Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Análise de Desempenho de Topologias de Redes em Chip(NoC)

Dino Macedo Amaral

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Engenharia Elétrica

Orientador
Prof. PhD. Alexandre Soares Romariz

Brasília
2008

Universidade de Brasília – UnB
Faculdade de Tecnologia
Departamento de Engenharia Elétrica
Mestrado em Engenharia Elétrica

Coordenador: Prof. Dr. Marco Antônio Brasil Terada

Banca examinadora composta por:

Prof. PhD. Alexandre Soares Romariz (Orientador) – Ene/UnB
Prof PhD. José Camargo – Ene/UnB
Prof Dr. Pedro de Azevedo Berger – CIC/UnB

CIP – Catalogação Internacional na Publicação

Amaral, Dino Macedo.

Análise de Desempenho de Topologias de Redes em Chip(NoC) / Dino Macedo Amaral. Brasília : UnB, 2008.
98 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2008.

1. Micro-eletrônica, 2. Network-on-Chip, 3. Modelo Analítico para Roteamento, 4. Throughput, 5. Latência, 6. Roteamento Wormhole

ENE/FT/UNB

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Análise de Desempenho de Topologias de Redes em Chip(NoC)

Dino Macedo Amaral

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Engenharia Elétrica

Prof. PhD. Alexandre Soares Romariz (Orientador)
Ene/UnB

Prof PhD. José Camargo
Ene/UnB

Prof Dr. Pedro de Azevedo Berger
CIC/UnB

Prof. Dr. Marco Antônio Brasil Terada
Coordenador do Mestrado em Engenharia Elétrica

Brasília, 22 de Fevereiro de 2008

Dedicatória

A Deus que tudo criou e que me dá forças para tudo realizar.

A minha esposa Suzanne e à prole.... ao Lucas e à Mel, pela compreensão e paciência nos momentos que me mantive ausente para a concretização deste sonho.

À minha família que mesmo longe torcem incansavelmente pelo meu sucesso.

Agradecimentos

Ao meu professor e orientador Romariz e ao Professor Camargo, por ter acreditado em mim.

Aos colegas que me ajudaram nesta etapa da minha vida, Genival, Fabrício, Gilmar, Letícia, Vítor, Peotta e à tantos outros que estiveram nas salas de aulas neste período.

A meus colegas de trabalho da Unidade, ops... Diretoria de Gestão da Segurança que me apoiaram e me ajudaram neste árdua caminhada.

Resumo

A necessidade de atender as demandas existentes no mercado de microeletrônica tem levado os projetistas a compactar um grande número de blocos IP's, o que produz uma diversidade enorme em suas funcionalidades. Do ponto de vista prático, a distribuíção destes blocos IP's torna-se um problema devido aos problemas físicos como alta impiedância devido ao número de fios que os interligam, o gasto de energia para manter todos os blocos IP's se comunicando, e uma ocupação otimizado da área do chip. Para ajudar os projetistas de *SoC*, os conceitos usado em rede de computadores têm sido a principal fonte para apontar a uma soluçào possível para estas situações. Este documento mostra os resultados apresentados usando o gpNoCsim [18] e o modelo analítico mostrado em [12], o que pode ajudar os projetistas de *NoC* encontrar possíveis gargalos quando for trabalhar com *NoCs*.

Palavras-chave: Micro-eletrônica, Network-on-Chip, Modelo Analítico para Roteamento, Throughput, Latência, Roteamento Wormhole

Abstract

The need to meet the existing demands in the microeletronic market has prompted designers to compact a big number of IP blocks in a small silicon area. From the pratical point of view, the distribuition of these IP blocks becomes a issue due to physical issues like high impedance caused by the number of wires that interconncet them, the power consupmtion to keep all IP blocks comunicating. An optimized occupation of the whole space used by the chip. In order to help the SoC designers , the concepts used in networking have been the main source to point out a possible solution for these situations. This paper shows the results of a benchmarck using gpNoCsim [18], which can help the NoC designers to find the bottlenecks when working with NoCs.

Keywords: Microelectronics, Network-on-Chip, Analytical Model, Throughput, Latency, Wormhole Routing

Conteúdo

Lista de Figuras

Tabela de Símbolos

IP - Intellectual Property
NoC - Network-on-Chip
gpNoCSim - General Purpose Network-on-Chip Simulator
SoC - System on Chip
AMBA - Advanced Microcontroller Bus Architecture
AHB - Advanced High-performance Bus
APB - Advanced Peripheral Bus
ASB - Advanced Serial Bus
AXI - Advanced eXtensible Interface
OSI - Open Systems Interconnection
RTAI - Real Time Application Interface
BFT - Butterfly Fat Tree
ER - Elemento de roteamento
EP - Elemento de processamento
ITRS - International Technology Roadmap for Semiconductors
GALS - Globally-Asynchronous Locally-Synchronous)
ISA - Industry Standard Architecture
PCI - Peripheral Component Interconnect
TDM - Time-Division Multiplexing
TCP/IP - Transmission Control Protocol / Internet Protocol
IPX/SPX - Internetwork Packet Exchange/Sequenced Packet Exchange
NIU - Network-on-Chip Interface Unit
SRAM - Static Random Access Memory
DRAM - Dynamic Random Access Memory
BE - Best Effort
GT - Guaranteed hroughput
OCP - Open Core Protocol
VCI - Virtual Component Interface
BVCI - Basic Virtual Component Interface
AVCI - Advanced Virtual Component Interface
VSIA - Virtual Socket Interface Alliance
UTP - Unshielded Twisted Pair
W - Tempo de Espera
x - Tempo de Serviço
D - Número de Canais
λ - Taxa de Mensagem
s - Tamanho da Mensagem
f - Tamanho do Flit (na mensagem)

Capítulo 1

Introdução

1.1 Motivação

Segundo estudos do ITRS (*www.itrs.net*) , nos próximos 15(quinze) anos será possível a integração de bilhões de transistores e centenas de núcleos em uma mesma pastilha de silício. Para uma integração destes bilhões de transistores, torna-se imperativo uma abordagem minuciosa . Vários fatores deverão ser levados em consideração para uma melhor performance do chip fabricado.

As razões que motivam a procura por uma nova abordagem de integração envolvem, além de outras, questões financeiras e tecnológicas. No que tange às financeiras, existe uma pressão do mercado para que os chips fabricados possuam uma estrutura reutilizável e padronizada, o que favorece a diminuição do *time-to-market* e permite a construção de chips com maior rapidez, fato que impacta no custo da construção do mesmo. Quanto às questões tecnológicas, os quais abordam os aspectos físicos na construção do chip, o consumo de energia surge como fator limitante para dispositivos sem fio e existe uma busca incessante, por partes dos projetistas de otimizar este recurso da melhor maneira possível. Os componentes reutilizáveis são denominados núcleos (blocos de propriedade individual) e podem ser desenvolvidos pela empresa responsável pelo projeto do sistema ou de empresas terceirizadas.

Os blocos de um sistema deverão ser interligados através de uma estrutura de canais denominada arquitetura de comunicação, ou rede de interconexão. Quanto a conexão que essas redes necessitam para efetivar a comunicação, o uso de conexões ponto-a-ponto com todos os componentes internos, pois uma malha de fios que conecte cada componente interno resultaria em uma capacitância maior, conseqüentemente em um atraso maior na movimentação de dados ao longo dos fios, e limitaria o desempenho do sistema. O consumo de energia está atrelado a freqüência , temos os chips com uma topologia de barramento operam por difusão, com cada sinal chegando a todos os pontos do barramento, exigindo grande quantidade de energia.

Diante de tantas exigências, surgem alguns questionamentos: como os chips,

com um arquitetura de comunicação em rede, serão projetados e como os blocos estarão organizados dentro desses chips ?

Em uma abordagem particular os blocos internos podem estar interligados em rede, em analogia às tradicionais redes de computadores, com conceitos já solidificados e que serviriam para ajudar a embasar a teoria que cerca o assunto. A solução proposta pela comunidade científica está no uso de uma redes de interconexão chaveada, como aquelas encontradas em computadores paralelos. Podemos ressaltar algumas vantagens nessa abordagem como modularidade, economia de energia, o uso de conexões ponto-a-ponto curtas, paralelismo de comunicação e performance [35]. Há alguns pontos, porém a serem observados com maior cuidado, tais como maiores custos e latência na comunicação. Vários termos têm sido designados para descrever essa nova abordagem: *Micronetworks*, *On-Chip Networks* (OCN) e *Network-on-Chip (NoC)*, todas elas se referem à mesma arquitetura, sendo que o termo *NoC* tem tido uma maior aceitação por parte da comunidade científica.

1.2 Visão geral deste documento

O objetivo desta dissertação é abordar uma estratégia de arquitetura para circuitos integrados : *NoC*. Para iniciar um projeto de circuito integrado, faz-se necessário a busca de simuladores, que possa mostrar situações semelhantes àquelas que os chips serão projetados. Para ajudar nesta tarefa, buscamos um modelo analítico de roteamento *wormhole*[12], amplamente usado em arquitetura em *NoC*, e confrontamos com os resultados obtidos em nossas simulações. O simulador escolhido foi o gpNoCSim [18] por possuir código-fonte disponível para alterações que se acharem necessárias e por implementar 4 (quatro) topologias diferentes (Torus, Mesh, Butterfly-Fat Tree, Extended Butterfly-Fat Tree).

Após a comparação ser realizada, iniciaremos simulações nos itens concernentes a Redes em Chip (NoC): como número de blocos IP´s (Intellectual Property), tamanho da mensagem e tamanho do buffer. Com os resultados obtidos dessas simulações, coletaremos dados que nos ajudarão a apontar uma topologia que apresente resultados considerados satisfatórios, e poderemos assim iniciar estudos sobre NoC.

Este documento está organizado da seguinte forma: no próximo capítulo serão mostradas diferenças entre uma topologia em rede e em barramento e conceitos essenciais de *NoC*; no capítulo 3 (três), mostraremos algumas implementações que se valeram dos conceitos de redes macroscópicas e as usaram em redes microscópicas; no capítulo 4 (quatro), apresentaremos o modelo analítico descrito em [12]; no capítulo 5 (cinco), o gpNoCsim é mostrado com detalhes de configuração, para simularmos as diferentes situações; no capítulo 6(seis), compararemos os resultados da simulação com os fornecidos do modelo analítico, na busca por uma arquitetura mais eficiente para situações que suportem altas taxas de

injeção de mensagens na rede; no capítulo 7(sete) iremos propor trabalhos futuros e conclusões de nossa pesquisa.

1.3 Tendências Tecnológicas para Circuitos Integrados

Na literatura corrente, temos algumas situações que foram abordadas para os chips na topologia em rede:

- Os chips serão projetados usando módulos pré-existentes, como processadores, memórias, controladores. As metodologias de projetos terão suporte a componentes reutilizáveis com interfaces de conexões padronizáveis. Com o intuito de atender as demandas, os chips deverão ter uma estrutura modular e escalonável [42] [28] ;

- Apenas possuir funcionalidades variadas não é suficiente, o objetivo do projeto deverá satisfazer algumas métricas de *QoS(Quality of Service)* com o menor consumo de energia possível, não prejudicando os requisitos de latência e *throughput* na rede em questão;

- Devido aos atrasos dominantes que teríamos na rede, uma única fonte de freqüência de *clock* pode se tornar inviável para atender aos diversos dispositivos, sendo o paradigma GALS [24] o mais aceito, na qual o chip usa uma freqüência síncrona localmente e para o chip com um todo usa uma freqüência assíncrona. Em [13] é mostrado o problema de links de comunicação cujo tamanho do mesmo torna impossível alcançar o destino com um único ciclo de clock ;

- Como itens a serem analisados, a área ocupada e consumo de energia, apresentam aspectos tidos como relevantes na construção de chips de qualquer natureza, em [31], Meloni at al. apresenta um modelo analítico para cálculos de ocupação de área e consumo de energia de *switches NoC*;

- Com algoritmos de roteamento otimizados é possível obter uma economia maior de energia, devido a uma movimentação menor de tráfego dentro do chip. Por isso a questão do roteamento é de vital importância nesta arquitetura em rede, então os conceitos de redes macroscópicas serão utilizados para as redes microscópicas como mostrados em [33] [10] [21] [3] [20] [47]

- A tecnologia de desenvolvimento das ferramentas de compilação e sintetização não acompanha o mesmo ritmo da indústria de circuitos integrados, então o uso de simulador para NoC torna-se um aliado para agilizar tomadas de decisões no que tange a estrutura do chip a ser projetado ;

Capítulo 2

Conceitos Teóricos

Estudos têm mostrado a viabilidade e vantagens de *NoC* sobre as arquiteturas de barramento. Neste capítulo, mostraremos as vantagens de uma topologia em rede para *chips* , possíveis comparações com a arquitetura em barramento e modelos conceituais que nos ajudem a elucidar esta abordagem.

2.1 Tendências da arquitetura para *chips*

Algumas tendências têm forçado uma evolução da arquitetura de sistemas, com o intuito de absorver as necessidades que o mercado pleiteia. Cabe salientar que esta evolução não acontece de maneira repentina e sim de maneira gradual, sendo apresentadas possíveis soluções para as demandas existentes. Abaixo enumeraremos algumas situações descritas em [2] e [5], que forçam uma busca por uma arquitetura que atenda às exigências dos próximos *chips*:

- Convergência das aplicações : uma variedade de tipos de tráfego, em alta velocidade, no mesmo projeto de SoC. Esses tipos de tráfego, embora bem diferentes em suas naturezas, do ponto de vista da Qualidade de Serviço (QoS), devem compartilhar o mesmo meio e ter diferentes prioridades no canal que trafégam;
- O lei de Moore leva a uma integração de um grande número de blocos IP em um único *chip*;
- A evolução do processo de silício no decorrer do tempo mostra que as unidades lógicas apresentam uma vantagem em relação aos fios, do ponto de vista de área e performance;
- O *time-to-market* exige que tenhamos blocos com interfaces padronizadas e reutilizáveis, o que exige uma interação entre as empresas fornecedoras destes blocos ou uma padronização por parte de um órgão regulador, sabemos que não é uma tarefa das mais fáceis.

Embora tenha variado em sua arquitetura visando atender às situações acima, a arquitetura em barramento encontrou alguns conflitos de compatilibidade. Em muitos casos, a introdução de novas características exigiu muitas mudanças na implementação de arquitetura em barramentos, mas principalmente nas interfaces(AMBA ASB para AHB2.0, e depois AMBA AHB-Lite e AMBA AXI) com maior impacto na reusabilidade dos blocos IP's e no novo modelo dos mesmos. A arquitetura em barramento não distribui as atividades destinadas às camadas de transação, transporte e física (serão explicadas com detalhes mais adiante), então a mesma não se adapta às mudanças e não tira vantagens dos avanços acontecidos nos processos de tecnologia de construção de *chips*.

Mediante estas situações, uma comparação com as tradicionais redes de computadores é inevitável, já que as mesmas lidam, com sucesso, com problemas similares às arquiteturas em barramento. Fazendo uso do conceito de modelo OSI, com cada camada realizando a sua atividade e encaminhando para a camada acima ou abaixo a tarefa correspondente, encontramos uma enorme flexibilidade, podemos usar de diversas aplicações com o mesmo modelo.

Os *chips* quando dispostos em rede estão fornecendo uma alternativa atraente para a construção dos *chips*, a implementação deste tipo de disposição acontecerá quando :

- Reduzir o custo de fabricação;
- Aumentar o desempenho;
- Reduzir o *time-to-market*;
- Reduzir o risco para projetar novos *chips*.

2.2 Arquitetura em barramento

Os exemplos típicos de arquitetura em barramento são ISA e PCI usados nos computadores. ISA tornou-se um padrão na década de 80, e deixou de ser produzido há pouco tempo para as placas de computadores pessoais. O barramento PCI funciona com o princípio TDM, onde o barramento *master* deve solicitar a utilização do barramento a uma unidade que arbitra o acesso ao barramento principal. Neste momento o barramento fica ocupado pelo período de número de ciclos de *clock* que a transferência acontece, durante este período nenhum dispositivo pode acessar o barramento. A existência de múltiplos *masters* faz com os mesmos exerçam uma competição entre eles para acessar o barramento, os *masters* devem receber autorização de uma unidade que decida a vez que cada um terá acesso ao barramento. Um maneira de aumentar o *throughput* do barramento PCI é dividir em domínio de barramentos que se comunicam através de pontes. O barramento faz uma conexão ,no caso uma ponte, com 2 (dois) ou

mais barramentos PCI, o que permite que um *master* possa acessar um outro barramento ou outro dispositivo, dentro do domínio que foi criado.

Na figura 2.1 temos um exemplo de divisão em AHB e APB, a razão desse fato é que alguns periféricos não necessitam de altas velocidades para se comunicarem. Então, a economia de energia nessa parte do *chip* torna-se um dos fatores para esta divisão de periféricos.

A arquitetura em barramento não afeta o desempenho quando o número de portas conectadas é pequena, mas torna-se um problema quando o número cresce. O primeiro problema é que a unidade que gerencia os acessos ao barramento principal deve estar preparado para possui conexões para os dispositivos que possuem a possibilidade de transferir dados, ou seja ser um *master*. Esta unidade gerenciadora torna-se um gargalo quando temos muitas conexões querendo ter acesso ao barramento. Outro problema é o meio compartilhado que é usado para transferir dados, não é difícil perceber que existe um limite para envio de dados em qualquer período de tempo, e quando o número de transferência aumenta, a largura de banda disponível diminui.

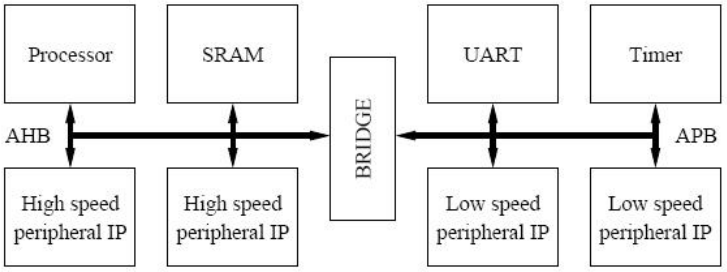

Figura 2.1: Barramento AMBA AHB/APB

2.3 A Arquitetura em *NoC*

Como o resultado de um crescente número de integração de blocos IP´s em um *chip*, várias pesquisas têm focado em desenvolver uma infra-estrutura eficiente de comunicação para o *chip* como um todo. Novas estruturas de interconexão estão sendo pesquisadas pela comunidade científica para projetos comerciais que possa agregar um grande número de blocos IP's em um único SoC. A proposta apresentada em [30], [35] e [23] usa a tecnologia de projeto de redes para analisar e projetar SoCs. A nova metodologia permitirá a conexão de vários núcleos, o que muda o foco dos projetos de desenvolvimento de SoC's para uma arquitetura de comunicação que forneça uma intensa troca de informações entre os núcleos componentes de SoC.

Existem muitos projetos de *SoC* que contém múltiplos processadores para aplicações como estações base de redes sem fio, HDTV, processamento de imagens, aparelhos sem fio. Seguindo as novas tendências nas arquiteturas de comu-

nicação para *chips*, *SoC´s* com múltiplos processadores podem ser construídos com estruturas de interconexões diferentes mas regulares, visando obter proveito de conceitos de computação paralela. Os fabricantes destas propostas de interconexões de comunicações estão procurando achar um ponto de equilíbrio entre pontos considerados essenciais na fabricação de *chips*, como latência, confiabilidade, dissipação de energia, *throughput* e área.

Um projeto de *SoC* complexo pode ser visto com uma micro-rede com múltiplos blocos IP´s, e os modelos e técnicas de redes e computação paralelas podem ser aplicadas a este contexto. Para a *NoC*, a micro-rede deve atender aos requerimentos de qualidade de serviço como confiabilidade e garantias de latência e largura de banda, e um consumo de energia que proporcione a viabilidade da implementação. As bibliotecas para *NoC*, incluindo roteadores, interfaces e as interconexões fornecerão, aos projetistas, componentes flexíveis para a construção de núcleos de processamento e armazenamento. O grau de maturidade destas bibliotecas irão influenciar na utilização das mesmas para auxiliar as ferramentas de otimização e sintetização.

Embora os projetos para *NoC´s* se baseie em alguns aspectos de computação paralela, eles são direcionados por um conjunto significante de diferenças. Do ponto de vista de desempenho, alto *throughput* e baixa latência são características desejáveis de plataformas de *SoC* multi-processadas. A questão de dissipação de energia na arquitetura par *NoC* é de grande importância, pois representa uma porção significante do total da energia a ser consumida pelo *chip*. A área ocupada pelas interconexões do *chip* é muito importante, as características deste tipo de arquitetura é que os blocos de processamento e armazenamento se comunicam entre si através de conexões de alto desempenho e *switches* inteligentes. Com base nesta afirmação, o projeto de comunicação pode ser extraído de projetos de redes computadores e ser representado com um alto nível de abstração.

A troca de dados entre blocos de processadores e armazenamento tem se tornado uma tarefa difícil com o crescimento de tamanho do sistema e os atrasos que acontecem dentro do *chip*. Para mitigar esta situação, a comunicação ponto-a-ponto necessita de ser dividida em múltiplos estágios de *pipeline*.

NoC fornece uma alternativa para as interconexões dos blocos IP´s, pois :

- (a) a rede estrutura e gerencia as interconexões em uma nova tecnologia;
- (b) compartilham as interconexões, diminuindo o número das mesmas e aumentando a sua utilização;
- (c) possui um consumo menor de energia e mais confiabilidade;
- (d) com a sua estrutura em camadas possui um grau de escalabilidade maior que as arquiteturas em barramento.

Na figura 2.2 temos uma estrutura de *NoC*, que consiste de blocos IP's (na figura mostrado como Resource) e *switches* que são conectados usando canais

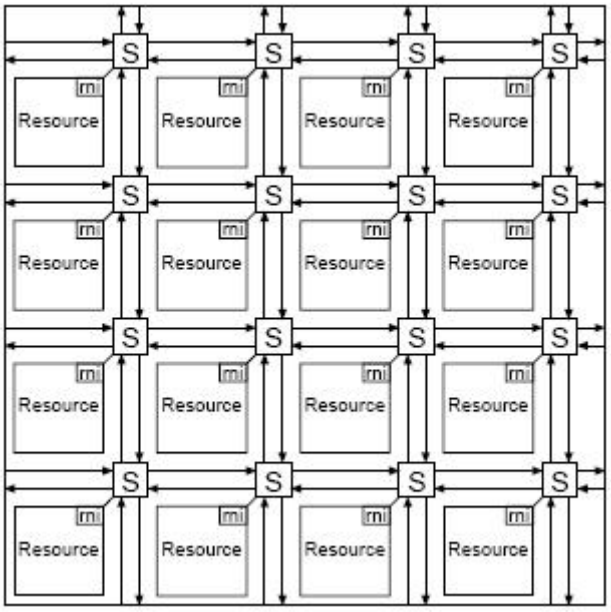

Figura 2.2: NoC com 16 blocos IP´s

como *mesh*, que permite a comunicação com cada integrante do *chip* através do envio de mensagem. O bloco IP (R) é uma unidade computacional ou de armazenamento, ou uma combinação de ambos. O *switch* S roteia e *buferiza* mensagens entre os blocos IP's, está conectado aos 4 (quatro)*switches* vizinhos através dos canais de entrada e saída. O canal C consiste de 2(dois) ligações ponto-a-ponto entre 2 (dois) *switches* ou entre um bloco IP e um *switch*

Com uma comunicação confiável entre os blocos IP´s, o que permite que a perda de dados seja minimizada dentro do *chip*, a *NoC* deve ser livre de "deadlock"[8], assegurando que um bloco IP não retenha nenhum recurso indefinidamente enquanto espera por outro recurso. A *NoC* deve trabalhar com os dados ordenados, o que dimimuiria o espaço do *buffer* e eliminaria a possibilidade de ordená-los, gastando menos processamento para tal tarefa. Enfim , temos várias características que fazem da *NoC* uma alternativa mais vantajosa em relação a arquitetura de barramento, para determinadas situações.

2.4 As camadas para um *NoC*

2.4.1 Modelo OSI

O modelo OSI [44], abordado nesta seção, promove um amplo debate de como a comunicação entre 2 (dois) dispositivos acontece, com uma clara divisão de funções. Vale afirmar, que a não implementação deste modelo tal qual o mesmo é concebido não prejudica a comunicação. Um exemplo desta abordagem ocorre

com as empresas de comunicação da internet, como a Cisco (*www.cisco.com*) que adotaram o modelo próprio de comunicação com 3 (três) camadas (Camada de Acesso, Camada de Distribuição e Camada de Núcleo). Então, podemos afirmar que o modelo OSI possui uma divisão bem ampla nas funções de comunicação.

Quando lidamos com sistemas de comunicação, é necessário uma divisão rígida de funções para que a mesma aconteça de maneira modular, flexível e com desempenho satisfatório. O conceito de camadas consiste de um conjunto de funcionalidades definidas para permitir a eficiência da comunicação. Com cada camada sabendo exatamente o que fazer, a divisão de tarefas confere a este modelo uma modularidade que facilita implementações e diminue o *time-to-market*.

Figura 2.3: Modelo OSI de camadas

As camadas são feitas de entidades, que podem ser um hardware ou software. Uma entidade se comunica com outra entidade da mesma camada, mas em outra localização na rede. O modelo em camadas chamado Modelo OSI (*Open Systems Interconnection*) descrito em [44], não é um protocolo, mas um conjunto de instruções de como implementar um modelo eficiente de redes de computadores, com cada camada mostrando a sua função na cadeia da comunicação. Muitas outras considerações poderiam ser feitas sobre o modelo OSI, mas pelo contexto do assunto não haveria importância. Abaixo segue uma descrição das funções de cada camada mostrada na Figura 2.3:

1. Camada Física : O sinal que vem do meio (Cabos UTP por exemplo), chega à camada física em formato de sinais elétricos e se transforma em bits (0 e 1). Esta camada lida com as características mecânicas e funcionais para acessar o meio físico;

2. Camada de Enlace : Fornece uma transferência confiável de dados através da camada física. Após o recebimento dos bits, ela os converte de maneira

inteligível, os transforma em unidade de dado, subtrai o endereço físico e encaminha para a camada de rede que continua o processo;

3. Camada de Rede : A camada 3 é responsável pelo tráfego no processo de *internetworking*. A partir de dispositivos como roteadores, ela decide qual o melhor caminho para os dados no processo, bem como estabelecimento das rotas;

4. Camada de Transporte : A camada de transporte é responsável pela qualidade na entrega/recebimento dos dados. Após os dados já endereçados virem da camada 3, é hora de começar o transporte dos mesmos. A camada 4 gerencia esse processo, para assegurar de maneira confiável o sucesso no transporte dos dados ;

5. Camada de Sessão : Fornece um controle da estrutura para comunicação entre as aplicações. Estabelece, gerencia e termina conexões entre as aplicações. Para obter êxito no processo de comunicação, a camada de sessão tem que se preocupar com a sincronização entre *hosts*, para que a sessão aberta entre eles se mantenha funcionando;

6. Camada de Apresentação : Fornece a formatação dos dados, e da representação destes, é a camada responsável por fazer com que duas redes diferentes (por exemplo, uma TCP/IP e outra IPX/SPX) se comuniquem, "traduzindo"os dados no processo de comunicação.;

7. Camada de Aplicação : A camada de aplicação e a que mais notamos no dia a dia, pois interagimos direto com ela através de softwares como cliente de correio, programas de mensagens instantâneas, etc.

2.4.2 Estudo de Caso : Estrutura de Camadas proposta pela Arteris

A implementação de um modelo em camada pode variar, mas a idéia central é simplificar as tarefas que envolvem a comunicação na rede, conferindo a cada componente envolvido uma função . Na figura abaixo, temos o modelo implementado pela Arteris [2], que mostra apenas 3 camadas sendo utilizadas, a saber :

- Camada de transação: Define a comunicação primitiva disponível para interconectar os blocos IP's. *NoC Interface Unit* (NIUs) fornece serviços para camada de transação aos blocos IP's aos quais eles estão conectados;

- Camada de transporte : Define as regras que devem ser aplicadas aos pacotes para um encaminhamento dos mesmos, é o roteamento propriamente dito;

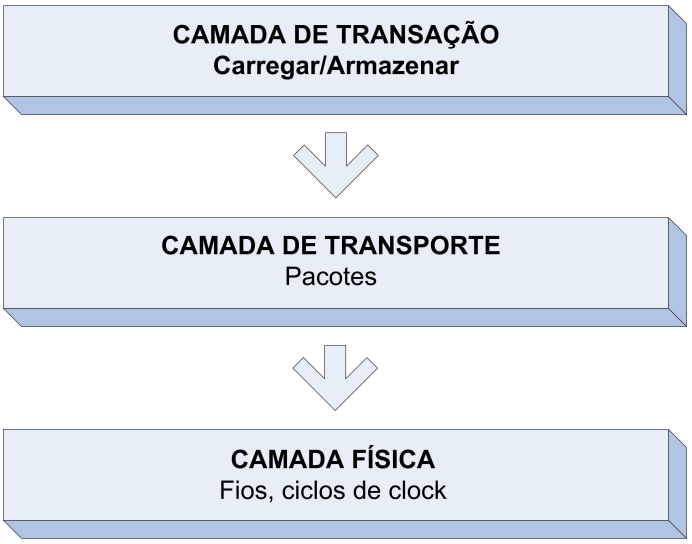

Figura 2.4: Modelo de camadas proposto pela Arteris

- Camada Física define como os pacotes serão fisicamente transmitidos na interface, como o padrão Ethernet que define a taxa de transmissão de 10Mb/s, 1Gb/s.

Na Figura 2.5 temos uma ilustração da arquitetura em camada proposto pela Arteris [2], fundada em 2003 com sede em Paris (França) e San Jose (Califórnia-EUA), foi a primeira a oferecer uma solução comercial de *NoC* para projetistas de *chip*.

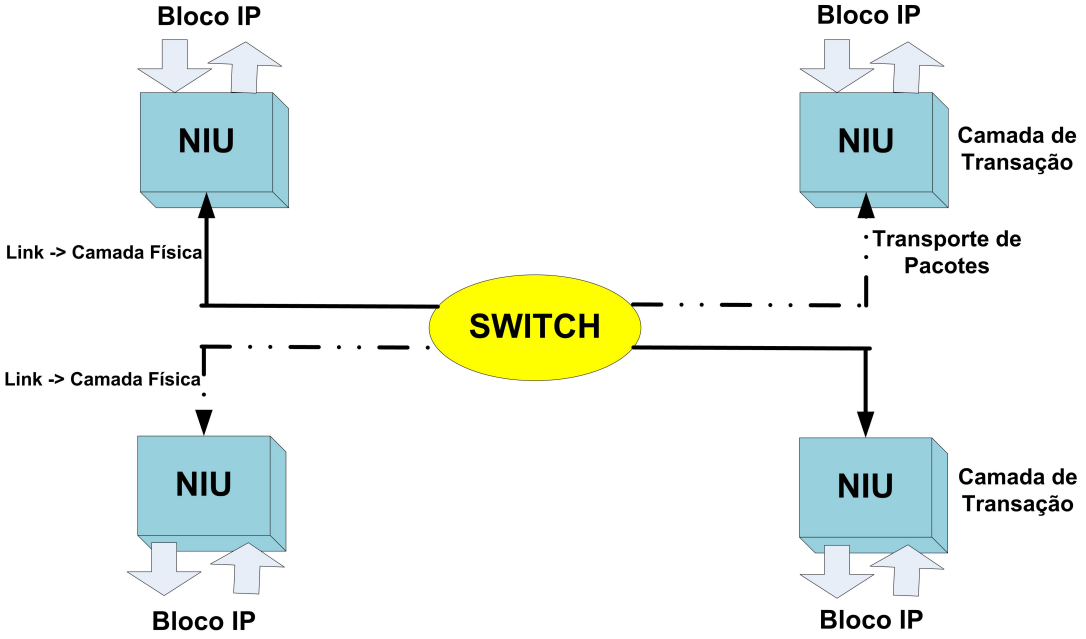

Figura 2.5: Sumário do mapeamento de camadas proposto pela Arteris

2.4.3 Benefícios de uma abordagem em camadas

Abaixo seguem algumas razões para a implementação de uma abordagem em camadas na construção de *chips*:

- Segregação de funções : O modelo em camadas permite que as funções sejam divididas, possibilitando que cada camada execute a sua função e encaminhe tanto para a camada superior ou inferior as tarefas concernentes às respectivas camadas;
- Escalabilidade : Como os roteadores lidam com o transporte de pacotes, eles suportam uma grande quantidade de transações simultâneas. A capacidade da interface de rede deve se enquadrar nos requerimentos dos blocos IP's que a mesma serve. Numa visão global, podemos ver que isso se refere a um ajuste no desempenho do conjunto (bloco IP + interface de rede), não possuindo nenhuma influência no desempenho e construção do roteador, o que contribui para adições de novos blocos IP's;
- Maior *throughput* : O *throughput* pode ser aumentado, pela escolha de um meio físico apropriado para transportar dados ou alocando algumas ligações físicas para um caminho lógico. Tomemos como exemplo uma situação que exija um intenso tráfego, entre memória e processador, a ligação física entre ambos pode ser otimizada para tal situação;
- Qualidade de Serviço : As regras para transportar pacotes permitem separar tráfego com requerimentos especiais, como tempo-real para serem isolados e encaminhados com maior prioridade que os outros. Permite também grandes pacotes de dados serem interrompidos por pacotes com prioridades maior;
- Tempo de Convergência : As diversas camadas existentes no modelo proposto não possuem nenhuma noção de tempo, isto acontece na camada física. Em uma implementação com vários blocos IP's, com diferentes *clocks* de freqüência, a sincronização entre os blocos acontece de maneira a não prejudicar as camadas acima, o que reduz o tempo de convergência da troca de mensagens entre os blocos IP's;
- Facilidade de Verificação : O modelo de camadas se encaixa na estratégia de "dividir e conquistar"nas fases de modelagem e verificação. A necessidade de verificar um item de uma camada não impacta em outra camada, o que torna esta tarefa mais fácil, pois se algum item não estiver de acordo com o que foi inicialmente proposto, não há necessidade de intervir nas outras camadas;
- Customização : Informações específicas podem ser adicionadas aos pacotes e transportadas entre às interfaces de redes. Um NoC customizado pode fazer uso destas informações, podemos ter um *firewall* que pode ser projetado para tratar determinados tráfegos sem autorização.

2.5 Compararações entre arquitetura em barramanto e *NoC*

Nesta seção procuraremos quantificar algumas métricas para comparar as arquiteturas em rede e barramento. O desafio desta comparação consiste em igualar as condições a que ambos foram submetidos. Ao longo desta seção, faremos algumas afirmações com este intuito.

2.5.1 Freqüência estimada

Do ponto de vista de uma implementação física, a grande diferença entre uma arquitetura em barramento e NoC está na freqüência adotada por cada arquitetura. Enquanto o barramento usa uma freqüência síncrona e interconexões multi-ponto, a NoC usa abordagem GALS e interconexões ponto-a-ponto.

O paradigma GALS particiona o sistema em blocos lógicos síncronos que se comunicam entre si assincronamente.GALS Em [24] é mostrado os benefícios desta implementação, com a economia de energia sendo o fator mais relevante. Para alguns barramentos, os controles de sinais devem atravessar o barramento por completo, algumas vezes em um intervalo de ciclo de *clock*, o tempo do barramento deve ser negociado entre os *masters* e em seguida enviado aos dispositivos que executarão as tarefas, também chamados *escravos*. No NoC, os modelos adotados (*GALS* e interconexões ponto-a-ponto) simplificam drasticamente este processo, diminuindo o tempo de convergência. Em [5], a comparação entre freqüências mostra uma vantagem da *NoC*, que chega a trabalhar a 800 Mhz, enquanto que o barramento trabalha a 250 Mhz.

2.5.2 Latência e *Throughput*

A latência e throughput são indicativos de desempenho do sistema. Na questão que abordamos, a comunicação intra-*chip*, os conflitos que existem em acessar recursos compartilhados acabam impactando no aumento da latência e diminuindo o *throughput*. Essas métricas são difícies de quantificar e com amparo de simuladores podemos ter uma idéia de como as mesmas se comportam em situações específicas.

O comportamento de sistemas de comunicação, com controle de fluxo, mostra que o nível de *throughput* ao atingir um nível de saturação, o qual não apresenta mais valores eficazes, faz com que a latência chegue a valores de saturação com ordens de grandeza muito maiores que a latência mínima. Esses picos de latência servem de motivação para os projetistas de *chips* com barramentos repensar em seus projetos alguns itens tidos como relevantes para que o mesmo apresente valores aceitáveis, de acordo com a aplicação a ser implementada, como : (a)

largura de banda, com o intuito de reduzir a taxa de utilização; (b) as transações de multiplexação de tempo para reduzir conflito de acesso ao barramento; (c) múltiplas camadas ao invés de acesso compartilhado, o que reduziria os conflitos de direcionados ao mesmo destino.

Na arquitetura em rede, o uso de switches dificulta que valores de latência alcancem valores altos. Esses valores de latência saturam quando as portas de comunicação entre roteadores ou entre roteador e a unidade de processamento atinje altos valores de utilização. A otimização desses valores poderá ser feita, por exemplo, através do aumento do número de canais virtuais.

Assumindo que a eficiência dentro do *cluster* do *chip* seja de 50%, o *throughput* em arquitetura em barramento é o resultado do produto entre a freqüência utilizada no barramento, o tamanho do *link* para tráfego de dados e a eficiência da comunicação. Com base nesses cálculos, podemos ter um throughput de : 250 MHz * 4 bytes * 50% = 0.5 GB/s.

Assumindo que para *NoC*, temos 20% do tráfico intra-*cluster*, *links* com 4 bytes de espaço, 50% de eficiência e caminhos diferenciados para requisição e resposta, então temos um throughput de (750 MHz * 4 bytes * 2 * 50%) / 20% = 15 GB/s. Os números apresentados acima não absolutos, mas nos ajudam a ter uma noção dos valores de *throughput* que podemos ter nas arquiteturas em rede e barramento. Podemos chegar a conclusão de que o ganho com uma arquitetura em rede é de , inicialmente, 30 (seis) vezes maior que a arquitetura em barramento.

2.5.3 Área e Energia

2.5.3.1 Área

Em uma arquitetura em barramento, na qual o canal de comunicação entre os blocos IP's é compartilhado entre todos os blocos, tem se mostrado eficiente no uso de sua área interna [2]. Vale afirmar que esse aspecto traga algum dano a escalabilidade do sistema e apresente uma única freqüência para todo o sistema. Para atenuar situações como estas, algumas pesquisas têm apresentado soluções como a implementação de *buffers* e *pipeline*. Embora tenham ajudado a arquitetura em barramento, estas implementações chegam a adicionar 250K "gates"segundo [2], chegando o mesmo a ter 400K *gates* com um *throughput* podendo chegar a 10 GB/s.

Em uma implementação em NoC, para as condições similares às mencionadas acima, temos um total de 210K *gates* para atingir *throughput* da ordem de 100GB/s.

2.5.3.2 Energia

Com respeito a dissipação de energia, a principal diferença entre NoC e barramanto é a maneira como é realizada a interligação entre os blocos IP's. Um planejamento adequado da topologia em rede leva a tamanho menores de ligações entre os blocos IP's e menor capacitância associada, o que resulta em menor consumo de energia. Também o desperdício de energia causado pela adoção de uma única freqüência de ciclos, o que não ocorre nos casos de NoC. Uma abordagem mais detalhada dos modelos utilizados em NoC serão mostrados em 3.4.

Capítulo 3

Aspectos relevantes para implementação de NoC

Neste capítulo serão abordados os aspectos que norteiam a construção de uma *NoC* como : (a) a arquitetura a ser usada na busca de um melhor desempenho da NoC; (b) a energia, sendo um fator relevante para dispositivos sem-fio que têm no consumo de energia um limitante para o seu uso; (c) o roteamento, como fator de otimização de desempenho para tráfego intra-chip.

3.1 Topologia

Uma *NoC* é composta de múltiplos recursos, ou blocos IP , conectados por canais em vários roteadores, os quais facilitam a troca de informações de um bloco para o outro. A disposição desses blocos IP's pode determinar o grau de escalabilidade e o desempenho de um sistema com um todo. A arquitetura [17] é um fator relevante para minimizar os alguns problemas encontrados nas arquiteturas em barramento, como escalabilidade e efeitos sub-micrômicos das propriedades elétricas usada no chip. Em [27] são mostrados os impactos significativos causados na escolha da arquitetura de um chip. Em [42], *Ivanov et al.* aborda itens essenciais que deverão acompanhar os projetos dos próximos chips, como modularidade e escalabilidade. Nesta seção mostraremos algumas arquiteturas usadas para *NoC*.

3.1.1 CLICHE

É a topologia mais simples, Chip-Level Integration of Communicating Heterogeneous Elements (CLICHE)[1] é uma rede em *mesh* bi-dimensional para projetos de *NoC*. Cada roteador é conectado a um bloco IP. A facilidade de roteamento nesta topologia ajuda a diminuir o tamanho do roteador, aumenta a capacidade

dos canais físicos e melhora a escalabilidade. A facilidade de uma topologia em *mesh* permite que o chip seja dividido em regiões de processamento ou de recursos, que permite : (a) usar protocolos de comunicação para determinadas regiões e não para todo o *chip*; (b) agrupar os blocos IP's que tenham uma intensa comunicação, o que ajudaria a economizar no gasto de energia; (c) facilitar implementação de uma comunicação com maior *throughput* na região necessária, por exemplo da memória para o processador. Em [1] mostra-se que, para carga de tráfego moderado, o tamanho do *buffer* pode ser próximo de 8 vezes do tamanho da mensagem, fator que elimina a possibilidade das mensagens serem descartadas, o que acontece quando as mesmas não encontram espaço suficiente no *buffer*

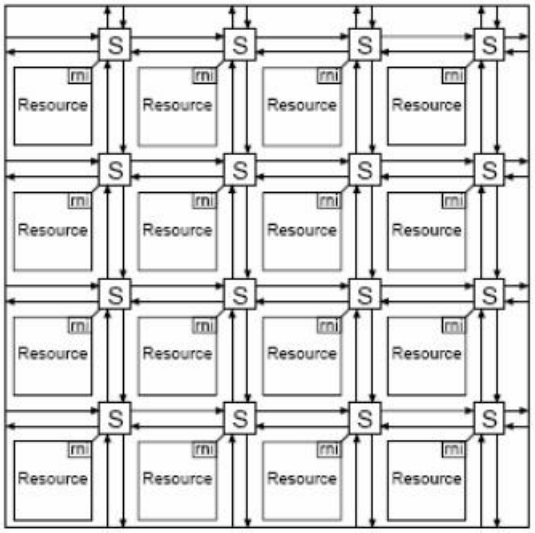

Figura 3.1: Topologia CLICHE[1]

3.1.2 *Torus*

Em [10] é mostrada uma topologia chamada *torus*. A topologia *torus* é similar à *mesh*, exceto pelo fato de que possui fios de interconexão que ligam os componentes das extremidades superior com a inferior, e os das direitas com os da esquerda. Devido à adição de interconexões que permitem que um número maior de blocos IP´s estejam interligados, temos as seguintes conseqüências: a largura de banda aumenta, e o nível de contenção que acontece nos *buffers* de entrada diminui. Devido ao tamanho dos *buffers*, o tamanho do roteador aumenta. Em [10], os autores afirmam que a área estimada a ser ocupada pelos roteadores é igual a 6,6 % da área total do chip. Uma fato a ser considerado pela topologia em *torus* é a interferência eletromagnética devido à grande quantidade de interconexões no chip.

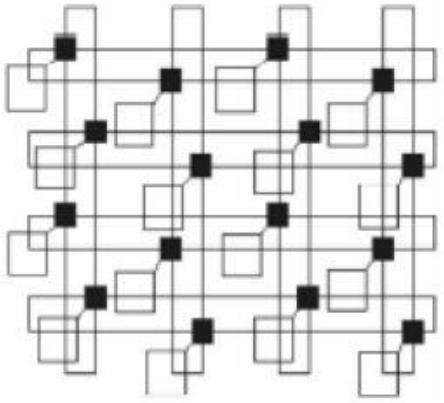

Figura 3.2: Topopologia em Torus

3.1.3 *Butterfly Fat Tree*

Na topologia *Butterfly Fat Tree* [37], os roteadores e blocos IP's estão dispostos em forma de uma árvore. Cada nó é representado por um conjunto de coordenadas (nível, posição), onde :

- nível é o nível na árvore que o *switch* ou bloco IP ocupa, com valores de 0 a n, onde "0"representa as folhas, ou seja os blocos IP que estão na parte inferior desta topologia;
- posição é a posição que o *switch* ou bloco IP ocupa, com valores de 0 a n, ordenando da direita para esquerda.

Cada roteador é conectado a outros 2(dois) roteadores do nível acima e a quatro "filhos"no nível abaixo, que podem ser outros *switches* ou blocos IP, que são chamadas de "folhas". Na seção 5.4.5 deste documento é mostrado um modelo analítico para cálculos de métricas de desempenho para a topologia *Butterfly Fat Tree*.

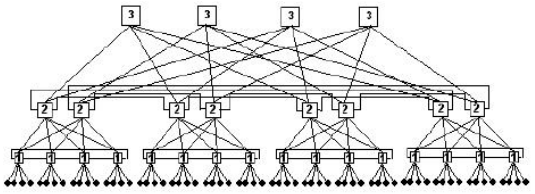

Figura 3.3: Topologia Butterfly Fat Tree com 64 blocos [19]

3.1.4 SPIN

A topologia *Scalable, Programmable, Integrated Network* (*SPIN*) [17] possui uma semelhança com a topologia *Butterfly Fat Tree*. Para uma árvore com "N"folhas

(ou blocos IP's), existem um total de 3N/4 *switches*. O número total de conexões é o mesmo que o número de "filhos"para cada *switch* na árvore, e todos os níveis de *switches* possuem o mesmo número de *switches*. Além disso, o tamanho da rede cresce na proporção de *(NlogN)/8*. A abordagem desta topologia é a preservação do *throughput* nos 2(dois) sentidos. Teoricamente, se existir tantos "pais"quantos "filhos"nas respectivas camadas, isto de fato se verifica. Para reduzir a possibilidade de contenção de tráfego na rede, os roteadores possuem a liberdade de usar os caminhos redundantes contidos nessa estrutura. Devido ao uso intenso de sua área total pelos componentes de sua estrutura, *SPIN* sacrifica os itens de área e energia por um *throughput* maior e em comparação com a topologia *Butterfly Fat Tree*.

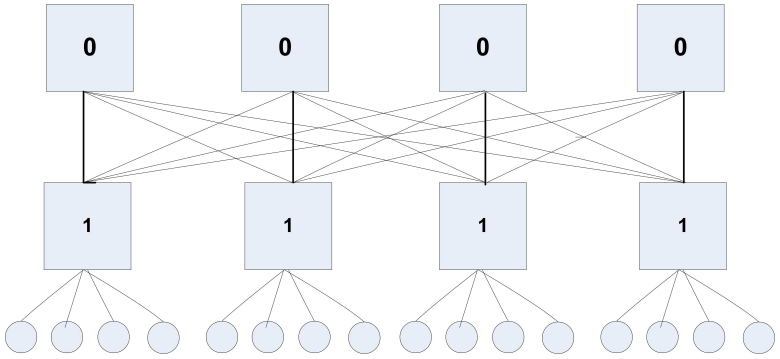

Figura 3.4: Topologia SPIN, com 16 terminais na árvore, 2 estágios de roteadores e 4 roteadores na raiz

3.1.5 Octogonal

Na topologia octogonal [25], nenhum componente atravessa mais do que 2(dois) saltos para se comunicar com outro componente. O modelo básico desta topologia é em anel, com 8 componentes sendo interligados através de 12 (doze) conexões. Cada componente possui 3(três) conexões, uma com o seu vizinho da direita, uma com o seu vizinho da esquerda e finalmente, uma conexão central com o componente oposto da rede. Esta topologia possui algumas vantagens : (a) como mencionado anteriormente, existem apenas 2(dois) saltos para a comunicação entre 2 (dois) quaisquer componentes; (b) alto *throughput*, devido ao aumento do números de caminhos sem sobreposição; (c) implementação simples do algoritmo de roteamento do caminho mais curto.

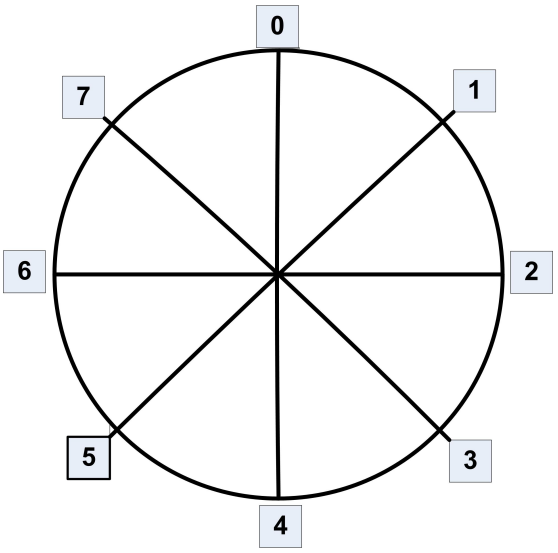

Figura 3.5: Topologia Octogonal

3.2 Energia

3.2.1 Introdução

Com respeito à dissipação de calor, uma diferença a ser observada entre a arquitetura de barramento e *NoC* é que a primeira possui ligação com todos os alvos, enquanto a segunda possui ligações ponto-a-ponto. No caso da NoC, um posicionamento adequado dos roteadores possibilita fios de tamanhos menores e uma menor capacitância associada de carga roteada por transação do que a de barramento, resultando em menor dissipação de energia. É mais fácil de implementar uma estratégia de economia de energia em arquitetura baseada em rede, devido a modularidade que a mesma confere. Comparações justas entre abordagens de barramento e NoC requerem experimentos em aplicações reais como também a freqüência em que cada arquitetura permite, mas existe uma expectativa que a *NoC* possua um consumo menor que a de barramento em sistema equivalentes.

Quando um *flit*(*flow digit*) é emitido em sua origem, o mesmo dissipa energia ao passar por fios e unidades lógicas no seu caminho até o destino. A preocupação por parte dos projetistas é com a dissipação da energia dinâmica causada pelo processo de comunicação na rede. A energia é fator limitante para a autonomia de dispositivos portáteis, embarcados e sem-fio.

Vários modelos de energia têm sido apresentado na literatura. Em [39], apresenta-se um levantamento de técnicas para uso eficiente de energia para *NoC*, com técnicas em diferentes níveis da comunicação, a saber : em nível de rede, que aborda a questão do roteamento de dado; nível de sistema sobre a questão da voltagem a ser fornecida; em nível de arquitetura, sobre a topologia a ser usada; em nível de circuito, sobre tamanho de *buffers*. Em [38], Patel enfatiza a necessidade de modelar o consumo de energia e desempenho em interconexões de

redes para projetos com multíplos processadores. É apresentado um modelo de energia para roteadores e interconexões. Um *framework* estimado para modelos de consumo de energia para a construção de *switches* é proposto em [49]. Os mesmos autores [47] propõem um modelo de energia para comunicação em chips através de pacotes.

O consumo de energia em uma rede provém de 3 (três) diferentes origens: 1) roteadores, localizados entre os nós da rede ; 2) *buffers* internos, usados para armazenar os pacotes quando algum tipo de contenção acontece ao longo do caminho a ser percorrido; 3) fios de interconexão, que conectam os nós e roteadores na rede. O consumo de energia está atrelado às configurações impostas a rede, como tráfego a ser injetado, número de canais virtuais, tamanho do *buffer*, entre outros.

3.2.2 Modelo de energia com *bit energy*

Em uma comunicação em rede, os pacotes percorrem diferentes caminhos como alternativa a encontrar o menor ou melhor caminho entre origem e destino. Convém ressaltar que o tráfego em cada caminho pode sofrer alterações de acordo com a carga a que o *link* é submetido. Para estimar o consumo de energia nestas conexões de rede, é proposto um modelo em [47], definido como Bit Energy E_{bit}.
No modelo proposto, E_{bit} é definido como a energia consumida por cada *bit*, quando o mesmo é transportado dentro do *switch* da porta de entrada para a porta de saída. Para calcular o E_{bit} é necessário efetuar a soma da energia do *bit* consumida nos itens mencionados anteriormente, o que nos forneceria a seguinte equação :

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \tag{3.1}$$

- no *switch* - $E_{S_{bit}}$;
- nos *buffers* internos - $E_{B_{bit}}$;
- e nos fios da interconexão - $E_{W_{bit}}$

Nos parágrafos a seguir iremos explicar cada item.

Consumo de energia no *switch* : Quando um *bit* atravessa o nó do *switch*, as unidades lógicas, que estão localizadas entre a origem e destino, consomem energia assim que mudam de estado.
Para efeitos de cálculos, a energia consumida dentro do *switch* depende dos processos de alocação e transmissão de pacotes como mostrado na Figura 3.7.

Os itens de destaque são o caminho de dados do cabeçalho e o caminho de dados da transmissão. Ambos os processos consomem energia, porém o cabeçalho

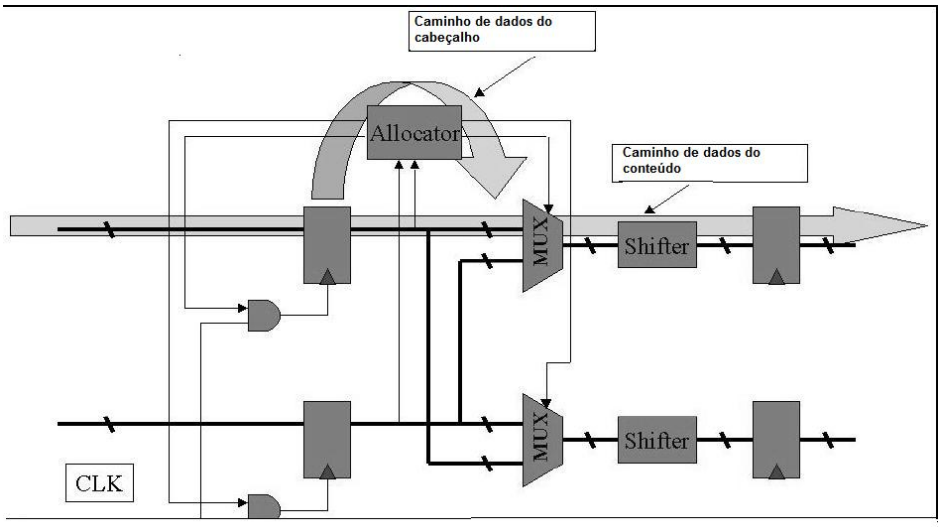

Figura 3.6: Processo de alocação e transmissão de dados

ocupa uma pequena parte do total do pacote, então é usual ter o *bit energy* do conteúdo como energia do *switch* E_{Sbit}.

Consumo de energia no *buffer* interno: Na fabricação de circuitos para *switches*, os *buffers* são implementados com memórias SRAM e DRAM compartilhadas. O consumo de energia em *buffers* possui 2(duas) origens : 1) Energia de acesso de dados, que é consumida nas operações de "READ"ou "WRITE"de acesso à memória, e 2) Energia de atualização, consumida pelas operações de atualização na memória. A energia do *bit* no *buffers* internos $E_{B_{bit}}$ pode ser expressa pela seguinte equação :

$$E_{Bbit} = E_{access} + E_{ref} \tag{3.2}$$

Na equação 3.2, temos E_{access} que é a energia consumida por cada acesso a memória e E_{ref}, que é a energia consumida por cada operação de atualização na memória. Sabemos que a memória é acessada por palavra ou byte e não por um único *bit*. O parâmetro E_{access} é uma média de energia consumida por um único *bit*. A energia consumida pelo acesso a memória é determinada pela contenção que possa haver entre as portas de ingresso nos *buffers* internos. Em [47], aborda-se os tipos de contenção que pode haver com o *bit*, forçando assim que o mesmo seja armazenado nos *buffers*.

Consumo de energia nos fios de interconexão: Quando o *switch* move um *bit* para o fios de interconexão, o sinal no fio irá oscilar entre os sinais lógicos "1"e "0". A energia é dissipada nesta troca de sinal lógico. Para calcular o *bit energy*, temos a equação 3.3 :

$$E_{W_{bit}} = 1/2C_{fio}V^2 + 1/2C_{ua}V^2 = 1/2C_WV^2 \tag{3.3}$$

Na equação acima C_{fio} é capacitância fios nas interconexões e C_{ua} é a capacitância das unidades lógicas que estão conectadas aos fios. $C_w = C_{fio} + C_{ua}$ é capacitância total propagada pelo *bit* em análise. E "V"denota a potencial elétrico.

3.3 Roteamento

Roteamento é o ato de mover informações entre origem e destino seguindo algumas regras pré-definidas, como menor caminho ou melhor custo. Quando analisamos a Internet, a mudança de metodologia na comunicação ocorreu de forma gradativa, com a comunicação baseada em pacotes substituindo a comunicação de circuitos comutados, adicionando um grau maior de escalabilidade a redes de computadores.
Dependendo do algoritmo de roteamento implementado, os roteadores podem ser classificados em 2 (dois) tipos : Roteamento baseado na distância do vetor (fixo) e Roteamento baseado nas condições do *link* (adaptativo)[29] :

- Roteamento baseado na distância do vetor: Este tipo de roteamento é simples, onde cada roteador mantém uma tabela com os destinos conhecidos e envia estas informações aos roteadores que estão conectados ao mesmo. Com esta metodologia para manter as rotas, o roteador mantém informações sobre os vizinhos, não possuindo informações sobre a rede como um todo. Em situações em que os links possuem velocidade similar, essa técnica é simples, pois é necessário efetuar o cálculo do número de saltos que deverá ser realizado até o destino, e então é escolhida a melhor rota de acordo com essa métrica.

- Roteamento baseado nas condições do *link*: O roteador envia para outros roteadores, em sua rede, dados sobre suas condições de processamento, consumo de largura de banda, tabela de roteamento e outros dados pertinentes a comunicação. A vantagen desta técnica é que os roteadores possuem conhecimento das condições dos outros roteadores que estão em sua rede.

As maiores vantagens em usar um algoritmo determinístico são a sua simplicidade de projeto, e a baixa latência de roteamento quando a rede não está congestionada. No entanto, a medida que a taxa de injeção de pacotes aumenta, roteadores com esta técnica, a de roteamento baseado em distância, possuem dificuldade em lidar com a degradação de *throughput*, pois não possuem uma solução para a questão de congestionamento de *links*.

Em contraste, roteadores adaptativos aumentam a chance de pacotes evitarem congestionamento através da busca de caminhos alternativos, o que leva a *throughput* maiores. No tráfego entre origem e destino, os roteamentos adaptativos possuem uma latência maior comparada com os roteamentos fixos, isto é levado em

consideração com baixos níveis de congestionamento na rede.

3.3.1 Roteamento para *NoC*

A complexidade que os *chips* estão adquirindo, como mencionado no capítulo 1, mostra a necessidade de implementação de algoritmos de roteamento otimizados com o intuito de fornecer uma comunicação ágil, eficaz e que permita uma economia de energia, fator que se apresenta como um grande gargalo para dispositivos sem-fio. Várias propostas foram apresentadas. Em [21], Jingcao e Marculescu combinam as vantagens dos esquemas de roteamento determinístico e adaptativo. Em [3] apresenta-se um roteamento dinâmico e que suporta uma comunicação entre os módulos que são dispostos dinamicamente. Em [48], Luca et al. mostra que diferentes esquemas de roteamento afetam o desempenho e o consumo de energia em sistemas com multi-processadores. Em [46] mostra-se uma técnica de roteamento baseado no nível de contenção de entrada dos roteadores.

O desempenho e eficiência de uma *NoC* é altamente dependente da infraestrutura de comunicação. O meio em que o tráfego interno flui é fator determinante na avaliação do desempenho dos roteadores nos chips e conseqüentemente, nos ajuda a encontrar métricas (*throughput* e latência) que avaliem a viabilidade de implementação do mesmo. As diferenças entre o roteamento em rede de computadores e em uma *NoC* são, como descritas em [20]:

- Espaço do *buffer*: Com o intuito de minimizar os custos de implementação, é mais razoável usar registradores para roteadores em vez de memórias. A implementação de registradores ajuda a diminuir a latência, o que é fator crítico para aplicações em *System-on-Chip*
- O uso de roteamento determinístico : Em comparação com o roteamento adaptativo, o roteamento determinístico requer menos recursos. Como os pacotes chegam fora de ordem, tanto no roteamento adaptativo como no determinístico, o tamanho de *buffer* deverá ser levado em consideração, o que impacta na área total do roteador.

A arquitetura de um roteador genérico que está conectado ao bloco IP (*Intellectual Property*) consiste em 5 (cinco) controladores de entrada e 5 (cinco) controladores de saída. A estrutura do roteador, mostrada na figura 3.8, possui em suas conexões um espaço para os *buffers* de entrada. A tabela de roteamento é mantida atualizada, e o roteador possui informações de outros roteadores que estão conectados. Este processo é mantido pelo algoritmo de roteamento usado. O roteamento dentro da *NoC* irá depender da arquitetura usada pelo roteador, mas de acordo com a literatura encontrada, as características apropriadas para o roteamento de uma *NoC* são : (a) fixo, na qual as rotas estão configuradas previamente; (b) *deadlock-free* [8], o qual permite que a rede sempre esteja desimpedida para tráfego; (c) mínimo, este termo se refere ao custo de se transportar

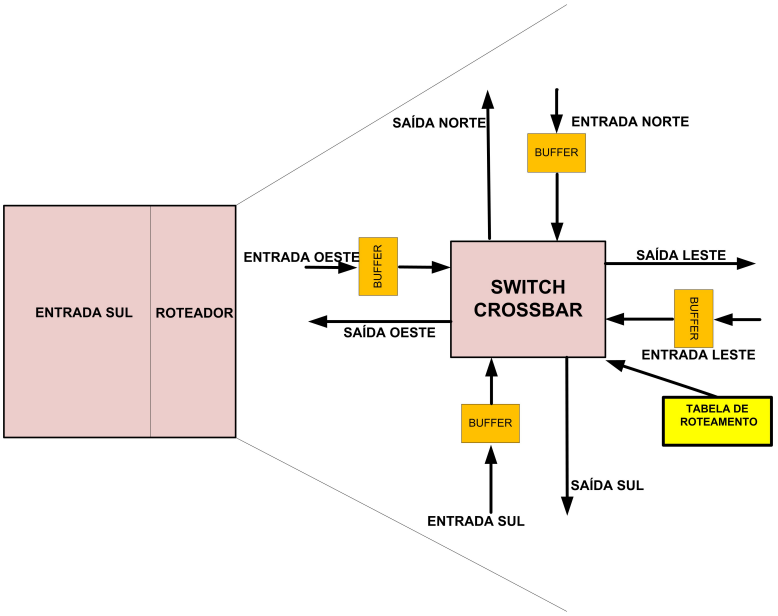

Figura 3.7: Arquitetura de um bloco IP com o roteador

dados na rede e também sobre a distância entre a origem e o destino, os quais devem ser os menores possíveis ; (d) *wormhole* [12] é o método usado na qual a mensagem avança imediatamente do canal de entrada para o canal de saída, se o mesmo estiver disponível, este assunto está detalhado em 5.3.3.

Após mostrar alguns aspectos que cercam em implementações, faz-se necessário pesquisar as implementações realizadas em projetos de NoC. No próximo capítulo mostraremos algumas implementações de NoC, convém afirmar que cada projeto procurou abordar um aspecto da NoC.

Capítulo 4

Implementações de *Network on Chip*

Neste capítulo mostraremos algumas implementações de *NoC* existentes, e as abordagens que cada implementação enfoca. As implementações mostradas abordam questões particulares de *NoC*, como consumo de energia, modularidade, menor *time-to-market*, reusabilidade, topologia entre outros.

Descrever as possíveis implementações serve de subsídios, para a escolha de uma abordagem que se encaixe na aplicação que suportará tal implementação. Vale ressaltar que o intuito final das variadas implementações é criar um ambiente com um melhor desempenho e com um menor gasto.

4.1 HERMES

As topologias de *NoC* são definidas pela estrutura de conexões de seus roteadores. A *NoC* HERMES, descrita em [32], assume que cada roteadores possui um conjunto bidirecional de portas conectadas com outros roteadores ou blocos IP's. Na topologia em *mesh* usada para este projeto, cada *switch* possui diferentes números de portas, dependendo da localização. O uso de topologias em *mesh* é justificado pela facilidade de disposição dos blocos IP's, roteador e tarefas de roteamento. A rede HERMES pode ser projetada usando outras topologias como *torus*, hipercubo ou similares. Porém, implementações em tais topologias implicam em mudanças de conexões dos roteadores e nos algoritmos de roteamento.

O principal objetivo de um *switch* é fornecer uma transferência de mensagem, de maneira eficaz e correta, entre os bloco IP's. Os roteadores geralmente possuem uma unidade lógica de roteamento, uma unidade lógica de arbitragem e portas de comunicação, que estão conectadas diretamente a outros roteadores ou blocos IP's. As portas de comunicação incluem canais, de saída e de entrada, os quais podem ter *buffers* para armazenarem temporariamente as informações.

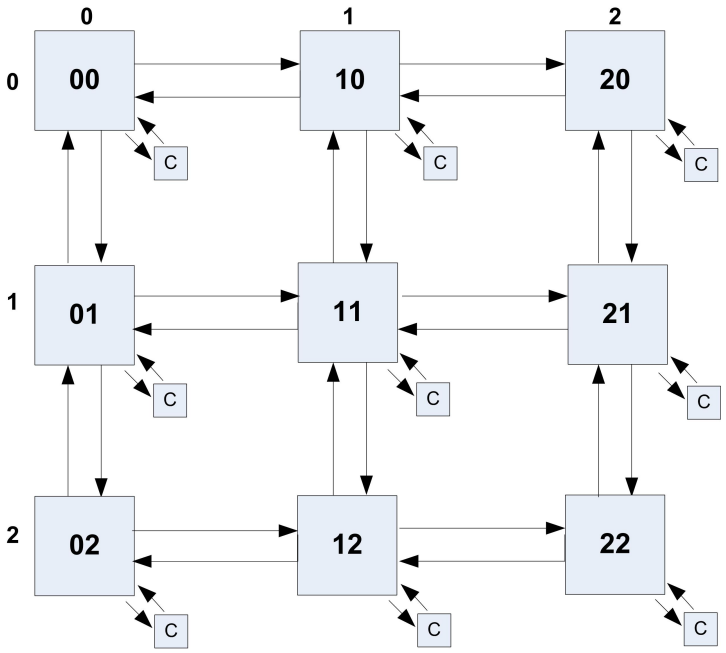

Figura 4.1: Topologia da *NoC Hermes*

O *switch* HERMES possui uma unidade lógica de roteamento e 5 (cinco) portas bidirecionais : Norte, Sul, Leste e Oeste. Cada porta possui um *buffer* para armazenar as informações temporariamente, caso o alvo de destino esteja ocupado ou alguma problema aconteceu no caminho entre origem e destino. A porta local estabelece uma comunicação entre o *switch* e seus blocos IP's. As outras portas do *switch* são conectadas com os outros roteadores, como mostrado na figura 4.1. A unidade lógica de controle de roteamento implementa a unidade lógica de arbitragem e o algoritmo de comutação de pacotes.

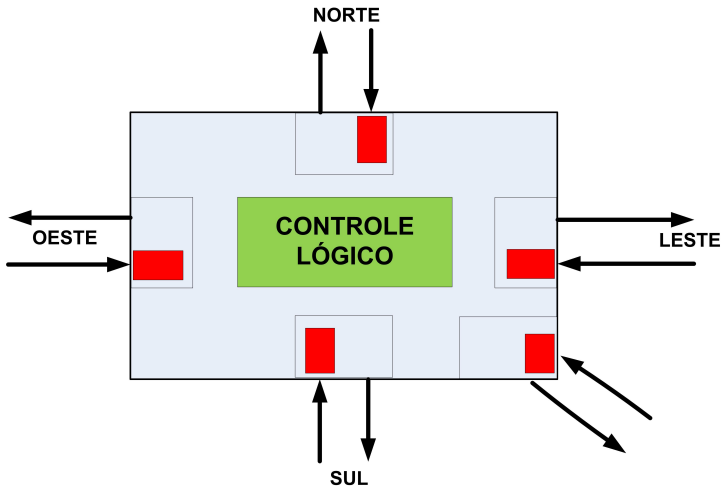

Figura 4.2: Roteador da *NoC Hermes*

A rede HERMES usa o roteamento *wormhole*, descrito em 5.3.3, que divide os pacotes em *flits*. Para efeitos da rede HERMES, o tamanho do flit é

parametrizável e o número de *flits* em um pacote é fixado em 2 *bits*. O primeiro e segundo *flit* de um pacote são os cabeçalhos das informações, sendo respectivamente o endereço de destino do *switch* e o número de *flits* no *payload* do pacote. Para efeitos de roteamento, cada roteador possui um endereço único na rede, que é expresso em forma de coordenadas "XY", onde "X"representa a posição horizontal e "Y"representa a posição vertical.

4.2 Æthereal - Philips NoC

Nesta seção, mostraremos o *Æthereal*[14], desenvolvido no *Philips Research Laboratories*. A *NoC Æthereal* faz uso intenso de serviços diferenciados que garantem uma comunicação, com o intuito de eliminar as incertezas que cercam as interconexões e facilitar uma integração. Os serviços diferenciados são implementados através de configurações individulizadas que são realizadas nas métricas de desempenho que cercam a comunicação entre os blocos IP's, como *throughput* e latência, que podem ser configurados para não ter nenhuma garantia na comunicação ou uma garantia limitada.

O *Æthereal* oferece serviços de "garantia"e "melhor esforço"na comunicação, embora antagônicos. A seguir mencionaremos alguns itens que tornam esta situação possível e que indicam a direção tomada por este laboratório de pesquisa na implementação de *NoC*. Para obter êxito em sua tarefa, o *Æthereal* concentrou esforços em 2(dois) componentes vitais para uma NoC : roteadores e interfaces de rede. O canal implementa uma comunicação ponto-a-ponto entre 2(duas) interfaces de rede, com a possibilidade de termos vários caminhos para se conectarem 2(dois) blocos IP's. O canal pode ter uma garantia de latência e *throughput* (GT), ou de melhor esforço (BE). Estes termos, GT e BE, usados para caracterizar a qualidade da comunicação, mostram as condições com que o tráfego flui nos canais de comunicação, sempre buscando ter confiabilidade e desempenho satisfatório. Para implementar *GT*, é usado roteamento livre de contenção, que utilizam uma multiplexação TDM (*Time Division Multiplexing*) para a comutação de circuito, onde um ou mais circuitos podem monstados para a conexão.

A ordem de entrega das mensagens não é garantida quando existe a possibilidade de vários canais ou caminhos, então as conexões no *Æthereal* combina múltiplos canais, controle de fluxo e uma reordenação das mensagens. O modelo de programação adotado no *Æthereal* pode ser centralizado ou descentralizado. Os blocos IP's podem montar ou remover conexões. Configurações que foram realizadas com os blocos "off-line"podem ser colocadas no chip e não trarão dano algum na comunicação.

Uma performance com garantia de serviços resulta em uma reserva no uso de *buffer* e ligações na NoC. Para fornecer 100% de garantia, essas reservas devem ser para o pior caso, consumindo qualquer espaço de largura de banda que não

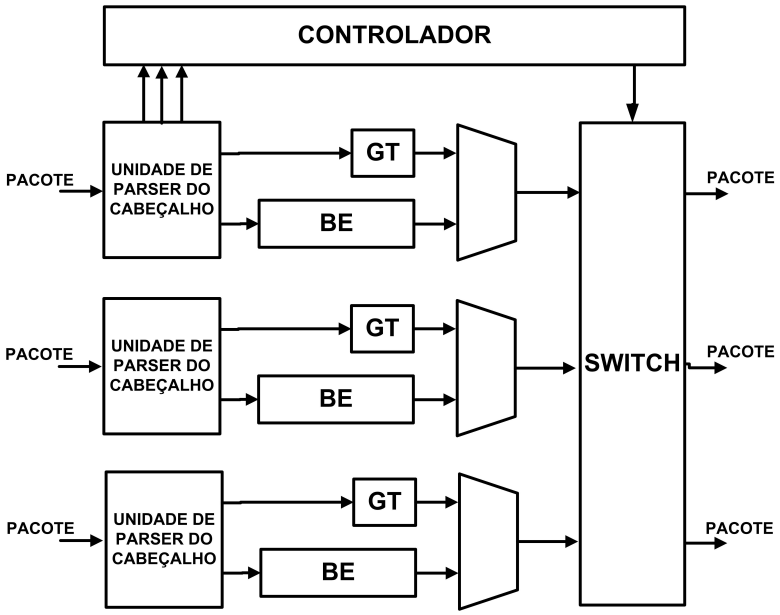

Figura 4.3: Roteador da *NoC Æthereal*

for utilizado. Para aumentar o uso dos recursos são introduzidas as conexões *BE*, que fazem uso dos recursos que eventualmente não estão sendo utilizados em seu nível máximo. O roteador na figura 4.3 é uma combinação destas abordagens, com roteadores *GT* e *BE* trabalhando em paralelo. O roteador *BE* possui uma prioridade menor. Um flit *BE* pode usar o *link* somente quando não houver nenhum bloqueio *GT* no *link*.

4.3 SoCIN - System on Chip Interconnection Network

SoCIN é apresentado em [50], desenvolvido pela Universidade Federal do Rio Grande do Sul. Pode ser construído usando topologia em 2-D tanto em *mesh* quanto *torus*. A topologia em *mesh* apresenta menores custos, enquanto *torus* reduz a latência da mensagem. Um item a ser comentado é a implementação

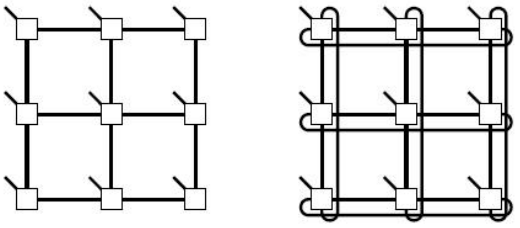

Figura 4.4: As duas topologias para SoCIN : mesh e torus

das interconexões do *SoCIN*, que possuem 2 (dois) canais unidirecionais, cada um com seus dados, controles de fluxo e empacotamento. A mensagem é transmitida com "n" *bits* de dados e 2 (dois) *bits* que servem para indicar o inicio do pacote (*bop : begin-of-packet*), e final do pacote (*eop : end-of-packet*). Os *bits* controle de fluxo são usados para validar dados no canal (*val*) e para dar conhecimento os dados recebidos (*ack*). A figura 4.5 mostra a estrutura do *link* do *SoCIN*.

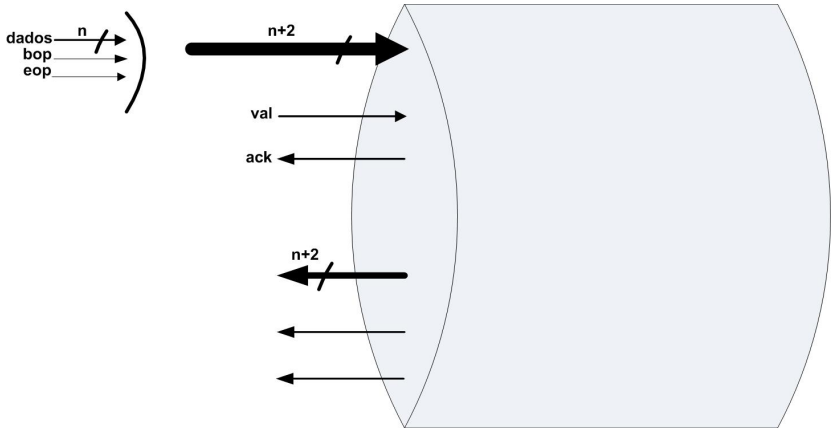

Figura 4.5: Link do SoCIN [50]

No *SoCIN*, os núcleos se comunicam enviando e recebendo mensagens. Neste modelo de comunicação, qualquer núcleo da topologia pode se apresentar tanto como cliente e como servidor, dependendo da necessidade. Como os núcleos podem apresentar diferentes modelos de comunicação, é necessário que haja uma padronização para a conexão dos mesmos no *chip*, para tal é intenção dos projetistas implementar empacotadores OCP/SoCIN e VCI/SoCIN que possam integrar esses núcleos e padronizar as interfaces de interconexão.

4.4 PROTEO - Tampere University of Technology, Finlândia

PROTEO [43] é o nome dado para uma proposta de rede. Nesse projeto, o foco consiste em pesquisar novos protocolos, arquiteturas e implementações de blocos IP's, deixando de lado as ferramentas de software. Na figura 4.6, temos um diagrama de fluxo dos passos, usados nessa abordagem, para a construção de chips. Os blocos sombreados, que estão na Figura 4.6, são comtemplados neste projeto. No projeto *PROTEO* para *NoC*, a topologia que está sendo explorada é uma rede hierárquica em anel com várias sub-redes com topologias em estrela ou barramento. Vale lembrar que estas sub-redes estarão conectadas aos *hubs* da rede em anel. A conexão do *host* com o anel principal ou a sub-rede depende da informação disponível na interface do *host*: a topologia em estrela é formada com BVCI(*Basic Virtual Component Interface*), enquanto os blocos implementam interfaces AVCI (*Advanced Virtual Component Interface*) que estarão diretamente

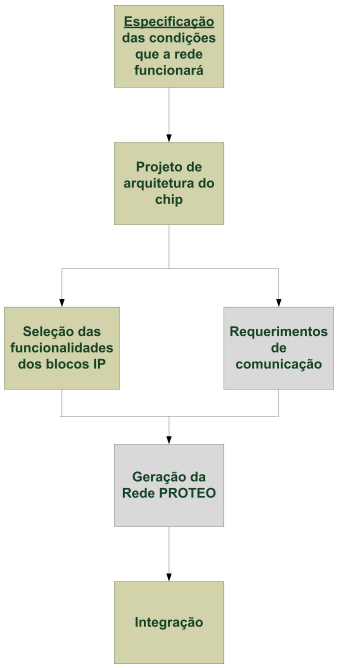

Figura 4.6: Diagrama de fluxo do PROTEO

conectadas ao anel principal. Estas interfaces são padrões de conexão estabelecidos pela VSIA(www.vsia.org). Na topologia em estrela, são definidos dois tipos de nós, ou blocos: (a) um empacotador, que encapsula os pacotes que estão na interface; (b) o *hub*, que roteia os pacotes de um nó para outro enquanto conecta a rede em estrela com o resto da rede.

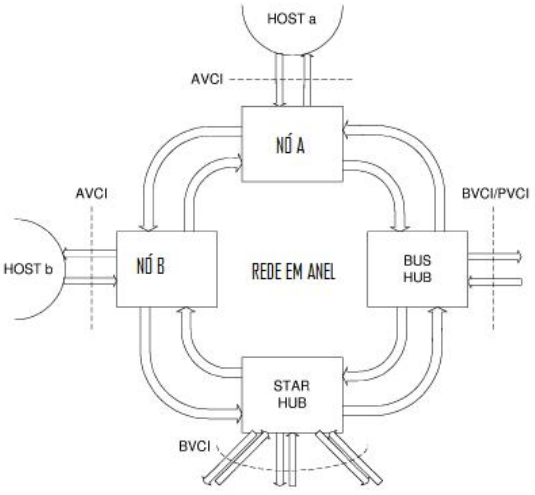

Figura 4.7: Exemplo da topologia para do projeto PROTEO

4.5 SoCBUS - Department of Electrical Engineering, Suécia

A implementação da *NoC* mostrada nesta seção foi desenvolvida na Universidade de Linköping, na Suécia [45]. As razões para a escolha desta topologia, em vez de outras como hexágono, octogonal ou *butterfly*, é que a *mesh* possui um custo aceitável para a interconexão dos fios e uma largura de banda razoável. Outro ponto a ser observado é a facilidade de agrupar os blocos IP's que se comunicam com bastante intensidade, de maneira a não consumir recursos necessários do *chip* e que comprometa o desempenho da rede.

A Figura 4.8 mostra a arquitetura da rede. O *switch* possui 5(cinco) portas, das quais 4(quatro) são usadas para conexão com os *switches* adjacentes e um para conexão com a interface do bloco IP local. Entre o *switch* e o bloco IP, existe um dispositivo com a função de acomodar o bloco IP e mitigar as diferenças de comunicação que possa haver como largura da porta e padronização das interfaces. O empacotador contém um espaço em *buffer* e faz a ponte entre as diferentes freqüências de *clock* da rede e do bloco IP.
As interfaces internas usam o mesmo formato, o que mostra uma preocupação

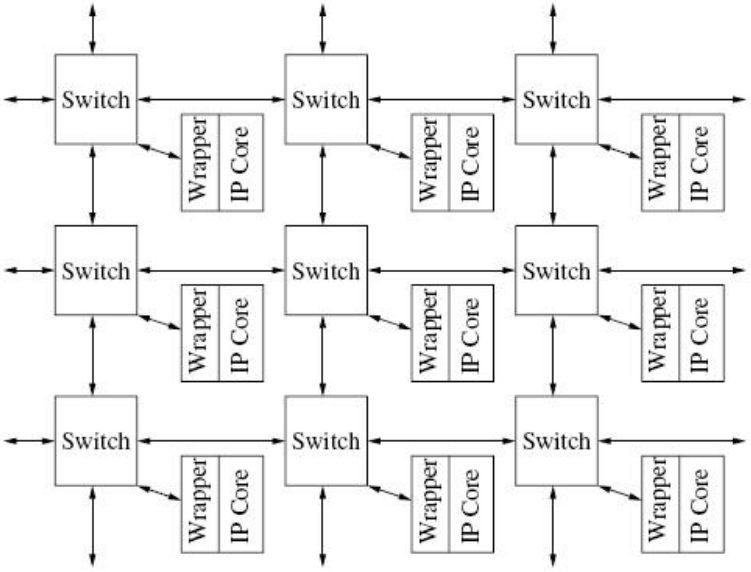

Figura 4.8: Topologia da rede SoCBus com os blocos IP's e empacotadores

em padronizá-las. Na Figura 4.9, vemos um total de 11 (onze) conexões em cada direção, onde: 8 (oito) conexões levam dados e roteamento de pacotes ; 1 (uma) conexão para controle da transmissão e informações de *clock* para facilitar as atualizações de tempo; 2(duas) conexões para controle reverso que possuem a finalidade de dar conhecimento das sessões estabelecidas. A linha na diagonal representa uma interface para a porta do empacotador do bloco IP local.

A simplicidade dos componentes do SoCBUS permite uma implementação

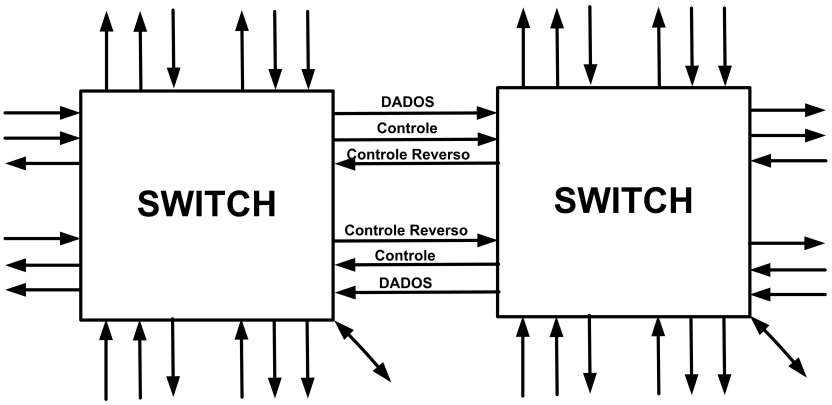

Figura 4.9: Topologia da rede SoCBus com os blocos IP's e empacotadores

de roteadores e dispositivos de padronização (empacotadores) com velocidade de 1.2GHz em tecnologia de 0.18 micrômetros. Considerando a alta taxa de freqüência e a natureza distributiva da redes em *chips*, os atrasos nas conexões entre os blocos IP´s acaba sendo um problema, se for usada a metodologia tradicional de sincronismo. Para atenuar esses atrasos, e o problema de sinais de *clock* que chegam em componentes diferentes em tempos diferentes, foi proposto o uso de uma metodologia. Chamada de mesócrono, em que a freqüência é a mesma para todos os componentes, mas com fases desconhecidas. O uso de mesócrono e a técnica de *retiming* permite que as conexões tenham atrasos diferentes sem causar problemas maiores. Esta abordagem possui vantagens : não há necessidade de repetidores; o sistema consume uma quantidade de energia mínima; e não há necessidade de dispor a rede de maneira padrão no *chip*.

A busca pela melhor topologia para uma rede é uma fonte para os pequisadores, os passos iniciais consistem em encontrar dados que justifiquem uma determinada escolha. No nosso trabalho, buscamos um modelo analítico que nos ajude a encontrar métricas de desempenho de várias topologias, e assim teremos subsídios para iniciarmos um estudo para a escolha de uma topologia que atenda às necessidades da aplicação que será usada no *chip*. No capítulo seguinte, mostraremos um modelo analítico amplamente usado para calcular a latência e *throughput* dentro da *NoC*, pois são importantes métricas de desempenho para *NoC* segundo mostrado em [16].

Capítulo 5

Modelo Analítico para Roteamento *Wormhole*

5.1 Introdução

Neste capítulo mostraremos uma abordagem geral descrita em [12] do modelo analítico para roteamento *wormhole*. Mostraremos a técnica de roteamento *wormhole*, amplamente usada entre os projetistas para *NoC*, devido à sua fácil implementação no hardware, que possui largura de banda satisfatória e latência menor em relação às outras técnicas, as quais serão detalhadas neste capítulo. As métricas alvos de nosso estudo, como mencionado ao longo desta dissertação, são latência média e *throughput*

5.2 Roteamento

A seleção de um caminho para uma mensagem é baseada no algoritmo *e-cube* ou *left-right*(LR). Neste algoritmo, as dimensões de rede n são estritamente ordenadas como d_{n-1}, d_{n-2},...,d_0 usando a relação $d_i > d_j$ se e somente se $i > j$. As dimensões são inspecionadas do maior para o menor, e as ligações na dimensão d_i são percorridas se e somente se o endereço de destino difere do endereço atual naquela dimensão. Esse algoritmo permite apenas um caminho para um determinado par origem-destino, o que assegura roteamento *deadlock-free* [11] para hipercubos binários. Em [8] são apresentados sugestões para os efeitos do *deadlock-free*, como o uso de canais virtuais [26].

As técnicas de roteamento podem classificadas em 2 grupos: comutação de pacotes e comutação de circuitos virtuais. A tradicional comutação de pacotes consiste em dividir as mensagens, que devem ser enviadas através da rede, em pequenos pedaços roteáveis chamados pacotes. Cada pacote possui um cabeçalho contendo o controle das informações necessárias para o roteamento do mesmo, e

os dados propriamente ditos.

Com o intuito de aumentar a performance da rede em questão, *NoC's*, buscamos técnicas de roteamento que permitam alcançar valores de latência e *throughput* considerados satisfatórios. Em redes macroscópicas, a comutação de pacotes geralmente explora a técnica *Store and Forward*: um pacote pode ser encaminhado para a porta de saída correspondente do roteador, somente se o mesmo tenha chegado por completo na porta de entrada. No método *Virtual Cut Through*, assim que a porta de saída fica livre, os elementos dos pacotes, *flits*, são enviados imediatemente mesmo que o pacote não tenha chegado por completo. Já o método de roteamento *Wormhole* é comparável ao método *Virtual Cut Through* mas exige espaços menores no *buffer*. A seguir iremos mostrar com detalhes essas técnicas mencionadas.

5.2.1 *Store and Forward*

Em [6], o método de transmissão *store-and-forward* é definido como uma quádrupla $SF = \{V, E, BN, B\}$, onde $V = \{v_1, ..., v_n\}$ é um conjunto de nós, E é um conjunto bi-direcional de *links*, BN é um conjunto de *buffers* e $B : V \mapsto N^+$ é uma função. Naturalmente, $\sum B(v_i) = |BN|$, que associa a cada nó o número de buffers que ele contém. Denote como $G_{SF} = (V, E)$ o grafo da rede SF.

As mensagens devem ser transmitidas entre nós vizinhos, através de roteadores R_i, isto é, caminhos finitos $< v_{i1}, v_{i2}, ..., v_{ik} >$ em G_{SF}.

A transmissão é realizada de acordo com os seguintes passos:

- (a) A mensagem *m* é originada no nó s(m), se existir pelo menos um *buffer* livre em $v = s(m)$;
- (b) A mensagem *m* pode ser transmitida para o nó *prox(m)* somente se existir pelo menos um *buffer* livre no nó prox(m). Esse *buffer* pode ser designado para *m*;
- (c) Qualquer mensagem *m* desaparece tão logo chegue no nó de destino, *desta(m)*.

5.2.2 *Virtual Cut-Through*

A técnica de roteamento *virtual cut-through* permite que os pacotes sejam encaminhados, embora os mesmos não estejam completamente armazenados no *buffer*. O pacote pode então ser armazenado em mais de 2 (dois) *buffers* ao mesmo tempo e mais de uma conexão pode ser alocada. No entanto, se o cabeçalho do pacote não puder ser encaminhado, devido a insuficiência ou indisponibilidade de recursos,

o pacote inteiro deve ser armazenado no *buffer* do nó em que o pacote encontrou alguma anormalidade, para que o cabeçalho seja encaminhado novamente assim que encontrar recursos disponíveis para tal. Esse comportamento torna este método mais vantajoso em relação ao método *Store-and-Forward* em casos de tráfego intenso. Em [40] é mostrado um modelo analítico para roteamento *virtual cut-through*, o qual mostra expressões que predizem o comportamento de grandes redes e ajudam a medir o custo-benefício de diferentes estratégias de roteamento.

Na Figura 5.2 temos uma ilustração detalhada do método apresentado:

- (a) O cabeçalho do flit é movido para o *buffer* de saída do primeiro roteador;
- (b) O cabeçalho segue para o segundo roteador, enquanto os subseqüentes flits estão seguindo o respectivo caminho;
- (c) Puxando a corrente de *flits*, o cabeçalho segue para o roteador onde seu *buffer* de saída está reservado para outra comunicação;
- (d) A corrente de *flits* se contrai e o pacote inteiro é armazenado no *buffer* no primeiro roteador, liberando todos os *links* previamente alocados;
- (e) Flit pipeline se movendo em direção ao destino.

5.2.3 Roteamento *Wormhole*

O roteamento *wormhole* é o método no qual a mensagem avança imediatamente do canal de entrada para o canal de saída, se o mesmo estiver disponível. O pacote é dividido em *flit* (*flow control digits*). O *flit* no cabeçalho do pacote é o responsável pelo roteamento. À medida que o cabeçalho avança na rota especificada, os *flits* restantes seguem em um modelo *pipeline*. Um pequeno *buffer* é associado a cada canal para armazenar o *flit*. Se o cabeçalho encontrar

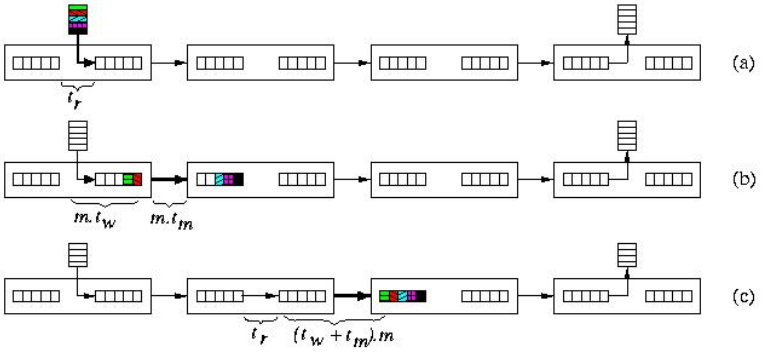

Figura 5.1: Roteamento *Store-and-Forward*

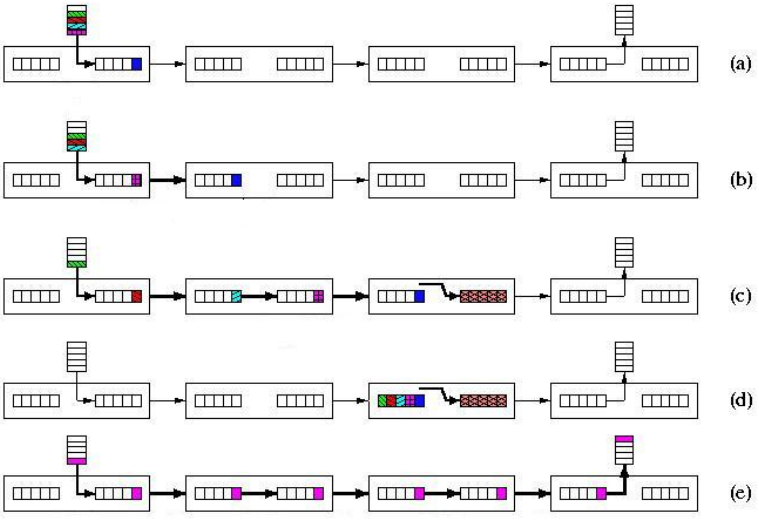

Figura 5.2: Roteamento Virtual Cut-Through

um canal já em uso, o mesmo é bloqueado até que o canal esteja livre e os *flits* restantes ficam armazenados nos *buffers* de cada nó da rota estabelecida. Devido ao fato de a maioria dos *flits* não conterem informações de roteamento, estes devem permanecer em canais contíguos da rede e não podem ser intercalados com os *flits* de outras mensagens. Uma vez que o *flit* da mensagem seja aceito e associado ao buffer correspondente do nó, os *flits* restantes devem ser aceitos pelo canal. Caso não haja nenhum bloqueio entre os nós de origem e destino, a mensagem é encaminhada para o nó de destino como uma rede de comutação de circuito [34].

Uma vantagem inicial de roteadores *wormhole* é o requisito de espaço de *buffer* que pode ser pequeno, permitindo que roteadores sejam extremamente pequenos e rápidos. A desvantagem óbvia é o fato da mensagem reter espaços nos *buffers* e canais, que aumentaria a possibilidade de *deadlocks*. Uma análise detalhada sobre performance de roteamento *wormhole* é mostrada em [7] [9] [12].

Na Figura 5.3, temos uma ilustração do método apresentado:

- (a) O cabeçalho é copiado para o *buffer* de saída depois de ter sido feita a decisão sobre o roteamento;
- (b) O *flit* com o cabeçalho da mensagem é transferido para o segundo roteador e os outros flits estão seguindo-o;
- (c) O *flit* com o cabeçalho chega no roteador com o canal de saída ocupado e a cadeia inteira de *flits*, ao longo do caminho, fica parada, bloqueando todos os *flits* em seus respectivos canais;
- (d) *Pipeline* de *flits*, em caso de não haver conflito de roteamento, estabelecendo o *wormhole* através do roteadores

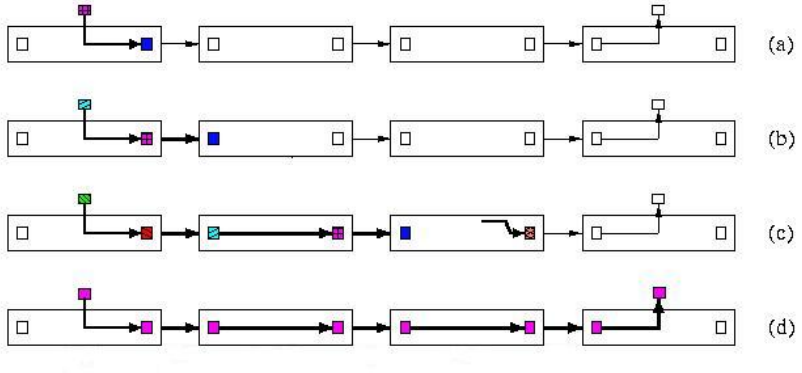

Figura 5.3: Roteamento Wormhole

5.2.4 Considerações entre as técnicas apresentadas

Para uma comparação entre as técnicas de roteamento devemos levar em consideração as condições na qual a técnica deverá ser implementada. A técnica *Store-and-Forward* é vantajosa quando as mensagens são pequenas e freqüentes, a necessidade de armazenar o pacote inteiro exige que o projeto do roteador para esta técnica seja caro. A latência é proporcional ao produto entre distância a ser percorrida e tamanho do pacote. A técnica *Virtual Cut-Through* mostra que somente o cabeçalho possui informações sobre o roteamento dos flits subseqüentes. A técnica *Wormhole* permite roteadores simples, pequeno, baratos e rápidos, é a técnica usada nas máquinas comerciais.

5.3 Estudo de Modelo Analítico para Roteamento *Wormhole*

Apresentar um modelo matemático com o objetivo de quantificar as métricas de performance consiste em uma árdua tarefa, pois as inúmeras variantes que compõem o assunto tornam este modelo complexo. A parte mais difícil, em desenvolver um modelo analítico, é o cálculo da probabilidade do bloqueio da mensagem em um determinado roteador, devido ao número de combinações que devemos levar em consideração quando enumeramos os caminhos que a mensagem deve percorrer.

Os unidirecionais *k-ary* e *n-cube*, onde k é o raio e n como a dimensão, possui $N = k^n$ nós. Cada nó pode ser identificado por um dígito n, raio k, endereço $(a_1, a_2, ..., a_n)$. O i-ésimo dígito do endereço do vetor, a_i, representa a posição do nó na dimensão i_{th}.

Cada nó consiste em um elemento de processamento (EP) e um elemento de roteamento (ER). O EP contém um processador e algum tipo de memória, enquanto o roteador possui *(n+1)* canais de entrada e *(n+1)* canais de saída. O

nó é conectado aos seus vizinhos através dos "n"canais de entrada e "n"canais de saída. Os canais restantes são usados pelos EP's para injetar/ejetar mensagens de/para a rede, nos quais estão inseridos. As mensagens geradas pelo EP são transferidas para o roteador através do canal de injeção e as mensagens para o destino são transferidas para o EP local através do canal de ejeção.

5.3.1 Análise

As hipóteses de trabalho geralmente empregadas [12] e [11] na literatura são:

1. O tráfego de mensagens gerado em cada nó é um processo de *Poisson*, e é independente do nó vizinho;
2. O tráfego em cada canal é aproximado pelo processo *Poisson* e indepedendente;
3. Mensagens de destino são distribuídas uniformemente através dos nós da rede;
4. O tamanho da mensagem é fixo e igual a M *flits*, e a transmissão de cada mensagem é efetuada em um ciclo de um roteador para o outro;
5. A fila no canal de injeção no nó de origem possui capacidade infinita. Além do mais, as mensagens são transferidas para o EP local assim que o mesmo chega em seus destinos através de um canal de ejeção;
6. Cada canal físico possui V canais virtuais [26].

5.3.2 Modelo Analítico

A latência de uma mensagem injetada a partir de um nó é determinada pelo tempo de espera e o tempo de serviço no canal de injeção, os quais dependem dos tempos de serviço e de espera dos canais sucessivos no caminho a ser percorrido. Quando a mensagem é gerada no EP, que está conectado ao switch j, a mesma esperará por um tempo denotado por $W_{inj,j}$. Uma vez que o cabeçalho seja aceito na rede, a mensagem permanecerá por um tempo no canal de injeção. Esse período, chamado de tempo de serviço, é denotado por $x_{inj,i}$. Ao final do tempo de serviço, quando a parte final da mensagem deixa o canal de injeção, a mensagem terá D-1 passos para toda a mensagem ser recebida no destino, onde D é o número de canais no caminho. Quando a parte final da mensagem deixa o canal de injeção, o cabeçalho deve ter chegado no destino onde não existe mais bloqueio. A latência L_j para a mensagem injetada no nó j é expressa na seguinte fórmula:

$$L_j = W_{inj,j} + x_{inj,j} + D - 1 \tag{5.1}$$

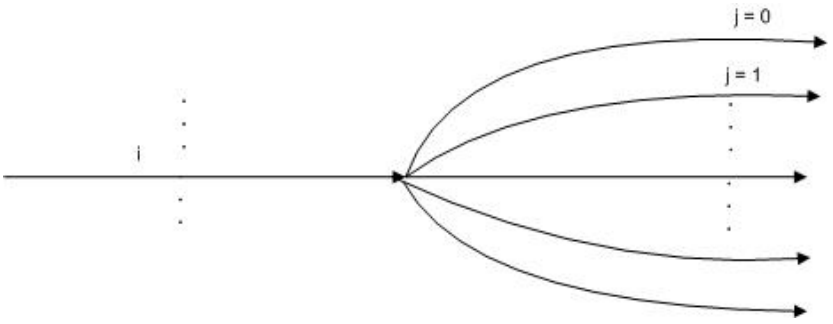

Figura 5.4: Seqüencia de canais para uma mensagem do canal i

Onde L_j é a latência total. $W_{inj,j}$ é o tempo de espera que a mensagem possui até ser injetada no canal de transmissão.$x_{inj,j}$ é o tempo de serviço que a mensagem possui no canal de transmissão. D é o número de canais.

Para calcular a média de todos os nós, a latência média $\overline{L}$ para a rede inteira é:

$$\overline{L} = \frac{1}{N} \times \sum_{j=0}^{N} \overline{L}_j \tag{5.2}$$

$$\overline{L} = \frac{1}{N} \times \sum_{j=0}^{N} (\overline{W}_{inj,j} + \overline{x}_{inj,j}) + \overline{D} - 1 \tag{5.3}$$

onde N é o número de nós na rede e $\overline{D}$ é a distância média da mensagem.

O significado do tempo de espera na equação acima pode ser aproximado pelo modelo tradicional de filas M/G/1 [4], que possui uma chegada de tempo exponencialmente distribuída e uma distribuição arbitrária para tempo de serviço, que é conhecido como $x_{inj,j}$. No roteamento *wormhole*, a mensagem é enviada através de muitos canais em um determinado tempo. O tempo de serviço em um canal arbitrário, no caminho da mensagem, pode ser analisado com a ajuda da Figura 5.4.

Mensagens que tenham origem no canal *i* com taxa de chegada λ_i^{in} podem ser roteadas para os canais de saídas denotados por j = 0, 1, 2, 3,... . O tempo de serviço para os canais de entrada depende do tempo de serviço e tempo de espera de todos os possíveis canais de saída. Seja $R_{i|j}$ a probabilidade de uma mensagem vindo do canal de entrada *i* ser roteada para o canal de saída *j*. O tempo de serviço para o canal de entrada *i* pode ser expresso pela seguinte fórmula:

$$x_i^{in} = \sum_{j=0}^{N} (x_j + w_j) \times R_{i|j} \tag{5.4}$$

Onde x_j é o tempo de serviço para o canal de saída *j* e w_j é o tempo de espera

para o canal de saída *j*. A equação acima vem a confirmar a dependência que o tempo de serviço, em um determinado canal, possui em relação aos tempos de serviço e tempos de espera dos canais subseqüentes. A média do tempo de espera w_j pode ser aproximada usando o modelo M/G/1. Neste caso, pode ser aproximada pela sugestão descrita em [12], com a expressão abaixo:

$$\overline{W}_j = \frac{\lambda_j \overline{x}_j^2}{2(1-\lambda_j \overline{x}_j)} \left[1 + \frac{(\overline{x}_j - s/f)^2}{\overline{x}_j^2}]\right] \tag{5.5}$$

Onde λ_j é a taxa de mensagem no canal de saída *j*, *s* é o tamanho de parte da mensagem (configurado pela aplicação ou simulação) e f é o tamanho em *flits*.

Mas o modelo M/G/1 assume uma chegada independente de mensagens no switch, que poderão bloquear um ao outro. Uma vez que o link de entrada estiver ocupado por um *flit*, não poderá haver mais entrada de *flits* naquele canal até que o primeiro *flit* seja servido, ou encaminhado. Logo, para usar o resultado do tempo de espera do modelo M/G/1, multiplicamos pela probabilidade de bloqueio $P_{i|j}$:

$$w_j = P_{i|j} \overline{W}_j \tag{5.6}$$

onde $P_{i|j}$ pode ser expresso como, seguindo [12] :

$$P_{i|j} = 1 - \frac{\lambda_i^{in}}{\lambda_j} R_{i|j} \tag{5.7}$$

Podemos ter uma idéia de que $P_{i|j}$ como a probabilidade de que a mensagem, que segue o modelo M/G/1, seja de outro canal diferente de *i*. Combinando as expressões 5.4, 5.6, 5.7, obtemos o tempo de serviço para mensagens incidentes no canal de entrada *i*:

$$x_i^{in} = \sum_{j=0}^{N} \left[x_j + (1 - \frac{\lambda_i^{in}}{\lambda_j} R_{i|j}) \overline{W}_j \right] R_{i|j} \tag{5.8}$$

Observemos que $\overline{W}_j$ é a média do tempo de espera obtida na Equação 5.5, λ_i^{in} é taxa de mensagem no canal de entrada *i* e λ_j é a taxa de mensagem no canal de saída *j*.

5.3.3 Modelo Analítico para Redes *Mesh* e *Torus*

Nesta seção aplicaremos o modelo apresentado na seção anterior para a rede com topologia em *Mesh* [15]. Nessa topologia de rede, os nós são conectados por

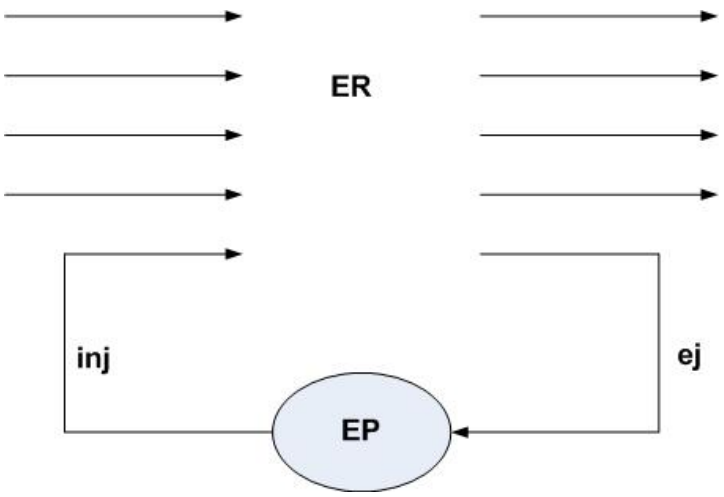

Figura 5.5: O nó possui um elemento de roteamento(ER) e um elemento de processamento(EP)

links bi-direcionais. Seja k^2 o número de nós, onde k é o número de nós em cada dimensão. Vale lembrar que cada nó possui um elemento de roteamento e elemento de processamento como mostrado na figura 5.5.

Iremos denotar cada nó na rede usando um par de coordenadas cartesianas x e y, $< j_i, j_j >$. Para o canal, denotaremos as coordenadas do nó juntamente com as direções a serem tomadas (N, S, L, O). Por exemplo, $< j_0, j_1, L >$ mostra um caminho indo para o leste a partir do nó $< j_0, j_1 >$. Assumindo que cada link seja bi-direcional, temos 2k(k-1) canais, onde k é o número de nós em cada dimensão.

Um outro conceito abordado nesta teoria é o de canais com equivalência de classe, definida em [12] como sendo um conjunto de canais que possuem características idênticas, no que tange ao tempo de serviço e a taxa de mensagem. A equivalência de classe pode ser observada quando analisamos que o canal localizado no limite a oeste $< j_0, j_1, O >$ possui características semelhantes ao canal localizado no limite a leste $< k - 1 - j_0, j_1, L >$. Estes canais mencionados pertencem a mesma classe de canais de equivalências. Isto ajuda a economizar tempo para mapear o comportamento dos nós na rede, pois temos como analisar parte dos nós.

5.3.3.1 Taxa de Mensagem

Com o modelo adotado, temos a taxa de mensagem, no canal de injeção, de acordo com o processo de Poisson [4] com taxa λ_{node}. Vamos assumir que a taxa de emissão de mensagens seja igual a taxa de recepção de mensagens. A taxa de mensagem para cada canal é determinada pelo número de nós na origem e pelo número de nós no destino que usam um determinado canal.

De acordo com a afirmação de que a taxa de emissão de mensagens seja igual

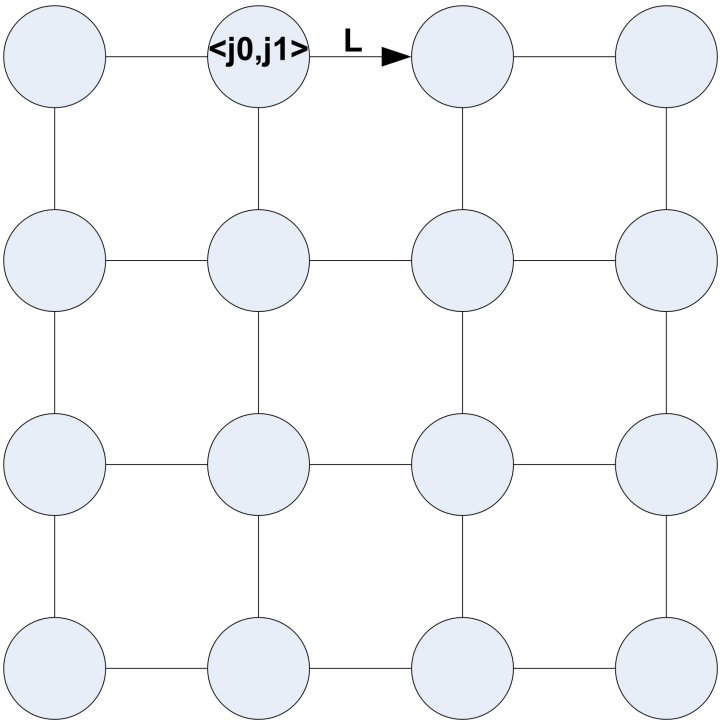

Figura 5.6: Topologia em Mesh com tráfego para "L"a partir do nó

a taxa de recepção de mensagens, temos:

$$\lambda_{<inj,j_0,j_1>} = \lambda_{<j_0,j_1,e_j>} = \lambda_{node}(0 \leq j_0, j_1 < k) \tag{5.9}$$

Para os "k - 1"canais com equivalência de classes na dimensão x, as taxas de mensagens são :

$$\lambda_{<j_0>} = \frac{j_0(k-j_0)k}{k^2-1}\lambda_{node}(0 \leq j_0 < k) \tag{5.10}$$

Existem "k(k -1)"canais com equivalência de classes na dimensão y, as taxas de mensagens são :

$$\lambda_{<j_0,j_1>} = \frac{j_1(k-j_1)k}{k^2-1}\lambda_{node}(0 \leq j_0 < k; 1 \leq j_1 < k) \tag{5.11}$$

5.3.3.2 Latência Média

O tempo de serviço é computado na ordem de ejeção dos canais, a saber: dimensão x, dimensão y e canais de injeção. No canal de ejeção, uma vez que o primeiro *flit* da mensagem seja recebido, o resto da mensagem será recebido um *flit* por vez, sem nenhum bloqueio. Logo, o tempo de serviço nos canais de ejeção é igual

ao tamanho, em flits, da mensagem, como mostrado na equação abaixo:

$$x_{<j_0,j_1,ej>} = s/f \tag{5.12}$$

Seguiremos com os cálculos de tempos de serviço nas dimensões "x"e "y".

No eixo "x", o tempo de serviço é determinado como se segue :

- A mensagem deixa o canal $< j_0 >$ com a probabilidade $R_{<j_0>|e_j} = \frac{1}{j_0}$;
- Ou continua para o próximo canal na mesma direção, com a probabilidade $R_{<j_0>|j_0-1} = \frac{j_0-1}{j_0}$.

O tempo de serviço x_{j_0} é mostrado na equação abaixo :

$$x_{<j_0,j_1>} = (s/f)\frac{1}{j_0} + \left[x_{j_0-1} + \frac{1}{k-j_0+1}W_{<j_0-1>}\right]\frac{j_0-1}{j_0} \tag{5.13}$$

No eixo "y", o tempo de serviço é determinado como se segue :

- Ao deixar o canal $< j_0, j_1 >$, a mensagem pode sair para o próximo canal de ejeção disponível, com probabilidade $R_{<j_0,j_1>|e_j} = \frac{1}{j_1 k}$;
- Continua para o próximo canal na mesma direção, com probabilidade $R_{<j_0,j_1>|<j_0,j_1-1>} = \frac{j_1-1}{j_1}$;
- Segue para o canal localizado mais a leste, com probabilidade $R_{<j_0,j_1>|<j_0>} = \frac{j_0}{j_1 k}$;
- Ou segue para o canal localizado mais a oeste, com probabilidade $R_{<j_0,j_1>|<j_0,O>} = \frac{k-1-j_0}{j_1 k}$.

O tempo de serviço $x_{<j_0,j_1>}$ é mostrado na equação abaixo :

$$\begin{aligned} x_{<j_0,j_1>} = (s/f)\frac{1}{j_1 k} &+ \left[x_{<j_0>} + \frac{k(k-j_0)-(k-j_1)}{k(k-j_0)}W_{<j_0>}\right]\frac{j_0}{j_1 k} \\ &+ \left[x_{<k-1-j_0>} + \frac{kj_0+j_1}{k(j_0+1)}W_{<k-1-j_0>}\right]\frac{k-1-j_0}{j_1 k} \\ &+ \left[x_{<j_0,j_1-1>} + \frac{1)}{k-j_1+1)}W_{<j_0,j_1-1>}\right]\frac{j_1-1}{j_1} \end{aligned} \tag{5.14}$$

No canal de injeção, o tempo de serviço é determinado como se segue :

- Ao deixar o canal de injeção no nó $< j_0, j_1 >$, a mensagem pode ir para o norte, com probabilidade $R_{inj}| < j_0, j_1, N > = \frac{(k-1-j_1)k}{k^2-1}$

- Ou seguir para o sul, com probabilidade $R_{inj}| < j_0, j_1 > = \frac{j_1 k}{k^2-1}$
- Ou seguir para o leste, com probabilidade $R_{inj}| < j_0, L > = \frac{(k-1-j_0)k}{k^2-1}$
- Ou seguir para o oeste, com probabilidade $R_{inj}| < j_0 > = \frac{j_0}{k^2-1}$

A equação de tempo de serviço para o canal de injeção, é mostrada pela equação abaixo :

$$\begin{aligned} x_{<inj,j_0,j_1>} = & \left[x_{<j_0>} + \frac{k(k-j_0)-1)}{k(k-j_0)} W_{<j_0>} \right] \frac{j_0}{k^2-1} + \\ & \left[x_{<k-1-j_0>} + \frac{k(1+j_0)-1}{k(1+j_0)} W_{<k-1-j_0>} \right] \frac{k-1-j_0}{k^2-1} + \\ & \left[x_{<j_0,j_1>} + \frac{k-j_1-1}{k-j_1} W_{<j_0,j_1>} \right] \frac{j_1 k}{k^2-1} + \\ & \left[x_{<j_0,k-1-j_1>} + \frac{j_1}{1+j_1} W_{<j_0,k-1-j_1>} \right] \frac{(k-1-j_1)k}{k^2-1} \end{aligned} \tag{5.15}$$

Os tempos de espera $W_{<inj,j_0,j_1>}$ são calculados usando $x_{<inj,j_0,j_1>}$ e a equação 5.5 deste capítulo. A latência média para a rede inteira é mostrada na equação abaixo :

$$\overline{L} = \sum_{j_0=0}^{k-1} \sum_{j_1=0}^{k-1} (\overline{W}_{<inj,j_0,j_1>} + \overline{x}_{<inj,j_0,j_1>}) + \overline{D} - 1 \tag{5.16}$$

5.3.4 Modelo Analítico para Redes *Butterfly Fat-Tree*

Para ilustrar o modelo analítico apresentado em [12], mostraremos nesta seção a rede *Butterfly Fat-Tree*, descrito em [16] como na Figura 3.3.

Cada nó é representado por um par de índices (n,e), onde "n"é o nível do nó na rede e "e"é o endereço do nó naquele nível, a designação de um nó nessa topologia será S(n,e). No nível mais baixo estão os "N"nós com endereços de 0 a N-1. Nesse nível, cada *switch* possui 6(seis) portas: pai_0 , pai_1, $filho_0$, $filho_1$, $filho_2$, $filho_3$, $filho_4$. No nível "x", sendo que o valor de "x"varia entre 1 e $log_4 N$, existem $N/2^{x+1}$ *switches*.

As conexões dos *switches* são determinadas pelos endereços dos *switches* como se segue : o pai_0 do S(n,e) é conectado ao $filho_1$ do $S(n+1, \left\lfloor \frac{e}{2^n+1} \right\rfloor 2^l + a \bmod 2^l)$ e o pai_1 de $S(n,e)$ é conectado ao $filho_1$ de $S(n+1, \left\lfloor \frac{e}{2^n+1} \right\rfloor 2^n + (e + 2^{n-1}) \bmod 2^n)$,

onde $i = \left\lfloor \frac{e \bmod 2^{n+1}}{2^{n-1}} \right\rfloor$. Nesta topologia, existe mais de um caminho mais curto possível entre um par de nós. A mensagem pode ir para qualquer um dos 2(dois) *links* para cima. Isto se aplica se o destino não estiver no mesmo nível, ou no nível abaixo, que a mensagem é endereçada. Quando um *flit* necessita ir para um nível acima, o *link* é selecionado randomicamente. Se o mesmo estiver bloqueado, tenta-se o outro e se ambos estiverem bloqueados, o flit espera até que algum *switch* do *link* acima possa processá-lo.

5.3.4.1 Taxa de Mensagem

Nesta topologia, os *links* que estão no mesmo nível e estão na mesma direção, sendo ascendente ou descendente, são simétricos, e para esses casos não há necessidade de distinguí-los. Podemos rotular os *links* e suas taxas de chegadas de mensagens pelo par de índices $< i, j >$, onde "i"é o nível do início do link e "j"é o nível final do link na rede, $0 \leq i,\ j \leq n$, com $n = log_4^N$. Vamos assumir que cada nó injeta mensagens na rede com a taxa de λ_0. Nestas condições, temos as seguintes afirmações $\lambda_{0,1} = \lambda_{1,0} = \lambda_0$ para *links* entre os nós no nível 0(zero) e os *switches* de nível 1(um). Vamos considerar a possibilidade de uma mensagem ir de um nível "n"para um nível "n+1"$(1 \leq n < x)$. Admitindo termos $N = 4^x$ nós no sistema, a mensagem pode ter $4^x - 1$ destinos os quais $4^n - 1$ podem ser alcançados sem passar pelo nível "n". Então a probabilidade que uma mensagem de um nível "n"subir a um nível acima é:

$$P_n^{\uparrow} = \frac{4^n - 4^x}{4^n - 1} \tag{5.17}$$

e a probabilidade que uma mensagem desça um nível :

$$P_n^{\downarrow} = 1 - P_n^{\uparrow} \tag{5.18}$$

A taxa total de mensagens saindo do nível "n"para o nível "n+1"é igual a $P^{\uparrow}4^n\lambda_0$. Existem no total $\frac{4^x}{2^l}$ *links* entre o nível "n"e nível "n+1". A taxa de mensagem para cada canal indo do nível "n"para o nível "n+1"é $\lambda_{n,n+1} = \frac{P^{\uparrow}4^n\lambda_0}{\frac{4^x}{2^n}}$. Devido à simetria a taxa de mensagem do nível "n"para o nível "n+1"é igual à do nível "n+1"para o nível "n", resumindo:

$$\lambda_{l,l+1} = \lambda_0 \frac{4^n - 4^x}{4^n - 1} 2^n \tag{5.19}$$

$$\lambda_{l+1,l} = \lambda_{l,l+1} \tag{5.20}$$

5.3.4.2 Tempo de Espera e de Serviço

De acordo com as equações mostradas anteriormente, temos inicialmente o tempo de serviço dos *links* do nível 1 para o nó de processamento (nível 0), igual ao tamanho da mensagem em *flits*, como mostrado na equação 5.12:

$$x_{1,0} = s/f \tag{5.21}$$

O tempo de espera é calculado usando a Equação 5.5 deste capítulo:

$$\overline{W}_{1,0} = \overline{W}_{M/G/1}(\lambda_{1,0}, x_{1,0}) \tag{5.22}$$

Para qualquer trânsito de mensagem entre os níveis "n"e "n+1", não importando o sentido, existem 4 possibilidades de saída do canal e cada um com a mesma probabilidade (1/4). O tempo de serviço médio é determinado pela Equação 5.11:

$$\overline{x}_{n+1,n} = \overline{x}_{n,n-1} + \left(1 - \frac{1}{4}\frac{\lambda_{n+1,n}}{\lambda_{n,n-1}}\right)\overline{W}_{n,n-1} \tag{5.23}$$

O tempo de espera médio $\overline{W}_{n+1,n}$ é calculado usando $\overline{x}_{n+1,n}$:

$$\overline{W}_{n+1,n} = \overline{W}_{M/G/1}(\lambda_{n+1,n}, \overline{x}_{n+1,n}) \tag{5.24}$$

Consideremos agora os canais ascendentes, começando no canal $\langle n-1, n\rangle$. Existem agora apenas 3(três) possibilidades de saída, cada um com probabilidade de 1/3. Então temos :

$$\overline{x}_{n-1,n} = \overline{x}_{n,n-1} + \left(1 - \frac{1}{3}\frac{\lambda_{n-1,n}}{\lambda_{n,n-1}}\right)\overline{W}_{n,n-1} \tag{5.25}$$

como resultado :

$$\overline{x}_{n-1,n} = \overline{x}_{n,n-1} + \frac{2}{3}\overline{W}_{n,n-1} \tag{5.26}$$

O tempo médio de espera $\overline{W}_{n-1,n}$ é determinado utilizando o modelo da Equação 5.5 deste capítulo, através do método de aproximação de Hokstad [36], e vale :

$$\overline{W}_{n-1,n} = \overline{W}_{M/G/2}(\lambda_{n-1,n}, \overline{x}_{n-1,n}) \tag{5.27}$$

A exceção está no caso em que $n = 1$. O canal $\langle 0, 1 \rangle$ é do nó de processamento para o primeiro nível sem redundância de canais. Para tal situação temos :

$$\overline{W}_{0,1} = \overline{W}_{M/G/1}\left(\lambda_{0,1}, \overline{x}_{0,1}\right) \tag{5.28}$$

Para qualquer canal que siga do nível "n-1"para o nível "n", a mensagem pode subir um nível com probabilidade $P_n^{\uparrow}$ ou descer um nível com probabilidade $P_n^{\downarrow}$, tendo as mesmas sido mencionadas nesta seção. Então temos o tempo de serviço médio como:

$$\overline{x}_{n-1,n} = \left[\overline{x}_{n,n+1} + \left(1 - \frac{\lambda_{n-1,n}}{\lambda_{n,n+1}} P_n^{\uparrow}\right) \overline{W}_{n,n+1}\right] P_n^{\uparrow} + \left[\overline{x}_{n,n-1} + \left(1 - \frac{P_n^{\uparrow}}{3}\right) \overline{W}_{n,n-1}\right] P_n^{\downarrow} \tag{5.29}$$

5.3.4.3 Latência

Usando a Equação 5.3, calcularemos a latência média. Para as redes *butterfly fat-tree*, $x_{inj,j} = x_{0,1}$ e $W_{inj,j} = W_{0,1}$. A latência é determinada abaixo :

$$\overline{L} = \overline{W}_{0,1} + \overline{x}_{0,1} + \left(\overline{D} - 1\right) \tag{5.30}$$

5.3.5 Gráficos resultantes das equações acima

Os gráficos foram gerados no GNUPlot (www.gnuplot.info) com a inserção das equações descritas acima.

Na Figura 5.6, variamos o tamanho da mensagem em *flits* afim de obtermos o comportamento da latência face a estas mudanças. O resultado obtido mostrou que quanto maior é número de *flits* inserido na mensagem, maior será a chance desta configuração atingir altos valores de latência.

Na Figura 5.7, comparamos as topologias *Mesh* e *Torus* com diferentes números de blocos IP´s. O resultado mostrou que a topologia em *Mesh* possui valores de latência menores que a topologia em *Torus* à medida que aumenta a taxa de envio de mensagem. Vale observar que o número de blocos IP's exerce uma influencia significativa, assim como acontece nas simulações mostrada na Figura 5.6.

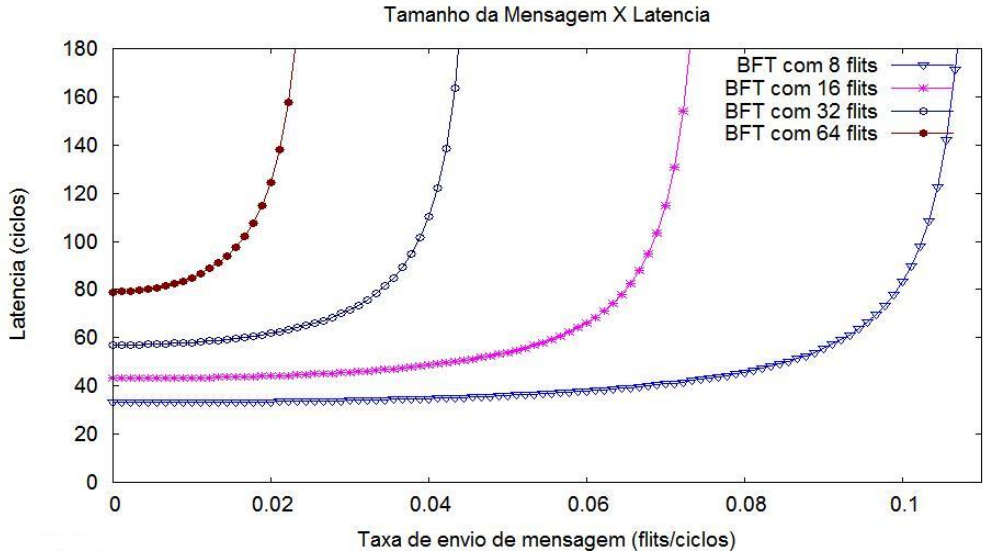

Figura 5.7: Latência de BFT

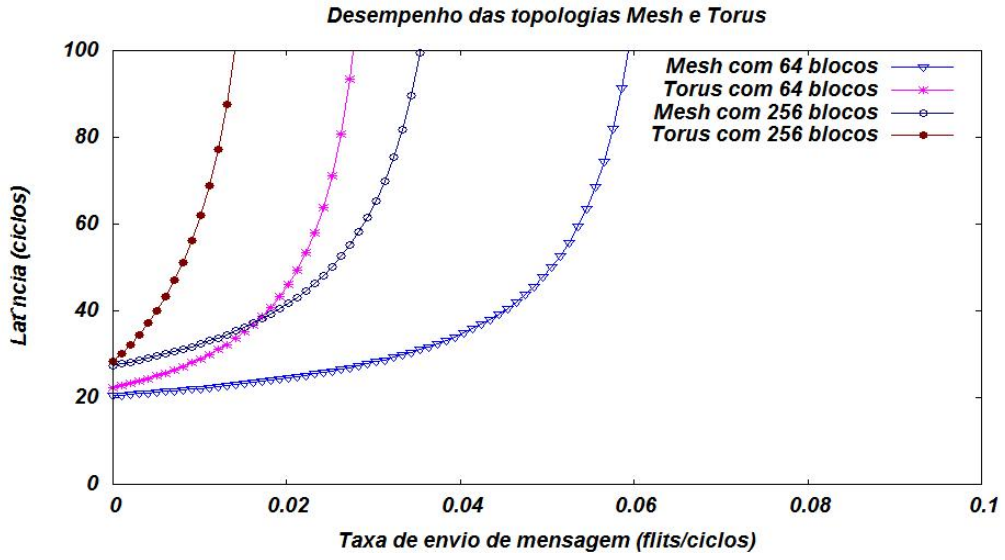

Figura 5.8: Latência Mesh and Torus

Capítulo 6

Simulações realizadas

6.1 Comparações de resultados de simulações com o modelo analítico

Ao iniciarmos os estudos sobre *Network-on-Chip*, nos vemos com o seguinte questionamento : "Qual a melhor topologia a ser implementada?". Esta pergunta não é algo trivial, pois leva em consideração uma série de fatores que foram mencionados ao longo deste trabalho.

A eficácia da transmissão dos *flits* consiste em entregá-los em seu destino com maior *troughput*, menor latência, menor consumo de energia, menor número de saltos entre origem e destino, maior utilização do canal de transmissão e de buffers de entrada/saída dos PE's e roteadores. Como podemos perceber, a lista de variáveis é extensa e mapeá-las, com todos os detalhes, consiste em uma fonte inesgotável para pesquisadores.

A idéia de projetar grandes chips com vários núcleos é uma tarefa que consome muito tempo por parte dos projetistas, e o uso de ferramentas de simulação consiste na maneira mais econômica de projetá-los. Mapear os dados de entrada e saída, e encontrar possíveis erros nesta fase de projeto, ajuda a minimizar os custos de fabricação, diminui o *time-to-market* e fornece uma idéia de como o chip fabricado irá se comportar mediante as situações em que são submetidas.

A nossa tarefa inicial é encontrar um simulador que satisfaça as condições descritas em [12]. Para confirmar a eficácia do modelo utilizado pelo simulador *gpNoCsim* [18], realizamos simulações que tentam se aproximar das métricas realizadas em [12], como números de blocos IP's, quantidade de ciclos de *clocks* e descarte nos dados obtidos durante os primeiros 10% da quantidade total de ciclos de *clocks*. Abaixo seguem os gráficos gerados pelas simulações com o *gpNoCsim*:

Os gráficos apresentados mostram níveis de saturação semelhantes aos encontrados em [12]. Com base nos resultados apresentados em nossas simulações

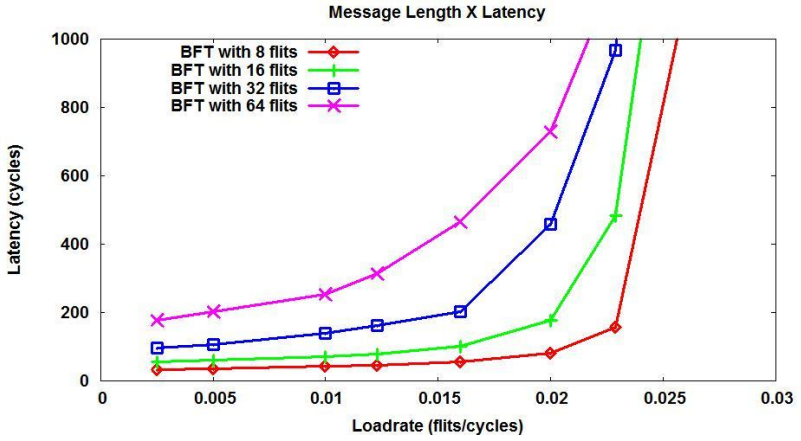

Figura 6.1: Resultado das simulações no gpNoCsim variando o tamanho da mensagem

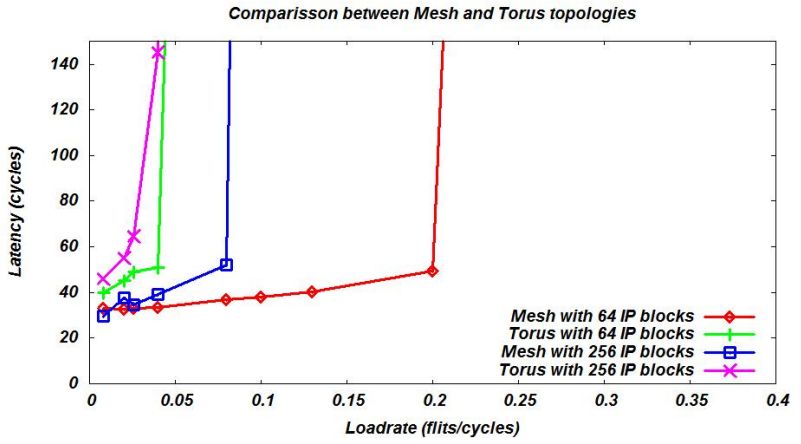

Figura 6.2: Resultado das simulações no gpNoCsim variando o número de blocos IP's

a topologia *Butterfly Fat Tree* apresenta um aumento explosivo nos valores de latência com taxas de injeção mais baixas às apresentadas nas topologias *Mesh* e *Torus*. Os itens avaliados apresentaram os seguintes resultados: (a) tamanho da mensagem: em nossas simulações com a topologia *Butterfly Fat Tree* realizamos testes com tamanhos de mensagens de 8, 16, 32, 64 *flits*, e quanto maior o tamanho da mensagem, menor será o valor da taxa de injeção; (b) quantidade de blocos IP's: as simulações envolvendo as topologia *Mesh* e *Torus* mostram altos valores de latência menores à medida que aumentamos o número de blocos IP's.

Com os resultados apresentados, podemos atestar a eficiência do simulador *gpNoCsim* para as situações apresentadas em [12]. Isto credencia *gpNoCsim* para ser usado em simulações na qual variaremos alguns parâmetros importantes de um projeto para *NoC*, como quantidade de blocos IP's, número de canais virtuais, tamanho de *buffer*, entre outros.

6.2 Variação de parâmetros

Em nossas simulações, como dito anteriormente, usaremos o *gpNoCsim* para simular diferentes topologias e as seguintes variáveis: (a) números de blocos IP´s : podemos analisar a escalabilidade da rede, e extrair dados que mostrem o comportamento a medida que variamos o quantidade de blocos IP's; (b) números de canais virtuais: o uso de canais virtuais permite uma minimização do uso dos *buffers* dos roteadores, o que impacta na área do *chip* e no gasto de energia; (c) quantidade de flits por buffer: o tamanho do roteador é determinado pelo tamanho do buffer, e o número de flits que o mesmo pode armazenar determina o espaço que será ocupado pelo buffer no *chip*, e conseqüentemente pelo roteador.

Para efeito de simulação, o arquivo de configuração injetará pacotes de 200 bytes a cada 150 ciclos em um total de 100.000 ciclos, apenas uma única vez. Para coleta das informações concernentes a performance, iremos avaliar o *troughput* e a latência obtida para cada item avaliado.

Os resultados obtidos servirão para discutirmos qual topologia apresenta métricas de performance consideradas satisfatórias para dar continuidade em outros itens que norteiam a construção de chips, tais como custo de fabricação, dissipação de calor, gasto de energia.

6.2.1 Número de blocos IP's

A indústria de circuitos integrados faz uso intenso de blocos IP, o que torna uma tarefa árdua para os projetistas selecionar e juntar os blocos IP´s atingindo funcionalidades precisas e atendendo aos requisitos de *time-to-market*, o que exige um tempo reduzido para disponibilizar o *chip* em condições de ser comercializado. Para atender os prazos de entrega e a crescente complexidade nos projetos de

circuitos integrados, os requisitos de modularidade e padronização nas interfaces de interconexão constituem como exigências primordiais de arquitetura em NoC. A escalabilidade, que permite a adição de novos blocos IP's sem prejuízo do desempenho do sistema como um todo, ainda é prematuro definir o grau de escalabilidade que uma NoC pode oferecer, mas o uso de simuladores nos ajuda a ter uma noção do impacto no aumento do número de blocos IP.

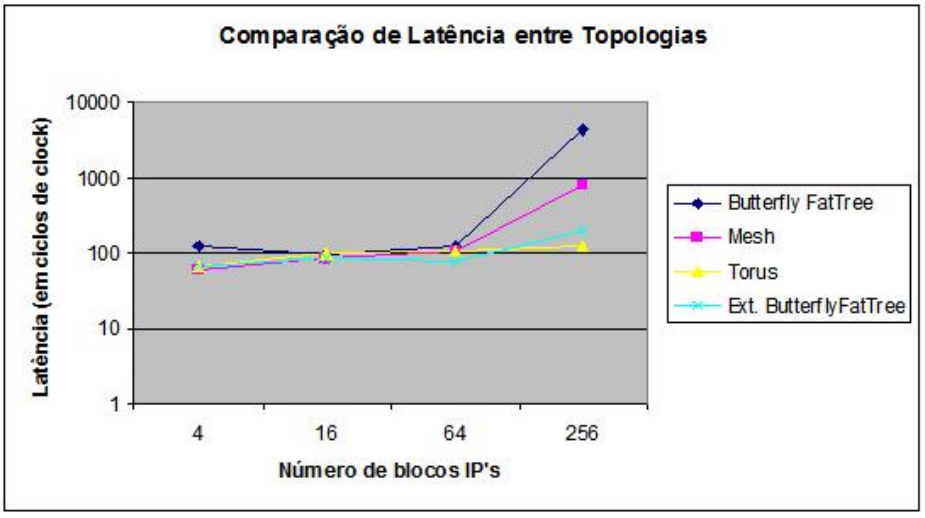

Figura 6.3: Variação de latência entre Blocos IP's

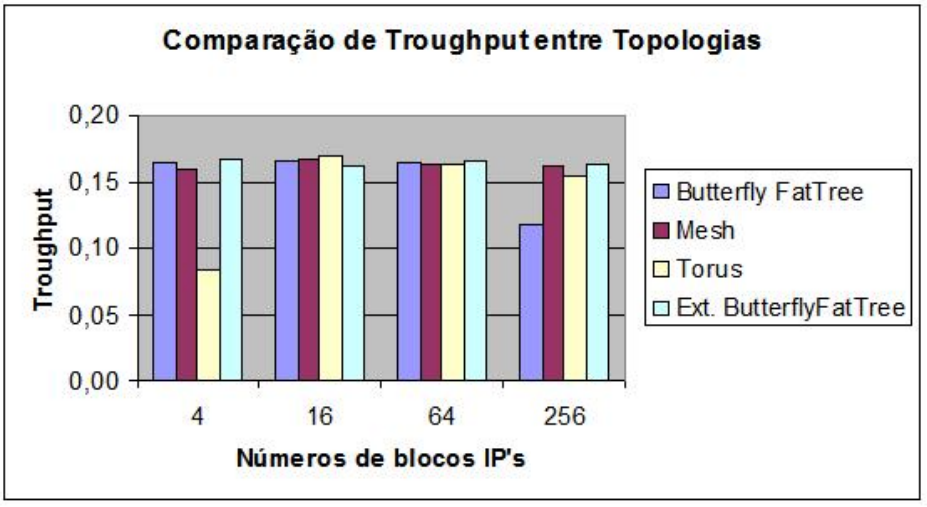

Figura 6.4: Variação de throughput entre Blocos IP's

6.2.2 Canais Virtuais

Nos projetos de roteadores para *NoC*, o campo de pesquisa tem focado na implementação de canais virtuais, com o intuito de oferecer economia de energia e redução na área ocupada pelos mesmos, considerado os grandes obstáculos para a

fabricação de chips. Em [22] é explicado sobre a utilidade de usar canais virtuais como um meio de aumentar o *troughput* e diminuir a latência entre as mensagens que trafegam na rede, seja microscópica ou macroscópica. Os mesmos fornecem múltiplos buffers para cada entrada ou saída do roteador, impedindo que um único pacote (ou *flit*) congestione o canal de comunicação. O uso de canais virtuais está associado à técnica de roteamento *wormhole* e fornece uma possibilidade de diferentes implementações. Em [26] propõe-se uma arquitetura de roteador para canais virtuais que difere da tradicional, em que os mesmos não são multiplexados depois de FIFOs em cada bloco de entrada, mas conectados diretamente no barramento. Os resultados desta implementação mostram que é possível reduzir em até pela metade o tamanho do roteador e aumentar a velocidade em 40 %.

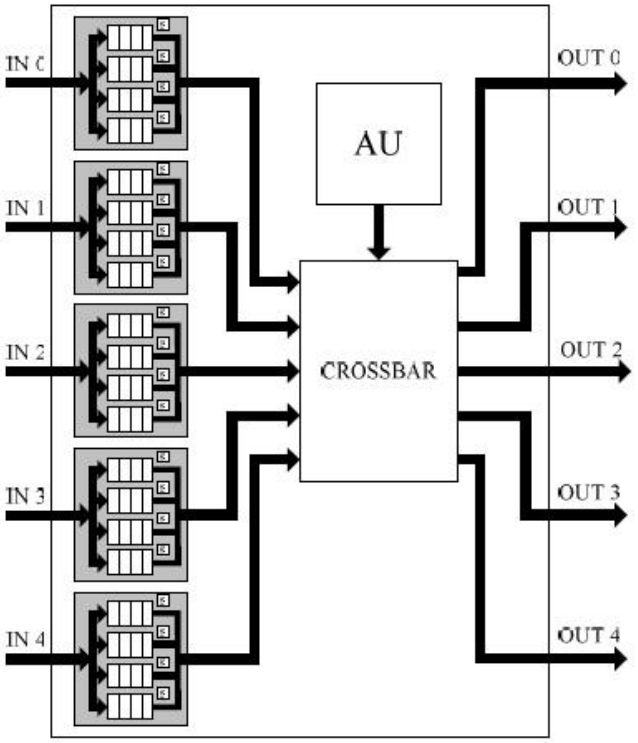

Figura 6.5: Estrutura geral de um roteador com 5 portas e 4 canais virtuais

Em nossas simulações, o aumento do números de canais virtuais possui um impacto maior na topologia de Butterfly Fat Tree até 20 canais virtuais, no entanto quando usamos 40 canais virtuais esta topologia permanece, no que tange a *troughput*, no mesmo patamar que as outras topologias. Com os resultados envolvendo latência, a topologia Butterfly Fat Tree apresenta uma discrepância em relação às outras até 20 canais virtuais, conseguindo alcançar valores compatíveis com às outras topologias com 40 canais virtuais. Vale lembrar que um número muito grande de canais virtuais impacta diretamente na área do roteador e no gasto de energia, o que em nossa avaliação preliminar torna esta topologia inviável para esta situação.

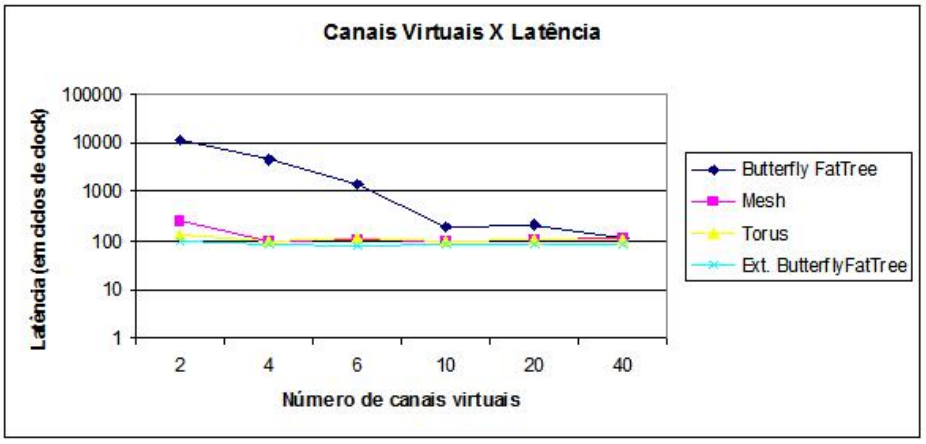

Figura 6.6: Variação de latência com Canais Virtuais

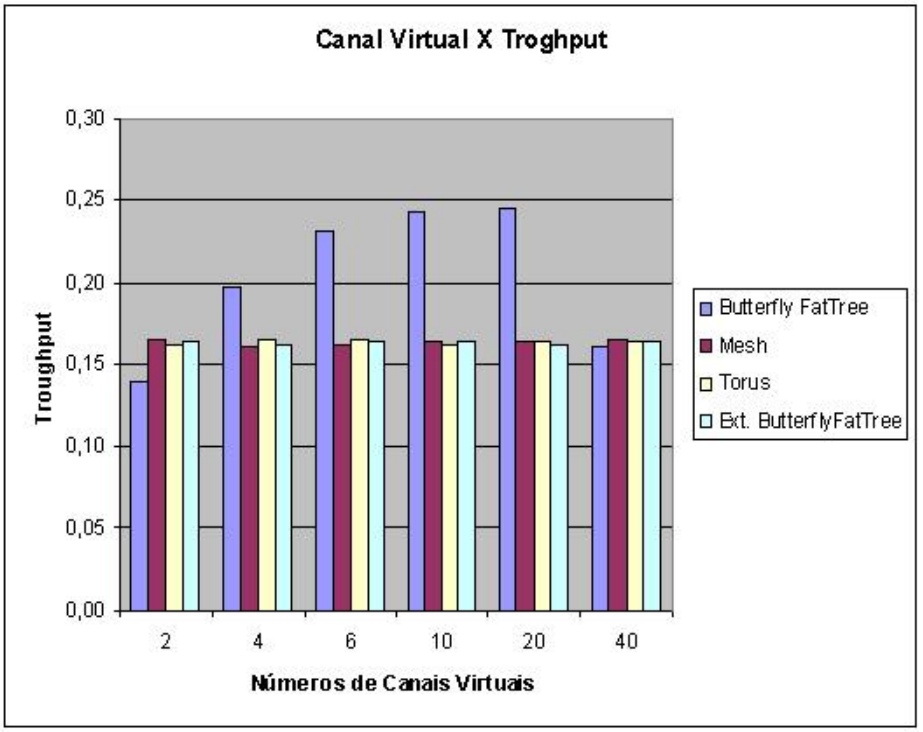

Figura 6.7: Gráfico mostra os diferentes valores de troughput entre as diferentes topologias apresentadas

6.2.3 Buffer

Em uma topologia em rede, um chip usa uma arquitetura de comunicação que consiste de vários clientes (processadores, memórias e blocos lógicos customizados) nas quais são conectados a uma rede que roteia pacotes entre eles. Cada bloco IP necessita de um roteador e consiste de vários controladores de entrada/saída. Com o intuito de otimizar essa comunicação os dispositivos de rede usam diversos recursos. O uso de *buffer* para armazenar pacotes que chegam aos nós da rede evita que os mesmos sejam devolvidos e assim retransmitidos, o que causaria um congestionamento no canal de comunicação. A otimização do tamanho do *buffer* torna-se crítico, pois a área do roteador é dominada pelo espaço ocupado pelo buffer, segundo [41].

Quando abordamos situações microscópicas, um tamanho inapropriado de *buffer* pode causar redução na performance e aumentar a dissipação de calor.No *gpNoCSim*, o canal virtual é simplesmente um *buffer*, que armazena temporariamente tantos objetos quantos canais virtuais são exigidos. Em nossa simulação, variamos a quantidade de *flits* (*flow digits*) que um canal virtual pode suportar de *buffer*, e analisaremos o comportamento desta variação dentre as topologias mencionadas ao longo deste documento.

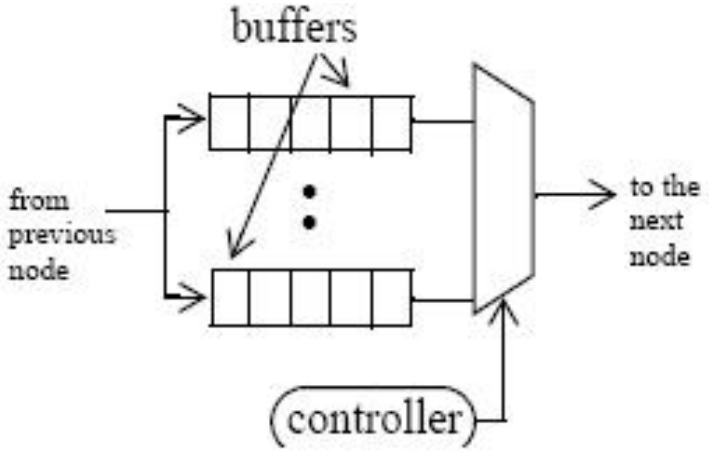

Figura 6.8: Estrutura de um buffer

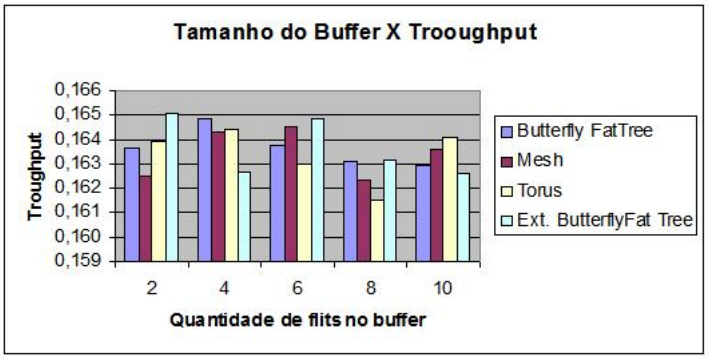

Figura 6.9: Gráfico mostrando a variação de troughput

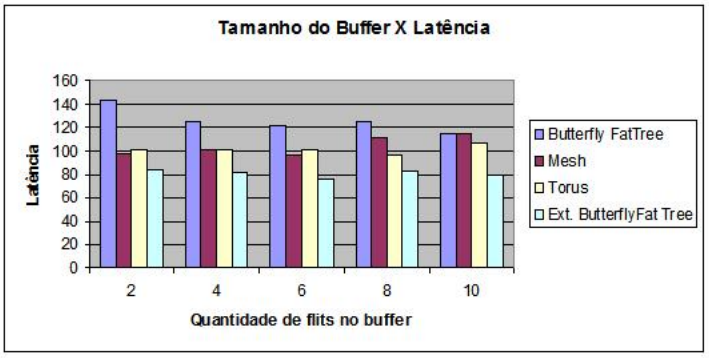

Figura 6.10: Gráfico mostrando a variação de latência

Capítulo 7

Conclusão

Ao iniciarmos os estudos sobre *NoC*, procuramos mapear os aspectos mais importantes para a implementação de *NoC* como roteamento das informações intra-chip, o consumo de energia e a área do chip. Mediante a análise destes itens, mostramos algumas implementações de NoC que têm feito uso das características que tornam a *NoC* uma abordagem bastante atratativa em relação a arquitetura em barramento.

A contribuição desta dissertação consiste em mostrar qual topologia apresenta melhores resultados de latência e *throughput*, devemos observar que não mencionamos para que tipo de aplicação esta topologia funcionará. Fizemos uma validação do simulador *gpNocSim* com o modelo analítico apresentado em [12], os quais apresentaram resultados congruentes e então, iniciamos estudos sobre outras situações que norteiam a construção de um chip.

Para a análise do comportamento das topologia para *NoC*, mostramos o modelo analítico descrito em [12], que fornece equações que nos ajudaram a calcular métricas de desempenho para *NoC*. O uso de simuladores vem a preencher uma lacuna, que não poderíamos fazer em implementações reais devido ao alto custo que implicaria, ajudando a manipular os dados de entrada de uma *NoC* como quantidade de blocos IP's, taxa de envio de mensagens, quantidade de canais virtuais, tamanho da mensagem, entre outros.

De acordo com as simulações efetuadas, temos 2 (duas) métricas que servem para iniciarmos os estudos em torno de topologia e arquitetura internas dos chips que mostraram resultados satisfatório. No nosso experimento, a topologia em Butterfly Fat Tree apresentou resultados diferenciados a medida que aumentamos o número de blocos IP's e diminuimos o número de canais virtuais, quanto às outras topologias apresentaram resultados semelhantes na análise de troughput e latência. Para uma possível implementação, devemos analisar o impacto de outras métricas como custo de implementação, capacitância gerada pelo número de fios, dissipação de calor. A topologia em Mesh apresentou resultados favoravéis nos itens de latência e *throughput* em nosso experimento, mas como mencionado

anteriormente neste documento, as métricas aqui expostas servem apenas como passo inicial na avaliação de topologia e arquitetura dos chips, que venham a ser construído usando esta metodologia.

Para trabalhos futuros, seria necessário uma implementação real de uma *NoC* para explorar os itens de modularidade e escalabilidade, o que será de grande proveito para a inserção de novas funcionalidades nos chips já construídos. Por exemplo, o comportamento do chip com a implementação de módulo criptográfico que assim o exigisse. Como a *NoC* possui a característica de modularidade, o módulo de criptografia seria um componente opcional e seria usado em situações em que o sigilo das informações fosse relevante.

Bibliografia

[1] A network on chip architecture and design methodology. In *ISVLSI '02: Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, page 117, Washington, DC, USA, 2002. IEEE Computer Society.

[2] Network-on-chip company. Arteris, 2005.

[3] Hellebrand S. Ali M., Welzl M. A dynamic routing mechanism for network on chip. In *Proceedings of IEEE NORCHIP, 2005. 23rd*, pages 70–73, 2005.

[4] N. Alzeidi, A. Khonsari, M. Ould-Khaoua, and L.M. Mackenzie. A new queueing model for the analysis of virtual channels occupancy in wormhole-switched networks. In *Technical Report TR-2005*, page 206, Glasgow, Scotland, 2005. Dept of Computing Science, University of Glasgow.

[5] Arteris. A comparison of network-on-chip and busses. In *Arteris*, page 8, Paris, FR, 2005. White paper.

[6] J. Blazewicz, D. P. Bovet, J. Brzezinski, G. Gambosi, and M. Talamo. Optimal centralized algorithms for store-and-forward deadlock avoidance. *IEEE Trans. Comput.*, 43(11):1333–1338, 1994.

[7] Andrew A. Chien. A cost and speed model for k-ary n-cube wormhole routers. *IEEE Trans. Parallel Distrib. Syst.*, 9(2):150–162, 1998.

[8] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[9] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Comput.*, 39(6):775–785, 1990.

[10] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference*, pages 684–689, 2001.

[11] W.J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 04(4):466–475, 1993.

[12] J. T. Draper and J. Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *Journal of Parallel and Distributed Computing*, 23(2):202–214, 1994.

[13] Maged Ghoneima, Yehea Ismail, Muhammad Khellah, and Vivek De. Variation-tolerant and low-power source-synchronous multicycle on-chip interconnect scheme. *VLSI Design*, 2007:Article ID 95402, 12 pages, 2007. doi:10.1155/2007/95402.

[14] Kees Goossens, John Dielissen, and Andrei Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, 2005.

[15] R. Greenberg and L. Guan. Modeling and comparison of wormhole routed mesh and torus networks. In *Proc. Ninth IASTED Int'l Conf. Parallel and Distributed Computing and Systems*, 1997.

[16] Ronald I. Greenberg and Lee Guan. An improved analytical model for wormhole routed networks with application to butterfly fat-trees. In *ICPP '97: Proceedings of the international Conference on Parallel Processing*, pages 44–48, Washington, DC, USA, 1997. IEEE Computer Society.

[17] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 250–256, New York, NY, USA, 2000. ACM.

[18] Hemayet Hossain, Mostak Ahmed, Abdullah Al-Nayeem, Tanzima Zerin Islam, and Md. Mostofa Akbar. *A General Purpose Simulator for Network-on-Chip*. Washington, DC, USA, 7-9 March 2007.

[19] Islam M. Hossain H., Akbar M. Extended-butterfly fat tree interconnection (efti) architecture for network on chip. *Communications, Computers and signal Processing, 2005. PACRIM. 2005 IEEE Pacific Rim Conference on*, pages 613–616, 24-26 Aug. 2005.

[20] Jingcao Hu and Radu Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10688, Washington, DC, USA, 2003. IEEE Computer Society.

[21] Jingcao Hu and Radu Marculescu. Dyad: smart routing for networks-on-chip. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 260–263, New York, NY, USA, 2004. ACM.

[22] Ting-Chun Huang, Umit Y. Ogras, and Radu Marculescu. Virtual channels planning for networks-on-chip. In *ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 879–884, Washington, DC, USA, 2007. IEEE Computer Society.

[23] Axel Jantsch and Hannu Tenhunen, editors. *Networks on chip*. Kluwer Academic Publishers, Hingham, MA, USA, 2003.

[24] Xin Jia and Ranga Vemuri. Cad tools for a globally asynchronous locally synchronous fpga architecture. In *VLSID '06: Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*, pages 251–256, Washington, DC, USA, 2006. IEEE Computer Society.

[25] Faraydon Karim, Anh Nguyen, and Sujit Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, 2002.

[26] Jansen P.G. Kavaldjiev N., Smit G.J.M. A virtual channel router for on-chip networks. *SOC Conference, 2004. Proceedings. IEEE International*, pages 289–293, 12-15 Sept. 2004.

[27] Rakesh Kumar, Victor Zyuban, and Dean M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. *SIGARCH Comput. Archit. News*, 33(2):408–419, 2005.

[28] Jian Liu, Li-Rong Zheng, and Hannu Tenhunen. Interconnect intellectual property for network-on-chip (noc). *Journal of Systems Architecture*, 50(2-3):65–79, 2004.

[29] Mateusz Majer, Christophe Bobda, Ali Ahmadinia, and Jurgen Teich. Packet routing in dynamically changing networks on chip. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 154.2, Washington, DC, USA, 2005. IEEE Computer Society.

[30] Radu Marculescu, Jan Rabaey, and Alberto Sangiovanni-Vincentelli. Is "network"the next "big idea"in design? In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 254–256, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[31] Paolo Meloni, Igor Loi, Federico Angiolini, et al. Area and power modeling for networks-on-chip with layout awareness. *VLSI Design*, 2007:Article ID 50285, 12 pages, 2007. doi:10.1155/2007/50285.

[32] Fernando Gehm Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration*, 38(1):69–93, 2004.

[33] Srinivasan Murali, David Atienza, Luca Benini, and Giovanni De Micheli. A method for routing packets across multiple paths in nocs with in-order delivery and fault-tolerance gaurantees. *VLSI Design*, 2007:Article ID 37627, 11 pages, 2007. doi:10.1155/2007/37627.

[34] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[35] J. Nurmi. Network-on-chip: A new paradigm for system-on-chip design. *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, pages 2–6, 15-17 Nov. 2005.

[36] Hokstad P. Approximations for the m/g/m queue. In *Operations Research 1978;26(3)*, pages 510–523, 1978.

[37] Partha Pratim Pande, Cristian Grecu, André Ivanov, and Res Saleh. Design of a switch for network on chip applications. In *ISCAS (5)*, pages 217–220, 2003.

[38] C. S. Patel. Power constrained design of multiprocessor interconnection networks. In *ICCD '97: Proceedings of the 1997 International Conference on Computer Design (ICCD '97)*, page 408, Washington, DC, USA, 1997. IEEE Computer Society.

[39] Vijay Raghunathan, Mani B. Srivastava, and Rajesh K. Gupta. A survey of techniques for energy efficient on-chip communication. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 900–905, New York, NY, USA, 2003. ACM.

[40] J. Rexford and K. Shin. Shortest-path routing in homogeneous point-to-point networks with virtual cut-through switching. In *Technical Report CSE-TR-14692*, 1992.

[41] I. Saastamoinen, M. Alho, and J. Nurmi. Buffer implementation for proteo network-on-chip. *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, 2:II–113–II–116 vol.2, 25-28 May 2003.

[42] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P.P. Pande, C. Grecu, and A. Ivanov. System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 94(6):1050–1069, June 2006.

[43] David A. Sigüenza-Tortosa and Jari Nurmi. Packet scheduling in proteo network-on-chip. In M. H. Hamza, editor, *Parallel and Distributed Computing and Networks*, pages 116–121. IASTED/ACTA Press, 2004.

[44] William Stallings. *Data and computer communications (5th ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[45] Daniel Wiklund and Dake Liu. Socbus: Switched network on chip for hard real time embedded systems. In *IPDPS*, page 78. IEEE Computer Society, 2003.

[46] Dong Wu, Bashir M. Al-Hashimi, and Marcus T. Schmitz. Improving routing efficiency for network-on-chip through contention-aware input selection. In *ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation*, pages 36–41, Piscataway, NJ, USA, 2006. IEEE Press.

[47] Terry Tao Ye, Luca Benini, and Giovanni De Micheli. Packetized on-chip interconnect communication analysis for mpsoc. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10344, Washington, DC, USA, 2003. IEEE Computer Society.

[48] Terry Tao Ye, Luca Benini, and Giovanni De Micheli. Packetization and routing analysis of on-chip multiprocessor networks. *J. Syst. Archit.*, 50(2-3):81–104, 2004.

[49] Terry Tao Ye, Giovanni De Micheli, and Luca Benini. Analysis of power consumption on switch fabrics in network routers. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 524–529, New York, NY, USA, 2002. ACM.

[50] Cesar Albenes Zeferino and Altamiro Amadeu Susin. Socin: A parametric and scalable network-on-chip. In *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pages 169–174. IEEE Computer Society, 2003.

Apêndice A

Simulador gpNoCsim

A.1 Introdução

A decisão de projetar chips em rede baseia-se nos resultados de simulações, antes de uma possível implementação, pois é mais barato e mais flexível realizar testes em softwares que nos mostre uma visão mais realista que será mostrada na implementação. Não existem muitos simuladores para arquiteturas em NoC. Geralmente, os pesquisadores constroem visando suprir suas próprias necessidades. Em [18], é mostrado um simulador construído em Java, na qual suporta diferentes arquiteturas usadas para NoC : Mesh, Torus, Butterfly Fat Tree, Extended Butterfly Fat Tree, etc. Além disso, a arquitetura modular do simulador, característica da linguagem de programação usada, fornece a possibilidade de incorporar novas arquiteturas de rede ao simulador. A versão do gpNoCsim está disponível em *http://www.buet.ac.bd/cse/research/group/noc/* em um único arquivo com formato ".ZIP". Pode ser executado em qualquer implementação em JDK 1.2.2 ou outra versão mais atual. Para compilar o código, deve-se extrair o arquivo em um diretório e mudar para o diretório onde o mesmo foi extraído e executar o seguinte comando :
javac *.java

O simulador pode ser executado com o seguinte comando :

java gpNoCsim

Como mostrado acima, *gpNoCsim.class* é a classe principal e os arquivos de entrada de dados para o simulador é o *nocSimParameter.txt*. Os resultados da simulação são escritos no arquivo *nocSimOutput.txt*. Os exemplos dos arquivos de configuração e resultados correspondentes são ilustrados nas subseções a seguir, neste capítulo.

A.2 Configuração do tráfego

No gpNoCsim, os usuários podem definir os diferentes tipos de distribuição de tráfego selecionando as inúmeras variáveis que podem ser configuradas, a citar algumas:

- Qual topologia a ser utilizada ?;
- Intervalo de envio de mensagem, que é exponencialmente distribuído;
- Número de flits de uma mensagem que pode ser fixo ou exponencialmente distribuído;
- Tamanho de um flits em bytes;
- Número de nós que irão compor a topologia a ser simulada;

Toda a informação usada para configurar uma simulação é inserida no arquivo (*nocSimParameter.txt*) como ilustrado abaixo. O arquivo abaixo é configurado para uma topologia em Mesh de 8 X 8 nós com uma geração de mensagem a cada 200 ciclos e com 4 canais virtuais por link físico. O tamanho médio da mensagem é de 200 bytes e o tamanho do flit é de 64 bits. A simulação é executada por 100.000 ciclos. Todos os nós operam de maneira assíncrona com os switches da rede.

- **CURRENT_NET = 1**
- **AVG_INTER_ARRIVAL = 150**
- **AVG_MESSAGE_LENGTH = 200**
- **FLIT_LENGTH = 64**
- **NUMBER_OF_IP_NODE = 64**
- **CURRENT_VC_COUNT = 4**
- **NUM_FLIT_PER_BUFFER = 2**
- **NUM_CYCLE = 100000**
- **NUM_RUN = 1**
- **TRAFFIC_TYPE = 0**
- **WARM_UP_CYCLE = 0.1**
- **FIXED_MESSAGE_LENGTH = false**
- **TRACE = false**

- **ASYNCHRONOUS = true**

Abaixo segue as definições dos itens acima:

CURRENT_NET: Define a topologia .0 (Butterfly Fat Tree), 1(Mesh), 2 (Torus), 3 (Extended Butterfly Fat Tree).

AVG_INTER_ARRIVAL: Mostra a taxa de geração de mensagens nos nós de origem. O intervalo atual de geração entre mensagens é calculado por uma distribuição exponencial usando este parâmetro. No exemplo acima, mostra que a cada 150 ciclos é produzido um pacote. A mudança deste parâmetro pode aumentar ou diminuir a carga da rede.

AVG_MESSAGE_LENGTH: Significa o tamanho da mensagem (tamanho do pacote, em bytes) gerada nos nós de origem. O tamanho do pacote é calculado por uma distribuição exponencial usando este parâmetro quando FIXED_MESSAGE_LENGTH=false. A mudança deste parâmetro pode aumentar ou diminuir a carga da rede.

FLIT_LENGTH : Tamanho do flit (cabeçalho e dados do flit) em bits. Se o tamanho é muito pequeno para transportar uma informação mínima para carregar o cabeçalho do flit então é acrescido em número de bits. O valor 64, no exemplo acima, significa que a cada flit que atravessa a rede será de 64 bits e os pacotes serão desmontados em tamanho fixo de 64 bits.

CURRENT_VC_COUNT: O número de canais virtuais em cada canal físico na rede, por exemplo 2,4,6,8, etc.

NUM_FLIT_PER_BUFFER: Tamanho do buffer de entrada e do buffer de saída em flits, isto é, 2, 4, 6, etc.

TRAFFIC_TYPE: Existem 2 tipos de tráfego : 0 para tráfego uniforme e 1 para localizado. A versão 1.0 do gpNoCsim possui implementado o tráfego uniforme.

NUM_CYCLE: O número de ciclos de simulações que a simulação ocorrerá.

NUM_RUN: O número de simulações que será executada para a coleta da média de valores para os parâmetros de saída

WARM_UP_CYCLE: 0.1 significa que 10 por cento do número total de ciclos NUM_CYCLE). Nenhum dado estatístico é acumulado durante os ciclos iniciais do warm up.

TRACE: 'true' para escrever os resultados no arquivo nocSimTrace.txt , 'false' não a realiza.

ASYNCHRONOUS: 'true' aponta para clocks dos nós em modo assíncrono com os dos switches, 'false' opera os nós de maneira síncrona com os switches.

A.3 Arquivo com Resultado das Simulações

O arquivo acima mostra inicialmente a configuração do arquivo de entrada de dados, o qual já foi explicado na seção 5.2. Em seguida fornecem dados que irão nos permitir uma avaliação no que tange ao desempenho da rede em questão. Os resultados fornecem dados como *throughput*, latência, utilização de *buffer*, número médio de saltos dentro da rede que uma mensagem leva para chegar ao destino. Os cálculos das métricas mencionadas serão detalhados na seção a seguir.

Input Configuration: .....
Network 1
Asynchronous Flow
Number of IP 64
Fixed Message Length false
Avg Msg Length(bytes) 200
Flit Length(bits) 64
Avg Msg Production Rate(cycle/msg) 150
Number of Virtual Channel/Link 4
Number of Flit/Buffer 2
Number of Cycles 100000
Number of Warm_Up_Cycles 10000
Traffic Type 0
.....
Performance Measurements.....
Throughput[Net]0.163759375
Throughput[Flits leaving Switch] 1.0379309375
Avg Packet Delay 109.25642838624526
Link Utilization 0.21848383522727272
Buffer(Node Switch) Utilization 0.0027072886705233564
Avg Packet Injection Rate 0.42409
Avg Packet Not Produced 0.0
Input Buffer(Node) Utilization 0.020469921875
Input Buffer(Switch) Utilization 0.0010814809841579861
Output Buffer(Node) Utilization 0.1022590234375
Output Buffer(Switch) Utilization 0.0012090658230251737
Avg Hop Count 7.342289471823517
Avg Packet Sent/Run 42409.0
Avg Packet Received/Run 42429.0

A.4 Métricas de Desempenho

Para comparar e contrastar as diferentes arquiteturas para *NoC*, devemos nos atentar nas variantes que norteiam a tendência de interligar os chips em rede, como número de blocos IP, número de canais virtuais, taxa de injeção de pacotes na rede, quantidade de *flits* nos *buffers*. Como analogia, as métricas que utilizaremos para analisar uma rede de chips é similar à uma rede de computadores. Para tal é desejável que a rede de

chips analisada tenha alto *throughput* , baixa latência, uso eficiente de energia e que os micro-dispositivos ocupem a menor área possível. Infelizmente, o gpNoCsim, em sua verão 1.0, não contempla os cálculos de energia e área. Para início de nossa pesquisa, consideramos o desempenho do chip, medido através de *throughput* e latência, como ponto de partida para a construção do mesmo. Os requerimentos de energia, área, custo para a construção e outros que se fizerem necessários serão detalhados em seguida, com os requerimentos de throughput e latência atingidos níveis aceitáveis para a aplicação que se destina, como vemos em [5]. A seguir, temos uma descrição das métricas de desempenho utilizadas pelo gpNoCsim :

A.4.1 Latência

A seguir a fórmula utilizada para cálculo de latência pelo gpNoCsim :

$$AvgPacketDelay = \frac{\sum_{i=1}^{N} PacketDelay_i}{N},$$
$$\text{onde N = NumberofPacketsReceived.}$$

Na expressão acima, *AvgPacketDelay* refere-se a média de todos os atrasos ocorridos na entrega dos pacotes, analisar a média nos permite ter uma visão mais global desta métrica a analisar pacote a pacote. A unidade de latência é em ciclos/pacote. A latência é definida como o tempo (em ciclos de clock) que leva da ocorrência da injeção do cabeçalho da mensagem na rede no nó de origem e a ocorrência da recepção da última parte da mensagem no nó de destino. Para alcançar o nó de destino, os flits percorrerão caminhos alternativos, de acordo com o algoritmo de roteamento, em [4] temos uma descrição dos resultados de um roteamento dinâmico e suas aplicabilidades em *NoC*. Dependendo do par origem/destino e do algoritmo de roteamento, as mensagens terão diferentes latências, para fins deste documento usaremos a média de latências dos pacotes, como uma das métricas a serem comparadas com o modelo matemático proposto na seção anterior, com o intuito de encontrar subsídios que nos permitam a encontrar uma topologia adequada para as situações indicadas neste trabalho.

A.4.2 Throughput

O número total de flits recebidos (*Totalnumberofflitsreceived*) refere-se ao número de *flits* (*flow digits*) que são recebidos pelos nós. O tamanho dos flits não é um valor absoluto e pode ser configurado no arquivo de entrada de dados para as simulações, no nosso caso usaremos o tamanho de 64 *bits*, o que significa dizer que, todo *flit* que será encaminhado através da rede terá o tamanho de 64 bits e os pacotes serão desempacotados em tamanho de 64 *bits*. A variável *TotalCycleCount* é definido como o número total de ciclos que a simulação sofrerá, por padrão temos 100.000 ciclos. A unidade de *troughput* é em *flits*/(nós x ciclos) para efeitos de simulação, pois a unidade de clock para simulação pode variar de acordo com o hardware utilizado, o qual poderá subestimar ou superestimar este valor. Iremos observar com os resultados mostrados, ao longo deste trabalho, que vários fatores irão influenciar o troughput como topologia usada,

números de blocos IP's (*intellectual property*), número de *flits* por *buffer*, números de canais virtuais, taxa de injeção de *flits* na rede.

A seguir a fórmula usada pelo gpNoCsim para cálculo de throughput:

$$Troughput = \frac{TotalNumbersOfFlitsReceived}{NumberOfIPNodes \times TotalCycleCount}$$

A.4.3 Utilização do Link

Esta é a indicação de quanto tempo a ligação física, entre os nós, é usada para transferir dados, em relação ao tempo total da duração desta transferência. O cálculo é efetuado através da divisão da soma do número de ligações físicas usados na transferência de *flits* em cada ciclo de simulação pela multiplicação do número total de ciclos de simulações e o número total de ligações físicas.

A utilização do link é computada pela seguinte fórmula :

$$LinkUtilization = \frac{\sum_{i=1}^{C}\left[\sum_{j=1}^{S} LinksUsedInSwitch_{i,j} + \sum_{j=1}^{N} LinksUsedInNode_{i,j}\right]}{TotalPhysicalLinks \times C}$$

Onde :

- $LinksUsedInSwitch_{i,j}$: Número de ligações de saída no Switch "j"usado no Ciclo "i"para transmitir os flits
- $LinkUsedInNode_{i,j}$: Número de ligações de saída no Nó "j"no Ciclo "i"para transmitir os flits
- C = Número total de Ciclos de Simulação
- N = Número Total de Nós
- S = Número Total de Switches

A.4.4 Utilização do Buffer

A utilização do *buffer* mostra a porcentagem dos *buffers* de entrada e saída que são usados para armazenar *flits*. O cálculo é efetuado pela divisão da soma do número total de *flits*, usados para armazená-los em cada simulação nos *buffers* de entrada e saída pela multiplicação do número total de ciclos de simulação e o número total de *flits* dos *buffers* de entrada e saída.

A utilização de *buffer* é computada pela seguinte fórmula:

$$BufferUtilization = \frac{\sum_{i=1}^{C}\left[\sum_{j=1}^{S} BufferSlotsUdesInSwitch_{i,j} + \sum_{j=1}^{N} BufferSlotsUsedInNode_{i,j}\right]}{NumVC \times FlitsPerBuffer \times C \times (TotalPhysicalLinks + N)}$$

Onde :

- $BufferSlotsUdesInSwitch_{i,j}$: Número de *slots* dos *buffers* e de saída ocupados por *flits* no switch "j"no ciclo "i";
- $BufferSlotsUsedInNode_{i,j}$: Número de *slots* dos *buffers* e de saída ocupados por *flits* no nó "j"no ciclo "i";
- C : Número total de ciclos de simulação;
- N : Número total de nós;
- NumVC : Número de canais virtuais por *link*;
- FlitsPerBuffer : Número de flits armazenados por *buffer*

A.4.5 Número Médio de Saltos

Mostra a média do número de saltos que um pacote percorreu:

$$AvgHopCount = \frac{\sum_{i=1}^{N} PacketHop_i}{N}$$

onde:

- N = número de pacotes;
- $PacketHop_i$ = Número de saltos que um pacote "i"percorreu.

A.5 Arquitetura do Simulador gpNoCsim

A.5.1 A Estrutura do Switch

No *gpNoCsim*, cada *switch* possui um número de portas para conectar-se com os *switches* adjacentes e com os blocos de propriedade intelectual. Cada porta inclui um conjunto de 2(dois) *buffers* (um de entrada e outro de saída), um roteador e um controlador. O roteador decodifica o cabeçalho e determina a porta de saída. O controlador é particionado em outros 2(dois) controladores, uma para o canal de saída e outro para o canal de entrada. O algoritmo implementado é o de *round-robin*.

A.5.2 Técnica de Comutação

O *gpNoCsim* usa a técnica de comutação wormhole[seção 4.3.3] e usa canais virtuais nas portas de entrada e saída. Esta técnica de comutação funciona da seguinte maneira,

o primeiro *flit*(*flow digit*), isto é, o cabeçalho do pacote possui informações de roteamento, o mesmo é decodificado e possibilita os *switches* para estabelecer um caminho, entre origem e destino, e os *flits* seguintes seguem este caminho já estabelecido. Como resultado, cada *flit* da mensagem do pacote é simplesmente encaminhada no mesmo canal de saída que os *flits* antecedentes e conseqüentemente, não há necessidade de ordenar os pacotes no destino.

A.5.3 Configuração do tráfego de rede

No *gpNoCsim*, os usuários podem definir tipos de distribuição no tráfego a ser injetado na rede. O intervalo de tempo com o qual as mensagens são geradas em um determinado tempo nó é exponencialmente distribuído, como também o número de flits em uma mensagem. O tamanho do flit pode ser configurado no simulador, convém afirmar que o mesmo deverá ser avaliado.

A.5.4 Fluxograma do Simulador

- 1. Os parâmetros de entrada são lidos no arquivo nocSimParameter.txt;
- 2-3. Se existir alguma configuração de rede no arquivo de entrada, então o programa salta para o item 4. Caso contrário a simulação é terminada;
- 4-5. A rede é criada e os parâmetros de desempenho são inicializados;
- 6. Se a quantidade de simulações é completada, o programa salta para o item 18;
- 7-9. Para cada simulação, a rede e os contadores estatísticos são inicializados e os eventos iniciais da rede são marcados;
- 10. Se todos os ciclos desta etapa estão completos, então o fluxo da simulação salta para o item 17. Caso contrário salta para o item 11;
- 11. Para cada nó, o tráfego de saída é movido do seu centro de mensagem para o switch adjacente, se possível
- 12. Para cada switch, o caminho requisitado do tráfego de entrada é atualizado;
- 13. Para cada switch, o tráfego de entrada é movido do buffer de entrada para o buffer de saída, se possível;
- 14. Para cada switch, o tráfego de entrada é movido do buffer de saída do switch para o buffer de entrada dos switches ou nós adjacentes;
- 15. Para cada nó, o tráfego de entrada é movido do buffer de entrada do nó para o centro de mensagem do mesmo;
- 16. Para cada nó e switch, o status é atualizado ao final de cada ciclo;
- 17. Os parâmetros estatísticos são atualizados para esta etapa de simulação. Então o fluxo do programa salta para o item 6;

- 18. Depois de todas as etapas são completas, os parâmetros de desempenho desta configuração de rede são escritas no arquivo de saída e o fluxo salta para o item 2.

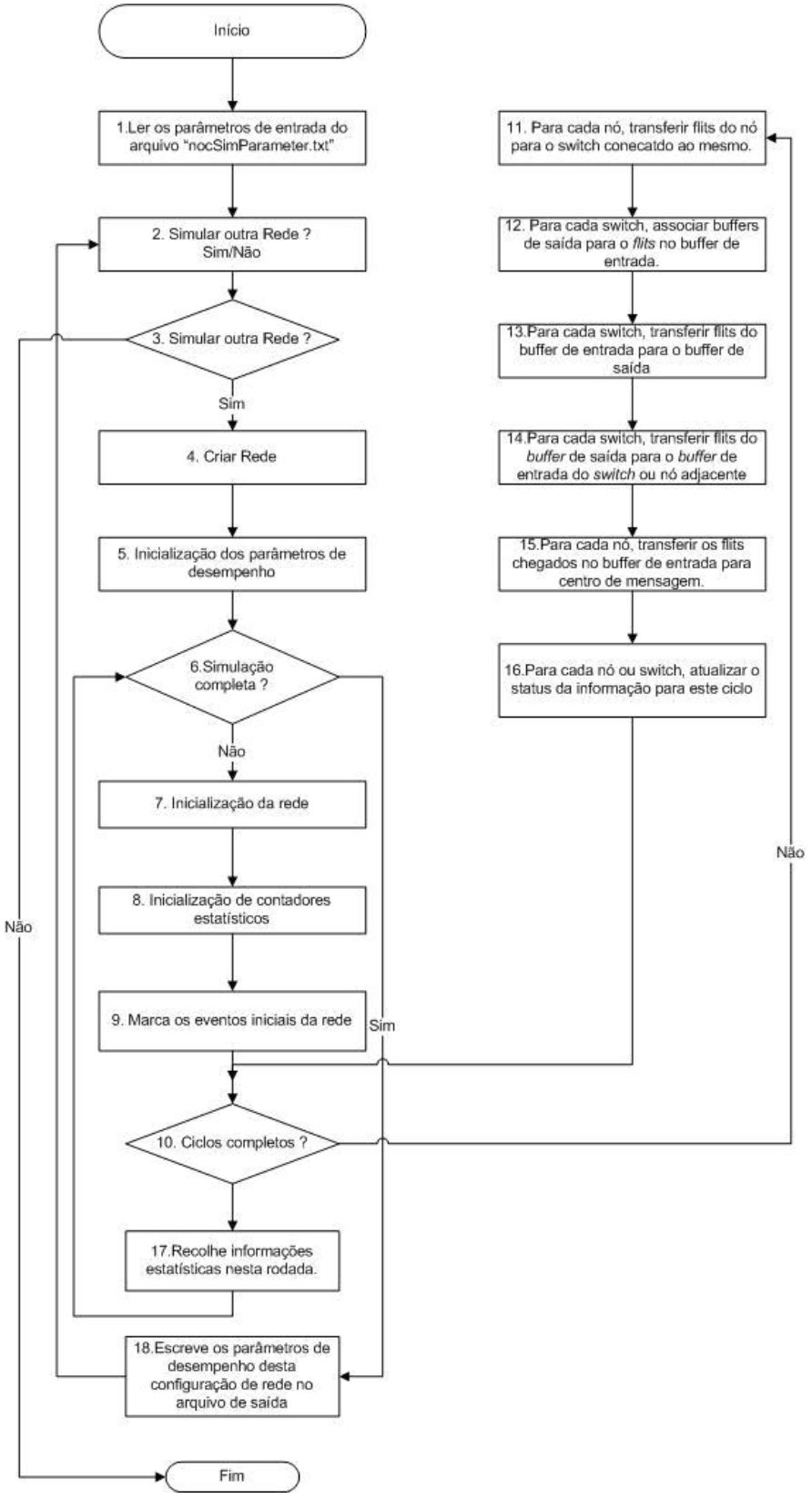

Figura A.1: Fluxograma do Simulador gpNoCsim

Apêndice B

Artigos Submetidos

Um artigo sobre este documento será submetido até o dia 01 de fevereiro de 2008 para a conferência SAMOS 2008(http://samos.et.tudelft.nl/samos_viii/)

B.1 Abstract

The need to meet the existing demands in the microeletronic market has prompted the designers to compact a big number of IP blocks in a small silicon area. From the pratical point of view, the distribuition of these IP blocks becomes a problem due to physical issues like high impedance caused by the number of wires that interconncet them, the power consupmtion to keep all IP blocks comunicating and an optimized occupation of the whole space used by the chip. In order to help the SoC designers, the concepts used in networking had been the main source to point out a possible solution for these situations. This paper shows the results of a benchmarck using gpNoCsim[1], which can help NoC designers to find the bottlenecks when working with NoCs.

Keywords - networking, network-on-chip, troughput, latency, gpNoCsim.

B.2 Introduction

Technological and financial questions impact research in integrated circuits. There is a demand for reusable blocks in an attempt to reduce time-to-market. The use of IP blocks thus becomes a natural approach. Technological issues include routing, power consumption and area. The integration of circuits of increasing complexity brings many concepts from the field of networking to that of Integrated Circuits. In NoCs, the cores will be interconnected through channels, with switches controlling the information flow. The point-to-point connection is obviously expensive as an interconnection choice, in both area and performance overhead. The bus architecture imply the use of long paths for all signals.

In the technological questions, there are a lot details that must be considered like internal architecture, routing of the information, a operating system to manage the whole chip, performance, power consumption, the area occupied for all IP blocks. The reused componentes are called IP blocks and may be developed by a company that is responsible for a specific project or by anyone that is researching in its subject, which usually receives the royalties to give a permission to use them.
The cores interconnect themselves through channels using switches to guide information. From the communication point of view, it is costly to use a point-to-point conection with all internal components, cause a mesh would result in a bigger impedance, consequently in a larger delay to move data across the wires, limiting the performance of the whole chip. The chips with a bus architecture work by difusion and each sinal must go to all points of the main bus. This task requires a great amount of power.
An alternative emerges, with all IP blocks interconnected as in a computer network [4,5], making use of well-established concepts from this field of research. A particular solution being studied in the literature is switched networking [3,6]. In [18], the description of the results of a dynamic routing and its applicabilities in NoC is made. We can enumerate some advantages in this approach: modularity, energy saving, pipeling capability [7], lower manufacture cost.
In this paper, using the gpNoCsim[1], we will simulate situations which help pointing out one architecture that fits the existing market needs, with maximum throughput and low latency.

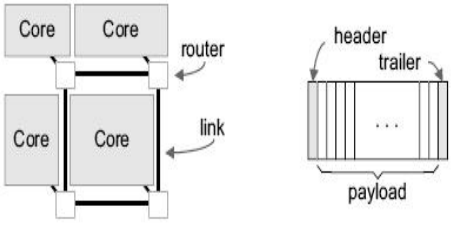

Figura B.1: NoC concepts : topology and message format [8]

B.3 Background

The challenges for Network-On-Chip architectures are usually the same ones found in LAN, MAN and WAN networking.
The packets, with fixed length or not, are partioned and transmited through the network, broken in flow control units called flits [9]. It is desired to transmit these flits with maximum throuput, lesser latency, lesser power consumption, lesser hop count, maximun usage of communication channels and of input/output buffers (processing elements and switches). As we have seen, there is a long list of variables to be considered and correlated.
To verify large chips, the use of simulation tools is particularly useful, in order to find possible errors in the begining of the project, which can improve design cost and time-to-market.
In this paper, gpNoCSim[1] will be used to collect the results and to compare with

the results presented in [24], which can help us to find an acceptable architecture that meets the basic requirements to build a chip.
Several papers propose and simulate performance for different architectures, in [10] the butterfly fat tree is improved and the extended-butterfly fat tree is introduced. The troughput and latency are analysed with increasing buffer size. [11] presents metrics that include throughput, latency, area requirements to compare the performace between Mesh and Butterfly Fat Tree topologies. In [12], the topologies Mesh and Butterfly Fat Tree are analysed again, varying the number of virtuals channels and load injection. [13] shows the benchmark with the SoCBUS Network, to find the bootleneck in network on chips, analysing other metrics like route blockings, offered/accepted load, locking and data for routers ports.

B.4 Analytical Model for Latency

The latency of a message injected from a node is determined by the waiting time and service time in the injection channel, which depends on the service time and waiting time of successive channels on the path to be followed. When a message is generated in the element processing, which is connected to the switch j, it waits for a time defined by $W_{inj,j}$. Once the header is accepted in the network, the message will wait for a while in the injection channel. This period, called service time, is defined by $x_{inj,i}$. At the end of the service time, when the final part of the message leaves the injection channel, the message will have D-1 steps until the entire message is received at the destination, where D is the number of channels on the path. This happens when the final part of the message leaves the injection channel. The header must have arrived at the destination when there are no blockages. The latency L_j of the message injected in the node j is given by:

$$L_j = W_{inj,j} + x_{inj,j} + D - 1 \tag{B.1}$$

The average latency $\overline{L}$ to calculate the average of all nodes for the entire network is:

$$\overline{L} = \frac{1}{N} \times \sum_{i=0}^{j} \overline{L}_j(2) \tag{B.2}$$

$$\overline{L} = \frac{1}{N} \times \sum_{i=0}^{j} (\overline{W}_{inj,j} + \overline{x}_{inj,j}) + \overline{D} - 1 \tag{B.3}$$

where N is the number of nodes in the network and $\overline{D}$ is the average distance of the message.

The waiting time in the equation above can be approximated by the traditional M/G/1 model of queues [4], since the service time $x_{inj,j}$ is known. In routing *wormhole*, a message is sent through many channels in a given time.

Messages originated from channel *i* with arrival rate λ_i^{in} can be routed to the output channels defined by j = 0, 1, 2, 3 ,.... The service time for the input channels depends on the service time and waiting time of all possible output channels. Given the probability of a message $R_{i|j}$ routed from the input channel *i* to the output channel *j*, the service time to the input channel *i* can be expressed by the following formula:

$$x_i^{in} = \sum_{i=0}^{j} (x_j + w_j) \times R_{i|j} \tag{B.4}$$

where x_j is the service time to the output channel *j* and w_j is the waiting time for the output channel output *j*. The equation above confirms the dependence of the service time in a particular channel on the service time and the waiting time of the channels thereafter. The average waiting time w_j can be approximated using the M/G/1 model, and the waiting time. In this case, it can be approximated by the suggestion described in [24], as shown in the following expression:

$$\overline{W}_j = \frac{\lambda_j \overline{x}_j^2}{2(1 - \lambda_j \overline{x}_j)} \left[1 + \frac{(\overline{x}_j - s/f)^2}{\overline{x}_j^2}] \right] \tag{B.5}$$

where λ_j is the rate of messages in the output channel *j*, *s* is the message size (configured by the application or simulation) and f is the size on *flits*.

But the M/G/1 model assumes an independent arrival of messages in the switch, which can block each other. Once an incoming link is busy by one flit, it will not have any incoming flits in that channel until the first flit is served, or forwarded. So, to use the result of waiting time from the M/G/1 model, it is necessary to multiply by the blockade probability $P_{i|j}$:

$$w_j = P_{i|j} W_j \tag{B.6}$$

where $P_{i|j}$ may be expressed as in [24] :

$$P_{i|j} = 1 - \frac{\lambda_i^{in}}{\lambda_j} R_{i|j} \tag{B.7}$$

We can have an ideia of $P_{i|j}$ as the probability that a message is from other channel than *i*, by the M/G/1 model. Combining the expressions 4, 6, 7, we obtain the service time for incident messages in the channel of entrance *i*:

$$x_i^{in} = \sum i = 0^j \left[x_j + (1 - \frac{\lambda_i^{in}}{\lambda_j} R_{i|j}) W_j \right] R_{i|j} \tag{B.8}$$

We observe that $\overline{W}_j$ is the mean waiting time obtained in Equation 5, λ_i^{in} is the loadrate in the incoming channel *i* and λ_j is the loadrate in the outgoing channel *j*.

We proceed now with specific calculations for different topologies.

B.4.0.1 Mesh and Torus

The service times are computed in the order of ejection of the channels. In the ejection channel, once the first flit of the message is received, the rest of the message will be received a flit per time without any blockade. Therefore, the service time in the ejection channels is same to the size, in flits, of the message as shown in the equation below

$$x_{<j_0,j_1,ej>} = s/f \tag{B.9}$$

We will proceed with the calculations of service time in the dimensions "x"and "y". In the "x"axis, the service time is determinated as it is follows :

- The message leaves the channel $< j_0 >$ with a probability $R_{<j_0>|e_j} = \frac{1}{j_0}$;
- Or it continues to the next channel in the same direction, with probability $R_{<j_0>|j_0-1} = \frac{j_0-1}{j_0}$.

The service time x_{j_0} is shown in the equation below :

$$x_{<j_0,j_1>} = (s/f)\frac{1}{j_0} + \left[x_{j_0-1} + \frac{1}{k-j_0+1}W_{<j_0-1>}\right]\frac{j_0-1}{j_0} \tag{B.10}$$

In the "y"axis, the service time is determinated as follows :

- When leaving the channel $< j_0, j_1 >$, the message can come out for the next channel of available ejection, with probability $R_{<j_0,j_1>|e_j} = \frac{1}{j_1 k}$;
- It continues for the next channel in the same direction, with probability $R_{<j_0,j_1>|<j_0,j_1-1>} = \frac{j_1-1}{j_1}$;
- It proceeds for the more channel to east, with probability $R_{<j_0,j_1>|<j_0>} = \frac{j_0}{j_1 k}$;
- Or it proceeds for the more channel to west, with probability $R_{<j_0,j_1>|<j_0,O>} = \frac{k-1j_0}{j_1 k}$.

The service time $x_{<j_0,j_1>}$ is shown in the equation below :

$$\begin{aligned} x_{<j_0,j_1>} = (s/f)\frac{1}{j_1 k} + \\ \left[x_{<j_0>} + \frac{k(k-j_0)-(k-j_1)}{k(k-j_0)}W_{<j_0>}\right]\frac{j_0}{j_1 k} + \\ \left[x_{<k-1-j_0>} + \frac{kj_0+j_1}{k(j_0+1)}W_{<k-1-j_0>}\right]\frac{k-1-j_0}{j_1 k} + \\ \left[x_{<j_0,j_1-1>} + \frac{1)}{k-j_1+1)}W_{<j_0,j_1-1>}\right]\frac{j_1-1}{j_1} \end{aligned} \tag{B.11}$$

In the injection channel, the service time is determinated as follows :

- When leaving the injection channel in the node $< j_0, j_1 >$, the message can go to the north, with probability $R_{inj}| < j_0, j_1, N > = \frac{(k-1-j_1)k}{k^2-1}$
- Or proceed for the south, with probability $R_{inj}| < j_0, j_1 > = \frac{j_1 k}{k^2-1}$
- Or proceed to the east, with probability $R_{inj}| < j_0, L > = \frac{(k-1-j_0)k}{k^2-1}$
- Or proceed to the west, with probability $R_{inj}| < j_0 > = \frac{j_0 k}{k^2-1}$

The equation of the service time to the ejection channel, is shown by the equation below:

$$\begin{aligned} x_{<inj,j_0,j_1>} = & \left[x_{<j_0>} + \frac{k(k-j_0)-1)}{k(k-j_0)} W_{<j_0>}\right] \frac{j_0}{k^2-1} + \\ & \left[x_{<k-1-j_0>} + \frac{k(1+j_0)-1}{k(1+j_0)} W_{<k-1-j_0>}\right] \frac{k-1-j_0}{k^2-1} + \\ & \left[x_{<j_0,j_1>} + \frac{k-j_1-1}{k-j_1} W_{<j_0,j_1>}\right] \frac{j_1 k}{k^2-1} + \\ & \left[x_{<j_0,k-1-j_1>} + \frac{j_1}{1+j_1} W_{<j_0,k-1-j_1>}\right] \frac{(k-1-j_1)k}{k^2-1} \end{aligned} \tag{B.12}$$

The waiting time $W_{<inj,j_0,j_1>}$ are determinated using $x_{<inj,j_0,j_1>}$ and equation (5) of this document. The average latency for the whole network is shown in the equation below:

$$\overline{L} = \sum_{j_0=0}^{k-1} \sum_{j_1=0}^{k-1} (W_{<inj,j_0,j_1>} + x_{<inj,j_0,j_1>}) + \overline{D} - 1 \tag{B.13}$$

B.4.0.2 Butterfly-Fat Tree

Each node is represented as a pair of indices (n, e), where n is the node level in the network and e is the node address at that level. At the lowest level there are N nodes, with address from 0 to $N-1$ and each switch has 6 gates: $father_0$, $father_1$, son_0, son_1, son_2, son_3, son_4. At level "x", where $1 \leq x \leq \log_4 N$, there are $N/2^{x+1}$ switches. Switch connections are established by its addresses as follows: $father_0$ of $S(n,e)$ is connected to son_1 of $S(n+1, \left\lfloor \frac{e}{2^n+1} \right\rfloor 2^l + a \bmod 2^l)$ and $father_1$ of $S(n,e)$ is connected to son_1 of $S(n+1, \left\lfloor \frac{e}{2^n+1} \right\rfloor 2^n + (e + 2^{n-1}) \bmod 2^n)$, where $i = \left\lfloor \frac{e \bmod 2^{n+1}}{2^{n-1}} \right\rfloor$. In this topology, the shortest path between any two nodes is not unique. The message can be transmitted through any of those two upper links, it applies if the destiny is not at the same level, or in the lower level, which the message is addressed. When a flit needs to go to an upper level, a link is randomly selected. If the selected link is blocked, then it tries the other one and when both are blocked, the flit waits until it can be processed by one of the upper link's switch.

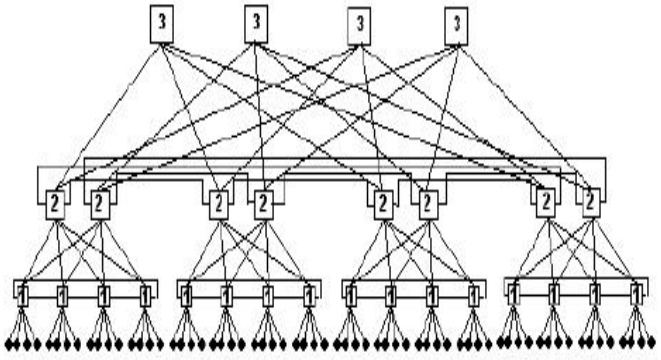

Figura B.2: Butterfly Fat Tree topology with 64 IP blocks[25]

To calculate the loadrate in this topology, links at the same level and same direction, which can be up or down, are symmetric and in this case it is not necessary to differentiate them. We can label links and their message reception rates by the pair of indices $\langle i, j \rangle$, where i is the start level and j is the end level of the link in the network, $0 \leq i,\ j \leq n$ where $n = \log_4^N$. For instance, let each node from some network put messages in a rate λ_0. Hence, $\lambda_{0,1} = \lambda_{1,0} = \lambda_0$ for links among nodes at level 0 and switches with level 1. Suppose a message goes from level n to level n+1 ($1 \leq n < x$). For a number $N = 4^x$ of nodes in the system, the message can have $4^x - 1$ destinies of which $4^n - 1$ are reached without changing to level n. Thus, the probability of a message at level n go up one level is:

$$P_n^{\uparrow} = \frac{4^n - 4^x}{4^n - 1} \tag{B.14}$$

and the probability that a message goes down one level is:

$$P_n^{\downarrow} = 1 - P_n^{\uparrow} \tag{B.15}$$

The total loadrate leaving the level n for the level $n + 1$ is $P^{\uparrow} 4^n \lambda_0$. A total of $\frac{4^x}{2^l}$ *l*inks exist links between level n and level $n + 1$. The loadrate for each channel going from level n to level $n+1$ is $\lambda_{n,n+1} = \frac{P^{\uparrow} 4^n \lambda_0}{\frac{4^x}{2^n}}$. Due to symmetry the loadrate from level n to level $n + 1$ is similar to the one from level $n + 1$ to level n. Summarizing:

$$\lambda_{l,l+1} = \lambda_0 \frac{4^n - 4^x}{4^n - 1} 2^n \tag{B.16}$$

$$\lambda_{l+1,l} = \lambda_{l,l+1} \tag{B.17}$$

In agreement with the equations shown previously, we have initially the service time of the links of level 1 for the processing node (level 0), equal to the size of the message in flits, as shown in the Equation 12:

$$x_{1,0} = s/f \tag{B.18}$$

The waiting time is calculated using the Equation B.5 of this chapter:

$$\overline{W}_{1,0} = \overline{W}_{M/G/1}(\lambda_{1,0}, x_{1,0}) \tag{B.19}$$

For any message traffic among levels n and $n+1$, regardless of direction, there are 4 possible of exits of the channel exist and each one with the same probability. The average service time is determinated for the Equation 11:

$$\overline{x}_{n+1,n} = \overline{x}_{n,n-1} + \left(1 - \frac{1}{4}\frac{\lambda_{n+1,n}}{\lambda_{n,n-1}}\right)\overline{W}_{n,n-1} \tag{B.20}$$

The average waiting time $\overline{W}_{n+1,n}$ is calculated using $\overline{x}_{n+1,n}$:

$$\overline{W}_{n+1,n} = \overline{W}_{M/G/1}(\lambda_{n+1,n}, \overline{x}_{n+1,n}) \tag{B.21}$$

Let us consider the ascending channels now, beginning with channel $\langle n-1, n\rangle$. Now there are 3 (three) exits , each one with probability 1/3. Then we have :

$$\overline{x}_{n-1,n} = \overline{x}_{n,n-1} + \left(1 - \frac{1}{3}\frac{\lambda_{n-1,n}}{\lambda_{n,n-1}}\right)\overline{W}_{n,n-1} \tag{B.22}$$

as result :

$$\overline{x}_{n-1,n} = \overline{x}_{n,n-1} + \frac{2}{3}\overline{W}_{n,n-1} \tag{B.23}$$

The average waiting time $\overline{W}_{n-1,n}$ is determined using the model of the Equation 5 of this document, through the method of Hokstad´s aproximation[17], resulting:

$$\overline{W}_{n-1,n} = \overline{W}_{M/G/2}(\lambda_{n-1,n}, \overline{x}_{n-1,n}) \tag{B.24}$$

The exception is in the case where $n = 1$. The channel $\langle 0, 1\rangle$ is from processing node to the first level without redundancy. For this situation the following equation is shown:

$$\overline{W}_{0,1} = \overline{W}_{M/G/1}\left(\lambda_{0,1}, \overline{x}_{0,1}\right) \tag{B.25}$$

For any channel that proceeds from level $n-1$ to level n, the message can go up a level with probability $P_n^{\uparrow}$ or go down a level with probability $P_n^{\downarrow}$, mentioned earlier. Then we have the average service time as :

$$\overline{x}_{n-1,n} = [\overline{x}_{n,n+1}] + \left[\left(1 - \frac{\lambda_{n-1,n}}{\lambda_{n,n+1}} P_n^{\uparrow}\right) \overline{W}_{n,n+1}\right] P_n^{\uparrow} + \left[\overline{x}_{n,n-1} + \left(1 - \frac{P_n^{\uparrow}}{3}\right) \overline{W}_{n,n-1}\right] P_n^{\downarrow}$$

The latency is calculated using the Equation 3, we will calculate the average latency. For a *Butterfly Fat-Tree* topology, $x_{inj,j} = x_{0,1}$ and $W_{inj,j} = W_{0,1}$. The latency is determined below :

$$\overline{L} = \overline{W}_{0,1} + \overline{x}_{0,1} + \left(\overline{D} - 1\right) \tag{B.26}$$

B.4.1 gpNoCsim

The gpNoCsim[1] (General Purpose Network-On-Chip Simulator) was developed by the Computer Science and Engineering Departament from Blangadesh University. It is open source code written in Java and implementes 5 (five) types of architecture, Mesh, Torus, Butterfly Fat Tree, Extend Butterfly Fat Tree and Octal.
The paremeters inserted into the simulator are input through the file "nocSimParameter.txt". It is possible to choose the topology, load injection, virtuals channels, number of IP blocks and others parameters . The output data from the simulation are provided trough the file "nocSimOutput.txt".

In the next lines, it be shown how the gpNoCsim calculates the latency.

B.4.1.1 Latency

The latency in gpNoCsim[1] is an average of all delays occurred during the delivery of the packages. The unit is cycles/package. Latency is defined as the time (in clock cycles) spent between the occurrence of the head injection of the message in the network at the origin node and the reception of the last part of the message at the destination node. Flits will cover alternative ways to reach the destination node, according to the routing algorithm. The messages will have different delays, depending on the pair origin/destination and on the routing algorithm. In this work, the average of the packages latencies will be evaluated.

B.5 Simulation Results and Conclusions

The analytical model presented in [24] predicts the performance of the mesh and the torus topologies for varying number of flits and IP blocks, and performance of the BFT topology for varying message size. Simulatios were done in the same conditions.

As we can see in Figure 3, the torus topology tends to reach high values for latency with low loadrates. The diferences between the latency values depends on the number of IP blocks used in the simulations. The same behaviour is observed in the mesh

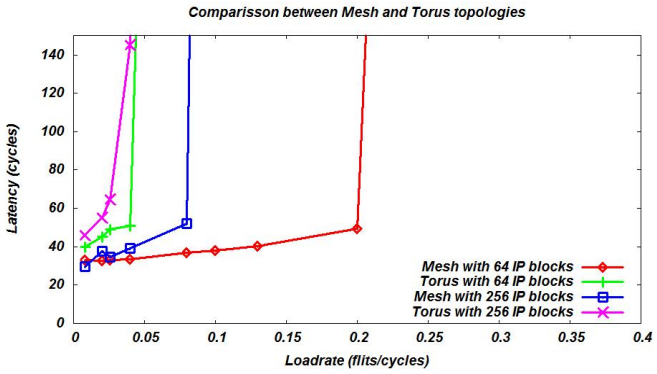

Figura B.3: The results using gpnocsim varying the number os IP blocks with mesh and torus topologies

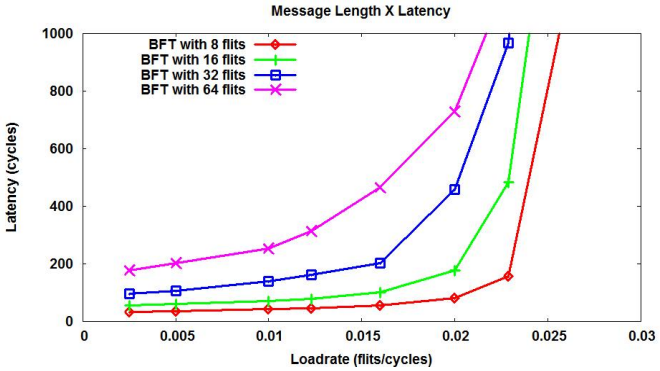

Figura B.4: The results using gpnocsim varying the message length with BFT topology

topology, but the latency values reach the same values as in the torus topology with loadrate that are 4 times larger. With the results shown in Figure 3, it is possible to say that the mesh topology has a better performance than torus at any loadrate. It is also easier to lay out, saving area and wire length.

With BFT topology, the simulations were done using message length with 8, 16, 32, 64 flits with 256 IP blocks. The Figure 4 shows the results and the comparisson between diferents message lengths.

The behavior of average latency with loadrate follows the ones reported in [24]. The mesh topology has shown favorable results, but this metric is to be taken as an initial step in topology and architecture evaluation, a task that will be continued as actual NoCs are designed and tested in our group.

B.6 bibliography

1. www.buet.ac.bd/cse/research/group/noc

2. www.itrs.net

3 W. J. Dally, B. Towles, Route packets, not wires: onchip interconnection networksProc. of DAC2001, Las Vegas, Nevada, USA, pp. 684-689, June, 2001.

4 Benini, L., and De Micheli, G.:Network on chips : a new SoC paradigm, Computer, 2002, 35, (1), pp.70-78

5 G. de Micheli and L. Benini. Networks on chip: A new paradigm for systems on chip design.In DATE 02: Proceedings of the conference on Design, automation and test in Europe, page 418, Washington, DC, USA, 2002. IEEE Computer Society.

6 A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. O¨ berg, M. Millberg and D. Lindqvist,Network on chip: an architecture for billion transistor era, Proceedings of NorChip 2000, November 6-7, Turku, Finland.

7 A Generic Architecture for On-Chip Packet-Switched Interconnections

8 SoCIN: A Parametric and Scalable Network-on-Chip

9 L. Benini and D. Bertozzi, Network-on-chip architectures and design methods, in Proceedings of IEEE Computer Digital Technology, March 2005.

10 Extented-Butterfly Fat Tree Interconnection (EFTI) Architecture for Network on Chip

11 Evaluation of MP-SoC Interconnect Architectures : a Case Study

12 GPNOCSIM - A General Purpose Simulator For Network-On-Chip

13 Network on chip simulations for benchmarking

14 M. Forsell and S. Kumar, Virtual Distributed Shared Memory for Network on Chip, In the Proceedings of the 19th IEEE NORCHIP Conference, November 12-13, 2001, Kista, Sweden, 192-197.

15 Y. Sun, Simulation and Performance Evaluation for Network on Chip, Master thesis, LECS, KTH, Stockholm December 2001.

16 S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. O¨ berg, K. Tiensyrja¨, and A. Hemani. A Network on Chip Architecture and Design Methodology. Accepted for publication at IEEE Computer Society Annual Symposium on VLSI, Pittsburgh, Pennsylvania, USA, April 25-26, 2002.

17 Hokstad P., Approximations for the M/G/M queue., 1978.

18 M. Ali, M. Welzl, S. Hellebrand. A Dynamic Routing Mechanism for Network-on-ChipNORCHIP Conference, 2005. 23rd Volume , Issue, 21- 22 Nov. 2005 Page(s): 70 - 73

19 Dan Moritz, Virage Logic Corp., Saverio Fazzari, Cadence Design Systems. How to Create Efficient IP Standards and Why You Should Care

20 W. J. Dally. Virtual-Channel Flow ControlIEEE Transactions on Parallel and Distributed Systems, pp. 194-205, March 1992.

21 I. Nousias, T. Arslan Wormhole Routing with Virtual Channels using Adaptive Rate Control for Network-on-ChipAdaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference.

22 N. Kavaldjiev, G.J.M. Smit, P.G. Jansen A Virtual Channel Router for On-Chip Networks SOC Conference, 2004. Proceedings. IEEE International

23 I. Saastamoinen, M. Alho, J. Nurmi Buffer implementation for Proteo network-on-chipCircuits and Systems, 2003. ISCAS apos;03. Proceedings of the 2003 International Symposium on Volume 2, Issue , 25-28 May 2003 Page(s): II-113 - II-116 vol.2

24 J. T. Draper and J. Ghosh A Comprehensive Analytical Model for Wormhole Routing in Multicomputer SystemsJournal of Parallel and Distributed Computing 1994 on Volume 23, Pages : 202-214

25 H. Hossain, M. Akbar,M.Islam, Extended-butterfly fat tree interconnection (EFTI) architecture for network on chip,Communications, Computers and signal Processing, 2005. PACRIM. 2005 IEEE Pacific Rim Conference on , vol., no., pp. 613-616, 24-26 Aug. 2005

26 Alzeidi,N. and Khonsari,A. and Ould-Khaoua,M. and Mackenzie, L.M.,A New Queueing Model for the Analysis of Virtual Channels Occupancy in Wormhole-Switched Networks,Dept of Computing Science, University of Glasgow, 2005.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo