ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

FREDERICO DE FARIA

UMA METODOLOGIA ANALÍTICO-DETERMINÍSTICA PARA A
AVALIAÇÃO DE DESEMPENHO NO TEMPO DE PROCESSADORES
DE REDE IMPLEMENTADOS COMO SISTEMAS-SOBRE-SILÍCIO

São Paulo
2007

FREDERICO DE FARIA

UMA METODOLOGIA ANALÍTICO-DETERMINÍSTICA PARA A
AVALIAÇÃO DE DESEMPENHO NO TEMPO DE PROCESSADORES
DE REDE IMPLEMENTADOS COMO SISTEMAS-SOBRE-SILÍCIO

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia Elétrica

São Paulo
2007

FREDERICO DE FARIA

UMA METODOLOGIA ANALÍTICO-DETERMINÍSTICA PARA A
AVALIAÇÃO DE DESEMPENHO NO TEMPO DE PROCESSADORES
DE REDE IMPLEMENTADOS COMO SISTEMAS-SOBRE-SILÍCIO

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia Elétrica

Área de Concentração:
Microeletrônica

Orientador: Prof. Livre-Docente
Wang Jiang Chau

São Paulo
2007

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com anuência de seu orientador.

São Paulo, 20 de julho de 2007.

Frederico de Faria

Prof. Dr. Wang Jiang Chau

FICHA CATALOGRÁFICA

Faria, Frederico de

**Uma metodologia analítico-determinística para a avaliação de desempenho no tempo de processadores de rede implementados como sistemas-sobre-silício / F. de Faria. -- São Paulo, 2007. - Edição Revisada
156 p.**

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

**1.Circuitos digitais (Modelagem) 2.Circuitos integrados
3.Sistema de computação 4.Sistema de comunicação 5.Avaliação de desempenho de computadores 6.Modelagem matemática
I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.**

DEDICATÓRIA

Dedico esta obra a meus pais, pela
sólida formação que me foi dada.

AGRADECIMENTOS

Acima de tudo ao Prof. Wang Jiang Chau por ser o exemplo de conduta, atitude, moral e comprometimento. Também por ter sido a fonte de estímulo intelectual e científico.

Ao Prof. Marius Strum por ter, ao longo dos anos, junto com o Prof. Wang, construído um grupo de trabalho cientificamente e academicamente sério e competente, ao mesmo tempo com um ambiente de trabalho descontraído.

Aos colegas do Grupo de Sistemas Eletrônicos Integrados e Software Aplicado por todas as discussões científicas que me foram proporcionadas e em especial aos colegas Edgar Romero e Sebastian Eslava por toda ajuda e acolhimento.

RESUMO

O grande aumento da capacidade de integração de transistores em um único circuito integrado tem exigido grande e constante evolução na metodologia de projeto e práticas de implementação de sistemas eletrônicos embarcados. Tal capacidade de integração resultou no surgimento de sistemas sobre silício (SoCs). O projeto de tais sistemas, mais complexos que seus predecessores, alteram significativamente os fluxos tradicionais de concepção de sistemas, fazendo surgir estratégias tais quais reuso, projetos orientados a plataformas, assim como modelagens e simulações em diferentes níveis de abstração. Um dos diferentes níveis de abstração estudados é o analítico, onde os sistemas são modelados através de representações abstratas. A adoção de modelos analíticos apresenta vantagens, como alta velocidade de execução (permitindo um grande número de análises de modelos diferentes) e facilidade de alteração. No entanto, por se tratarem de modelagens distantes, em termos de abstração, de implementações reais, podem oferecer prognósticos não exatos. Faz-se então necessária a investigação de metodologias que tenham como propósito o aperfeiçoamento de tais modelos em termos de acurácia e fidelidade. O presente trabalho apresenta uma metodologia de modelagem analítica para avaliação de desempenho de sistemas-sobre-silício orientada a aplicação de processadores de redes de pacotes. A metodologia de *Network Calculus*, a ser implementada nos estágios iniciais de projeto de sistemas-sobre-silício baseados em plataforma, contribui para reduzir o espaço de avaliação de projeto. Trata do equacionamento analítico de representações abstratas das cargas de entrada e também da capacidade de processamento de recursos, visando obter prognósticos mais pessimistas e mais otimistas de parâmetros como latência, requisição de buffer e utilização do sistema, descrito de modo abstrato através de grafos.

Palavras-chave: sistemas sobre silício. avaliação de desempenho. modelagem analítica. processadores de rede. barramento. política de arbitragem.

ABSTRACT

The great increase in terms of integration capacity of transistors on integrated circuits has demanded great and constant evolution in the design methodology and practical implementation of embedded electronic systems. Such capacity of integration resulted in the sprouting of systems-on-chips (SoCs). The design of such systems, more complex than their predecessors, significantly changes the traditional flow in the conception of systems, bringing up strategies such like reuse, platform based design, as well as modeling and simulation in different abstraction levels. One of the different abstraction levels under study is the analytical one, where the systems are shaped through abstract representations. The adoption of analytical models presents advantages, such as high speed of execution (allowing a great number of analyses of different models) and easiness for alteration. However, due to their distant representation models, in terms of abstraction, from real implementations, they cannot offer accurate prognostics on several design metrics. Therefore, it is necessary the investigation on methodologies aiming to the enhancement of such models in terms of accuracy and fidelity. The present work shows a methodology of analytical modeling for evaluation of system-on-chip performance guided to the application of network processors of packages. The methodology of Network Calculus, to be implemented in the initial steps of of system-on-chip's design cycle, contributes to reduce the design space exploration. It deals with the building of analytical equations for abstract representations of workloads and also the processing capacity of resources, aiming at to get most pessimistic and most optimistic prognostics of parameters such like latency, buffer requirements and the system utilization, described in abstract way through graphs.

Keywords: systems on chip. performance evaluation. analytical modeling. network processors. bus. arbitration policies.

LISTA DE ABREVIATURAS E SIGLAS

ADL - *Architecture Description Language*

AHB - *AMBA High-performance Bus*

AMBA - *Advanced Microcontroller Bus Architecture*

ANL - Analítico

APB - *Advanced Peripheral Bus*

ASB - *Advanced System Bus*

ASIC - *Application Specific Integrated Circuits*

B –*Buffer*

BCA - *Bus Cycle Accurate*

CRC –*Cyclic Redundancy Check*

CSMA-CD - *Carrier Sense Multiple Access with Collision Detection*

DiffServ - Ambiente de serviços diferenciados

DMA - *Direct Memory Access*

DSE - *Design Space Exploration*

FIFO - *buffer First-In-First-Out*

FPGA –*Field Programmable Gate Array*

GRT –Grafo de Recursos-Tarefas

GRTE –Grafo de Recursos-Tarefas Expandido

GT –Grafo de Tarefas

HDL - *Hardware Description Language*

IFG - *Interframe Gap*

IHL –*Internet Header Length* (Comprimento do cabeçalho do pacote Internet)

L –Latência

L-B-U - Latência, *Buffer* e Utilização

MAC - *Ethernet Media Access Control*

MII - *Media Independent Interface*

MMU - *Memory Management Unit*

Netcal - *Network Calculus*

NPU - *Network Processing Units*

OSI –*Open Systems International*

Perl - *Practical extraction and resporting language*

PHY - camada física

PROM –*Programmable Read Only Memory*

RAM –*Random Access Memory*

RISC - *Reduced Instruction Set Computing*

RTL - *Register Transfer Level*

RTP –*Real-time Transport Protocol*

RX - Recepção

RXD –*Receive Descriptors*

SAN - *Storage Area Networks*

SDRAM –*Synchronous Dynamic RAM*

SFD –*Start Frame Delimiter* (delimitador de início de quadro)

SoC - *Sytem-on-Chip*

SPARC –*Scalable Processor Architecture*

SRAM –*Static RAM*

SREC - *S-RECORDS*

TCP – *Transmission Control Protocol*

TLM - *Transaction Level Modeling*

TSH - *Time Sequenced Header*

TSPEC – *Traffic Specification*

TTL – *Time-to-Live* (Tempo de Vida)

TX – *Transmissão*

TXD - *Transmit Descriptors*

U – *Utilização*

UDP – *User Datagram Protocol*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very-High-Speed Integrated Circuit*

WLF - *Wave Log Format*

Sumário

Capítulo 1 - Introdução.....	5
1.1 Projeto Top-Down de SOCs e Análise de Desempenho	6
1.2 Modelos Analíticos para Processadores de Rede	9
1.3 Motivações e Justificativas	12
1.3.1 Motivações.....	12
1.3.2 Justificativas.....	13
1.4 Objetivos.....	13
1.5 Organização da dissertação.....	15
Capítulo 2 - Trabalhos correlatos.....	16
Capítulo 3 - Fundamentos teóricos.....	22
3.1 Introdução.....	22
3.2 Fundamentação teórica.....	22
3.3 Terminologia.....	23
3.4 Formalizações.....	27
3.5 Caracterização de curvas de chegada e curvas de serviço.....	29
3.5.1 Curvas de chegada.....	29
3.5.2 Curvas de serviço.....	33
3.5.3 Curvas remanescentes.....	35
3.6 Buffer e Latência.....	36
3.7 Grafo de recursos-tarefas (GRT).....	38
3.7.1 Grafo de tarefas (GT).....	38
3.7.2 Formalização do grafo recurso-tarefa (GRT).....	40
Capítulo 4 - Extensões ao arcabouço analítico de NetCal.....	44
4.1 Introdução.....	44
4.2 NetCal sob a ótica de curvas inferiores e superiores	44
4.3 Relações para α e β	48
4.4 Aproximação de curva por múltiplas partes lineares.....	49
4.4.1 Motivação.....	49
4.4.2 Validação algébrica.....	50
4.5 Critério de estabilidade.....	52
4.6 Limiares mínimos de Latência, requisição de Buffer e Utilização de recurso – o trio (L-B-U)mínimo.....	54

4.6.1 Limiares mínimos de Latência e Buffer.....	55
4.6.2 Limiares mínimos de Utilização.....	60
4.7 Grafo de recursos-tarefas com extensões.....	60
4.8 Grafo de recursos-tarefas expandido (GRTE).....	61
Capítulo 5 - Modelagem de um processador de rede	64
5.1 Introdução.....	64
5.2 Aspectos de processadores de rede.....	65
5.2.1 Módulos de processamento de datagramas.....	65
5.2.2 Operações de processamento de pacotes.....	67
5.2.3 Tipos de arquiteturas de encaminhamento.....	68
5.2.4 Processadores programáveis ou microengenhos dedicados?	70
5.3 Microarquitetura de base.....	71
5.3.1 Processador Leon.....	71
5.3.2 Sistema de barramentos AMBA AHB e APB (especificação 2.0)	73
5.3.3 Memórias PROM/SRAM/SDRAM.....	78
5.3.4 Núcleo periférico Ethernet	79
5.3.5 Aplicativo de driver e encaminhamento de pacotes IPv4 (firmware).....	79
5.4 Modelo RTL do soft-core	80
5.4.1 A distribuição Leon.....	81
5.4.2 A implementação.....	83
5.4.2.1 Hardware.....	83
5.4.2.2 Software.....	85
5.5 Aplicativo para construção de quadros Ethernet.....	85
Aplicativo para construção de quadros Ethernet.....	85
5.6 Modelo analítico.....	89
5.6.1 Latência e vazão associados aos elementos do microssistema.....	89
5.6.2 Modelagem analítica: problemas, metodologia e passos.....	92
5.6.2.1 Passo 01 – Construção do grafo de tarefas (GT).....	92
5.6.2.2 Passo 02 – Construção do grafo de recursos-tarefas expandido (GRTE).....	93
5.6.2.3 Passo 03 – Levantamento das curvas de chegada e de serviço.....	94
5.6.2.4 Passo 04 - Obtenção de (L-B-U)mínimo e (L-B-U)máximo.....	95
5.7 Modelo analítico aplicado à implementação do microssistema.....	96
5.7.1 Modelagem da carga de entrada.....	96

5.7.2 Modelagem dos recursos no GRT – Aspectos de simultaneidade, preemptividade e prioridade.....	96
5.7.3 Modelagem dos recursos no GRTE.....	99
5.7.3.1 Recepção de quadros.....	100
5.7.3.2 Escrita na memória.....	101
5.7.3.3 Leitura da memória.....	103
5.7.3.4 Processamento de TX (Transmissão).....	103
5.7.3.5 Processamento de interrupções (IRQHANDLER).....	104
5.7.3.6 Transferência em barramento.....	105
5.7.3.7 Transmissão de quadros.....	106
5.7.4 Levantamento quantitativo de curvas de serviço.....	106
5.7.4.1 Vértice (RX)@Eth no GRT ou (RX)@Eth1 no GRTE.....	106
5.7.4.2 Vértice (eth_rx,w)@AHB ou (eth_rx,w)@AHB1	107
5.7.4.3 Vértice (eth_rx,w)@Mem ou (eth_rx,w)@Mem1.....	107
5.7.4.4 Vértice (IRQ)@Proc ou (IRQ)@Proc1.....	107
5.7.4.5 Vértice (Proc_rtx,r)@AHB ou (Proc_rtx,r)@AHB2.....	108
5.7.4.6 Vértice (Proc_rtx,r)@Mem ou (Proc_rtx,r)@Mem2.....	108
5.7.4.7 Vértice (RTX)@Proc ou (RTX)@Proc2.....	108
5.7.4.8 Sequência de tarefas com vértice (proc_rtx,w)@AHB ou (proc_rtx,w)@AHB3, e vértice (proc_rtx,w)@Mem ou (proc_rtx,w)@Mem3...	109
5.7.4.9 Vértice (eth_tx,r)@AHB ou (eth_tx,r)@AHB4.....	109
5.7.4.10 Vértice (eth_tx,r)@Mem ou (eth_tx,r)@Mem4.....	110
5.7.4.11 Vértice (TX)@Eth ou (eth_tx,r)@Eth2.....	110
Capítulo 6 - Ambiente de simulação e discussão de resultados.....	111
6.1 Metodologia de avaliação de desempenho.....	111
6.1.1 Metodologia Analítica (ANL).....	111
6.1.2 Nível de transferência entre registradores (RTL).....	112
6.1.3 Acurácia e fidelidade na comparação entre ANL e RTL.....	117
6.2 Resultados e análises.....	119
6.2.1 Validação RTL da microarquitetura alvo.....	119
6.2.2 Análise comparativa entre prognósticos de parâmetros de L-B-U.....	128
6.2.3 Análise comparativa entre recursos Eth, AHB, Mem e Proc.....	131
6.2.4 Análise comparativa entre parâmetros L-B-U.....	136
6.2.5 Qualificação da fidelidade para GRT e GRTE.....	138

6.2.6 Seqüência de vértices no GRTE e melhoria da predição de acurácia. .	139
Capítulo 7 - Conclusões e recomendações de trabalhos futuros.....	140
APÊNDICE A - Definições algébricas básicas.....	143
APÊNDICE B – Detalhamento de operações para levantamento da curva de serviços relativos ao firmware.....	145
B.1 Rotina IRQHANDLER.....	145
B.2 Rotina RTX.....	146
APÊNDICE C – Quadros de histogramas.....	147
Referência Bibliográfica.....	149

Capítulo 1 - Introdução

A capacidade de integração de várias dezenas de milhões de transistores em um único circuito integrado possibilitou o surgimento de Sistemas-sobre-Silício (*Systems-on-chips, SoCs*). Um SoC, por definição (JERRAYA; WOLF, 2005), é um circuito integrado que implementa a maioria ou todas as funções de um sistema eletrônico completo. A execução das funções de um SoC são realizadas por seus componentes, os quais variam com a aplicação determinada ao sistema.

Para conduzir e finalizar projetos de SoCs, engenheiros de projeto têm enfrentado cada vez mais desafios de desenvolvimento em todos os seus estágios devido ao vertiginoso aumento da complexidade de sistemas, surgimento de arquiteturas inovadoras e capacidade de investimento financeiro limitada para financiar novos projetos. Nos anos 2001-2002, em um período de 12 meses, 59 novas arquiteturas de conjunto de instruções surgiram (GOERING, 2002). Um sistema embarcado típico tinha, em 1995, 100 mil linhas de código HDL (*hardware description language*) ao passo que, atualmente, pode ter mais de 1 milhão. Por volta de 98% dos processadores desenvolvidos atualmente têm a eletrônica embarcada como finalidade (GOERING, 2002). Engenheiros devem, então, ter em mãos um conjunto de metodologias e ferramentas que auxiliem o desenvolvimento destes novos sistemas e, ao mesmo tempo, cumprir as diversas exigências e restrições de projeto.

Tecnologicamente, a possibilidade de implementação de sistemas eletrônicos embarcados através de SoCs é fruto da capacidade de fabricação, sendo que hoje a marca de 1 bilhão de transistores já foi superada (BOLSENS, 2007). Um SoC pode conter diversos tipos diferentes de núcleos funcionais, como memórias internas, processadores programáveis de propósito genérico, blocos de lógica digital dedicada, barramentos hierárquicos e heterogêneos de comunicação, blocos analógicos, portos de entrada e saída, conforme a Figura 1.1. Conceitualmente, núcleo é a representação da agremiação de diferentes circuitos eletrônicos detendo um conjunto de funcionalidades lógicas.

Figura 1.1 Diagrama estrutural com os componentes e suas distribuições em um SoC (Sistema-sobre-silício). A linha mais externa delimita o SoC.

Um dos aspectos estruturais que torna o uso do SoC interessante como implementação de sistemas eletrônicos é a vantagem em termos de interconexão, uma vez que os blocos passam a comunicar-se dentro do mesmo circuito integrado. As vantagens passam a ser (IBM, 2001) uma maior taxa de transferência, maior frequência de trabalho do barramento, menor latência de interconexão, menor consumo de energia e maior confiabilidade.

1.1 Projeto Top-Down de SOC's e Análise de Desempenho

A evolução tecnológica em processos de fabricação apontada na seção anterior traz consigo novos desafios também na área de projeto de sistemas eletrônicos.

Segundo (CHANG et al., 1999), os objetivos gerais da eletrônica embarcada atualmente devem ser reduzir o custo financeiro do produto, e ao mesmo tempo

reduzir os tempos e custos intrínsecos de desenvolvimento, além de atingir as metas de funcionalidade e desempenho estabelecidas. Uma das maneiras de contemplar este desafio é definir uma plataforma sobre a qual os diversos produtos irão co-existir. Nesta metodologia, fica nítida a existência de duas etapas distintas: o projeto do núcleo e a integração destes núcleos no contexto de sistema. No projeto do núcleo, usa-se basicamente a mesma metodologia de projeto aplicadas a circuitos de pequeno ou médio portes, com a diferença que interfaces padronizadas devem sempre ser consideradas para a fácil integração com outros núcleos. Na integração, faz parte da metodologia começar o projeto nos níveis arquiteturais mais altos e mais genéricos (portanto com pequeno número de parâmetros sendo considerados) para poder identificar rapidamente os núcleos disponíveis a serem usados.

Ainda segundo (CHANG et al., 1999), outros aspectos importantes atualmente para a implementação de dispositivos embarcados de alta complexidade são:

- 1) os projetos devem ser iniciados nos níveis mais altos de abstração para que todos os graus de liberdade possam ser avaliados;
- 2) os dispositivos embarcados usarão um número menor de componentes de alta complexidade lógica, e um número maior de componentes de menor complexidade, e
- 3) a plataforma de núcleos coexistentes e cooperativos deve ser altamente programável.

Para atender a estas novas metodologias, faz-se então necessária a construção de um conjunto de ferramentas computacionais que auxiliem o projeto de sistemas eletrônicos sobre silício e que permitam aos engenheiros tomarem, de maneira rápida e efetiva, as decisões corretas a respeito de critérios de projeto. Uma das categorias de ferramentas computacionais está no domínio da análise de desempenho no tempo do sistema a ser implementado.

Existem diversos passos de projeto a serem seguidos para trazer um aumento no grau de detalhamento das descrições ou implementações do sistema. Sendo assim, a equipe de projetos pode começar a projetar o seu sistema levando em conta equações analíticas dos diversos núcleos, estruturas de comunicação e a relação entre elas, para prever aspectos de desempenho do sistema (metodologia analítica), e somente evoluir para um maior grau de detalhamento do projeto quando grande parte de - ou todo - um determinado espaço de avaliação de projetos (*DSE - design space exploration*) tiver sido avaliado segundo especificações de análise, e

significativamente reduzido através da análise de resultados fornecidos por uma metodologia analítica, que pode incluir o particionamento entre hardware e *software*, e também uma seleção de plataforma arquitetural e seus componentes (JERRAYA; WOLF, 2005).

O segundo passo pode ser, por exemplo, a simulação do núcleo ou sistema eletrônico (conjunção de núcleos de computação, armazenamento e estruturas de comunicação e *firmware*) em um grau de detalhamento de execução de instruções (funcional) que, apesar de mais lento que a modelagem analítica, oferece maior acurácia. O termo acurácia refere-se, aqui, ao cumprimento rigoroso de determinado conjunto de tarefas cujos resultados são muito próximos entre si (ou idênticos) *quando comparados a uma referência*.

O passo seguinte pode ser, por exemplo, tratar o projeto no grau de detalhamento com precisão aproximada de ciclos (*cycle-accurate*), onde novamente o detalhamento das descrições ou implementações do sistema é maior e a velocidade de simulação é menor. Neste estágio de detalhamento, pode-se dizer que um processador tem o seu esquema de *pipeline* e a hierarquia de memória detalhados, além da inclusão do conceito de mudanças de estado no ciclo de relógio. Os próximos passos são gerar o código em nível de transferência entre registradores (*register transfer level, RTL*) sintetizá-lo gerando um *netlist*, e fazer simulações de alteração de valores de sinais com precisão de ciclos e tempo, respectivamente.

Cada uma destas diversas etapas torna-se mais demorada para o engenheiro, na medida em que o grau de detalhamento aumenta e o poder de processamento computacional necessário para as simulações também aumenta. As etapas podem ser ditas como iterativas e cíclicas diante de um espaço de avaliação de projetos (SAKAMOTO et al., 2003). O conjunto de estudos realizados na etapa analítica deve eliminar o maior volume possível de possibilidades de projeto que não se enquadrarão nas restrições de projeto e ao mesmo tempo deve ser o mais rápido de todos (do ponto de vista de uma única microarquitetura), tendo como um dos resultados a diminuição da quantidade de iterações necessárias entre as diferentes fases de projeto.

O espaço de avaliação de projeto restante deve ser reduzido à medida que as simulações são realizadas para que, idealmente, quando os projetistas forem codificar em RTL e sintetizar os circuitos, reste apenas uma possibilidade que trata

do conjunto ótimo de parâmetros estruturais, funcionais e arquiteturais, e esta possibilidade única cumpra todas as especificações de projeto. A Figura 1.2 serve para ilustrar o compromisso entre grau de detalhamento e custos de avaliação de espaço de projeto, à medida que o projeto evolui. Inicialmente o espaço de avaliação de projetos é maior, ao mesmo tempo que o custo de avaliação de arquiteturas individuais é menor, mas quando os estágios mais avançados do projeto são atingidos, tal figura tende a inverter-se.

As recomendações contidas em (CHANG et al., 1999) são no sentido de “realizar o maior número possível de simulações no nível arquitetural porque é quando é mais fácil e rápido se achar e corrigir erros”. À medida que o nível de detalhamento aumenta, deve-se reduzir o número de possibilidades de análises e simulações já realizadas, isto porque de um nível para outro o tempo de simulação, sem contar o tempo de desenvolvimento de modelos com maiores graus de detalhamento, pode aumentar de duas a três ordens de grandeza (OLUKOTUN; HEINRICH; OFELT, 1998), conforme Tabela 1.1. Nesta tabela são apresentados exemplos de 4 níveis diferentes de abstração de projeto, as respectivas primitivas e a redução no desempenho da simulação quando um dado processador está rodando simulações dele mesmo.

Nível de Projeto	Primitivas	Redução no desempenho da simulação
Analítico	equações	0,01k-0,1k
Arquitetura	blocos funcionais	1k -10k
Transferência entre registradores	RTL	1M-10M
Físico (transistores)	portas lógicas	10M-100M

Tabela 1.1 Níveis de abstração no projeto de sistemas digitais. A redução no desempenho da simulação assume que o processador está simulando seu próprio modelo (OLUKOTUN; HEINRICH; OFELT, 1998).

1.2 Modelos Analíticos para Processadores de Rede

As modernas metodologias de projeto (a partir de 1995) recomendam que, dentre outros aspectos, os projetos de SoCs iniciem-se com um alto grau de abstração, migrando gradativamente para níveis de maior detalhamento e próximos

ao *hardware* (CHANG et al., 1999). Toma-se a modelagem matemática no domínio analítico (modelagem analítica) como passo inicial e evolui para modelagens no domínio computacional. Os motivos principais para estas recomendações são a facilidade de construção de modelos de alto nível, seja em HDLs (VHDL, Verilog, SystemC (SYSTEMC, 2001), etc.), em ADL (*Architecture Description Language*) como o ArchC (RIGO et al., 2004), , sejam em linguagens tradicionais de programação (C, C++, MatLab, etc.), e a rapidez com que estes modelos são construídos e principalmente modificados, quando comparados com os modelos HDL sintetizáveis. Antes de passar de um nível de detalhamento para outro, um grande número de estudos deve ser realizado.

Figura 1.2 Diferentes estágios da avaliação do espaço de projetos e técnicas de avaliação associadas. (CHAKRABORTY et al., 2003a)

Mais especificamente, a modelagem analítica consiste na construção de um modelo matemático - com diferentes graus de comportamento equivalentes ao comportamento do sistema modelado - para posteriormente resolvê-lo. Trata-se portanto de um processo de geração de uma entidade abstrata que representa conceitualmente alguns fenômenos. Tem como desafio inicial o entendimento do

problema natural, sua descrição e também representação. A construção do modelo é acompanhada por um crescente entendimento dos detalhes do problema natural alvo, assim como o conjunto de interrelações entre as partes constituintes, ao mesmo tempo que exige o entendimento de grande parte dos conceitos envolvidos em cada uma das partes que formam o objeto a ser modelado. A modelagem pode ser o início da concepção de entidades complexas e extensas, ainda que um modelo analítico não ofereça a solução para diversas variáveis.

A abordagem analítica para estudo de desempenho de processadores é um assunto amplamente estudado desde os primórdios dos projetos de sistemas computacionais e até o momento, com significativa quantidade de artigos publicados (NOONBURG ; SHEN, 1997), (HARPER; KERBYSON; NUDD, 1999), (KRISHNASWAMY; SCHERSON, 2000) e (YI; LILJA; HAWKINS, 2003), no entanto, existe um aspecto atual que é o equacionamento de elementos surgidos com o advento dos SoCs (ALDWORTH, 1999), (LAHIRI; RAGHUNATHAN; DEY, 2001), (LAHIRI ;RAGHUNATHAN; DEY, 2004), (WOLF, 2002) que integram de modo raramente visto um conjunto de núcleos capazes de executar tarefas de alta complexidade, com restrições de consumo de energia e área raramente experimentados.

Esta metodologia (analítica) pode ser uma solução de menor complexidade quando comparada às baseadas em simulações, e com modelos construídos mais rapidamente, o que indica a viabilidade da modelagem analítica também no estudo de SoCs.

Existem técnicas que rodam modelos analíticos de SoCs em poucos segundos (CHAKRABORTY et al., 2003a) (CHAKRABORTY, 2003b), sendo indicadas para uma ampla, sistemática e efetiva redução no espaço de avaliação de projetos, antes de a equipe de projetos passar para a fase seguinte. Com isto, é possível reduzir o tempo de projeto de SoCs, eliminando alternativas não apropriadas ou inconsistentes com as especificações e critérios de projeto. Como inicialmente o conjunto de avaliação de projeto é o maior de todos, conforme Figura 1.2 (CHAKRABORTY et al., 2003a), ele deve ser reduzido de maneira mais rápida.

Em (CHAKRABORTY et al., 2003a), uma modelagem analítica para um processador de rede implementado em uma plataforma específica foi proposta, integrando peças padrões como o processador programável PowerPC, barramento *CoreConnect*, e bloco Ethernet. Em tal trabalho o processador de rede é analisado

do ponto de vista estatístico para conhecimento do perfil de utilização do barramento, atraso entre instante de chegada e saída de pacotes e utilização total de *buffers* pelo sistema. A modelagem analítica mostrou-se bastante eficiente para esta plataforma, cujos *traces* (rastros) analisados eram advindos de roteadores *core*.

1.3 Motivações e Justificativas

1.3.1 Motivações

Apresentados os problemas das seções anteriores, faz-se necessário o estudo de modelos matemáticos e a construção de ferramenta que seja capaz de analisar e eliminar (de modo significativamente mais rápido que os tempos apresentados por ferramentas de simulação de portas lógicas ou de transferência entre registradores) centenas de possibilidades diferentes de arquiteturas, topologias ou associação entre componentes e tarefas, com vistas às métricas e critérios de desempenho. Pode-se assim reduzir a quantidade de iterações entre etapas de projeto.

Em (SAKAMOTO et al., 2003) é proposta uma metodologia cíclica e iterativa, seguida durante o desenvolvimento do processador Fujitsu SPARC64 V, onde a fidelidade e acurácia dos modelos variam conforme a evolução das ferramentas. Dentro desta metodologia, existem ciclos de realimentação após a validação entre as equipes de projeto, tanto nas fases de avaliação analítica de desempenho quanto na fase de simulação lógica e avaliação.

Ao mesmo tempo que é interessante a redução da quantidade de iterações entre diferentes etapas no ciclo de desenvolvimento de projeto, faz-se necessária a validação do modelo analítico para efetiva quantificação do grau de confiança esperado em suas predições.

Trabalhos de modelagem analítica falham em estabelecer uma comparação e estudo detalhado de acurácia e fidelidade entre predições analíticas destes modelos e resultados de simulação em baixos níveis de abstração, como os sintetizáveis.

1.3.2 Justificativas

Este trabalho é derivado do interesse que o Grupo de Projeto de Sistemas Eletrônicos Integrados e Software Aplicado (GSEIS) do Laboratório de Microeletrônica da Escola Politécnica da USP (LME-EPUSP) (LME, 2007) tem na metodologia de projetos baseada em plataforma. O grupo tem participado, em conjunto com outros seis grupos de pesquisa de diferentes universidades brasileiras, no projeto Brazil IP (BRAZIL, 2004), o qual prevê o desenvolvimento de uma plataforma de projeto de sistemas (plataforma Fênix) (FENIX, 2004).

Dentro das atividades do GSEIS ainda em relação ao Projeto Brazil IP, a plataforma Fênix previa a utilização do processador RISC (*Reduced Instruction Set Computing*) LEON (GAISLER, 2002) como o processador central do sistema. Neste contexto, tem-se disponível para uso o código sintetizável do processador assim como o da memória *cache* nível 1, memória local, barramento Amba (*Advanced Microcontroller Bus Architecture*) AHB (ARM, 1999) , e bloco Ethernet (MAC – *Ethernet Media Access Control*), o que permite a simulação RTL do conjunto.

1.4 Objetivos

Esta dissertação tem o propósito de realizar a concepção, implementação e teste do modelo analítico, com o propósito de contribuir para a solução de desafios de complexidade de projeto que surgiram com os SoCs. Além disso, tem-se por objetivo propor melhorias, seja no modelo, sejam nas metodologias envolvidas, através da construção de ferramentas.

Esta dissertação contempla então os seguintes objetivos específicos a serem alcançados :

1. A modelagem analítica foi realizada tomando como referência o trabalho apresentado em (CHAKRABORTY et al., 2003a). A metodologia seguida é semelhante, mas com uma microarquitetura de referência inicial ligeiramente diferente e com *traces* (marcas de passagem) originados de *benchmarks* diferentes. Dentre as características e compromissos perseguidos, podemos enunciar a obtenção de alta fidelidade nos

resultados comparativos, ainda que com baixa acurácia entre eles (fidelidade versus acurácia). Tal modelagem analítica foi concebida para representar de modo abstrato o sistema de acordo com o sugerido em (CHAKRABORTY et al., 2003a), e de modo similar à adotada nesta dissertação. Trata-se aqui de construir e adaptar o modelo original para as condições da microarquitetura baseada no processador Leon, Amba e bloco Ethernet, que assumem propósitos diversos ao de (CHAKRABORTY et al., 2003a). Some-se o fato que na referência as simulações foram realizadas tomando os modelos funcionais construídos sobre SystemC comportamental, ao passo que aqui as simulações serão realizadas sobre o código VHDL RTL sintetizável.

2. Comparação entre os resultados obtidos com o modelo analítico e os resultados obtidos com o modelo RTL. Esta comparação fornece o grau de fidelidade entre a ferramenta que implementa o modelo analítico e o modelo RTL. Esta abordagem é diferente das abordagens mais tradicionais, as quais comparam os resultados dos modelos analíticos com os das ferramentas de simulação em níveis intermediários de abstração. Por fazer a comparação com o modelo RTL, o grau de fidelidade pode ser determinado com confiança maior (SAKAMOTO et al., 2003) porque fica eliminada a propagação de erros que surgem devido aos erros e ambigüidades inerentes aos modelos intermediários de grau de abstração.
3. Desenvolvimento de outras ferramentas de auxílio à implementação do modelo analítico. Uma ferramenta cuja necessidade foi detectada é a que realiza as análises determinísticas de pior caso sobre os *traces* obtidos, com o intuito de determinar os valores necessários para alimentar as equações analíticas; visto que o volume de dados extraídos de *traces* é muito grande e somente parte da informação extraída destes dados é usada nos cálculos. As teorias matemáticas envolvidas (BOUDEC; THIRAN, 2001) nas fases posteriores simplificam bastante os cálculos quando se têm o conjunto de valores necessários obtidos na fase de processamento de *traces*.
4. Melhorias no modelo analítico apresentado em (CHAKRABORTY et al., 2003a) através de refinamentos do modelo do processador. No trabalho de referência, algumas simplificações foram realizadas para a demonstração da validade do modelo proposto, e como passo adicional nesta dissertação

algumas simplificações foram eliminadas, através de mais detalhes no modelo e mais parâmetros analíticos foram incorporados. Além disso, teve-se como objetivo deste trabalho demonstrar que a eliminação de algumas simplificações trouxeram um ganho de fidelidade ao modelo analítico.

1.5 Organização da dissertação

Esta dissertação apresenta a seguinte organização. No Capítulo 2 são apresentados e discutidos os trabalhos anteriores sobre os quais esta dissertação baseia-se. No Capítulo 3 a fundamentação teórica clássica de *Network Calculus* é apresentada, com a definição de conceitos e apresentação dos teoremas mais relevantes à esta dissertação. No Capítulo 4 são expostas as contribuições teóricas algébricas e analíticas deste trabalho juntamente com as definições e demonstrações relacionadas. A implementação da microarquitetura de processador de rede sob estudo é apresentada no Capítulo 5, assim como as entidades que a compõe. Resultados e discussões são apresentados no Capítulo 6 sob óticas diferentes: acurácia, fidelidade, recursos e parâmetros, estabelecendo uma relação entre cada um destes aspectos e o modelo analítico. O último capítulo apresenta as conclusões desta dissertação e as recomendações de trabalhos futuros.

Capítulo 2 - Trabalhos correlatos

Neste capítulo, são apresentadas as teorias fundamentais que norteiam as metodologias de modelagem analítica utilizadas neste trabalho. Elas são basicamente três: análise determinística de comportamento de eventos em redes de comunicação, álgebra de mínimos e máximos, e cálculo de rede (*Network Calculus*, ou NetCal), o mapeamento entre recursos e tarefas e cálculo de tempo real (*real time calculus*). Pode-se chamar de NetCal o conjunto de regras e equações que, juntas, são utilizadas para a computação de limiares de latência, requisição de *buffers* e utilização de recursos de sistemas que compõem as redes de pacotes. Entendemos que tais parâmetros são de grande importância logo nos estágios de concepção de sistemas, principalmente a latência, já que dentre as diversas funcionalidades que processadores de rede devem apresentar, a entrega de pacotes em tempo real pode ser citada como uma das principais. Portanto, em tal categoria de sistemas, a eficiência em termos de latência é mais importante que a eficiência em termos de área ocupada pelos circuitos ou por sua potência consumida. Por este motivo, grande parte dos trabalhos referenciados nesta dissertação tem o parâmetro específico latência como foco de estudo. O parâmetro utilização é uma função do tempo, e também de grande importância na medida em que contribui para que diversos núcleos componentes do sistema tenham graus balanceados e equivalentes de utilização. Por último, o parâmetro *buffer*, que não é dimensionado em unidades de tempo, está intimamente relacionado à latência e por consequência grande parte dos trabalhos referenciados também abordam o estudo deste parâmetro.

A evolução da teoria que fundamenta esta dissertação tem por origem os trabalhos de Rene Cruz (CRUZ, 1991a, 1991b) no estudo de comportamento de eventos em redes de comunicação. Neste contexto, tal trabalho apresentou, de modo não probabilístico, as primeiras abordagens para cálculo de limiares de latência e *buffer* em redes de comunicação de dados operando em modo de comutação de pacotes. Somam-se a estes os trabalhos apresentados em (CUNNINGHAME-GREENE, 1979) sobre álgebra de mínimos e máximos, e posteriormente, em (BOUDEC, 1998), diretamente relacionado a *Network Calculus* e aplicado a redes de pacotes. A referência (BOUDEC, 1998) forma o fundamento

teórico de NetCal e também o início do fundamento aplicado de *real-time calculus*, álgebra de mínimos e máximos (ou álgebra maximal ou álgebra intervalar) e NetCal, de forma agregada ao estudo do comportamento de eventos em redes de comunicação por pacotes.

Em um segundo momento, as aplicações de NetCal evoluíram para contribuírem na resolução de problemas de exploração e avaliação do espaço de soluções de projetos em microarquiteturas de circuitos processadores de rede comunicação por pacotes (KUNZLI; THIELE; ZITZLER, 2005) (portanto expandindo a atuação das teorias propostas em (BOUDEC, 1998), de redes de comunicação de dados para microarquiteturas de circuitos processadores). Atualmente alguns artigos mostram que as aplicações de NetCal podem ultrapassar os limites da área de processadores de rede, incluindo o estudo de sistemas multiprocessados orientados a algoritmos de codificação, com cunho mais intenso no processamento e menos intenso em termos de comunicação.

Em (THIELE; CHAKRABORTY; NAEDELE, 2000), é estabelecido o elo entre teoria de sistemas lineares, sistemas de eventos discretos e redes de comunicação e os estende a sistemas de tempo real. São mostrados os princípios da modelagem teórica de sistemas e suas dependências, recursos computacionais e algoritmos de escalonamento através da aplicação de escalonamento em tempo real. Um equacionamento inicial é estabelecido, com o uso de álgebra de mínimos e máximos e, a partir de equações lineares, de funções que representam a quantidade total de esforço computacional requisitado por um determinado conjunto de eventos e a quantidade total de esforço computacional efetivamente realizado por este conjunto de eventos; da mesma forma são engendradas as funções que representam a máxima capacidade computacional que pode ser disponibilizada e a máxima capacidade computacional efetivamente utilizada. As funções acima são então vinculadas no domínio do tempo (CRUZ, 1991a, 1991b) e adaptadas ao conceito de curvas de chegada e curvas de serviço no domínio de intervalos de tempo; o primeiro referindo-se ao perfil dos dados entrantes no sistema e o segundo à sua capacidade de fornecer computação. É também apresentado o equacionamento das funções remanescentes de chegada e de serviço em forma de proposições (as demonstrações encontram-se em (NAEDELE; THIELE; EISENRING, 1999), ainda sem os conceitos de curvas superiores e inferiores.

Nos trabalhos (THIELE; CHAKRABORTY; GRIES, 2001) e (THIELE et al.,

2002a), são apresentados refinamentos em relação a (THIELE; CHAKRABORTY; NAEDELE, 2000), já orientados à aplicação para o caso de sistemas eletrônicos de tempo real dedicado ao processamento de pacotes em redes de comunicação de dados por pacotes. A modelagem do conjunto de tarefas e recursos de processadores de rede, assim como a aplicação do conceito de curvas de chegada e de serviços são apresentadas. Tais curvas são obtidas a partir de funções no tempo e os conceitos associados são expandidos para assimilar também os conceitos de curvas superiores e inferiores a cada uma delas. A partir de então, tais conceitos são estudados para o caso de uma microarquitetura dedicada a processamento de pacotes. Uma formalização da definição de microarquitetura é exposta e é realizado um estudo sobre o espaço de avaliação de projeto incluindo os conceitos mencionados acima em conjunto com alocação, associação, mapeamento, escalonamento e particionamento. Em (THIELE et al., 2002a), é apresentada uma maneira mais enxuta computacionalmente para o cálculo das equações através da aproximação das curvas por meio de segmentos. O cálculo das equações deixa de ser realizado ponto-a-ponto e passa-se a utilizar apenas as inclinações e *offsets* (2 pontos) para tais segmentos. Apresenta formalmente a proposição de como obter latência e *buffer*, ainda que não realize as demonstrações. Formaliza também o conceito de fator de escala e passa a incorporar a influência de diferentes tarefas através do uso do conceito de pesos.

Thiele et al. propõe em (THIELE et al., 2002b) um aumento no número de componentes do sistema sob estudo (processador de rede), mas ao mesmo tempo implementa uma redução no número dos detalhes de componentes da estrutura de comunicação, um barramento compartilhado. Além de não fornecer um número elevado de detalhes dos parâmetros extraídos para componentes da barramento, não aborda a modelagem de componentes de memória (a microarquitetura do sistema sob estudo nesta dissertação, assim como as dos trabalhos acima, são explicadas no Capítulo 5). O trabalho formaliza os limiares máximos de latência e de *buffer*, muito embora o cálculo de limiares mínimos em tais parâmetros não esteja no arcabouço proposto pelos autores.

Em (CHAKRABORTY; GRIES; THIELE, 2002), a modelagem apresentada em (THIELE et al., 2002a) e (THIELE et al., 2002b) é aplicada ao estudo de escalonamento de pacotes em tempo real. Desta forma apresenta meios de como extrair parâmetros a partir de *traces* (rastros) no tempo para a montagem de curvas

(de chegada e de serviços, tanto superiores quanto inferiores), mas agora com o propósito de analisar o sistema sob a ótica do escalonamento e das garantias de níveis de serviços a pacotes, com e sem prioridades de entrega, previamente estipulados.

Mais adiante, em (CHAKRABORTY; KUNZLI; THIELE, 2003), o conceito de utilização de recurso no tempo é adicionado ao estudo de latência e *buffer*, assim como a modelagem de diversos fluxos de eventos (pacotes) com diferentes características e necessidades, de modo simultâneo. Os diversos fluxos entrantes têm alto grau de independência de dados entre si, ainda que utilizem um subconjunto comum de tarefas e compartilhem os recursos computacionais necessários. Passa-se assim a poder realizar-se o estudo sobre três parâmetros, a saber, latência percebida, requisição de *buffer* e utilização de recursos no tempo.

Com um foco em estudo de caso, Gries et al.(GRIES et al., 2003) montam um ambiente de trabalho de simulação composto de compilador C e ferramenta que simula o processador de rede Intel IXP1200, ferramenta esta fornecida pelo fabricante do processador e com nível de acurácia da ordem de ciclos fim-a-fim de operações, mas não de instruções. A ferramenta detalha os tempos transcorridos para operações e não pode ser considerada *cycle-accurate* para sinais. Apresenta como resultados a utilização no tempo da memória e dos circuitos dedicados (*microengines*).

Tal aplicativo é dividido em 16 *threads*, sendo 8 para os circuitos de recepção (RX) e 8 para os circuitos de transmissão (TX). O clock do sistema é de 200MHz, para um processador baseado no StrongARM (RISC), comunicando-se através de um barramento de largura 64 bits a 80MHz, com memórias SRAM e DRAM *off-chip*, 2 circuitos IXF440 (com 8 MACs - *media access controllers* - cada) *FastEthernet*, com endereços de destino criados aleatoriamente e tamanhos de pacote variando de 40B a 256B, onde cada uma das portas recebe pacotes. Os parâmetros são gerados para a latência entre tarefas e o uso de memória, tanto para escrita quanto para leitura. Os resultados da simulação do IXP1200 apresentam vazão para diferentes tamanhos de pacotes, latência e utilização de recursos, mas não menciona estudo de *buffer*.

A precisão que o trabalho apresenta está em ciclos de relógio para finalização de tarefas definidas sob o aspecto funcional e não no nível de registradores. Apresenta *buffer* de 16 entradas de 64B, mas não modela aspectos específicos do

microprocessador StrongARM.

O método analítico de (GRIES et al., 2003) mostrou que bons níveis de acurácia podem ser obtidos para o modelo quando a comparação é restrita à latência fim-a-fim do sistema e à utilização dedicada de microcircuitos, mesmo que simplificações na modelagem do barramento tenham sido aplicadas e parâmetros individuais dos componentes como latência, *buffer* e utilização não tenham sido extraídos. O estudo do *buffer* é desconsiderado e somente os limites máximos são apresentados para a utilização e latência de recursos. Os níveis de acurácia são levantados pela calibração do fator do peso (CHAKRABORTY; KUNZLI; THIELE, 2003).

Em um trabalho mais extensivo (CHAKRABORTY et al., 2003), a estrutura analítica e as simulações comportamentais modeladas na camada de transação (TLM, *transaction level modeling*) - representando um nível intermediário de abstração de um processador de rede – são correlacionadas e os limites máximos dos resultados obtidos são ilustrados. Embora, neste trabalho, uma aproximação algébrica na modelagem do sistema seja introduzida para minimizar erros no cálculo de limites máximos, a discussão sobre limites mínimos está ausente. O estudo ficou primariamente em operações dos componentes da barramento, e desconsiderou outros componentes como processadores, memória e circuitos dedicados, na extração de parâmetros analíticos para a latência, as exigências do *buffer* e a utilização do recurso.

Uma evolução na aplicação do *framework* de Network Calculus é verificada no trabalho (KUNZLI et al., 2006) na medida em que não trata de processamento de pacotes de redes de comunicação mas sim de algoritmos de codificação de áudio no padrão GSM, baseado em cálculo de matrizes em sistemas embarcados multiprocessados. Neste trabalho é realizado um estudo para análise de esforço computacional tanto nos recursos de processamento quanto nos recursos de estrutura de comunicação. Um modelo híbrido é formalizado, com a presença de dados vindos de simulação assim como dados de resultados analíticos, de modo iterativo. Esta mescla entre a formalização analítica e a simulação tem por objetivos reduzir os tempos de análise e as alternativas viáveis na análise de desempenho de sistemas. É feita então uma comparação entre resultados analíticos e outros obtidos através de simulação de modelo construído em System-C comportamental com precisão de ciclos.

Mais recentemente, em (WANDELER; MAXIAGUINE; THIELE, 2006) é apresentada uma metodologia analítica utilizando o *framework* de Network Calculus para a análise de dimensionamento, assim como de posicionamento, de *greedy shapers* (limitadores de tráfego) em uma microarquitetura; em particular, realiza o estudo de alteração de latência e *buffers* sob a influência dos limitadores em sistemas embarcados distribuídos ou multiprocessados. No trabalho é mostrado como se obtém as curvas referentes aos *greedy shapers*, além da demonstração do teorema associado. Através de um caso de aplicação, a obtenção de curvas superiores e inferiores do *greedy-shaper* é ilustrada, além do seu posicionamento e das alterações em termos de latência fim-a-fim e utilização de *buffer* no sistema. É mostrado que, para o caso de um sistema multiprocessado de prioridade fixa, é mais vantajoso limitar a capacidade do recurso de maior prioridade, tendo como consequência a diminuição do *buffer* total requerido assim como a diminuição da latência fim-a-fim.

Capítulo 3 - Fundamentos teóricos

3.1 Introdução

Neste capítulo são apresentadas definições, conceitos e teoremas retirados principalmente das referências (NAEDELE; THIELE; EISENRING, 1999a) e (BOUDEC; THIRAN, 2001), com dois propósitos principais:

- expor ao leitor os conceitos fundamentais e mais importantes para o entendimento da modelagem analítica;
- explicar e interpretar tais conceitos de forma a facilitar a sua compreensão.

Estes aspectos teóricos, introduzidos a partir de uma série de trabalhos anteriores, faz-se muito importante, visto que no Capítulo 4 são apresentadas novas definições, conceitos e teoremas, derivados do presente trabalho. Neste capítulo serão apresentadas inicialmente a fundamentação do *Network Calculus* e a terminologia adotada. A seguir, as funções que caracterizam a capacidade de processamento de um recurso, a partir de uma tarefa, e a quantidade de processamento que lhe é requisitada, são descritas, assim como a relação entre tais funções. Os conceitos de curvas de chegada e de serviço são então introduzidos e ilustrados para, finalmente, apresentar a derivação das medidas de latência e *buffer*.

3.2 Fundamentação teórica

A fundamentação de *Network Calculus* (NetCal) encontra-se na teoria matemática chamada de Max-plus Algebra, ou álgebra de mínimos e máximos (CUNNINGHAME-GREENE, 1979) (BACCELLI et al., 1992). Através de NetCal é possível extrair algumas propriedades fundamentais de Redes de Serviços Integrada (BRADEN; CLARK; SHENKER, 1994) (IETF, 1998), como comportamento de latências entre tarefas e dimensões de *buffers*, além do escalonamento temporal de pacotes. Esta teoria é composta por um conjunto de resultados matemáticos que dão suporte à compreensão de sistemas sintéticos, tais como *softwares* de

execução simultânea, circuitos eletrônicos digitais e, naturalmente, redes de comunicação de dados por pacotes. Para uma discussão geral desta área, recomenda-se a leitura do artigo (GUNAWARDENA, 1999) com conceitos gerais e os capítulos 3, 4 e 6 do livro (BACCELLI et al., 1992), assim como os trabalhos pioneiros em (CHANG, 2000).

NetCal é uma teoria determinística (todos os resultados apresentados neste capítulo são determinísticos) de pior caso de sistemas de fila (como os encontrados em redes de comunicação de dados por pacotes). Pode também ser vista como uma teoria de sistemas que se aplica às redes de computador. A diferença principal em relação a outras teorias de sistemas é que, aqui, considera-se uma outra álgebra, onde as operações são modificadas como se segue: a adição transforma-se em computação do mínimo, a multiplicação transforma-se em adição e a integral transforma-se em convolução.

A convolução na teoria tradicional de sistemas é comutativa e associativa, e estas propriedades permitem facilmente a extensão da análise de circuitos de pequena a grande escalas. Por exemplo, a resposta do impulso do circuito RC é a convolução das respostas do impulso de cada uma das entrada elementares, como mostra a expressão abaixo para duas entradas h_1 e h_2 .

$$h(t) = (h_1 \otimes h_2)(t) = \int_0^t h_1(t-s)h_2(s)ds.$$

A seguir são apresentados os conceitos básicos necessários para o entendimento da teoria apresentada neste capítulo, ao passo que conceitos e operadores algébricos mais específicos podem ser encontrados no APÊNDICE A.

3.3 Terminologia

Esta seção define a terminologia usada ao longo deste trabalho e apresenta exemplos que serão estendidos posteriormente.

Tarefa: Uma tarefa representa, de modo abstrato, uma atividade atômica a ser executada. Exemplo de tarefa é uma operação de escrita de um mestre do barramento em um circuito de memória ou a checagem da validade do campo CRC (*cyclic redundancy check*) do cabeçalho de um pacote IPv4.

Recurso: Um recurso é a representação analítica abstrata de um determinado

objeto físico que tem a capacidade de execução de tarefas. Um recurso pode representar um elemento de armazenamento como um registrador, um elemento de processamento como um periférico, ou uma estrutura de comunicação como um barramento. A integração de diversos recursos que executam tarefas, forma, do ponto de vista analítico, um sistema.

Fluxo: Um fluxo é uma seqüência de eventos (por exemplo, pacotes em uma rede de comunicação de dados), tratados de modo similar por um conjunto de recursos. Todos os eventos pertencentes ao mesmo fluxo são processados de modo idêntico (os mesmos recursos executam o mesmo conjunto de tarefas).

No que se refere à álgebra de mínimos e máximos, as principais definições (BACCELLI et al., 1992) são apresentadas a seguir, ilustradas pela Figura 3.1.

Conjunto ordenado: um conjunto designado com uma relação de ordem.

Elemento superior (de um conjunto ordenado): um elemento que é maior que qualquer outro elemento do conjunto (também chamado de elemento universal);

Elemento inferior (de um conjunto ordenado): um elemento que é menor que qualquer outro elemento do conjunto (também chamado de “zero”, é o elemento neutro da adição). Ambos os conceitos são coincidentes neste trabalho.

Elemento máximo (de um subconjunto): um elemento do subconjunto que é maior que qualquer outro elemento do subconjunto; caso ele exista, ele é único; ele coincide com o elemento superior se o subconjunto é igual ao conjunto todo;

Elemento mínimo (de um subconjunto): definição similar.

Elemento maximal (de um subconjunto): um elemento do subconjunto que não é menor que qualquer outro elemento do subconjunto.

A diferença entre um elemento máximo e um elemento maximal é explicitada na Figura 3.1; se um subconjunto tem um elemento máximo ele é o único elemento maximal.

Elemento minimal (de um subconjunto): definição similar.

Elemento Majorante (de um subconjunto): um elemento não necessariamente pertencente a um subconjunto que é maior que qualquer outro elemento do subconjunto; se o majorante pertencer ao subconjunto, ele é o elemento máximo.

Elemento Minorante (de um subconjunto): definição similar.

Limiar superior (de um subconjunto): o menor majorante, isto é, o mínimo elemento do subconjunto de majorantes. Representado pelo operador *sup* (apresentado no APÊNDICE A).

Figura 3.1: Ilustração da terminologia dos elementos para o conjunto ordenado e para os subconjuntos A e B.

Limiar inferior (de um subconjunto): definição similar. Representado pelo operador *inf* (apresentado no APÊNDICE A).

As definições a seguir apresentam os conceitos fundamentais de funções analíticas que são tratadas ao longo desta dissertação. A partir destas funções outros conceitos são abstraídos para fortalecer a dinâmica do estudo de NetCal.

Função monotônica crescente. Uma função f é monotônica crescente se e somente se $f(s) \leq f(t)$ para todo $s \leq t$. Denotaremos por G o conjunto de seqüências ou funções monotônicas crescentes e não-negativas e por F o conjunto de seqüências ou funções monotônicas crescentes tais que $f(t) = 0$ para $t < 0$. O parâmetro t pode ser contínuo ou discreto: no último caso, $f = \{f(t), t \in \mathbb{Z}\}$ é chamado de seqüência e não de função. O intervalo das funções ou das seqüências de F e de G é $\mathbb{R}^+ = [0, +\infty]$.

Taxa de execução e . A taxa de execução de trabalho $e(t)$, é uma função não-negativa que representa a medida da taxa de progressão de tarefas de um recurso; pode representar, por exemplo, o número de instruções por segundo que certa tarefa (pedaço de microcódigo) exige de um recurso (*pipeline*).

Taxa de requisição r . A taxa de requisição $r(t) \geq 0$ representa um trem de impulsos de dados que requer uma certa quantidade de computação para a conclusão das respectivas tarefas. Um fluxo de pacotes ingressante em um recurso para a conclusão de determinada tarefa irá requerer uma certa quantidade de

computação deste recurso, que é representada analiticamente pela taxa de requisição.

Função requisição acumulada R. A função de requisição acumulada

$$R(t) = \int_0^t r(\tau) d\tau, R(0) = 0, R(t) \geq 0, t \in \mathbb{R}^+ \text{ é uma função monotônica crescente que}$$

representa a quantidade total de computação que *foi solicitada* (de um recurso por uma tarefa) até o instante de tempo t .

Função capacidade acumulada C. A função capacidade acumulada

$$C(t) = \int_0^t e(\tau) d\tau, C(0) = 0, C(t) \geq 0, t \in \mathbb{R}^+ \text{ é uma função monotônica crescente que}$$

representa a quantidade total de computação que o recurso *pode oferecer* até o instante de tempo t .

Função computação acumulada R'. A função de computação acumulada é uma função $R'(0) = 0, R'(t) \geq 0, t \in \mathbb{R}^+$ monotônica crescente que representa a *quantidade total de computação apresentada* (de um recurso por uma tarefa) até o instante t como resultado de uma dada função de requisição acumulada R .

Função capacidade ociosa acumulada C'. A função capacidade ociosa acumulada $C'(0) = 0, C'(t) \geq 0, t \in \mathbb{R}^+$ é uma função monotônica crescente que representa a capacidade computacional que *não foi utilizada* até o instante t , por consequência de o recurso não ser utilizado a plena capacidade.

Figura 3.2: Ilustração do modelo de um recurso computacional processando requisições de tarefas. Os modelos de função fluída (a) e sua representação abstrata (b) são ilustrados.

Na Figura 3.2.a é ilustrado o modelo de um recurso processando requisições de tarefas. O fluxo ingressante (que demanda capacidade do recurso para a

execução de determinada tarefa) é representado pela função requisição acumulada R , e é tratado por um recurso com capacidade representada pela função capacidade acumulada C . Após o tratamento do fluxo (através do processamento de tarefas pelo recurso) representado pela função computação acumulada R' , resta ao recurso uma capacidade de processamento representada pela função capacidade ociosa acumulada C' .

Nas Figura 3.3.a e Figura 3.3.b são ilustradas as funções $r(t)$ e $e(t)$, enquanto na Figura 3.3.c são ilustrados os processos de composição de $R(t)$ e $C(t)$, em um exemplo de um fluxo de eventos ingressante que acompanha determinada taxa de execução.

3.4 Formalizações

Nesta seção são apresentados os corolários que relacionam as funções $R(t)$, $R'(t)$, $C(t)$ e $C'(t)$. As demonstrações de todos os teoremas e corolários deste capítulo estão presentes nas referências (NAEDELE; THIELE; EISENRING, 1999a) e (BOUDEC; THIRAN, 2001), e serão omitidas por uma questão de limitação de espaço.

O comportamento em execução de tarefas, com dependências entre capacidade computacional ociosa e a carga de trabalho corrente (representante de um determinado fluxo ingressante de pacotes) é calculada através das seguintes equações.

Corolário 3.1 (Relação entre C , R e R') *Para qualquer recurso com função requisitada acumulada $R(t)$ e função capacidade acumulada $C(t)$, é dado que:*

$$R'(t) = \min_{0 \leq u \leq t} \{R(u) + (C(t) - C(u))\}. \quad (3.1)$$

A quantidade de computação realmente executada, $R'(t)$, deve naturalmente ser limitada superiormente por ambas as funções capacidade acumulada $C(t)$ e função requisição acumulada $R(t)$. Em particular, $R'(t)$ é claramente igual à quantidade máxima de computação que poderia ter sido executada até o instante t , $C(t)$, menos a máxima capacidade ociosa até t . Capacidade está ociosa durante intervalos os quais $R(t)$ é menor que $C(t)$. na Figura 3.3.c é ilustrada a construção de $R'(t)$.

Figura 3.3: Exemplo de correlação entre: (a) taxa de requisição r ; (b) taxa de execução e . A partir da integração das taxas “ r ” e “ e ” obtém-se as funções requisição acumulada R e capacidade acumulada C . A função R' é obtida a partir da equação 3.1.

A capacidade ociosa pode então ser calculada através do Corolário 3.2 abaixo.

Corolário 3.2 (Relação entre C , C' e R') Para cada recurso com função requisição acumulada $R(t)$ e função capacidade acumulada $C(t)$, $C'(t)$ é dado por:

$$C'(t) = C(t) - R'(t). \quad (3.2)$$

Ou seja, a capacidade ociosa acumulada C' é a diferença entre a capacidade acumulada C e a quantidade de computação realmente executada R' , como seria de se esperar.

A próxima seção descreve como caracterizar formalmente um conjunto de tarefas dentro de um sistema de modo a solucionar os problemas em questão, da

quantificação de requisição e computação.

3.5 Caracterização de curvas de chegada e curvas de serviço

3.5.1 Curvas de chegada

Mok e Chen (MOK; CHEN, 1997) apresentam o conceito de *general real-time task* que é caracterizado pela mínima distância entre os inícios das tarefas e uma sequência onde o i -ésimo elemento representa o tempo acumulado de computação no pior caso, para quaisquer requisições consecutivas i .

Definição 3.1 (Curva de chegada α (BOUDEC; THIRAN, 2001)) *Dada uma função monotônica crescente α definida para todo $t \geq 0$, pode-se dizer que a função requisição acumulada R é delimitada por α se e somente se para todo $s \leq t$:*

$$R(t) - R(s) \leq \alpha(t - s), \forall s, t \text{ onde } 0 \leq s \leq t, \quad (3.3)$$

Pode-se dizer que R tem α como uma representação abstrata, como ilustrado na Figura 3.2.b.

O significado de uma curva de chegada α para uma sequência de eventos é que, para um dado intervalo de tempo de comprimento Δ , somente uma quantidade total de computação $\alpha(\Delta)$ pode ser solicitada para execução da série de eventos, ainda que através de solicitações freqüentes com pequenas demandas computacionais, ou de solicitações esparsas com alta demanda computacional ou por qualquer outra combinação aplicável.

Para todas as situações práticas, somente curvas de chegada sub-aditivas fazem sentido (para uma explicação de funções sub-aditivas, $\alpha(s + t) \leq \alpha(s) + \alpha(t)$, veja o APÊNDICE A.

Definição 3.2 (Seqüência gulosa/greedy-task) *Uma seqüência de eventos onde $R(t) = \alpha(t)$ é chamada de seqüência gulosa.*

Uma seqüência gulosa é representada pela curva de chegada α que é idêntica à sua função requisição acumulada R . Caso isto ocorra, um evento requer a máxima

capacidade de computação possível. Esta definição será usada adiante quando forem demonstrados os limiares de curvas.

Se um conjunto de eventos é delimitado por uma curva de chegada α que não é sub-aditiva então ela também é delimitada pela curva de chegada sub-aditiva $\bar{\alpha}$ (envólucro sub-aditivo de α). Dada uma função requisição acumulada R no domínio de t , então a curva de chegada no domínio de Δ (intervalo de tempo) pode ser construída da maneira descrita a seguir.

Corolário 3.3 (Construção da curva de chegada mínima) *Para um conjunto de eventos caracterizado por uma função de requisição acumulada $R(t)$, a curva de chegada delimitada minimamente, α , pode ser calculada como (ou seja, a chegada de eventos é delimitada por):*

$$\begin{aligned}\alpha(\Delta) &= \max_{u \geq 0} \{ R(\Delta + u) - R(u) \} \\ \alpha(\Delta) &= (R \oslash R)\end{aligned} \quad \text{(conforme APÊNDICE A)} \quad (3.4)$$

Exemplo 3.1. Queremos agora construir uma curva α a partir de uma função requisição acumulada $R(t)$, que é mostrada na Figura 3.4.a, previamente construída, por exemplo, a partir de rastros coletados. Conforme o Corolário 3.3, $\alpha(\Delta) = (R \oslash R)$, ou de modo expandido $\alpha(\Delta) = \max_{t \geq 0} \{ R(\Delta + t) - R(t) \}$.

Tomando inicialmente $\Delta = 0$, $\alpha(0) = \max_{t \geq 0} \{ R(0 + t) - R(t) \}$, que ocorre quando $t = T$. Note que $\Delta = 0$ representa o tamanho da maior rajada de $R(t)$.

Tomando $\Delta = 2$, $\alpha(2) = \max_{t \geq 0} \{ R(2 + t) - R(t) \}$. O instante selecionado t deve variar de 0 ao limite conhecido da função $R(t)$, imaginando-se uma janela deslizante de comprimento $\Delta = 2$.

$$\begin{aligned}\alpha(2) &= \max_{t \geq 0} \{ \{ R(2+0) - R(0) \}, \{ R(2+1) - R(1) \}, \{ R(2+2) - R(2) \}, \{ R(2+3) - R(3) \}, \\ &\quad \{ R(2+4) - R(4) \}, \dots, \{ R(2+10) - R(10) \} \} \\ \alpha(2) &= \max_{t \geq 0} \{ \{ 2-0 \}, \{ 2-1 \}, \{ 4-2 \}, \{ 4-2 \}, \dots, \{ 4-4 \}, \{ 10-5 \} \} = 5\end{aligned}$$

Portanto para o limite conhecido de $R(t)$, $\alpha(2) = 5$.

Fazendo-se a mesma seqüência de cálculos para t e Δ variando de 0 ao limite conhecido de $R(t)$, obtém-se todos os pontos da curva α , como representado na Figura 3.4.b. A partir da Definição 3.1 pode-se perceber que α na Figura 3.4.c corresponde à mínima curva de chegada que acomoda todas as possíveis funções requisição acumuladas que seguem a especificação deste exemplo. ■

A relação entre a função de requisição acumulada R e α mostra-se algebricamente mais robusta e α pode ser obtida a partir de dados da função R (por deconvolução). Por exemplo, se $\alpha(t) = rt$, então a delimitação para o conjunto de eventos significa que, para qualquer intervalo de tempo τ , o número de bits para um determinado fluxo é delimitado por $r \cdot \tau$. Pode-se dizer que, neste caso, o fluxo é limitado pela taxa de pico. Isto ocorre quando é conhecido o fato de o fluxo chegar em um enlace o qual a taxa de bits é limitada em r b/s. Um fluxo onde o único delimitador é o limite da taxa de pico é chamado de "fluxo determinístico a taxa de bits".

Tendo $\alpha(t) = b$, onde b é uma constante, a curva de chegada indica que o máximo número de bits que pode ser enviado a qualquer instante em tal fluxo é de no máximo b .

Figura 3.4: A partir da função requisição acumulada R , em (a), a curva α pode ser obtida, como em (b). O resultado é indicado em (c). Note que o comprimento da maior rajada b é representado em $\alpha(0_+)$.

De maneira mais abrangente, pode-se usar curvas de chegada como $\gamma_{r,b}$, definida como $\gamma_{r,b} = rt + b$, para $t > 0$, $\gamma_{r,b}(0)=0$. Exemplos de curvas de chegada como $\gamma_{r,b}$ estão na Figura 3.5. Isto permite à fonte enviar no máximo b bits simultaneamente, mas não mais que r bits/s ao longo do tempo. Parâmetros b e r são chamados de tolerância a rajadas (e expresso em unidades de dados) e taxa de transferência (em unidade de dados por unidade de tempo), respectivamente.

Em termos de aplicação para Rede de Serviços Integrados (BRADEN; CLARK; SHENKER, 1994), pode-se tomar a seguinte função como curva de chegada:

$$\alpha(t) = \min(M + pt, rt + b) \quad (3.5)$$

onde M é interpretado como o maior tamanho de pacote, p é a taxa pico de transferência, b é a tolerância a rajada (dimensão da maior rajada) e r é a taxa de transferência sustentada. Estes conceitos seguem o chamado *traffic specification* (TSPEC) (SHENKER; WROCLAWSKI, 1997) e estão ilustrados na Figura 3.5.

A computação das curvas de chegada através dos métodos apresentados acima ainda são muito exigentes computacionalmente; isto porque envolvem a obtenção de curvas a partir do cálculo de cada ponto originário nas funções capacidade acumulada C e requisição acumulada R , sendo necessárias abordagens algébricas para sua otimização. Para tanto, é apresentado o conceito de função “ideal”, através do qual poderá obter-se envólucros de curvas mais próximos às suas funções iniciais.

Definição 3.3 (Função ideal) *Uma função α é chamada de “ideal” quando qualquer das propriedades a seguir é satisfeita (o APÊNDICE A contém detalhes sobre os operadores a seguir):*

- α é sub-aditiva e $\alpha(0)=0$;
- $\alpha = \alpha \otimes \alpha$
- $\alpha \oslash \alpha = \alpha$
- $\alpha = \bar{\alpha}$ (envólucro sub-aditivo de α).

Lema 3.1: Um fluxo R é delimitado por uma curva de chegada α se e somente se

$$R \leq R \otimes \alpha \quad (3.6)$$

O Lema 3.1 mostra que a curva α que aproxima a função R é sempre maior ou igual a R , pelas propriedades de convolução, conforme (BOUDEC; THIRAN, 2001). Este lema será muito útil nas demonstrações do próximo capítulo e é a partir dele

que torna-se possível calcular um conjunto de curvas α que servem como envólucros de R .

Figura 3.5: Curva de chegada α representativa de um fluxo restrito pela especificação T_{spec} e de acordo com a equação (3.5).

As curvas de chegada devem ser escolhidas de modo a garantir que sejam sub-aditivas, e além disso $\alpha(0) = 0$. Neste caso, conforme (BOUDEC; THIRAN, 2001), $\bar{\alpha} \leq \alpha$ também é uma curva de chegada válida para representar um dado fluxo $R(t)$. Portanto o envólucro sub-aditivo de α , $\bar{\alpha}$, é sempre menor que α e pode tornar-se um melhor candidato a representar R .

A função $\gamma_{r,b}$ apresentada na discussão anterior é uma função sub-aditiva porque $\gamma_{r,b}(0)=0$ e trata-se de uma função côncava.

3.5.2 Curvas de serviço

Vimos que um dos princípios do estudo de redes de serviços integrados é colocar a curva de chegada como delimitadora de fluxo de pacotes. O conceito mais geral de uma curva de serviços no *Variable Processor Model* (NAEDELE; THIELE; EISENRING, 1999a), que é inspirado pelo uso de curvas de serviço no domínio de rede em (BOUDEC, 1996), caracteriza um recurso através da quantidade mínima de computação que é garantida para ser entregue em um intervalo de tempo de comprimento Δ , começando a partir de um ponto arbitrário no tempo, t .

Definição 3.4 (Curva de serviços β) A curva de serviços de um recurso com função capacidade acumulada $C(t)$ é uma função estritamente crescente, não-negativa. Diz-se que o recurso oferece uma curva de serviços β , se e somente se:

$$C(t) - C(s) \geq \beta(t-s), \forall s, t \text{ onde } 0 \leq s \leq t, \quad (3.7)$$

O valor da curva de serviços $\beta(\Delta)$ para qualquer intervalo de tempo de comprimento Δ pode ser obtido *a posteriori*, a partir de registros da real taxa de execução, $e(t)$, e capacidade acumulada, $C(t)$, como mostrado nas Figura 3.6.a e Figura 3.6.b, e, olhando-se para uma janela deslizante de tamanho Δ . $\beta(\Delta)$ é então o mínimo entre as taxas de todas estas janelas da integral sobre a capacidade total, como é ilustrado na Figura 3.6.c.

Figura 3.6: Exemplo de obtenção da curva beta (c) a partir das funções taxa de execução “e” (a) e capacidade acumulada C (b)

Para um recurso caracterizado por uma curva de serviços β , o limite inferior na quantidade de computação efetivamente cedida pode ser dada pelo corolário a seguir.

Corolário 3.4 (limite inferior em R') Dada uma curva de serviços β de um determinado recurso, se $\beta(0) = 0$ então para todo t existe t_0 , $0 \leq t_0 \leq t$, tal que:

$$R'(t) - R(t_0) \geq \beta(t - t_0), \quad (3.8)$$

que pode ser expresso usando a convolução de *min-plus*, conforme definição no APÊNDICE A:

$$R'(t) \geq (R \otimes \beta)(t) \quad (3.9)$$

Isto significa que, para um dado recurso caracterizado por uma curva de serviços β , a alocação de uma certa quantidade de capacidade computacional pode ser garantida para uma tarefa que requisita uma demanda computacional de acordo com o apontado pela função de requisição acumulada $R(t)$.

3.5.3 Curvas remanescentes

O teorema a seguir mostra formalmente como estão relacionadas a curva de serviços β para todas as tarefas em consideração (incluindo a tarefa τ), a capacidade requisitada pela tarefa de maior prioridade ($\alpha_{prioridade}$) e a curva de serviços remanescente β' garantida a τ .

Como ilustra a Figura 3.2.b, a curva de serviços remanescente β' representa a abstração da capacidade computacional que um recurso apresenta após ter processado uma determinada carga de entrada representada por uma curva de chegada $\alpha_{prioridade}$ e tendo iniciado o processamento desta carga com capacidade de serviço β .

Teorema 3.1 (Curva de serviços remanescente β') *Considere um conjunto de tarefas Γ de n tarefas $\tau_1, \tau_2, \dots, \tau_n$, assumindo-se que $\tau_1, \tau_2, \dots, \tau_{n-1}$ tenham prioridade mais alta que τ_n . Cada uma com uma função de requisição acumulada R_j é delimitada por uma curva de serviços α_j , com $1 \leq j \leq n$. As funções de requisição acumulada R_i , $1 \leq i \leq n-1$, de $\tau_1, \tau_2, \dots, \tau_{n-1}$ são delimitadas por curvas de chegada sub-aditivas α_i e o serviço oferecido a Γ é caracterizado por uma curva de serviços super-aditiva β . Então o serviço oferecido a τ_n é caracterizado por*

$$\beta'(\Delta) = \max_{0 \leq u \leq \Delta} \{ \beta(u) - \alpha_{prioridade}(u) \}, \text{ onde } \alpha_{prioridade} = \sum_{i=1}^{n-1} \alpha_i \quad (3.10)$$

De maneira similar, existe uma relação entre a curva de serviços β para todas as tarefas em consideração (incluindo a tarefa τ), a curva de chegada α e a curva de chegada remanescente α' , como também ilustrado Figura 3.2.b. O teorema a seguir mostra formalmente tal relacionamento.

Teorema 3.2 (Fluxo de saída, ou curva de chegada remanescente) *Assumindo um fluxo delimitado por uma curva de chegada α , que atravessa um dado sistema que oferece uma curva de serviços β , o fluxo de saída é delimitado pela curva de chegada através da operação de deconvolução dada por*

$$\alpha' = \alpha \oslash \beta. \quad (3.11)$$

Deste modo, a curva de chegada remanescente é a curva de chegada vista na saída de um recurso, após a carga representada por α ter sido processada pelo recurso com capacidade de serviço β . Desta maneira relacionamos a curva de chegada remanescente α' com a curva de serviços β . Quando tratar-se de um sistema, a curva de chegada remanescente poderá representar a curva de entrada de outro recurso.

3.6 Buffer e Latência.

A combinação entre as garantias providas pelas curvas de serviço com as delimitações das curvas de chegada formam os limites determinísticos básicos usados nesta dissertação. Abaixo, as definições e interpretações físicas das medidas de *buffer* e latência são apresentadas, a partir das funções requisição acumulada R e computação acumulada R' .

Definição 3.5 (*buffer*) Para um sistema sem perdas, o *buffer* no instante t é $R(t) - R'(t)$.

O *buffer* é a representação de toda a quantidade de dados (dado em *bits*, por exemplo) que permanece dentro de um recurso (ou de um sistema, quando visto como uma integração de diversos recursos) durante o intervalo de tempo em que estes recursos executam tarefas. Se o recurso for um elemento de armazenamento, então ele representará o comprimento da fila; em contraste, em se tratando de um sistema (portanto composto de diversos recursos) mais complexo, então *buffer* é a representação da quantidade de dados (ou do número de *bits*, como queira) “em trânsito”, supondo que possamos observar as entradas e as saídas simultaneamente.

Definição 3.6 (Latência) A latência $L(t)$ é o menor intervalo de tempo tal que todas as tarefas iniciadas no instante t terão sido completadas no instante $t+L(t)$. Pode ser descrita da seguinte forma:

$$L(t) = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t+T) \}. \quad (3.12)$$

O atraso (latência) no instante t é o atraso que seria experimentado por um *bit* que chega no tempo t se todos os *bits* recebidos antes dele já fossem servidos. Se a função requisição acumulada R e a função computação acumulada R' forem

contínuas (o modelo é fluído), então é fácil ver que existe um $d(t)$, tal que $R'(t+d(t))=R(t)$, e que existe um $d(t_0)$, onde t_0 é o instante em que ocorre o menor valor de $d(t)$ que satisfaz a esta equação.

Para calcular o limiar de latência, o conceito de distância horizontal entre duas curvas deve ser apresentado.

Definição 3.7 (Distância horizontal) *Para duas funções monotônicas crescentes α e β , o desvio horizontal $h(\alpha, \beta)$ é dado por:*

$$h(\alpha, \beta) = \max_{s \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha(s) \leq \beta(s+T) \} \}. \quad (3.13)$$

A distância horizontal $h(\alpha, \beta)$ é o parâmetro usado para a obtenção do limiar de latência e limiar de *buffer*, cujos conceitos são discutidos abaixo. A distância horizontal assume valores positivos nos intervalos onde a curva de chegada tem valores maiores que a curva de serviços. Durante estes intervalos a capacidade de processamento é menor que a carga de trabalho de entrada e haverá portanto a necessidade de *buffer* e a ocorrência de latência.

Teorema 3.3 (Limiar de latência) *Para uma tarefa delimitada por uma curva de chegada α , com instantes diferentes e consecutivos de início, sendo servidas por uma política de *first-come-first-serve*, e um recurso que garante uma curva de serviços β , a latência $L(t)$ para todo t satisfaz:*

$$L(t) \leq h(\alpha, \beta) \forall t. \quad (3.14)$$

O significado físico desta inequação é que o limiar de latência é dado pelo intervalo de tempo transcorrido entre a ocorrência de um evento (curva de chegada) e sua finalização, como ilustrado na Figura 3.7.

Tem-se finalmente, em outra forma, o limite de Latência dado por

$$Latência \leq \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha(t) \leq \beta(t+\tau) \} \} \quad (3.15)$$

Teorema 3.4 (Limiar de *buffer*) *Assuma um fluxo delimitado por uma curva de chegada α , que atravessa um sistema que oferece uma curva de serviços β . O *buffer* $R(t) - R'(t)$ para todo t satisfaz:*

Figura 3.7: Curvas α e β explicitando as medidas de latência e buffer.

$$R(t) - R'(t) \leq \sup_{s \geq 0} \{\alpha(s) - \beta(s)\} \quad (3.16)$$

O significado físico desta inequação é que o limiar de *buffer* é dado pela máxima distância vertical entre a curva de chegada e a curva de serviços, conforme ilustrado na Figura 3.7 que representa a quantidade de dados que deve ser armazenada no sistema dada a incapacidade de o fluxo ingressante ser tratado imediatamente. Outra forma de descrever o limite de *buffer* é dada por

$$Buffer \leq \sup_{t \geq 0} \{\alpha(t) - \beta(t)\} \quad (3.17)$$

3.7 Grafo de recursos-tarefas (GRT)

3.7.1 Grafo de tarefas (GT)

Na Seção 3.3, tarefas e recursos foram definidos, respectivamente, como uma atividade atômica a ser executada e um determinado objeto físico que tem a capacidade para a sua execução. A integração de diversos recursos que executam

tarefas, forma, do ponto de vista analítico, um sistema. O grafo de tarefas (THIELE; CHAKRABORTY; NAEDELE, 2000) é a representação isolada de tarefas com a seqüência em que cada uma das delas é realizada.

Tarefas retratam um conjunto de operações implementadas como *software* ou *hardware* para manipular um conjunto de dados e apresentar um resultado ou mudança de estado. Cada uma das tarefas deve ser precisamente obtida, seja a partir da especificação ou da implementação de um determinado aplicativo ou de núcleo de *hardware*. Cada tarefa pode ser fragmentada em um subconjunto de tarefas atômicas em relação ao grau de abstração do modelo em estudo.

O grafo de tarefas (GT) pode ser formalizado como a seguir. Considere que para o conjunto de fluxos F entrando em um sistema existe um grafo de tarefas $G = (V, E)$, acíclico e direcionado. Qualquer vértice $v \in V$ denota uma tarefa de computação do sistema (processamento, armazenamento ou comunicação) sendo concluída em um recurso. Para qualquer fluxo $f \in F$, considera-se que $V(f) \subseteq V$ denota a seqüência de tarefas de processamento que deve ser implementada sobre todo evento de f . Adicionalmente, um subconjunto de arestas direcionadas de E define a ordem em que as tarefas em $V(f)$ devem ser implementadas para qualquer evento de f . Se as tarefas $u, v \in V(f)$ representam duas tarefas tais que para qualquer evento pertencente a f , a tarefa v deve ser executada imediatamente após a tarefa u , então a aresta direcionada (u, v) é pertencente a E , e v é sucessor de u . Portanto para cada fluxo f existe um único caminho que atravessa o grafo G iniciando-se em uma das fontes e terminando em um dos sumidouros. Os vértices deste caminho representam as tarefas de processamento a serem implementadas sobre os eventos de f . Com exceção da fonte, o grau de recepção dos vértices é 1, do mesmo modo, com exceção do sumidouro, o grau de emissão dos vértices é 1.

Como ilustração, a Figura 3.8 apresenta um exemplo de grafo de tarefas representativo das tarefas de encaminhamento de roteador IPv4 (DARPA, 1981). As caixas retangulares representam tarefas, cujas sub-tarefas estão listadas logo abaixo. Para a modelagem de um grafo de tarefas, deve-se realizar uma análise meticulosa a respeito do aplicativo, código fonte e/ou protocolo a ser analisado, tendo em vista o cômputo de todas e cada uma das operações funcionais a serem realizadas.

Data Link – Camada 2 (Recepção)

- a. recebe quadros Ethernet da interface PHY0 e realiza *checksum*
- b. remove preâmbulo e sequencia quadros Ethernet
- c. envia ack para remetente
- d. armazena pacote IPv4 no FIFO de RX
- e. escreve o descritor de *buffer* e carga útil na memória

Rede – Camada 3 (*Internet Layer*)

- f. processador lê o descritor de *buffer* e cabeçalho do pacote IPv4 na memória
- g. checa o conteúdo do cabeçalho IPv4
- h. checa o endereço de destino do pacote
- i. consulta a tabela de roteamento para determinar o porto de saída
- j. decrementa o campo TTL (*time-to-live*)
- k. cabeçalho IPv4 e descritor de *buffer* de transmissão são escritos na memória
- l. pacote IPv4 (cabeçalho e carga útil) e descritor de *buffer* de transmissão são enviados da memória para o MAC

Data Link – Camada 2 (Transmissão)

- m. recebe do recipiente o *ack* do quadro anterior
- n. gera o quadro Ethernet a partir do pacote IPv4
- o. insere o preâmbulo
- p. calcula o CRC a partir do quadro Ethernet
- q. envia os quadros ao PHY1.

Figura 3.8: Grafo de tarefas e tarefas associadas ilustrando a chegada de pacotes do elemento de rede ER0 e sendo enviados ao elemento de rede ER1.

3.7.2 Formalização do grafo recurso-tarefa (GRT)

O grafo recurso-tarefa, GRT (CHAKRABORTY et al., 2003), é uma expansão do grafo de tarefas, onde as tarefas são associadas a recursos físicos, portanto, trata-se de uma representação onde a microarquitetura é determinada. No GRT, o fluxo dos dados através de um sistema é modelado como uma seqüência de tarefas executadas por recursos, representada por vértices recurso-tarefa. A seqüência do GRT é a mesma do grafo de tarefas.

A exemplo do grafo de tarefas, do ponto de vista estrutural, o GRT modela a microarquitetura de *hardware* e também de *software*, assim como as correlações

entre eles, em cada um de seus vértices. Do ponto de vista funcional, o GRT deve conter, de modo distribuído, cada uma das funções a serem executadas por um determinado sistema, modeladas através de tarefas. Os problemas de mapeamento entre recursos-tarefas e a granularidade atribuída à sua modelagem não são tratadas nesta dissertação. Trabalhos mais detalhados sobre mapeamento entre recursos e tarefas através de NetCal é apresentado em (THIELE et al., 2001), (THIELE et al., 2002b), (THIELE et al., 2002b), (CHAKRABORTY et al., 2003).

O GRT correspondente ao grafo de tarefas da Figura 3.8 é ilustrado na Figura 3.9, mostrando que os vértices devem estar dispostos em colunas e em linhas, tendo finalidades específicas. As colunas representam os diferentes recursos alocáveis na microarquitetura, enquanto as linhas no GRT refletem a seqüência na qual as tarefas são executadas. Pode-se observar que as tarefas executadas por um recurso específico devem ser dispostas em linhas diferentes e uma mesma tarefa não pode ser associada a mais de um recurso. Cada vértice na Figura 3.9 representa uma tarefa específica t que está sendo executada pelo recurso s . Uma tarefa t pode ter um peso $w(s, t)$ (THIELE et al., 2002b) associado ao recurso e pode ser medida em termos de ciclos, por exemplo. Caso o recurso seja um barramento, $w(s, t)$ representa o número de ciclos que toma uma transferência. No exemplo de um processador, o peso mede o esforço em ciclos realizados pelo processador para executar a tarefa. O grafo, na forma como é definido, não suporta o desvio condicional, já que é modelado de modo determinístico para o pior ou melhor caso.

Como apresentado na Figura 3.2.b, cada vértice tem associado a si, parâmetros como as curvas de chegada (4), as curvas de serviço (4), o recurso, a tarefa e o peso; além disso, cada vértice tem vizinhos (no máximo, um sucessor, um predecessor, um inferior e um superior). Os diferentes vértices que compartilham recursos (presentes em uma mesma coluna) herdam as curvas remanescentes de serviço do vértice imediatamente superior. Os vértices do sucessor têm as curvas de chegada remanescentes dos predecessores como curvas de chegada. Ao vértice mais superior de cada recurso de um GRT, deve ser associada uma curva de serviço inicial, enquanto que, ao primeiro vértice de um grafo deve ser associada uma curva de chegada inicial. Todas as curvas de chegada e de serviço remanescentes, os *buffers*, as latências e as taxas da utilização são determinadas usando os métodos algébricos mencionados nas Seções 3.5 e 3.6. A curva de

entrada remanescente do último vértice representa toda a carga de trabalho gerada pelo fluxo inicial subtraída da carga de trabalho realizada por todos os recursos. Cada uma das curvas de serviço remanescentes (de cada recurso) representa a capacidade ociosa ao longo do cômputo de todas as tarefas do recurso.

Figura 3.9: Grafo de recursos-tarefas (GRT). Na parte superior estão representados os recursos, seguidos pelos vértices representativos do par recurso-tarefa.

Os recursos da microarquitetura podem ser divididos, do ponto de vista analítico, em três categorias: módulos de computação, as estruturas de comunicação, e estruturas de armazenamento. Os exemplos de módulos de

computação são microprocessadores programáveis de uso geral e microengenhos dedicados a entrada e saída ou criptografia. As estruturas de comunicação são arbitradores, pontes entre diferentes barramentos, multiplexadores, decodificadores e trilhas. As estruturas de armazenamento representam posições dedicadas de memória. No contexto do GRT esta distinção é importante já que cada uma delas terá curvas de serviço conceitualmente diferentes. Na seqüência do grafo de tarefas, um recurso pode ser utilizado diversas vezes, repetidamente, seja para executar tarefas diferentes de um mesmo fluxo seja para executar a mesma tarefa, obrigatoriamente para fluxos diferentes. As contenções de acesso aos recursos vistas por diferentes fluxos são explicitadas em cada vértice do GRT pela latência associada às curvas de serviços, assim como a vazão diminuída do recurso que enxerga contenção, dada pela diminuição da declividade da curva de serviços.

A informação instantânea sobre cada fluxo do pacote não é conhecida em curvas de chegada, apenas as taxas superiores, o tamanho máximo do dado (pacote) e o tamanho da rajada máxima pois são representadas no domínio do intervalo (Δ). Daí, o fluxo é calculado em cada vértice para uma tarefa dada, fazendo esta abordagem extremamente rápida quando comparada a outros métodos de simulação.

Para um fluxo dado, há a necessidade de somente um GRT ser percorrido para obter-se os valores formulados nas equações das Seções 3.5 e 3.6, porque adota-se um algoritmo não-iterativo. Um sistema (como um processador de rede) pode suportar múltiplos fluxos simultâneos, cada um com sua própria seqüência de vértices no GRT definida imediatamente após o fluxo com prioridade maior, conduzindo a curvas diferentes de chegada. A representação de diversos fluxos de dados (pacotes) ingressantes é modelada mapeando-se as tarefas do grafo de tarefas aos recursos existentes da seguinte forma: os vértices de fluxos de menor prioridade são alocados como vértices inferiores aos últimos vértices do fluxo anterior (de maior prioridade, de modo preemptivo) para cada recurso. A curva de entrada do primeiro vértice do fluxo de menor prioridade é referente exclusivamente a este fluxo ingressante, mas as curvas de serviço são as curvas de serviço remanescentes do vértice do fluxo de prioridade imediatamente maior, para o mesmo recurso. O trabalho (CHAKRABORTY; GRIES; THIELE, 2002) apresenta um estudo sobre o escalonamento temporal de fluxos com diferentes prioridades associadas.

Capítulo 4 - Extensões ao arcabouço analítico de NetCal

4.1 Introdução

No capítulo anterior foram apresentadas as questões teóricas levantadas na literatura, e formam a base conceitual para o que será apresentado neste capítulo. Aquele arcabouço teórico trata da abstração de funções ou seqüências em curvas de chegada e de serviços, além de um estudo determinístico de pior caso de latência e de *buffer*. Ainda assim, o estudo apresentado no Capítulo 3 é insuficiente para uma exploração de limiares mínimos de latência e de *buffer*, assim como os limiares de utilização de um recurso, conforme será explicitado nas próximas seções.

Neste capítulo, o arcabouço teórico é incrementado, através das contribuições específicas deste trabalho, com a inclusão do estudo de limiares mínimos para latência, *buffer* e utilização, assim como a inclusão do Critério de Estabilidade, que permite determinar a tendência de comportamento de um recurso (ou sistema). A maneira de representar as curvas é também expandida de modo a aumentar o grau de descrição, ao mesmo tempo que mantém a simplicidade conceitual. Com as contribuições presentes neste capítulo, é possível claramente aumentar a fidelidade do estudo de desempenho de sistemas, no que concerne ao estudo do trio latência, *buffer* e utilização, conforme será mostrado, de modo aplicado, no Capítulo 6.

A seguir, uma série de novos conceitos serão apresentados e, posteriormente, alguns corolários serão estabelecidos e demonstrados. O Critério de Estabilidade será apresentado e demonstrado através de um teorema, para então, serem introduzidos os conceitos de limiares mínimos de latência, *buffer* e utilização, com as suas respectivas demonstrações.

4.2 NetCal sob a ótica de curvas inferiores e superiores

Em (THIELE et al., 2002b), é proposta uma estrutura baseada em NetCal, fundamentada em álgebra de mínimos e máximos (chamada *Max-plus Algebra*), que

inclui conceitos como convolução e deconvolução e a álgebra determinística de pior caso (CRUZ, 1991a). O uso de curvas de chegada e de serviço (BOUDEC; THIRAN, 2001), assim como a aproximação linear por parte de curvas através de dois segmentos, contribuem para tornar factível o estudo de grande parte das - ou de todas - possibilidades do espaço de avaliação de projeto com a incorporação de parâmetros restritivos e conflitantes que fazem uso de alocação, mapeamento de tarefas-recursos, associação, escalonamento e particionamento (CHAKRABORTY et al., 2003). NetCal relaciona curvas da chegada às cargas de trabalho e curvas de serviço às potencialidades computacionais dos recursos; o mapeamento recurso-tarefa e o escalonamento são representados em um grafo, o GRT, onde os vértices representam os parâmetros arquiteturais que descrevem o sistema em uma maneira *coarse-grained* (a granularidade da descrição do sistema é realizada através de “grandes componentes”, com baixo grau de detalhamento em suas descrições). Tais métodos representam dinamicamente as atividades do processador, alternância de acessos de memória e operações da relação de entrada e saída.

Fizemos, então, a extensão dos aspectos acima para formular novas equações, métodos analíticos para resolvê-las e novos conceitos para modelar a dinâmica de um sistema eletrônico.

A chegada de eventos a partir de qualquer fluxo em tempo real é delimitada pelas curvas de chegada, que são descritas por funções sub-aditivas, cada uma especificando a quantidade máxima de tráfego que pode chegar dentro de um intervalo de tempo qualquer (CHAKRABORTY; GRIES; THIELE, 2002). Uma função cumulativa $R(t)$ pode descrever o rastro de uma seqüência de eventos - no caso atual, pacotes - definido como o número dos eventos vistos no intervalo de seqüência de eventos no intervalo de tempo $[0, t]$. Quando um rastro concreto de um seqüência de eventos for descrito por R , uma dupla $(\alpha_u(\Delta), \alpha_l(\Delta))$ (Figura 4.1) de curvas superiores e inferiores de chegada fornecem um modelo abstrato da seqüência de eventos, representando todos os rastros possíveis daquela seqüência de eventos. A curva $\alpha_u(\Delta)$ é delimitada pelo número máximo de pacotes que chegam em um recurso (limite superior no número dos eventos), visto na seqüência de eventos em um intervalo de tempo de comprimento Δ , e é delimitado por $\alpha_l(\Delta)$, o número mínimo de pacotes que chegam em um recurso (limite inferior no número dos eventos) em um intervalo de tempo Δ . Assim, R , α_u e α_l são relacionados entre si como se segue:

Figura 4.1: Ilustração das curvas de chegada superiores (côncavas) e inferiores (convexas).

$$\alpha_l(t-s) \leq R(t) - R(s) \leq \alpha_u(t-s), \forall s, t$$

$$\text{onde } 0 \leq s \leq t, \exists(t-s) \text{ onde } 0 \leq (t-s) \leq t. \quad (4.1)$$

com $\alpha_l(0) = \alpha_u(0) = 0$.

Pode-se notar que a curva de entrada α da equação 3.3 corresponde aqui a α_u ; portanto o conceito de curvas superiores e inferiores oferece mais flexibilidade aos conceitos apresentados no Capítulo 3, ao mesmo que exige novas demonstrações. As abstrações das curvas em componentes superiores e inferiores conferem uma abordagem determinística de pior caso (e portanto pessimista) e uma abordagem determinística de melhor caso (e portanto otimista), respectivamente.

As curvas da chegada generalizam substancialmente os modelos tradicionais de eventos tais como esporádico, periódico, periódico com *jitter*, ou todo padrão de chegada com comportamento determinístico no tempo (BOUDEDEC; THIRAN, 2001).

De uma forma similar, a capacidade de computação (seja de processamento, de armazenamento ou de comunicação) de um recurso é representada como uma curva de serviços (ver Figura 4.2). Conseqüentemente, a curva β_l é representativa do limiar inferior no número de eventos (pacotes, ou bytes no caso de recursos de comunicação ou de armazenamento) que podem ser processados por um recurso dentro de um dado intervalo de tempo; da mesma maneira, β_u é o limiar superior correspondente. Assim, a capacidade de computação de um recurso para um intervalo de tempo de comprimento limitado é sempre maior que ou igual a β_l e menor que ou igual a β_u . Assim, C , β_u e β_l são relacionados entre si como se segue:

$$\beta_l(t-s) \leq C(t) - C(s) \leq \beta_u(t-s) \quad (4.2)$$

Portanto as curvas β_l e β_u representam o envólucro mínimo e máximo, respectivamente, da função capacidade acumulada C .

Figura 4.2: Ilustração das curvas de serviço superiores (côncavas) e inferiores (convexas).

Pode-se notar que a curva de serviços β da equação 3.7 corresponde aqui a β_l . Novamente, as abstrações das curvas em componentes superiores e inferiores conferem uma abordagem determinística de pior caso (e portanto pessimista) e uma abordagem determinística de melhor caso (e portanto otimista), respectivamente. Juntos, o conceito de curvas de entrada e serviços, superiores e inferiores, contribuem para tornar mais robustos os limiares das funções capacidade e requisição acumuladas.

As expressões 4.3 a 4.6 apresentadas a seguir, propostas em (THIELE et al., 2002b), limitam a curva de chegada do fluxo de pacotes processado e a curva remanescente de serviço do recurso. α'_l e α'_u denotam as curvas *remanescentes* inferiores e superiores de chegada, respectivamente, do *fluxo processado*, e β'_l e β'_u denotam as curvas *remanescente* inferiores e superiores de serviço, respectivamente, do recurso.

$$\begin{aligned}\alpha'_l(\Delta) &= \inf_{0 \leq t \leq \Delta} \{ \alpha_l(t) + \beta_l(\Delta - t) \} \\ \Rightarrow \alpha'_l(\Delta) &= (\alpha_l \otimes \beta_l)(\Delta)\end{aligned}\tag{4.3}$$

$$\begin{aligned}\alpha'_u(\Delta) &= \inf_{0 \leq t \leq \Delta} \{ \sup_{v \geq 0} \{ \alpha_u(t+v) - \beta_l(v) \} + \beta_u(\Delta - t), \beta_u(\Delta) \} \\ \Rightarrow \inf_{0 \leq t \leq \Delta} \{ ((\alpha_u \otimes \beta_l)(\Delta) \otimes \beta_u)(\Delta), \beta_u(\Delta) \}\end{aligned}\tag{4.4}$$

$$\beta'_l(\Delta) = \sup_{0 \leq t \leq \Delta} \{ \beta_l(t) - \alpha_u(t) \}\tag{4.5}$$

$$\beta'_u(\Delta) = \sup_{0 \leq t \leq \Delta} \{\beta_u(t) - \alpha_l(t)\} \quad (4.6)$$

O significado físico destas equações é que, para o caso de curvas de chegada remanescentes, elas representam o resultado mínimo e máximo da computação da carga de trabalho egressa de um dado recurso. No caso das curvas de serviço remanescentes, tratam-se das capacidades mínimas e máximas de computação que um recurso pode oferecer para a execução de *outra tarefa*, de modo preemptivo.

4.3 Relações para α_l e β_u

As seguintes demonstrações propostas estabelecem as relações entre as funções de capacidade acumulada C , função capacidade ociosa acumulada C' , função requisição acumulada R , função computação acumulada R' com as representações abstratas em curvas β_u e α_l . Note-se que as relações apresentadas no Capítulo 3 são, de fato, referentes a β_l e α_u .

Corolário 4.1 (Relação entre R' , R e β_u - $R'(t) \leq (R \otimes \beta_u)(t)$)

Prova:

Sejam as Equações 4.1 e 4.2.

Tomando apenas

$$\begin{aligned} R(t) - R(s) &\leq \alpha_u(t-s) \\ C(t) - C(s) &\leq \beta_u(t-s) \\ R'(t) &= C(t) - \max_{0 \leq u \leq t} \{C(u) - R(u)\} \quad (\text{da equação 3.1}) \end{aligned}$$

onde o termo $\max_{0 \leq u \leq t} \{C(u) - R(u)\}$ representa o maior desperdício computacional.

Como $C(t)$ é independente na minimização,

$$\begin{aligned} R'(t) &= -\max_{u \geq t} \{C(u) - C(t) - R(u)\} \\ R'(t) &= \min_{0 \leq u \leq t} \{R(u) + C(t) - C(u)\} \\ R'(t) &\leq \min_{0 \leq u \leq t} \{R(u) + \beta_u(t-u)\} \\ R'(t) &\leq (R \otimes \beta_u)(t) \end{aligned} \quad (4.7)$$

cqd ■

Corolário 4.2 (Relação entre R e α_i - $R(t) \geq (R \otimes \alpha_i)(t)$)

Prova:

Tomando apenas parte da equação 4.1

$$\alpha_i(t-s) \leq R(t) - R(s)$$

$$\alpha_i(t-s) + R(s) \leq R(t)$$

é equivalente a

$$\begin{aligned} \min\{R(t)\} &\geq \min\{\alpha_i(t-s) + R(s)\} \\ R(t) &\geq (\alpha_i \otimes R)(t) \end{aligned} \quad (4.8)$$

Portanto $R(t) \geq \alpha_i(t), \forall t \geq 0$.

cqd ■

4.4 Aproximação de curva por múltiplas partes lineares

4.4.1 Motivação

Cruz (CRUZ, 1991a) iniciou a adoção de curvas de entrada para representar o fluxo de ingresso em nós de redes de serviços. No seu trabalho, a curva era definida com uma única inclinação (porque modelava um fluxo ingressante de taxa de transferência constante) e era limitada superiormente pela função requisição acumulada. Ainda não estava presente o conceito de aproximação de curvas por uso de segmentos. Mais tarde, em (BOUDEC, 1998), foi proposta a adequação de curvas de chegada representadas através de 2 segmentos para minimizar os erros no cálculo de latências e *buffer* e porque tinha o propósito de modelar fluxos com taxa de transferência variáveis, de modo conceitualmente semelhante aos sugeridos em (SHENKER; WROCLAWSKI, 1997). As curvas de serviço, por sua vez, somente foram representadas da mesma maneira, com o emprego de 2 segmentos, em (THIELE; CHAKRABORTY; NAEDELE, 2000). No sentido de otimizar o esforço computacional na obtenção das equações 4.3 a 4.6, Thiele et al. (THIELE et al., 2002b) mantiveram a aproximação de curvas por 2 segmentos. Note-se que em (BOUDEC; THIRAN, 2001) foi comentado, de modo breve e superficial, que tais curvas poderiam ser aproximadas por mais de 2 segmentos, ainda que não tivesse sido apresentada qualquer demonstração algébrica.

Nesta dissertação, no entanto, é proposta a não limitação no número de

segmentos na aproximação de curvas de entrada ou de serviço para a solução das equações 4.3 a 4.6. Esta abordagem é validada pela demonstração algébrica que se segue e pela comparação de resultados obtidos com a utilização de apenas 2 segmentos contra a utilização de múltiplos segmentos na aproximação de curvas, conforme aplicação e resultados apresentados no Capítulo 6.

4.4.2 Validação algébrica

A teoria de NetCal foi aplicada a processadores de rede com uma limitação de 2 no número de segmentos na aproximação de uma curva, conforme explicitado em (CHAKRABORTY et al., 2003) e (THIELE et al., 2002b). Nesta dissertação adotamos a formalização de uma aproximação de curvas por múltiplos segmentos lineares, sem limitações no número de segmentos, o que permite a modelagem mais exata para curvas de serviço e de chegada.

Deixe α_i ser a curva de chegada inferior de um conjunto de eventos ingressantes em um recurso, aproximada por m segmentos, $m \geq 1$ e denotada por (conforme ilustrado na Figura 4.3.a):

$$\alpha_i(t) = \max \{s_1 t + p_1, \dots, s_m t + p_m\}; s_m > 0 \wedge s_1 < \dots < s_m$$

Deixe β_u ser a curva superior de serviço de um recurso, aproximada por n segmentos, $n \geq 1$, e denotada por (conforme ilustrado na Figura 4.3.b):

$$\beta_u(t) = \min \{r_1 t + q_1, \dots, r_n t + q_n\}; r_n > 0 \wedge r_1 > \dots > r_n.$$

A obtenção de β'_u e α'_i (ver Figura 4.3.c) é demonstrada a seguir. De acordo com a equação 4.6,

$$\begin{aligned} \beta'_u(\Delta) &= \sup_{0 \leq t \leq \Delta} \{\beta_u(t) - \alpha_i(t)\} \\ \beta'_u(\Delta) &= \sup_{0 \leq t \leq \Delta} \{\min \{r_1 t + q_1, \dots, r_n t + q_n\} - \max \{s_1 t + p_1, \dots, s_m t + p_m\}\}; \end{aligned}$$

da teoria de álgebra de mínimos e máximos: $\max(a) = -\min(-a)$,

$$\beta'_u(\Delta) = \sup_{0 \leq t \leq \Delta} \{ \min \{ r_1 t + q_1, \dots, r_n t + q_n \} + \min \{ -s_1 t - p_1, \dots, -s_m t - p_m \} \}$$

$$\beta'_u(\Delta) = \min_{0 \leq t \leq \Delta} \{ r_1 t + q_1 - s_1 t - p_1, r_1 t + q_1 - s_2 t - p_2, \dots, r_1 t + q_1 - s_m t - p_m, \dots, r_2 t + q_2 - s_1 t - p_1, \dots, r_n t + q_n - s_m t - p_m \}$$

Figura 4.3: Exemplo de aproximação de curvas por múltiplas partes lineares. (a) Curvas convexas inferiores e (b) curvas côncavas superiores. (c) Recurso representando as equações de entrada-saída das curvas.

Assim a equação 4.6 é ainda válida e a β'_u demonstrada é formada por até mn segmentos.

Da mesma forma é possível calcular as equações 4.3 a 4.5 baseando-se na formalização de múltiplos segmentos e, claramente, esta aproximação emerge como vantajosa porque não introduz outros métodos numéricos nesse cálculo. Embora a aproximação de 2 segmentos seja interessante em termos de redução da complexidade no cálculo da equação, pode trazer erros mais elevados, dado que o cálculo das equações 4.3 a 4.6 evolui naturalmente às curvas formadas por mais de 2 segmentos, como demonstrado acima. Desta maneira, mantendo-se forçosamente em 2 o número de segmentos na aproximação é uma medida restritiva que exige o emprego de outros métodos numéricos para re-aproximação, ou, então, através do descarte de segmentos. Não fica claro, pelas informações colhidas dos trabalhos

precedentes, o grau de perda de acurácia dos resultados aproximados das equações 4.3 a 4.6, a dois segmentos, após cada computação do nó, principalmente na obtenção de α'_u e do α'_l . Em termos de demanda computacional no caso de múltiplos segmentos, é ainda possível calcular as equações 4.3 a 4.6 em uma questão de segundos, como indicado no Capítulo 6, e manter resultados mais exatos. A demonstração de cada uma das outras equações (4.3 a 4.5) pode ser feita de modo semelhante ao acima.

4.5 Critério de estabilidade

O critério de estabilidade consiste na condição para que a seqüência de computações das equações 4.3 a 4.6 venha a convergir. A condição é dada no Teorema 4.1, apresentado a seguir, como sendo a exigência de uma curva de serviço β_u maior ou igual a uma curva de chegada α_u para um intervalo Δ suficientemente longo.

Teorema 4.1 (Critério de estabilidade) - *Deixe β_u ser a curva superior côncava de serviço de um recurso representado como um vértice, e deixe α_u ser a curva superior côncava de chegada de um fluxo de eventos incorporado por este vértice. Então, o critério de estabilidade é satisfeito se, e somente se, para cada vértice, para um intervalo Δ suficientemente longo*

$$\lim_{\Delta \rightarrow \infty} \beta_u(\Delta) \geq \lim_{\Delta \rightarrow \infty} \alpha_u(\Delta) \quad (4.9)$$

Prova:

Como demonstração, obteremos β'_u (curva de serviços superior remanescente). Naturalmente, para que o recurso (ou o sistema) seja estável, este deve ter capacidade de serviço com $\beta'_u > 0$ para todo Δ suficientemente longo. Caso $\beta'_u(\Delta \rightarrow \infty) = 0$ o recurso (ou o sistema) não tem mais capacidade máxima de serviço e consequentemente será instável, naturalmente considerando-se que exista uma curva de entrada α_u não nula.

Sabemos que $C'(t) = C(t) - R'(t)$ (da equação 3.2) onde $R'(t)$ pode ser escrito como $R'(t) = \sum_{i=1}^{n-1} R'_i(t)$. Agora se faz necessário provar que β'_u cumpre $C'(t) - C'(s) \leq \beta'_u(t-s) \forall t, s: 0 \leq s \leq t$ com respeito a C' .

$$C'(t) - C'(s) = C(t) - C(s) - \left[\min_{0 \leq u \leq t} \{R(u) + (C(t) - C(u))\} - \min_{0 \leq u \leq s} \{R(u) + (C(s) - C(u))\} \right], t \geq s$$

$C(t)$ e $C(s)$ são independentes da minimização, então:

$$\begin{aligned} C'(t) - C'(s) &= C(t) - C(s) - C(t) + C(s) - \\ &\quad \left[\min_{0 \leq u \leq t} \{R(u) - C(u)\} - \min_{0 \leq u \leq s} \{R(u) - C(u)\} \right], t \geq s \\ C'(t) - C'(s) &= \left[\min_{0 \leq u \leq s} \{R(u) - C(u)\} - \min_{0 \leq u \leq t} \{R(u) - C(u)\} \right], t \geq s \\ C'(t) - C'(s) &= \left[\min_{0 \leq u \leq s, 0 \leq v \leq t} \{R(u) - C(u)\} - \{R(v) - C(v)\} \right], t \geq s, \forall v \leq t \\ C'(t) - C'(s) &= \min_{0 \leq u \leq s, 0 \leq v \leq t} \{C(v) - C(u) - R(v) + R(u)\}, t \geq s, \forall v \leq t \\ C'(t) - C'(s) &\leq \min_{0 \leq u \leq s, 0 \leq v \leq t} \{\beta_u(v-u) - \alpha_u(v-u)\}, t \geq s, \forall v \leq t \\ \text{Mas } C'(t) - C'(s) &\leq \beta'_u(t-s) \end{aligned}$$

Tomando $d = v-u$

$$\begin{aligned} \beta'_u(t-s) &\geq \min_{0 \leq u \leq s, d \leq t-u} \{\beta_u(d) - \alpha_u(d)\}, t \geq s, \text{ ou} \\ \beta'_u(t-s) &= \sup_{0 \leq d \leq t-s} \{\beta_u(d) - \alpha_u(d)\} \geq \min_{0 \leq u \leq s, d \leq t-u} \{\beta_u(d) - \alpha_u(d)\}, t \geq s \end{aligned}$$

Fazendo t suficientemente alto e $d=v-u$; $d \rightarrow \infty$.

Neste caso, $\alpha_l(d) \geq \alpha_u(d)$, pois são convexas e côncavas, respectivamente.

$$\begin{aligned} \beta'_u(t-s) &= \sup_{0 \leq d \leq t-s} \{\beta_u(d) - \alpha_l(d)\} \leq \min_{0 \leq u \leq s, d \leq t-u} \{\beta_u(d) - \alpha_u(d)\}, t \geq s, d \rightarrow \infty \\ \beta'_u(t-s) &\leq \min_{0 \leq u \leq s, d \leq t-u} \{\beta_u(d) - \alpha_u(d)\}, t \geq s, d \rightarrow \infty \end{aligned}$$

Caso $\{\beta_u(d) - \alpha_u(d)\} \leq 0$, o sistema não apresenta capacidade mínima de serviço remanescente a partir de um intervalo “ d ” e portanto é instável.

cqd ■

O teorema estabelece que a declividade do último segmento da curva β_u deve ser maior que a declividade do último segmento da curva α_u , como exemplificado na Figura 4.3.b.

4.6 Limiares mínimos de Latência, requisição de *Buffer* e Utilização de recurso – o trio (L-B-U)_{mínimo}

Em (CHAKRABORTY; GRIES; THIELE, 2002), (CHAKRABORTY et al., 2003), (GRIES et al., 2003) e (THIELE et al., 2002b), os autores forneceram um estudo de sistema através dos limiares superiores de latência, *buffer* e utilização de recurso, o trio (L-B-U)_{máximo}, correspondendo às equações 4.10, 4.11 e 4.12. Neste trabalho, formalizamos o conceito (recíproco) de limiares inferiores como uma forma de prever os valores mais otimistas para o trio (L-B-U), que chamaremos de (L-B-U)_{mínimo}. Em termos práticos, um sistema não deve ser projetado com valores menores do que o trio (L-B-U)_{mínimo} para um conjunto dado de curvas de chegada e de serviços.

Se o critério de estabilidade descrito na Seção 4.5 for satisfeito, o recurso é capaz de terminar as tarefas relacionadas aos eventos confinados pela curva superior de chegada, decorrida uma latência limitada e tendo consumido um *buffer* limitado. Se o critério não for satisfeito, a latência e o *buffer*, mínimos e máximos, tenderão ao infinito, saindo da estabilidade. O critério de estabilidade deve ser satisfeito, portanto, também para possibilitar o cálculo de (L-B-U)_{máximo}.

Conforme mostrado no Capítulo 3, a máxima latência fim-a-fim vista por um fluxo de eventos ingressante e o máximo *buffer* requerido para poder processá-lo são dados pela maior distância horizontal e vertical, respectivamente, entre duas curvas α e β (equações 3.14 e 3.16).

Com a aplicação do conceito de curvas superiores e inferiores, as equações 3.15 e 3.17 passam a ser particularizadas nas equações 4.10 e 4.11 abaixo.

$$Latência_{máxima} \leq \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha_u(t) \leq \beta_l(t + \tau) \} \} \quad (4.10)$$

$$Buffer_{maximo} \leq \sup_{t \geq 0} \{ \alpha_u(t) - \beta_l(t) \} \quad (4.11)$$

Em (CHAKRABORTY; KUNZLI; THIELE, 2003) é formalizado o seguinte conceito, de forma intuitiva:

$$utilização_{máxima} \leq \lim_{\Delta \rightarrow \infty} \frac{\{ \beta_u(\Delta) - \beta'_l(\Delta) \}}{\beta_u(\Delta)} \quad (4.12)$$

que corresponde à máxima utilização em função do tempo para um dado recurso,

tomando o último segmento das curvas côncavas superiores e convexas inferiores.

4.6.1 Limiares mínimos de Latência e *Buffer*

Deixe α_l e α_u representarem as curvas inferiores e superiores de chegada de um fluxo de eventos que incorpora um recurso, tal que as curvas de serviço são dadas por β_l e β_u . Então a latência mínima (ou o tempo de resposta) experimentada por um evento no recurso representado pelas curvas do serviço, e o *buffer* mínimo (número de eventos reservados) do fluxo que está esperando para ser processado, podem ser dados pelas desigualdades apresentadas a seguir.

Se o critério de estabilidade for satisfeito,

$$\begin{aligned} \text{latência}_{\text{mínima}} \geq & \min \left\{ \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha_u(t) \leq \beta_u(t + \tau) \} \}, \right. \\ & \left. \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha_l(t) \leq \beta_l(t + \tau) \} \}, \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha_l(t) \leq \beta_u(t + \tau) \} \} \right\} \end{aligned} \quad (4.13)$$

$$\text{buffer}_{\text{mínimo}} \geq \min \{ \sup_{t \geq 0} \{ \alpha_u(t) - \beta_u(t) \}, \sup_{t \geq 0} \{ \alpha_l(t) - \beta_l(t) \} \} \quad (4.14)$$

Fisicamente, o *buffer* deve ser imaginado como possuindo apenas uma entrada e uma saída, com capacidade infinita de armazenamento.

Corolário 4.3 (Limiar mínimo de latência para α_u e β_u) Para uma tarefa delimitada por uma curva de chegada α_u e com um recurso garantindo uma capacidade de serviço disponível de β_u , a latência $L(t)$ satisfaz:

$$L(t) = \max_{v \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha_u(v) \leq \beta_u(v + T) \} \}$$

Prova: Dado que $L(t) = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t + T) \}$ (da equação 3.12)

e tomando apenas

$$\begin{aligned} R(t) - R(s) & \leq \alpha_u(t - s) \\ C(t) - C(s) & \leq \beta_u(t - s) \end{aligned}$$

das equações 4.1 e 4.2, de acordo com o Corolário 4.1,

$$\begin{aligned} L(t) & = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t + T) \} \\ L(t) & = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t + T) \leq (R \otimes \beta_u)(t + T) \} \\ L(t) & \geq \min \{ T : T \geq 0 \wedge R(t) \leq (R \otimes \beta_u)(t + T) \} \\ L(t) & \geq \min \{ T : T \geq 0 \wedge R(t) \leq \inf_{0 \leq u \leq t + T} \{ R(u) + \beta_u(t + T - u) \} \} \end{aligned}$$

Para um dado u_s onde ocorre o mínimo

$$L(t) = \min \{ T : T \geq 0 \wedge R(t) \leq R(u_s) + \beta_u(t + T - u_s) \}$$

Tem-se agora as possibilidades de u_s .

1. Para $u_s \leq t : R(t) - R(u_s) \leq \alpha_u(t - u_s)$
2. Para $u_s > t : R(t) - R(u_s) \leq \alpha_u(t - u_s)$

Para $u_s \leq t : R(t) - R(u_s) \leq \alpha_u(t - u_s)$

$$L(t) = \min \{ T : T \geq 0 \wedge R(t) - R(u_s) \leq \beta_u(t + T - u_s) \}$$

$$L(t) \leq \min \{ T : T \geq 0 \wedge \alpha_u(t - u_s) \leq \beta_u(t + T - u_s) \}$$

Pois caso $\alpha_u(t - u_s) \geq \beta_u(t + T - u_s)$ tem latência percebida nula.

Para um dado $v = t - u_s$

$$L(t) \leq \min \{ T : T \geq 0 \wedge \alpha_u(v) \leq \beta_u(v + T) \}$$

Como a equação de latência deve ser válida para todo t , é também válida para todo $v = t - u_s$ e em particular para o caso de v que causa o máximo T .

$$L(t) = \max_{v \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha_u(v) \leq \beta_u(v + T) \} \}$$

que é correspondente a $h(\alpha_u, \beta_u)$.

Para $u_s > t$, $R(t) \leq \inf_{0 \leq u \leq t+T} \{ R(u) + \beta_u(t + T - u) \}$ continua sendo verdadeira e vale o resultado acima.

cqd ■

Corolário 4.4 (Limiar mínimo de latência para α_l e β_l) Para uma tarefa delimitada por uma curva de chegada α_l e com um recurso garantindo uma capacidade de serviço disponível de β_l , a latência $L(t)$ satisfaz:

$$L(t) = \max_{v \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha_l(v) \leq \beta_l(v + T) \} \}$$

Prova: Dado que $L(t) = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t + T) \}$ (da equação 3.12)

e tomando apenas

$$\alpha_l(t - s) \leq R(t) - R(s)$$

$$\beta_l(t - s) \leq C(t) - C(s)$$

das equações 4.1 e 4.2, de acordo com o Corolário 3.4,

$$\begin{aligned}
L(t) &= \min \{ T : T \geq 0 \wedge R(t) \leq R'(t+T) \} \\
L(t) &= \max \{ T : T \geq 0 \wedge R(t) > R'(t+T) \} \\
L(t) &= \max \{ T : T \geq 0 \wedge R(t) > R'(t+T) \geq (R \otimes \beta_l)(t+T) \} \\
L(t) &\leq \max \{ T : T \geq 0 \wedge R(t) > (R \otimes \beta_l)(t+T) \} \\
L(t) &\leq \max \{ T : T \geq 0 \wedge R(t) > \inf_{0 \leq u \leq t+T} \{ R(u) + \beta_l(t+T-u) \} \}
\end{aligned}$$

Para um dado u_s onde ocorre o mínimo

$$L(t) = \max \{ T : T \geq 0 \wedge R(t) > R(u_s) + \beta_l(t+T-u_s) \}$$

Tem-se agora as possibilidades de u_s .

1. Para $u_s \leq t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$
2. Para $u_s > t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$

Para $u_s \leq t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$

$$\begin{aligned}
L(t) &= \max \{ T : T \geq 0 \wedge R(t) - R(u_s) > \beta_l(t+T-u_s) \} \\
L(t) &= \min \{ T : T \geq 0 \wedge R(t) - R(u_s) \leq \beta_l(t+T-u_s) \} \\
L(t) &\geq \min \{ T : T \geq 0 \wedge \alpha_l(t-u_s) \leq R(t) - R(u_s) \leq \beta_l(t+T-u_s) \} \\
L(t) &\geq \min \{ T : T \geq 0 \wedge \alpha_l(t-u_s) \leq \beta_l(t+T-u_s) \}
\end{aligned}$$

Para um dado $v=t-u_s$

$$L(t) \geq \min \{ T : T \geq 0 \wedge \alpha_l(v) \leq \beta_l(v+T) \}$$

Como a equação de latência deve ser válida para todo t , é também válida para todo $v=t-u_s$ e em particular para o caso de v que causa o máximo T .

$$L(t) = \max_{v \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha_l(v) \leq \beta_l(v+T) \} \}$$

que é correspondente a $h(\alpha_l, \beta_l)$.

Para $u_s > t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$ é necessário que $R(t) < \inf_{0 \leq u \leq t+T} \{ R(u) + \beta_l(t+T-u) \}$

e portanto viola a condição do Corolário 3.4 e não pode acontecer.

cqd ■

Corolário 4.5 (Limiar mínimo de latência para α_l e β_u) Para uma tarefa delimitada por uma curva de chegada α_l e com um recurso garantindo uma capacidade de serviço disponível de β_u , a latência $L(t)$ satisfaz:

$$L(t) = \max_{v \geq 0} \{ \min \{ T : T \geq 0 \wedge \alpha_l(v) \leq \beta_u(v+T) \} \}$$

Prova: Dado que $L(t) = \min \{ T : T \geq 0 \wedge R(t) \leq R'(t+T) \}$ (da equação 3.12)

e tomando apenas

$$\begin{aligned}\alpha_l(t-s) &\leq R(t) - R(s) \\ C(t) - C(s) &\leq \beta_u(t-s)\end{aligned}$$

das equações 4.1 e 4.2, de acordo com o Corolário 4.1,

$$\begin{aligned}L(t) &= \min\{T : T \geq 0 \wedge R(t) \leq R'(t+T)\} \\ L(t) &= \min\{T : T \geq 0 \wedge R(t) \leq R'(t+T) \leq (R \otimes \beta_u)(t+T)\} \\ L(t) &\leq \min\{T : T \geq 0 \wedge R(t) \leq (R \otimes \beta_u)(t+T)\} \\ L(t) &\leq \min\{T : T \geq 0 \wedge R(t) \leq \inf_{0 \leq u \leq t+T} \{R(u) + \beta_u(t+T-u)\}\}\end{aligned}$$

Para um dado u_s onde ocorre o mínimo

$$L(t) = \min\{T : T \geq 0 \wedge R(t) \leq R(u_s) + \beta_u(t+T-u_s)\}$$

Tem-se agora as possibilidades de u_s .

1. Para $u_s \leq t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$
2. Para $u_s > t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$

Para $u_s \leq t : \alpha_l(t-u_s) \leq R(t) - R(u_s)$

$$L(t) \geq \min\{T : T \geq 0 \wedge \alpha_l(t-u_s) \leq \beta_u(t+T-u_s)\}$$

Para um dado $v=t-u_s$

$$L(t) \geq \min\{T : T \geq 0 \wedge \alpha_l(v) \leq \beta_l(v+T)\}$$

Como a equação de latência deve ser válida para todo t , é também válida para todo $v=t-u_s$ e em particular para o caso de v que causa o máximo T .

$$L(t) = \max_{v \geq 0} \{\min\{T : T \geq 0 \wedge \alpha_l(v) \leq \beta_u(v+T)\}\}$$

que é correspondente a $h(\alpha_l, \beta_u)$.

Para $u_s > t$, $R(t) \leq \inf_{0 \leq u \leq t+T} \{R(u) + \beta_u(t+T-u)\}$ continua sendo verdadeira e vale o resultado acima.

cqd ■

Vamos demonstrar adiante os limiares mínimos de *buffer*. Sabemos que o *buffer* é a distância vertical entre a curva de chegada e a curva de serviço.

Corolário 4.6 (Limiar mínimo de *buffer* para α_u e β_u) Para uma tarefa delimitada por uma curva de chegada α_u e com um recurso garantindo uma capacidade de

serviço disponível de β_u , a requisição de *buffer* satisfaz:

$$R(t) - R'(t) \geq \sup_{s \geq 0} \{\alpha_u(s) - \beta_u(s)\}$$

Prova: Tomando apenas

$$R(t) - R(s) \leq \alpha_u(t-s)$$

$$R(t) - R(t-s) \leq \alpha_u(s)$$

das equações 4.1 e 4.2, de acordo com o Corolário 4.1,

$$R'(t) \leq (R \otimes \beta_u)(t)$$

$$R(t) - R'(t) \geq R(t) - (R \otimes \beta_u)(t)$$

$$R(t) - R'(t) \geq R(t) - \inf_{0 \leq s \leq t} \{R(t-s) + \beta_u(s)\}$$

$$R(t) - R'(t) \geq R(t) + \sup_{s \geq 0} \{-R(t-s) - \beta_u(s)\}$$

Como $R(t)$ é independente na minimização,

$$R(t) - R'(t) \geq \sup_{s \geq 0} \{R(t) - R(t-s) - \beta_u(s)\}$$

$$-R(t) + R'(t) \leq -\sup_{s \geq 0} \{R(t) - R(t-s) - \beta_u(s)\}$$

$$-R(t) + R'(t) \leq -\sup_{s \geq 0} \{\alpha_u(s) - \beta_u(s)\}$$

$$R(t) - R'(t) \geq \sup_{s \geq 0} \{\alpha_u(s) - \beta_u(s)\}$$

cqd ■

Corolário 4.7 (Limiar mínimo de *buffer* para α_l e β_l) Para uma tarefa delimitada por uma curva de chegada α_l e com um recurso garantindo uma capacidade de serviço disponível de β_l , a requisição de *buffer* satisfaz:

$$R(t) - R'(t) \geq \sup_{s \geq 0} \{\alpha_l(s) - \beta_l(s)\}$$

Prova: Tomando apenas

$$\alpha_l(t-s) \leq R(t) - R(s)$$

$$\alpha_l(s) \leq R(t) - R(t-s)$$

das equações 4.1 e 4.2, de acordo com o Corolário 3.4,

$$R'(t) \geq (R \otimes \beta_l)(t)$$

$$\begin{aligned}
R(t) - R'(t) &\leq R(t) - (R \otimes \beta_l)(t) \\
R(t) - R'(t) &\leq R(t) - \inf_{0 \leq s \leq t} \{ R(t-s) + \beta_l(s) \} \\
R(t) - R'(t) &\leq R(t) + \sup_{s \geq 0} \{ -R(t-s) - \beta_l(s) \}
\end{aligned}$$

Como $R(t)$ é independente na minimização,

$$\begin{aligned}
R(t) - R'(t) &\leq \sup_{s \geq 0} \{ R(t) - R(t-s) - \beta_l(s) \} \\
-R(t) + R'(t) &\geq -\sup_{s \geq 0} \{ R(t) - R(t-s) - \beta_l(s) \} \\
-R(t) + R'(t) &\geq -\sup_{s \geq 0} \{ \alpha_l(s) - \beta_l(s) \} \\
R(t) - R'(t) &\leq \sup_{s \geq 0} \{ \alpha_l(s) - \beta_l(s) \}
\end{aligned}$$

cqd ■

Corolário 4.8 (Limiar mínimo de *buffer* para α_l e β_u)

Prova-se de modo similar às demonstrações anteriores.

4.6.2 Limiares mínimos de Utilização

Agora, a respeito dos limiares mínimos, se β_u e o β_l forem a curva superior inicial de serviço e a curva inferior final de serviço respectivamente, nós propomos, neste trabalho, o limiar mínimo de utilização, que é dado por:

$$\begin{aligned}
Utilização_{mínima} &\geq \min \left\{ \lim_{\Delta \rightarrow \infty} \frac{|\beta_u'(\Delta) - \beta_l(\Delta)|}{\beta_l(\Delta)}, \right. \\
&\quad \left. \lim_{\Delta \rightarrow \infty} \frac{\beta_l(\Delta) - \beta_l'(\Delta)}{\beta_l(\Delta)}, \lim_{\Delta \rightarrow \infty} \frac{\beta_u(\Delta) - \beta_u'(\Delta)}{\beta_u(\Delta)} \right\} \quad (4.15)
\end{aligned}$$

A utilização é calculada como a relação entre a diferença entre as declividades das curvas de serviço iniciais e remanescentes, e a curva de serviço inicial.

4.7 Grafo de recursos-tarefas com extensões

Na Seção 3.7 o grafo de recursos-tarefas (GRT) foi apresentado como uma modelagem da microarquitetura de hardware e também de software, assim como as

correlações entre eles. Do ponto de vista funcional, o GRT deve conter, de modo distribuído, cada uma das funções a serem executadas por um determinado sistema, modeladas através de tarefas. Em um GRT, o fluxo dos dados através de um sistema é modelado como uma seqüência de tarefas executadas por recursos, representada por vértices recurso-tarefa, como mostrado na Figura 3.9. Cada vértice tem parâmetros associados, como as curvas de chegada e de serviço, iniciais e remanescentes. Os diferentes vértices que compartilham recursos (presentes em uma mesma coluna) herdam as curvas remanescentes de serviço do vértice imediatamente superior. O vértice sucessor tem as curvas de chegada remanescentes do predecessor como curva de chegada. No GRT, todas as curvas de chegada e de serviço remanescentes, os *buffers*, as latências e as taxas da utilização são determinadas usando os métodos algébricos mencionados no Capítulo 3.

Com as extensões introduzidas neste capítulo, para cada recurso de um GRT, ao seu vértice mais superior devem ser associadas curvas de serviço superiores e inferiores. O primeiro vértice de um grafo deve ter associado uma curva de chegada inferior e uma curva de chegada superior. Em um estudo detalhado do GRT, pode-se avaliar a latência, o *buffer* e a utilização de recursos correspondentes a cada vértice, expondo as exigências de cada tarefa. A partir do cômputo das equações 4.3 a 4.6 e das equações 4.10 a 4.15, é possível obter os trios $(L-B-U)_{\text{máximo}}$ e $(L-B-U)_{\text{mínimo}}$ para cada recurso e para o sistema.

A informação instantânea sobre cada fluxo do pacote não é conhecida em curvas de chegada, apenas as taxas inferiores e superiores, o tamanho máximo do pacote e o tamanho da rajada máxima, pois são representadas no domínio do intervalo de tempo (Δ). Daí, o fluxo é calculado em cada vértice para uma tarefa dada, fazendo esta abordagem extremamente rápida quando comparada a outros métodos de simulação.

4.8 Grafo de recursos-tarefas expandido (GRTE)

Na seção anterior e na Seção 3.7 foi feita uma breve apresentação do GRT; nesta seção é apresentada uma contribuição desta dissertação para estender conceitualmente a modelagem do comportamento de microarquitetura de processadores de rede, através do conceito de GRT expandido (GRTE). Um GRTE

baseia-se na fragmentação da representação de um recurso no GRT de modo que um mesmo recurso passe a constar mais de uma vez no GRT; como consequência, na estrutura do grafo mais de uma coluna será relacionada a um único recurso.

A justificativa para a fragmentação de vértices do GRT é que, em alguns tipos de recursos, caso a sua funcionalidade seja particionada em diferentes tarefas, estas não são executadas de modo segmentado (em *pipeline*) e simultâneo. Uma tarefa de tal recurso não deveria apresentar ao vértice inferior uma curva de serviço remanescente. O vértice inferior deveria, isto sim, enxergar uma curva de serviço idêntica à curva inicial, assim como a vista pelo primeiro vértice na coluna do recurso. Tal situação é ilustrada na Figura 3.9, onde a memória é modelada (ainda que com múltiplos portos) como executando mais de uma tarefa (de escrita ou leitura) ao longo de um intervalo de tempo. Assim, a próxima atividade de leitura ou de escrita não enxergará toda a capacidade de processamento desta memória (a curva de serviço inicial), como seria mais preciso.

Os controladores de armazenamento, assim como os de barramento, tipicamente, não necessitam manter informação de estado na realização de trabalhos diferentes, ou seja, são *stateless*. As curvas de serviço são funções *burst-delay* (funções degrau). No caso mais comum, memórias e barramentos, estão restritos a operações de escrita e leitura, sendo bons candidatos à fragmentação. A segmentação de um barramento geralmente envolve apenas uma ou muito poucas operações simultâneas de decodificação de endereços, escrita e leitura em multiplexadores dedicados a cada uma delas. Esta característica reflete-se na construção do GRTE pela fragmentação deste recurso, por exemplo, em dois recursos (e portanto duas colunas no GRTE), sendo um deles para escrita e outro para leitura. Uma memória com múltiplos portos (*multiported*) apresenta conceituação semelhante. Na Figura 4.4 é ilustrado um GRTE, onde o recurso memória (Mem) foi expandido de um único (na Figura 3.9) para quatro, o mesmo ocorrendo com os componentes do barramento (AHB). É importante notar que a fragmentação de recursos não viola a formalização do GRT apresentada anteriormente.

A saber, módulos processadores ou as estruturas de comunicação, quando segmentadas, mantém informação do estado sobre trabalhos diferentes em um determinado intervalo do tempo (funções da *rate-latency*), portanto os vértices inferiores de um mesmo recurso herdam as curvas remanescentes de serviço do vértice imediatamente superior. Processadores genéricos normalmente apresentam segmentação por diversos estágios, onde cada estágio cumpre parte de uma instrução (do conjunto de instruções daquela microarquitetura). No entanto, uma tarefa pode exigir a execução de uma a milhares destas instruções, tornando um recurso como este um candidato não ideal à fragmentação de recursos.

Figura 4.4: GRTE representando a microarquitetura sob simulação (MSS)

A fragmentação contribui para o aumento na precisão dos resultados de $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$ na medida em que a interferência de vértices (*stateless*) nas curvas de serviço remanescente de um mesmo recurso é significativamente diminuída. Também contribui para uma análise mais detalhada e precisa a respeito da dinâmica de funcionamento do microssistema sob estudo pois mais recursos *conceituais* são avaliados. Como desvantagem desta abordagem está o fato de passar a exigir maior quantidade de cálculos para a obtenção dos parâmetros desejados.

Capítulo 5 - Modelagem de um processador de rede

5.1 Introdução

A escolha de processadores de rede como aplicação-alvo nesta dissertação é baseada na consciência da importância destes sistemas dentro da área de eletrônica. A necessidade de conectividade entre dispositivos de forma segura e global (ao redor do mundo), associada às possibilidades de uma computação presente em todos os lugares sem ser notada, determina a criação de tráfego de comunicações de dados entre sistemas de redes de pacotes ou redes locais de armazenamento sem precedentes, e que necessitará ser gerenciado de modo confiável e sistemático. Tais objetivos e avanços podem depender fundamentalmente de processadores de rede, que estão presentes nos equipamentos de comunicação de dados por pacotes.

Em redes locais de armazenamento SAN, (*Storage Area Networks*), encontram-se, também, funcionalidades especiais que exigem circuitos eletrônicos sofisticados capazes de prover a identificação, processamento, comutação e encaminhamento de todos os pacotes de dados de modo efetivo.

A demanda por largura de banda em redes de comunicação digital por pacotes tem sido decorrente de uma combinação de fatores como a grande quantidade de atividades baseada em Web, o comércio eletrônico, as redes locais de armazenamento de dados e a procura por aplicativos de multimídia remotos. Somem-se a estas funcionalidades, os aspectos tecnológicos criados com a invenção de novas tecnologias e protocolos de comunicação de dados.

Neste capítulo, definimos um modelo microarquitetural de processador de rede, adotado como objeto de estudo. Primeiramente, alguns aspectos gerais relacionados a processadores de rede são apresentados, para, depois, descrevermos os componentes da microarquitetura. Também, em seções posteriores, detalhamos o modelo no nível de transferência entre registradores do processador de rede adotado, o qual será usado para a validação do modelo analítico correspondente. O modelo analítico é também apresentado, assim como o ferramental desenvolvido para a sua utilização.

5.2 Aspectos de processadores de rede

5.2.1 Módulos de processamento de datagramas

Processadores de rede, também conhecidos na literatura como Unidades de Processamento de Rede (*NPU - Network Processing Units*) são circuitos eletrônicos integrados altamente especializados e dedicados à implementação de tarefas especificadas em protocolos de redes de comunicação de dados. Têm como um dos principais objetivos, a otimização das operações de processamento de pacotes. O termo pacote neste momento é utilizado de modo genérico para designar uma unidade de datagrama, seja uma célula, um pacote propriamente dito ou um quadro. Quadro e pacote representam praticamente o mesmo conteúdo, mas diferem em uma questão de nomenclatura, pois quadro é utilizado para referência entre a camada física (PHY) e a camada de enlace de dados, ao ponto que pacote é utilizado para referência entre a camada de enlace de dados e a camada de rede.

Processadores de rede formam um conjunto de circuitos integrados especializados, cuja arquitetura alvo é concebida com uma única finalidade (ou muito poucas) em mente e, portanto, altamente dedicados, com um conjunto de instruções refinadas para o propósito de processamento de pacotes; apesar disso, é bastante comum, de modo diferente, encontrar microarquiteturas baseadas em processadores programáveis de propósito genérico, como veremos na Seção 5.2.4. Os chamados micro-engenhos contidos em processadores de rede são conectados para desempenhar operações a taxas nominais de transferência. Em geral, há ainda um ou mais processadores embarcados que permitem a execução de sistema operacional a tempo real.

Os componentes encontrados e tarefas tipicamente executadas em um processador de rede podem ser enumeradas como se segue e são ilustradas na Figura 5.1.

A **interface PHY** é a primeira camada (a de mais baixo nível) conceitual de funcionalidade, geralmente como parte de em um componente integrado ou como o circuito transceptor. O enfileiramento de bits, que devem ser transmitido por uma rede, deve ser modificado eficientemente de sua forma digital para uma forma

analógica, a ser transmitida por um meio físico de comunicação, e vice-versa, na recepção.

Switch fabric: trata-se de um módulo funcional que lê pacotes em um determinado porto de entrada (também conhecido como ponto de ingresso) e os roteia até a saída apropriada (também chamada de ponto de egresso). Sua velocidade, dada pela vazão (*throughput*) e latência são os fatores mais críticos para definir a capacidade de comutação de um sistema digital de comunicação em rede. Como alternativa, pode-se optar por um projeto personalizado de um circuito projetado e construído com técnicas de VLSI.

Processamento de pacotes: o conjunto genérico de operações de processamento de pacotes está localizado logicamente entre as interfaces do circuito transceptor PHY e o *switch fabric*. Usualmente estas operações são discriminadas em duas categorias: caminho de dados rápido e caminho de dados lento de processamento de pacotes. O caminho rápido de processamento de pacotes refere-se a uma via de dados que é responsável por operações em tempo real executadas diretamente sobre o pacote. Isto inclui, mas não se limita a, cinco operações consideradas fundamentais: de enquadramento e/ou análise sintática, classificação, modificação, compressão e/ou criptografia e enfileiramento. Um caminho de dados lento refere-se a operações requeridas mas que são executadas de modo independente do fluxo de pacotes, como, por exemplo, a resolução de endereços desconhecidos, novos cálculos para atualizações de tabelas de roteamento e encaminhamento.

Processamento externo: trata-se de um conjunto de operações genéricas, não diretamente relacionadas ao tratamento de pacotes e que não estão envolvidas no fluxo destes pacotes. Tipicamente são associadas a uma unidade central de processamento (que pode ser um processador programável de propósito genérico ou um circuito dedicado) que não trata pacotes diretamente. Tais operações incluem, por exemplo, rotinas de gerenciamento, recarga de valores, configuração de registradores de controle, execução de diagnósticos. São tarefas comumente implementadas na forma de *firmwares* que são executadas na unidade central de processamento.

Figura 5.1: Ilustração da concepção funcional do particionamento de processador de rede (PANOS, 2003)

5.2.2 Operações de processamento de pacotes

A seguir apresentamos um breve apanhado conceitual sobre a série de operações de processamento de pacotes.

- **Enquadramento de pacotes:** no ambiente Ethernet, as camadas de enlace (MAC) e física (PHY) são geralmente implementadas em diferentes circuitos (encapsulamentos), o que pode exigir o processo de serialização (fragmentação do quadro com adição de estrutura de dados para futura reconstrução), antes ou depois de serem enviados ao transceptor.
- **Busca por padrões e classificação de pacotes:** uma tarefa de classificação de pacotes tem por propósito garantir que algumas regras e condições sejam aplicadas em tempo real para todo pacote ingressante e, mais especificamente para seus cabeçalhos ou para parte de sua carga útil; isto para designar o pacote em um ou mais enquadramentos lógicos,

geralmente relacionado à qualidade de serviço ou decisões de encaminhamentos, durante o roteamento ou comutação. Tipicamente uma tabela de endereços ou todo um conjunto de regras e políticas devem ser pesquisadas em tempo real, como, por exemplo, em um ambiente de serviços diferenciados (DiffServ), onde tais buscas são executadas em diversos campos dos cabeçalhos TCP e IP. É na tarefa de classificação que ocorre a procura dos cinco campos na extração da informação apropriada para definir o encaminhamento. Mais especificamente, tais campos são: o endereço IP de origem (32 bits), o endereço IP de destino (32 bits), o protocolo especificado (8 bits), a porta TCP de origem (16 bits), a porta TCP de destino (16 bits).

- **Análise sintática de pacote:** a inspeção minuciosa de pacotes de comprimento variável pode não ser trivial, dependendo do particionamento definido entre *hardware* e *software*. Circuitos integrados dedicados a estas funções são potencialmente bem enquadrados.

- **Encaminhamento de pacotes:** encaminhamento refere-se à seleção de um porto de saída por um roteador ou comutador baseado no endereço de destino do pacote em conjunto com a tabela de roteamento que estipula qual porta de saída deve ser o destino. Por outro lado, o roteamento, macroscopicamente falando, refere-se ao processo de montagem da tabela de roteamento em si.

- **Modificação:** aplica-se este conceito ao fato de que certos campos do cabeçalho do pacote devem ser totalmente alterados, como o CRC e *checksums* (checagem da integridade dos dados recebidos).

- **Enfileiramento (tratamento de tráfego e outras operações):** em certo estágio da cadeia de processos, o escalonamento temporal deve ser aplicado ao pacote. As filas de pacotes devem ser ordenadas segundo determinadas classes ou categorias de serviços.

5.2.3 Tipos de arquiteturas de encaminhamento.

Em uma arquitetura do tipo armazena-e-encaminha, o pacote é armazenado temporariamente e somente então é decidido o que deve ser feito com ele. Por um lado, dá à arquitetura mais flexibilidade e aumenta o espectro de aplicações no

resultado final do processo de manuseio deste pacote. Por outro lado, implica em maior custo de implementação, já que mais armazenamento passa a se fazer necessário e também maior latência fim-a-fim passa a acompanhar o pacote.

Uma arquitetura de “atalho” (*cut-through*) elimina ambas as desvantagens de maior armazenamento e maior latência da arquitetura anterior, por processar o encaminhamento do pacote tão logo o seu cabeçalho seja recebido, baseado em bits específicos que são tratados em tempo real. Em muitos casos, dadas as taxas de transmissão, todo um conjunto de decisões deve ser tomado antes mesmo de o pacote ter entrado no roteador ou comutador.

Roteadores que estão localizados, na topologia de rede, nas suas margens, são chamados de roteadores *edge* (ainda que seja uma nomenclatura baseada na referência de onde o elemento está situado na topologia da rede, e por conseqüência, sob influência da quantidade e tipos de tráfego gerados). A localização na topologia da rede faz com que diferentes exigências em termos de suporte a protocolos, velocidade e quantidade de portas, façam-se necessárias. Roteadores *core* estão localizados no centro da topologia da rede, e de modo típico suportam múltiplas interfaces e protocolos de comunicação. Outro nível na topologia de rede é a rede de acesso, também conhecida como “rede da operadora de telecomunicações”, como a de um provedor de conexão de acesso à Internet. Redes de acesso agregam o tráfego de usuários de banda larga, por exemplo. Tal tráfego é então encaminhado aos roteadores *edge* e, posteriormente, aos roteadores *core* da operadora. Roteadores *edge*, ainda que expostos a menores vazões agregadas em suas portas, podem ter a incubência de realizar tarefas sobre campos pertencentes não somente ao cabeçalho, exigindo alto poder computacional para manter o processamento de pacotes em tempo real. Por outro lado, roteadores *core* estão, tipicamente, expostos a tarefas menos complexas, porém a altas taxas de vazões agregadas, exigindo também alto poder computacional para manter o processamento de pacotes em tempo real com relação às taxas de transferências nominais.

5.2.4 Processadores programáveis ou microengenhos dedicados?

Operações de processamento de pacotes podem ser implementadas como *software* que roda em processadores programáveis de propósito genérico, por conta da facilidade e flexibilidade com que um processador pode ser programado. Para obter novas funcionalidades, uma nova versão de aplicativo com as adições e correções necessárias podem ser programadas no sistema. Normalmente o tempo requerido para estas modificações é muito menor que o tempo de alteração de todo um circuito eletrônico integrado, o que traz vantagens não somente para os responsáveis pelo projeto em si, mas também para os usuários de tal sistema.

O inconveniente desta abordagem é o decréscimo de desempenho, uma vez que processadores de propósito genérico são projetados para um conjunto amplo e genérico de operações. De modo geral, pode-se dizer que passam diversos ciclos de relógio em tarefas não relacionadas diretamente ao processamento de pacotes e, portanto, apenas uma fração dos ciclos de relógio será utilizada para efetivamente processá-los.

As instruções necessárias para manusear e tratar pacotes em tempo real, em operações de processamento de pacotes, requerem tratamento no nível da camada de bits; tais instruções devem ser concluídas em taxas nominais de transferência e não estão disponíveis como instruções padrão em processadores genéricos. Como processadores tenderão a utilizar mais de uma de suas instruções em programas que implementam as funcionalidades desejadas, estes programas serão executados ao longo de múltiplos ciclos de clock, dificultando o propósito da manutenção e conclusão de operações à taxas nominais de transferência.

Tipicamente um sistema de processador de rede implementado apenas como *hardware* (como ASIC, *Application Specific Integrated Circuits*) oferece melhor desempenho no tempo que um processador de rede implementado sobre processador de propósito geral para o processamento de pacotes de mesma demanda. Existem, no entanto, dois fatores negativos no projeto de ASICs para núcleos dedicados:

- ASICs sofrem de capacidade muito limitada (senão impossível) de programabilidade, o que os torna uma solução rígida em termos de funcionalidades. Quando novas funcionalidades forem necessárias, ou novos

protocolos devam ser suportados, o circuito deve ser re-projetado em parte ou totalmente.

- O tempo requerido para o projeto de um ASIC pode ser proibitivo, dependendo de diversos fatores como tamanho da equipe de projeto, ferramentas de automação de projetos eletrônicos disponíveis e restrições financeiras.

5.3 Microarquitetura de base

O trabalho de aplicação para simulação desta dissertação foi realizado sobre uma microarquitetura de processador de rede baseado em SoC, com atividade de roteador *core*, e foca com bastante ênfase a metodologia de reuso.

A microarquitetura de referência é composta pelos seguintes componentes: um processador RISC LEON com 7 estágios de pipeline, controlador de memória cache de dados e de instruções, memória ROM e SRAM on-chip, um periférico de rede Ethernet IEEE802.3 2002 de 100Mbps e um barramento AMBA, com AHB e APB (*advanced peripheral bus*). O barramento, hierárquico e segmentado, é composto de arbitrador, multiplexadores de escrita e de leitura, e decodificador de endereços, além de um conjunto de circuitos necessários para garantir as funcionalidades desejadas. A microarquitetura dos núcleos implementados para compor o processador de rede é ilustrada na Figura 5.2.

As seções a seguir descrevem a microarquitetura e os critérios para a escolha de cada um dos módulos principais, com o propósito de facilitar, mais tarde, o entendimento das modelagens RTL e analítica utilizadas.

5.3.1 Processador Leon

A microarquitetura RISC SPARC é de especificação completamente aberta e não proprietária o que reduz o custo de sua utilização (ainda que tenha custo de certificação de compatibilidade). Existem, atualmente, duas implementações em linguagem de descrição de *hardware* que estão disponibilizados para uso livre:

- LEON, um processador genérico de 32 bits, *single-threaded*, que segue a implementação SPARC V8 (SPARC, 1991). O código fonte está em VHDL e é licenciado sob o formato GPL (GNU, 1991). Faz parte de um microssistema completo, escrito em VHDL e sob o licenciamento GPL. Das versões existentes, adotamos o LEON3 (GAISLER, 2002) para este trabalho por ser a que oferece maior grau de integração entre os núcleos constituintes.

Figura 5.2: Diagrama de blocos representando a microarquitetura dos núcleos implementados no LEON3.

- OpenSPARC T1 (OPENSARC, 2006) é um processador genérico de 64 bits, implementado para suporte a até 32 *threads* e compatível com a especificação SPARC V9 (SPARC, 2005). O código fonte é escrito em Verilog, e licenciado sob diferentes formas. O código disponível não

contempla um microsistema completo, o que dificulta o uso deste processador dentro dos propósitos deste trabalho.

Pelo fato de ser compatível com a microarquitetura SPARC, toda a ampla gama de ferramentas de compilação e aplicativos de códigos abertos estão disponíveis para o processador LEON. Ele já oferece uma série de interfaces com componentes externos de entrada e saída, o que facilita a sua integração dentro de um SoC. O processador implementa uma sofisticada hierarquia de memória, que além de ser opcional, possui diversas características parametrizáveis da memória *cache*, como tamanho de acordo com o uso (de instruções e de dados), algoritmo de substituição, grau de associatividade, tamanho da linha, além de outras características gerais de memória, como *snooping* e MMU (*Memory Management Unit*).

5.3.2 Sistema de barramentos AMBA AHB e APB (especificação 2.0)

A comunicação dentro de um sistema refere-se ao trânsito de sinais entre os módulos que o compõem ou entre o sistema e o mundo externo (GARZON, 2003). A estrutura de comunicação pode ser um sistema hierárquico, caracterizado por barramentos principais (rápidos) e secundários (lentos), arbitradores e pontes entre os diferentes barramentos (CoreConnect da IBM (IBM, 1999), AMBA da ARM (ARM, 1999), etc.). Uma ponte é um componente que conecta dois barramentos, que podem ou não seguir o mesmo padrão (por exemplo, pode haver uma ponte entre AHB-AHB, AHB-APB ou AHB-CoreConnect OPB). Pontes podem conter um esquema de armazenamento interno para armazenar ou capturar parte dos dados que serão transferidos de um barramento ao outro.

As características de acesso a dados de um componente do microsistema define a categoria a qual este componente é enquadrado, podendo ser mestre ou escravo. O sentido do controle é sempre do mestre para o escravo (com exceção de transferências *split*, nas quais um escravo solicita acesso ao barramento para continuar uma transferência interrompida). Em determinados barramentos (tipicamente de periféricos, ou “lentos”) todos os membros são escravos, com exceção da ponte, que atua como mestre (entretanto, este trabalha como escravo na interface com o barramento “rápido”). A atribuição de mestre ou escravo pode

depender da microarquitetura, na medida em que o módulo pode ter diferentes interfaces de comunicação com o mesmo barramento ou barramentos diferentes; pode ser mestre em um e escravo em outro ou mestre e escravo no mesmo barramento (atribuídas a diferentes interfaces).

A ocorrência de colisões e contenção (contenção ocorre quando dois ou mais elementos necessitam usar o mesmo barramento simultaneamente) nos barramentos faz com que o desempenho do sistema diminua. Um núcleo arbitrador é usado quando existem múltiplos elementos mestres ligados ao barramento. Todas as solicitações de acesso (*requests*) devem ser encaminhadas ao arbitrador, que determina qual mestre tem assegurado o barramento (*grant*). Algoritmos de arbitragem devem ser adotados com base em fatores como microarquitetura, aplicação alvo e perfil da carga de dados transitando pelo sistema. Exemplos de algoritmos de arbitragem são:

- *Round-robin*. Neste tipo de arbitragem, a determinação do mestre ao qual o barramento será assegurado é feita de maneira seqüencial. Se um mestre não precisar do barramento, o módulo seguinte é qualificado, e assim por diante. Usa-se o termo “qualificado” pois um mestre pode não estar em posição de fazer qualquer transferência ainda que tenha o barramento assegurado.
- *Prioridade fixa (fixed priority)*. Neste tipo de arbitragem, o mestre é escolhido segundo a sua prioridade previamente atribuída e fixa, pelo projetista do sistema. Basicamente o arbitrador tem um vetor de solicitações, que é ordenado segundo a prioridade estabelecida. Este tipo de arbitragem pode incorrer no fenômeno de *starvation* porque pode-se sistematicamente e repetidamente ser negado a determinado mestre o seu acesso ao barramento.

Transferências em segmentação: são conjuntos de transferências intercaladas e executadas em vários ciclos de relógio e em diferentes estágios. Esta característica permite obter uma maior vazão nominal (ver definição na Seção 5.6.1), pois conta com a existência de circuitos adicionais que permitem, em um mesmo ciclo de relógio, a execução de diferentes operações. A quantidade e o tipo de operações simultâneas suportadas depende dos circuitos eletrônicos implementados. O barramento pode ser segmentado em dois estágios (por exemplo, acerto inicial de comunicação e a transferência de dados) ou múltiplos estágios (por exemplo *request*, *grant*, *initiate*, *acknowledge* seguidos pela transferência de dados).

O AMBA é o sistema de comunicação adotado na distribuição do microssistema LEON, cuja especificação é seguida pelo processador e o bloco Ethernet que acompanha o conjunto. O AMBA foi concebido como uma especificação aberta, bastante utilizado em projetos de SoCs, sendo que existe hoje implementações do barramento em HDLs. O AMBA possui 3 tipos diferentes de barramentos, o AHB, o APB e o ASB (*advanced system bus*), cada um deles orientado a um tipo de aplicação dentro do SoC, tendo um conjunto de propriedades parametrizáveis. O AMBA AHB é o mais indicado para suportar conexões eficientes e de maior desempenho entre processadores, memórias *on-chip*, memórias externas e alguns periféricos, enquanto o APB é recomendado para conexão de blocos periféricos que tenham menores exigências em termos de largura de banda e latência. O ASB é também um barramento de sistema, porém sem as características de alto desempenho do AHB.

A decisão de utilizar o AMBA (em conjunto com o LEON) foi feita após a análise de outras possibilidades de sistemas de barramento: o fato de o AMBA ser uma especificação aberta, cujos aspectos de licenciamento são pouco complexos, apresentar alto desempenho e baixa latência, além de ser muito bem documentado, fez com que ele apresentasse maior conformidade com os objetivos do trabalho.

O desempenho do microssistema fica dependente também das características do sistema de barramentos, com sensibilidade a mudança de políticas de arbitragem e ao número de interfaces mestres e escravos no barramento AHB.

Outro importante aspecto referente a esta microarquitetura é que com a adoção de um barramento com arbitragem não existe possibilidade de tentativas simultâneas de acesso de leitura ou escrita de dois módulos a um terceiro, fato este que traz maior confiabilidade em termos de funcionamento ao microssistema e facilidade de desenvolvimento de novos blocos de *hardware* ou mesmo de novos aplicativos.

O funcionamento do AMBA não é similar ao de um *crossbar-switch*. Isto porque no caso de operação de escrita, o arbitrador define qual dispositivo com interface mestre¹ ganha o barramento, mas a informação é transmitida a todas as interfaces escravos. Cada escravo tem uma lógica adicional que implementa uma máscara

1 Um dispositivo pode ter uma ou mais interfaces mestre e escravo. Sem perda de generalidade, neste trabalho, consideramos que cada dispositivo apresentada apenas uma interface mestre ou escravo, portanto, o próprio dispositivo passa a ser denominado mestre ou escravo, respectivamente.

para aceitar somente os dados enviados a um intervalo de endereços pertencente a ele. O mesmo acontece no sentido inverso, para operações de leitura.

O AMBA é um sistema de barramentos com *pipeline*, podendo realizar em certo instante até 3 operações diferentes, mas obrigatoriamente apenas para um tipo específico de operação (são possíveis 3 tipos de operações, sendo elas escrita ou leitura, decodificação e endereçamento, e controle). Isto porque o barramento é implementado com multiplexadores e cada tipo de operação tem um multiplexador associado (multiplexador de escrita, outro de leitura e outro de decodificação de endereços). O *fan-in* destes multiplexadores é igual ao número de mestres para os multiplexadores de escrita e igual ao número de escravos para os multiplexadores de leitura e de decodificação. O *fan-out* é igual ao número de escravos para os multiplexadores de escrita e igual ao número de mestres para o multiplexador de leitura ou de decodificação.

A partir do instante em que o mestre obtém permissão de acesso a um barramento, seja para escrita ou para leitura, a comunicação torna-se ponto-a-ponto; abstraindo-se os detalhes, é equivalente a um conjunto de trilhas interligando o mestre ao escravo (um conjunto para leitura e um conjunto para escrita). Não existe operação de mestre para mestre sem o intermédio de um escravo.

Em um determinado instante, apenas um mestre pode realizar a operação de escrita nos barramentos e apenas um mestre pode realizar operação de leitura nos barramentos. Isto significa que cada multiplexador de escrita ou de leitura está sendo usado por apenas um mestre. O mestre pode eventualmente ser o mesmo, mas para isto é necessária uma lógica específica no seu circuito. O mesmo pode ser dito a respeito do escravo, que, dependendo da lógica interna, pode realizar ao mesmo tempo operação de leitura e escrita. Pelas condições descritas acima, fica claro que existem restrições na comunicação entre elementos do barramento para operações de escrita/leitura. Um exemplo ilustrativo destas limitações, assim como dos casos que dependem da implementação, é apresentado na Tabela 5.1, onde considera-se um barramento com 4 elementos conectados: dois como mestre, mst1, mst2, e dois como escravo, scra, scrb.

Tabela 5.1: Operações simultâneas permitidas no AMBA AHB para diferentes mestres e escravos.

Operação	núm.	MST1	MST2	SCRA	SCRB	Validade
Escrita	1	e			e	s
	2	e		e		s
	3		e	e		s
	4		e		e	s
	5	e		e	e	d
	6		e	e	e	d
	7	e	e			n
	8			e	e	n
Leitura	9	l			l	s
	10	l		l		s
	11		l	l		s
	12		l		l	s
	13	l		l	l	d
	14		l	l	l	d
	15	l	l			n
	16			l	l	n

Legenda:

colunas - mestres (m) e escravos (s) onde a presença de uma letra mostra a operação e quais os elementos envolvidos.

e - operação de escrita entre elementos.

l - operação de leitura entre elementos.

s - sim, a operação é válida

n - não, a operação é não válida

d - pode ser possível mas depende da implementação.

Pode-se observar na Tabela 5.1, que existem 12 operações válidas, sendo que quatro delas, as operações 05, 06, 13 e 14, de escrita de um mestre para vários endereços ao mesmo tempo, dependem da implementação². Existem algumas operações simultâneas que, para que sejam contempladas, há necessidade de implementação especial. São elas:

- 01 e 09: depende da implementação de mst1 e scrb;
- 01 e 10: depende da implementação de mst1 e scra;
- 01 e 12: depende da implementação de scr;
- Idem para 02, 03 e 04.

São considerados como recursos disponíveis a este trabalho a especificação

² No nível do projeto em HDL, como o VHDL, por exemplo, isto envolve o uso de `std_Ulogic`, já que precisa haver um *don't care* (-) no endereço enviado ao mux de escrita ou ao de leitura, assim como a máscara de endereços nos escravos e mestres devem aceitar *don't cares* (-).

AMBA, que é um documento de alto nível de abstração, o código do AMBA implementado em VHDL no nível RTL e a carta de ondas, que é uma representação de sinais no tempo. A metodologia para simulação cobre diversos níveis, de alta a baixa abstração. Quanto melhor a documentação a respeito do sistema analisado, mais fácil é o seu entendimento e mais tranqüila será a sua modelagem. A documentação a respeito da especificação AMBA é boa o suficiente para garantir que seja possível realizar um vasto conjunto de simulações sem a necessidade de entendimento detalhado do código VHDL.

5.3.3 Memórias PROM/SRAM/SDRAM

O microssistema suporta 3 tipos diferentes de memória: ROM, SRAM e SDRAM. Como ilustrado na Figura 5.2, os dois blocos de memória *on-chip*, AHB RAM (que é uma SRAM) e AHB ROM, podem ser instanciados no microssistema, sendo acessados diretamente por mestres do barramento através do arbitrador do sistema, não necessitando de controladores de memória. Ainda que tenham capacidade reduzida (AHB RAM de no máximo 64KB), o tempo de acesso é menor que nas outras memórias acessadas por intermédio de controladores.

O microssistema também suporta outras memórias externas via um núcleo controlador de memória. Este atua como escravo no barramento AHB e é mais sofisticado que o funcionamento apresentado no parágrafo anterior. Como pode ser observado na Figura 5.2, tal núcleo controla as memórias PROM, SRAM e SDRAM, e é, por padrão, o escravo 0 do barramento AHB. Tem espaço de endereçamento de até 2GB e controla uma memória PROM (512MB), uma SDRAM (1GB), uma SRAM (512MB), e dispositivos do E/S mapeados por memória.

A PROM é inicializada com o *firmware* já no início da simulação através de arquivos SREC (S-RECORDS) (FREESCALE, 2001) previamente montados. Qualquer que seja a memória de dados utilizada, o espaço de endereçamento será dividido em 3 partes, uma para descritores de transmissão (*transmit descriptors TXD*), uma para descritores de recepção (*receive descriptors, RXD*) e outra para a carga útil dos pacotes IPv4.

5.3.4 Núcleo periférico Ethernet

Trata-se de um módulo baseado no padrão de especificação aberta IEEE 802.3 2002 (IEEE Std 802.3, 2002a) CSMA-CD (*Carrier sense multiple access with collision detection*) (camada MAC) que conta com ampla gama de aplicativos de código aberto disponíveis. A camada de enlace contempla a recepção e a transmissão de dados simultâneas e independentes, com seus geradores de CRC e FIFOs (*buffer first-in-first-out*) para armazenamento de quadros. A interface entre o módulo Ethernet (nas tarefas de recepção e transmissão) e o resto do microsistema é realizada através de um bloco controlador de acesso direto à memória (DMA, *direct memory access*), junto ao barramento AHB, como mostrado na Figura 5.3. O controlador de DMA lê os dados dos FIFOs, solicita ao arbitrador do barramento AHB a transmissão de dados entre o periférico Ethernet e a memória, e controla o estado das transferências. A máquina de estados do controlador DMA para transmissão pode solicitar ao arbitrador permissão para acessar dados na memória e escrevê-los no FIFO de transmissão. A leitura destes dados passa a ser incumbência do circuito de transmissão que faz interface com o transceptor. Por outro lado, a máquina de estados do DMA para recepção pode solicitar simultaneamente ao arbitrador permissão para escrever dados na memória, após lê-los do FIFO de recepção. A escrita de dados no FIFO de recepção é de incumbência do circuito de recepção que faz interface com o transceptor.

5.3.5 Aplicativo de *driver* e encaminhamento de pacotes IPv4 (*firmware*)

Para este trabalho foi desenvolvido um *driver* e *firmware* de encaminhamento de pacotes IPv4. No caso de recepção de pacotes, após estes terem sido escritos na memória, o processador lê dois conjuntos de dados, o *Receive Buffer Descriptor* (RXD) e a carga útil do quadro Ethernet (cabeçalho e dados do pacote IPv4). Com o conhecimento do endereço de destino do pacote, o aplicativo faz uma consulta a uma tabela de roteamento previamente conhecida e determina a qual núcleo periférico Ethernet o pacote deve ser enviado. Para isso o processador escreve na memória o *Transmit buffer Descriptor* (TXD) e o pacote Ethernet modificado. O

periférico Ethernet é responsável, através do controlador DMA, por ler o TXD e o conteúdo do pacote Ethernet da memória, através de seguidas solicitações de leitura ao controlador de arbitragem do AHB. A comunicação entre o processador e o periférico Ethernet é realizada não somente através de interrupções mas também através de escrita/leitura de registradores de controle e descritores, cada qual com diferentes permissões de escrita e leitura em trechos destes registradores.

Núcleo Ethernet

Figura 5.3: Diagrama de blocos do núcleo Ethernet

5.4 Modelo RTL do *soft-core*

Em trabalhos similares a este, a comprovação da eficácia de modelos analíticos tem sido realizada tipicamente através de comparações com simulações em outras camadas de abstração (CHAKRABORTY et al., 2003) (GRIES et al., 2003) . Têm sido considerados níveis de abstração como o comportamental

algorítmico ou o de instruções (quando envolvem processadores programáveis), e já foram realizadas simulações em sistemas escritos em System-C *bus cycle-accurate* (BCA) não sintetizáveis. Entretanto, não haviam sido realizadas simulações no nível RTL sintetizável, apesar de ser reconhecido que a validação de modelos analíticos contra simulações em níveis menos abstratos é interessante para reforçar o modelo, ainda que tenham tornado-se incomuns (SAKAMOTO et al., 2003).

A simulação RTL como forma de validação representa um grau de inovação em relação a trabalhos anteriores na mesma linha de pesquisa e, assim, melhorias podem ser propostas ao modelo analítico, baseadas nos diversos estudos e análises realizados sobre os resultados. Como fruto do confronto entre o modelo analítico proposto em trabalhos anteriores e simulações RTL, algumas inovações a tal modelo são também apresentadas nesta dissertação. Como desvantagem desta abordagem, pode-se citar o alto tempo de simulação necessário (1 segundo para o trânsito de 7000 pacotes tomou 58 horas de simulação). A hipótese de realizar simulações em níveis ainda mais baixos de abstração (e portanto maior detalhamento) como o temporizado (pós-síntese, que toma em conta o formato de propagação de sinais) tomaria significativamente mais tempo de simulação apesar de um aumento de precisão de dados, mas provavelmente sem ganhos significativos em termos de acurácia ou fidelidade dos prognósticos analíticos.

Dado que esta dissertação propõe a aferição de um modelo analítico baseada no seu modelo RTL correspondente, a implementação no nível de abstração RTL da microarquitetura apresentada na seção anterior passa a ser a referência de estudo prático nesta dissertação. Além disso, como toda a coleta de dados foi conduzida de modo não-intrusivo e o número de sinais monitorados foi da ordem de uma centena (com múltiplos bits de largura), a coleta de sinais em uma implementação FPGA (*Field programmable gate array*), por exemplo, seria inviável.

5.4.1 A distribuição Leon

Nesta dissertação, uma implementação da microarquitetura da Seção 1.3 é adotada como referência para a aferição da modelagem analítica. Trata-se de uma

distribuição do processador LEON (GAISLER, 2002) e seus periféricos descritos em VHDL sintetizável, em um ambiente testado e certificado, podendo ser alterado para outros fins além ser programável em FPGA. A distribuição tem por base o conceito de uso de padrões e especificações abertas (SPARC e AMBA), assim com manter abertas as implementações. Algumas ferramentas disponibilizadas (GR, 2007) para simulação no grau de instrução são gratuitas, porém apresentam restrições – não permitem a simulação do microssistema todo, com o barramento e periféricos de entrada e saída. Simulam apenas o núcleo do processador com o controlador de cache e, parcialmente, a hierarquia de memória. Para se ter acesso às funcionalidades completas, deve-se licenciar a ferramenta.

O LEON corresponde, portanto, a um microssistema completo que possui módulos dedicados a processamento (microprocessador RISC SPARC, núcleo Ethernet, núcleo controlador DMA, núcleo controlador de memória, núcleo controlador de interrupções), estrutura de comunicação de dados entre circuitos (barramento AMBA AHB e APB, ligações de trilhas ponto-a-ponto) e blocos de armazenamento de dados (memória cache, memória SRAM, memória SDRAM). Como todos os módulos do microssistema foram projetados para estarem implantados em um único circuito integrado, pode-se dizer que trata-se então de um sistema sobre silício ou SoC.

A opção pela distribuição do LEON traz como vantagens o fato de poder-se trabalhar com um sistema extremamente complexo (note-se que são ao todo 70 mil linhas de código VHDL) já construído e testado, com ótimo nível de confiabilidade, tornando praticamente inexistente o aparecimento de *bugs*. O microssistema, tendo sido concebido inicialmente como sistema tolerante a falhas, segue o fluxo de projeto da Agência Espacial Européia (ESA, 1994).

5.4.2 A implementação

5.4.2.1 Hardware

A implementação adotada do microsistema, no que se refere às categorizações apresentadas na Seção 5.2.4, é pertencente ao modelo de processador de rede baseado em processador programável de propósito geral (a microarquitetura SPARC). O *core* do processador possui 7 estágios de pipeline, unidades aritméticas de multiplicação e divisão, de números inteiros e de ponto flutuante.

O Núcleo Ethernet é um módulo periférico baseado no padrão aberto IEEE 802.3 2002 CSMA-CD (*Carrier sense multiple access com collision detection*) (IEEE Std 802.3, 2002a) implementando a camada MAC, portanto sendo denominado de núcleo MAC. A camada MAC é implementada através de dois circuitos independentes, MAC_RX e MAC_TX, cada um com um conjunto de máquinas de estado, geradores de CRC e FIFO para armazenamento. A interface entre o MAC_RX (também MAC_TX) e a memória é implementada através de um controlador de acesso direto à memória (DMA), junto ao barramento AHB, que impõe o algoritmo de arbitragem para determinar qual dos circuitos (MAC_RX ou MAC_TX) terá acesso ao barramento em caso de conflitos.

O periférico é modelado através de 6 processos VHDL que são executados em paralelo, o que garante a funcionalidade *full duplex*. Os FIFOs de RX e de TX são adotados para armazenamento dos pacotes, o que é muito conveniente, pois a sua utilização corresponde ao conceito de ocupação de *buffer* no modelo analítico. Note-se que é possível configurar o código RTL para que os FIFOs assumam diversos tamanhos na simulação.

Um controlador de DMA lê os dados dos FIFOs, solicita ao arbitrador do barramento AHB a transmissão de dados entre o periférico Ethernet e a memória, e controla o estado das transmissões. O periférico Ethernet utiliza também uma interface com o barramento APB no estabelecimento de compartilhamento de registradores de controle e de estados do periférico (GRETH_REGS), como pode ser visto na Figura 5.2. Estes registradores são dispostos em 7 endereços consecutivos de memória e após a leitura pelo periférico, são utilizados na recepção

ou transmissão de pacotes. O conteúdo destes registradores pode ser acessado e alterado parte pelo periférico, parte pelo *driver* que roda no processador. Com esta inicialização efetuada, as máquinas de estados para transmissão e recepção do DMA realizam as suas tarefas da forma apresentada na Seção 5.3.4.

Outros registradores importantes para o funcionamento do sistema e para a comunicação entre os *driver* e *firmware* de roteamento e o periférico são os RXD e TXD. Eles têm 64 *bits* de comprimento cada e possuem até 7 bits de controle e de estado da transferência. Outros *bits* correspondem ao endereço de memória onde o datagrama IPv4 deve ser armazenado, assim com a extensão deste datagrama em bytes. Assim, de maneira similar aos registradores de controle e estado do periférico (GRETH_REGS), os RXD e TXD estabelecem a interface entre o *firmware* e o periférico.

De todos os estados possíveis para cada uma das máquinas de estados presente no circuito, nem todos os estados serão avaliados em tempo de simulação, isto porque um dos objetivos do trabalho é realizar um estudo baseado em análise determinística de pior e melhor casos, o que é alcançado, tomado como critério, com a maior incidência possível de dados válidos. Os casos onde poderiam haver colisões de pacotes, *timeout*, queda de pacotes por quaisquer motivos, não são contemplados no estudo por se tratarem de situações causadas pelo comportamento da rede e não do microsistema.

Sendo assim, para a máquina de estados do circuito de TX, os seguintes estados serão avaliados: *idle*, *preamble*, *sfd* (delimitador de início de quadro), *data1*, *data2*, *fcs*, *finish*, *idle*. Para a máquina de estados do circuito de RX, os seguintes estados serão avaliados: *idle*, *wait_sfd* (delimitador de início de quadro), *data1*, *data2*, *check_crc*, *report_status*, *wait_rep*, *idle*.

A camada física (PHY) é representada através de um módulo de transmissão e recepção (transceptor) escrito em VHDL não sintetizável, e que simula um circuito analógico. Este módulo lê dados previamente escritos em um arquivo onde estão armazenados cada um dos *nibbles* (4 bits) de quadros Ethernet. A interface entre o enlace de dados (MAC) e a camada física (PHY) é MII (*Media Independent Interface*), estabelecida pelo padrão IEEE 802.3.

As tarefas de montagem e funcionamento das baterias de testes do sistema esbarraram nas dificuldades de depurar o sistema em VHDL, pela ausência de documentação detalhada a respeito dos núcleos implementados em *hardware*. A isto

soma-se a ausência de *software* de simulador do periférico de comunicação e barramento, o que acarreta, em ampla medida, passos de tentativa e erro, com atrasos de projeto pelo fato de a simulação RTL ser lenta.

5.4.2.2 *Software*

O modelo implementado inclui, para as operações de tratamento de pacotes IPv4, um *firmware* que implementa um *driver* e o algoritmo de encaminhamento armazena-e-encaminha (desenvolvido neste trabalho). No caso de recepção de pacotes, após estes terem sido escritos na memória, o processador realiza a leitura dos dois conjuntos de dados, o *Receive buffer Descriptor* (RXD) e a carga útil do pacote IP (cabeçalho e dados). Com o conhecimento do endereço de destino do pacote, o aplicativo faz uma consulta a uma tabela de roteamento previamente conhecida e determina a qual núcleo periférico Ethernet o pacote deve ser enviado. Para isso o processador escreve na memória o *Transmit buffer Descriptor* (TXD) e o pacote Ethernet modificado.

Apesar de existir ampla documentação referente a desenvolvimento de *software*, muitas dificuldades tiveram que ser enfrentadas. Exemplo de desafio vencido foi o caso do desenvolvimento tanto do *driver* como o *firmware*, que apresentou enormes dificuldades pela ausência de um modelo de alto nível de barramento e núcleo periférico Ethernet que pudesse facilitar a depuração, simulação e validação do código de programa. De modo geral, a ausência de um simulador de todo o sistema (independente do grau de abstração) é um grande empecilho para o desenvolvimento do *software* e depuração do conjunto hardware-*software* de sistemas completos.

5.5 Aplicativo para construção de quadros Ethernet (desenvolvido neste trabalho)

O modelo RTL do microssistema cumpre duas finalidades no estudo e desenvolvimento do correspondente modelo analítico a ser apresentado na próxima

seção: a) por ser mais preciso, serve de base na comparação de resultados de métricas de desempenho; b) é usado na obtenção de alguns parâmetros do modelo analítico. Para ambas as funções, deve-se gerar de modo consistente os estímulos das baterias de testes (*testbenches*), sendo isto feito a partir de diversos rastros de dados no formato TSH (*Time Sequenced Header*) (IETF, 1989), oriundos do monitoramento de roteadores IPv4. Os roteadores de pacotes geralmente têm a capacidade de coletar os pacotes que trafegam por eles e formatar as informações correspondentes no formato TSH, apresentado a seguir.

```
# 0          1          2          3
# 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
# +-+-+-+-+-+-+-+-+
#0 |          timestamp (seconds)          |Time
# +-+-+-+-+-+-+-+-+
#1 |  i/f#   | timestamp (microseconds)    |
# +-+-+-+-+-+-+-+-+
#2 |Version| IHL |Type of Service|      Total Length      | IP
# +-+-+-+-+-+-+-+-+
#3 |  Identification  |Flags|  Fragment Offset  |
# +-+-+-+-+-+-+-+-+
#4 | Time to Live |  Protocol  |      Header Checksum   |
# +-+-+-+-+-+-+-+-+
#5 |          Source Address          |
# +-+-+-+-+-+-+-+-+
#6 |          Destination Address     |
# +-+-+-+-+-+-+-+-+
#7 |  Source Port    |  Destination Port    | TCP
# +-+-+-+-+-+-+-+-+
#8 |          Sequence Number          |
# +-+-+-+-+-+-+-+-+
#9 |          Acknowledgment Number    |
# +-+-+-+-+-+-+-+-+
# | Data |          |U|A|P|R|S|F|          |
#10 | Offset| Reserved |R|C|S|S|Y||      Window      |
# |   |          |G|K|H|T|N|N|          |
# +-+-+-+-+-+-+-+-+
#
```

De todas as informações presentes (44 bytes), apenas parte delas é útil dentro

deste trabalho, ainda que todas elas sejam extraídas do rastro e posteriormente usadas na construção do quadro. São elas:

- **Timestamp:** indicação, em micro-segundos, do instante de chegada do pacote no roteador, que segundo a referência (IETF, 1989), deve ser escrito no instante de chegada. Este valor é importante pois ao longo de 1 segundo milhares de pacotes podem chegar. No utilitário de geração de quadros, este valor será usado para determinar o intervalo entre quadros válidos (*Interframe gap*, IFG);
- **Interface:** indica a interface física do roteador pela qual o pacote entrou;
- **Versão:** IPv4, constante;
- **Comprimento do cabeçalho do pacote Internet (IHL):** constante e igual a 20 *bytes*, pois todos os cabeçalhos são de 5 linhas;
- **Comprimento total do pacote** (incluindo cabeçalho): variável;
- **Tempo de Vida (TTL):** sempre diminuído de 1 pelo aplicativo de roteamento;
- **Checksum do cabeçalho:** preenchido com zeros pelo roteador;
- **Endereço de origem:** não utilizado, pois é indiferente ao roteamento;
- **Endereço de destino:** será usado para gerar a consulta à tabela de roteamento;
- **Cabeçalho TCP** (camada de transporte): dados descartados.

É importante lembrar que os campos IP *header checksum*, endereço de origem, e endereço de destino do datagrama IP são alterados na criação do arquivo TSH e o *payload* é eliminado, por questões de privacidade e segurança. Com os dados acima coletados, o *quadro* MAC, utilizado como entrada nas simulações com o núcleo Ethernet, é construído. Adicionam-se o preâmbulo e o delimitador de início de quadro (SFD), como indicado na Figura 5.4.

O *National Laboratory for Applied Network Research*, NLANR, disponibiliza diversos perfis diferentes de rastros coletados para análises (NLANR, 2005). Dentre eles, os de maior interesse são:

- Comunicação por video-conferência entre dois pontos separados por até 16 roteadores. Trata-se de uma hora de gravação no formato H.323 e H.263;
- Tráfego entre roteadores do *backbone* norte-americano;

A Figura 5.5 apresenta os passos da metodologia no uso do utilitário de construção de quadros MAC. O utilitário *tcpdump.pl* é um conversor dos rastros do formato TSH para o formato texto simples, com cada linha representando um datagrama IP diferente. A título de informação, um arquivo *tcpdump* pode conter

Figura 5.5: Fluxo de dados e aplicativos para geração de estímulos de entrada baseados em dados de mundo real. mais de 1 milhão de linhas.

O utilitário *gera_mac_frame.c*, escrito em linguagem C tem como entrada um arquivo do tipo *tcpdump*. Tem a incumbência calcular o CRC para cada um dos quadros MAC, e já que o número de quadros MAC é alto, este cálculo deve ser realizado com grande eficiência. Após construídos os quadros, estes são convertidos para os formatos apropriados e a saída é formatada em *nibbles* (metade de um *byte*) pois a interface MII (IEEE Std 802.3, 2002a) estabelece que os sinais de dados são de 4 *bits*. O conteúdo de dados dos datagramas IP (*payload*) é montado com dados fictícios, mas de modo a facilitar a depuração no momento da simulação RTL. A saída do utilitário é um arquivo *indata* que trata-se de uma coluna de 4 *bits* de largura e é o estímulo de entrada do sinal *rxdata* do módulo *transceptor*.

5.6 Modelo analítico

5.6.1 Latência e vazão associados aos elementos do microssistema

No capítulo 3 os conceitos de latência e taxas de execução e de requisição de um sistema foram definidos de forma genérica; nesta seção, o conceito de vazão, fortemente relacionado às taxas acima, é explorado pois, assim como a latência, é

uma métrica de intensa utilização no restante da dissertação. Tanto vazão como latência são definidos de forma particularizada para diversas situações e componentes do microssistema de tal forma a não haver diferentes interpretações para um mesmo termo.

Latência: é o intervalo de tempo percebido entre o instante em que determinado fenômeno é iniciado e o instante em que o efeito do fenômeno começa. Durante este intervalo, tal fenômeno pode ser descrito como em um estado latente (ver Seção 3.6 para formalização da definição).

Latência de acesso ao barramento (latência de contenção): é o intervalo de tempo transcorrido entre o instante em que um sinal de controle de um membro do barramento é imposto, solicitando acesso ao barramento (para operação de leitura ou escrita), e o instante em que o primeiro *bit* de dados sai do porto de origem deste membro solicitante. É decorrência da contenção no barramento.

Latência de transferência de dados (leitura ou escrita) no barramento: é o intervalo de tempo transcorrido entre o instante em que um *bit* de dados entra no barramento (o sinal do barramento recebe o valor do bit imposto no porto de saída do membro solicitante) e o instante em que o primeiro *bit* de dados chega ao porto do membro de destino.

Latência de operação de comunicação: é o intervalo de tempo transcorrido entre o instante em que um sinal de controle é imposto e o instante em que o primeiro *bit* de dados chega ao porto de destino. É o resultado entre a soma da latência de acesso ao barramento e a latência de transferência (leitura ou escrita) no barramento.

Latência de operação: é o intervalo de tempo transcorrido entre o início de uma operação, para o primeiro conjunto de bits a chegar nos portos de entrada e o primeiro conjunto de bits a sair do porto de saída.

Vazão: é a quantidade de dados que é transferida através de um determinado recurso (de processamento, armazenamento ou estrutura de comunicação), dividida pelo intervalo de tempo transcorrido entre a chegada do primeiro conjunto de bits e a saída do último conjunto de bits.

Vazão nominal: é o valor teórico dado pelo produto entre a frequência de operação do núcleo e a largura desta estrutura. É conceitualmente o valor máximo que pode ser transferido entre dois portos de uma estrutura a cada intervalo de tempo.

Vazão agregada: é a soma das vazões nominais de diferentes recursos (ou núcleos).

Vazão sustentada: é a quantidade de dados que é efetivamente transferida através de um determinado núcleo, em um intervalo de tempo grande o suficiente para que seja incluída a influência de sobrecarga de protocolo, latência de operação ou conflitos decorrentes de compartilhamento de recursos.

Para ilustrar os conceitos acima, podemos usar como exemplo um bloco Ethernet padrão IEEE 802.3, cuja vazão nominal é de 100Mbps *full-duplex*. Este valor decorre dos 25Mhz de frequência nominal multiplicado por 4 *bits* nos portos de RX ou TX. Entretanto, tal vazão não se sustenta na prática, uma vez que o padrão especifica um intervalo mínimo de tempo que deve ser aguardado para um porto poder enviar um novo pacote após o envio do último *nibble* (4 bits) do quadro imediatamente anterior, IFG. Por exemplo, no caso de uma taxa de transmissão de 100Mbps, o IFG é especificado em 960ns e portanto equivalente a 12 *bytes*. Como o tamanho máximo do quadro é (geralmente) estipulado em 1538 *bytes* (1500 *bytes* de carga útil + 12 *bytes* do IFG + 8 *bytes* do preâmbulo+ 14 *bytes* do cabeçalho + 4 *bytes* de *trailer*), uma utilização máxima do canal corresponde a $(1538 - 12)/1538 \times 100\% = 99,2\%$ em uma conexão Ethernet 100Mbps, com uma vazão sustentada de 99.2 Mbps. No caso de um fluxo formado por quadros de 64 *bytes* (o menor permitido pela especificação com carga útil é de 38 bytes), a utilização máxima do canal é de $(64 - 12)/64 \times 100\% = 81,25\%$, portanto com uma vazão sustentada de 81,25 Mbps em uma conexão Ethernet 100Mbps.

O exemplo acima mostra também que a vazão sustentada poderia ser calculada como 100Mbps, caso a amostragem fosse realizada em um intervalo de tempo mais curto que um pacote completo. Portanto o conceito de vazão nominal ou mesmo sustentada podem ser erroneamente utilizados na análise de desempenho de sistemas eletrônicos, já que pode-se eventualmente não ser dada a devida atenção à sobrecarga intrínseca de protocolos. A vazão nominal raramente (ou nunca) é atingida, ao ponto que a vazão sustentada deve ser estudada seguindo critérios mais específicos.

5.6.2 Modelagem analítica: problemas, metodologia e passos

Tendo sido apresentados o equacionamento algébrico e a modelagem analítica através de equações, curvas e grafos, resta solucionar o problema de como determinar os parâmetros das curvas de entrada e saída do modelo sem a existência de qualquer modelo mais detalhado (comportamental ou nível mais baixo) ou, no caso que se tenha a implementação do microssistema, como extraí-los.

Como fontes de ajuda para coletar dados e parâmetros, pode-se mencionar as especificações do protocolo a ser modelado, a experiência do especialista de modelagem, o conhecimento prévio dos perfis de comportamento de carga de entrada, a capacidade de entendimento do protocolo e a percepção de como os seus detalhes serão implementados em termos de algoritmos ou em pseudo-códigos. Com o conhecimento prévio de alguns detalhes de módulos (núcleos) reutilizáveis, como vazão e latência específicas, o especialista em modelagem deve visualizar, posteriormente, como os pseudo-códigos devem ser encaixados nestas especificações.

São quatro os passos principais da metodologia apresentada nesta dissertação para a implementação do modelo analítico:

1. Geração do grafo de tarefas;
2. Geração do grafo de recursos-tarefas;
3. Levantamento das curvas de serviço e entrada;
4. Obtenção de resultados de $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$.

5.6.2.1 Passo 01 – Construção do grafo de tarefas (GT)

O Passo 01 consiste em construir o grafo de tarefas, conforme definido na Seção 3.7.1. O problema se resume a determinar os vértices (as tarefas) e as arestas direcionadas (a seqüência entre os vértices, indicando o fluxo de transferência de dados). Enumera-se as tarefas a serem executadas, baseando-se nas especificações, protocolos, código fonte, ou qualquer outra fonte que deixe claro o aspecto funcional sendo implantado.

A seqüência de tarefas reflete a ordem de execução nos recursos. Tal seqüência pode ser modelada apenas com o conhecimento das especificações textuais do protocolo (linguagem natural, escrita ou verbal) ou fluxograma, diagrama, código fonte.

Nesta fase não há qualquer especificidade a respeito de conhecimento de métricas estruturais como as de tempo ou derivados (vazão, utilização) em relação às tarefas. Quanto maior, já nesta fase, o conhecimento detalhado das funcionalidades do protocolo a ser implementado, mais precisa a modelagem será, auxiliando as fases seguintes de projeto.

5.6.2.2 Passo 02 – Construção do grafo de recursos-tarefas expandido (GRTE)

O Passo 02 consiste em construir o grafo de recursos-tarefas (GRT) ou o grafo de recursos-tarefas expandido (GRTE), conforme definidos nas Seções 4.7 e 4.8, respectivamente. Tendo sido definidos os propósitos e especificações desejadas do sistema e, também, a microarquitetura e implementações possíveis para se atingir as especificações, neste ponto deve-se realizar a alocação de recursos computacionais.

A fase de construção do GRT (usaremos aqui o termo GRT na referência a ambos os grafos, expandido ou não) consiste em definir a associação dos vértices do grafo de tarefas aos elementos de *hardware* candidatos a pertencerem à microarquitetura. Haverá uma relação direta entre o grafo de tarefas e recursos que poderão executá-las (mapeamento). Tarefas podem ser realizadas por diferentes recursos e vice-versa. O resultado da etapa de mapeamento é o conjunto de grafos de recursos-tarefas que associam unicamente tarefas a recursos, ficando formado, então, após esta etapa, um espaço de avaliação de projetos. Em (ETHZ, 2005) é dado um exemplo de processador de redes de 25 tarefas e 8 recursos com espaço de avaliação de projetos com até $2 \cdot 10^{16}$ possibilidades.

Nesta dissertação, o foco da pesquisa e a avaliação do modelo analítico frente à uma implementação de alta precisão (RTL), além de algumas extensões no modelo analítico, enquanto o mapeamento não foi objeto de estudo. A relação entre recursos e tarefas é 1 para 1, ou seja, não há uma tarefa que pudesse ser realizada em mais de um recurso. Para acessar um estudo mais aprofundado sobre

mapeamento utilizando NetCal, pode-se consultar (THIELE et al., 2002b).

5.6.2.3 Passo 03 – Levantamento das curvas de chegada e de serviço

Como visto no Capítulo 4, um GRT é modelado por suas tarefas, seqüências e recursos, e para cada par recurso-tarefa existem as curvas de chegada e de serviço ingressantes, a partir das quais são computadas as remanescentes (de saída). Desta forma, a rigor, para se preceder a computação de um modelo analítico baseado em GRT, deve-se ter as curvas de chegada ingressantes em cada fluxo do sistema e as curvas de serviço iniciais de cada recurso (do topo de cada coluna do GRT).

Conforme já mencionado na Seção 3.5.1, o TSPEC é uma das formas mais convenientes para a modelagem de curvas de chegada, como abstração de fluxos de pacotes ingressantes. Tal especificação foi proposta para simbolizar o tráfego de pacotes e os aspectos de tempo associados. A partir desta proposta, foi estabelecida uma relação mais detalhada entre curvas de chegada e tráfego em redes de serviços diferenciados (DiffServs) (IETF, 1998) porque apresentou meios de levantar as curvas a partir de parâmetros de fácil obtenção.

O TSPEC é uma especificação simbólica com uma taxa de pico $[p]$ e um tamanho máximo do pacote $[M]$. A especificação simbólica inclui uma taxa média ou simbólica $[r]$ e um comprimento do pacote $[b]$, que devem ser positivos. A taxa simbólica $[r]$ é medida em *bytes* de datagramas IP por segundo. O comprimento do pacote, $[b]$, é medido em *bytes*.

A taxa de tráfego de pico $[p]$ é medida também em *bytes* de datagramas IP por segundo. Não se espera que um particular elemento de rede possa suportar toda a amplitude de valores. Este tamanho inclui dados da aplicação e todos os cabeçalhos do protocolo no ou acima do nível do IP (IP, TCP, UDP, RTP, etc.). Não inclui o tamanho do cabeçalho no enlace de dados, porque estes cabeçalhos mudarão em tamanho à medida que o pacote cruza parcelas diferentes da rede interna.

Todos os datagramas IP menores do que o tamanho $[M]$ são tratados como sendo do tamanho M para finalidades de alocação e provisionamento de recursos. A finalidade deste parâmetro é permitir uma estimativa razoável dos recursos

necessitados por pacote para processar pacotes de um fluxo (taxa máxima do pacote pode ser computada dos parâmetros $[b]$ e $[M]$) e para limitar razoavelmente a banda de rede consumida pelos cabeçalhos do enlace de rede. O tamanho máximo do pacote, $[M]$, é o maior pacote que se conformará à especificação do tráfego, sendo medido também em *bytes*.

As curvas de serviço iniciais associadas aos vértices de topo (primeiras tarefas) do GRT devem ser obtidas com dados de latência e vazão dos recursos. São duas as fontes: uma de origem teórica e outra, prática. Como exemplos de dados teóricos pode-se citar as exigências de especificações em termos de restrições de tempo ou especificações de sistemas já implementados e disponibilizadas em bibliotecas. Do lado prático, dados oriundos de simulações contribuem para popular as curvas de serviço, ainda que isto exija a disponibilidade prévia da implementação do módulo ou núcleo. Basicamente, dois parâmetros devem ser extraídos do recurso: a latência q (de operação ou de operação de comunicação) e a vazão s (nominal ou sustentada). Algebricamente a curva de serviço é descrita no tempo como a equação $f(t) = s \cdot (t - q)$, com $f(t) = 0$ para $t < q$.

5.6.2.4 Passo 04 - Obtenção de $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$

De posse do GRT ou GRTE e das curvas de chegada e serviço, deve-se aplicar as equações 4.3 a 4.6 e 4.10 a 4.15 e para o cômputo do par de trios $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$ para cada um dos vértices. Com estes valores é possível também determinar a latência fim-a-fim para uma determinada microarquitetura candidata.

5.7 Modelo analítico aplicado à implementação do microsistema

5.7.1 Modelagem da carga de entrada

O problema se resume a obter os parâmetros b , p , M e r descritos na Seção 5.6.2.3, que devem ser obtidos da seguinte forma:

- o tamanho do maior pacote M , assim como o tamanho do pacote médio b , deve ser extraído diretamente dos rastros;
- a taxa média r deve ser o resultado da divisão da quantidade total de *bytes* transferidos pelo intervalo de tempo coletado;
- a taxa de pico p deve ser o maior salto nos rastros, como na função requisição acumulada $R(t)$.

5.7.2 Modelagem dos recursos no GRT – Aspectos de simultaneidade, preemptividade e prioridade.

A microarquitetura sob estudo, assim como a carga de trabalho à qual está exposta, está modelada no GRT apresentado na Figura 5.6, contendo quatro recursos: o processador, o módulo ETH, incluindo os circuitos MAC_RX, MAC_TX e DMA, o arbitrador AHB e a memória. A definição dos pares recurso-tarefa na modelagem do GRT segue o procedimento mostrado na Seção 4.7 enquanto nesta seção será feita uma discussão sobre o significado e o alcance do modelo analítico com tais definições. O detalhamento das tarefas será deixado para a seção seguinte, onde ficará evidente a maneira como os parâmetros associados, tanto ao GRT como ao GRTE, são definidos.

O problema da disposição de vértices e recursos é um campo aberto para pesquisas, do mesmo modo que a alocação e associação entre vértices e recursos. Também, ainda é um assunto não definido qual o melhor ponto de detalhamento e granularidade de recursos e tarefas no grafo. A maior granularidade (e portanto

maior número de vértices no GRT) contribui para o estudo detalhado do comportamento dinâmico da microarquitetura, mas ainda são necessárias mais pesquisas sobre o tema para avaliar a extensão das contribuições desta abordagem.

Figura 5.6: GRT representativos da microarquitetura sob simulação

Diversos grafos de recursos-tarefas podem resultar das decisões tomadas, não ficando clara a repercussão destas escolhas no comportamento quantitativo dos modelos adotados. Pode-se, porém, afirmar que a disposição dos vértices na grade do GRT tem impacto direto nos resultados do trio (L-B-U) dado o grau de dependência entre as equações 4.3 a 4.6. No presente trabalho, o par recurso-tarefa foi definido como aquele que contém todas as operações independentes de outros recursos como, por exemplo, o vértice RTX que representa a consolidação de todas as operações deste circuito para transmissão, e que interage diretamente com o barramento somente quando deve escrever dados na memória.

Sabe-se que, para um mesmo recurso, a curva de serviço remanescente do vértice superior alimenta a curva de serviço do vértice inferior, representando um comportamento preemptivo na relação entre as diversas tarefas desempenhadas por um mesmo recurso. É o caso do vértice IRQ do GRT da Figura 5.6, relacionado ao recurso PROC, que, por ser o primeiro vértice do recurso, tem-lhe disponibilizada toda a capacidade de processamento do recurso. O vértice imediatamente inferior (RTX) tem disponibilizada para si a capacidade remanescente do recurso (representada pela curva de serviço remanescente do seu vértice superior).

No tempo, pelo fato de processadores serem, de modo típico, segmentados, a

modelagem analítica consolida a ocupação do sistema. O exemplo a seguir ilustra a diferença entre a representação analítica e a física.

Exemplo 5.1: Na representação da execução no tempo de duas tarefas por um processador genérico programável segmentado, suponhamos que duas tarefas estejam sendo executadas neste processador, sem prioridade definida entre elas, conforme ilustrado na Figura 5.7. As tarefas “a” e “b” compartilham simultaneamente o mesmo circuito.

Por ter maior prioridade no modelo analítico (ocorre primeiro no fluxo do grafo de tarefas), a tarefa A apresenta menor latência de execução que no caso físico. Do mesmo modo, a tarefa B apresenta maior latência de execução quando comparada ao seu caso físico. Portanto a simultaneidade de tarefas em andamento em um recurso é um importante aspecto da modelagem analítica.

Figura 5.7: Exemplo de modelagem de duas tarefas simultâneas executadas por um circuito segmentado. (a) Modelo real no tempo; (b) Modelo analítico no domínio de Δ .

A memória é um exemplo de outro aspecto relacionado à simultaneidade; a capacidade de realizar operações de escrita e leitura simultâneas depende da quantidade de portas disponíveis para isso. Em um exemplo de memória de porto duplo, capaz de comportar simultaneamente um par de operações escrita-leitura independentes, a modelagem é diferente daquela exposta anteriormente para o processador. No caso exposto na Figura 5.8.a, como o grau de simultaneidade

possível é menor que o número total de vértices na coluna do recurso, a capacidade disponibilizada ao vértice “c” é aquela remanescente do vértice “b”. No entanto, como o recurso não é capaz de tratar mais que duas tarefas simultâneas, o vértice “c” termina recebendo uma curva de serviços com menor capacidade de processamento que teria no caso real. Portanto, o vértice “c” seria melhor representado como recebendo também uma curva de serviços inicial, pois o recurso completou as tarefas representadas em “a” e “b”, como mostrado na Figura 5.8.b .

Figura 5.8: Definição de alocação e vértices de recursos

5.7.3 Modelagem dos recursos no GRTE

Como ficou registrado na seção anterior, a modelagem detalhada dos recursos do GRT e as suas tarefas, apresentadas na Figura 5.6, é realizada nesta seção. Evita-se assim a redundância no tratamento do assunto, uma vez que o GRTE, descrito na Figura 4.4, é uma expansão do GRT e os parâmetros utilizados no primeiro correspondem ao do último.

Seguindo a característica dos processadores de rede de ter espectro limitado de aplicações, a microarquitetura implementada contém um processador programável de propósito genérico com uma aplicação de encaminhamento IPv4, descrita na Figura 3.8, englobando rotinas de interrupção para atendimento de DMA e tarefas de encaminhamento no envio e recepção de pacotes.

A RAM local (*on-chip* e rápida) é modelada como um espaço de endereçamento infinito. Somente depois que o GRTE (ou GRT) é percorrido, o *buffer* total realmente usado é conhecido. Estas memórias têm latências diferentes de leitura e escrita (*wait states*) e aceitam chamadas de leitura/escrita de DMA, armazena descritores, cabeçalhos, carga útil dos pacotes e *firmware* (que não é modelado como ocupante de espaço em memória).

O arbitrador/decodificador do barramento é modelado como latência e vazão. Há quatro chamadas ao arbitrador no GRT/GRTE: pedidos de escrita em memória feitos pelo controlador de DMA, do processador para leitura de memória, do processador para escrita na memória e do controlador de DMA para leitura de memória. Todas as operações têm latência dependente da política de arbitragem do barramento aplicada, da relação entre mestres e escravos e da natureza da operação. Enquanto DMAs são responsáveis pela leitura e escrita dos pacotes IPv4 completos e os descritores de *buffer*, os processadores realizam a leitura e escrita do descritor de *buffer* e o cabeçalho IPv4, que representam somente uma fração das transferências de DMA. O barramento é segmentado e tem uma latência inicial e uma taxa de transferência da largura de faixa multiplicadas pela frequência do barramento.

5.7.3.1 Recepção de quadros

O GRTE do microssistema é ilustrado na Figura 4.4 e, diferente do caso do GRT, o protocolo Ethernet é distribuído nos blocos Eth1 e Eth2, correspondendo aos circuitos MAC_RX e MAC_TX adicionados ao circuito controlador DMA (que é compartilhado pelos dois), tendo ciclos de processamento e taxa de *throughput* para o pior caso.

No GRTE a tarefa de recepção é modelada no módulos Eth1 como o vértice (RX)@Eth1 (na transmissão, o vértice é (TX)@Eth2 em Eth2). É conveniente lembrar que no GRT (Figura 5.6), é modelado um único recurso Eth, onde ocorrem tanto RX e TX. As tarefas realizadas são seriais, com os dados percorrendo os módulos por diferentes estágios. Os quadros do enlace de dados (camada 2 na pilha OSI (ISO, 1994) são montados a partir de *nibbles*, após o qual têm o preâmbulo e o delimitador retirados e passam pelo circuito de checagem do CRC e da

conformidade do endereço físico (MAC). Além disso, há o envio de sinal de conhecimento (ACK) pelo porto específico e o enfileiramento do quadro no *buffer* de recepção (FIFO_RX). A partir deste estado, o controlador de DMA passa à leitura do *buffer* para a escrita do cabeçalho do quadro e pacote IPv4 na memória do microssistema.

A latência do circuito Eth1 é constante e independente do comprimento do pacote (obtidas por resultados de simulação); do mesmo modo, a vazão nominal destes circuitos é constante e independente do comprimento do pacote. Tanto o MAC_RX quanto o MAC_TX têm o propósito de transferir dados à taxa de transferência nominal de 100Mbps.

5.7.3.2 Escrita na memória

A modelagem de escrita em memória é detalhada na Figura 4.4 (página 63) pelos vértices (eth_rx,w)@Mem1 e (proc_rtx,w)@Mem3. É conveniente relembrar que no GRT (Figura 5.6), é modelado um único recurso Mem, onde ocorrem as tarefas de escrita. Em (eth_rx,w)@Mem1, o módulo de processamento Eth1 tem dados a escrever na memória e solicita ao arbitrador (eth_rx,w)@AHB1 uma operação de escrita na memória e aguarda a concessão do acesso ao barramento (a memória é escravo do barramento). Após decorrida a latência de arbitragem, o acesso é concedido ao controlador de DMA e daí, as palavras são transmitidas pelas interconexões aos portos de memória para escrita, sendo computada a latência de transferência de dados. Novamente, deve-se salientar que no GRT, um único recurso de arbitragem, AHB, é modelado.

O processador (assim como o controlador de DMA, é mestre do barramento) realiza a escrita na memória de modo similar, com suas operações representadas pelos vértices (proc_rtx,w)@AHB3 e (proc_rtx,w)@Mem3, como apresentadas na Figura 4.4.

Aqui é relevante explicar o significado destes vértices no GRT/GRTE. O processo de escrita é dependente de um sinal de interrupção extra-banda do núcleo periférico Ethernet, Eth, para o processador Leon. O Eth tem um pacote IPv4 recém-recebido e um descritor do *buffer* de recepção em seu *buffer* de saída, pronto para ser enviado à memória através do controlador de DMA. O DMA apresenta latência

de arbitragem uma vez que os diferentes circuitos de recepção e transmissão (MAC_RX e MAC_TX) estarão concorrendo na solicitação de leitura e escrita à memória através de sinais de interrupção. No nosso estudo, há somente um par MAC_RX e MAC_TX para cada controlador de DMA. Estas solicitações serão concedidas seguindo uma política implementada no controlador de DMA. Observe-se, entretanto, que a latência de operação de comunicação compreende não somente o tempo necessário para aplicar esta política, mas também o tempo necessário para garantir o acesso da concessão depois de uma solicitação. Isto depende de muitos fatores extras à atividade específica de DMA, como conflitos de recurso e políticas diferentes de arbitragem.

O valor da latência de arbitragem é dependente da política de arbitragem estabelecida no barramento. Após o pedido do controlador de DMA para o acesso ao barramento ser concedido pelo arbitrador, ocorre o fluxo de dados entre os portos. De acordo com a política do arbitrador, a latência é constante (toma-se o pior caso a partir dos rastros) e é dependente de muitos fatores alheios ao núcleo do arbitrador propriamente como, por exemplo, o tráfego entre diferentes núcleos, tamanhos do pacote IP e conflitos por recursos. O tempo gasto pelo arbitrador para processar a política e a concessão do acesso é da ordem de 1 ciclo. A quantidade de informação mínima e máxima transmitidas são representadas pelas curvas de entrada inferiores e superiores remanescentes do vértice (RX)@Eth1, que equivale ao tempo requerido pelo barramento para transferir eficazmente os dados, e por uma inclinação que é função da largura e da frequência do barramento. O fato de as latências mencionadas acima não serem dependentes somente do bloco propriamente, mas mais intensamente de fatores externos a eles, faz com que a análise sobre elas e a simulação com dados reais sejam extremamente importantes.

A terceira latência representada nesta operação de escrita é o número de ciclos necessário para a memória mover dados do porto de entrada às células de armazenamento, que é uma contagem de ciclos pequena, dependente da implementação (número de *wait states*), pois o barramento é segmentado. Esta latência é contabilizada somente uma vez por operação. Depois que os dados são escritos na memória um sinal de interrupção viaja ao processador advertindo-o que a operação de recepção foi finalizada e, portanto, alguns dados estão prontos para serem lidos.

5.7.3.3 Leitura da memória

A modelagem de leitura de memória é detalhada na Figura 4.4 através dos vértices (proc_rtx, r)@Mem2 e (eth_tx, r)@Mem4. Após o processador atualizar o cabeçalho IPv4 e o descritor do *buffer* de transmissão (TXD), ou logo após a carga útil do pacote e o descritor do *buffer* de recepção (RXD) serem escritos na memória, um sinal de controle é emitido do processador ao Eth, ou de interrupção do Eth ao processador, o que começa uma operação de transmissão ou leitura da memória, respectivamente.

Primeiramente, o processador solicita operação de leitura em endereços específicos e espera a concessão para continuar a operação de leitura representada no vértice (proc_rtx,r)@AHB2. Após transcorrida a latência de arbitragem, o acesso é concedido e são necessários até 2 ciclos (dado pela especificação *read wait states*) para fazer os dados requeridos disponíveis no porto da memória, aos quais adiciona-se uma latência de transferência de dados do porto da memória ao porto do controlador do *cache* do processador, através das interconexões do barramento (proc_rtx)@Mem2.

De modo semelhante, o periférico Ethernet, Eth_TX, através do controlador DMA, solicita ao arbitrador uma operação de leitura nos endereços especificados pelo TXD e aguarda a concessão para completar a operação de leitura, representada no vértice (eth_tx,r)@AHB4. Após transcorrida a latência de comunicação, a transferência tem início, representada no vértice (eth_tx,r)@Mem4.

5.7.3.4 Processamento de TX (Transmissão)

Após o reconhecimento do sinal de interrupção enviado pelo núcleo Eth indicando a finalização do recebimento de um pacote, a tarefa (RTX)@Proc2 passa a executar. Formalmente, esta tarefa está implementada em uma rotina de *firmware* (route.c). É conveniente lembrar que no GRT (Figura 5.6), é modelado como um único recurso Proc, onde ocorrem o processamento de endereço e o tratamento de interrupção (a ser tratado na próxima subseção). Na tarefa de processamento da transmissão, o descritor do *buffer* de recebimento e o campo específico do cabeçalho IPv4 (endereço IP de destino) é lido e sobre estes valores é feita uma

busca binária na tabela de roteamento para determinar qual é o porto de saída do núcleo Ethernet para onde o pacote deve ser encaminhado. Na atual implementação, o resultado desta busca é sempre “0” pois há somente um núcleo Ethernet implementado, ainda assim, a busca é realizada como se existissem mais portos. A tabela de roteamento foi construída baseando-se nos endereços dos rastros coletados, e suas linhas foram dispostas de modo a facilitar a busca binária.

O campo de tempo de vida, TTL (ver Seção 5.5), do cabeçalho do pacote é decrementado em uma unidade e o cabeçalho em questão é alterado. O campo *checksum* do cabeçalho do pacote é checado com o algoritmo *checksum* (IETF, 1990) e novamente o cabeçalho do pacote é alterado. Um dos registradores do núcleo Ethernet é alterado para receber o endereço do ponteiro do descritor de *buffer* de transmissão para que se saiba o endereço de leitura. A latência de operação de comunicação desta passagem não é constante.

Pode-se notar que o núcleo periférico Ethernet (Figura 5.2) está conectado tanto ao barramento AHB como ao APB. Como os registradores de controle do Eth são acessíveis pelo APB, para o processador alterá-los, este deve solicitar ao arbitrador uma operação de escrita na ponte AHB-APB (escrava no AHB e mestra no APB). Após concedido o barramento e finalizada a operação (latência de operação de comunicação) inicia-se uma nova operação nos domínios do APB, onde a ponte realiza a escrita no registrador do núcleo Eth (após nova latência de operação de comunicação), quando então o registrador, no ETH, é alterado.

O descritor do *buffer* de transmissão é criado a partir dos dados recebidos e, então, armazenado em fila para ser utilizado no momento da transmissão deste referido pacote. O tratamento desta fila segue o modelo FIFO.

5.7.3.5 Processamento de interrupções (IRQHANDLER)

A sub-rotina de interrupções IRQHANDLER do *firmware* é executada sempre que o núcleo Eth envia um sinal de interrupção ao processador e é modelada no vértice (IRQ)@Proc1. A latência desta operação não é determinística porque, como no caso do processamento de RTX, alguns registradores de controle no núcleo Eth devem ser alterados, e para isto surgem latências de operação de comunicação no

AHB e no APB. Tais latências são dependentes, de modo genérico, do tráfego em ambos os barramentos. A sub-rotina de tratamento de interrupção ocupa aproximadamente 210 ciclos de processador para finalizar, em uma situação de baixíssimo grau de conflito nos dois barramentos.

Após ocorrer o reconhecimento da interrupção o descritor do *buffer* de recebimento é montado pelo IRQHANDLER. O campo com o comprimento do pacote é lido e com esta informação o endereço do ponteiro para o pacote é alocado. Novamente os registradores de endereço (GRETHREGS) são alterados para armazenar dados do ponteiro e para alterar o estado da máquina de estado do circuito de RX e autorizar novos recebimentos (RX). O registrador do controlador e a máscara de interrupções é esvaziado a fim de autorizar novas interrupções.

5.7.3.6 Transferência em barramento

Toda operação (de escrita ou leitura) no barramento é controlada pelo arbitrador. Elas são representadas na Figura 4.4 pelos vértices (eth_rx,w)@AHB1, (proc_rtx,w)@AHB3, (proc_rtx,r)@AHB2, (eth_tx,r)@AHB4. Todos os núcleos (mestres e escravos) conectados ao barramento são implementados contendo uma interface que, nativamente, interpreta os estados no núcleo e altera os seus estados para tratar os sinais de comunicação com o barramento. No instante em que dados devem ser transferidos do mestre para o escravo, a interface no mestre impõe sinais de solicitação de barramento, junto a sinais descrevendo a operação (escrita ou leitura, comprimento dos dados, endereço de destino, tipo de rajada, conjunto de dados iniciais). Antes de a transferência ocorrer, o arbitrador deve impor ao barramento um sinal determinando a que mestre (ou escravo, em caso de transferência com *split*) o barramento está assegurado; a partir deste instante é iniciada a transferência. Para o arbitrador determinar a que núcleo deve ser assegurado o barramento, ele deve ter implementada uma política de arbitragem, que é de grande relevância na dinâmica de transferências uma vez que as latências de comunicação dependem dela (LAHIRI; RAGHUNATHAN; LAKSHMINARAYANA, 2001) (CONTI et al., 2004). Na presente implementação, tanto a política por prioridade fixa e *round robin* estão presentes .

5.7.3.7 Transmissão de quadros

A operação de transmissão funciona de modo equivalente à recepção de quadros.

5.7.4 Levantamento quantitativo de curvas de serviço

Nesta seção, um exemplo de levantamento quantitativo de curva de serviço para a implementação do processador de rede é apresentado. As tarefas dentro dos recursos são detalhadas e expostas nos seus vários passos.

5.7.4.1 Vértice (RX)@Eth no GRT ou (RX)@Eth1 no GRTE

- tratamento do preâmbulo do quadro MAC: 14 ciclos Ethernet;
- tratamento do delimitador: 2 ciclos Ethernet;
- comparação do endereço MAC (endereço físico) de destino com o próprio: 12 ciclos Ethernet (endereço físico de fonte não importa);
- comprimento total do pacote IPv4: 4 ciclos Ethernet;
- processamento de dados: uma vez que o quadro é aceito, inicia-se a escrita do conteúdo do pacote IPv4 no FIFO_RX e o cálculo do CRC: 2 ciclos. Esta etapa depende dos comprimentos dos pacotes de entrada. Os pacotes IPv4 válidos variam de 64 bytes a 1518 bytes. Como são recebidos na forma de *nibbles*, levam de 128 a 3036 ciclos Ethernet.

Parâmetros das curvas: Dos dados anteriores, temos que, entre a chegada do primeiro *nibble* do preâmbulo e o término da escrita da última palavra no FIFO_RX, usa-se: de 160 a 3068 ciclos Ethernet. Computando-se o ciclo Ethernet como 40ns (vide exemplo da Seção 5.6.1), a latência de operação é de 34ns e a vazão sustentada máxima é 1518 bytes a cada 3068 ciclos, totalizando $12,4 \cdot 10^{-3}$ bytes/ns, enquanto a vazão sustentada mínima é de 64 bytes a cada 160 ciclos, resultando em $10,0 \cdot 10^{-3}$ bytes/ns. Os parâmetros que determinam os segmentos da

curva de serviço inferior é $(64, 10,0.10^{-3}(t-34))$ e os da curva de serviço superior é $(1518, 12,4.10^{-3}(t-34))$.

5.7.4.2 Vértice (eth_rx,w)@AHB ou (eth_rx,w)@AHB1

- o controlador de DMA captura dados no *buffer* FIFO_RX e já sabe qual o endereço na memória onde armazenar o dados.
- barramento solicitado pelo núcleo controlador de DMA para escrita na memória: 1 ciclo AHB;
- após uma latência de arbitragem de 1 ciclo o barramento é garantido: a latência de escrita em memória é, no melhor caso, 3 ciclos AHB. No pior caso, de prioridade fixa, processador escreve na memória: 3 ciclos AHB;
- vazão nominal do barramento dada por $50\text{Mhz} \times 32\text{bits} = 32\text{bits} / 20\text{ ns} = 2.10^{-1}\text{ bytes/ns}$.

Parâmetros das curvas: as curvas de serviço inferior e superior podem ser obtidas com os dados acima. A latência de arbitragem é de 0 ciclo AHB, enquanto as vazões sustentada mínima e máxima são de $2.10^{-1}\text{ bytes/ns}$.

5.7.4.3 Vértice (eth_rx,w)@Mem ou (eth_rx,w)@Mem1

- a escrita leva (comprimento do pacote IPv4[bytes] / 4 [bytes]) ciclos pois é realizada em palavras de 32 bits: 64 ciclos AHB
- dados escritos na memória e interrupção enviada: 2 ciclos AHB;

Parâmetros das curvas: a latência de operação de escrita é de 3 ciclos AHB e a vazão nominal mínima ou máxima de $32\text{bits} / 20\text{ ns} = 2.10^{-1}\text{ bytes/ns}$.

5.7.4.4 Vértice (IRQ)@Proc ou (IRQ)@Proc1

- após a recepção de todos os bytes do pacote, a interrupção de RX é gerada. O barramento é todo garantido à operação de recepção no caso de baixíssima disputa. As latências ($\delta_{\text{max_round robin}}$ e $\delta_{\text{max_prioridade fixa}}$) passam a ser

diferentes de zero quando houver mais disputa.

- interrupção recebida pelo *firmware*: 1 ciclo de processador;
- a sub-rotina de interrupção IRQHANDLER é executada, sendo gerado novo descritor do *buffer* de recepção para novo pacote a ser recebido. Escreve-se em três novos registradores (GRETH_REGS) do periférico e faz-se três solicitações de escrita pelo processador: total de 300 ciclos do processador (detalhes de operações para levantamento da curva de serviços relativos à subrotina IRQHANDLER podem ser vistas no APÊNDICE B);

5.7.4.5 Vértice (Proc_rtx,r)@AHB ou (Proc_rtx,r)@AHB2

- após 72 ciclos, o processador solicita leitura no endereço do descritor do *buffer* anterior;
- faz-se a solicitação de acesso ao barramento: 1 ciclo AHB;
- o barramento é garantido: 0 ciclo AHB;

5.7.4.6 Vértice (Proc_rtx,r)@Mem ou (Proc_rtx,r)@Mem2

- Faz-se a leitura de cinco linhas do cabeçalho IP; cada linha contém 32bits (5 ciclos de barramento)

5.7.4.7 Vértice (RTX)@Proc ou (RTX)@Proc2

- a sub-rotina de transmissão TX é executada, com o processamento do endereço e porta para envio de dados. Escreve-se em registrador (GRETHREGS_ADDR) do periférico Ethernet para indicar a este a habilitação de transmissão: total de 100 ciclos do processador (detalhes de operações para levantamento da curva de serviços relativos à subrotina RTX podem ser vistas no APÊNDICE B);

5.7.4.8 Sequência de tarefas com vértice (proc_rtx,w)@AHB ou (proc_rtx,w)@AHB3, e vértice (proc_rtx,w)@Mem ou (proc_rtx,w)@Mem3

- processador solicita barramento para escrita na memória: 1 ciclo do processador;
- o barramento é concedido para escrita após 1 a 40 ciclos do processador;
- após 5 ciclos de AHB (5 linhas de cabeçalho do pacote) os dados estão escritos na SRAM;
- processador solicita barramento para escrita na memória : 1 ciclo do processador;
- o barramento é concedido para escrita: 1 ciclo do processador;
- após 2 ciclos de AHB (64 bits do descritor) os dados são escritos;
- processador solicita barramento para escrita nos GRETH_REGS: 1 ciclo do processador;
- o barramento é concedido: 1 ciclo do processador;
- o núcleo Ethernet está ciente que deve buscar um pacote na memória. O controlador DMA muda estado para leitura no endereço apontado pelo TXD: 3 ciclos do processador;
- Estado *check_desc*, verifica conformidade de TXD: 5 ciclos do processador;

5.7.4.9 Vértice (eth_tx,r)@AHB ou (eth_tx,r)@AHB4

- o controlador de DMA já sabe qual o endereço na memória onde ler os dados e os transfere ao FIFO_TX.
- barramento solicitado pelo núcleo controlador de DMA para leitura na memória: 1 ciclo AHB;
- após latência de arbitragem de 1 ciclo o barramento é garantido; a latência de leitura em memória é 1 ciclo AHB.

- vazão nominal do barramento dada por $50\text{Mhz} \times 32\text{bits} = 32\text{bits} / 20\text{ ns} = 2 \cdot 10^{-1} \text{ bytes/ns}$.

Parâmetros das curvas: as curvas de serviço inferior e superior podem ser obtidas com os dados acima. A latência de arbitragem é de 1 ciclo AHB, enquanto as vazões sustentadas é de $2 \cdot 10^{-1} \text{ bytes/ns}$.

5.7.4.10 Vértice (eth_tx,r)@Mem ou (eth_tx,r)@Mem4

- a leitura leva (comprimento do pacote IPv4[bytes] / 4 [bytes]) ciclos pois é realizada em palavras de 32 bits: 64 ciclos AHB
- dados lidos da memória: 2 ciclos AHB;
- (desta vez lê na memória o cabeçalho e o payload, que varia de 64 bytes a 1518 bytes; em paralelo, o controlador de DMA passa os dados para calc_crc (VHDL), adiciona o preâmbulo, delimitador e tamanho total do mac_, sendo então escrito no TX FIFO: 14 ciclos do processador;

Parâmetros das curvas: a latência específica de leitura é de 1 ciclo AHB e a vazão nominal mínima ou máxima de $32\text{bits} / 20\text{ ns} = 2 \cdot 10^{-1} \text{ bytes/ns}$.

5.7.4.11 Vértice (TX)@Eth ou (eth_tx,r)@Eth2

- lê dados do FIFO_TX e inicia a transmissão no formato de nibbles: 160 ciclos do processador;

Capítulo 6 - Ambiente de simulação e discussão de resultados

Neste Capítulo são apresentados os detalhes da metodologia empregada, com seus diferentes passos para obtenção dos parâmetros construtivos das curvas de chegada e serviço. Os utilitários e ferramentas usados são apresentados, assim como os aspectos dos rastros e os parâmetros construtivos do microssistema. Os resultados e prognósticos são então apresentados, interpretados e discutidos.

6.1 Metodologia de avaliação de desempenho

Com o propósito de determinar o par de trios $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$ a seguinte metodologia de avaliação de desempenho foi elaborada e seguida. Na Figura 6.1 estão ilustrados as etapas percorridas, os produtos de saídas e o propósito de cada um dos quadros mostrados.

6.1.1 Metodologia Analítica (ANL)

Tendo sido construídas as curvas de entrada e serviço iniciais (representado nos quadros “Determinação de parâmetros” e “Levantamento de curvas de chegada e serviço” da Figura 6.1), conforme descrito na Seção 5.6.2 e montado o GRT e GRTE, conforme descrito na Seção 5.7, o conjunto de equações de NetCal (equações 4.3 a 4.6 e 4.10 a 4.15) é calculado para cada um dos vértices do grafo (representado no quadro “NetCal”). Tais equações foram implementadas no utilitário NetCal.c, que resultam nos valores de $(L-B-U)_{\text{mínimo}}$ e $(L-B-U)_{\text{máximo}}$.

6.1.2 Nível de transferência entre registradores (RTL)

Figura 6.1: Fluxo da metodologia de análise para obtenção de fidelidade entre RTL e modelo analítico. Note que a metodologia é realimentada

Na Figura 6.1 o quadro *Rastros* representa os quadros Ethernet que alimentam as simulações *cycle-accurate* dos *testbenches*. Foi usada a categoria de rastros pseudo-sintética, com os rastros coletados originariamente a partir de (NLNR, 2007) e modificados com o propósito de gerar mínimos níveis de conflitos por recursos e conflitos de dados. Os dados coletados de (NLNR, 2007) estão em formato binário conforme o padrão de Cabeçalho de Tempo Sequenciado (*TSH* –

Time Sequenced Headers) (BRADEN, 1989). O utilitário “tsh2tcpdump.pl” (escrito na linguagem Perl) converte os rastros do formato TSH para arquivos texto e extrai parâmetros de cada um dos pacotes IPv4 constituintes e como produto devolve um arquivo em formato similar ao “tcpdump”. O utilitário “gera_mac_frame.c” coleta os rastros já no formato texto e monta os quadros Ethernet em um arquivo contendo múltiplas linhas de 4 bits (*nibbles*). O *nibble* formado é compatível com o padrão de Interface Independente de Meio (*MII – Media Independent Interface*) IEEE 802.3 2002 (IEEE Std 802.3, 2002a). Por se tratarem de rastros pseudo-sintéticos, o comprimento do pacote é estipulado previamente, mas outros dados de cabeçalho são obtidos de rastros de caso real (NLANR, 2007). Dados de cabeçalho do quadro Ethernet (Figura 5.4) apontam os dados de caso real. O campo conteúdo do pacote TCP é alterado na origem e não é publicamente disponível. A partir da informação de porta do roteador, endereços IPv4 de origem e de destino contido nos rastros, o utilitário “gera_tabela_roteamento.pl” monta a tabela de roteamento que será populada no *firmware*.

O campo dados do pacote IPv4 é preenchido pelo utilitário “gera_mac_frame.c” com dados que seguem um seqüência conhecida e que facilitam a verificação nas simulações RTL. A partir da análise dos rastros são definidos os parâmetros pertinentes às curvas de entrada, como a vazão sustentada no porto de entrada e os tamanhos máximos e médio dos pacotes. O utilitário “gera_mac_frame.c” gera como saída o arquivo “indata”, com o preâmbulo, o delimitador, endereço MAC de destino, endereço MAC de origem, o intervalo entre quadros, o número de interface, o protocolo, as 5 linhas de cabeçalho IPv4, a primeira linha do cabeçalho TCP, o comprimento da carga útil (em bytes), o comprimento do quadro Ethernet (em bytes), a contagem do número de linhas de 32 bits, além de calcular o valor do CRC do pacote IPv4 embutido no quadro recém construído. Durante a simulação, o circuito PHY.VHD, que emula o transceptor Ethernet, lê cada linha do arquivo “indata”.

O quadro “*firmware*” da Figura 6.1 representa o aplicativo “route.c” executado no processador Leon ao longo da simulação. O detalhamento deste *firmware* foi apresentado na seção 5.4.2.2. O código fonte é compilado com o compilador de plataforma cruzada “sparc_gcc”, e o código executável produto da compilação é convertido ao formato “SREC” e populado na memória SRAM da microssistema no tempo de compilação do *testbench* pela ferramenta Modelsim (MENTOR, 2007) .

Na Tabela 6.1 abaixo são apresentados os parâmetros alterados nos *testbenches*, sendo estes basicamente 3: o comprimento do pacote (que pode ser fixo ou variável para os rastros de caso real), o comprimento do FIFO de RX, que é igual ao de TX e a política de arbitragem. O comprimento do pacote foi determinado de modo a poder simular pacotes curtos (256 bytes) a longos (1024 e 1025 bytes). Com os quatro comprimentos dos pacotes pode-se analisar com segurança o efeito deste parâmetro no comportamento dinâmico do sistema. O FIFO pode assumir diferentes valores, bastando alterar o código VHDL. Foi escolhido um valor mínimo (4 palavras de 32 bits) e um valor médio (32 palavras) e um valor alto (64 palavras), sendo que 64 palavras é o maior comprimento que a atual implementação do *soft-core* suporta. Novamente, esta variação permite detectar alterações diversas no comportamento dinâmico do sistema. A política de arbitragem permaneceu inalterada nos casos de baixa disputa pelo barramento pois, dada esta situação, não há alteração nas respostas por influência deste parâmetro.

Tabela 6.1: Parâmetros utilizados para os diferentes testbenches.

<i>Testbench</i>	Comprimento do Pacote (bytes)	FIFO (FIFO RX = FIFO TX)	Política de Arbitragem	Tempo de simulação (<i>cpu time</i>)	Comentários
tb_0001	256	32	PRIO	58h24'	Baixo nível de disputa
tb_0002	512	32	PRIO	55h40'	Baixo nível de disputa
tb_0003	1024	32	PRIO	54h59'	Baixo nível de disputa
tb_0004	1025	32	PRIO	60h20'	Baixo nível de disputa
tb_0005	256	4	PRIO	63h59'	Baixo nível de disputa
tb_0006	512	4	PRIO	51h40'	Baixo nível de disputa
tb_0007	1024	4	PRIO	58h45'	Baixo nível de disputa
tb_0008	1025	4	PRIO	62h32'	Baixo nível de disputa
tb_0009	256	64	PRIO	31h14'	Baixo nível de disputa
tb_0010	512	64	PRIO	45h14'	Baixo nível de disputa
tb_0011	1024	64	PRIO	41h56'	Baixo nível de disputa
tb_0012	1025	64	PRIO	45h39'	Baixo nível de disputa

PRIO = Prioridade fixa (maior prioridade ao processador, menor prioridade o controlador Ethernet)

O arquivo binário de saída do *testbench* possui o formato WLF (*wave log format*) (MODELSIM, 2005), de especificação proprietária e fechada. O quadro “Extração e coleta de sinais” da Figura 6.1 representa o conjunto de roteiros Tcl/Tk executados no interpretador do Modelsim que extraem os dados pertinentes (utilitário “coleta_instantes.do”) do conjunto de sinais para arquivos-texto com dados tabulado (formato LST).

No utilitário “coleta_instantes.do” é exposto o completo detalhamento que relaciona os valores de utilização, latência e *buffer* (apresentados na Tabela 6.2) com os sinais do *soft-core*, extraídos na tarefa representada no quadro como “Pré-processamento dos *testbenches*”. Os arquivos LST são extensos (dezenas de

Tabela 6.2: Resultados de todos os testbenches RTL

		eth			ahb					mem					proc				
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc		
tb_01	Util MAX	0,2342	0,2400	0,4742	0,0010	0,0199	0,0370	0,0194	0,0200	0,0584	0,0005	0,0090	0,0179	0,0172	0,0446	0,0407	0,0257	0,0663	
	contagem				28486	252163	55988	34994	49029										
	LAT MAX [ns]	6091,0	3632,5	9723,5	0	860	0	0	0	860	60	60	60	60	6000	5900	11720		
	LAT MIN [ns]	5951,0	3272,0	9223,0	0	20	0	0	0	20	20	20	20	20	5720	3280	9200		
	buffer MAX [bytes]	128	128	128	4	4	4	4	4	4	288	288	288	288	288				
	buffer MIN [bytes]	128	128	128	4	4	4	4	4	4	288	288	288	288	288				
tb_02		eth			ahb					mem					proc				
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc		
	Util MAX	0,3635	0,3817	0,7452	0,0008	0,0162	0,0481	0,0158	0,0305	0,0798	0,0004	0,0073	0,0270	0,0264	0,0610	0,0407	0,0257	0,0663	
	contagem				28430	252154	83981	34995	77091										
	LAT MAX [ns]	6091,0	3632,5	9723,5	0	860	0	0	0	860	60	60	60	60	6000	5860	11760		
	LAT MIN [ns]	5951,0	3272,0	9223,0	0	20	0	0	0	20	20	20	20	20	5720	3200	9120		
tb_03	buffer MAX [bytes]	128	128	128	4	4	4	4	4	4	544	544	544	544	544				
	buffer MIN [bytes]	128	128	128	4	4	4	4	4	4	544	544	544	544	544				
	tb_03		eth			ahb					mem					proc			
			eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
		Util MAX	0,3726	0,5497	0,9223	0,0006	0,0118	0,0612	0,0115	0,0429	0,1050	0,0003	0,0053	0,0377	0,0373	0,0805	0,0386	0,0243	0,0623
		contagem				26987	238951	132653	33164	126020									
LAT MAX [ns]		6091,0	3632,0	9723,0	0	860	0	0	0	860	60	60	60	60	6000	5820	11800		
LAT MIN [ns]		5951,0	3272,0	9223,0	0	160	0	0	0	160	20	20	20	20	5720	3540	9260		
tb_04	buffer MAX [bytes]	128	128	128	4	4	4	4	4	4	1056	1056	1056	1056	1056				
	buffer MIN [bytes]	128	128	128	4	4	4	4	4	4	1056	1056	1056	1056	1056				
	tb_04		eth			ahb					mem					proc			
			eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
		Util MAX	0,4215	0,4341	0,8556	0,0005	0,0093	0,0476	0,0090	0,0332	0,0815	0,0002	0,0042	0,0299	0,0294	0,0638	0,0304	0,0193	0,0497
		contagem				21850	188560	109894	26164	99420									
LAT MAX [ns]		6091,0	3472,5	9563,5	0	660	0	0	0	660	60	60	60	60	6000	5860	11760		
LAT MIN [ns]		5951,0	3272,0	9223,0	0	160	0	0	0	160	20	20	20	20	5720	3540	9260		
tb_05	buffer MAX [bytes]	128	128	128	4	4	4	4	4	4	1057	1057	1057	1057	1057				
	buffer MIN [bytes]	128	128	128	4	4	4	4	4	4	1057	1057	1057	1057	1057				
	tb_05		eth			ahb					mem					proc			
			eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
		Util MAX	0,2342	0,2400	0,4742	0,0010	0,0199	0,0872	0,0162	0,0399	0,1317	0,0005	0,0090	0,0242	0,0237	0,0573	0,0407	0,0236	0,0643
		contagem				28429	252076	251932	34994	251964									
LAT MAX [ns]		1611,0	3192,5	4803,5	0	380	0	0	0	380	60	60	60	60	6000	5660	11520		
LAT MIN [ns]		1471,0	2992,0	4463,0	0	20	0	0	0	20	20	20	20	20	5720	3260	8980		
tb_06	buffer MAX [bytes]	16	16	16	4	4	4	4	4	4	288	288	288	288	288				
	buffer MIN [bytes]	16	16	16	4	4	4	4	4	4	288	288	288	288	288				
	tb_06		eth			ahb					mem					proc			
			eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
		Util MAX	0,3635	0,3817	0,7452	0,0008	0,0162	0,1297	0,0132	0,0624	0,1959	0,0004	0,0073	0,0372	0,0368	0,0816	0,0407	0,0236	0,0643
		contagem				28329	252065	475886	34995	475996									
LAT MAX [ns]		1611,0	3192,0	4803,0	0	380	0	0	0	380	60	60	60	60	6000	5300	11160		
LAT MIN [ns]		1471,0	2992,0	4463,0	0	20	0	0	0	20	20	20	20	20	5720	3260	8980		
	buffer MAX [bytes]	16	16	16	4	4	4	4	4	4	544	544	544	544	544				
	buffer MIN [bytes]	16	16	16	4	4	4	4	4	4	544	544	544	544	544				

milhões de linhas) e apontam basicamente três informações: o nome do sinal, os instantes de pré e pós chaveamento e o nível lógico nestes instantes. Os roteiros

tb_07		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,3762	0,5494	0,9256	0,0006	0,0118	0,1845	0,0098	0,0965	0,2836	0,0003	0,0053	0,0516	0,0490	0,1061	0,0386	0,0223	0,0600
tb_07	contagem				26876	23883	4875001	33164	4873688									
	LAT MAX [ns]	1831,0	3192,5	5023,5	0	380	0	0	0	380	60	60	60	60	60	6240	5260	11260
	LAT MIN [ns]	1471,0	2992,0	4463,0	0	80	0	0	0	80	20	20	20	20	20	5720	3260	8980
	buffer MAX [bytes]	16	16	16	4	4	4	4	4	4	1056	1056	1056	1056	1056			
	buffer MIN [bytes]	16	16	16	4	4	4	4	4	4	1056	1056	1056	1056	1056			
tb_08		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,4215	0,4341	0,8556	0,0004	0,0093	0,1422	0,0076	0,0701	0,2146	0,0002	0,0042	0,0417	0,0412	0,0873	0,0304	0,0178	0,0480
	contagem				21731	18843	7696002	26164	690694									
	LAT MAX [ns]	1611,0	3172,5	4783,5	0	380	0	0	0	380	60	60	60	60	60	6760	5300	11220
	LAT MIN [ns]	1471,0	2992,0	4463,0	0	20	0	0	0	20	20	20	20	20	20	5720	3260	8980
	buffer MAX [bytes]	16	16	16	4	4	4	4	4	4	1057	1057	1057	1057	1057			
buffer MIN [bytes]	16	16	16	4	4	4	4	4	4	1057	1057	1057	1057	1057				
tb_09		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,2342	0,2400	0,4742	0,0010	0,0199	0,0335	0,0232	0,0185	0,0498	0,0005	0,0090	0,0175	0,0168	0,0437	0,0651	0,0450	0,1100
	contagem				28264	250075	41645	34704	34700									
	LAT MAX [ns]	11211,0	0,0	11211,0	0	1180	0	0	0	1180	60	60	60	60	60	6000	6480	12360
	LAT MIN [ns]	11071,0	0100000,0	111071,0	0	160	0	0	0	160	20	20	20	20	20	5720	3860	9580
	buffer MAX [bytes]	256	256	256	4	4	4	4	4	4	288	288	288	288	288			
buffer MIN [bytes]	256	256	256	4	4	4	4	4	4	288	288	288	288	288				
tb_10		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,3636	0,3817	0,7453	0,0008	0,0162	0,0423	0,0188	0,0283	0,0688	0,0004	0,0073	0,0262	0,0257	0,0596	0,0529	0,0366	0,0890
	contagem				28133	249346	55367	34605	48446									
	LAT MAX [ns]	11211,0	3952,5	15163,5	0	1180	0	0	0	1180	60	60	60	60	60	6000	6560	12420
	LAT MIN [ns]	11071,0	3592,0	14663,0	0	160	0	0	0	160	20	20	20	20	20	5720	3860	9580
	buffer MAX [bytes]	256	256	256	4	4	4	4	4	4	544	544	544	544	544			
buffer MIN [bytes]	256	256	256	4	4	4	4	4	4	544	544	544	544	544				
tb_11		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,3704	0,5497	0,9201	0,0006	0,0118	0,0528	0,0137	0,0397	0,0912	0,0003	0,0053	0,0366	0,0362	0,0784	0,0386	0,0266	0,0650
	contagem				27001	238938	79593	33164	72959									
	LAT MAX [ns]	11211,0	3952,5	15163,5	0	1180	0	0	0	1180	60	60	60	60	60	6000	6460	12440
	LAT MIN [ns]	11071,0	3592,0	14663,0	0	160	0	0	0	160	20	20	20	20	20	5720	3860	9580
	buffer MAX [bytes]	256	256	256	4	4	4	4	4	4	1056	1056	1056	1056	1056			
buffer MIN [bytes]	256	256	256	4	4	4	4	4	4	1056	1056	1056	1056	1056				
tb_12		eth			ahb					mem					proc			
		eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	w_mst4	ahb	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc	
	Util MAX	0,4215	0,4341	0,8556	0,0005	0,0093	0,0419	0,0109	0,0314	0,0722	0,0002	0,0042	0,0291	0,0286	0,0621	0,0304	0,0211	0,0510
	contagem				21863	188562	68030	26164	57559									
	LAT MAX [ns]	11211,0	3952,5	15163,5	0	1180	0	0	0	1180	60	60	60	60	60	6000	6480	12360
	LAT MIN [ns]	11071,0	3592,0	14663,0	0	160	0	0	0	160	20	20	20	20	20	5720	3860	9580
	buffer MAX [bytes]	256	256	256	4	4	4	4	4	4	1057	1057	1057	1057	1057			
buffer MIN [bytes]	256	256	256	4	4	4	4	4	4	1057	1057	1057	1057	1057				

Tabela 6.3: Resultados de todos os testbenches RTL

escritos na linguagem Perl (*Practical extraction and reporting language*) representados no quadro “Extração e coleta de parâmetros” percorrem os arquivos LST e extraem dados úteis como utilização do recurso no tempo e latência para cada um dos recursos. Tais dados são consolidados para comparação com resultados de rodadas analíticas (quadros RTL e ANL).

O ambiente de simulação foi composto de um servidor SunFire V20z, com dois processadores AMD Opteron de 2.4Ghz, 2 GB RAM DDR 400, discos USCSI 320 de 170GB 10000rpm, sistema operacional OpenSUSE 10.1, compilador C o SunStudio 10 com otimização nível “1”, simulador VHDL o Mentor Graphics Modelsim 6.1c 32bits, com resolução de 1ps, bibliotecas Gaisler Research (GR, 2007), sem aceleração de pacotes VITAL, sem otimização automática do projeto, com plataforma alvo Virtex 2, xc2v3000-fg676-4.

6.1.3 Acurácia e fidelidade na comparação entre ANL e RTL

No quadro “Cálculo de acurácia e fidelidade” da Figura 6.1 está representado o utilitário que realiza os cálculos da acurácia e fidelidade entre os resultados RTL e ANL, segundo as equações 6.1 a 6.3. Os índices *2seg* e *mseg* dos parâmetros indicam casos de 2 segmentos e múltiplos segmentos, respectivamente.

$$\text{Acurácia } A_{2seg} = \frac{(ANL_{2seg} - RTL)}{(RTL)}; \text{ para } RTL \neq 0 \quad (6.1)$$

$$\text{Acurácia } A_{mseg} = \frac{(ANL_{mseg} - RTL)}{(RTL)}; \text{ para } RTL \neq 0 \quad (6.2)$$

$$\text{Fidelidade } F = \frac{(A_{mseg})}{(A_{2seg})}; \text{ para } A_{2seg} \neq 0. \quad (6.3)$$

Se $A_{2seg} = 0, F = 0.$

Entenda-se fidelidade como uma relação entre acurácias, as relações de intensidade constantes entre dois ou mais resultados de medidas sendo comparados entre si, ainda que com valores significativamente diferentes. Dos resultados, pelo menos um deles é uma referência conhecida, ainda que com baixa exatidão.

Um histograma com três intervalos para cada uma das equações é determinado, conforme a Figura 6.2. Para as equações 6.1 e 6.2 são obtidos os números de ocorrências para cada um dos 3 intervalos, sendo eles $\{ [-0,5; 0,5] ; [(-4; -0,5(; 0,5; 4,0)] ; []-\infty; -4,0(; 4,0; \infty(] \}$ e os intervalos $\{ []-1,0; 1,0(] ; [-1,0]; [1,0]; []-\infty; -1,0(; 1,0; \infty(] \}$ para a Equação (6.3).

Figura 6.2: Intervalos (a) para contagem de ocorrências de acurácia e (b) para contagem de ocorrências de fidelidade.

Naturalmente, quanto menor os valores de acurácia (mais próximo de zero) forem os resultados das equações 6.1 ou 6.2, maior a proximidade entre o resultado de RTL e o ANL. No caso de fidelidade, resultados entre -1.0 e +1.0 (exclusivo) apontam para o aumento da proximidade entre os resultados analíticos, obtidos com a técnica de múltiplos segmentos, e os RTL; portanto, com maior acurácia frente aos obtidos com a técnica de 2-segmentos. O que se procura então é a maior quantidade de resultados de fidelidade naquele intervalo do histograma de fidelidade, com melhores valores próximos ao zero. Note-se que os resultados de fidelidade são independentes dos intervalos de acurácia escolhidos. Os intervalos de acurácia tem o fim de evidenciar a quantidade de dados, com o propósito de facilitar o grau de exatidão do método analítico: a) cujos resultados de ANL apresentaram valores superiores ou inferiores ao valores de RTL em até a metade do valor destes últimos; b) cujos resultados de ANL apresentaram valores superiores ao valores de RTL em até cinco vezes; c) cujos resultados de ANL apresentaram valores inferiores ao valores de RTL em até cinco vezes; d) cujos valores de ANL em relação ao RTL está acima dos valores anteriores (ou abaixo dos valores anteriores).

6.2 Resultados e análises

6.2.1 Validação RTL da microarquitetura alvo

O objetivo desta seção é apresentar os resultados de análise RTL, comparando os diversos casos de diferentes parâmetros de simulação (comprimento dos pacotes e dos FIFOs), para a explicação do comportamento dinâmico do sistema e corroborar a validade dos dados coletados. As Tabelas 6.2 e 6.3 (por simplicidade, no restante desta dissertação, usaremos Tabela 6.2 para nos referirmos a ambas as tabelas) consolidam os valores de utilização, latência e *buffer* obtidos por simulação RTL para os *testbenches* listados na Tabela 6.1.

Na Tabela 6.2 os casos referentes aos diversos *testbenches* estão dispostos ao longo das linhas, onde para cada um deles, são mostrados os valores de L-B-U e contagem de ocorrência no acesso ao barramento. Nas colunas estão dispostos os recursos da microarquitetura sob estudo com o respectivo detalhamento de tarefas. A seguir os dados desta tabela são discutidos e nas seções seguintes são comparados com os resultados ANL.

6.2.1.1 Controlador Ethernet - IEEE 802.3 2002 (ETH)

Em termos de utilização no tempo, é esperado que esta aumente com o comprimento do pacote de entrada, porém seja independente do comprimento do FIFO (refira-se à Tabela 6.2). A utilização do ETH não é proporcional ao comprimento do pacote pois o IFG (*interframe gap*) não é constante ao longo dos *testbenches* como pode ser visto na Figura 6.3, onde estão ilustrados os intervalos de transmissão de dados e os intervalos entre quadros. Não é possível manter o IFG constante porque isto exige a alteração do código VHDL do ETH. As latências máximas e mínimas são iguais para pacotes de comprimentos diferentes, mas

dependentes do comprimento do FIFO. Como o tempo de preenchimento do FIFO é dependente do seu comprimento, quanto menor o FIFO, menores serão as latências ($Lat_{FIFO4} < Lat_{FIFO32} < Lat_{FIFO64}$).

Figura 6.3: Especificações de dados e IFGs

Como mostrado na Tabela 6.2, a latência de ETH_TX é sempre da ordem de 3500ns, isto porque na transmissão do pacote não há contenção por parte de nenhum outro núcleo. A latência ETH_RX varia conforme o comprimento do *buffer* sendo da ordem de 1500ns (FIFO₄), 6000ns (FIFO₃₂) e 11000ns (FIFO₆₄). O circuito ETH está implementado de tal modo que, quando metade do FIFO está completo, o controlador DMA solicita uma operação de escrita (ou leitura) em memória para, somente depois de garantido acesso ao barramento, iniciar a transferência. Claramente, no caso de RX este comportamento é diferente do TX porque há contenção por parte de outro núcleo.

Dados de utilização e latência máxima do ETH são apresentados no Gráfico 6.1, para os *testbenches* sem disputa pelo barramento. A utilização de ETH_RX e ETH_TX depende exclusivamente do comprimento do pacote e é independente do comprimento do FIFO. Para os *testbenches* com pacotes de 1024bytes e 1025 bytes, a utilização foi desbalanceada (diferente de 0,50) por conta dos diferentes IFGs. Por exemplo, na Figura 6.3.a são recebidos dados por 252000ns de um

intervalo de 452120ns, o que resulta em uma utilização de 0.56. Para a Figura 6.3.b a utilização resulta em 0.44.

Tanto para ETH quanto para AHB a utilização depende muito do comprimento do pacote enquanto a latência depende muito do FIFO (quanto menor o FIFO, menor a latência). À medida que o comprimento dos pacotes aumenta, o mesmo ocorre com a utilização pois a influência do IFG diminui. Note que a utilização é independente do FIFO e a latência independente do comprimento do pacote.

Gráfico 6.1: Utilização e latência máxima de ETH para os 12 testbenches.

6.2.1.2 Barramento Amba AHB

Em termos de utilização, há um desbalanceamento entre as utilizações para operações do processador PROC (Master 0- MST0) e do ETH (Master 4- MST4). Dentre as operações do MST0, há aproximadamente 20 de escrita para cada 9 operações de leitura porque o processador lê apenas 5 linhas do cabeçalho IPv4 da memória (refletido na coluna AHB r_mst0 da Tabela 6.2), mas escreve o conteúdo de muitos ponteiros do *firmware* e também altera de modo sistemático e repetido os valores dos registradores de controle do ETH (GRETHREGS). Esta relação é constante para os *testbenches* 0001 a 0012. Nas operações realizadas pelo ETH (MST4), existe um forte balanceamento (indicado na coluna AHB r_mst4 e w_mst4 da Tabela 6.2) entre operações de leitura e escrita; a explicação é que, por se tratar de um *firmware* que implementa a política de armazena-e-encaminha, todo o pacote (cabeçalho e carga útil) deve ser escrito e posteriormente lido da memória. Na

utilização deste recurso, não está contabilizada a ocorrência de contenção no acesso ao barramento, somente as transferências efetivas de dados. O efeito de contenção de acesso ao barramento é contabilizado nas latências máxima e mínima. Na Tabela 6.2, a coluna AHB r_mst4_txnnoten (TX não habilitado) representa a utilização do barramento por parte do ETH para leitura de dados de controle, acionados pelo *firmware*, e por TX não estar habilitado, não se refere à leitura de dados de pacotes (como AHB r_mst4 e AHB w_mst4). A contagem de solicitações ao barramento tem forte equilíbrio entre AHB r_mst4 e AHB w_mst4, pelo mesmo motivo que as utilizações.

Na coluna AHB w_mst0 da Tabela 6.2 a latência apontada é diferente de zero pois não foi possível evitar completamente as contenções no acesso ao barramento dada a rigidez imposta pelo *soft-core* em RTL.

A utilização do AHB (Gráfico 6.2) tem comportamento ligeiramente diferente do ETH no que diz respeito ao comprimento do pacote e comprimento do FIFO. A utilização cresce com o comprimento do pacote para o mesmo FIFO porque a influência de ETH é maior que a do PROC. Para FIFOs de comprimentos crescentes a utilização decresce para o mesmo comprimento de pacote. FIFO de comprimento menor causa maior uso no barramento pois completa o FIFO mais rapidamente, mas transfere menos dados por rajada. A Figura 6.4 ilustra através de um exemplo quantitativo, o fenômeno de, para o mesmo comprimento de pacote, a utilização do AHB ser diferente para FIFOs diferentes; para rajada de 1024 bits (metade do comprimento do FIFO) a transferência de dez linhas do pacote IPv4 toma 260 ns, ao ponto que para rajada de 64 bits, toma-se 560 ns; a diferença entre as linhas no que se refere aos tempos de transferência ocorre por conta de passos adicionais de controle, como diversas solicitações de acesso ao barramento (mesmo sem ocorrência de contenção).

A utilização de AHB r_mst0 e AHB w_mst0 são constantes pois as operações do *firmware* são dependentes somente do cabeçalho do pacote IPv4, ainda que se varie o comprimento do FIFO (o comprimento do cabeçalho do pacote IPv4 independe do comprimento da carga útil).

Gráfico 6.2: Utilização de AHB: (a) Leitura do processador na memória, (b) escrita do processador na memória, (c) total de utilização. A queda para pacotes de 1025 bytes deve-se aos diferentes valores de intervalos entre quadros.

A latência em AHB w_mst0 é não nula pois após o *firmware* autorizar o reinício de recepção, o ETH imediatamente lê descritores na memória, simultaneamente às instruções restantes do *firmware* ainda por serem executadas, implicando em contenções no acesso ao barramento.

Figura 6.4: Comparação de tempos de transferência para duas rajadas (batidas), para um conjunto de dados das 10 primeiras linhas de um pacote IPv4, para FIFOs de comprimentos diferentes.

A contagem de acessos AHB w_mst0 (Tabela 6.2) é significativamente maior que as outras pois o processador escreve muitos dados (comprovado pela utilização de AHB w_mst0) mas espaçados entre si, impedindo a transferência por rajadas. A contagem de ocorrências de acesso ao barramento de AHB r_mst4_nottxen é alta quando comparada às contagens do AHB r_mst4 e AHB w_mst4, tomando em conta a transferência de 7000 pacotes. Na Tabela 6.4 é ilustrada a proporção de acessos ao barramento por pacote. A transferência é solicitada ao controlador DMA quando metade das linhas do FIFO está completa (as rajadas têm sempre largura de 32 bits – sinal HSIZE = 32b - e sua extensão não é declarada pelo mestre no momento da solicitação do barramento – sinal HBURST = incremental). Para completar a leitura de um pacote da memória, o ETH acessa o barramento 5 vezes para leitura de descritores de transmissão. Note a proximidade entre os valores coletados e calculados com as equações 6.4 e 6.5 das colunas em destaque nos quadros. Para escrever um pacote na memória o ETH realiza rajadas mais 3 operações de escrita (de controle) por pacote, de acordo com as seguintes equações:

$$Rajadas = \frac{(Comprimento\ do\ pacote\ em\ bits)}{\left(\frac{(Largura \times Comprimento)_{FIFO}}{2} \right)} \quad (6.4)$$

$$Operações\ no\ barramento = Rajadas + 3 \quad (6.5)$$

Tabela 6.4: Relação entre utilização do barramento pelo processador para operações de escrita e leitura; número de ocorrências de acesso ao barramento por pacote.

	Coletado					Calculado				
	relação entre utilização de ahb_w_mst0 e ahb_r_mst0	relação entre contagem de acesso de ahb_w_mst0 e ahb_r_mst0	número ocorrências de ahb_r_mst4 por pacote	número ocorrências de ahb_r_mst4 nottxen por pacote	número ocorrências de ahb_w_mst4 por pacote	FIFO	Bits da metade do FIFO	Pacote (bytes)	nº rajadas	nº total de operações no AHB
01	20	9	7,998	4,999	7,0	32	512	256	4	7,0
02	21	9	11,997	4,999	11,0			512	8	11,0
03	20	9	20,002	5,001	19,0			1024	16	19,0
04	20	9	21,000	5,000	19,0			1025	16,02	19,0
05	21	9	35,990	4,999	36,0	4	64	256	32	35,0
06	21	9	67,984	4,999	68,0			512	64	67,0
07	21	9	131,936	5,001	131,7			1024	128	131,0
08	21	9	133,002	5,000	132,0			1025	128,13	131,1
09	20	9	5,949	4,958	5,0	64	1024	256	2	5,0
10	20	9	7,910	4,944	6,9			512	4	7,0
11	20	9	12,001	5,001	11,0			1024	8	11,0
12	19	9	13,000	5,000	11,0			1025	8,01	11,0

O número de ocorrências de acesso ao barramento para FIFO32 e FIFO64 é significativamente menor que o do FIFO4 pois neste último caso apenas 2 palavras são transferidas por rajada.

6.2.1.3 Memória SRAM (MEM)

Como esperado, a utilização da memória pelo MST4 está balanceada para leitura e escrita e está desbalanceada entre MST0 e MST4. Claramente, dentro das expectativas, a utilização de memória pelo ETH é significativamente maior que a do PROC. A utilização de memória é crescente com o comprimento do pacote (Gráfico 6.3.b, c e d). A relação entre utilização de MEM w_mst0 e MEM r_mst0 (Gráfico 6.3.a) é menor quando comparada à do AHB r_mst0 e AHB w_mst0 (Gráfico 6.2.a e b). Isto porque naquele caso o processador também escreve na ponte AHB/APB (escravo do AHB) para mudar o valor dos GRETH_REGS.

Gráfico 6.3: Utilização de MEM (a) mostrando a relação entre r_mst0 e w_mst0 (20:1) para FIFO4, (b) utilização de memória para leitura pelo Ethernet, (c) utilização de memória para escrita do Ethernet, (d) utilização geral de memória no tempo.

6.2.1.4 Processador SPARC Leon (PROC)

A utilização do processador manteve-se praticamente constante para todos os *testbenches* (Gráficos 6.4.a, b e c) porque somente o cabeçalho do pacote IPv4 é avaliado, conforme o esperado. A latência acompanha o comportamento da utilização permanecendo praticamente inalterada (Gráfico 6.5) em todos os *testbenches*. O *buffer* do processador não foi avaliado pois não coletamos a ocupação de cache.

Gráfico 6.4: Utilização do processador para (a) subrotina de interrupções , (b) subrotina de encaminhamento e roteamento e (c) utilização total.

Gráfico 6.5: Latência máxima e mínima para operação de RTX, IRQHANDLER e total do processador para FIFO de 32 palavras de 32 bits. Note a independência em relação ao comprimento dos pacotes.

6.2.2 Análise comparativa entre prognósticos de parâmetros de L-B-U

Para cada simulação do modelo analítico (ANL), quatro rodadas foram processadas, para 2 segmentos e múltiplos segmentos, GRTs e GRTEs, de forma cruzada entre eles. A comparação entre ANL e RTL foi feita para as quatro configurações.

Como exemplo de avaliação detalhada de resultados, foi tomado o caso particular do *testbench* tb_01 (pacotes de 256bytes e FIFO32). Os valores discutidos nas seções seguintes estão distribuídos nas Tabelas 6.5 a 6.13. Outros *testbenches* realizados apresentaram resultados semelhantes e não serão discutidos em detalhes.

CrITÉRIOS de comparação: Por se tratar de comparação entre previsões analíticas, para as quais verifica-se grande diferença e distância entre os graus de detalhamento, há o surgimento de valores com diferenças de algumas ordens de grandeza, como poderá ser percebido nas seções seguintes. Por este motivo, não cabe, ainda, neste estudo, uma análise estatística detalhada sobre os resultados obtidos. É apresentado, então, nas seções seguintes, um estudo quantitativo e qualitativo a respeito do comportamento do modelo analítico quando se isola os recursos, os parâmetros L-B-U, a acurácia e a fidelidade. Por se tratar de um estudo determinístico, de casos mais otimistas ou pessimistas, as abordagens estatística clássica para o cálculo de acurácia, que incluem determinação de variância, também não são aplicáveis.

Acurácia: vem da estimação da correlação entre o verdadeiro valor e o valor predito, sendo assim uma medida da confiança de tal previsão. No caso deste trabalho, valor verdadeiro é definido como os resultados de simulação RTL em precisão de ciclos, como os mostrados na Tabela 6.2. Para o caso particular do *testbench* tb_01 que será utilizado na comparação, os valores estão listados na Tabela 6.5.

Tabela 6.5: Resultados RTL para o testbench tb_01 com pacote de 256B e FIFO32.

	eth_rx	eth_tx	eth	r_mst0	w_mst0	r_mst4	4_nottxer	w_mst4	ahb
Util MAX	0,2342	0,2400	0,4742	0,0010	0,0199	0,0370	0,0194	0,0200	0,0584
contagem				28486	252163	55988	34994	49029	
LAT MAX [ns]	6091,0	3632,5	9723,5	0	860	0	0	0	860
LAT MIN [ns]	5951,0	3272,0	9223,0	0	20	0	0	0	20
buffer MAX [bytes]	128	128	128	4	4	4	4	4	4
buffer MIN [bytes]	128	128	128	4	4	4	4	4	4

	mem					proc		
	r_mst0	w_mst0	r_mst4	w_mst4	mem	irq	rtx	proc
Util MAX	0,0005	0,0090	0,0179	0,0172	0,0446	0,0407	0,0257	0,0664
contagem								
LAT MAX [ns]	60	60	60	60	60	6000	5900	11720
LAT MIN [ns]	20	20	20	20	20	5720	3280	9200
buffer MAX [bytes]	288	288	288	288	288			
buffer MIN [bytes]	288	288	288	288	288			

Como exposto anteriormente, quanto mais próximo de zero, melhor o valor de acurácia. A análise qualitativa a ser realizada tem o propósito de entender o comportamento e as limitações do modelo analítico, averiguando a efetividade das contribuições teóricas deste trabalho, além de descobrir onde devem ser atribuídas possíveis contribuições e melhorias ao arcabouço algébrico. A faixa de valores de acurácia entre (-1,0 1,0) foi definida com o propósito de facilitar a interpretação dos dados e tida como a mais importante, pois uma predição até duas vezes maior que o valor verdadeiro pode ser aceitável durante a fase de projeto na qual se enquadra o uso da metodologia proposta neste trabalho. Este intervalo pode ser livremente determinado pelo projetista de sistemas.

Fidelidade: para o caso de fidelidade, o intervalo é o mesmo, mas por razões diferentes. A fidelidade menor que a unidade aponta para a melhoria no modelo com o uso de múltiplos segmentos; a fidelidade igual à unidade aponta para a manutenção nos índices de acurácia. Portanto, o que se quer é avaliar as influências que causam melhorias ou estagnação nas estimações de fidelidade.

Conforme abordado no Capítulo 5, a latência é um importante parâmetro na especificação e projeto de processadores de rede. Por esta razão, ela será considerada prioritária em relação ao *buffer* e utilização, nas avaliações dos resultados desta dissertação. Em relação a recursos, a importância é indistinta pois considera-se que todos os recursos realizam tarefas fundamentais no tratamento do pacotes.

Tanto a acurácia quanto a fidelidade devem ser objetos de análise pois a

primeira apresenta uma estimaco esttica entre o verdadeiro valor e sua predico analtica correspondente; j a fidelidade indica a alteraco desta acurcia para melhor ou para pior.

Os resultados analticos do *testbench* tb_01 esto listados nas Tabelas 6.6 a 6.8. A Tabela 6.6 apresenta os resultados analticos para o GRT com ambos os casos de 2 e mltiplos segmentos, enquanto que nas Tabelas 6.7 e 6.8 esto listados os resultados analticos para GRTEs com 2 e mltiplos segmentos, respectivamente. Pode-se observar que nos resultados analticos de L e B mnimos, h grande incidncia de valores nulos porque o grau de conflito por acesso a recursos foi propositalmente imposto como baixo. Desta forma, pode-se estudar, de forma isolada, os atributos dependentes da influncia de tais conflitos. Acreditamos que futuros *testbenches* com imposio de graus mdios e altos de conflito e conteno no acesso a recursos prediro valores diferentes de nulo. Naturalmente, ainda continuam vlidas as predices obtidas neste trabalho.

Tabela 6.6: Resultados analticos para os quatro recursos do microssistema. (a) GRT de 2 segmentos, (b) GRT de mltiplos segmentos.

		eth	ahb	mem	proc
GRT – 2 segmentos	Util MAX	0,338	0,195	0,137	0,273
	Util MIN	0,338	0,219	0,706	0,273
	LAT MAX [ns]	16652	3116	0	8907
	LAT MIN [ns]	0	0	0	0
	buffer MAX [bytes]	139,2	149,4	4,0	91,5
	buffer MIN [bytes]	0,00	0,00	0,00	0,00

(a)

		eth	ahb	mem	proc
GRT – mltiplos segmentos	Util MAX	0,338	0,195	1,000	0,273
	Util MIN	0,338	0,000	0,000	0,273
	LAT MAX [ns]	40305	2176	2942	0
	LAT MIN [ns]	0	0	0	0
	buffer MAX [bytes]	4,0	72,4	142,2	4,0
	buffer MIN [bytes]	0,00	0,00	0,00	0,00

(b)

Tabela 6.7: Resultados analíticos para os doze recursos do microssistema, para o GRTE de 2 segmentos.

		eth		ahb			
		eth_rx	eth_tx	r_mst0	w_mst0	r_mst4	w_mst4
GRTE – 2 segmentos	Util MAX	0,253	0,338	0,553	0,363	0,955	0,036
	Util MIN	0,253	0,507	0,553	0,363	0,935	0,156
	LAT MAX [ns]	16662	267728	0	0	0	2262
	LAT MIN [ns]	0	0	0	0	0	0
	buffer MAX [bytes]	189,067	2270,486	4,000	0,000	0,000	181,000
	buffer MIN [bytes]	0,0	0,0	0,0	0,0	0,0	0,0

		mem				proc	
		r_mst0	w_mst0	r_mst4	w_mst4	rqhandlel	rtx
GRTE – 2 segmentos	Util MAX	0,695	0,229	0,612	0,029	0,067	0,072
	Util MIN	0,695	1,000	1,000	0,125	0,293	0,314
	LAT MAX [ns]	0	0	3337	40	2388	6035
	LAT MIN [ns]	0	0	0	0	0	0
	buffer MAX [bytes]	0,000	0,000	65,037	4,000	101,896	240,206
	buffer MIN [bytes]	0,0	0,0	0,0	0,0	0,0	0,0

6.2.3 Análise comparativa entre recursos Eth, Ahb, Mem e Proc

Com os resultados analíticos apresentados na seção anterior, foram construídas tabelas de prognósticos de acurácia e fidelidade (GRT e GRTE, com 2 e múltiplos segmentos), que serão mostradas mais adiante, da Tabela 6.9 à Tabela 6.13. Nesta seção faremos uma análise destes resultados com foco nos recursos e na comparação entre eles. Para facilitar este trabalho colocamos os resultados das tabelas em forma de histogramas e consolidamos nos quadros que estão no APÊNDICE C para consulta.

6.2.3.1 Acurácia A_{GRT} 2 segmentos versus A_{GRT} múltiplos segmentos

No que se refere a recursos, Eth apresenta melhor predição de acurácia para os dois modelos analíticos (GRT, com 2 e múltiplos segmentos), como pode ser visto na Tabela 6.9. Este recurso apresenta apenas 1 caso em 12 fora do intervalo $[-1,0; 1,0]$, ao passo que os outros recursos apresentaram metade dos 32 casos neste intervalo. Acreditamos que o recurso Eth apresenta melhor predição de acurácia porque tem valores medidos RTL (verdadeiros) do trio L-B-U significativamente mais altos que Ahb, Mem e Proc.

O Eth, por oferecer a melhor estimativa da correlação entre o valor verdadeiro e o valor predito, na comparação com os recursos Ahb, Mem e Proc, terminou por apresentar a pior predição de fidelidade, como pode ser visto na Tabela 6.10. Os recursos Ahb e Mem apresentam melhorias na fidelidade, ao contrário do Proc e Eth, mas este último apresenta piores índices de fidelidade (4,41 e -11,06).

Tabela 6.8: Resultados analíticos para os doze recursos do microssistema, para o GRTE de múltiplos segmentos.

		eth		ahb			
		eth_rx	eth_tx	r_mst0	w_mst0	r_mst4	w_mst4
GRTE – múltiplos segmentos	Util MAX	0,253	0,338	0,037	0,039	0,415	0,036
	Util MIN	0,253	0,338	0,037	0,039	0,415	0,156
	LAT MAX [ns]	16661,590	2,29E+006	0,000	0,000	0,000	2262,499
	LAT MIN [ns]	0,000	0,000	0,000	0,000	0,000	0,068
	buffer MAX [bytes]	189,067	19396,082	4,000	0,000	0,000	181,000
	buffer MIN [bytes]	0,0	0,0	0,0	0,0	0,0	0,0

		mem				proc	
		r_mst0	w_mst0	r_mst4	w_mst4	rhandle	rtx
GRTE – múltiplos segmentos	Util MAX	0,030	0,716	0,781	0,029	0,067	0,072
	Util MIN	0,030	0,716	0,253	0,125	0,293	0,314
	LAT MAX [ns]	0,000	0,065	0,000	40,000	2388,189	6035,366
	LAT MIN [ns]	0,000	0,000	0,000	0,068	0,068	0,027
	buffer MAX [bytes]	0,000	0,000	0,000	4,000	101,896	240,206
	buffer MIN [bytes]	0,0	0,0	0,0	0,0	0,0	0,0

Pode-se concluir portanto que, porque houveram piores e melhoras nas predições, apenas a adoção de múltiplos segmentos não é suficiente para sempre garantir que a fidelidade melhore.

É interessante o comportamento de melhora da fidelidade de recursos com predições de acurácia iniciais mais pobres porque reflete a capacidade do modelo em fazer com que recursos com predições pobres de acurácia (valores ANL afastados do RTL em GRT com 2 segmentos) tenham as melhores predições de fidelidade, pois passam a apresentar melhores valores de acurácia (valores ANL mais próximos de RTL em GRT com múltiplos segmentos).

Tabela 6.9: Predição de acurácia para os quatro recursos do microssistema. (a) GRT de 2 segmentos, (b) GRT de múltiplos segmentos. Os valores no intervalo $[-1,0 +1,0]$ estão em destaque. Melhores valores são os mais próximos de zero.

		eth	ahb	mem	proc
GRT – 2 segmentos	Util	-0,29	2,33	2,09	3,11
	Util	-0,29	2,74	14,84	3,11
	LAT MAX [ns]	0,71	2,62	-1,00	-0,32
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00
	buffer MAX [bytes]	0,09	36,36	-71,00	
	buffer MIN [bytes]	-1,00	-1,00	-1,00	

(a)

		eth	ahb	mem	proc
GRT – múltiplos segmentos	Util MAX	-0,29	2,33	21,44	3,11
	Util MIN	-0,29	-1,00	-1,00	3,11
	LAT MAX [ns]	3,15	1,53	48,04	-1,00
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00
	buffer MAX [bytes]	-0,97	17,10	-0,51	
	buffer MIN [bytes]	-1,00	-1,00	-1,00	

(b)

6.2.3.2 Acurácia A_{GRTE} 2 segmentos versus A_{GRTE} múltiplos segmentos

Apresenta comportamento diferente em relação ao exposto na seção 6.2.3.1, o que pode ser confirmado nas Tabelas 6.11, 6.12 e 6.13. e no Quadro C.1.d. Percebe-se que, com a adoção de GRTE, todas as predições de acurácia apresentam estabilidade ou melhora, o que mostra que a adoção *simultânea* de

Tabela 6.10: Predição de fidelidade para os quatro recursos do microssistema representado pelo GRT. Os valores no intervalo $[-1,0 +1,0]$ estão em destaque. Melhores valores são os iguais ou menores que a unidade .

	eth	ahb	mem	proc
Util MAX	1,00	1,00	10,28	1,00
Util MIN	1,00	-0,36	-0,07	1,00
LAT MAX [ns]	4,41	0,58	-48,02	3,17
LAT MIN [ns]	1,00	1,00	1,00	1,00
buffer MAX [bytes]	-11,06	0,47	0,01	
buffer MIN [bytes]	1,00	1,00	1,00	

GRTE com múltiplos segmentos é a mais indicada no estudo de acurácia e fidelidade por recursos.

Tabela 6.11: Predição de acurácia para os doze recursos do microssistema representados através do GRTE de 2 segmentos. Os valores no intervalo $[-1,0 +1,0]$ estão em destaque. Melhores valores são os mais próximos de zero.

		eth		ahb			
		eth_rx	eth_tx	r_mst0	w_mst0	r_mst4	w_mst4
GRTE – 2 segmentos	Util MAX	0,08	0,41	566,09	17,22	24,83	0,79
	Util MIN	0,08	1,11	566,09	17,22	24,30	6,81
	LAT MAX [ns]	1,74	72,70	-1,00	-1,00	-1,00	###
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00	-1,00	67499,00
	buffer MAX [bytes]	0,48	16,74	0,00	-1,00	-1,00	44,25
	buffer MIN [bytes]	-1,00	-1,00	-1,00	-1,00	-1,00	-1,00

		mem				proc	
		r_mst0	w_mst0	r_mst4	w_mst4	rqhandle	rtx
GRTE – 2 segmentos	Util MAX	1517,07	24,63	33,19	0,66	0,65	1,80
	Util MIN	1517,07	110,67	54,84	6,25	6,20	11,22
	LAT MAX [ns]	-1,00	-1,00	54,61	-0,33	-0,60	0,02
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00	-1,00	-1,00
	buffer MAX [bytes]	-1,00	-1,00	-0,77	-0,99		
	buffer MIN [bytes]	-1,00	-1,00	-1,00	-1,00		

Tabela 6.12: Predição de acurácia para os doze recursos do microssistema representados através do GRTE de múltiplos segmentos. Os valores no intervalo $[-1,0 +1,0]$ estão em destaque. Melhores valores são os mais próximos de zero.

		eth		ahb			
		eth_rx	eth_tx	r_mst0	w_mst0	r_mst4	w_mst4
GRTE – múltiplos segmentos	Util MAX	0,08	0,41	37,10	0,94	10,23	0,79
	Util MIN	0,08	0,41	37,10	0,94	10,23	6,81
	LAT MAX [ns]	1,74	629,78	-1,00	-1,00	-1,00	###
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00	-1,00	67499,00
	buffer MAX [bytes]	0,48	150,53	0,00	-1,00	-1,00	44,25
	buffer MIN [bytes]	-1,00	-1,00	-1,00	-1,00	-1,00	-1,00

		mem				proc	
		r_mst0	w_mst0	r_mst4	w_mst4	rqhandle	rtx
GRTE – múltiplos segmentos	Util MAX	63,48	78,96	42,62	0,66	0,65	1,80
	Util MIN	63,48	78,96	13,11	6,25	6,20	11,22
	LAT MAX [ns]	-1,00	-1,00	-1,00	-0,33	-0,60	0,02
	LAT MIN [ns]	-1,00	-1,00	-1,00	-1,00	-1,00	-1,00
	buffer MAX [bytes]	-1,00	-1,00	-1,00	-0,99		
	buffer MIN [bytes]	-1,00	-1,00	-1,00	-1,00		

6.2.3.3 Análise comparativa entre prognósticos de acurácias

A acurácia, com o uso de múltiplos segmentos, sob a perspectiva de recursos, não é significativamente melhorada no caso do GRT, ou seja, o número de ocorrências no intervalo de interesse, $[-1,0 +1,0]$, manteve-se praticamente inalterado. Para a Mem e Ahb aumentaram as ocorrências e para Eth e Proc diminuíram as ocorrências, na mesma proporção; por exemplo, pela Tabela 6.9 no caso do Ahb, a predição de acurácia para utilização mínima, que era de 2,74, passou para -1,0 (causando uma nova ocorrência no intervalo). Porém, no caso do GRTE associado à adoção de múltiplos segmentos, a predição de acurácia permanece ou melhora em todos os recursos, quando avaliado apenas o intervalo de interesse. Neste caso o número de ocorrências no intervalo, em todos os recursos, permaneceu inalterado ou aumentou. Este fenômeno pode ser observado nas Tabelas 6.11 e 6.12, indicando que a adoção simultânea de GRTE com múltiplos segmentos é a mais apropriada no estudo de desempenho por recursos.

Tabela 6.13: Predição de fidelidade para os doze recursos do microssistema representado pelo GRTE. Os valores no intervalo $[-1,0 +1,0]$ estão em destaque. Melhores valores são os iguais ou menores que a unidade.

	eth		ahb			
	eth_rx	eth_tx	r_mst0	w_mst0	r_mst4	w_mst4
Util MAX	1,00	1,00	0,07	0,05	0,41	1,00
Util MIN	1,00	0,37	0,07	0,05	0,42	1,00
LAT MAX [ns]	1,00	8,66	1,00	1,00	1,00	1,00
LAT MIN [ns]	1,00	1,00	1,00	1,00	1,00	1,00
buffer MAX [bytes]	1,00	8,99	1,00	1,00	1,00	1,00
buffer MIN [bytes]	1,00	1,00	1,00	1,00	1,00	1,00

	mem				proc	
	r_mst0	w_mst0	r_mst4	w_mst4	rqhandler	rtx
Util MAX	0,04	3,21	1,28	1,00	1,00	1,00
Util MIN	0,04	0,71	0,24	1,00	1,00	1,00
LAT MAX [ns]	1,00	1,00	-0,02	1,00	1,00	1,00
LAT MIN [ns]	1,00	1,00	1,00	1,00	1,00	1,00
buffer MAX [bytes]	1,00	1,00	1,29	1,00		
buffer MIN [bytes]	1,00	1,00	1,00	1,00		

6.2.3.4 Análise comparativa entre prognósticos de fidelidade

A fidelidade do GRT, conforme pode ser visto na Tabela 6.10 e no Quadro C.1.e, melhora ou permanece inalterada de 67% (para 2 recursos) a 83% (para 1

recurso) de todas as ocorrências. No caso do GRTE, visto na Tabela 6.13 e no Quadro C.1.f, a fidelidade permanece ou melhora de 50% (para 2 recursos) a 100% (para 9 recursos) de todas as ocorrências. Estes resultados reforçam ainda mais que, no que se refere a recursos, a adoção de GRTE com múltiplos segmentos é a melhor modelagem analítica no escopo deste estudo.

6.2.4 Análise comparativa entre parâmetros L-B-U

Nesta seção faremos uma análise dos resultados com foco nos parâmetros L-B-U e na comparação entre eles. Para facilitar este trabalho utilizamos novamente os resultados das tabelas em forma de histogramas e consolidamos nos quadros que estão no APÊNDICE C para consulta.

6.2.4.1 Acurácia A_{GRT} 2 segmentos versus A_{GRT} múltiplos segmentos

No estudo da acurácia com GRT, como ilustrado na Tabela 6.9, utilizamos o histograma do Gráfico 6.6 como um auxílio na visualização dos resultados. Nele é mostrada a acurácia para os casos de 2 e múltiplos segmentos com dados normalizados nos intervalos $(-1,0; 1,0)$ (exclusivo) e 1,0. Note que para o caso de 2 segmentos, 75% dos casos de latência, 25% de utilização e 16,67% de *buffer* estão no primeiro intervalo, enquanto 50% dos casos de *buffer* ocorre para o valor 1,0 (para latência e utilização, não ocorreu nenhum caso neste valor). A interpretação para o histograma de múltiplos segmentos é análoga. Pode-se observar que a abordagem de múltiplos segmentos melhorou os resultados para os parâmetros utilização e *buffer*, se comparada à de 2 segmentos, mas piorou para o caso de latência, tendo sido mais significativa a evolução da acurácia de utilização quando comparada com a de *buffer*. Porque houve piora com a adoção de múltiplos segmentos, podemos perceber que, quando trata-se de parâmetros, apenas a adoção de múltiplos segmentos não é suficiente para sempre garantir melhora nos prognósticos de acurácia.

A predição de fidelidade de utilização foi a mais destacada, ainda que a acurácia predita deste parâmetro tenha tido índices de qualidade inferiores aos de L-

B, o que mostra a importância do estudo simultâneo de acurácia e fidelidade. Este comportamento é o antagônico ao do recurso Eth, conforme exposto na Seção 6.2.3.1. Neste sentido, nota-se a contribuição de múltiplos segmentos pois assegura melhoria no prognóstico da pior predição de acurácia. Indicando, novamente, o comportamento de melhora da fidelidade de parâmetros com predições de acurácia iniciais mais pobres porque reflete a capacidade do modelo em fazer com que parâmetros com predições pobres de acurácia (valores ANL afastados do RTL em GRT com 2 segmentos) tenham as melhores predições de fidelidade, pois passam a apresentar melhores valores de acurácia (valores ANL mais próximos de RTL em GRT com múltiplos segmentos).

Gráfico 6.6: Histograma indicando os índices de predição de acurácia para o GRT de 2 e de múltiplos segmentos.

6.2.4.2 Acurácia A_{GRTE} 2 segmentos versus A_{GRTE} múltiplos segmentos

O número de ocorrências normalizado para a acurácia predita do trio L-B-U no GRTE de 2 segmentos foi de (0,75; 0,90; 0,25) no intervalo $[-1,0 +1,0]$ e no GRTE de múltiplos segmentos foi de (0,834; 0,90; 0,375), conforme pode ser inferido nas Tabelas 6.11 e 6.12 e no Quadro C.2.c e C.2.d. A predição de acurácia melhorou para L e U, mas permaneceu constante para B; note-se que o índice de acurácia para B já era alto (0,90) para o caso de 2 segmentos. Isto reforça novamente a capacidade do modelo analítico de fazer com que predições pobres de acurácia tenham melhores predições de fidelidade, e também mantendo índices de acurácia para os casos em que inicialmente são altos.

Em termos de comparação entre latência, comprimento de *buffer* e utilização, esta última teve piores prognósticos de acurácia frente aos dois primeiros parâmetros. No entanto, o novo modelo analítico (contemplando as contribuições deste trabalho, diga-se GRTE aliado à adoção de múltiplos segmentos) proporcionou os melhores índices de fidelidade ao parâmetro utilização. Diz-se, portanto, que melhorou a medida de confiabilidade da sua predição.

Para o parâmetro *buffer*, obteve-se as melhores predições de acurácia frente à latência e utilização, e obteve-se, ao mesmo tempo, altos índices de fidelidade (próximos aos de utilização). Isto é interessante porque o *buffer* é o único parâmetro não dimensionado no tempo.

A latência, tomada como o parâmetro mais importante dentre os três, apresentou prognósticos intermediários de acurácia (tendo piorado do caso de GRT com 2 segmentos para o de GRT de múltiplos segmentos), o que não é uma situação desejável. Note-se, no entanto, que para a latência obteve-se o maior salto nas predições de fidelidade de GRT e GRTE, de 0,50 para 0,83, respectivamente. Conclui-se, deste modo, que a melhor opção para o estudo de processadores de rede (dentro do escopo do estudo deste trabalho) é a adoção do GRTE com múltiplos segmentos, porque assim pode-se obter, para a latência, os melhores prognósticos.

6.2.5 Qualificação da fidelidade para GRT e GRTE

Na análise comparativa de fidelidade do GRT (F_{GRT}) nota-se, pela Tabela 6.10 e pelo Quadro C.2.e e C.2.f, que a qualidade de fidelidade (concentração de índices menores ou iguais à unidade) é destacada na abordagem do GRT, onde o número de ocorrências normalizado para a fidelidade predita do trio L-B-U foi de (0,50; 0,83; 0,88). A fidelidade, permanecendo a mesma ou melhorando, indica a efetiva contribuição da abordagem de múltiplos segmentos para a melhoria da acurácia.

No caso de fidelidade do GRTE (F_{GRTE}), nota-se pela Tabela 6.13, que a seqüência obtida foi de (0,82; 0,90; 0,92) para o trio L-B-U no intervalo $[-1,0 +1,0]$, portanto com mais ocorrências que com o F_{GRT} . Já que todos os prognósticos de fidelidade foram melhores que F_{GRT} para tais parâmetros, conclui-se que a

abordagem de GRTE oferece melhorias ao modelo, não somente por conta da predição de acurácia como também dado o número 3 vezes maior na quantidade de detalhes do sistema sob análise.

Tal qualidade de fidelidade mostra que é possível fazer com que os prognósticos de acurácia sejam melhorados, aumentando a proximidade entre as métricas verdadeiras (aquelas obtidas na simulação RTL) e tais prognósticos.

6.2.6 Seqüência de vértices no GRTE e melhoria da predição de acurácia

Apesar de aparentemente não-forte, existe correlação entre a posição do vértice recurso-tarefa na seqüência do grafo e os níveis de predição de acurácia e de fidelidade. De modo típico, os vértices finais na seqüência do GRTE (Figura 4.4), por exemplo, (tx@eth2), (eth_tx,r)@mem4 e (eth_tx,r)@ahb4 apresentam piores prognósticos de acurácia e fidelidade, como pode ser visto nas Tabelas 6.11, 6.12 e 6.13, possivelmente como resultado da influência dos desdobramentos do cálculo das curvas de chegada remanescentes em cada um dos vértices. Isto pode indicar que o modelo analítico enxerga saturação após o cômputo de determinada quantidade de vértices, sugerindo a necessidade de mais investigação sobre este efeito

Capítulo 7 - Conclusões e recomendações de trabalhos futuros

Nesta dissertação diversas questões relacionadas à análise de tempo de sistemas processadores de pacotes foram estudadas, assim como aspectos de ocupação de área representada pela requisição de *buffer* para os componentes do sistema. Tais processadores surgiram muito recentemente, com o intuito de gerenciar a crescente complexidade de tarefas de processamento de pacotes e também a crescente capacidade dos meios de comunicação.

Neste trabalho apresentamos uma extensão do NetCal, um arcabouço analítico e recurso valioso para guiar os passos do ciclo de projeto que seguem a fase analítica. Algumas melhorias importantes em relação a pesquisas anteriores foram apresentadas, como a formalização do conceito de critério de estabilidade, aproximação linear de curvas por partes sem limitação no número de segmentos, grafo detalhado de recurso-tarefa, limiares mínimos no cálculo de latência, *buffer* e utilização, L-B-U, assim como a comparação de resultados analíticos, ANL, versus resultados da simulação de implementação RTL (rodando aplicativo de encaminhamento de pacotes IPv4 sujeito a tráfego pseudo-sintético).

Nós pesquisamos o problema da determinação da viabilidade do mapeamento de diferentes tarefas de processamento de rede em diferentes componentes da microarquitetura de um processador de rede implementado como sistema-sobre-silício, de modo dinâmico. Tal estudo foi baseado em aplicação orientada a controle de dados e a caminho de dados, de modo simulatâneo, não sendo puramente controle de dados ou puramente caminho de dados.

Uma análise sobre os efeitos das extensões propostas nesta dissertação foi feita através de valores de acurácia e fidelidade. Foi possível perceber um incremento de ambos nos resultados analíticos quando as metodologias apresentadas neste trabalho são aplicadas, enquanto os esforços computacionais demandados para explorar tais metodologias permanecem na mesma ordem de magnitude de outros trabalhos relacionados, de modo quantitativo, é possível obter a análise de 500 mil arquiteturas por minuto (incluindo prognósticos de acurácia e fidelidade) apontando para uma proporção de $1.8E9$ segundos de simulação RTL para cada 1 segundo de analítico.

Esta é uma corroboração do arcabouço de NetCal, visto que tais comparações

entre dois níveis de abstração muito distintos foram conduzidas. Em todo caso, mais aperfeiçoamentos às equações são necessários, principalmente aqueles relacionados aos requerimentos de *buffers*, e também ainda merece atenção a necessidade de se expandir os níveis de carga e conflito no barramento para conduzir futuras avaliações.

Os resultados deste trabalho mostraram-se bastante satisfatórios refletidos nas duas publicações decorrentes dele:

- F. de Faria, M. Strum, W. J. Chau, “Performance Evaluation of a Network Processor's Bus Arbiter Policies Using Worst-Case Deterministic Analytical Methodologies”, Proc. of XII IBERCHIP, páginas 306-310, Março 2006, San Jose, Costa Rica.
- F. de Faria, M. Strum, W. J. Chau, “A system-level Performance Evaluation Methodology for Network Processors Based on Network Calculus Analytical Modeling”, Proc. of ISVLSI, 2007 - IEEE Computer Society Annual Symposium on VLSI, páginas 265-272, Maio de 2007, Porto Alegre.

As contribuições teóricas adotadas neste trabalho mostraram-se efetivas no sentido de melhorar a acurácia e a fidelidade das predições analíticas de L-B-U, em parte ajudada pelo fato de o microssistema sob estudo (MSE) ter sido analisado de modo fragmentado. Ainda que a modelagem tenha sido limitada no tocante à extensão dos graus de contenção no acesso a recursos do MSE, formou-se uma base sólida para o estudo mais detalhado dos efeitos e conseqüências da ocorrência de contenção. Como trabalhos futuros para prosseguimento desta linha de pesquisa, sugere-se :

- Testar a metodologia analítica em condições mais realistas, simulando o microssistema sujeito a carga de trabalho que gere maior grau de conflitos e contenções no acesso a recursos como o barramento e memória; instanciando mais interfaces de rede e alterando o *firmware* para suportar o encaminhamento de pacotes de e para diferentes enlaces de rede (realizar a modelagem através do conceito de múltiplos fluxos de pacotes ingressantes e egressantes); e alterando o soft-core do MSE para adotar o política de encaminhamento através de *cut-through*, em detrimento da atual política armazena-e-encaminha, assim como para suportar tarefas de *software* mais complexas, como as presentes em outras camadas da pilha OSI, e comumente encontradas em processadores de rede comerciais.

- Um estudo detalhado sobre a granularidade das representações abstratas do par recurso-tarefas. A relevância disto seria o desconhecimento a respeito das influências que causam perda de fidelidade e acurácia nas predições do modelo quando atinge-se determinada quantidade de vértices nos grafo, conforme discutido na Seção 6.2.7.
- Investigar a capacidade de modelagem de sistemas com exigências de qualidade de serviços sob a influência de múltiplos fluxos de pacotes; analisar a possibilidade de gerar GRTEs condicionados incorporando a influência de probabilidades de desvios no fluxo de controle de tarefas.

APÊNDICE A - Definições algébricas básicas

Definição (convolução *min-plus*) Para duas funções γ_1 e γ_2 defini-se $\gamma_1 \otimes \gamma_2$ como:

$$\gamma_1 \otimes \gamma_2 = \min_{u: 0 \leq u \leq t} \{ \gamma_1(u) + \gamma_2(t-u) \}$$

As seguintes propriedades se aplicam a convolução *min-plus* (BOUDEC; THIRAN, 2001):

- convolução é associativa e comutativa;
- Se $\gamma_1(0) = \gamma_2(0) = 0$ então $\gamma_1 \otimes \gamma_2 \leq \min\{\gamma_1(u), \gamma_2(t-u)\}$;
- Se γ_1 e γ_2 são côncavas então $\gamma_1 \otimes \gamma_2 = \min\{\gamma_1(u), \gamma_2(t-u)\}$;

Definição (deconvolução *max-plus*) Para duas funções γ_1 e γ_2 defini-se $\overline{\gamma_1 \otimes \gamma_2}$ como:

$$\overline{\gamma_1 \otimes \gamma_2} = \min_{u \geq 0} \{ \gamma_1(t+u) - \gamma_2(u) \}$$

Definição (convolução *max-plus*) Para duas funções γ_1 e γ_2 defini-se $\overline{\gamma_1} \otimes \overline{\gamma_2}$ como:

$$\overline{\gamma_1} \otimes \overline{\gamma_2} = \max_{u: 0 \leq u \leq t} \{ \gamma_1(u) + \gamma_2(t-u) \}$$

Definição (deconvolução *min-plus*) Para duas funções γ_1 e γ_2 defini-se $\gamma_1 \oslash \gamma_2$ como:

$$\gamma_1 \oslash \gamma_2 = \max_{u \geq 0} \{ \gamma_1(t+u) - \gamma_2(u) \}$$

O operador deconvolução tem as seguintes propriedades (

$\gamma_1 \leq \gamma_2$ significa que $\gamma_1(t) \leq \gamma_2(t) \forall t \geq 0$ (BOUDEC; THIRAN, 2001).

- Se $\alpha_1 \leq \alpha_2$ então para todo β : $\alpha_1 \otimes \beta \leq \alpha_2 \otimes \beta$.
- Se $\beta_1 \leq \beta_2$ então para todo α : $\alpha \otimes \beta_1 \leq \alpha \otimes \beta_2$.
- $(\gamma_1 \otimes \gamma_2) \otimes \gamma_3 = \gamma_1 \otimes (\gamma_2 \otimes \gamma_3)$.
- $\gamma_1 \otimes \gamma_2 \leq \gamma_3$ é equivalente a $\gamma_1 \leq (\gamma_2 \otimes \gamma_3)$.

Observação: Intuitivamente, $(\gamma_1 \otimes \gamma_2)(t)$ é a maior diferença em valores entre pontos em γ_1 e γ_2 que estão separados entre si por uma distância t .

Definição (sub-aditividade) Uma função γ é dita ser sub-aditiva se $\gamma(s+t) \leq \gamma(s) + \gamma(t) \forall s, t \geq 0$. Esta definição equivale dizer que $\gamma \leq \gamma \otimes \gamma$. Com $\gamma(0) = 0$, sub-aditividade é equivalente a $\gamma = \gamma \otimes \gamma$ (BOUDEC; THIRAN, 2001).

Corolário: Qualquer função côncava γ definida em $\mathbb{R}^+$ tal que $\gamma(0) = 0$ é sub-aditiva. Demonstração em [ref netcal].

Definição (envólucro sub-aditivo) O envólucro sub-aditivo de uma função $\gamma, \bar{\gamma}$, é sub-aditivo. Demonstração em (BOUDEC; THIRAN, 2001). Observação: se a função γ é sub-aditiva, então $\bar{\gamma} = \gamma$.

Definição (super-aditividade) Uma função γ é dita ser super-aditiva se $\gamma(s+t) \geq \gamma(s) + \gamma(t) \forall s, t \geq 0$.

Definição (envólucro super-aditivo) O envólucro super-aditivo de uma função γ é $\bar{\gamma} = \max_{n \geq 1} \gamma^{(n)}$, onde $\gamma^{(n)} = \gamma \otimes \gamma \otimes \dots \otimes \gamma$.

Corolário O envólucro super-aditivo de uma função $\gamma, \bar{\gamma}$, é super-aditivo.

APÊNDICE B – Detalhamento de operações para levantamento da curva de serviços relativos ao *firmware*.

B.1 Rotina IRQHANDLER

- dados foram escrito nos endereços de SRAM diretamente pelo DMA ao longo da operação de RX.
- recebe interrupção (sinal `irqi(0).intack = 1`). Toma 3680ns para carregar a subrotina `IRQhandler()`.
- Inicia a sub-rotina do *firmware* `IRQHANDLER()` com a função `disable_irq(12)`: 80 ciclos do processador (clock de 50Mhz);
- lê valor do registrador `imask (0x80000240)`: 8 ciclos do processador;
- escreve novo valor de `imask (0x80000240)`: 72 ciclos do processador;
 - habilita a recepção no núcleo Ethernet com escrita em `GRETHREGS_ADDR`;
 - pede o barramento (sinal `msto(4).hbusreq = 1`), aguarda por (`msti.hgrant = 0x00001`). Barramento garantido ao Ethernet (mestre 4): 10 ciclos do processador;
- disputa pelo barramento resulta em latência de operação de 6180 ns para processar `IRQHANDLER`. Ocorre uma disputa pelo barramento, em que o processador perde.
- sub-rotina `IRQHANDLER` executa a função `enable_irq(12)`: 7 ciclos do processador;
- atualiza a variável `rxpcnt (RX package count)` com escrita na memória (endereço `0x40032924`). Os endereços `0x4` são da SRAM e os `0x8` são os endereços dos registradores do núcleo Ethernet: 12 ciclos do processador;
- sub-rotina de interrupção `IRQhandler` finalizada.

B.2 Rotina RTX

- Início da sub-rotina de transmissão TX com escrita da variável `dest_ip_addr` em `0x40032784`: 1 ciclo do processador;
- Termina a operação sobre `dest_ip_addr` com escrita em `0x400ffe84` && `0x0a00`: 110 ciclos do processador;
- localiza a porta de saída para o pacote com busca binária na tabela de roteamento: 130 ciclos do processador;
- diminui o TTL: 1 ciclo do processador;
- calcula o *checksum* do cabeçalho IPv4: 200 ciclos do processador;
- habilita a transmissão no núcleo Ethernet com escrita no `GRETHREGS_ADDR`: 4 ciclos do processador; (sinal `r.ctrl.txen` = 1 e `r.txdstate` = `read_desc`, `msti.hgrant` = `0x00001` e `msto(4).hbusreq` = 1)
- Disputa pelo barramento (= latência de contenção). Núcleo Ethernet deseja ler dados na memória para enviar pacote (TX) pois já foi habilitada a enviar no passo acima e simultaneamente o processador deseja executar as 4 últimas linhas da subrotina (RTX) com escrita em `0x40032930`. Término da sub-rotina (RTX): 5 ciclos do processador;
- Sinal `txfo.data` = `0x5e00`, `@16818851 slvo(0).hrdata` = `0x5e00`: 1 ciclo do processador;
- Encaminhamento: Compara endereços para não estourar a pilha `rx_desc` ou *payload*; faz busca binária em tabela de roteamento previamente construída, para comparar endereço de destino IPv4 contra a tabela. Latência δ_{θ} depende da extensão da tabela; decrementa TTL (1 ciclo); calcula *checksum* do cabeçalho IPv4; as 5 linhas do cabeçalho IPv4 foram lidas ou alteradas e estão prontas.

Comparação entre parâmetros

acuracia GRT 2 segmentos			
Histograma de Acurácia			
	lat	buffer	util
Módulo			
(0, 1,0)	0,750	0,167	0,250
1.0	0,000	0,500	0,000
(1,0, 1,5]	0,125	0,000	0,000
(1,5, 2,0]	0,000	0,000	0,000
(2,0, 2,5]	0,000	0,000	0,250
(2,5, 3,0]	0,125	0,000	0,125
(3,0, 3,5]	0,000	0,000	0,250
(3,5, 4,0]	0,000	0,000	0,000
(4,0, inf)	0,000	0,333	0,125

(a)

acuracia GRT m segmentos			
Histograma de Acurácia			
	lat	buffer	util
Módulo			
(0, 1,0)	0,500	0,333	0,250
1.0	0,000	0,500	0,250
(1,0, 1,5]	0,125	0,000	0,000
(1,5, 2,0]	0,125	0,000	0,000
(2,0, 2,5]	0,000	0,000	0,125
(2,5, 3,0]	0,000	0,000	0,000
(3,0, 3,5]	0,125	0,000	0,250
(3,5, 4,0]	0,000	0,000	0,000
(4,0, inf)	0,125	0,167	0,125

(b)

acuracia GRTE 2 segmentos			
Histograma de Acurácia			
	lat	buffer	util
Módulo			
(0, 1,0)	0,375	0,200	0,250
1.0	0,375	0,700	0,000
(1,0, 1,5]	0,042	0,000	0,042
(1,5, 2,0]	0,042	0,000	0,042
(2,0, 2,5]	0,000	0,000	0,000
(2,5, 3,0]	0,000	0,000	0,000
(3,0, 3,5]	0,000	0,000	0,000
(3,5, 4,0]	0,000	0,000	0,000
(4,0, inf)	0,167	0,100	0,667

(c)

acuracia GRTE m segmentos			
Histograma de Acurácia			
	lat	buffer	util
Módulo			
(0, 1,0)	0,292	0,150	0,375
1.0	0,542	0,750	0,000
(1,0, 1,5]	0,000	0,000	0,000
(1,5, 2,0]	0,042	0,000	0,042
(2,0, 2,5]	0,000	0,000	0,000
(2,5, 3,0]	0,000	0,000	0,000
(3,0, 3,5]	0,000	0,000	0,000
(3,5, 4,0]	0,000	0,000	0,000
(4,0, inf)	0,125	0,100	0,583

(d)

fidelidade GRT			
Histograma de Fidelidade			
	lat	buffer	util
Módulo			
(0, 1,0)	0,25	0,33	0,25
1.0	0,25	0,50	0,63
(1,0, 1,5]	0,13	0	0,00
(1,5, 2,0]	0,00	0	0,00
(2,0, 2,5]	0,00	0	0,00
(2,5, 3,0]	0,00	0	0,00
(3,0, 3,5]	0,13	0	0,00
(3,5, 4,0]	0,00	0	0,00
(4,0, inf)	0,25	0,17	0,13

(e)

fidelidade GRTE			
Histograma de Fidelidade			
	lat	buffer	util
Módulo			
(0, 1,0)	0,08	0,00	0,46
1.0	0,75	0,90	0,46
(1,0, 1,5]	0,13	0	0,04
(1,5, 2,0]	0,00	0	0,00
(2,0, 2,5]	0,00	0	0,00
(2,5, 3,0]	0,00	0	0,00
(3,0, 3,5]	0,00	0	0,04
(3,5, 4,0]	0,00	0	0,00
(4,0, inf)	0,04	0,05	0,00

(f)

Quadro C.2: Conjunto de tabelas com histogramas de acurácia e fidelidade para GRT e GRTE de 2 e de múltiplos segmentos para comparação entre parâmetros.

Referência Bibliográfica

(ALDWORTH, 1999) ALDWORTH, P.J. **System-on-a-chip bus architecture for embedded applications**. International Conference on Computer Design, 1999. (ICCD '99), 297 - 298 .

(ARM, 1999) Advanced Risc Machines. **AMBA Specification Rev 2.0**. 1999. Disponível em <<http://www.arm.com>>. Acesso em 05 de maio 2007.

(BACCELLI et al., 1992) BACCELLI, F.; COHEN, G.; OLSDER, G. J. ; QUADRAT J.-P. **Synchronization and Linearity, An Algebra for Discrete Event Systems**. John Wiley and Sons, 1992.

(BLUETOOTH, 1999) Bluetooth Special Interest Group. **The Bluetooth Core Specification**, 1999. . Disponível em <<https://www.bluetooth.org/spec/>>. Acesso Junho 2005.

(BOLSENS, 2007) BOLSENS, I. **Programming modern FPGAs**. MPSOC 2006. Disponível em <<http://www.xilinx.com/univ/mpsoc2006keynote.pdf>>. Acesso em 5 de maio de 2007.

(BOUDEC, 1996) LE BOUDEC, J-Y.. **Network Calculus made easy**. Technical Report EPFL-DI 96/218. École Polytechnique Fédérale de Lausanne, 1996.

(BOUDEC, 1998) LE BOUDEC, J.Y. **Application of network calculus to guaranteed service networks**. IEEE Trans on Information theory, 44(3), May 1998.

(BOUDEC; THIRAN, 2001) LE BOUDEC, J-Y.; THIRAN, P. **NETWORK CALCULUS - A Theory of Deterministic Queuing Systems for the Internet**. Lectures on Network and Computer Systems - LNCS 2050, Springer, 2001. Verlag, 1ª edição (Agosto de 2001).

(BRADEN, 1989) BRADEN, R. **Requirements for Internet Hosts -- Communication Layers - RFC1122**. Outubro 1989.

(BRADEN; CLARK; SHENKER, 1994) BRADEN, R. ; CLARK, D.; SHENKER, S. **Integrated Services in the Internet Architecture: an Overview – RFC 1633**, Junho 1994.

(BRAZIL, 2004) Brazil-IP Network. Disponível em <<http://www.brazil-ip.org>>. Acesso em Junho 2005.

(CHAKRABORTY et al., 2003a) CHAKRABORTY, S.; KÜNZLI, S.; THIELE, L.;

HERKERSDORF, A; SAGMEISTER, P. **Performance evaluation of network processor architectures: combining simulation with analytical estimation.** Computer Networks: The International Journal of Computer and Telecommunications Networking; Volume 41 , Issue 5 páginas: 641 – 665 (Abril 2003).

(CHAKRABORTY, 2003b) CHAKRABORTY , S.; **System-Level Timing Analysis and Scheduling for Embedded Packet Processors**; 2003. 196pp.Tese de Doutorado - Swiss Federal Institute of Technology, Zurique, 14 de abril de 2003.

(CHAKRABORTY; GRIES; THIELE, 2002) CHAKRABORTY, S.; GRIES, M.; THIELE, L. **Supporting a low delay best-effort class in the presence of real-time traffic.** Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 45–54, San Jose, CA, September 2002.

(CHAKRABORTY; KUNZLI; THIELE, 2003) CHAKRABORTY, S.; GRIES, M.; THIELE, L. **A general framework for analysing system properties in platform-based embedded system design.** In Design, Automation and Test in Europe (DATE), Munich, Germany, Mar. 2003.

(CHANG et al., 1999) CHANG, H; COOKE, L. R.; HUNT, M; MARTIN, G; MCNELLY, A; TODD, L. **Surviving the SoC revolution: A guide to Platform Based Design**, Kluwer Academic Publishers, Boston , 1ª edição 1999.

(CHANG, 2000) CHANG , C.S. **Garantias de desempenho em redes de comunicação.** Springer-Verlag, New York, 2000.

(CONTI et al., 2004) CONTI, M.; CALDARI, M.; VECE, G.B.; ORCIONI, S.; TURCHETTI, C. **Performance analysis of different arbitration algorithms of the AMBA AHB bus** Design Automation Conference, 2004. Proceedings. 41st, Vol., Iss., 2004 Pages: 618- 621, San diego, EUA, 2004

(CRUZ, 1991a) CRUZ, R. **A calculus for network delay – Part I.** IEEE Trans. Information Theory, 37(1); 114-141, 1991.

(CRUZ, 1991b) CRUZ, R. **A Calculus for network delay, Part II: Network Analysis** IEEE Trans. Information Theory, 37(1); 114-141, 1991.

(CUNNINGHAME-GREENE, 1979) CUNNINGHAME-GREENE, R. **Minimax algebra.** Lecture Note 166 in Economics and Mathematical Systems, New York: Springer-Verlag, 1979.

(DARPA, 1981) Estados Unidos. Defense Advanced Research Projects Agency. **Internet Protocol.** Setembro 1981. Disponível em <<http://www.ietf.org/rfc/rfc791.txt>>. Acesso em 05 de maio de 2007.

(ESA, 1994) European Space Agency. **VHDL Modelling Guidelines**, 1994.

Disponível em <<http://microelectronics.esa.int/vhdl/vhdlpage.html>>. Acesso em 05 de maio de 2007.

(ETHZ, 2005) Eidgenössische Technische Hochschule Zürich – TIK. **A Tool for Design Space Exploration of Network Processor Architectures**, 2005. Disponível em <<http://www.tik.ee.ethz.ch/pisa/variators/expo/documentation.html>>. Acesso em 05 de maio de 2007.

(FENIX, 2004) Projeto Fênix – Disponível em <<http://brazilip.cin.ufpe.br/fenix/index.asp>> . Acesso em Junho 2005.

(FREESCALE, 2001) Freescale Semiconductor. **What is the format of an S-Record?**, 2001. Disponível em <<http://www.freescale.com/webapp/sps/utis/SingleFaq.jsp?FAQ-2792.xml>>. Acesso em 05 de maio de 2007.

(GAISLER, 2002) GAISLER, J. **A portable and fault-tolerant microprocessor based on the SPARC v8 architecture** International Conference on Dependable Systems and Networks, 2002, 23-26 Junho 2002 Page(s):409 - 415

(GAISLER, 2004) GAISLER, J. **LEON2 Processor User's manual v.1.0.29**, Disponível em <<http://www.gaisler.com/manuals>>. Acesso em Junho de 2005.

(GARZON, 2003) GARZON, J. S. E. **Estimativas de desempenho de estruturas de comunicação para projetos de sistemas-sobre-silício**. 2003. 100pp. Dissertação de mestrado. Escola Politécnica da USP.

(GNU, 1991) GNU Project. **GNU General Public License**, 1991. Disponível em <<http://www.gnu.org/copyleft/gpl.html>> . Acesso em 05 de maio de 2007.

(GOERING, 2002) GOERING, R. **Chip value is in its intellectual property**, 2002. Disponível em www.eetimes.com/story/OEG20020614S0100. Acesso em Junho 2005.

(GR, 2007) Gaisler Research. **TSIM ERC32/LEON simulator**, 2007. Disponível em <http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=38&Itemid=56>. Acesso em 05 de maio de 2007.

(GRIES et al., 2003) GRIES, M.; KULKARNI, C.; SAUER, C.; KEUTZER, K. **Comparing analytical modeling with simulation for network processors: A case study**. In Proc. of the Designer's Forum at the 6th Design, Automation and Test in Europe (DATE), Munich, Germany, March 2003.

(GUNAWARDENA, 1999) GUNAWARDENA, J.. **From max-plus algebra to nonexpansive mappings: a nonlinear theory for discrete event systems**. Journal of Theor. Comp. Sci. Vo. 293, 2003. Páginas 141-167. Elsevier Science Publishers Ltd. Essex, UK

(HARPER; KERBYSON; NUDD, 1999) HARPER, J.S.; KERBYSON, D.J.; NUDD, G.R. **Analytical modeling of set-associative cache behavior**, IEEE Transactions on Computers, Oct. Vol. 48. N 10, pp. 1009-1024 1999.

(IBM, 1999) International Business Machines Microelectronics. **The CoreConnect Bus Architecture**, 1999. Disponível em <<http://www-306.ibm.com/chips/products/coreconnect>>. Acesso em 05 de maio de 2007.

(IBM, 2001) International Business Machines. **Core Connect FAQs** - IBM Microelectronics, 2001. Disponível em www-306.ibm.com/chips/techlib/techlib.nsf/techdocs. Acesso em Junho 2005.

(IEEE Std 802.3, 2002a) International Electrical and Electronic Engineers. **IEEE Standard For Information Technology-Telecommunications And Information Exchange Between Systems- Local And Metropolitan Area Networks- Specific Requirements**. IEEE Std 802.3-2002 (Revision of IEEE Std 802.3, 2000 edn), Vol., Iss., 2002.

(IETF, 1989) Internet Engineering Task Force. **Requirements for Internet Hosts -- Communication Layer - RFC 1122**. 1989. Disponível em <<http://rfc.net/rfc1122.html>>. Acesso em 05 de maio de 2007.

(IETF, 1990) Internet Engineering Task Force. **TCP Alternate Checksum Options – RFC 1146**. 1990. Disponível em <<http://rfc.net/rfc1122.html>>. Acesso em 05 de maio de 2007.

(IETF, 1998) Internet Engineering Task Force. **An Architecture for Differentiated Services RFC 2475**, 1998. Disponível em <<http://rfc.net/rfc2475.html>>. Acesso em 05 de maio de 2007.

(ISO, 1994) International Organization for Standardization. **Open Systems Interconnection Basic Reference Model - ISO/IEC 7498**, 1994. Disponível em <<http://www.iso.org/iso/en/ISOOnline.frontpage>>. Acesso em 05 de maio de 2007.

(JERRAYA; WOLF, 2005) JERRAYA, A; WOLF, W. **Multiprocessors Systems-on-chips**, 1ª edição. Morgan Kaufmann. New York, 2005 .

(KRISHNASWAMY; SCHERSON, 2000) KRISHNASWAMY, U.; SCHERSON, I.D. **A framework for computer performance evaluation using benchmark sets**. IEEE Transactions on Computers, Dez. 2000 Vol. 49 Iss. 12, páginas:1325 – 1338 .

(KUNZLI et al., 2006) KÜNZLI, S.; POLETTI, F.; BENINI, L.; THIELE, L. **Combining simulation and formal methods for system-level performance analysis**. Proc. of the conference on Design, automation and test in Europe, 2006. Pp. 236-241, Munich, Germany .

(KUNZLI; THIELE; ZITZLER, 2005) KUNZLI, S.; THIELE, L.; ZITZLER, E. **Modular design space exploration framework for embedded systems** .Computers and

Digital Techniques, IEE Proceedings-, Vol.152, Iss.2, Mar 2005 Pages: 183- 192

(LAHIRI ;RAGHUNATHAN; DEY, 2004) 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, 952- 961, 2004.

(LAHIRI; RAGHUNATHAN; DEY, 2001) LAHIRI, K.; RAGHUNATHAN, A; DEY , S. **System-level performance analysis for designing on-chip communication architectures**, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 768-783, 2001.

(LAHIRI; RAGHUNATHAN; LAKSHMINARAYANA, 2001) LAHIRI, K.; RAGHUNATHAN, A; LAKSHMINARAYANA, G. **LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs** Design Automation Conference, 2001. Proceedings, Vol., Iss., 2001, Pages: 15- 20

(LME, 2007) Laboratório de Microeletrônica. Universidade de São Paulo, 2007. Disponível em <<http://www.lme.poli.usp.br>>. Acesso em 05 de maio de 2007.

(MENTOR, 2007) Mentor graphics. Disponível em <<http://www.mentor.com>>. Acesso em 5 de maio de 2007.

(MIBENCH, 2001) **Mibench LAME MP3 encoder** . Disponível em <<http://www.eecs.umich.edu/mibench/index.html>>. Acesso em Junho 2005.

(MODELSIM, 2005) Mentor graphics. **Modelsim User's Manual version 6.1c**. Publicado 08 de novembro de 2005.

(MOK; CHEN, 1997) MOK, A; CHEN, DEJI. **A multiframe model for real time tasks** IEEE Transactions on *software* Engineering 23(10):635-644, October 1997

(NAEDELE; THIELE; EISENRING, 1999a) NAEDELE, M.; THIELE, L.; EISENRING , M. **Characterizing Variable Task Release and Processor Capacities**. ETH - Swiss Federal Institute of Technology, Zurique, 14 de abril de 2003. Report n. 45, Junho 99.

(NAEDELE; THIELE; EISENRING, 1999b) NAEDELE, M.; THIELE, L.; EISENRING , M. **Characterizing variable task releases and processor capacities** in Proceedings of the 14th IFAC World Congress, (Peking, China), pp. Q251--Q256, 1999

(NLANR, 2005) National Laboratory for Applied Network Research. **Internet Measurement, Internet Analysis**, 2005. Disponível em <<http://moat.nlanr.net/>>. Acesso em 05 de maio de 2007.

(NLANR, 2007) Estados Unidos. **National Laboratory for Applied Network**

Research (NLANR). Disponível em <<http://www.nlanr.net/>>. Acesso em 05 de maio de 2007.

(NOONBURG ; SHEN, 1997) NOONBURG, D.B.; SHEN , J.P. **A framework for statistical modeling of superscalar processor performance** Third International Symposium on High-Performance Computer Architecture, 1-5 Feb. 1997 página(s):298 - 309 .

(OLUKOTUN; HEINRICH; OFELT, 1998) OLUKOTUN , K.; HEINRICH, M.; OFELT, D. **Digital system simulation: methodologies and examples**. Design Automation Conference, Jun 1998 página(s):658 - 663 .

(OPENSARC, 2006) Open Sparc. **Open Sparc T1**, 2006. Disponível em <<http://opensparc-t1.sunsource.net/index.html>>. Acesso em 05 de maio de 2007.

(PANOS, 2003) PANOS, L.K. **Network Processors : Architectures, Protocols and Platforms** . 456 p. McGraw-Hill Professional; 1 edition, July 28, 2003.

(RIGO et al., 2004) RIGO, S; ARAUJO, G; BARTHOLOMEU , M; AZEVEDO, R. **ArchC: A SystemC-Based Architecture Description Language** ,16th Symposium on Computer Architecture and High Performance Computing (SBAC'04). Foz do Iguaçu - Brasil, Outubro de 2004.

(RYU; SHIN; MOONEY, 2001) RYU, K. K; SHIN, E.; MOONEY , V.J. **A comparison of five different multiprocessor SoC bus architectures** Digital Systems, Design, 2001. Proceedings. Euromicro Symposium on, Vol., Iss., 2001, 202 – 209.

(SAKAMOTO et al., 2003) SAKAMOTO, M.; KATSUNO, A; INOUE, A; ASAKAWA, T.; UENO, H.; MORITA, K.; KIMURA, Y. **Microarchitecture and performance analysis of a SPARC-V9 microprocessor for enterprise server systems** The Ninth International Symposium on High-Performance Computer Architecture, HPCA-9; 8-12 Feb. 2003 página(s):141 - 152 .

(SHENKER; WROCLAWSKI, 1997) ref SHENKER , S.; WROCLAWSKI, J.; **General characterization parameters for integrated service network elements**. Request for Comments 2215, Internet Engineering Task Force (IETF), September 1997.

(SPARC, 1991) Sparc International. SPARC V8 Specification, 1991. Disponível em <<http://www.sparc.org>> . Acesso em 05 de maio de 2007.

(SPARC, 2005) Sparc International. SPARC V9 Specification, 2005. Disponível em <<http://www.sparc.org>> . Acesso em 05 de maio de 2007.

(SYSTEMC, 2001) The Open SystemC Initiative. **SystemC Specification**. Disponível em <<http://www.systemc.org>>. Acesso em Junho 2005.

(THIELE et al., 2001) THIELE, L.; CHAKRABORTY, S.; GRIES, M.; MAXIAGUINE,

A; GREUTERT, J.. **Embedded software in network processors - models and algorithms.** In First Workshop on Embedded *software*, Lecture Notes in Computer Science 2211, pages 416--434, Lake Tahoe, CA, USA, 2001. Springer Verlag.

(THIELE et al., 2002a) THIELE,L.; CHAKRABORTY, S.; GRIES, M.; KUNZLI , S.. **A framework for evaluating design tradeoffs in packet processing architectures.** In Proc. 39th Design Automation Conference (DAC), New Orleans, USA, 2002. ACM Press.

(THIELE et al., 2002b) THIELE,L.; CHAKRABORTY, S.; GRIES, M.; KUNZLI , S. **A framework for evaluating design tradeoffs in packet processing architectures.** In Proc. 39th Design Automation Conference (DAC), New Orleans, USA, 2002. ACM Press.

(THIELE et al., 2002b) THIELE,L.; CHAKRABORTY, S.; GRIES, M.; KUNZLI , S.. **Design Space Exploration of Network Processor Architectures** First Workshop on Network Processors at the 8th International Symposium on High Performance Computer Architecture (HPCA8), Cambridge MA, USA, February, 2002.

(THIELE; CHAKRABORTY; NAEDELE, 2000) THIELE,L.; CHAKRABORTY, S.; NAEDELE , M.. **Real-time calculus for scheduling hard real-time systems.** In Proc. IEEE International Conference on Circuits and Systems, 2000. -]

(WANDELER; MAXIAGUINE; THIELE, 2006) WANDELER, E; MAXIAGUINE, A; THIELE, L. **Performance analysis of Greedy Shapers in Real-Time systems.** Proceedings of the conference on Design, automation and test in Europe (DATE) 2006. Páginas 444-449. Munique, Alemanha.

(WOLF, 2002) WOLF , W. **How many system architectures?** Trans. on Computer. Páginas 93 a 95. vol. 36 Iss. 3 2003.

(YI; LILJA; HAWKINS, 2003) YI,J.J.; LILJA, D.J.; HAWKINS , D.M. **A statistically rigorous approach for improving simulation methodology,** The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003, 281- 291.