados e o recolhimento do resultado do processamento realizado pelas demais

[49] Para o método de integração de Romberg, o número de elementos de processamento cresce proporcional ao quadrado do número de linhas. Mas precisamente, seja n o número de linhas, então o número de elementos de processamento é $(n^2 + n)/2$.

threads.

O algoritmo que implementa esta FFT é executado em cinco estágios: (*i*) entrada dos dados para cada thread processadora; (*ii*) permutação de dados entre as threads processadoras; (*iii*) iterações com processamento e sem comunicação; (*iv*) iterações com processamento e comunicação; e (*v*) sincronização, que envolve o salvamento dos dados processados.

Durante o estágio de processamento com comunicação, a cada iteração, as threads processadoras devem trocar informações sobre todos os seus pontos de imagem com threads parceiras. Isso é feito pela leitura e escrita em canais de comunicação cuja profundidade é igual ao tamanho do vetor de pontos a ser trocado (igual ao número de pontos de imagem por thread). Da mesma forma, ao término do estágio de processamento com comunicação, as threads processadoras enviam seus resultados finais para a thread sincronizadora, escrevendo em canais de comunicação reservados para essa finalidade.

Semelhante ao método de integração de Romberg, a versão implementada para a FFT também é escalonável e tem um fluxo de dados bem definido. Para exemplificar uma implementação da aplicação, a Fig. 3 apresenta uma FFT composta por quatro threads processadoras (FFT0, FFT1, FFT2 e FFT3) e uma thread sincronizadora Sync. Nesta ilustração a FFT está modelada por CWM.

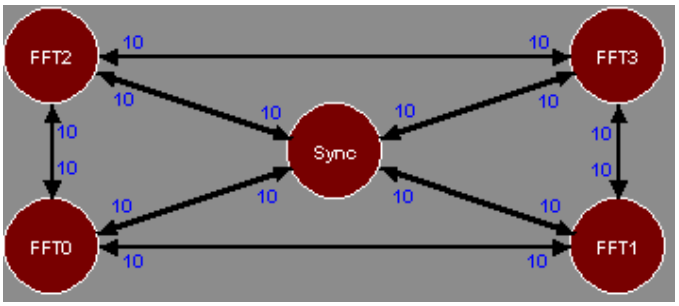


Fig. 3: Modelagem de uma Transformada Rápida de Fourier com CWM. Nesta descrição a FFT é composta por quatro threads processadoras (FFT0, FFT1, FFT2 e FFT3) e uma thread sincronizadora (Sync). Todas as arestas têm o mesmo volume de comunicação hipotético igual a 10 phits.

III. PABX DIGITAL

PABX Digital é uma central telefônica que tem como principal objetivo a comutação de ligações entre diversas tecnologias de comunicação, tais como ramais e troncos analógicos e digitais. A central telefônica descrita nesta Seção foi baseada no projeto comercial do PABX Digital XT130 (HESSEL, 2004) (HESSEL, 2005).

O PABX Digital XT130 é composto por vários processos executando sobre processadores, micro-controladores e DSPs. Na implementação comercial deste equipamento, muitos dos processos aqui descritos executam sobre o mesmo elemento de processamento, e a comunicação entre estes processos ocorre através de memória compartilhada. Todavia, com o objetivo acadêmico de avaliar este mesmo equipamento implementado com maior grau de concorrência, os principais processos foram divididos em conjuntos e associados a um elemento de processamento. Além disto, as necessidades de comunicação entre os processos foi contabilizada. Assim, cada conjunto de processos passa a ser percebido como um elemento de processamento, com

uma necessidade de comunicação bem definida. Estes elementos de processamento, por sua vez, podem ser mapeados sobre uma infra-estrutura de comunicação, que para este trabalho é uma rede intrachip, perfazendo assim um novo PABX com um maior grau de concorrência.

Ao contrário das aplicações descritas nas Seções I e II, o PABX Digital não é uma aplicação facilmente escalonável. O interesse nele está na complexidade dos processos e na grande aleatoriedade da comunicação entre processos. A complexidade dos processos advém das diferentes naturezas tecnológicas e da variedade de protocolos implementados. A aleatoriedade, por sua vez, advém da forte influência dos *estímulos externos*[50] no fluxo de dados.

Para obter estimativas razoáveis da necessidade de comunicação entre os processos escolhidos, os códigos fontes do PABX foram simulados com estímulos que correspondem a 10 minutos de operação do PABX em uma empresa de médio fluxo de operação (o uso concorrente dos recursos telefônicos não ultrapassa a 30%). Os valores obtidos estão apresentados na Fig. 4. Esta Figura, que descreve o PABX através do modelo CWM, destaca três conjuntos de elementos de processamento: (*i*) o primeiro conjunto está relacionado aos elementos de processamento que implementam parcialmente as funcionalidades dos troncos digitais – vértices ligados direta ou indiretamente ao vértice TrDig; (*ii*) o segundo conjunto de está relacionado aos elementos de processamento que implementam as funcionalidades dos ramais digitais – vértices ligados direta ou indiretamente ao vértice RamDig; e (*iii*) e o terceiro conjunto que contém os demais elementos de processamento, sendo estes ligados ao processo principal, que é o vértice Proc. Neste último estão contemplados tanto os ramais quanto os troncos analógicos, as portas de comunicação serial e os demais processos.

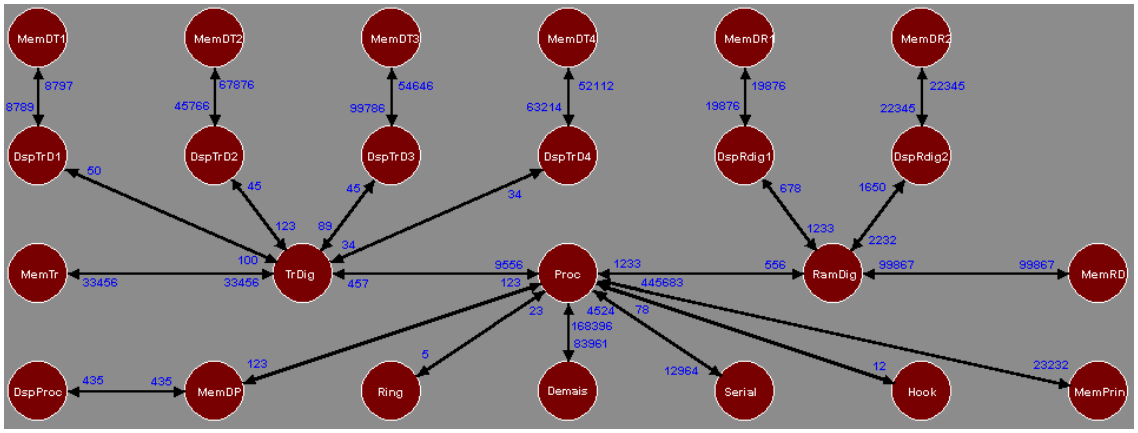


Fig. 4: Modelagem do volume de comunicação de uma central telefônica através do modelo CWM. Aqui não está sendo medido o volume de dados referente à comunicação digital, mas apenas os sinais de dados e controle entre diversos processos.

[50] Estímulos externos a um PABX são ações realizadas por um usuário local, tal como tirar um telefone do gancho e discar o número de um ramal ou pegar uma linha externa, e também ações provenientes de outras centrais telefônicas, que podem, por exemplo, informar através de um protocolo a entrada de uma ligação por um tronco digital. Nenhum destes eventos tem instante certo para acontecer e a alta possibilidade de concorrência dificulta muito a estimativa da necessidade real de comunicação.

IV. MPEG-4

MPEG-4 é um padrão de multimídia definido pelo grupo MPEG (*Moving Picture Experts Group*). Ele foi desenvolvido para prover alta qualidade de áudio e vídeo sobre diversas mídias com diferentes larguras de banda. O padrão implica a implementação de subsistemas de codificação e decodificação de alta eficiência de forma a não prejudicar a qualidade do áudio ou vídeo.

A complexidade natural desta aplicação implica a utilização de núcleos IP com grande poder de processamento e uma infra-estrutura de comunicação capaz de atender uma grande largura de banda. Estes fatores levaram Murali e De Micheli (2004[B]) a escolher esta aplicação, entre outras, para avaliar a ferramenta *SUNMAP*, que foi desenvolvida por eles para auxiliar na geração automática de topologias de rede. A aplicação está ilustrada na Fig. 5.

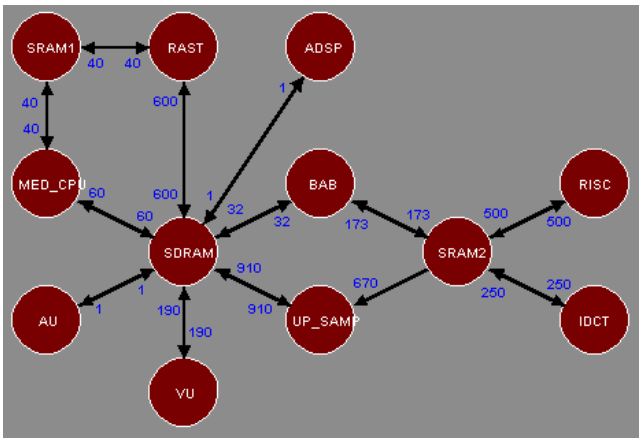


Fig. 5: Modelagem de um MPEG4 com CWM. A aplicação é descrita pela comunicação entre 12 núcleos IP.

A comunicação entre os núcleos, descrita na Fig. 5, está na ordem de MBytes/seg, indicando que para alguns casos, o volume da comunicação pode ultrapassar a máxima largura de banda dos canais, motivo pelo qual os autores Murali e De Micheli (2004) sugerem que a comunicação seja distribuída por diversos caminhos na rede.

V. MULTIMEDIA SYSTEM

MultiMedia System (MMS) é um sistema de multimídia genérico adotado por Hu e Marculescu (2003) e (2005) para testar algoritmos de mapeamento sugeridos por estes autores. O MMS integra sistemas de áudio e vídeo, que incluem codificador e decodificador de vídeo H236, e codificador e decodificador de áudio MP3.

Este sistema foi inicialmente particionado em 40 processos. De acordo com critérios de afinidade e volume de comunicação, os processos foram reagrupados em núcleos IP, tais como DSPs, processadores, memórias e ASICs. No total, o MMS é implementado por 16 núcleos IP. Os núcleos e as comunicações entre estes estão ilustrados na Fig. 6. Para obter a quantidade de comunicação entre os núcleos, os autores (HU; MARCULESCU, 2003) e (HU; MARCULESCU, 2006) utilizaram aplicações de áudio e vídeo reais, e através de simulação obtiveram estimativas da comunicação entre cada núcleo. A arquitetura do MMS e os valores obtidos pelos

autores foram utilizados neste trabalho para que fosse possível comparar estratégias e algoritmos de mapeamento.

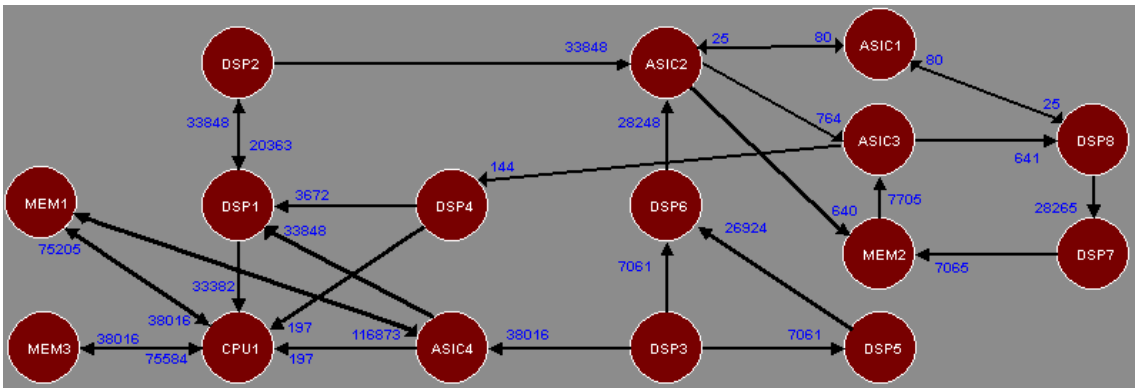


Fig. 6: Modelagem do MultiMedia System (MMS) com CWM. A aplicação é implementada com 16 núcleos IP, sendo estes DSPs, ASICs, Memórias e processadores de uso geral.

APÊNDICE B: GRAU DE CONCORRÊNCIA E DEPENDÊNCIA DE APLICAÇÕES

Este Apêndice apresenta heurísticas para comparar diferentes aplicações, modeladas com CDCM, frente às características dependência versus concorrência e computação versus comunicação. Detalhes sobre o uso destas heurísticas para obtenção de mapeamentos de qualidade podem ser encontrados em (MARCON et al., 2005c).

I. INFLUÊNCIA DA MODELAGEM DO GRAU DE DEPENDÊNCIA E CONCORRÊNCIA DE UMA APLICAÇÃO NA ATIVIDADE DE MAPEAMENTO

Não é tarefa trivial determinar quanto uma aplicação pode ser caracterizada como concorrente ou dependente. Os formatos internos adotados neste trabalho permitem estimar estas características pelas relações ditadas entre vértices e arestas, dado que uma aresta no formato interno CDCG representa uma relação de dependência entre dois vértices a ela conectados, e vértices que não se relacionam por caminhos de dependência são potencialmente concorrentes.

Esta Seção propõe heurísticas para relacionar proporcionalmente o *grau de dependência* de uma aplicação (*gd*) com o seu *grau de concorrência* (*gc*).

Sendo *n* o número de vértices de um grafo, e *vd*(*v*) a lista de vértices que o vértice *v* depende. Então, *gd* é calculado pela soma de todos os *vd* dos *n* vértices, tal como ilustrado na Eq. 1. Ou seja, a soma de todas as dependências de todos os vértices do grafo que representa uma aplicação expressa o grau de dependência desta.

Eq. 1: $$gd = \sum_{i=1}^{n} vd(v_i)$$

O grau de concorrência é obtido pelas combinações de *vértices CDCGs potencialmente simultâneos*[51]. Sendo *cc*(*v*) o *conjunto de todas as combinações de concorrência do vértice v*, então a Eq. 2 descreve *CC*, que é o *conjunto união de todos os cc* de todos os vértices da aplicação.

Eq. 2: $$CC = cc(v_1) \cup cc(v_2) \cup \ldots \cup cc(v_n)$$

[51] Dois *vértices CDCGs* são considerados potencialmente simultâneos quando o intervalo de tempo que está ocorrendo a computação ou a comunicação de um vértice pode coincidir com o intervalo de tempo que está ocorrendo a computação ou a comunicação do outro vértice.

O grau de concorrência *gc*, descrito na Eq. 3, é a cardinalidade de *CC*, que contém todas as possíveis combinações de concorrências dos vértices da aplicação.

Eq. 3: $gc = |CC|$

O percentual do grau de dependência frente ao grau de concorrência da aplicação é calculado através da Eq. 4.

Eq. 4: $gd\_gc = \frac{gd}{gc + gd} \times 100\%$

A heurística adotada está exemplificada na Fig. 7, que descreve o grafo de uma aplicação sintética. Neste, a aplicação é composta por seis vértices:

$$V = \{V_1, V_2, V_3, V_4, V_5, V_6\},$$

que representam que representam a computação e a comunicação de núcleos da aplicação, além dos vértices especiais INÍCIO e FIM, que determinam o início e fim da aplicação. Na parte direita de cada vértice *v* existe uma lista de vértices dos quais *v* depende. Por exemplo, V_5 depende dos vértices INÍCIO, V_1 e V_3, e o vértice FIM depende da *execução completa*[52] de todos os demais vértices descritos no grafo. A solução de todas as dependências de FIM indica o encerramento da execução da aplicação.

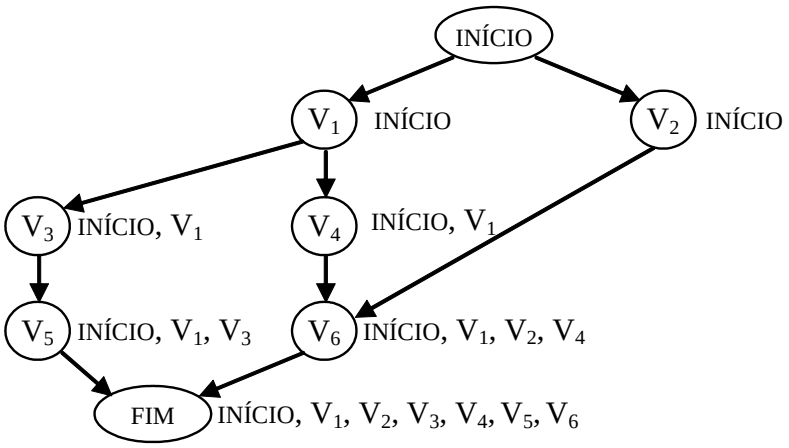


Fig. 7: Grafo de uma aplicação sintética contendo seis vértices que representam a computação e a comunicação da aplicação e mais os vértices INÍCIO e FIM, que delimitam o início e fim da aplicação. Ao lado de cada vértice está descrito o conjunto dos vértices que este depende.

As combinações das concorrências dos vértices da aplicação da Fig. 7 são obtidas percorrendo todos os caminhos que, a partir do vértice especial INÍCIO, levam a este vértice. A Tab. 1 ilustra todas estas combinações para todos os seis vértices da aplicação.

Tab. 1: As seis primeiras linhas representam todas as combinações concorrentes de todos os vértices da aplicação sintética descrita na Fig. 7. Por exemplo, o vértice V_1 pode ser concorrente apenas com V_2, os demais vértices não podem ser concorrentes, pois dependem de V_1. A última linha representa o conjunto

[52] A *execução completa de um vértice* da aplicação indica que já transcorreu todo intervalo de tempo necessário para a computação e comunicação descrita neste vértice.

união de todas as combinações concorrentes.

Combinações de todas as concorrências de um vértice	
$cc(V_1)$	{ (V_1,V_2) }
$cc(V_2)$	{ (V_2,V_3,V_4), (V_2,V_4,V_5), (V_1,V_2), (V_2,V_3), (V_2,V_4), (V_2,V_5) }
$cc(V_3)$	{ (V_2,V_3), (V_3,V_4), (V_3,V_6), (V_2,V_3,V_4) }
$cc(V_4)$	{ (V_2, V_4), (V_3,V_4), (V_4,V_5), (V_2,V_3,V_4), (V_2,V_4,V_5), }
$cc(V_5)$	{ (V_5,V_6), (V_4,V_5), (V_2,V_4,V_5) }
$cc(V_6)$	{ (V_3,V_6), (V_5,V_6) }
CC	{ (V_1,V_2), (V_2,V_3), (V_2,V_4), (V_2,V_5), (V_3,V_4), (V_3,V_6), (V_4,V_5), (V_5,V_6), (V_2,V_3,V_4), (V_2,V_4,V_5) }

A Eq. 2 fornece a união de todos os conjuntos compostos pelas combinações de concorrência de todos os vértices (CC = {(V_1,V_2), (V_2,V_3), (V_2,V_4), (V_2,V_5), (V_3,V_4), (V_3,V_6), (V_4,V_5), (V_5,V_6), (V_2,V_3,V_4), (V_2,V_4,V_5)}). Este conjunto está ilustrado na última linha da Tab. 1. Utilizando a Eq. 3 chega-se ao grau de concorrência da aplicação dado por $gc = |CC| = 10$.

Aplicando a Eq. 1 sobre as listas de dependência apresentadas na Fig. 7 obtém-se $gd = 20$. Finalmente, aplicando a Eq. 4, para os valores de gd e gc, obtém-se $gd\_gc = 66.67\%$, que indica que a aplicação é mais dependente que concorrente.

II. INFLUÊNCIA DA MODELAGEM DO GRAU DE COMPUTAÇÃO E COMUNICAÇÃO DE UMA APLICAÇÃO NA ATIVIDADE DE MAPEAMENTO

O CDCM modela o tempo de computação, associando a cada vértice uma constante que indica o tempo transcorrido entre a liberação de todas as dependências de uma mensagem e o envio desta. Por sua vez, a quantidade de bits de cada mensagem associada ao caminho que esta percorre é que determina o tempo de comunicação. Alterando estes elementos, é possível gerar diversas aplicações sintéticas caracterizadas por 0 a 100% de computação ou comunicação.

Para evitar que a relação entre as características da aplicação seja mascarada por valores discrepantes de comunicação ou computação, este trabalho utilizou heurísticas baseadas em distribuição estatística normal, e determina um valor de *desvio padrão* (σ)[53] que define quais amostras devem ser consideradas na avaliação do grau de computação e comunicação.

Seja s_i a i-ésima amostra e μ o valor médio das amostras calculado pela Eq. 5, então o desvio padrão pode ser computado pela Eq. 6. Utiliza-se aqui um desvio padrão composto por amostras que estejam em um intervalo de 30 a 70% do valor médio, para melhor caracterizar a aplicação e evitar discrepâncias.

[53] O objetivo do desvio padrão é evitar que na média das amostras sejam considerados valores discrepantes que descaracterizariam a aplicação. Embora as aplicações sejam sintéticas, foi escolhido empiricamente o intervalo de 30 a 70%, respeitando que este mesmo intervalo seja utilizado na avaliação de aplicações reais. Este intervalo depende da regularidade da computação e/ou comunicação da aplicação. Aplicações com poucas comunicações ou computações díspares podem aceitar uma faixa maior para calcular a média das amostras. O inverso também é verdadeiro.

Eq. 5: $$\mu = \frac{\sum_{i=1}^{n} s_i}{n}$$

Eq. 6: $\sigma_- = \mu \times 0.3$ e $\sigma_+ = \mu \times 1.7$

Seja $f(s_i)$ o valor da amostra s_i, representando a computação ou comunicação de uma aplicação. A obtenção da média e desvio padrão está exemplificada na Fig. 8.

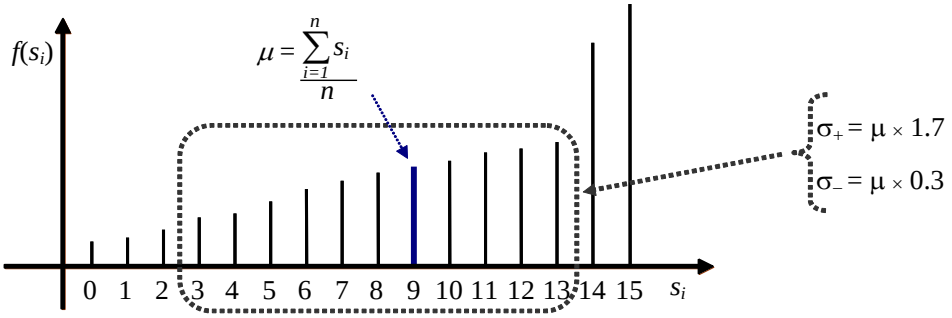


Fig. 8: Exemplo de uma aplicação com 16 amostras, tendo como valor médio a décima amostra e desconsiderando as três amostras menos significativas e as duas mais significativas, por estas serem discrepantes da média.

Dado os valores da média e desvio padrão, determina-se ss_i como sendo uma amostra pertencente ao intervalo considerado. Ou seja, $ss_i \in \{\mu - \sigma_- < s_i < \mu - \sigma_+\}$ $\forall\ 1 \leq i \leq n$. Assim, a Eq. 7 permite calcular a *média das amostras consideradas* (ω).

Eq. 7: $$\omega = \frac{\sum_{i=1}^{n} ss_i}{/\ ss_i\ /}$$

O *grau de computação da aplicação* (*gct*) pode ser obtido utilizando a Eq. 7, considerando que cada amostra de tempo de computação é obtida em cada vértice do CDCG. Sendo que para a seleção de amostras não discrepantes deve ser aplicando o desvio padrão descrito na Eq. 6.

O *grau de comunicação da aplicação* (*tgc*) é obtido de forma mais complexa. Como não é possível estimar com precisão o tempo de comunicação de cada mensagem antes de realizar o mapeamento de núcleos, pois o mapeamento é que determina os caminhos para o tráfego das mensagens, foi utilizada uma heurística baseada na avaliação probabilística do tráfego e no tamanho da NoC. Os parâmetros probabilísticos foram extraídos de trabalhos que mostram a eficiência de mapeamentos projetados para reduzir o tempo de execução de aplicações. Estes trabalhos sugerem que mapeamentos de qualidade, cujo objetivo seja a redução da latência de comunicação, obtém em média caminhos de comunicação que consomem menos que 20% do *tempo médio de comunicação*. Sendo este último calculado pela média do comprimento de todos os caminhos com o tempo gasto para percorrer cada caminho. A redução para menos que 20% acontece porque os núcleos com alta taxa de comunicação são posicionados em tiles vizinhos, enquanto que os tiles distantes são deixados para os pares de núcleos que se comunicam menos.

Para NoCs do tipo malha com topologia regular e tiles de lados quadrados,

como ilustrado na Fig. 9, a média do comprimento dos caminhos pode ser estimada pelo *número médio de roteadores do caminho* ($\eta_{\text{médio}}$).

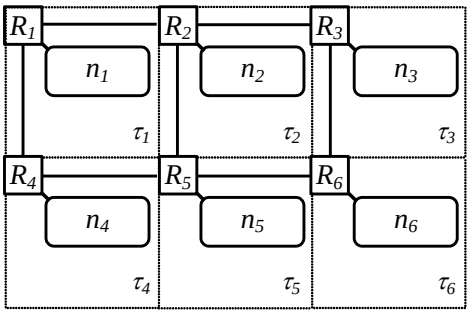


Fig. 9: Ilustração de uma NoC malha 2 × 3.

Na NoC da Fig. 9, cada um dos núcleos mapeados em um dos 2 tiles centrais (τ_2 e τ_5) pode se comunicar com 3 núcleos mapeados em tiles vizinhos através de comunicações que passam por 2 roteadores ($\eta = 2$), e com os 2 núcleos restantes através de comunicações que passam por 3 roteadores ($\eta = 3$). Cada um dos 4 núcleos mapeados nos tiles periféricos (τ_1, τ_3, τ_4, e τ_6) pode se comunicar com 2 núcleos mapeados em tiles vizinhos passando por 2 roteadores ($\eta = 2$), com 2 núcleos mapeados em tiles conectados a um tile vizinho passando por 3 roteadores ($\eta = 3$), e com o núcleo mapeado no tile posicionado na diagonal oposta passando por 4 roteadores ($\eta = 4$).

Contabilizando todos os caminhos para todos os roteadores, tal como ilustrado na Eq. 8, chega-se a $\eta_{\text{médio}}$ igual a 2.67.

$$\text{Eq. 8:}\quad \eta_{\text{médio}} = \frac{2 \times ((3\times2) + (2\times3)) + 4 \times ((2\times2) + (2\times3) + (1\times4))}{2 \times 5 + 4 \times 5} = 2.67$$

Aplicando-se a redução de 20% em $\eta_{\text{médio}}$ pode-se estimar que as comunicações passam em média por menos de 2.14 roteadores.

Seja n_{Fij} o número de phits transmitido do núcleo i para o núcleo j, então a estimativa de latência de comunicação medida entre estes núcleos ($d_{ij\_\text{médio}}$), é dada pela Eq. 9.

$$\text{Eq. 9:}\quad d_{ij\_\text{médio}} = (\eta_{\text{médio}} \times (t_R + t_L) + n_{Fij} \times t_L) \times \lambda$$

Considerando que o tempo de roteamento de um phit e o tempo de transmissão de um phit por uma conexão é igual a um ciclo, e considerando o caminho médio determinado por $\eta_{\text{médio}} = 2.14$, com a Eq. 9 chega-se a Eq. 10.

$$\text{Eq. 10:}\quad d_{ij\_\text{médio}} = (2.14 \times (1 + 1) + n_{Fij} \times 1) \times \lambda$$

$$= (4.28 + n_{Fij}) \times \lambda$$

O uso da heurística $\eta_{\text{médio}}$ na Eq. 10 faz com que a diferença de latência entre pacotes seja apenas o número de phits do pacote, não importando o mapeamento dos núcleos. Assim, *tgc* é obtido aplicando a Eq. 7, considerando que as amostras são obtidas pelo $d_{ij\_\text{médio}}$ que tem como variável a cada quantidade de phits descrita em cada vértice.

Similarmente ao cálculo do grau de dependência versus o grau de concorrência apresentado na Eq. 4, aqui é utilizada a Eq. 11 para estimar a percentagem entre o grau de comunicação e o grau de computação da aplicação.

Eq. 11: $$tgc\_gct = \frac{tgc}{gct + tgc} \times 100\%$$

Para exemplificar os cálculos de *tgc* e *gct*, a Fig. 10 apresenta uma aplicação sintética, modelada por CDCM, considerando o mapeamento de seis núcleos em uma NoC 2 × 3. Neste exemplo, a relação *tgc_gct* vale 54%, mostrando que a aplicação tem o grau de comunicação levemente dominante sobre o grau de computação.

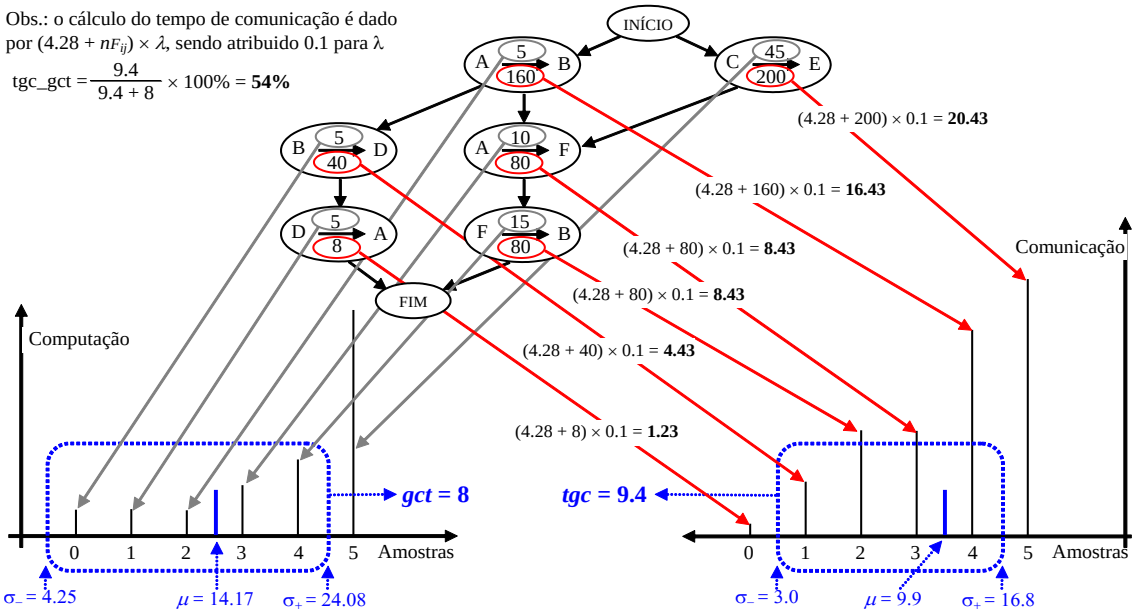


Fig. 10: Cálculo do grau de computação (*gct*) e comunicação (*tgc*) de uma aplicação sintética com seis núcleos mapeados em uma NoC malha 2 × 3. Para ambos os casos são mostrados as análises estatísticas de como foram consideradas as amostras.