

EXPLORANDO REDES ÓTICAS
COMO MEMÓRIA CACHE

Enrique Vinicio Carrera Erazo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.


Aprovada por:



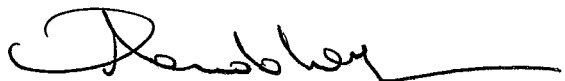
Prof. Ricardo Bianchini, Ph.D.



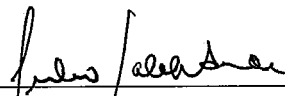
Prof. Cláudio Luis de Amorim, Ph.D.



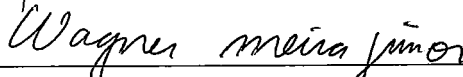
Prof. Inês de Castro Dutra, Ph.D.



Prof. Orlando Gomes Loques Filho, Ph.D.



Prof. Júlio Salek Aude, Ph.D.



Prof. Wagner Meira Júnior, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

NOVEMBRO DE 1999

CARRERA ERAZO, ENRIQUE VINICIO

Explorando Redes Óticas como Memória
Cache [Rio de Janeiro] 1999

XIV, 163 p. 29,7 cm (COPPE/UFRJ, D.Sc.,
Engenharia de Sistemas e Computação, 1999)

Tese – Universidade Federal do Rio de
Janeiro, COPPE

1 - Arquitetura de Computadores

2 - Sistemas Paralelos e Distribuídos

I. COPPE/UFRJ II. Título (série)

*Aos meus pais,
Enrique e Nelva*

Agradecimentos

Primeiramente, gostaria de agradecer ao meu orientador, Ricardo Bianchini, pela paciência comigo ao longo de todo este tempo, pelo estímulo constante para que eu superasse meus próprios limites, pelas longas jornadas de trabalho ao meu lado, pela ajuda desinteressada fora do campo acadêmico e por ser, mais do que um simples orientador, um amigo e companheiro.

Gostaria também de agradecer a todo o grupo de colegas e amigos das reuniões de quarta-feira (Carla, Clicia, Cristiana, Eduardo, Lauro, Raquel, Rodrigo S., Rodrigo M. e Silvio) pelas discussões relacionadas ao meu trabalho e pela amizade ao longo de todo o meu doutorado. Agradeço especialmente a Cristiana, Eduardo e Lauro pela ajuda na revisão desta Tese.

Agradeço também à Universidade Federal do Rio de Janeiro e, em especial ao Programa de Sistemas e Computação da Coordenação dos Programas de Pós-graduação em Engenharia, por ter contribuído para a minha formação. Não posso deixar de agradecer também ao Conselho Nacional de Pesquisa e Desenvolvimento pelo apoio financeiro recebido durante todos os meus anos de estudo.

Um agradecimento especial à minha esposa e ao meu filho que não tiveram a minha companhia e carinho, durante estes dois últimos anos, para que eu pudesse fazer de um dos meus sonhos realidade.

Finalmente, agradeço ao meu pai, mãe e irmãs que sempre me apoiaram em todas as minhas decisões, me dando forças para continuar em frente.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc)

EXPLORANDO REDES ÓTICAS COMO MEMÓRIA CACHE

Enrique Vinicio Carrera Erazo

Novembro/1999

Orientador: Ricardo Bianchini

Programa: Engenharia de Sistemas e Computação

Os recentes avanços na tecnologia de componentes óticos têm permitido a construção de redes óticas extremamente rápidas. Como consequência disso, essas redes vêm sendo consideradas por vários pesquisadores como uma opção para o desenvolvimento de sistemas paralelos e distribuídos. No entanto, essas pesquisas não exploram totalmente o potencial da tecnologia ótica. A capacidade que as fibras óticas têm de atuar como memórias de linha de retardo, por exemplo, não é explorada. Esta Tese visa completar essa lacuna. Mais especificamente, nossa proposta é utilizar redes de interconexão ótica como memórias cache em diferentes níveis do sistema de memória de sistemas paralelos e distribuídos. Com este objetivo, projetamos e avaliamos quatro tipos de redes óticas, três das quais são capazes de armazenar informação na própria rede. A avaliação de todas estas redes é plenamente satisfatória. Os resultados das nossas simulações confirmam que a utilização de cacheamento ótico tem grande potencial de melhorar o desempenho de sistemas paralelos e distribuídos, especialmente se suportado por redes óticas eficientes.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

EXPLOITING OPTICAL NETWORKS AS CACHE MEMORY

Enrique Vinicio Carrera Erazo

November/1999

Advisor: Ricardo Bianchini

Department: Computing and Systems Engineering

The recent improvements in optical technology have enabled the construction of very fast interconnection networks. As a result of these improvements, optical networks are being considered by several researchers as a serious option in the development of parallel and distributed systems. However, the designs proposed by these researchers do not fully exploit the potential of optical technology. For instance, the ability of optical fibers to act as delay line memories has not been exploited thus far. This Thesis fills this gap. More specifically, our proposal is to use optical interconnection networks as cache memories at different levels of the memory system in parallel and distributed systems. In this direction, we design and evaluate four optical networks, three of them are able to store data in the network itself. The evaluation of these networks is fully satisfactory. The results of our simulations confirm that optical caching has an enormous potential to improve the performance of parallel and distributed systems, specially if supported by efficient optical networks.

Índice

1	Introdução	1
1.1	Contribuições da Pesquisa	5
1.2	Organização da Tese	5
2	Redes Óticas	6
2.1	Comunicação Ótica	6
2.1.1	Fibras Óticas	6
2.1.2	Transmissores e Receptores	8
2.1.3	Técnicas de Acesso ao Meio	9
2.2	Redes WDM	10
2.2.1	Classificação das Redes WDM	11
2.2.2	Redes WDM Single-Hop	13
2.2.3	Redes WDM Multiple-Hop	16
2.2.4	Processamento Paralelo e Distribuído com Redes WDM	17
2.3	Redes OTDM	18
2.3.1	A Tecnologia OTDM	19
2.3.2	Processamento Paralelo e Distribuído com Redes OTDM	20
2.4	Outras Redes Óticas	21
2.4.1	Redes FDDI	21
2.4.2	Redes ATM	22
2.4.3	Redes Gigabit-Ethernet	24
2.5	Conclusões	24
3	Sistemas de Memória	26
3.1	Memórias Cache Tradicionais	26

3.1.1	Associatividade das Caches	27
3.1.2	Identificação de Blocos	28
3.1.3	Substituição de Blocos	28
3.1.4	Estratégias de Escrita	29
3.2	Memórias de Linha de Retardo	29
3.2.1	Memórias Síncronas	31
3.2.2	Memórias Assíncronas	35
3.2.3	Problemas Principais	36
3.3	Utilização em Computação Paralela e Distribuída	39
4	OPTNET	43
4.1	Fundamentos	43
4.1.1	DMON	44
4.1.2	LambdaNet	46
4.2	Arquitetura de OPTNET	47
4.2.1	Arquitetura Básica	47
4.2.2	Protocolo de Coerência Básico	48
4.2.3	Suportando Múltiplos Pedidos de Leitura Pendentes	50
4.3	Metodologia	52
4.3.1	Simulação	52
4.3.2	Aplicações	55
4.4	Resultados	55
4.4.1	Desempenho Geral	56
4.4.2	Desempenho das Leituras	58
4.4.3	Desempenho das Escritas	60
4.4.4	Impacto dos Parâmetros Arquiteturais	63
4.5	Trabalhos Relacionados	66
4.6	Conclusões	67
5	NetCache	69
5.1	Arquitetura de NetCache	70
5.1.1	Arquitetura Básica	70

5.1.2	Protocolo de Coerência	72
5.1.3	Implementação Alternativa	75
5.2	Metodologia	76
5.2.1	Simulação	76
5.2.2	Aplicações	79
5.3	Resultados	80
5.3.1	Desempenho Geral	81
5.3.2	Eficiência da Cache Compartilhada	83
5.3.3	Avaliação de Diferentes Organizações e Políticas de Substituição	85
5.3.4	Impacto dos Parâmetros Arquiteturais	91
5.3.5	Comparação com Outros Sistemas	94
5.4	Trabalhos Relacionados	95
5.5	Conclusões	96
6	OWCache	97
6.1	Uma Cache Ótica para Escritas	99
6.1.1	Arquitetura do Multiprocessador e Gerência da Memória Virtual	99
6.1.2	OWCache = OPTNET + Anel Ótico	101
6.2	Metodologia	105
6.2.1	Simulação	105
6.2.2	Aplicações	106
6.3	Resultados Experimentais	107
6.3.1	Benefícios de Desempenho	107
6.3.2	Impacto dos Parâmetros Arquiteturais	113
6.4	Estendendo um Multiprocessador Tradicional	120
6.4.1	Arquitetura Básica	120
6.4.2	Resultados Experimentais	121
6.5	Trabalhos Relacionados	124
6.6	Conclusões	125
7	Implementação de uma Memória de Rede na Internet	127
7.1	Fundamentos	129

7.1.1	A Internet	129
7.1.2	Redes Ativas	130
7.1.3	Exemplos de Aplicações Distribuídas	131
7.2	Memória de Rede	132
7.3	Metodologia	137
7.3.1	A Aplicação	138
7.3.2	Configurações Básicas	139
7.4	Resultados	141
7.4.1	Validação do Simulador	141
7.4.2	Resultados Base	144
7.4.3	Variação de Parâmetros	146
7.5	Trabalhos Relacionados	149
7.6	Conclusões	150

8 Conclusões e Trabalhos Futuros 151

Lista de Figuras

2.1	Propagação da Luz em uma Fibra Ótica	7
2.2	A Técnica de Multiplexação WDM	11
2.3	Uma Rede WDM <i>Single-Hop</i>	13
2.4	Topologia da Rede TeraNet	16
2.5	Sistema de Transmissão OTDM	19
2.6	Sistema OTDM para Multiprocessadores e Multicomputadores	20
2.7	Uma Rede FDDI	22
2.8	Uma Rede ATM	22
2.9	Uma Rede Gigabit-Ethernet	23
3.1	Diagrama de Blocos de uma Memória de Linha de Retardo	30
3.2	Memória de Linha de Retardo Síncrona	31
3.3	<i>Pulse-Stretching</i> para a Memória de Linha de Retardo Síncrona	32
3.4	Memória Síncrona com Contador Ótico	33
3.5	Interface Optoeletrônica da Memória Síncrona	34
3.6	Memória de Linha de Retardo Assíncrona	35
3.7	Evolução da Velocidade de Transmissão nos Sistemas Óticos	40
3.8	Latência Média de Acesso em Função da Capacidade de Armazenamento	40
4.1	A Interface de Rede da Arquitetura DMON	45
4.2	A Interface de Rede da Arquitetura LambdaNet	46
4.3	Detalhe da Arquitetura dos Nós	47
4.4	A Interface de Rede da Arquitetura OPTNET	48
4.5	Ganho de Desempenho em um Multiprocessador de 16 Nós com OPTNET	56
4.6	Tempos de Execução de OPTNET, LambdaNet, DMON-U e DMON-I	57
4.7	Latência Média das Leituras	58

4.8	Latência Média de uma Falha de Leitura para OPTNET, LambdaNet, DMON-U e DMON-I	58
4.9	Tempos de Execução para 1, 2 e 4 Canais de Atualização em 16 Nós . . .	62
4.10	Tempos de Execução para 1, 2 e 4 Canais de Atualização em 32 Nós . . .	62
4.11	Tempos de Execução como Função do Tamanho da Cache Secundária . .	64
4.12	Tempos de Execução como Função da Taxa de Transmissão	65
4.13	Tempos de Execução como Função da Latência de Leitura na Memóri . .	65
5.1	Detalhe da Arquitetura de NetCache	70
5.2	Ganho de Desempenho em um Multiprocessador de 16 Nós com NetCache	80
5.3	Tempos de Execução de NetCache, LambdaNet, DMON-U e DMON-I . .	81
5.4	Porcentagem da Latência de Leitura no Tempo de Execução, Taxa de Acerto na Cache Compartilhada, Redução da Latência de Leitura em uma Falha e Redução da Latência de Leitura Total	84
5.5	Taxas de Acerto para uma Cache Compartilhada de 16, 32 e 64 KBytes .	85
5.6	Latências de Leitura em uma Cache Compartilhada de 16, 32 e 64 KBytes	86
5.7	Tempos de Execução em uma Cache Compartilhada de 16, 32 e 64 KBytes	87
5.8	Taxas de Acerto para a Cache Compartilhada como Função da Associatividade de Cada Canal	89
5.9	Taxas de Acerto para a Cache Compartilhada como Função da Política de Substituição	90
5.10	Tempos de Execução como Função do Tamanho da Cache Secundári . . .	91
5.11	Tempos de Execução como Função da Taxa de Transmissão	92
5.12	Tempos de Execução como Função da Latência de Leitura na Memória .	93
6.1	Arquitetura dos Nós	100
6.2	Arquitetura de OWCACHE	102
6.3	Tempo de Execução do MP OPTNET e do MP OWCACHE sob <i>Prefetching</i> Ótimo	111
6.4	Tempo de Execução do MP OPTNET e do MP OWCACHE sob <i>Prefetching</i> Básico	111

6.5	Tempo de Execução do MP OWCACHE para 2, 4, e 8 Nós de Entrada/Saída sob <i>Prefetching</i> Ótimo	113
6.6	Tempo de Execução do MP OWCACHE para 2, 4, e 8 Nós de Entrada/Saída sob <i>Prefetching</i> Básico	114
6.7	Tempo de Execução do MP OWCACHE com 256, 512, e 1024-KB sob <i>Prefetching</i> Ótimo	114
6.8	Tempo de Execução do MP OWCACHE com 256, 512, e 1024-KB sob <i>Prefetching</i> Básico	115
6.9	Tempo de Execução do MP OPTNET para Vários Tamanhos de Cache de Disco Combinada sob <i>Prefetching</i> Ótimo	116
6.10	Tempo de Execução do MP OPTNET para Vários Tamanhos de Cache de Disco Combinada sob <i>Prefetching</i> Básico	116
7.1	Tráfego para um Leilão de Mercadorias	128
7.2	Estrutura Topológica da Internet	130
7.3	Estrutura Topológica da Internet-2	137
7.4	Número de Roteamentos para Cada Configuração	144
7.5	Número de Ofertas Atendidas por Segundo para Cada Configuração	145
7.6	Exemplos de Topologias de Rede	148
7.7	Desempenho de Cada Configuração para Diferentes Topologias	148

Lista de Tabelas

2.1	Características da Tecnologia OTDM	18
4.1	Tempos de Leitura para OPTNET, LambdaNet e DMON	53
4.2	Tempos para uma Transação de Coerência em OPTNET, LambdaNet, DMON-U e DMON-I	54
4.3	Descrição das Aplicações e Principais Parâmetros de Entrada	55
4.4	Porcentagens de <i>Write-Stall</i> e <i>Write-buffer Flush</i> para OPTNET, Lamb- daNet, DMON-U e DMON-I.	60
5.1	Tempos de Leitura para NetCache em Ciclos de Processador	76
5.2	Tempos de Leitura para LambdaNet e DMON em Ciclos de Processador	78
5.3	Tempos de uma Transação de Coerência para NetCache, LambdaNet, DMON-U e DMON-I em Ciclos de Processador	79
5.4	Descrição das Aplicações e Principais Parâmetros de Entrada	80
6.1	Parâmetros Base Principais e os seus Valores	105
6.2	Descrição das Aplicações e os seus Principais Parâmetros de Entrada	107
6.3	Tempos Médios de <i>Swap-Out</i> sob <i>Prefetching</i> Ótimo	108
6.4	Tempos Médios de <i>Swap-Out</i> sob <i>Prefetching</i> Básico	109
6.5	Número Médio de Páginas Escritas sob <i>Prefetching</i> Ótimo	109
6.6	Número Médio de Páginas Escritas sob <i>Prefetching</i> Básico	110
6.7	Taxas de Acerto para OWCACHE sob Diferentes Técnicas de <i>Prefetching</i>	110
7.1	Parâmetros Usados pelos Simuladores	137
7.2	Parâmetros da Aplicação	139
7.3	Parâmetros da Rede Virtual	141

Capítulo 1

Introdução

Os recentes avanços na tecnologia de componentes óticos têm permitido a construção de redes óticas extremamente rápidas [1, 2]. Uma rede ótica é normalmente constituída por uma ou mais fibras óticas, além dos respectivos receptores e transmissores de cada nó. Em cada fibra ótica pode-se implementar um número relativamente grande de canais de comunicação independentes, onde cada canal pode ter taxas de transmissão bastante elevadas. Assim, uma das principais características de uma rede ótica é a sua grande largura de faixa (agregada e por canal).

Devido a essa característica, redes óticas vêm sendo consideradas por vários pesquisadores como uma opção no desenvolvimento de sistemas paralelos e distribuídos (e.g., [3, 1]). Na maioria dos casos, essas pesquisas simplesmente substituem a rede eletrônica tradicional por uma rede ótica equivalente, produzindo grandes ganhos de desempenho devido à maior largura de faixa existente no subsistema de comunicação. No entanto, esse tipo de estratégia não explora totalmente o potencial da tecnologia ótica. A possibilidade de implementar canais ponto-a-ponto e canais de disseminação em uma única rede, por exemplo, não é suficientemente explorada.

Nesta Tese propomos a exploração de uma outra característica das fibras óticas no projeto de sistemas paralelos e distribuídos: a sua capacidade para armazenar informação ou, em outras palavras, a capacidade da fibra ótica de atuar como uma memória de linha de retardo (*Delay Line Memory*) [4]. Como a luz se propaga a uma velocidade finita e constante dentro da fibra (aproximadamente $2,1 \times 10^8$ m/s), transcorre um tempo fixo a partir do instante que um dado entra até que esse mesmo dado saia da fibra. Desta forma, se os extremos da linha de retardo são conectados entre si e o sinal ótico é regenerado

periodicamente, a fibra se transforma em uma memória; os dados enviados através da fibra permanecem acessíveis nela, até serem substituídos (sobrescritos) por outros. Devido à sua grande largura de faixa, é possível armazenar uma quantidade razoável de dados em uns poucos metros de fibra (e.g., em um canal de comunicação de 100 metros de comprimento e com uma taxa de transmissão de 10 Gbits/s podem ser armazenados 5 Kbits). A quantidade de informação armazenada na fibra é de fundamental importância, na medida em que o tempo de acesso a esse tipo de memória depende do comprimento da fibra; quanto mais longa a fibra, maior o tempo médio de acesso aos dados.

De forma a alcançar um tempo de acesso relativamente baixo, estamos propondo utilizar redes óticas como memórias cache em diferentes níveis do sistema de memória de sistemas de computação paralelos e distribuídos. Tal esquema tem várias vantagens, tais como: a) o tamanho da nossa memória cache (ou memória de rede) não precisa ser extremamente grande, o que permite tempos de acesso relativamente baixos; b) a cache ótica permite evitar acessos aos níveis mais baixos do sistema de memória, podendo diminuir o tráfego de informação no restante da rede e nos barramentos dos níveis inferiores do sistema de memória; c) a cache ótica pode ser compartilhada por todos os processadores sem contenção; d) a cache ótica pode reduzir qualquer problema de acesso não uniforme aos níveis mais baixos do sistema de memória; e e) a memória de rede garante exclusão mútua no acesso a dados compartilhados sem necessidade de *hardware* ou *software* adicional. Além disso, quando a tecnologia de roteadores ativos [5, 6] é adicionada à nossa memória de rede, ela permite a implementação do que denominamos “cacheamento ativo”, o cacheamento e processamento simultâneos da informação de forma distribuída.

Para quantificar o impacto dessas vantagens no desempenho de vários sistemas paralelos e distribuídos, projetamos e avaliamos quatro tipos de redes óticas, três das quais são capazes de armazenar informação na própria rede. A primeira dessas redes com capacidade de armazenamento é NetCache [7], uma rede de interconexão para os processadores de um multiprocessador que serve como uma cache de terceiro nível para os dados compartilhados pelas aplicações paralelas. Descendo na hierarquia do sistema de memória, também projetamos e avaliamos OWCACHE [8], uma rede de interconexão ótica para multiprocessadores que atua como uma cache compartilhada para escritas ao disco. Tanto NetCache como OWCACHE baseiam-se em uma nova rede ótica, também projeta-

da por nós, chamada de OPTNET (*OPTimized OPTical NETwork*) [9], para realizar a comunicação dos dados. Finalmente, descendo mais ainda na hierarquia do sistema de memória e relaxando o acoplamento existente entre os diferentes elementos de processamento, projetamos e avaliamos um sistema que permite o cacheamento ativo dos dados dinâmicos acessados pelas aplicações distribuídas que executam sobre a Internet.

A avaliação de todos esses sistemas é plenamente satisfatória. Um multiprocessador de 16 nós baseado em OPTNET foi avaliado utilizando simulações detalhadas da execução de aplicações paralelas. Esse multiprocessador foi comparado a outros sistemas também baseados em duas redes de interconexão óticas: DMON [10] e LambdaNet [11]. Essas duas redes são de grande interesse, uma vez que DMON utiliza um *hardware* bastante simples e LambdaNet tem excelente desempenho. Os resultados dessa comparação demonstram que o multiprocessador baseado em OPTNET supera consistentemente os sistemas baseados em DMON para todas as aplicações avaliadas, utilizando a mesma quantidade de *hardware* ótico. Uma comparação entre OPTNET e LambdaNet mostra diferenças de desempenho de 4%, em média, em favor dos multiprocessadores baseados em LambdaNet. Esses resultados são vantajosos para OPTNET, já que LambdaNet requer um *hardware* ótico muito mais custoso do que a nossa rede. Concluimos a partir desses resultados que OPTNET apresenta uma excelente relação custo/desempenho para a implementação de multiprocessadores.

Para verificar que um multiprocessador baseado em NetCache atinge melhores desempenhos do que aqueles baseados em redes óticas que não implementam cacheamento ótico, usamos simulações detalhadas da execução de aplicações paralelas em um multiprocessador de 16 nós com 32 KBytes de cache ótica. NetCache foi também comparada a DMON e LambdaNet. Nossas comparações demonstram que o multiprocessador baseado em NetCache supera consistentemente os sistemas baseados em DMON para todas as aplicações avaliadas. As diferenças de desempenho nesse caso podem ser tão grandes quanto 105%. NetCache também apresenta resultados favoráveis em relação a LambdaNet. Para 9 aplicações, as vantagens no tempo de execução de NetCache estão entre 7 e 79%. Para as outras 3 aplicações, o desempenho dos dois sistemas é comparável. Concluimos, a partir desses resultados, que NetCache apresenta melhor desempenho que qualquer outra rede ótica já proposta.

Na avaliação de OWCACHE, por sua vez, foram utilizadas simulações detalhadas da execução de aplicações paralelas *out-of-core* em um multiprocessador com 8 nós de processamento, com coerência de caches e com 4 nós habilitados para operações de entrada/saída. Foram também considerados os dois extremos em termos de técnicas de *prefetching* de páginas: uma estratégia de *prefetching* ótimo e uma estratégia de *prefetching* básico. Sob a estratégia de *prefetching* ótimo, os resultados demonstram que OWCACHE melhora os tempos de *swap-out* de 1 a 3 ordens de magnitude com relação aos resultados de OPTNET. Os benefícios de OWCACHE não são tão significativos sob a estratégia de *prefetching* básico, mas ainda são consideráveis. Em resumo, OWCACHE melhora o tempo de execução em até 64% sob um *prefetching* ótimo e em até 39% sob um *prefetching* básico, novamente em relação aos resultados de OPTNET. Os benefícios de desempenho de OWCACHE vêm principalmente dos *swap-outs* mais rápidos e da sua capacidade para atuar como uma *victim-cache*. Com a finalidade de mostrar que a memória de rede não é útil apenas para multiprocessadores interconectados por uma rede ótica, projetamos e avaliamos uma extensão ao subsistema de entrada/saída de um multiprocessador convencional que também atua como uma cache de escritas ao disco. Esta extensão é baseada em OWCACHE e apresenta resultados de desempenho similares aos já mencionados [12].

Para avaliar a nossa memória de rede no ambiente da Internet foram utilizadas simulações detalhadas de aplicações distribuídas executando sobre uma rede que reproduz a topologia encontrada na Internet-2 e é baseada na tecnologia de roteadores ativos. Os resultados destes experimentos mostram que, para o caso do cacheamento ativo de dados dinâmicos em leilões eletrônicos, o ganho no número de transações realizadas por unidade de tempo pode chegar a 436% quando comparado a um sistema convencional, onde são utilizados apenas 4 roteadores ativos. Este ganho de desempenho é produto da eliminação da contenção existente no servidor e da diminuição do número médio de *hops* realizados pelas mensagens (66% em média). Como mostra o nosso variado espectro de simulações, estes ganhos são dificilmente atingíveis através de propostas baseadas exclusivamente em roteadores ativos ou em algum outro tipo de processamento distribuído.

Todos estes resultados confirmam que a utilização do cacheamento ótico tem grande potencial de melhorar o desempenho dos sistemas paralelos e distribuídos, especialmente se suportado por redes óticas eficientes.

1.1 Contribuições da Pesquisa

A idéia de usar uma rede ótica como memória cache em sistemas paralelos e distribuídos nunca foi explorada anteriormente. Sendo assim, as contribuições mais importantes da nossa pesquisa podem ser resumidas nos seguintes pontos:

- Propor e avaliar novas redes óticas que permitam explorar a elevada largura de faixa e a capacidade de disseminação oferecidas pelas comunicações óticas na implementação de sistemas paralelos. Nossa contribuição específica nesse ponto é OPTNET.
- Propor e avaliar novas redes óticas que atuem também como memórias cache na implementação de sistemas de computação paralela. Nossas contribuições específicas nesse ponto são NetCache e OWCache.
- Estudar e aplicar estes mesmos conceitos na construção de sistemas de computação distribuída, especificamente nos sistemas distribuídos de grande porte como os encontrados na Internet. Nossa contribuição específica nesse ponto é o cacheamento ativo de dados na Internet.

1.2 Organização da Tese

O resto desta Tese está organizado da seguinte forma. Nos capítulos 2 e 3 apresentamos os conhecimentos básicos que suportam o trabalho que estamos propondo. No capítulo 2 são apresentados os fundamentos das redes óticas, assim como alguns trabalhos relacionados com o uso destas redes na implementação de sistemas paralelos e distribuídos. No capítulo 3, por sua vez, são apresentados os fundamentos das memórias cache e das linhas de retardo óticas. Alguns trabalhos relacionados com a utilização das memórias de linha de retardo ótica são também discutidos. A seguir, o capítulo 4 propõe e avalia a rede OPTNET, enquanto o capítulo 5 propõe e avalia a rede NetCache. O capítulo 6, por sua vez, propõe e avalia OWCache. Posteriormente, o capítulo 7 propõe e avalia a utilização da memória de rede para a otimização de aplicações distribuídas executando sobre a Internet. Finalmente, no capítulo 8 apresentamos as principais conclusões da nossa pesquisa e algumas propostas de trabalhos futuros.

Capítulo 2

Redes Óticas

Neste capítulo apresentamos os fundamentos da comunicação ótica, introduzindo os termos que serão utilizados no restante deste documento. Algumas redes óticas que estão sendo aplicadas na construção de computadores paralelos são também estudadas. Entre as redes mais importantes temos as redes WDM (*Wavelength Division Multiplexing*) e as redes OTDM (*Optical Time Division Multiplexing*). As redes WDM têm sido aplicadas tanto a multicomputadores como a multiprocessadores. OTDM, por sua vez, é uma tecnologia que ainda não está madura, mas tem grande potencial para o futuro.

2.1 Comunicação Ótica

Devido ao potencial das fibras óticas para uma transmissão de dados quase sem perdas e com uma largura de faixa bastante grande, as redes óticas tiveram um crescimento bastante rápido nos últimos 25 anos. Alguns dos conceitos mais importantes nesta área são definidos a seguir.

2.1.1 Fibras Óticas

Uma propriedade interessante dos raios de luz é que quando eles atravessam de uma substância a outra, uma parte da luz é refletida e a outra passa à nova substância. Os raios de luz que ingressam na nova substância sofrem uma mudança na sua trajetória. Este fenômeno é chamado de refração. A quantidade de luz refletida e refratada depende do índice de refração entre as duas substâncias e do ângulo com que a luz atinge a separação entre elas. Desta forma, existe um ângulo de incidência, chamado de ângulo crítico, onde a luz é completamente refletida.

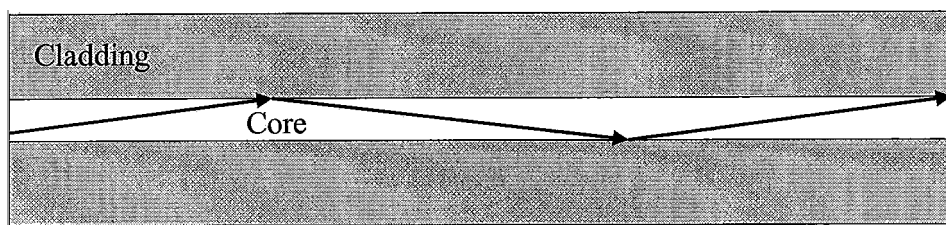


Figura 2.1: Propagação da Luz em uma Fibra Ótica

Usando esta propriedade, é possível criar fios de fibra de vidro extremamente finos que podem transmitir a luz a grandes distâncias. A figura 2.1 mostra a estrutura da fibra e a forma como a luz é transmitida por ela. Um fino fio de vidro, chamado de núcleo, é recoberto de uma outra camada de vidro com um índice de refração ligeiramente diferente do índice do núcleo. Como resultado deste processo, um raio de luz enviado por um extremo do núcleo permanecerá nele até alcançar o outro extremo, já que o raio de luz será refletido a cada vez que tentar passar do núcleo para o material que o recobre.

O índice de refração também tem outro significado: ele estabelece a velocidade com que a luz viaja dentro de uma determinada substância. Para uma fibra ótica o índice de refração é aproximadamente 1,45, o que significa que a luz viaja na fibra a 0,69 da sua velocidade no vácuo. Em outras palavras, a velocidade da luz na fibra é aproximadamente $2,1 \times 10^8$ m/s.

A largura de faixa da fibra é determinada pela quantidade de pulsos de luz que podem ser transmitidos por unidade de tempo. Devido a vários fatores físicos, as fibras apenas permitem usar 3 faixas do espectro ótico. Cada faixa tem uma largura aproximada de 200 nanômetros e elas estão centradas em torno dos 0,85, 1,3 e 1,5 micrômetros. Cada faixa do espectro tem uma largura de aproximadamente 25 THz. Equipamentos convencionais podem transmitir entre 0,7 e 1 bits por Hz, de forma que uma simples fibra pode ser usada para transmitir entre 50 e 75 Tbits/s [2].

Na realidade, a taxa de bits atingível é um pouco menor, já que a construção de tais equipamentos é cara. Mais eficiente é construir vários dispositivos transmitindo em paralelo através de canais independentes com a ajuda de alguma técnica de multiplexação como WDM ou OTDM. WDM permite a multiplexação de centenas de canais através da sua transmissão por diferentes comprimentos de onda. OTDM, por outro lado, aloca um *slot* para cada canal em intervalos de tempo fixos, permitindo a multiplexação de milhares

de canais em uma fibra só. Mesmo deixando espaço entre os diferentes canais, uma única fibra pode ser capaz de transmitir dezenas de Tbits/s com estas técnicas de multiplexação [1].

Apesar de todo esse potencial, a comunicação ótica sofre de alguns problemas. A seguir descrevemos os mais importantes dentre esses problemas:

- **Dispersão.** A dispersão é um problema importante já que ela limita a taxa de bits transmissível em uma fibra de um determinado comprimento. Quanto mais um pulso viaja na fibra, mais aumenta a sua largura, tornando cada vez mais difícil a sua detecção no receptor, pois os pulsos se misturam. Este fenômeno tem sido minimizado através da utilização de fibras mono-modo, de melhorias na fabricação dos lasers e das fibras óticas, e em especial, através da transmissão por *solitons*. Solitons são pulsos de luz que não sofrem nenhuma distorção independente da distância de propagação na fibra, já que, devido à sua forma, os efeitos de dispersão e outros fenômenos físicos na fibra se compensam.
- **Absorção.** Em alguns casos, a luz que atravessa a fibra encontra impurezas no seu caminho ou interage com o material do núcleo de modo a se transformar em outro tipo de energia, e.g., calor. Assim, um pulso de luz pode ser absorvido pela fibra conforme ele avança na sua trajetória. Isto faz com que seja necessário amplificar o sinal periodicamente. Os dispositivos usados para regerar a potência do sinal são chamados de repetidores ou amplificadores. Os repetidores são dispositivos optoeletrônicos que recebem o sinal de entrada, o convertem para o domínio eletrônico, e posteriormente voltam a convertê-lo para o domínio ótico. Os amplificadores óticos, por sua vez, não requerem essa conversão entre domínios, mas não são capazes de consertar distorções como as produzidas pela dispersão. Note que mesmo a transmissão por *solitons* precisa de fases de regeneração, já que pode sofrer absorção.

2.1.2 Transmissores e Receptores

Transmissores e receptores são termos genéricos para os dispositivos ligados à fibra que transmitem e recebem os sinais, respectivamente. Os transmissores são tipicamente lasers

semicondutores fabricados em silício ou *GaAs*. Os receptores, por sua vez, são fotodiodos, as vezes acoplados a filtros que extraem unicamente as frequências de interesse.

Os transmissores e receptores podem ser de dois tipos: fixos e sintonizáveis. Os transmissores e receptores fixos são aqueles que podem transmitir ou receber em um único canal. Os dispositivos sintonizáveis, por outro lado podem selecionar dinamicamente o canal no qual eles transmitem ou do qual recebem a informação. Embora os dispositivos sintonizáveis sejam mais flexíveis e reduzam custos na maioria dos casos, eles apresentam o problema de requerer uma latência adicional para a sintonização, sincronização e recuperação de relógio do novo canal. Estes tempos estão diminuindo conforme avança a tecnologia de componentes óticos, mas ainda apresentam valores consideráveis (na ordem de centenas de nanosegundos) [13].

2.1.3 Técnicas de Acesso ao Meio

Quando vários transmissores e/ou receptores compartilham um mesmo meio de propagação, e.g., um único canal WDM, mecanismos de controle são necessários para garantir o uso correto desse meio durante a transmissão das mensagens. Desta forma, um conjunto de transmissores e/ou receptores precisa ganhar acesso ao meio antes de proceder com a transmissão das mensagens. A seguir apresentamos as técnicas de acesso mais comuns:

- TDMA (*Time Division Medium Access*). Esta técnica de acesso aloca um *slot* em intervalos de tempo fixos para a comunicação entre dois elementos quaisquer. Alguns problemas com esta técnica são a perda da largura de faixa do meio quando um elemento transmissor não tem nada a transmitir, e o fato das mensagens estarem limitadas a um tamanho máximo igual ao tamanho do *slot*.
- TDMA com *slot* variável. Esta técnica de acesso permite que o *slot* alocado para cada transmissão seja de tamanho variável, aliviando assim os problemas apresentados por TDMA. Desta forma, os elementos que não tem nada a transmitir reduzem o seu *slot* a um tempo mínimo e as mensagens podem ter qualquer tamanho. Para determinar o tamanho do *slot* usado para cada transmissão, bits adicionais no cabeçalho da mensagem ou canais de controle adicionais podem ser utilizados.

- CSMA/CD (*Carrier Sense Multiple Access/Collision Detection*). Nesta técnica de acesso ao meio todos os receptores “escutam” todas as transmissões e todos os transmissores podem iniciar uma comunicação a qualquer momento. Como existe a possibilidade de ter transmissões simultâneas, cada elemento transmissor monitora a sua própria comunicação. Se uma transmissão já foi iniciada, o transmissor espera o fim da mesma, antes de tentar iniciar uma nova. No caso de dois ou mais transmissores iniciarem uma comunicação simultaneamente, uma colisão acontece, um sinal de alerta é emitido, e uma retransmissão ocorre após um tempo aleatório. A principal desvantagem desta técnica é o número elevado de colisões em sistemas com grande demanda de comunicação.
- *Token Passing*. Para usar esta técnica, um anel lógico entre os elementos transmissores com direito de acesso ao meio é estabelecido. Através desses elementos um *token* é continuamente transmitido. Quando um elemento transmissor deseja se comunicar, ele espera a chegada do *token*, o retém até transmitir a mensagem, e posteriormente o envia para o próximo elemento no anel. Cada receptor deve sempre verificar a chegada de um *token* ou de uma mensagem dirigida a ele. O maior problema deste esquema é o tempo gasto com a circulação do *token* em sistemas com pouca demanda de comunicação ou quando o número de elementos que compartilham o meio é bastante grande. No entanto, a possibilidade de *starvation* é eliminada, já que existe um tempo máximo para a transmissão de qualquer mensagem.

2.2 Redes WDM

Atualmente, através de uma cuidadosa fabricação das fibras óticas, dos transmissores e dos receptores, é possível construir sistemas de comunicação óticos livres de dispersão, com baixa atenuação e com uma grande largura de faixa. No entanto, devido ao *hardware* associado aos pontos terminais de um sistema de comunicação ótico ser normalmente de natureza eletrônica, as taxas de transmissão são limitadas a uns poucos Gbits/s. Assim, para poder aproveitar o enorme potencial dos sistemas de comunicação óticos, técnicas de multiplexação devem ser utilizadas.

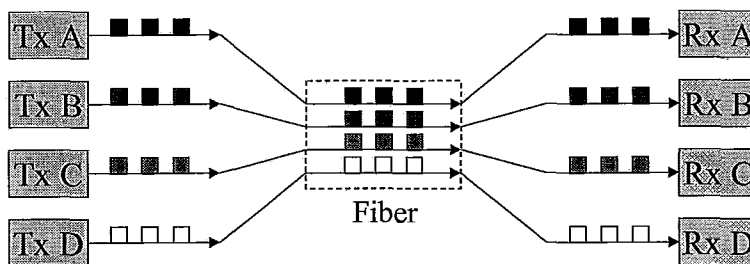


Figura 2.2: A Técnica de Multiplexação WDM

WDM é uma dessas técnicas. Com WDM, vários canais de comunicação independentes podem ser implementados na mesma fibra ótica mediante a divisão da sua largura de faixa. Para isso, cada canal utiliza um comprimento de onda diferente para a transmissão dos seus dados (figura 2.2).

As redes óticas que usam esta técnica de multiplexação são denominadas redes WDM. A forma mais simples de implementar uma rede WDM é através de um acoplador passivo em estrela e um conjunto de receptores e transmissores operando em diferentes comprimentos de onda [13]. O acoplador passivo em estrela dissemina cada comprimento de onda (canal WDM) a todos os nós de processamento conectados à rede. Normalmente, os nós não escutam todos os canais, pois o número de dispositivos óticos determina em grande parte o custo da rede. Na maioria dos casos, cada canal possui um único nó transmissor e um único nó receptor, mas às vezes canais de disseminação são implementados, permitindo que um nó possa transmitir a todos os outros nós da rede.

Devido ao rápido desenvolvimento da tecnologia usada na sua implementação, WDM tem se convertido em uma das mais populares técnicas de multiplexação. Atualmente, podem ser encontrados comercialmente multiplexadores e demultiplexadores WDM com mais de uma centena de canais. Além disso, a chegada dos transmissores e receptores sintonizáveis tem contribuído para uma significativa redução no número de dispositivos óticos usados por estas redes [14, 15].

2.2.1 Classificação das Redes WDM

Em [16], Mukherjee apresentou a seguinte classificação das redes WDM baseado nos tipos de transmissores e receptores utilizados na sua implementação:

1. Transmissores e Receptores Fixos. Nas redes FT-FR (*Fixed Transmitters and Fixed Receivers*), cada nó possui um transmissor ou receptor para cada um dos canais pré-estabelecidos de transmissão ou recepção, respectivamente. Estas são as redes menos flexíveis, já que cada nó está limitado pelo número de canais de transmissão e recepção disponíveis.
2. Transmissores Sintonizáveis e Receptores Fixos. As redes TT-FR (*Tunable Transmitters and Fixed Receivers*) são mais flexíveis que as redes FT-FR, pois um nó pode transmitir para qualquer outro nó com base na sintonização do canal de recepção correspondente. O problema principal destas redes são as colisões que surgem quando dois ou mais transmissores tentam contatar um mesmo receptor. Este problema pode ser tratado usando algum esquema convencional para a resolução de colisões (e.g., CSMA/CD), esquemas de reserva (e.g., TDMA), ou dispositivos óticos especiais (e.g., *Protect-Against-Collision Switches*) que suprimem múltiplas transmissões por um mesmo canal.
3. Transmissores Fixos e Receptores Sintonizáveis. Como as redes TT-FR, as redes FT-TR (*Fixed Transmitters and Tunable Receivers*) são bastante flexíveis. O problema maior destas redes é determinar que canal um nó deve sintonizar para receber uma mensagem qualquer. No entanto, as redes FT-TR apresentam a vantagem de poder implementar *multicast* de forma relativamente simples (um grupo de receptores sintonizando um mesmo canal).
4. Transmissores e Receptores Sintonizáveis. As redes TT-TR (*Tunable Transmitters and Tunable Receivers*) possuem, provavelmente, a arquitetura de rede mais flexível. Mas essa flexibilidade faz com que o protocolo de acesso seja complexo devido à dificuldade de coordenar tanto os transmissores como os receptores para alcançar a conectividade desejada.

Existe também uma outra forma de classificar as redes WDM: redes *single-hop*, onde todos os nós podem se comunicar diretamente com todos os outros nós, e redes *multiple-hop*, onde cada nó pode-se comunicar diretamente com apenas uns poucos nós. Nestas últimas redes, para um nó poder se comunicar com todos os outros nós da rede, um ou

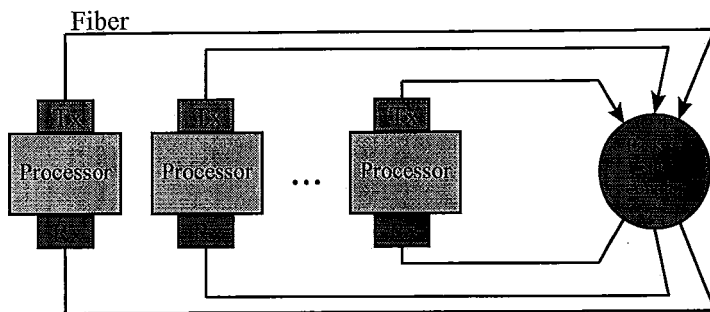


Figura 2.3: Uma Rede WDM *Single-Hop*

mais nós intermediários são utilizados para retransmitir a mensagem até chegar ao nó destino. A seguir analisamos, com mais detalhes, estas classes de redes.

2.2.2 Redes WDM Single-Hop

As redes WDM *single-hop* são as mais simples de implementar. O seu esquema básico é apresentado na figura 2.3. Cada nó é ligado através de uma fibra a algum tipo de dispositivo ótico que multiplexa todas as entradas em todas as saídas. Esse dispositivo de multiplexação pode ser um acoplador passivo em estrela, o qual é um bloco de sílica especialmente projetado que envia uma cópia do sinal recebido em cada entrada através de cada saída.

Note que o efeito de dividir o sinal incidente é que cada um dos N nós recebe só $1/N$ do sinal original. Se N for muito grande, o sinal resultante é muito fraco, tornando difícil a operação a grandes taxas de transmissão. Assim, o sinal incidente deve ter uma potência mínima para manter uma taxa de erros aceitável. No entanto, com a tecnologia atual, podem ser suportadas centenas de nós transmitindo a velocidades na ordem dos Gbits/s.

A rede WDM *single-hop* mais geral é a rede FT-FR na qual cada nó possui C transmissores e C receptores, onde C é o número total de canais. Mas, essa configuração utiliza um número extremamente grande de componentes óticos. Assim, existem vários protótipos de redes WDM *single-hop* sendo construídos que utilizam transmissores e receptores sintonizáveis para diminuir o número de componentes óticos utilizados. A seguir são analisadas algumas dessas propostas:

LambdaNet [11]. O esquema de interconexão LambdaNet permite a ligação de N nós

usando uma abordagem FT-FR. Cada nó na rede tem um único transmissor e N receptores, um para cada canal. O receptor escuta sempre todos os canais e recebe as transmissões que deseja, sem necessidade de um protocolo de acesso para os diferentes canais, nem tempo de sintonização. Além disso, permite a implementação de *multicast*. A desvantagem desta abordagem é a necessidade de N receptores em cada nó, o que aumenta o custo do projeto e do sistema final. LambdaNet será descrita em mais detalhes no capítulo 4.

Optimul [17]. O esquema de interconexão de Optimul também usa uma abordagem FT-FR, e objetiva prover baixa latência com um número pequeno de canais. A idéia básica consiste em dividir os N nós de processamento em C grupos, onde C é o número de canais. Os nós dentro de cada grupo usam um canal de transmissão comum. Em cada nó existem C receptores que permitem a recepção simultânea dos C canais. Além disso, cada grupo tem uma lógica externa (eletrônica) de acesso que permite a seleção dos pedidos de transmissão provenientes de nós distintos para um mesmo canal. A sua arquitetura é similar a LambdaNet exceto pelo fato de reduzir o número de receptores através do compartilhamento dos canais.

Rainbow [18]. Esta é uma rede que usa a abordagem FT-TR, onde cada nó possui um canal exclusivo para a transmissão das suas mensagens. Para resolver o problema de determinar que canal um nó deve sintonizar, Rainbow utiliza um protocolo de busca do receptor. Se o nó A deseja se comunicar com o nó B, A sintoniza o canal de transmissão de B, e envia um sinal de *polling*, pelo seu canal, contendo os endereços de A e B. Se B não está se comunicando com um outro nó, o seu receptor estará procurando entre os diferentes canais por uma mensagem de *polling* com o seu endereço. Eventualmente, B receberá a mensagem de *polling* de A, e enviará uma mensagem de conexão pelo seu canal. O nó A, então, receberá essa mensagem de conexão e começará a transmitir a mensagem. A principal vantagem destas redes, além de suportar *multicast*, é o fato de requererem só um transmissor e um receptor por nó. As suas desvantagens são, por outro lado, o tempo elevado para estabelecimento de uma conexão e a necessidade de mecanismos para evitar *deadlocks*.

Star-Track [13]. O esquema apresentado por esta rede é também FT-TR, mas para resolver o problema de determinar que canal um nó deve sintonizar, Star-Track utiliza uma

rede eletrônica em anel para a passagem de *tokens*. Os *tokens* são usados para ganhar acesso aos canais WDM e determinar a sintonização dos receptores em um instante determinado. Em cada *token* existem *subtokens*, um para cada receptor. Quando o *token* chega a um nó que deseja transmitir, o nó escreve o seu endereço no *subtoken* associado ao nó destino (só se o *subtoken* estiver vazio). Quando o *token* eventualmente chegar ao nó receptor, ele verificará o seu *subtoken* e determinará o canal que deve ser sintonizado para poder receber a próxima mensagem.

DMON [10]. Esta rede utiliza um esquema $\{F,T\}T\text{-FR}$, ou seja, cada nó pode receber mensagens através de um único canal. Para garantir o mecanismo de controle de acesso a estes canais, um canal adicional de disseminação é usado. Este canal, chamado de canal de controle, usa TDMA como protocolo de acesso ao meio. Quando um nó quer transmitir por um dos canais de comunicação ponto-a-ponto, ele deve primeiro esperar pela sua vez de acessar o canal de controle e então disseminar a sua intenção de transmitir no canal ponto-a-ponto. Essa disseminação faz com que os outros nós saibam da comunicação que vai se realizar, evitando qualquer conflito. Como DMON é uma rede orientada ao suporte de multiprocessadores, ela também possui um segundo canal adicional de disseminação para a transmissão de eventos globais. Este último canal também usa TDMA como protocolo de acesso ao meio. DMON será descrita em mais detalhes no capítulo 4.

OPTNET [9]. Esta rede utiliza um esquema $FT\text{-}\{F,T\}R$. De forma similar a DMON, ela também apresenta um canal de disseminação, chamado de canal de pedidos, que usa TDMA como protocolo de acesso ao meio. As mensagens que fluem através dos canais de comunicação ponto-a-ponto são as respostas às solicitações enviadas pelo canal de pedidos. Assim, um nó sabe antecipadamente qual é o canal que o seu receptor deve sintonizar, pois conhece o nó para quem ele fez o pedido. Além disso, esta rede possui outros dois canais de disseminação para a transmissão de eventos globais. Estes dois últimos canais adicionais usam TDMA com *slot* variável como protocolo de acesso ao meio. OPTNET está sendo proposta no contexto desta tese e será descrita em mais detalhes no capítulo 4.

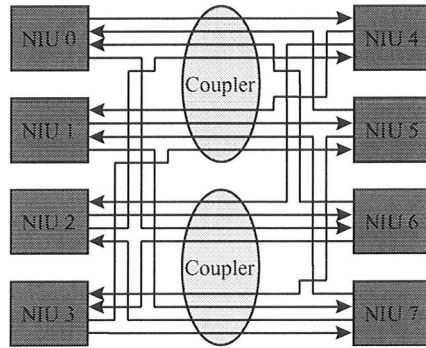


Figura 2.4: Topologia da Rede TeraNet

2.2.3 Redes WDM Multiple-Hop

Uma forma de eliminar a limitação de tamanho das redes *single-hop*, assim como evitar a necessidade de transmissores e/ou receptores sintonizáveis, é construir uma rede WDM *multiple-hop* [19]. Neste tipo de rede, os transmissores e receptores são geralmente fixos em um determinado comprimento de onda,¹ e cada nó pode se comunicar diretamente com apenas uns poucos nós. Para um nó poder se comunicar com todos os outros nós da rede, um ou mais nós intermediários devem ser utilizados para retransmitir a mensagem até o nó destino. Devido ao fato de que a maioria de redes WDM *multiple-hop* não precisam dividir o sinal de entrada entre várias saídas, a potência mínima requerida para ter uma taxa de erros baixa é muito menor que em uma rede *single-hop*.

As redes WDM *multiple-hop* podem ser projetadas de várias formas. A idéia básica é construir um grafo de conectividade entre os nós, tal que o número de *hops* necessários para que dois nós quaisquer possam se comunicar seja pequeno. Desta forma, o número de vezes que uma mensagem tem que ser retransmitida até chegar ao nó destino também é pequeno. Poucas redes deste tipo têm sido construídas, alguns exemplos são apresentados a seguir.

TeraNet [2]. Neste sistema, os nós de processamento são ligados à rede através das suas NIUs (*Network Interface Units*). Cada NIU tem dois transmissores e dois receptores, como mostra a figura 2.4. Todos os transmissores e receptores são fixos em diferentes comprimentos de onda. Os comprimentos de onda ao longo da rede são arranjados de forma que cada um deles possua um único transmissor e um único receptor. Assim,

¹Algumas vezes são usados dispositivos sintonizáveis para adaptar a rede a possíveis falhas ou como resposta a uma mudança substancial nos padrões de tráfego.

cada NIU pode enviar dados a duas NIUs e recebê-los de duas outras, possivelmente diferentes das primeiras. Note que na figura 2.4, a NIU 0 pode enviar dados às NIUs 4 e 6, e pode receber das NIUs 5 e 6. As NIUs, por sua vez, estão conectadas através de múltiplos acopladores passivos em estrela. A tarefa é então garantir que exista um caminho de comunicação entre duas NIUs quaisquer. Na figura 2.4, por exemplo, para a NIU 0 se comunicar com a NIU 3 é necessário passar pela NIU 6. Observe também que existem outros possíveis caminhos para essa mesma comunicação (e.g., NIU 0, NIU 4, NIU 1, NIU 5, NIU 3). Uma vantagem importante da rede TeraNet é que ela não necessita protocolos de controle de acesso para mediar a comunicação, apesar do pequeno número de receptores e transmissores utilizados.

2.2.4 Processamento Paralelo e Distribuído com Redes WDM

Uma rede WDM é ideal para a computação paralela ou distribuída, pois ela pode prover canais ponto-a-ponto entre cada par de nós ou canais compartilhados com capacidade de disseminação. Quando se usa uma rede WDM na implementação de computadores paralelos ou distribuídos, os diferentes canais de comunicação devem ser arranjados de forma a permitir que os elementos de processamento se comuniquem sob condições ótimas de baixa latência, uma grande largura de faixa e escalabilidade. No entanto, três desafios surgem em torno deste tipo de projeto:

- Fazer uso efetivo da largura de faixa oferecida pela fibra ótica, tanto implementando um número maior de canais WDM em uma única fibra (*dense WDM*), como melhorando as condições efetivas de operação dos diferentes dispositivos optoeletrônicos.
- Implementar protocolos de coerência e sincronização que se beneficiem da imensa largura de faixa das redes WDM.
- Minimizar a quantidade de tempo gasto na decisão de que processadores vão se comunicar por que canais, e maximizar o tempo da transferência de dados.

Alguns exemplos da utilização de redes óticas na implementação de computadores paralelos são LambdaNet, Optimul, DMON e OPTNET. DMON e OPTNET apresentam

Propriedade	1995	Futuro
Número de canais	250	5000
Largura de faixa de cada canal	1 Gbit/s	50 Gbits/s
Tempo de seleção de um canal	$< 5 \eta\text{seg}$	$< 2 \eta\text{seg}$
Custo por nó	$\sim \text{US\$ } 20000$	$\sim \text{US\$ } 1000$

Tabela 2.1: Características da Tecnologia OTDM

um número de componentes óticos menor do que LambdaNet e Optimul, sendo que o número de componentes de Optimul é inferior ao de LambdaNet. Além disso, o protocolo de coerência de DMON e OPTNET é um tanto mais elaborado que o de LambdaNet e Optimul. No que se refere ao tempo utilizado na arbitragem dos canais, LambdaNet não consome tempo nenhum, Optimul gasta menos que OPTNET e OPTNET, por sua vez, gasta menos que DMON. Todas estas redes serão descritas e comparadas com maiores detalhes no capítulo 4.

2.3 Redes OTDM

As redes óticas OTDM (*Optical Time Division Multiplexing*) foram propostas como uma alternativa às redes WDM, e.g., [20, 21]. Esta abordagem usa um único comprimento de onda que é disseminado a todos os nós para transportar os diferentes sinais ou canais de comunicação, sendo que cada canal usa frequências de relógio compatíveis com as dos componentes eletrônicos.

As redes OTDM possuem algumas vantagens em relação às redes WDM. Entre as vantagens principais estão: a) a sincronização inerente à multiplexação OTDM permite a implementação de um controle e arbitragem mais eficientes, b) OTDM requer, na maioria das vezes, um demultiplexador ativo e um sistema para o alinhamento de canais, mas não requer um controle preciso do filtro do receptor nem do comprimento de onda do transmissor, e c) requer um único laser. No entanto, a tecnologia OTDM ainda não é suficientemente madura. Espera-se que os rápidos avanços nas pesquisas nesta área permitam um grande desenvolvimento da tecnologia OTDM nos próximos anos. Na tabela 2.1 são resumidas as propriedades mais importantes das redes OTDM, mostrando tanto os valores alcançados em 1995 como os potencialmente alcançáveis no futuro [21].

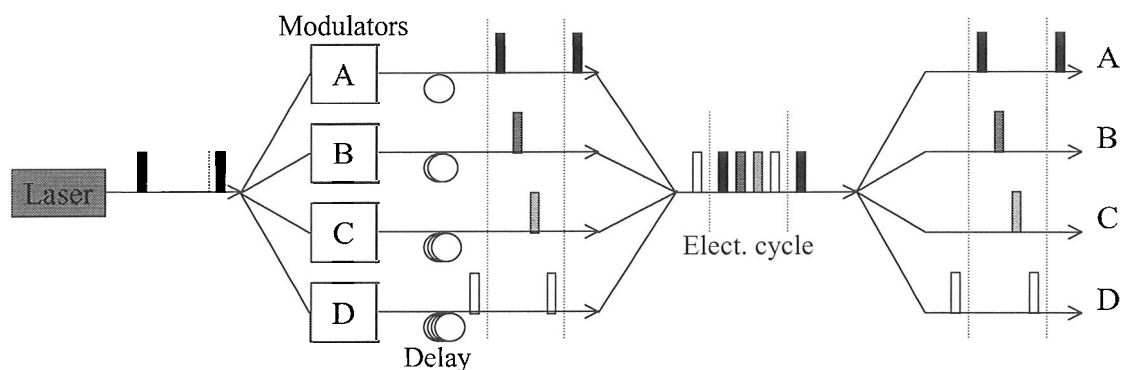


Figura 2.5: Sistema de Transmissão OTDM

2.3.1 A Tecnologia OTDM

A figura 2.5 mostra o esquema de um sistema de transmissão OTDM com $N=4$ canais. Uma seqüência de pulsos de relógio óticos extremamente curtos (sua duração é na ordem dos pico ou femtosegundos) proveniente de um laser é dividida em N partes. Cada seqüência é individualmente modulada por um sinal de dados eletrônico e retardada por uma fração do período de relógio. Essas seqüências de pulsos modulados são multiplexadas de forma passiva (operação OR) para serem transmitidas através de uma única fibra. Isto permite obter uma taxa de transmissão agregada N vezes superior a taxa de transmissão eletrônica, a qual é observada pelos moduladores e receptores. Note que tanto os moduladores como os receptores são dispositivos optoeletrônicos controlados, a maioria das vezes, por *drivers* eletrônicos. Note também que a necessidade do laser produzir pulsos extremamente curtos (menores que $1/N$ do período do relógio) é para evitar interferência entre os diferentes canais multiplexados.

Do lado do receptor, essa seqüência de pulsos única é demultiplexada e o seu relógio recuperado. Desta forma, o sinal ótico de entrada é separado nos diferentes canais de dados. A demultiplexação pode ser feita através de um esquema optoeletrônico ou totalmente ótico.

A potência e comprimento de onda do sinal ótico são ajustados para otimizar as características de transmissão na fibra. Amplificadores óticos podem ser utilizados para manter a potência correta do sinal e assegurar uma relação sinal/ruído suficiente para operar livre de erros. A dispersão pode também ser facilmente controlada nestes sistemas através da utilização de técnicas de transmissão por *solitons*.

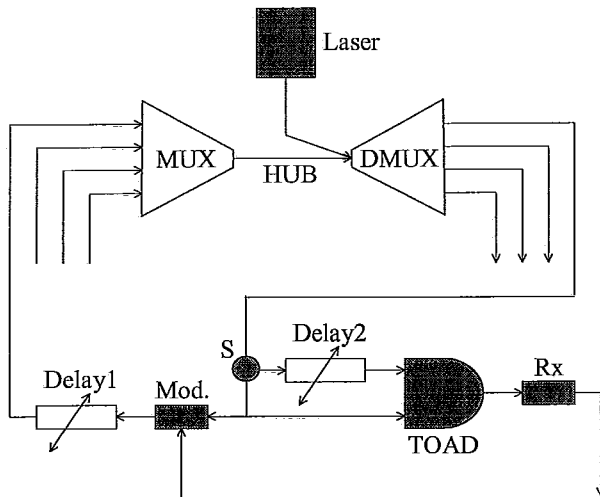


Figura 2.6: Sistema OTDM para Multiprocessadores e Multicomputadores

2.3.2 Processamento Paralelo e Distribuído com Redes OTDM

Esta tecnologia, embora ainda não muito desenvolvida, também tem sido utilizada para o projeto de computadores paralelos e distribuídos. Por exemplo, Nowatzky e Prucnal [21] propuseram um multiprocessador totalmente interconectado através de uma rede OTDM para tomar vantagem da sua capacidade de disseminação escalável. A sua proposta reconhece que a ótica produz oportunidades únicas para a simplificação do protocolo de coerência de caches e a sincronização nos multiprocessadores escaláveis.

Na figura 2.6 mostramos o esquema do sistema proposto em [21]. Os pulsos de relógio do laser são separados por um divisor ótico passivo (“DMUX”) que distribui o sinal a todos os nós. Em cada nó, um modulador (“Mod.”) codifica os dados nos pulsos de luz, para posteriormente serem enviados a um elemento de retardo programável (“Delay1”) que determina em qual canal esses dados aparecerão. A seguir, a saída de todos os nós é enviada de volta para um acoplador central (“MUX”) que combina todos os pulsos para redistribuí-los novamente a todos os nós.

O receptor em cada nó, por sua vez, usa um dispositivo (“S”) que separa os pulsos de dados dos pulsos de relógio (e.g., em virtude da sua polarização), e passa os dados a um outro elemento de retardo programável (“Delay2”) que seleciona o canal a ser recebido. O pulso de relógio é então utilizado para abrir uma porta lógica AND que permite isolar os dados do canal selecionado dos outros dados. O fluxo de bits resultante é então enviado a um fotodiodo que converte os dados de volta ao domínio eletrônico. Note que a porta

lógica AND é implementada com um dispositivo chamado de TOAD (*Terahertz Optical Asymmetric Demultiplexer*) [22] que permite taxas de transmissão de 250 Gbits/s, sendo que os limites tecnológicos indicam que uma operação a taxas superiores aos 5 Tbits/s é concebível.

Uma característica importante deste sistema é que os componentes eletrônicos determinam a frequência do canal. Além disso, cada nó é capaz de transmitir e receber em qualquer um dos canais. Em particular, os nós são também capazes de receber a sua própria transmissão. Esta capacidade permite que um nó compense o retardo da propagação do sinal entre o seu modulador e o acoplador central, de forma a alinhar as transmissões de todos os nós.

2.4 Outras Redes Óticas

Embora existam muitas propostas de canais de interconexão óticos, como por exemplo Fibre-Channel [23], OPTOBUS [24] e SuperHIPPI, entre outros, poucas são as redes comerciais que utilizam eficientemente a comunicação ótica na transmissão de suas mensagens. Isso talvez se deva aos preços relativamente altos que a tecnologia ótica ainda apresenta na atualidade. Os exemplos mais conhecidos deste tipo de rede, além das redes WDM e OTDM, são as redes FDDI, ATM e Gigabit-Ethernet.

2.4.1 Redes FDDI

As redes FDDI (*Fiber Distributed Data Interface*) [25] são redes originalmente projetadas para operar sobre enlaces de fibra ótica a 100 Mbits/s e com distâncias entre estações que podem chegar a 60 quilômetros. Atualmente, o padrão FDDI também inclui suporte para a interconexão de nós através de cabos de cobre, mas com distâncias menores que 100 metros.

Fisicamente, as redes FDDI são formadas por dois anéis que ligam vários dispositivos chamados de concentradores (figura 2.7). Cada nó da rede deve estar ligado a um desses concentradores. A transmissão de dados através dos anéis da rede é feita em sentidos contrários para facilitar a recuperação de falhas. Além disso, estas redes permitem a transmissão síncrona ou assíncrona dos pacotes de dados através dos anéis. O protocolo de acesso ao meio usado pelas redes FDDI é o de passagem de *tokens*.

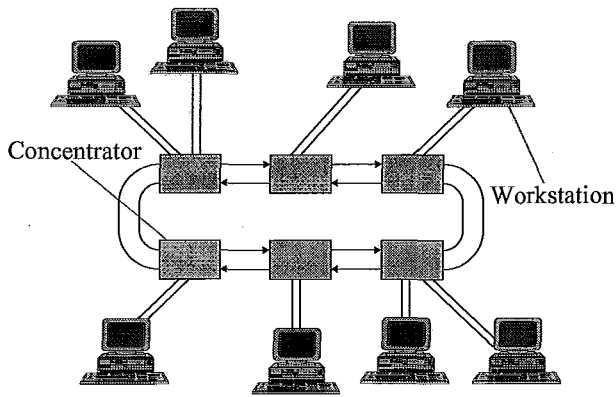


Figura 2.7: Uma Rede FDDI

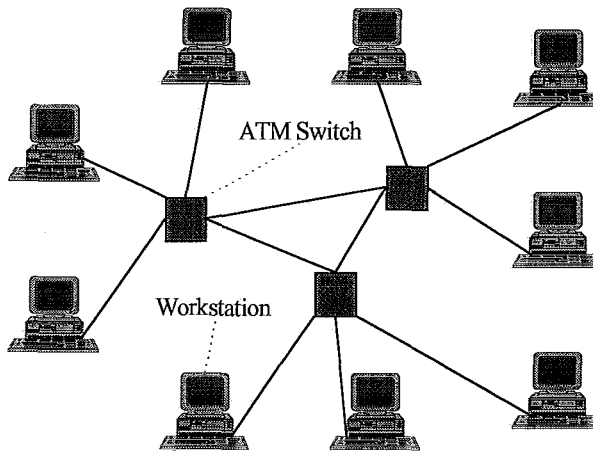


Figura 2.8: Uma Rede ATM

Estas redes não foram utilizadas mais intensamente na área de processamento paralelo por apresentarem características de desempenho ainda inferiores às requeridas por estes sistemas. Apesar da taxa de transmissão nominal ser de 100 Mbits/s, taxas de 75 Mbits/s apenas são atingidas em média na prática. Além disso, com a chegada de novos padrões de rede, como por exemplo ATM e Gigabit-Ethernet, elas estão sendo substituídas.

2.4.2 Redes ATM

As redes ATM (*Asynchronous Transfer Mode*) usam um protocolo de comutação de rede orientado a conexão para transmitir pequenas unidades de tamanho fixo, chamadas de células. A idéia das células é facilitar a comutação por *hardware*. A camada física das redes ATM normalmente utiliza outros padrões já existentes, como por exemplo SONET (*Synchronous Optical Network*) ou Fibre-Channel, para permitir taxas de transmissão de 155, 622 e potencialmente 2480 Mbits/s sobre fibras óticas. Uma rede ATM típica é

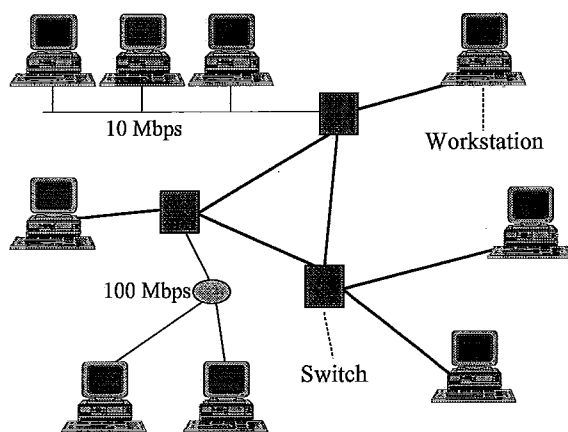


Figura 2.9: Uma Rede Gigabit-Ethernet

apresentada na figura 2.8. Cada processador está conectado a um comutador ATM, e todos os comutadores estão ligados por enlaces ponto-a-ponto com dois canais, um para a transmissão e outro para a recepção das células.

No que se refere à utilização destas redes no processamento paralelo, as redes ATM têm sido estudadas principalmente no contexto dos multicomputadores fracamente acoplados. Alguns exemplos de tais estudos são: o uso de redes ATM para simplesmente aproveitar a sua maior taxa de transmissão em relação a outras redes de área local existentes [26, 27], a implementação de primitivas de comunicação coletiva com ajuda dos comutadores ATM [28], o suporte de acessos a memória remota com baixa latência usando redes ATM [29], a implementação de mensagens ativas em redes ATM [30, 27], e uma interface de rede a nível do usuário [31].

Tais estudos mostram que a comunicação através de redes ATM atuais não alcança o patamar de largura de faixa e latência desejado em computação paralela. Desta forma, mesmo sendo o desempenho das redes ATM superior ao de outras redes locais existentes (e.g., *Ethernet*, *Token-Ring*, *Fast-Ethernet*, entre outras), a sua utilização ainda está orientada às aplicações paralelas e distribuídas com uma relação computação/comunicação elevada. Além disso, a natureza em rajadas dos padrões de comunicação encontrados nas aplicações paralelas requer algumas modificações no padrão ATM para que o seu desempenho seja tão robusto como o obtido em computadores paralelos dedicados.

2.4.3 Redes Gigabit-Ethernet

Gigabit-Ethernet é uma extensão do padrão IEEE 802.3 (*Ethernet*) que permite a atualização progressiva da infraestrutura de rede local já existente. Anteriormente, o padrão 802.3 incluía taxas de transmissão de 1 a 100 Mbits/s (*Fast-Ethernet*). Gigabit-Ethernet aumenta essas taxas de transmissão para 1 Gbit/s, suportando uma comunicação comutada bidirecional tanto através de cabos de cobre como por meio de redes óticas. Na prática, estas redes apresentam taxas de transmissão efetivas próximas dos 950 Mbits/s [32]. Como Gigabit-Ethernet utiliza a mesma filosofia e protocolos dos seus predecessores, a interconexão com redes *Ethernet* e *Fast-Ethernet* é simples e facilmente escalável através de comutadores e/ou repetidores (figura 2.9). Além disso, assim como ATM, Gigabit-Ethernet permite a implementação de mecanismos que garantem qualidade de serviço na transmissão dos dados.

Mesmo quando são utilizadas fibras óticas para ligar os elementos de processamento aos comutadores, a distância máxima entre os equipamentos terminais da rede é de apenas 320 metros, devido às restrições temporais do protocolo de acesso ao meio usado por este tipo de redes (CSMA/CD) [33].

Como Gigabit-Ethernet é uma tecnologia relativamente nova, ainda não existem trabalhos avaliando o impacto deste tipo de rede no desempenho de aplicações paralelas e/ou distribuídas. No entanto, a expectativa é que Gigabit-Ethernet apresente um desempenho igual ou melhor que as redes ATM devido a sua grande largura de faixa e a sua capacidade natural de disseminação.

2.5 Conclusões

Devido às características das redes óticas, essas redes têm sido consideradas no projeto de sistemas de computação paralela e distribuída. É importante notar, entretanto, que os ganhos de desempenho apresentados por estas redes não provêm somente da sua maior largura de faixa, mas principalmente da possibilidade de simplificar e otimizar os protocolos de coerência e/ou sincronização usados nesses sistemas, através da comunicação por disseminação. Desta forma, as diferenças de desempenho entre um sistema baseado em uma rede eletrônica e um sistema baseado em uma rede ótica podem ser bastante

significativas, mesmo que as latências de comunicação em ambos os tipos de rede sejam bastante próximas.

Para aproveitar todo o potencial da ótica, técnicas de multiplexação (e.g., WDM ou OTDM) são normalmente utilizadas. Nos sistemas apresentados nos capítulos 4, 5 e 6 será utilizada a técnica de multiplexação WDM devido a sua imediata disponibilidade, mas nada nesses sistemas é intrinsecamente dependente desta técnica de multiplexação.

Capítulo 3

Sistemas de Memória

Neste capítulo apresentamos alguns fundamentos sobre os sistemas de memória. Como as caches óticas que propomos podem ser organizadas de forma semelhante às caches eletrônicas, começamos o capítulo descrevendo os conceitos principais dessas caches tradicionais. Em seguida, abordamos os conceitos principais relativos às memórias de linha de retardo ótica.

3.1 Memórias Cache Tradicionais

Uma das propriedades mais importantes e mais exploradas nos programas de computador é a sua localidade de referência. Em programas seqüenciais, a localidade de referência pode ser de dois tipos. O primeiro é a localidade temporal, a qual estabelece que os dados e instruções recentemente acessados têm uma probabilidade maior de serem acessados em um futuro próximo. O segundo tipo é a localidade espacial, a qual estabelece que os dados e instruções com endereços próximos apresentam uma certa tendência a serem referenciados juntos.

Esses dois tipos da localidade de referência, aliados ao fato de que memórias mais rápidas são mais caras, produziram o conceito de hierarquia de memória: memórias pequenas e rápidas são usadas para manter os elementos mais recentemente acessados mais próximos do processador. A hierarquia de memória pode ter vários níveis, sendo os mais comuns (de menor a maior capacidade): registradores, cache, memória principal e memória secundária (disco). Nesse trabalho nos concentramos nas memórias cache.

Uma cache é uma memória pequena e rápida localizada perto do processador para manter os dados e instruções mais recentemente acessados. Quando o processador en-

contra na cache o elemento que solicita, temos um “acerto” na cache. Por outro lado, quando o processador não encontra na cache o elemento solicitado, temos uma “falha” na cache. Neste último caso, um bloco de dados ou instruções de tamanho fixo contendo o elemento solicitado, chamado de bloco de cache, é recuperado da memória de nível inferior na hierarquia e colocada na cache. As caches possuem uma entrada para cada bloco de cache que elas podem armazenar. O fato de um bloco da cache ser geralmente maior que o tamanho dos elementos individuais acessados pelo processador é baseado no princípio da localidade espacial.

Nos sistemas atuais, entre o processador e a memória principal pode existir mais de um nível de cache. Normalmente existe uma cache primária no mesmo *chip* do processador, e uma cache secundária externa a ele. No entanto, existem também sistemas que incorporam caches de terceiro nível na sua hierarquia. Em alguns casos, a cache primária é dividida em duas, uma cache para dados e outra para instruções. A cache secundária em geral é única, misturando tanto blocos de dados como de instruções.

Desta forma, cada memória cache possui um conjunto de características que determinam, por exemplo, a posição de um bloco de memória dentro da cache, a forma de identificação de cada um desses blocos, o procedimento para a substituição de um bloco, a maneira como são feitas as escritas através da cache. Nas subseções seguintes discutimos essas características, abordando as opções mais comuns para a sua implementação.

3.1.1 Associatividade das Caches

Existem três abordagens para determinar qual é a entrada da cache que um bloco recuperado do nível inferior da hierarquia deve ocupar:

1. Se o bloco tem uma única entrada onde ele pode aparecer, a cache é denominada diretamente mapeada. O mapeamento é calculado normalmente através do módulo do endereço do bloco pelo número de blocos na cache.
2. Se o bloco pode ser colocado em qualquer entrada, a cache é denominada totalmente associativa.
3. Se o bloco pode ser colocado em um conjunto restrito de entradas, a cache é denominada associativa por conjunto. Assim, um bloco é primeiro mapeado em um

conjunto e depois colocado em qualquer uma das entradas desse conjunto. O mapeamento do conjunto é calculado normalmente através do módulo do endereço do bloco pelo número de conjuntos na cache.

3.1.2 Identificação de Blocos

As caches possuem uma etiqueta de endereço para cada uma das suas entradas. Esta etiqueta armazena o endereço do bloco presente nessa entrada naquele instante. Em resposta a um pedido do processador, todas as etiquetas da cache que podem conter o bloco solicitado são verificadas em paralelo para determinar se o bloco está ou não presente na cache.

Normalmente, um bit extra, chamado de *valid-bit*, é também incluído na etiqueta de cada entrada. Esse bit tem como função indicar se o endereço daquela entrada é válido ou não. Se este bit não está ativado, a verificação do seu endereço não será realizada.

3.1.3 Substituição de Blocos

Quando acontece uma falha na cache, o controlador deve selecionar um bloco presente nela para ser substituído pelo bloco solicitado. Uma vantagem das caches diretamente mapeadas é a simplificação desta decisão: existe somente uma entrada onde o novo bloco pode residir. Nas caches associativas por conjunto ou totalmente associativas, no entanto, uma das seguintes estratégias de seleção é normalmente implementada:

- Aleatória. Os blocos são aleatoriamente selecionados para espalhar uniformemente as alocações dos blocos nas entradas.
- LRU (*Least Recently Used*). O bloco selecionado é o que não foi utilizado por um tempo maior. Neste caso, o princípio da localidade temporal é aplicado.

Para a implementação da estratégia LRU, cada entrada da cache deve possuir alguns bits adicionais para gravar a ordem de acesso aos blocos. Devido à limitação no número de bits disponíveis nas etiquetas de cada entrada, políticas aproximadas são implementadas na maioria dos casos.

3.1.4 Estratégias de Escrita

As políticas de escrita normalmente diferenciam os projetos das caches. As duas opções básicas quando acontece uma escrita na cache são:

1. *Write-through*. A informação é escrita tanto no bloco da cache como na memória de nível inferior.
2. *Write-back*. A informação é escrita unicamente no bloco da cache. O bloco modificado é escrito na memória de nível inferior somente quando ele for substituído.

No caso da política *write-back*, um bit extra, chamado de *dirty-bit*, é usado para marcar as entradas com blocos modificados que ainda não foram atualizados na memória. Adicionalmente, como os dados não são necessários em uma escrita, duas são as opções mais comuns em uma falha de escrita:

1. *Write-allocate*. O bloco é primeiro carregado, como no caso de uma falha de leitura, e só depois feita a escrita.
2. *Write-around*. O bloco é modificado na memória de nível inferior, não precisando carregá-lo na cache.

3.2 Memórias de Linha de Retardo

A velocidade finita de propagação da luz nas fibras e as altas taxas de transmissão que podem ser atingidas nos sistemas óticos fazem com que as fibras possam ser utilizadas como linhas de retardo, e estas, por sua vez, como meios de armazenamento óticos [4, 34]. Esta é uma idéia que ainda não foi explorada para cacheamento de dados e muito menos dentro da área de processamento paralelo e distribuído.

Uma opção para a implementação de uma memória de linha de retardo é apresentada na figura 3.1 [35]. O módulo de entrada e realimentação permite escrever novos dados na memória ao mesmo tempo que mantém circulando os dados anteriormente ingressados. O módulo de restauração do sinal faz com que os erros por perdas de dispersão e/ou absorção sejam praticamente eliminados. A leitura dos dados armazenados neste tipo de memória

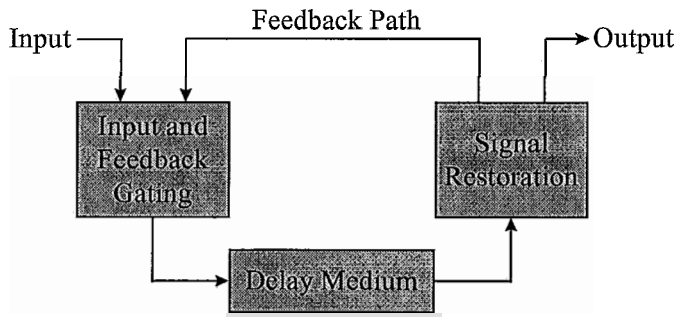


Figura 3.1: Diagrama de Blocos de uma Memória de Linha de Retardo

é normalmente realizada após a fase de restauração do sinal para diminuir a probabilidade de erros de leitura.

As linhas de retardo usadas para a implementação da memória podem usar tanto uma comunicação síncrona como uma comunicação assíncrona. Na comunicação síncrona, cada bloco de dados tem um tempo específico para ser inserido ou chegar em um ponto determinado da fibra. Em contraste, na comunicação assíncrona, um bloco de dados não tem um tempo específico para chegar ou ser inserido na fibra, mas os bits que o formam têm restrições de tempo específicas a partir do instante em que o bloco é detectado na fibra.

Na comunicação síncrona, as restrições de temporização apresentam algumas vantagens e desvantagens. As principais vantagens são a utilização completa da largura de faixa do canal e a simplificação do *hardware* necessário para a sua implementação, já que os pulsos de luz que codificam a informação só podem aparecer em tempos específicos e previamente estabelecidos. Por outro lado, a sua maior desvantagem é que os retardos obtidos nas fibras óticas dependem de alguns fatores externos que dificultam a manutenção das restrições de temporização do sistema. As variações de temperatura, por exemplo, mudam o comprimento da fibra, alterando assim o retardo produzido pela fibra.

A comunicação assíncrona também apresenta algumas vantagens e desvantagens. A sua vantagem principal é que mesmo que ocorram variações pequenas e contínuas no retardo produzido pela fibra, o funcionamento correto do sistema é garantido. Por outro lado, as desvantagens deste tipo de comunicação são o desperdício da largura de faixa do canal com bits que identificam o início e fim de cada bloco de dados, e a maior complexidade do *hardware* necessário para a leitura desses blocos.

A seguir apresentamos alguns exemplos de implementação das memórias de linha de

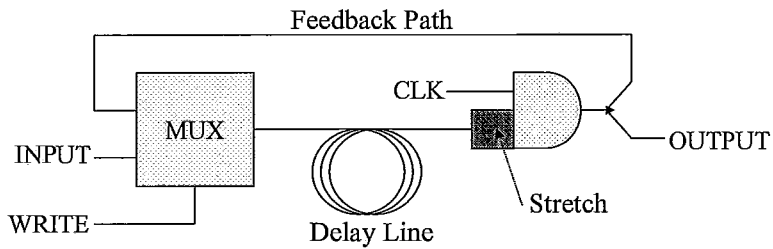


Figura 3.2: Memória de Linha de Retardo Síncrona

retardo síncronas e assíncronas, assim como os principais problemas que enfrentam esses tipos de estruturas.

3.2.1 Memórias Síncronas

Existem duas alternativas tecnológicas para a implementação de memórias síncronas: a que utiliza componentes totalmente óticos e a que só utiliza componentes optoeletrônicos. A implementação de uma memória de linha de retardo síncrona através de componentes totalmente óticos oferece a vantagem de poder usar taxas de transmissão bastante elevadas, permitindo armazenar mais dados em um comprimento de fibra menor e com latências também menores.

Por ter características interessantes e mostrar as possibilidades que oferecem os componentes totalmente óticos, a seguir descrevemos a implementação da memória de linha de retardo síncrona proposta em [35].

A memória foi implementada de acordo com o esquema apresentado na figura 3.2. O multiplexador (“MUX”) atua como o módulo de entrada e realimentação da figura 3.1, permitindo a passagem de somente uma das suas entradas dependendo do sinal externo *WRITE*. Quando *WRITE* está ativado, a entrada de novos dados é permitida. Por outro lado, quando o sinal *WRITE* está desativado, a realimentação dos dados é realizada. A restauração do sinal é feita através da operação lógica AND entre o sinal de relógio *CLK* e os dados provenientes da linha de retardo. Desta forma, como se observa na figura 3.3, pulsos novos, livres de dispersão e absorção, são gerados para continuar com o processo de realimentação.

Para garantir uma sobreposição entre o sinal de entrada e o sinal de relógio *CLK* (figura 3.3), um circuito de *stretching* é utilizado na entrada dos dados para aumentar a largura dos pulsos por um fator $2 \times \delta t$. Desta forma, o circuito de *stretching* também

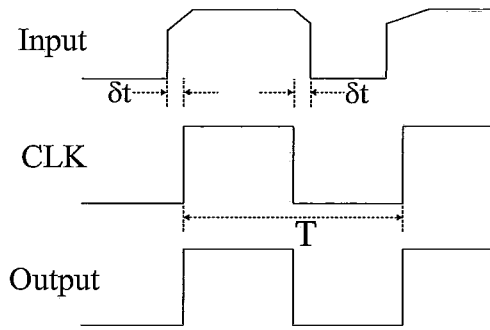


Figura 3.3: *Pulse-Stretching* para a Memória de Linha de Retardo Síncrona

permite certa tolerância a variações no retardo da linha ótica. Note que a modulação dos dados no sistema proposto é em banda base usando o código de linha unipolar RZ.

Retornando à figura 3.2, a leitura dos dados armazenados na memória ótica é feita através do *splitter* S . Este elemento simplesmente divide a potência ótica de entrada em duas. Uma parte vai para o processo de realimentação e a outra para o receptor de leitura.

Note que a capacidade de armazenamento deste tipo de memórias é proporcional ao comprimento da fibra e à taxa de transmissão utilizada. Mais especificamente, a capacidade de armazenamento da fibra em bits é calculada pela expressão:

$$(\text{comprimento_da_fibra} \times \text{taxa_de_transmissão}) \div \text{velocidade_da_luz}$$

onde a velocidade da luz é aproximadamente $2,1 \times 10^8$ m/s. Podemos então concluir, que existe uma relação entre o comprimento da linha de retardo ótica (e por conseqüência, a sua capacidade de armazenamento) e a latência de acesso à memória, já que, quanto maior a linha de retardo, maior será o tempo de espera (em média) para um dado passar por um determinado ponto da fibra.

Finalmente, para poder acessar os dados armazenados na memória através de circuitos eletrônicos operando a frequências inferiores às dos componentes totalmente óticos precisamos de mecanismos de sincronização e de uma interface de entrada/saída optoeletrônica. Estes mecanismos serão descritos a seguir.

Sincronização. Cada dado armazenado na memória de linha de retardo possui um endereço único como em qualquer memória tradicional. Mas, como o acesso aos dados na linha de retardo é seqüencial e depende do tempo que eles demoram para percorrer a fibra, um contador que indique o endereço do dado que está passando pela frente da

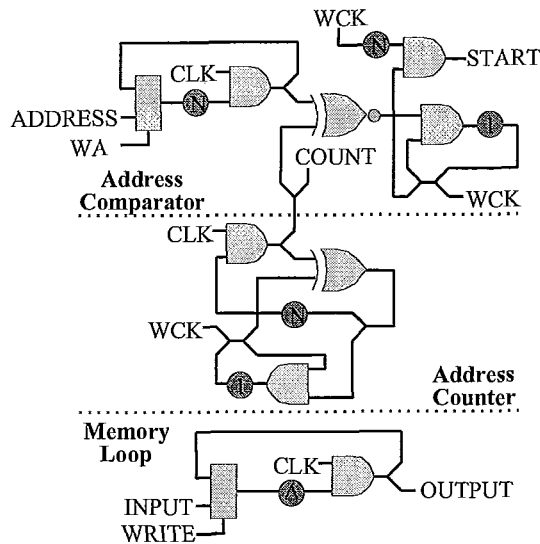


Figura 3.4: Memória Síncrona com Contador Ótico

interface de entrada/saída é indispensável para uma correta sincronização do acesso. No caso da memória de linha de retardo implementada com componentes totalmente óticos, as elevadas taxas de transmissão utilizadas criam a necessidade de que o contador também seja baseado nesta tecnologia.

Um contador totalmente ótico é apresentado na figura 3.4 [36]. Note que a estrutura do contador é completamente isolada da estrutura da memória ótica. Quando um dado vai ser lido ou escrito, o endereço do dado fornecido através da entrada *ADDRESS* é comparado com o endereço do dado que está a disposição nesse instante (*COUNT*). Quando estes dois endereços forem iguais, o sinal de início de operação (*START*) será ativado, permitindo o acesso ao dado correspondente.

No diagrama da figura 3.4, além do sinal de relógio *CLK*, é necessário um sinal *WCK* que indica o início de uma nova palavra. Este sinal é um pulso de relógio a cada N ciclos, onde N é o tamanho em bits de cada palavra. O sinal *WA* é ativado cada vez que um novo endereço precisa ser inserido no sistema. Finalmente, Δ representa o número de bits que podem ser armazenados na memória ótica.

Interfaces de Entrada/Saída. Devido às elevadas taxas de transmissão que podem ser atingidas através da utilização de componentes totalmente óticos, uma divisão muito clara deve existir entre as partes eletrônica e ótica. Para conseguir essa divisão e ao mesmo tempo permitir uma fácil interação entre as duas partes, uma interface de entrada/saída foi projetada em [35]. Esta interface, além de transformar os sinais eletrônicos em sinais

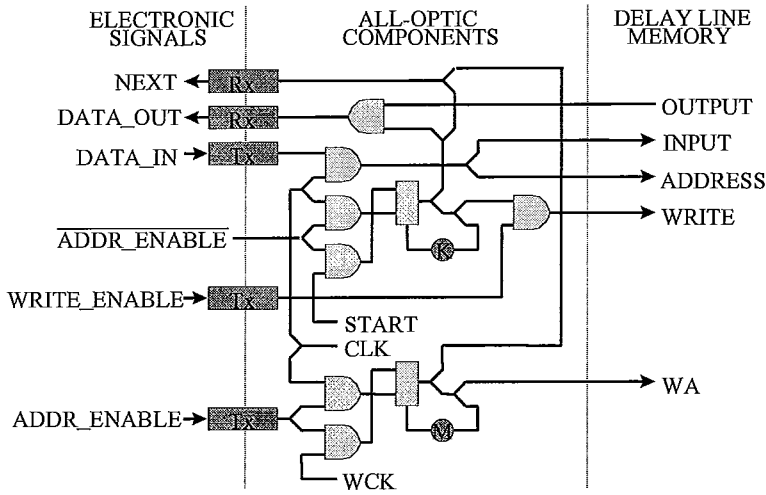


Figura 3.5: Interface Optoeletrônica da Memória Síncrona

óticos, e vice-versa, permite que a velocidade de operação da parte eletrônica seja inferior à velocidade de operação da parte ótica por praticamente qualquer fator. A figura 3.5 mostra o diagrama desta interface.

Quando o controlador eletrônico da memória deseja fazer uma leitura, ele ativa o sinal *ADDR_ENABLE* para inserir o endereço do dado através do pino *DATA_IN*. Após configurado o endereço, o sinal *ADDR_ENABLE* é desativado, e o dado é lido através do pino *DATA_OUT*. Em uma escrita, o processo de programação do endereço é o mesmo. A diferença está em que quando o sinal *ADDR_ENABLE* é desativado, o sinal *WRITE_ENABLE* é ativado para poder escrever os dados através do pino *DATA_IN*. Imediatamente após a escrita do dado, o sinal *WRITE_ENABLE* deve ser novamente desativado.

Todas as escritas e leituras através de *DATA_IN* e *DATA_OUT*, respectivamente, devem estar sincronizadas com o sinal *NEXT*. Este sinal indica que o bit presente em *DATA_IN* já foi lido pelos circuitos óticos, ou que existe um outro bit presente em *DATA_OUT* para ser lido pelos circuitos eletrônicos. Os sinais que ligam esta interface com a memória de linha de retardo são os mesmos que foram definidos para a figura 3.4.

A relação entre as velocidades de operação da parte eletrônica e da parte ótica é estabelecida pelos retardos K e M . Espera-se que esta relação possa ser da ordem de 1:20000 quando lógicas de controle totalmente óticas mais desenvolvidas forem utilizadas. Desta forma, relógios eletrônicos de 500 MHz poderiam se misturar com elementos óticos operando próximos dos 10 THz.

Como se pode observar na descrição acima, a lógica de controle de uma memória

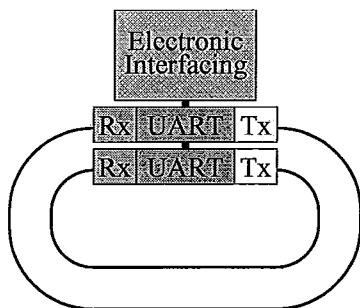


Figura 3.6: Memória de Linha de Retardo Assíncrona

de linha de retardo síncrona é bastante simples. A implementação desse mesmo tipo de memória apenas com componentes optoeletrônicos é ainda mais simples, já que a lógica de controle pode ser construída no domínio eletrônico, simplificando as tarefas de regeneração do sinal e de sincronização dos acessos, e eliminando a necessidade de uma interface de entrada/saída.

3.2.2 Memórias Assíncronas

Embora os componentes totalmente óticos sejam cada vez mais comuns, tarefas complexas ou com requerimentos específicos, como as utilizadas pelas memórias assíncronas, podem ser melhor realizadas no domínio eletrônico, onde os componentes têm se desenvolvido bastante e a tecnologia é suficientemente madura. Desta forma, na atualidade a única alternativa de implementação de memórias assíncronas é através de componentes optoeletrônicos.

No entanto, pelo fato de permitir que as tarefas de controle e regeneração do sinal possam ser realizadas no domínio eletrônico, a conversão do sinal do domínio eletrônico para o ótico, e vice-versa, é necessária antes e depois dos dados serem transmitidos pela linha de retardo. Isto faz com que as taxas de transmissão usadas na implementação da memória assíncrona sejam bastante baixas (na ordem dos Gbits/s) em relação à largura de faixa das fibras óticas. Técnicas de multiplexação devem ser então utilizadas para aumentar a capacidade de armazenamento da linha de retardo sem aumentar a latência de acesso aos dados. Exemplos destas técnicas de multiplexação são WDM e OTDM, como mostra o capítulo 2.

Para a implementação de uma linha de memória assíncrona usando técnicas de multiplexação, cada canal deve ter elementos eletrônicos de recepção/transmissão

assíncrona, similares aos atuais UARTs (*Universal Asynchronous Receiver-Transmitter*), além dos seus correspondentes receptores e transmissores optoeletrônicos (figura 3.6).

A capacidade de armazenamento deste tipo de memória é proporcional ao número de canais disponíveis, ao comprimento dos canais e à taxa de transmissão utilizada. Mais especificamente, a capacidade de armazenamento da fibra em bits é aproximadamente igual a:

$$(\text{número_canais} \times \text{comprimento_da_fibra} \times \text{taxa_de_transmissão}) \div \text{velocidade_da_luz}$$

O fato da capacidade de armazenamento não ser exatamente igual à expressão acima, é devido à perda de largura de faixa com a transmissão dos bits adicionais que determinam o início e final de cada bloco de dados. A perda da largura de faixa depende do tamanho de cada bloco e do número de bits utilizados para marcar o início e fim dos blocos. Mas, esta característica desvantajosa é compensada com a maior tolerância do sistema às variações que possa ter o retardo produzido pela fibra.

3.2.3 Problemas Principais

Tanto a alternativa da memória síncrona quanto da memória assíncrona apresentam alguns problemas de implementação que felizmente estão sendo resolvidos com o avanço da tecnologia ótica. Entre os principais problemas podemos citar a variação do comprimento da fibra com as variações de temperatura, erros de transmissão produzidos por falhas próprias dos sistemas de comunicação ótica, e o tardio desenvolvimento dos componentes totalmente óticos. A seguir discutimos esses problemas, mostrando algumas das suas possíveis soluções.

Compensação de Temperatura. A variação do comprimento da fibra (Δl) em função das variações de temperatura (ΔT) é determinado pela equação:

$$\Delta l = \alpha \times l_0 \times \Delta T$$

onde α é o coeficiente de dilatação térmica da fibra e l_0 é o seu comprimento inicial. O valor típico do coeficiente de dilatação térmica é $10^{-6} \text{ } ^\circ\text{C}^{-1}$. Devido a este fenômeno, uma memória de linha de retardo ótica que suporte uma variação de até $8 \text{ } ^\circ\text{C}$ não pode armazenar mais de 3800 bits, independente do comprimento da fibra ou da taxa de

transmissão ótica utilizados [4]. Esta limitação se deve a que as variações no tamanho da fibra impediriam a correta determinação do endereço do bit acessado. Para superar esta restrição foram sugeridas algumas técnicas, dentre as quais:

1. Troca de Materiais. Uma alternativa para eliminar os indesejáveis efeitos da variação de temperatura é usar materiais pouco sensíveis a este fator. Um exemplo são as fibras com coberturas especiais que diminuem a variação de comprimento na presença de mudanças de temperatura. Outra opção é a utilização de fibras com índices de refração inversamente proporcionais às mudanças de temperatura. O objetivo é compensar as mudanças no comprimento da fibra com um aumento proporcional no índice de refração. Isso muda a velocidade de propagação da luz na fibra e mantém o retardo constante.
2. Detecção do Deslocamento de Fase. Uma segunda alternativa para a compensação de temperatura é a detecção do deslocamento de fase. Esta técnica mede o desvio de fase do sinal que chega ao ponto de regeneração com respeito ao relógio do sistema. Para conseguir este objetivo existem duas variantes:
 - Enviar o sinal de relógio pelo meio de propagação, seja usando uma fibra paralela ou alguma técnica de multiplexação. Esse sinal será comparado ao final da linha com o relógio do sistema.
 - Derivar o sinal de relógio deslocado a partir do próprio sinal de dados mediante a utilização de um dispositivo DRF (*Dielectric Resonance Filter*), e posteriormente compará-lo com o relógio do sistema.

Uma vez detectado o deslocamento de fase, pode-se usar este valor para controlar a temperatura, ou melhor ainda, para controlar a taxa de transmissão dos dados.

3. Utilização de Múltiplos Laços. A utilização de múltiplos laços de fibra ou múltiplos canais multiplexados, além de diminuir a latência de acesso aos dados, diminui a dependência do sistema às variações de temperatura. Esta é uma alternativa interessante do ponto de vista de desempenho, e inclusive de projeto, mas é custosa. A quantidade da lógica de componentes totalmente óticos é, atualmente, um fator

chave na determinação dos custos do sistema. No entanto, espera-se que com os avanços na tecnologia de dispositivos óticos e com uma fabricação em grande escala, os custos diminuam, permitindo a construção de múltiplos laços de fibra para a implementação de uma grande memória de retardo.

Falhas Intermitentes. Existem outros fatores que intervêm em menor grau na geração de erros dentro da memória ótica [37]. Embora estes erros sejam classificados como intermitentes, merecem especial interesse devido às elevadas taxas de transmissão que se deseja atingir.

1. Absorção. Apesar da absorção ser muito baixa nos sistemas de comunicação ótica modernos (0,5 dB/Km), os outros componentes do sistema (portas lógicas, *splitters*, etc.) podem introduzir perdas significativas, produzindo uma incorreta regeneração do sinal. Experimentos demonstram que com a tecnologia atual se tem taxas de erro inferiores a 10^{-13} [38]. Estes valores deverão ser superados conforme a tecnologia de componentes totalmente óticos alcance a sua maturidade.
2. Dispersão. Atualmente, os sistemas óticos que usam a faixa de 1,3 micrômetros têm uma dispersão praticamente nula, permitindo a implementação de memórias bastante grandes e com perdas por dispersão insignificantes. Com o aparecimento de novas tecnologias de lasers, a situação das faixas de 0,85 e 1,5 micrômetros deve também melhorar consideravelmente.
3. Interferência. Este fenômeno tem a ver com a correlação existente entre as saídas de um mesmo componente ótico. Essa correlação, denominada também *cross-talk*, não apresenta maiores problemas nos sistemas atuais porque os componentes óticos existentes apresentam interferências inferiores a -20 dB [35]. Espera-se, no entanto, algumas melhoras para a próxima geração de componentes totalmente óticos.
4. Efeitos de Polarização. Certos comutadores óticos são sensíveis à polarização dos sinais de entrada. Devido à despolarização introduzida pela fibra, algumas perdas podem acontecer especialmente na fase de regeneração do sinal. As principais alternativas para solucionar este problema são a utilização de fibras com polarização

preservada, o uso de elementos pouco sensíveis à polarização, ou a utilização de elementos adicionais que polarizam novamente o sinal antes de chegar ao dispositivo ótico.

Tardio Desenvolvimento dos Componentes Totalmente Óticos. Devido ao estágio inicial de desenvolvimento em que se encontram os componentes totalmente óticos, apenas as lógicas de controle muito simples podem ser construídas com estes elementos. Por causa desta limitação, somente memórias de linha de retardo síncronas podem ser atualmente implementadas com componentes totalmente óticos. Espera-se que com o amadurecimento da tecnologia e a possível integração da lógica ótica, o número de alternativas para a implementação de circuitos digitais através de dispositivos totalmente óticos seja cada vez maior e ofereça uma relação custo/desempenho muito mais vantajosa que a existente atualmente. Exemplos dessa evolução já começam a aparecer até mesmo comercialmente. Um exemplo é a porta lógica ultra-rápida totalmente ótica Sagnac [39]. Essa porta permite operar a frequências de até 1,6 Tbits/s e foi utilizada na implementação de demultiplexadores e registradores de deslocamento óticos. Um outro exemplo é o dispositivo TOAD [22], mencionado no capítulo anterior.

Felizmente, o desenvolvimento da tecnologia ótica promete avanços ainda maiores nos próximos anos além de uma diminuição considerável no custo dos dispositivos óticos. Espera-se que dispositivos optoeletrônicos operando a velocidades superiores aos 10 Gbits/s, e podendo multiplexar centenas de canais em uma única fibra apareçam em menos de uma década. Da mesma forma, novas soluções para o fenômeno da dilatação térmica e para o acoplamento de dispositivos permitirão uma melhor recuperação, sincronização e regeneração do sinal, especialmente nos sistemas síncronos.

3.3 Utilização em Computação Paralela e Distribuída

De acordo com a tendência apresentada pela taxa de transmissão das comunicações óticas (figura 3.7), nada impede supor que nos próximos anos vão se atingir taxas de transmissão agregadas (usando WDM ou OTDM) próximas aos 10 Tbits/s [40] em uma única fibra. Com essa taxa de transmissão, e sabendo que a velocidade de propagação da luz nas fibras óticas é aproximadamente $2,1 \times 10^8$ m/s, 64 MBytes de informação podem ser

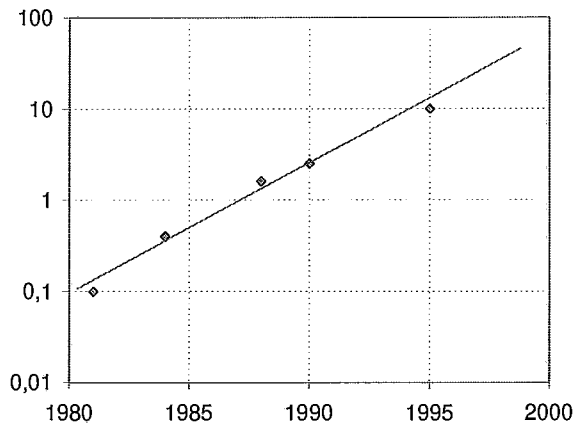


Figura 3.7: Evolução da Velocidade de Transmissão (em Gbits/s) nos Sistemas Óticos

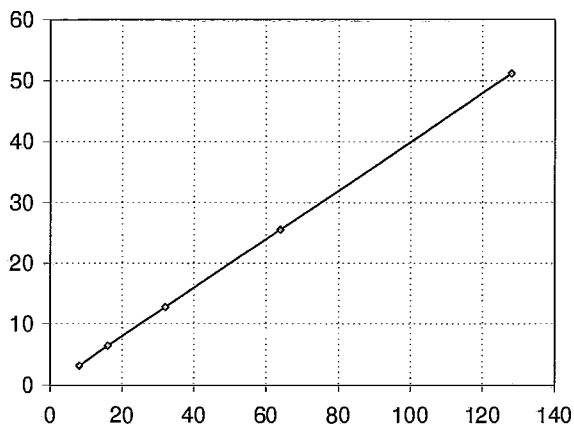


Figura 3.8: Latência Média de Acesso (em Microsegundos) em Função da Capacidade de Armazenamento (em MBytes)

armazenados em um laço de fibra de 10,24 quilômetros, com uma latência média de 25,6 microsegundos.

A figura 3.8 mostra a latência média para outras quantidades de memória supondo uma taxa de transmissão agregada de 10 Tbits/s. Da figura pode-se concluir que, nos sistemas seqüenciais de computação, as memórias de linha de retardo têm utilidade limitada, uma vez que somente podem armazenar pequenas quantidades de dados, devido às elevadas latências que as memórias maiores apresentam. Mas, quando se analisa o caso de um sistema paralelo ou distribuído, no qual a memória ótica deve interconectar todos os nós do sistema e permitir o acesso de todos esses nós aos dados armazenados na fibra, existem certos fatores que mudam esse cenário. Entre os fatores principais podemos mencionar:

- A memória ótica pode armazenar um conjunto de dados específico (e.g., variáveis de sincronização) ou agir como uma cache de todos os dados compartilhados, evi-

tando assim acessos a níveis inferiores do sistema de memória e, conseqüentemente, permitindo a diminuição da latência média de acesso à memória. Como resultado dessa diminuição, uma rede ótica com capacidade de armazenamento pode apresentar uma relação custo/desempenho melhor que a oferecida pelas redes de interconexão existentes.

- Os processadores podem ter acesso simultâneo aos dados armazenados na fibra sem gerar contenção. Além disso, a implementação de técnicas de tolerância à latência, como *prefetching*, não consome largura de faixa.
- O custo para manter a coerência dos dados armazenados na fibra é praticamente nulo. Além disso, as características de disseminação da fibra permitem a implementação eficiente de protocolos de coerência de cache baseados em atualizações e/ou primitivas de comunicação coletiva.
- A memória ótica pode reduzir qualquer problema de acesso não uniforme aos níveis mais baixos do sistema de memória.
- O sistema se adapta facilmente ao modelo de memória compartilhada, facilitando a programação tanto a nível do usuário como do sistema (o modelo de troca de mensagens também é suportado naturalmente).

No entanto, ainda existem várias questões para serem resolvidas e alguns desafios que precisam ser superados. Entre as principais questões que ainda precisam ser resolvidas, podemos mencionar as seguintes:

1. Que tipos de configurações para o cacheamento de dados apresentam uma melhor relação custo/desempenho? Cacheamento de dados da memória principal? Da memória secundária?
2. Quais são os protocolos de consistência, técnicas de cacheamento e tolerância à latência mais adequados para esses tipos de sistemas de cacheamento ótico?
3. Que classes de aplicações paralelas ou distribuídas se beneficiam mais do cacheamento ótico em cada nível do sistema de memória?

Estamos atacando todas estas questões na nossa Tese. Os próximos capítulos apresentam os nossos resultados preliminares nessa direção.

Capítulo 4

OPTNET

Este capítulo descreve OPTNET (*OPTimized OPTical NETwork*) [9], uma nova rede ótica com o seu próprio protocolo de coerência associado que explora algumas das características principais da comunicação ótica no projeto de multiprocessadores escaláveis. De acordo com a classificação apresentada no capítulo 2, esta é uma rede *single-hop* FT- $\{F,T\}R$. As três características principais que diferenciam OPTNET de outras redes óticas são: a) os seus canais de disseminação se comportam bem sob alta contenção, b) os seus canais para comunicação ponto-a-ponto não requerem nenhum mecanismo de controle de acesso, e c) atinge um ótimo desempenho de comunicação com um custo de *hardware* ótico baixo.

4.1 Fundamentos

O desempenho de um multiprocessador baseado em OPTNET vai ser comparado em relação a outros multiprocessadores baseados nas redes de interconexão ótica DMON e LambdaNet. Estas redes foram seleccionadas pelos seguintes motivos: DMON (*Decoupled Multichannel Optical Network*) é uma das poucas redes propostas especificamente para multiprocessadores, e quando acoplada com o protocolo de coerência I-SPEED, tem mostrado ser superior àqueles multiprocessadores baseados unicamente em *snooping* ou em diretórios. LambdaNet, por sua vez, introduz complexidade na busca de desempenho. Desta forma, esta rede pode-se converter em um limite superior de desempenho se combinada com protocolos de coerência eficientes. A seguir, estas redes serão descritas com mais detalhes.

4.1.1 DMON

Esta é uma rede WDM que foi proposta por Ha e Pinkston em [10]. A rede divide os seus $p + 2$ canais (onde p é o número de nós no sistema) em 2 grupos: o primeiro é usado para uma comunicação por disseminação, enquanto que o segundo é usado para uma comunicação ponto-a-ponto entre os nós. O primeiro grupo é formado por 2 canais compartilhados por todos os nós do sistema: o canal de controle e o canal de disseminação. Os outros p canais, chamados de *home-channels*, pertencem ao segundo grupo de canais.

O canal de controle é usado para controlar, de forma distribuída, todos os outros canais através de um esquema de reserva [3]. Um nó que quer transmitir por um dos canais deve primeiro esperar pelo seu turno de acesso ao canal de controle e então disseminar a sua intenção de transmissão. Essa disseminação faz com que os outros nós conheçam a comunicação que se realizará, evitando assim qualquer conflito. O canal de controle, por sua vez, é multiplexado usando o protocolo TDMA.

O canal de disseminação é usado para a comunicação de eventos globais como as operações de coerência e sincronização, enquanto que os *home-channels* são usados unicamente para operações de pedido e envio de blocos de memória. Cada nó pode transmitir em qualquer *home-channel*, mas só pode receber de um único *home-channel*. Cada nó atua como o *home* (o nó responsável por prover cópias atualizadas) de $1/p$ dos blocos de memória. Um nó recebe pedidos dos seus blocos através do seu próprio *home-channel*, e as respostas são enviadas pelo *home-channel* do solicitante.

Note que as transações de leitura e escrita seguem diferentes caminhos em DMON. Este desacoplamento dos recursos baseado no tipo de referência é uma das características principais de DMON. Embora este desacoplamento apresente benefícios que ajudam a reduzir a latência de acesso à memória, ele pode também produzir condições de corrida quando uma operação de coerência e uma leitura, correspondentes ao mesmo bloco, acontecem simultaneamente.

Como se observa na figura 4.1, a interface de rede (“NI”) da arquitetura DMON apresenta dois transmissores fixos (“Tx”), um para cada canal de disseminação¹, um transmissor sintonizável (“TTx”), para os *home-channels*, e três receptores fixos (“Rx”), dois para

¹Estes transmissores fixos não são parte da proposta original de DMON. Foram adicionados para evitar penalidades adicionais devido à constante resintonização do transmissor sintonizável.

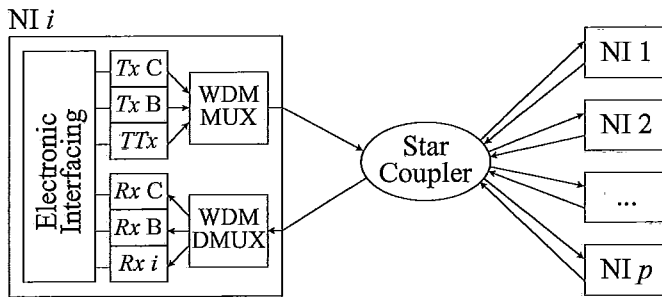


Figura 4.1: A Interface de Rede da Arquitetura DMON

os canais de disseminação e um para o *home-channel* do nó. O custo de *hardware* total da arquitetura DMON em termos do número de componentes óticos é $6 \times p$.

SPEED (*Snoopy Protocol Enhanced and Extended with Directory*) é um protocolo de coerência de caches de alto desempenho criado para explorar as características de comunicação de DMON. Na sua versão com invalidações (I-SPEED), a única descrita em [10], o protocolo define 4 estados para os blocos de memória: *clean*, *exclusive*, *shared* e *invalid*. O protocolo só permite que uma cópia do bloco esteja no estado *exclusive* ou *shared*. Um nó que possui na sua cache um bloco em um destes estados é o dono do bloco. A cópia de um bloco em estado *exclusive* ou *shared* é repassada ao solicitante como *clean*. O *home* de cada bloco de memória possui uma entrada de diretório que armazena o atual dono do bloco. Todas as falhas em um bloco de memória são enviadas ao seu *home* e, se necessário, repassadas ao nó dono do bloco.

I-SPEED também define estados para manipular condições de corrida. Uma condição de corrida é detectada quando uma operação de coerência é vista por um bloco que tem uma leitura pendente. I-SPEED trata esta condição forçando a invalidação da cópia do bloco (possivelmente inconsistente) após a leitura pendente ser completada. Mais detalhes acerca de I-SPEED podem ser encontrados em [10].

Um protocolo baseado em atualizações também foi proposto para DMON em [9]. O protocolo é bastante simples pois todas as escritas a dados compartilhados são enviadas aos seus correspondentes *homes*. Assim, uma falha na cache pode ser satisfeita imediatamente pelo *home*, eliminando a necessidade dos diretórios. Este protocolo também inclui suporte para manipular condições de corrida. Como em I-SPEED, uma condição de corrida é detectada quando uma operação de coerência é vista por um bloco que tem uma leitura pendente. Neste caso, as atualizações recebidas durante a operação de leitura

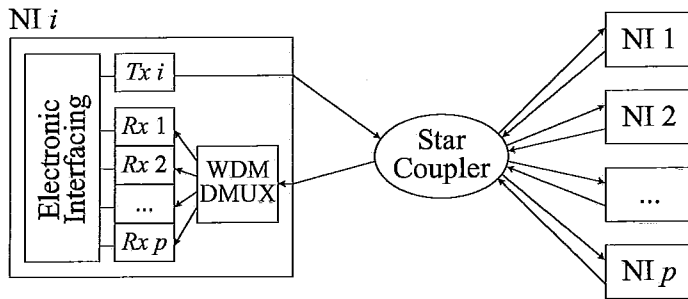


Figura 4.2: A Interface de Rede da Arquitetura LambdaNet

pendente são armazenadas e aplicadas ao bloco após terminada a operação de leitura. Devido a que um único canal de disseminação não é capaz de suportar o tráfego pesado de atualizações relativo a um conjunto grande de aplicações, a arquitetura básica de DMON foi estendida com um canal de disseminação extra para a transferência das atualizações. Um nó pode transmitir somente em um dos canais de coerência, o qual é determinado como uma função da identificação do nó, mas pode receber de ambos os canais. Com exceção deste canal extra (e os receptores associados), o *hardware* desta rede é o mesmo (figura 4.1). Assim, o custo de *hardware* total da versão modificada de DMON, em termos de número de componentes óticos, é $7 \times p$.

4.1.2 LambdaNet

A arquitetura LambdaNet foi proposta por Goodman *et al.* em [11]. A rede aloca um canal WDM para cada nó, permitindo que cada nó transmita a todos os outros nós sem necessidade de esquemas de arbitragem. Nesta organização cada nó usa um transmissor fixo (“*Tx*”) e p receptores fixos (“*Rx*”), como mostra a figura 4.2. Assim, cada nó recebe todo o tráfego da rede simultaneamente, e procede a selecioná-lo por intermédio de circuitos eletrônicos. Este esquema permite que os canais sejam usados para uma comunicação ponto-a-ponto ou por disseminação. O custo total de *hardware* de LambdaNet é então $p^2 + p$.

Diferentemente de DMON, LambdaNet não foi proposta com um protocolo de coerência associado. Mas, por motivos de comparação, pode ser considerado um multiprocessador com LambdaNet e com um protocolo de coerência de caches baseado em atualizações, onde as transações de escrita e sincronização são disseminados a todos os nós, enquanto que o tráfego de leitura usa uma comunicação ponto-a-ponto entre o soli-

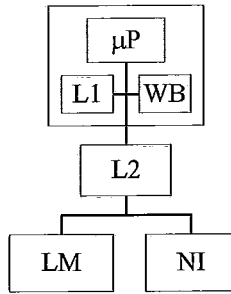


Figura 4.3: Detalhe da Arquitetura dos Nós

citante e o *home*. Como no protocolo proposto para DMON baseado em atualizações, os módulos de memória são mantidos atualizados todo o tempo, permitindo que os *homes* possam responder imediatamente aos pedidos de bloco resultantes de falhas nas caches.

Note que a arquitetura LambdaNet não é prática devido ao seu custo de *hardware*. Ela é incluída na maioria de estudos como uma base para a comparação dos outros esquemas. A combinação de LambdaNet e o protocolo de coerência sugerido representa um limite superior de desempenho para multiprocessadores, pois o protocolo de coerência baseado em atualizações evita falhas de coerência na cache, os canais de LambdaNet não requerem nenhum protocolo de acesso ao meio, e o seu *hardware* não requer a sintonização de transmissores ou receptores.

4.2 Arquitetura de OPTNET

Esta seção inicia descrevendo a arquitetura e o protocolo de coerência básico associado, para então descrever as extensões ao protocolo que permitem suportar múltiplos pedidos de leitura pendentes.

4.2.1 Arquitetura Básica

OPTNET supõe que cada nó no multiprocessador é extremamente simples. Na verdade, todos os componentes de *hardware* do nó são convencionais com exceção da interface de rede. Mais especificamente, cada nó inclui um processador (“P”), um *write-buffer* (“WB”), caches primárias (“L1”) e secundárias (“L2”), memória local (“LM”), e a interface de rede (“NI”) que liga o nó a OPTNET (figura 4.3).

A figura 4.4 mostra a arquitetura da rede. Como em DMON, OPTNET divide os canais de comunicação em dois grupos: o primeiro para o tráfego por disseminação e o

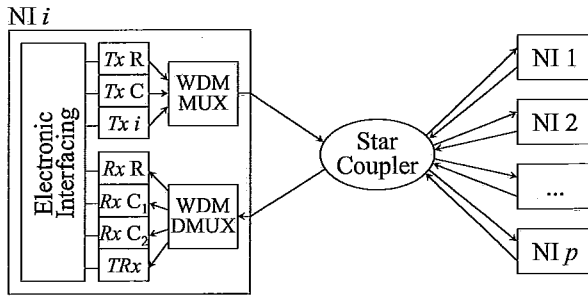


Figura 4.4: A Interface de Rede da Arquitetura OPTNET

segundo para uma comunicação ponto-a-ponto. Três canais, um canal de pedidos e dois canais de coerência, são destinados ao primeiro grupo, enquanto que p canais, chamados de *home-channels*, são destinados ao segundo.

O canal de pedidos é usado para solicitar blocos de memória. A resposta a tal pedido é enviada pelo *home* (o nó responsável por prover cópias atualizadas do bloco) através do seu correspondente *home-channel*. Os canais de coerência são utilizados para disseminar transações de coerência e sincronização. Como o canal de controle em DMON, o canal de pedidos usa TDMA para controlar o acesso ao meio. O acesso aos canais de coerência, por outro lado, é controlado usando TDMA com *slots* de tempo variável. À diferença de DMON, os *home-channels* não requerem nenhum tipo de arbitragem, pois só o *home* pode transmitir pelo seu canal.

Cada nó pode transmitir no canal de pedidos, em um dos canais de coerência (determinado como uma função da identificação do nó), e no seu *home-channel*, mas pode receber de qualquer um dos canais por disseminação ou de qualquer um dos *home-channels*. Desta forma, cada nó requer 3 transmissores fixos (“Tx”) (um para o canal de pedidos, um para o *home-channel*, e o último para um dos canais de coerência), 3 receptores fixos (“Rx”) (para os canais por disseminação), e um receptor sintonizável (“TRx”) (para os *home-channels*). O custo de *hardware* de OPTNET é então $7 \times p$ componentes óticos.

4.2.2 Protocolo de Coerência Básico

Com a finalidade de explorar na sua totalidade os benefícios potenciais de OPTNET, o protocolo de coerência do multiprocessador deve ser ajustado à rede. Assim, o protocolo de coerência proposto para OPTNET é baseado em atualizações e suportado tanto por comunicações ponto-a-ponto como por disseminação. O tráfego de atualizações flui

através dos canais de coerência, enquanto que os blocos de memória são enviados por intermédio dos *home-channels*. O canal de pedidos transporta todos os pedidos de leitura. A descrição a seguir mostra em detalhes o protocolo de coerência em termos das ações tomadas nos acessos de leitura e escrita.

Leituras. Em um acesso de leitura, a hierarquia de memória é atravessada de forma a permitir que a palavra requerida seja encontrada o mais rápido possível. Assim, os conteúdos das caches primárias e secundárias são verificados, como em qualquer outro sistema de computação com múltiplas caches. Uma falha nas caches é tratada diferentemente dependendo do tipo de dado a ser lido. No caso do bloco ser privado ou mapeado na memória local, o acesso é tratado pela memória local, a mesma que retorna o bloco diretamente ao processador. Se o bloco é compartilhado e está mapeado em um outro nó, o pedido é enviado ao seu *home* através do canal de pedidos e o receptor sintonizável é ajustado ao *home-channel* do respectivo *home*. Quando o pedido chega ao *home*, ele lê o bloco e o retorna por intermédio do *home-channel*. O nó solicitante espera pelo bloco a ser recebido, o retira da interface de rede, e o entrega ao sistema de caches.

Escritas. A arquitetura de multiprocessador baseado em OPTNET implementa o modelo de consistência de memória *release-consistency* [41]. Escritas consecutivas para um mesmo bloco de cache são agrupadas no *write-buffer*. O conjunto de escritas a um bloco privado é enviado diretamente à memória local através do sistema de caches. Conjuntos de escritas a blocos compartilhados são sempre enviadas a um dos canais de coerência na forma de uma atualização, novamente através do sistema de caches. Uma atualização somente contém as palavras que foram modificadas em cada bloco.

Cada atualização deve ser reconhecida pelo *home* correspondente antes que outra atualização com o mesmo *home* seja emitida, desta forma os módulos de memória não requerem filas de entrada extremamente grandes (i.e., os reconhecimentos são usados para controle de fluxo). Os outros nós que possuem na sua cache o bloco para o qual está sendo enviada a atualização, simplesmente atualizam as suas caches locais. Quando o *home* recebe a atualização, ele a insere na fila de memória e envia o reconhecimento através do canal de pedidos. No entanto, o reconhecimento pode não ser enviado imediatamente se a fila de memória estiver preenchida até um determinado ponto. Neste caso, o *home*

adia a transferência do reconhecimento até que ele possa permitir a emissão de uma outra atualização proveniente do mesmo nó. Um nó só pode adquirir um *lock* ou passar por uma barreira após esvaziar a sua fila de memória. Normalmente as mensagens de reconhecimento não sobrecarregam o canal de pedidos devido a serem mensagens curtas, que cabem em um único *slot* do canal de pedidos.

Finalmente, o protocolo de coerência trata as condições de corrida que resultam do desacoplamento das transações de leitura e escrita, armazenando as atualizações para combiná-las posteriormente com o bloco recebido de memória.

4.2.3 Suportando Múltiplos Pedidos de Leitura Pendentes

O protocolo de coerência básico de OPTNET não suporta múltiplos pedidos de leitura pendentes. Esta limitação resulta do fato de que a rede possui um único receptor sintonizável que deve ser ajustado a um único *home-channel* durante um acesso de leitura. No entanto, facilitar múltiplos pedidos de leitura pendentes é importante se o multiprocessador baseado em OPTNET explora técnicas de tolerância à latência ou usa processadores superescalares. Assim, a seguir será descrita uma simples extensão ao protocolo de coerência de OPTNET que permite o suporte de múltiplos pedidos de leitura pendentes.

A extensão só afeta os pedidos de leitura que são emitidos enquanto outros pedidos estão pendentes. Nesta situação, uma seqüência de pedido/resposta é transformada em um par de seqüências pedido/resposta. Essas seqüências serão chamadas de *read-and-buffer/block-buffered* e *transfer-block/block-reply*. O pedido *read-and-buffer* é enviado ao *home* imediatamente após uma falha de leitura na cache secundária. Quando o pedido chega à interface OPTNET do *home*, ela lê o bloco e o armazena em uma memória interna. Após armazenado o bloco, a interface do *home* envia a resposta *block-buffered* ao nó que solicitou a leitura. Depois de receber a mensagem *block-buffered* do *home*, a interface solicitante inclui o correspondente número do bloco em uma fila FIFO. Quando o bloco atinge a cabeça da fila, esta interface envia o pedido *transfer-block* ao *home*, sintoniza o *home-channel* adequado e espera pelo bloco chegar. Após receber a mensagem *transfer-block*, a interface do *home* responde com o bloco de memória e libera o espaço ocupado por ele. Quando o bloco de memória é recebido pela interface solicitante, o número de bloco correspondente é jogado fora da fila FIFO. As mensagens *read-and-buffer*, *block-*

buffered e *transfer-block* são enviadas através do canal de pedidos normal. A transferência do bloco de memória é feita através dos *home-channels* como no protocolo básico de OPTNET.

Esta extensão ao protocolo não deve impactar no desempenho notavelmente. Na verdade, quando não existem outros pedidos de leitura pendentes para esse nó, a seqüência pedido/resposta acontece como no sistema básico. Quando existem outros pedidos de leitura pendentes, o envio de uma seqüência extra de pedido/resposta nem sempre afeta o *overhead* de acesso aos dados, pois uma falha de leitura não está necessariamente no caminho crítico da computação. Além disso, o custo de enviar as mensagens extras é relativamente pequeno (19 ciclos de processador em média para cada mensagem), e podem ser enviadas em paralelo com a sintonização do receptor e o acesso à memória.

As únicas duas questões que permanecem são: quanto tráfego adicional será gerado pelas mensagens adicionais e se um único canal pode tratar todo esse tráfego. Os experimentos mostram que o aumento máximo do tráfego pode variar entre 18 e 198%, com uma média de 116%. Embora esses aumentos sejam significantes, eles não degradam o desempenho, já que o canal de pedidos é extremamente subutilizado. Os experimentos que serão apresentados mais à frente mostram que a porcentagem de *slots* livres no canal de pedidos varia de 88 a 98%, com uma média de 91%. Adicionalmente, a contenção para os *slots* no canal de pedidos é bastante baixa, somente 13% dos pedidos ou reconhecimentos competem por acesso ao canal.

A implementação destas modificações não tem custo de *hardware* ótico, mas existe um custo eletrônico bastante baixo: uma pequena quantidade de memória extra (DRAM) em cada interface OPTNET. Cada interface deve incluir $o \times \text{tamanho_do_endereço}$ Bytes para manter a fila de pedidos pendentes, onde o é o máximo número de pedidos pendentes por nó e o tamanho do endereço de um bloco é 4. Além disso, para simplificar o gerenciamento, a quantidade de memória para armazenar os blocos de memória deve ser $p \times o \times (b + \text{tamanho_do_endereço})$, onde p é o número de nós no multiprocessador e b é o tamanho dos blocos de memória. Cada grupo de $o \times (b + \text{tamanho_do_endereço})$ Bytes deve ser alocado aos blocos pedidos por um nó diferente. Assim, 4368 Bytes por interface são suficientes para permitir 4 pedidos pendentes em um sistema de 16 nós com blocos de memória de 64 Bytes.

Finalmente, é interessante notar que, à diferença de DMON e LambdaNet, o suporte para múltiplos pedidos de leitura pendentes de OPTNET não requer *hardware* ótico extra nem produz gargalos no desempenho do sistema.

4.3 Metodologia

Para avaliar o desempenho de OPTNET e compará-lo em relação a propostas de multiprocessadores baseados em redes óticas previamente estudadas, utilizamos simulações de aplicações paralelas reais. Como o simulador não implementa processadores superescalares, as simulações correspondem à proposta básica de OPTNET.

4.3.1 Simulação

Simulamos multiprocessadores de 16 nós com as redes de interconexão OPTNET, DMON e LambdaNet. Os simuladores são baseados no MINT [42]. Cada nó das máquinas simuladas contém um processador de 200 MHz, um *write-buffer* de 16 entradas, uma cache primária de 4 KBytes diretamente mapeada e com blocos de 32 Bytes, uma cache secundária de 16 KBytes diretamente mapeada e com blocos de 64 Bytes, memória local, e uma interface de rede. Note que propositalmente simulamos caches pequenas, já que as limitações no tempo de simulação nos impedem de usar entradas com tamanho real. Na verdade, a capacidade das caches primária e secundária foram reduzidas (em relação a sistemas reais) por aproximadamente um fator de 32. O objetivo destas reduções é produzir um tráfego de acessos à memória similar ao dos sistemas reais.

Os dados compartilhados são intercalados entre as memórias a nível de bloco. Foi assumido que todas as instruções e acertos de leitura na cache primária demoram um ciclo de processador. Falhas de leitura na cache primária bloqueiam o processador até que o pedido de leitura seja satisfeito. Um acerto de leitura na cache secundária demora 12 ciclos para completar. As escritas entram no *write-buffer* e demoram um ciclo, exceto quando o *write-buffer* está cheio. Neste último caso, o processador é bloqueado até que uma entrada seja liberada. As leituras podem ser escalonadas antes das escritas enfileiradas no *write-buffer*. Um módulo de memória pode prover as primeiras duas palavras 12 ciclos após a emissão do pedido. As outras palavras são entregues a uma taxa de duas palavras

Operação	Latência		
	OPTNET	LambdaNet	DMON
Falha na cache secundária			
1. Verificação da cache primária	1	1	1
2. Verificação da cache secundária	4	4	4
3. Retardo médio TDMA	16	—	16
4. Reserva do canal	—	—	2*
5. Retardo de sintonização	—	—	4
6. Pedido à memória	2*	2*	3
7. Retardo de propagação	1	1	1
8. Leitura de memória	44 ⁺	44 ⁺	44 ⁺
9. Retardo médio TDMA	—	—	16
10. Reserva do canal	—	—	2*
11. Transferência de bloco	22	22*	23
12. Retardo de propagação	1	1	1
13. Transferência da NI para a cache sec.	16	16	16
Total	107	91	133

Tabela 4.1: Tempos de Leitura para OPTNET, LambdaNet e DMON

por cada 4 ciclos do processador. A contenção nas memórias e na rede é completamente modelada.

No protocolo de coerência baseado em atualizações, unicamente a cache secundária é atualizada quando uma atualização chega ao nó. A cópia do bloco na cache primária é invalidada. Além disso, para reduzir o tráfego de escritas, o *write-buffer* agrupa as escritas para um mesmo bloco em todos os sistemas simulados. Uma atualização somente contém as palavras que foram modificadas em cada bloco. As implementações dos protocolos assumem um modelo de memória *release-consistency* [41].

A taxa de transmissão ótica foi estabelecida em 5 Gbits/s, o que produz as latências listadas na tabela 4.1 para uma falha de leitura. As latências de uma transação de coerência em OPTNET, LambdaNet, DMON com coerência baseada em atualizações (DMON-U), e DMON com I-SPEED (DMON-I) são mostradas na tabela 4.2, assumindo que 8 palavras foram escritas em cada bloco de cache². Todos os números nas tabelas estão em ciclos de processador e assumem um cenário livre de contenção nos canais de comunicação e na memória. Os valores marcados com '*' e '+' são os que podem ser aumentados

²Note que a maior parte das transações de coerência são normalmente realizadas fora do caminho crítico do processador pelo *write-buffer*.

Operação	Latência (em ciclos)			
	OPTNET	LambdaNet	DMON-U	DMON-I
1. Verificação da cache secundária	4	4	4	4
2. Escrita na NI	10	10	10	2
3. Retardo médio TDMA	8*	—	16	16
4. Reserva do canal	—	—	2*	2*
5. Atualização/Invalidação	15	13	14	3
6. Retardo de propagação	1	1	1	1
7. Retardo médio TDMA	16	—	16	16
8. Reserva do canal	—	—	2*	2*
9. Reconhecimento	2*	2*	2	2
10. Retardo de propagação	1	1	1	1
11. Escrita na cache secundária	—	—	—	8
Total	57	31	68	57

Tabela 4.2: Tempos para uma Transação de Coerência em OPTNET, LambdaNet, DMON-U e DMON-I

pela contenção/serialização na rede e memória, respectivamente. As latências totais de uma falha de leitura na cache secundária (tabela 4.1) mostra que LambdaNet possui 18% menos *overhead* que OPTNET nestas operações, pelo menos na ausência de qualquer tipo de contenção. Sob as mesmas condições, OPTNET possui 24% menos *overhead* que DMON. As latências totais das transações de coerência (tabela 4.2) mostram que LambdaNet possui 46% menos *overhead* que OPTNET e DMON-I nestas operações, pelo menos na ausência de contenção e assumindo 8 palavras escritas por bloco. Sob as mesmas condições, OPTNET e DMON-I apresentam 19% menos *overhead* que DMON-U.

Note que, nestas simulações, a duração mínima de um *slot* TDMA é 2 ciclos de processador, tanto para DMON como para OPTNET. Assim, cada *slot* do canal de controle em DMON e do canal de pedidos em OPTNET é de 2 ciclos. Cada *slot* dos canais de coerência em OPTNET é de pelo menos 2 ciclos, neste caso, a duração real de cada *slot* depende do número de palavras atualizadas.

Os parâmetros de simulação assumidos representam uma percepção pessoal do que é razoável para os multiprocessadores atuais e em um futuro próximo. O estudo do espaço de parâmetros apresentado na seção de resultados permitirá investigar o impacto das mais importantes hipóteses arquiteturais realizadas.

Programa	Descrição	Tamanho da Entrada
CG	Gradiente Conjugado (kernel)	1400 × 1400 doubles, 78148 ≠ 0
Em3d	Propagação de ondas eletromagnéticas	8 K nós, 5% remotos, 10 iter.
Gauss	Eliminação de Gauss sem blocos	256 × 256 floats
Mg	Poisson 3D usando técnicas de multigrid	24 × 24 × 64 floats, 6 iterações
Ocean	Simulação da movimentação de oceanos	Grade 66 × 66
Radix	Ordenação de inteiros	512 K chaves, radix 1024
Raytrace	Traçador paralelo de raios luminosos	teapot
SOR	Relaxamento progressivo	256 × 256 floats, 100 iterações
Water	Simulação de moléculas de água (spatial)	512 moléculas, 4 passos
WF	Algoritmo do caminho mais curto	384 vértices, conectividade 50%

Tabela 4.3: Descrição das Aplicações e Principais Parâmetros de Entrada

4.3.2 Aplicações

O conjunto de aplicações consiste de 10 programas: CG, Em3d, Gauss, Mg, Ocean, Radix, Raytrace, SOR, Water e WF. A tabela 4.3 mostra as aplicações e os seus parâmetros de entrada. Ocean, Radix, Raytrace e Water são do SPLASH-2 e foram amplamente descritas em outros trabalhos [43]. CG e Mg são implementações paralelas do conjunto de aplicações NAS, as mesmas que são descritas com mais detalhes em [44]. Em3d é da Universidade de Berkeley [45] e simula a propagação de ondas eletromagnéticas através de objetos em 3D. Gauss, SOR e WF foram desenvolvidas na Universidade de Rochester. Gauss realiza uma eliminação de Gauss sem agrupamento em blocos. SOR realiza o relaxamento sucessivo de uma matriz de elementos. WF é uma versão paralela do algoritmo de *Warshall-Floyd* para calcular o caminho mais curto entre todos os pares de nós de um grafo representado por uma matriz adjacente.

4.4 Resultados

Nesta seção é avaliado o desempenho de um multiprocessador baseado em OPTNET em relação a sistemas baseados em LambdaNet e DMON. Primeiro são mostrados os resultados de tempo de execução, para posteriormente analisar detalhadamente o desempenho das leituras e das escritas em cada sistema. Finalmente, é estudado o efeito de alguns dos parâmetros de simulação.

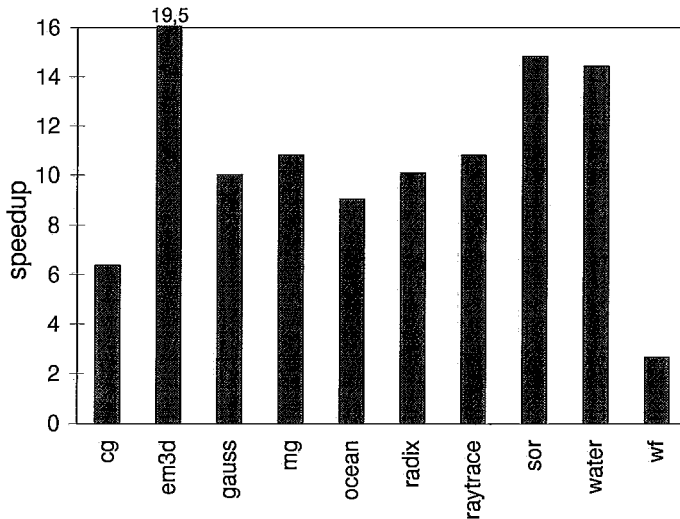


Figura 4.5: Ganho de Desempenho em um Multiprocessador de 16 Nós com OPTNET

4.4.1 Desempenho Geral

A figura 4.5 mostra o ganho de desempenho das aplicações executando em um multiprocessador baseado em OPTNET com 16 nós. A figura demonstra que, exceto para CG e WF, as aplicações exibem ganhos bons para 16 nós. Em3d, SOR e Water, em particular, atingem valores excelentes, acima de 14. Os dois extremos no ganho de desempenho, Em3d e WF, requerem uma explicação mais detalhada. Em3d apresenta um ganho super-linear como resultado das elevadas taxas de falha nas caches primárias e secundárias quando executada em um único nó. As caches não são efetivas para esta aplicação em um único nó. WF mostra um ganho baixo com 16 nós devido às grandes penalidades das barreiras como produto da falta de balanceamento de carga significativa existente nessa aplicação.

A figura 4.6 mostra os tempos de execução das 10 aplicações em um multiprocessador de 16 nós. Para cada aplicação são mostrados, da esquerda para a direita, os desempenhos de OPTNET, LambdaNet, DMON-U e DMON-I, normalizados aos resultados de OPTNET. Esta figura demonstra que o desempenho de DMON-U é igual ou melhor que o de DMON-I para todas as aplicações, exceto Water. As diferenças entre estes dois sistemas são em média 11%, sendo mais significativas para Em3d (16%), Gauss (16%), Ocean (43%) e Radix (14%).

Como era esperado, uma comparação entre LambdaNet e DMON-U é sempre favorável ao primeiro sistema. O ganho de desempenho de LambdaNet é em média 19%.

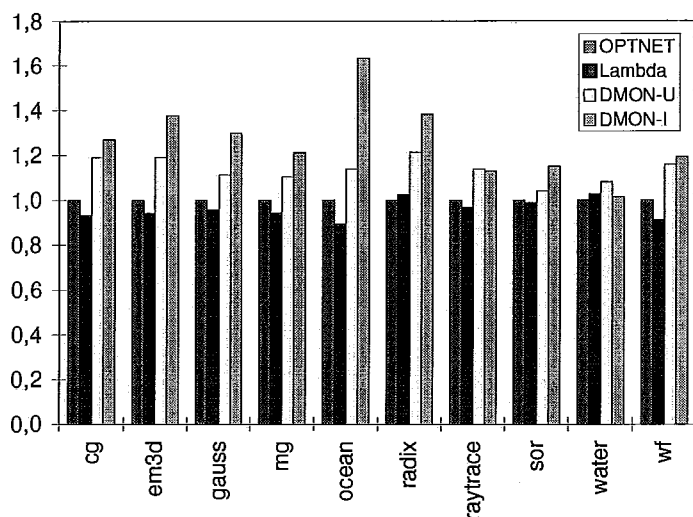


Figura 4.6: Tempos de Execução (com Relação ao MP OPTNET) de OPTNET, LambdaNet, DMON-U e DMON-I

SOR e Water apresentam um ganho pequeno em LambdaNet. Para as outras aplicações, as diferenças de desempenho variam de 16% para Gauss a 28% para CG, com uma média de 22%. A razão principal para este resultado é que a latência de uma falha de leitura na cache secundária em DMON-U é muito maior que em LambdaNet, especialmente quando os canais de DMON-U estão sujeitos à contenção.

Uma comparação entre OPTNET e DMON-U é claramente favorável a OPTNET em todos os casos com exceção de SOR e Water onde os desempenhos são similares. Para as outras 8 aplicações, o ganho de OPTNET varia de 10% para Mg a 21% para Radix, com uma média de 16%. Levando em conta todas as aplicações, a vantagem de OPTNET é em média 14%. A principal explicação para esta disparidade de desempenho é que a latência de uma falha de leitura na cache secundária em DMON-U é notavelmente maior que em OPTNET, especialmente quando DMON-U está sujeito à contenção na rede.

A figura 4.6 demonstra que os multiprocessadores baseados em OPTNET e LambdaNet são essencialmente equivalentes para 4 aplicações: Radix, Raytrace, SOR e Water. Para as outras 6 aplicações, os ganhos de desempenho de LambdaNet nunca são maiores que 12% e são em média 8%. Levando em conta todas as aplicações, o ganho de LambdaNet é em média 4%. Devido a Lambdanet requerer $O(p^2)$ hardware ótico, um fator de p vezes mais hardware que OPTNET, os resultados obtidos são excelentes em favor de OPTNET.

A explicação das diferenças de desempenho apresentadas está no custo médio das

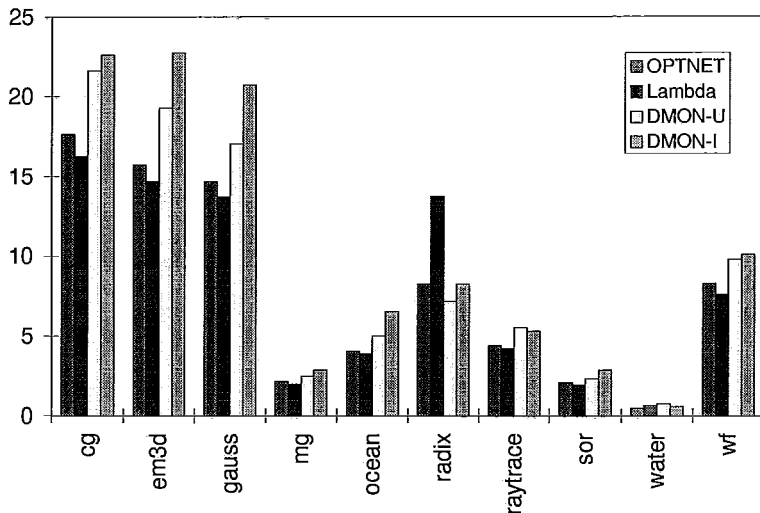


Figura 4.7: Latência Média das Leituras em Ciclos de Processador

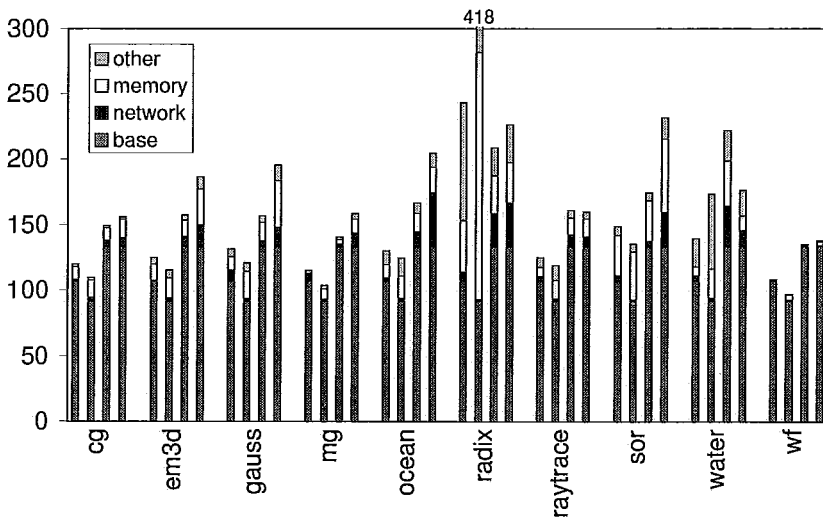


Figura 4.8: Latência Média de uma Falha de Leitura (em Ciclos de Processador) para OPTNET, LambdaNet, DMON-U e DMON-I

leituras e das escritas nos vários sistemas para cada aplicação. Assim, a seguir será feito um estudo destas operações para todos os sistemas e aplicações.

4.4.2 Desempenho das Leituras

As figuras 4.7 e 4.8 estão concentradas no desempenho das operações de leitura dos sistemas que estão sendo avaliados. A figura 4.7 apresenta a latência média das operações de leitura, enquanto que a figura 4.8 mostra a latência média de uma falha de leitura na cache secundária. Todas as latências são em ciclos de processador. A figura 4.8 divide as latências no seu componente livre de contenção (“base”) e nos retardos causados pela contenção na rede (“network”), nas memórias (“memory”), e no acesso aos pinos do pro-

cessador e barramento de memória (“other”). Nas duas figuras, as barras correspondem a, da esquerda para a direita, OPTNET, LambdaNet, DMON-U e DMON-I.

As figura 4.7 mostra que a latência de leitura média obtida pelos 3 sistemas baseados em atualizações (OPTNET, LambdaNet e DMON-U) é menor que a de DMON-I, exceto nos casos de Radix e Water. Este resultado pode ser explicado em parte pelo fato de que os sistemas baseados em atualizações exibem uma menor taxa de falhas na cache secundária que DMON-I. No entanto, as diferenças nas taxas de falha não são extremamente significativas, pois as aplicações simuladas são dominadas por falhas de substituição. Como é mostrado na figura 4.8, o fator mais importante nesta comparação é que as falhas de leitura demoram mais para serem satisfeitas nos sistemas baseados em DMON que em OPTNET e LambdaNet, mesmo na ausência de contenção. Adicionalmente, DMON-I sofre mais com a contenção em memória e na rede que os outros sistemas. Por exemplo, descartando os resultados de Radix e Water, DMON-I apresenta latências para uma falha de leitura na cache secundária 42% maiores que as de OPTNET em média, enquanto que em um cenário livre de contenção a diferença é só de 24%. A contenção na rede e em memória são mais acentuadas em DMON-I devido às escritas dos blocos de cache modificados, à leitura de diretórios requerida em todos os pedidos de memória, e às mensagens extras necessárias para conduzir um pedido ao atual dono do bloco.

Entre os sistemas baseados em atualizações, LambdaNet apresenta a menor latência de leitura, enquanto que DMON-U apresenta a maior. A latência de leitura média de OPTNET está entre estes dois extremos. Descartando os resultados de Radix e Water, a latência de leitura no sistema LambdaNet é somente 7% menor em média que a de OPTNET, enquanto que as leituras no sistema DMON-U são 20% mais custosas em média que em OPTNET.

Como se pode observar na figura 4.8, o sistema baseado em LambdaNet é usualmente mais propenso à contenção que os sistemas baseados em OPTNET e DMON-U, devido a duas características do primeiro sistema: a) as transações de leitura e escrita não são desacopladas, e b) a ausência de pontos de serialização para as atualizações dos diferentes nós gera um enorme tráfego de atualizações. Como resultado destas características, quando uma aplicação apresenta uma excessiva quantidade de atualizações (Radix e Water são os casos extremos), as transações de leitura são retardadas, pois leituras e escritas

Programa	OPTNET		LambdaNet		DMON-U		DMON-I	
	Stall	Flush	Stall	Flush	Stall	Flush	Stall	Flush
CG	0,1	0,5	0,0	0,1	0,0	0,4	0,1	2,2
Em3d	0,0	0,2	0,0	0,4	0,0	0,2	0,0	0,5
Gauss	0,1	0,6	0,0	0,4	0,1	0,6	0,2	0,8
Mg	0,5	0,2	0,0	0,3	0,7	0,2	2,2	0,6
Ocean	1,8	3,4	0,1	1,1	1,4	3,4	2,2	11,3
Radix	23,5	0,1	14,6	0,1	38,6	0,1	43,7	0,1
Raytrace	0,0	0,1	0,0	0,0	0,0	0,1	0,0	0,1
SOR	0,2	0,4	0,0	0,2	0,2	0,5	0,3	1,4
Water	1,1	0,0	0,0	0,0	0,2	0,0	0,0	0,0
WF	0,0	0,0	0,0	0,0	0,4	0,0	0,0	0,0

Tabela 4.4: Porcentagens de *Write-Stall* e *Write-buffer Flush* para OPTNET, LambdaNet, DMON-U e DMON-I.

competem pelos mesmos recursos de comunicação, cache e memória principal. Por outro lado, a degradação de desempenho gerada pela contenção não é suficiente para diminuir a contribuição das ótimas latências de LambdaNet sem contenção. Por exemplo, descartando os resultados de Radix e Water, LambdaNet apresenta, em média, uma latência para uma falha de leitura menor que a de OPTNET em só 8%, enquanto que as latências livres de contenção diferem 15%.

A contenção afeta os sistemas baseados em DMON-U e OPTNET em forma similar. Nos dois multiprocessadores, as latências de uma falha na cache secundária livre de contenção e total diferem em 24% na média. Como uma falha na cache secundária em um cenário livre de contenção demora mais tempo para ser satisfeita em DMON-U, este sistema apresenta um comportamento das leituras pior que OPTNET.

Em resumo, estes resultados são favoráveis a OPTNET para todas as aplicações, levando em conta a complexidade do sistema baseado em LambdaNet. Mesmo nos casos de Radix e Water, onde o comportamento é um tanto diferente das outras aplicações devido ao seu elevado tráfego de coerência, OPTNET mostra um bom desempenho para as leituras.

4.4.3 Desempenho das Escritas

Tendo discutido o desempenho das operações de leitura em cada um dos sistemas que estão sendo avaliados, resta estudar o desempenho das operações de escrita. A tabela

4.4 apresenta os atrasos de escrita e de esvaziamento do *write-buffer* como porcentagem do tempo total de execução de cada uma das aplicações executando nos diferentes sistemas. A tabela mostra que, exceto para Radix, as latências das operações de escrita são desprezíveis em todos os sistemas, demonstrando que um *write-buffer* de 16 entradas é normalmente suficiente para esconder o custo das operações de coerência. Em Radix as escritas são muito frequentes (aproximadamente uma escrita a cada 5 ciclos) e não podem ser agrupadas pelos *write-buffers*, causando frequentemente paradas na execução do programa. Adicionalmente, a tabela mostra que o tempo gasto com o esvaziamento dos *write-buffers* é desprezível como uma porcentagem do tempo de execução, mesmo no caso de Radix. A única exceção é Ocean executando em DMON-I, onde o tempo gasto com o esvaziamento do *write-buffer* representa 11,3% do tempo total.

Estes resultados sugerem que o *overhead* das operações de coerência não é um sério problema de desempenho na maioria dos casos, mesmo para os sistemas baseados em atualizações que forcem o sistema de comunicação com um elevado número de atualizações. No entanto, isto acontece somente porque estes sistemas incluem múltiplos canais de disseminação para as operações de coerência. O aumento do número de canais de coerência tem um significativo impacto no retardo de acesso ao meio e na serialização imposta pelas transações de coerência dos diferentes nós. Como um exemplo deste impacto, considere um sistema com 16 nós e um único canal de coerência TDMA. Nesse sistema, um nó será retardado em média 8 *slots* TDMA antes de obter acesso ao canal de coerência. Além disso, só uma transação de coerência pode ser iniciada em cada *slot*. Por outro lado, com dois canais de coerência, o mesmo nó seria somente retardado em média 4 *slots* TDMA para iniciar uma transação de coerência. Adicionalmente, duas transações de coerência podem ser iniciadas em paralelo a cada *slot* TDMA.

Para quantificar este efeito no caso de OPTNET, considere as figuras 4.9 e 4.10. As figuras mostram o tempo de execução de cada aplicação em um sistema OPTNET com 16 e 32 nós, respectivamente, assumindo 1, 2 e 4 canais de coerência. As barras na figura são divididas em tempo de processador (“busy”) e tempo gasto com operações de leitura (“read”), escrita (“write”) e sincronização (“sync”). Todos os resultados são normalizados para os resultados de um único canal.

Três observações principais podem ser feitas a partir destas figuras. A primeira é que,

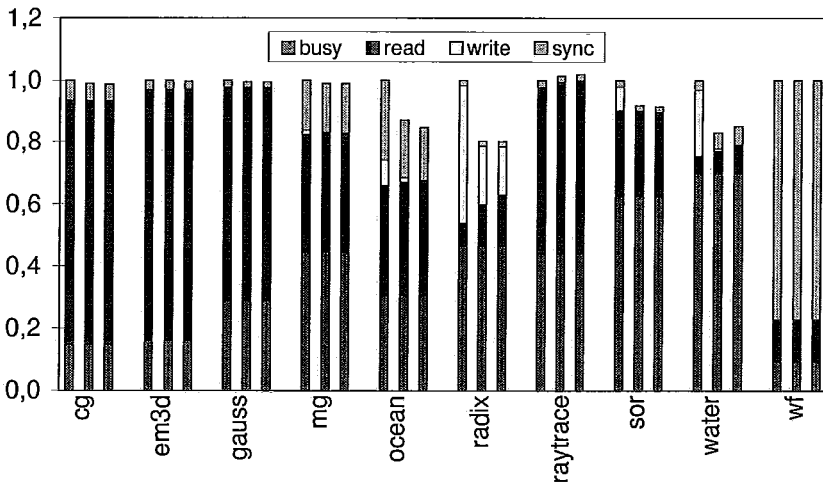


Figura 4.9: Tempos de Execução (com Relação a 1 Canal de Atualização) para 1, 2 e 4 Canais de Atualização em 16 Nós

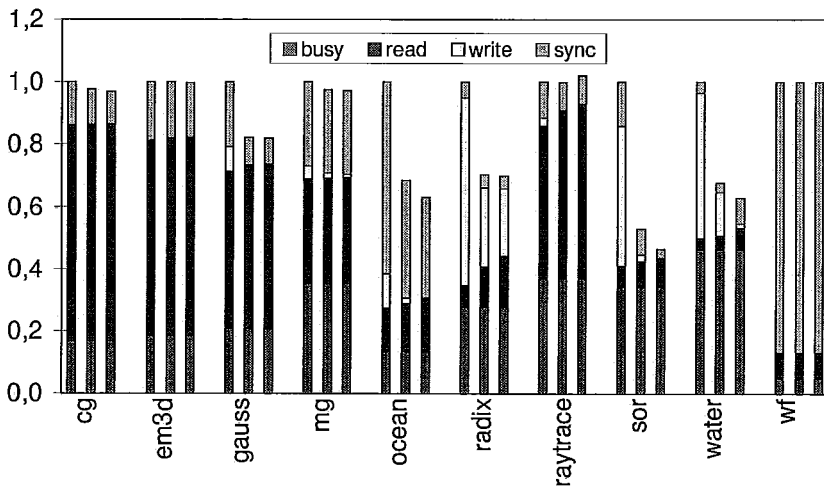


Figura 4.10: Tempos de Execução (com Relação a 1 Canal de Atualização) para 1, 2 e 4 Canais de Atualização em 32 Nós

para algumas aplicações, o desempenho pode ser significativamente melhorado usando mais que um canal de coerência. Este efeito é mais acentuado em configurações com um número maior de nós, onde a serialização do acesso a um único canal de coerência tem um elevado impacto negativo no desempenho do sistema. Os ganhos de desempenho resultam, principalmente, dos ganhos no desempenho das escritas, i.e., reduz os tempos gastos com escritas e esvaziamento do *write-buffer*. Note, no entanto, que estes ganhos causam às vezes um aumento significativo das latências de leitura, como nos casos de Radix e Water, devido ao aumento de contenção. Este efeito é particularmente acentuado em Radix com 16 ou 32 nós. Na tabela 4.4 se observa que o tempo de escrita representa uma fração significativa do tempo de execução com dois canais de coerência. Como mostram

as figuras 4.9 e 4.10, a diminuição do tempo de escrita é completamente contrabalançado pelo aumento na latência de leitura.

A segunda observação é que dois canais de coerência são suficientes para obter a maioria de benefícios atingíveis pela utilização de múltiplos canais, pelo menos até 32 nós. Devido a decrescerem exponencialmente os ganhos alcançados pelo aumento no número de canais de atualização, pode-se achar que dois canais de coerência devem oferecer uma melhor relação custo/desempenho para máquinas com até 64 ou 128 nós.

A terceira observação importante que resulta destas figuras é que as aplicações podem ser claramente divididas em dois grupos: aquelas para as quais um canal de coerência é suficiente (CG, Em3d, Mg, Raytrace e WF) e aquelas para as quais dois canais são suficientes (Gauss, Ocean, Radix, SOR e Water).

Em resumo, os resultados desta seção mostram que todos os sistemas são equivalentes em termos do desempenho das operações de escrita na maioria dos casos. A exceção é a aplicação Radix, para a qual LambdaNet apresenta o menor *overhead* de escritas. Os resultados acima também justificam a seleção de dois como o número de canais de coerência em OPTNET.

4.4.4 Impacto dos Parâmetros Arquiteturais

Nesta subseção será avaliado o impacto de algumas das suposições feitas nas simulações anteriores e, assim, poder entender o comportamento de OPTNET mais detalhadamente. Inicialmente, será feito um estudo do efeito do tamanho da cache secundária. Logo após será realizada uma avaliação do impacto da taxa de transmissão e, finalmente, será abordado o efeito de diferentes latências de leitura nas memórias eletrônicas. Para simplificar a análise, os resultados estarão concentrados em uma aplicação representativa de cada um dos grupos identificados na seção anterior: Mg e Ocean.

Tamanho da Cache Secundária. O tamanho da cache secundária pode potencialmente afetar a comparação entre os sistemas estudados, devido às suas diferenças nas latências de falha de uma leitura. A intuição inicial é que aumentos nos tamanhos das caches deve reduzir as diferenças do tempo de execução entre os sistemas, tanto como esses aumentos reduzam a taxa de falhas de leitura. Adicionalmente, para caches muito grandes, os sistemas baseados em atualizações devem se beneficiar mais dos aumentos nos tamanhos das

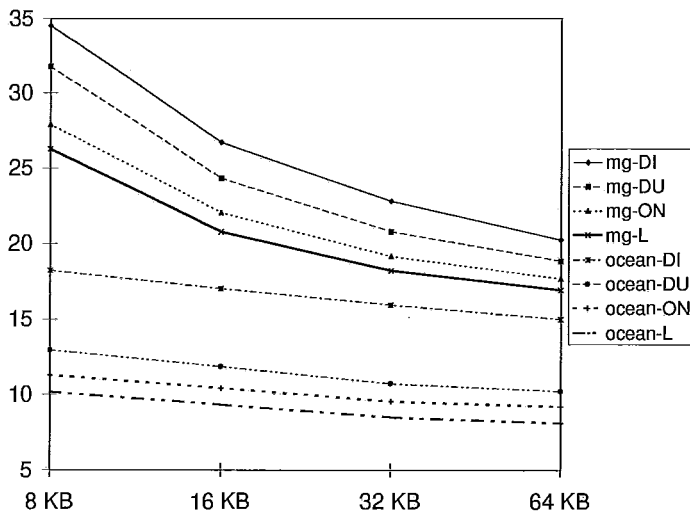


Figura 4.11: Tempos de Execução (em Milhões de Ciclos) como Função do Tamanho da Cache Secundária

caches que DMON-I, pois a taxa de falha nos primeiros sistemas tende a ser igual à taxa de falhas de inicialização, enquanto que a taxa de falhas de DMON-I tende a ser a soma das taxas de falha de inicialização e coerência.

A figura 4.11 apresenta o impacto do tamanho da cache secundária no tempo de execução de Mg e Ocean em um multiprocessador de 16 nós com OPTNET (“ON”), LambdaNet (“L”), DMON-U (“DU”) e DMON-I (“DI”). A figura confirma a intuição mencionada acima. Em particular, a figura mostra que um aumento no tamanho da cache reduz as diferenças no tempo de execução para Mg. Esta aplicação apresenta uma excelente localidade de referência e, assim, os aumentos no tamanho das caches reduzem notavelmente as taxas de falha de leitura tanto para os sistemas baseados em atualizações como para DMON-I. As reduções nas taxas de falha de leitura como uma função do aumento do tamanho das caches não são significativas em Ocean, especialmente para o caso de DMON-I. Apesar disto não ser completamente óbvio na figura, o desempenho de DMON-I melhora, mas a uma taxa inferior que a dos sistemas baseados em atualizações.

Taxa de Transmissão. A taxa de transmissão óptica também tem um efeito potencial sobre as nossas comparações. Intuitivamente, taxas de transmissão maiores devem reduzir as diferenças no tempo de execução entre os sistemas baseados em atualizações, devido às menores discrepâncias de tempo nas latências de leitura e nas transações de coerência. A figura 4.12 apresenta o tempo de execução de um sistema com 16 nós como uma função

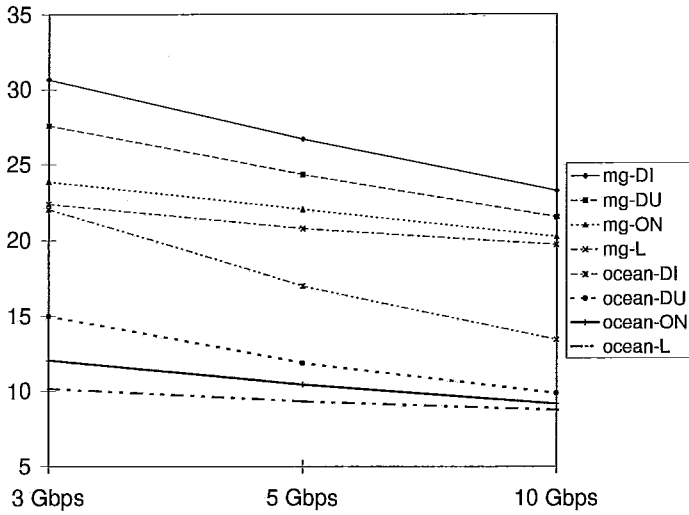


Figura 4.12: Tempos de Execução (em Milhões de Ciclos) como Função da Taxa de Transmissão

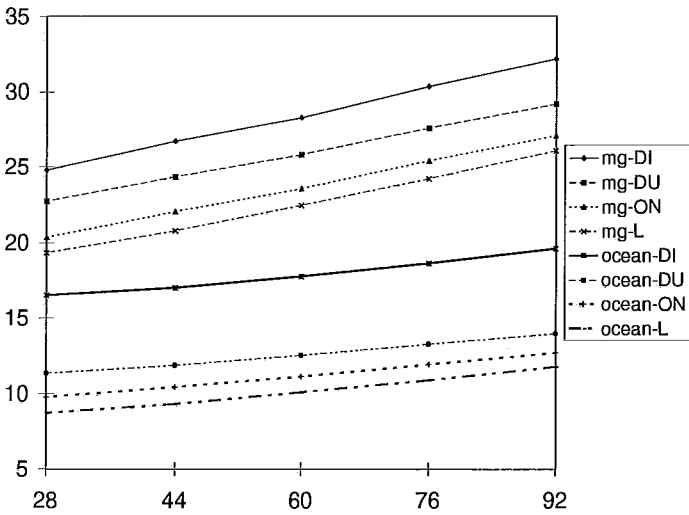


Figura 4.13: Tempos de Execução (em Milhões de Ciclos) como Função da Latência de Leitura na Memória (em Ciclos de Processador)

da taxa de transmissão (em Gbits/s) de cada canal. A figura confirma o esperado, mostrando que, com o avanço da tecnologia, as diferenças de desempenho entre OPTNET e LambdaNet decrescem, fazendo com que OPTNET seja ainda mais competitiva.

Latência da Leitura de Memória. O tempo de serviço das memórias é outro fator que pode afetar a comparação dos sistemas. É esperado que aumentos nas latências de leitura reduzam as diferenças percentuais entre os sistemas baseados em atualizações, como resultado das menores diferenças nas suas latências de leitura. Em comparação com DMON-I, os sistemas baseados em atualizações devem ser mais atrativos sob elevada latência, já que qualquer diferença na taxa de falhas produz um efeito ainda mais acen-

tuado. A figura 4.13 mostra o tempo de execução de um multiprocessador de 16 nós em função da latência de leitura de um bloco de memória (em ciclos de processador). Embora as tendências de desempenho não sejam muito acentuadas na figura, os experimentos confirmam a intuição.

Em resumo, o tamanho da cache secundária, a taxa de transmissão, o tempo de serviço das memórias têm um efeito significativo no desempenho destes sistemas. No entanto, este efeito é só quantitativo, i.e., a variação destes parâmetros não muda qualitativamente as tendências observadas nos resultados da seção anterior.

4.5 Trabalhos Relacionados

Uma abordagem comum para a utilização da comunicação ótica em redes de computadores é através das redes WDM, como mencionado no capítulo 2. Redes óticas OTDM também têm sido propostas como uma alternativa às redes WDM [21, 20], mas a tecnologia OTDM ainda não está madura.

Redes óticas com WDM têm sido parte de outros projetos de computação paralela, embora os resultados obtidos sejam os primeiros a comparar várias destas redes sob as mesmas condições arquiteturais. Ghose *et al.* [17] propuseram uma rede ótica WDM chamada de Optimul para explorar os benefícios da operação concorrente de múltiplos canais tanto em computadores paralelos com memória compartilhada como com passagem de mensagens. LambdaNet é o limite superior de desempenho para Optimul, já que a contenção originada pelo compartilhamento dos canais de Optimul degrada o seu desempenho em proporção ao número de nós que compartilham cada canal. Para o caso de um multiprocessador, Optimul utiliza um esquema de atualizações com *snooping* para tomar vantagem da enorme largura de faixa dos enlaces óticos. A comparação feita entre OPTNET e LambdaNet indica que OPTNET também deve apresentar uma melhor relação custo/desempenho que Optimul, já que as diferenças de desempenho entre os sistemas OPTNET e Optimul devem ser ainda menores que as existentes entre os sistemas OPTNET e LambdaNet, e o custo do *hardware* ótico de Optimul é somente um fator constante melhor que o de LambdaNet.

Ha e Pinkston [10] propuseram a rede DMON e o sistema DMON-I estudado anterior-

mente. Embora DMON seja uma rede bastante similar a OPTNET, as diferenças principais são duas: a) os canais de disseminação de OPTNET são mais bem-comportados sob alta contenção, e b) os canais de comunicação ponto-a-ponto de OPTNET não requerem nenhum mecanismo para controle de acesso. Além de outras contribuições, o nosso estudo estendeu o seu trabalho propondo um sistema baseado em atualizações para DMON. Os resultados de desempenho mostram que OPTNET é melhor que os sistemas baseados em DMON em todos os casos.

Dowd e Chu [3] estudaram a interação dos diferentes mecanismos de controle de acesso com os protocolos de coerência nos multiprocessadores escaláveis. Mais especificamente, o desempenho de um sistema com um protocolo de coerência baseado em diretórios e com um protocolo de acesso TDMA foi comparado a um sistema com um protocolo de coerência baseado em *snooping* e com um protocolo de acesso baseado em reserva. Ambos os sistemas utilizaram protocolos de coerência baseados em invalidações. Os resultados mostram que o sistema baseado em *snooping* é melhor que o baseado em diretórios para a maioria das hipóteses arquiteturais. DMON-I, por outro lado, usa tanto *snooping* como diretórios para criar um sistema baseado em invalidações superior ao uso exclusivo de *snooping* ou diretórios. Os sistemas baseados em OPTNET, LambdaNet e DMON-U não usam diretórios, mas utilizam protocolos baseados em atualizações, os quais se sabe, são superiores aos baseados em invalidações na presença de uma largura de faixa extremamente grande [3].

4.6 Conclusões

Neste capítulo foi apresentada OPTNET, uma nova rede ótica e o seu protocolo de coerência associado para multiprocessadores. Através de um grande conjunto de detalhadas simulações, foi demonstrado que o desempenho dos sistemas baseados em OPTNET é superior ao apresentado pelos sistemas baseados em DMON para todas as aplicações avaliadas. Note que OPTNET não requer mais *hardware* que DMON. Além disso, a comparação entre OPTNET e LambdaNet mostra que as diferenças de desempenho são de 0 a 12% em favor de LambdaNet (em média 4%). Estes resultados são extremamente favoráveis a OPTNET devido à diferença considerável em requerimentos de *hardware*

ótico existente entre os dois multiprocessadores. Mesmo que, no futuro, o custo dos componentes óticos se torne muito baixo, a complexidade linear do *hardware* ótico de OPTNET permitirá uma maior escalabilidade que a oferecida por LambdaNet.

Um estudo de variação de parâmetros mostra resultados qualitativamente similares para a maioria de hipóteses arquiteturais. Baseado nisso, pode-se concluir que OPTNET apresenta a melhor relação custo/desempenho para multiprocessadores para a maioria das suposições arquiteturais. As características principais para a excelente relação custo/desempenho de OPTNET podem-se resumir nos seguintes pontos: a) a disseminação dos pedidos de acesso à memória e das transações de sincronização simplifica o *hardware* eliminando a necessidade de diretórios, b) a disseminação das transações de sincronização otimiza o protocolo de coerência ao informar eficientemente aos processadores das mudanças feitas nos dados compartilhados, e c) o agrupamento de canais melhora o desempenho ao desacoplar o tráfego de escritas do tráfego de leituras que é mais crítico para o tempo de execução.

Capítulo 5

NetCache

A disparidade que existe entre as velocidades do processador e da memória continua crescendo. Na atualidade, um acesso a memória demora algumas dezenas de ciclos para completar, especialmente nos multiprocessadores escaláveis. As memórias cache são consideradas, normalmente, a melhor técnica para tolerar as elevadas latências de acesso a memória. Os sistemas de computação modernos usam vários níveis de caches para reduzir o custo médio de um acesso a memória. Infelizmente, às vezes as caches são menores que os conjuntos de trabalho associados aos seus processadores, o que produz uma porcentagem relativamente grande de referências aos módulos de memória. Essa possibilidade afeta particularmente o desempenho de um sistema quando as referências à memória são dirigidas aos módulos remotos através de uma rede de interconexão escalável.

Baseados nas observações anteriores e na necessidade de reduzir a latência média de acesso à memória, este capítulo propõe a utilização de uma rede ótica, não só como um meio de comunicação, mas também como uma cache para os dados compartilhados dentro de um multiprocessador. Mais especificamente, a nossa proposta é usar uma rede ótica em anel para manter circulando continuamente uma certa quantidade de dados compartilhados recentemente acessados. Esses dados são organizados em uma estrutura similar à de uma cache compartilhada por todos os processadores. A maioria dos aspectos aplicáveis às caches tradicionais, tais como taxas de acerto/falha, capacidade, unidades de armazenamento e associatividade, são também aplicáveis a essa rede, chamada de NetCache [7].

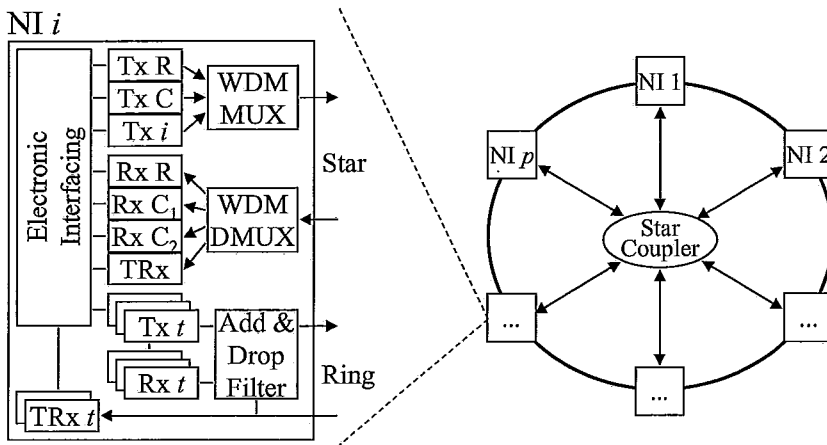


Figura 5.1: Detalhe da Arquitetura de NetCache

5.1 Arquitetura de NetCache

Esta seção inicia descrevendo a arquitetura básica da rede para posteriormente, descrever com mais detalhes o protocolo de coerência e as operações de acesso a memória. Finalmente, algumas implementações alternativas que cumprem os mesmos objetivos de NetCache serão discutidas.

5.1.1 Arquitetura Básica

Cada nó do multiprocessador baseado em NetCache é extremamente simples e tem uma estrutura similar à apresentada no capítulo anterior para OPTNET (veja a figura 4.3). A única diferença neste caso é que a interface de rede conecta o nó a duas subredes: uma subrede WDM em estrela e uma subrede WDM em anel.

A figura 5.1 detalha a arquitetura de NetCache, incluindo as suas interfaces e subredes. A subrede em estrela é a mesma apresentada para OPTNET, enquanto que a subrede em anel é diferente de qualquer outra rede ótica previamente proposta. Assim, a subrede em anel será a única descrita nesta seção.

A subrede em anel é o aspecto mais interessante da arquitetura de NetCache, já que ela é usada para armazenar certa quantidade de dados compartilhados recentemente acessados. Os dados estão continuamente circulando no anel através de canais WDM, referidos como *cache-channels*. De acordo com as classificações apresentadas nos capítulos 2 e 3, os *cache-channels* formam uma rede FT- $\{F,T\}$ R e implementam uma memória ótica síncrona baseada unicamente em componentes optoeletrônicos. Além disso, os dados

estão organizados como em uma cache compartilhada por todos os nós da máquina. Desta forma, o anel se transforma em um nível extra de cache que armazena dados de qualquer nó. No entanto, o anel não respeita as propriedades de inclusão da hierarquia de caches, i.e., os dados armazenados pelo anel não são necessariamente um superconjunto dos dados armazenados nas caches de níveis superiores. Em outras palavras, a capacidade de armazenamento do anel é totalmente independente dos tamanhos individuais ou combinados das caches secundárias. A capacidade de armazenamento do anel é simplesmente proporcional ao número de canais disponíveis, à largura de faixa e ao comprimento dos canais, como mostrado no capítulo 3.

Note que existe uma relação entre o comprimento da fibra ótica (e por conseqüência, a capacidade de armazenamento do anel) e a latência de acesso a memória remota, já que quanto maior seja a fibra maior será a latência de circulação dos dados no anel, sendo que o tempo de acesso aos dados no anel deve ser inferior ao tempo de acesso aos dados remotos. Quanto maior seja a latência de acesso aos dados remotos, maior pode ser a cache compartilhada. Aumentos na capacidade de armazenamento podem também ser produzidos por um aumento no número de canais ou na taxa de transmissão. No entanto, esses fatores dependem diretamente dos desenvolvimentos tecnológicos futuros. Felizmente, esses desenvolvimentos acontecerão naturalmente, já que a necessidade por largura de faixa continua crescendo e a comunicação ótica se faz mais popular.

O anel trabalha da seguinte maneira. Cada *cache-channel* armazena os blocos de um *home* em particular. Assumindo um total de c *cache-channels* e p processadores, cada *home* possui $t = c/p$ *cache-channels* para os seus blocos. Como os *cache-channels* e blocos são associados aos *homes* de uma forma *round-robin* intercalada, o *cache-channel* destinado a um certo bloco pode ser determinado através do módulo do endereço do bloco pelo número total de canais (c). Um bloco determinado pode ser introduzido em qualquer lugar dentro do *cache-channel*.

Como em uma cache convencional, a cache compartilhada possui uma etiqueta de endereço junto a cada bloco (linha de cache compartilhada). A etiqueta contém parte do endereço do bloco de cache e consome uma fração (usualmente bastante pequena) da capacidade de armazenamento do anel. O acesso a cada bloco por parte dos diferentes processadores é estritamente seqüencial como determinado pelo fluxo de dados no *cache-*

channel. A interface de rede acessa um bloco através de um registrador de deslocamento que obtém réplicas dos bits que circulam pelo anel. Cada registrador de deslocamento é da largura de um bloco de cache. Quando o registrador é preenchido totalmente, o *hardware* move o bloco para um outro registrador, chamado de registrador de acesso, onde os dados podem ser manipulados. Com uma taxa de transmissão de 10 Gbits/s e considerando um bloco de cache de 64 Bytes, o registrador de deslocamento demora 50 nanosegundos para ser preenchido. Esse tempo é suficiente para comparar a etiqueta e possivelmente copiar o bloco antes que o registrador de acesso seja sobrescrito.

A interface de NetCache em cada nó pode ler qualquer um dos *cache-channels*, mas só pode escrever nos t *cache-channels* associados ao nó. Além disso, a interface de NetCache resincroniza e regeira os t *cache-channels*. Para cumprir com esta funcionalidade, a interface requer 2 receptores sintonizáveis (“*TRx*”) e t conjuntos de receptores (“*Rx*”) e transmissores (“*Tx*”) fixos, como mostra a figura 5.1. Um dos receptores sintonizáveis é usado para acessar o *cache-channel* do último pedido, enquanto que o outro é pré-sintonizado ao canal seguinte¹. Os t transmissores fixos são usados para inserir novos dados em qualquer um dos *cache-channels* associados ao nó. Em conjunto com esses transmissores, os t receptores fixos são utilizados para recircular os dados nos *cache-channels* associados ao nó. Assim, o custo de *hardware* para esta subrede é de $2 \times p + 2 \times c$ componentes óticos.

Como a complexidade de *hardware* da subrede em estrela é $7 \times p$ (OPTNET), o custo total de *hardware* da arquitetura NetCache é $9 \times p + 2 \times c$. Nas simulações realizadas, c é igual a $8 \times p$, o que dá um total de $25 \times p$ componentes óticos. Esta complexidade de *hardware* é 4 vezes maior que a de DMON, mas é linear em p , o que é oposto à complexidade quadrática em p de LambdaNet.

5.1.2 Protocolo de Coerência

O protocolo de coerência é uma parte integrante da arquitetura de NetCache. Ele é baseado em uma coerência com atualizações e suportado tanto por uma comunicação ponto-a-ponto como por disseminação. O tráfego de atualizações flui através dos canais de

¹O objetivo é ocultar o custo de sintonização mediante a predição do canal que será requerido posteriormente.

coerência, enquanto que os blocos de dados são enviados pelos *home-channels* e *cache-channels*. O canal de pedidos transporta todos os pedidos de leitura à memória e os reconhecimentos das atualizações. A descrição, a seguir, mostra o protocolo de coerência para as operações de leitura e escrita com mais detalhes.

Leituras. Em um acesso de leitura, a hierarquia de memória é atravessada de forma a permitir que a palavra requerida seja encontrada o mais rápido possível. Uma falha na cache secundária bloqueia o processador e é tratada diferentemente dependendo do tipo de dado a ser lido. No caso do bloco solicitado ser privado ou mapeado na memória local, o acesso de leitura é dirigido à memória local, sendo o bloco retornado diretamente ao processador. Se o bloco é compartilhado e está mapeado em um outro nó, a falha causa um acesso à NetCache. O pedido é enviado ao correspondente nó através do canal de pedidos, e o receptor sintonizável na subrede em estrela é ajustado ao correspondente *home-channel*. Em paralelo, o nó solicitante sintoniza um dos receptores na subrede em anel ao respectivo *cache-channel*. O nó solicitante então espera até o bloco ser recebido, seja através do *home-channel* ou do *cache-channel*. Posteriormente, o bloco será lido e retornado à cache secundária.

Quando um pedido chega ao *home*, ele verifica se o bloco já está em algum dos *cache-channels*. Para manter esta informação, a interface de NetCache mantém uma *hash-table* que armazena os endereços dos blocos que atualmente estão no anel. Se o bloco já está na cache compartilhada, o *home* simplesmente ignora o pedido, pois o bloco será eventualmente recebido pelo nó solicitante. Se o bloco não está atualmente na cache, o *home* lê o bloco da memória e o coloca no respectivo *cache-channel*, substituindo um dos blocos aí presentes, se necessário². Além de retornar o bloco pedido através do *cache-channel*, a interface de NetCache também envia o bloco através do seu *home-channel*.

É importante notar que o protocolo inicia as transações de leitura nas duas subredes de forma a garantir que uma falha na cache compartilhada não demore mais que um acesso à memória remota. Se as leituras fossem iniciadas unicamente na subrede em anel, as falhas de acesso na cache compartilhada demorariam uma meia volta a mais (em média) no anel para serem satisfeitas.

²As substituições são aleatórias e não requerem escritas adicionais à memória, pois a cache compartilhada e a memória estão sempre atualizadas.

Escritas. A arquitetura do multiprocessador simulado implementa o modelo de memória *release-consistency* [41]. Escritas consecutivas a um mesmo bloco são agrupadas no *write-buffer*. Escritas a um bloco privado são enviadas diretamente à memória local através das caches primária e secundária. Escritas a blocos compartilhados são sempre enviadas por um dos canais de coerência na forma de uma atualização através das caches locais. Uma atualização só leva as palavras do bloco que foram modificadas.

Cada atualização deve ser reconhecida pelo seu correspondente *home* antes que uma outra atualização do mesmo nó possa ser emitida. Desta forma, se evita que os módulos de memória possuam filas de entrada grandes (i.e., os reconhecimentos às atualizações são usados como um mecanismo de controle de fluxo). Os outros nós, que possuem cópias do bloco nas suas caches, simplesmente atualizam as caches locais. Quando o *home* recebe a atualização, ela é inserida em uma fila FIFO, e o *home* envia um reconhecimento através do canal de pedidos. No entanto, o reconhecimento não pode ser enviado imediatamente se a fila de memória estiver cheia acima de um ponto de histerese. Neste caso, o *home* retarda o envio do reconhecimento até que possa permitir, de forma segura, o envio de outra atualização por parte do mesmo nó. Um nó pode unicamente adquirir um *lock* ou passar uma barreira depois de ter esvaziado a sua fila de memória. Note que os reconhecimentos das atualizações não sobrecarregam o canal de pedidos, pois os reconhecimentos são mensagens curtas que ocupam um único *slot*.

Depois de ter inserido a atualização na fila de memória, o *home* verifica se o bloco atualizado está presente em um dos seus *cache-channel*. Se está presente, além de atualizar a sua memória e as caches locais, o *home* atualiza o seu *cache-channel*. Existem duas condições de corrida no protocolo de coerência que devem ser levadas em conta. A primeira ocorre quando uma operação de coerência é observada por um bloco que tem uma leitura pendente. De forma similar ao protocolo proposto para OPTNET, esta condição é tratada armazenando as atualizações e posteriormente combinando-as com o bloco recebido da memória. O segundo tipo de corrida ocorre devido à existência de uma janela de tempo entre a disseminação da atualização e a modificação da cópia do bloco na cache compartilhada. Durante esta janela de tempo, um nó pode iniciar a leitura de um bloco, lê-lo da cache compartilhada, e nunca aplicar a atualização correspondente. Para evitar este problema, cada interface de rede inclui uma pequena fila FIFO que armazena os

endereços dos blocos que foram previamente atualizados. Qualquer acesso à cache compartilhada, procurando um bloco presente na fila, deve ser atrasado até que o endereço do bloco saia da fila. Isto garante que quando a leitura for realizada, a cópia do bloco na cache compartilhada estará também atualizada. A gerência desta fila extra é muito simples: a entrada na cabeça da fila pode ser jogada fora quando ela tiver residido na fila por um tempo igual a duas voltas na subrede em anel, i.e., a máxima quantidade de tempo que o *home* pode demorar para atualizar o bloco na cache compartilhada. Assim, o tamanho máximo da fila é igual ao número máximo de atualizações que podem ser emitidas nesse período de tempo. Nas simulações realizadas, esse valor é 54.

5.1.3 Implementação Alternativa

O principal objetivo da arquitetura de NetCache, trazer os dados das memórias remotas para mais perto do processador, pode ser alcançado com uma implementação diferente da rede. Mais especificamente, ao invés de armazenar os dados na fibra ótica da subrede em anel, eles podem ser armazenados em uma memória eletrônica e continuamente disseminados através de canais adicionais na subrede em estrela. Na verdade, os *c cache-channels* do anel podem ser trocados por *c* canais extras na subrede em estrela, transmitindo os mesmos dados anteriormente transmitidos em cada um dos canais extra. Esta arquitetura modificada teria um desempenho idêntico ao de NetCache, já que manteria a sua capacidade de armazenamento, a sua organização lógica, o protocolo de coerência e as latências para o caso de uma falha de leitura na cache secundária. Adicionalmente, a nova arquitetura reduziria o *hardware* ótico, pois *c* receptores fixos poderiam ser eliminados. No entanto, seriam necessárias memórias eletrônicas adicionais, suficientes para armazenar os dados a serem disseminados.

Devido às similaridades entre estas duas arquiteturas, os resultados apresentados nas próximas seções são válidos para as duas alternativas. A seleção da forma de implementação unicamente depende do custo do *hardware* (eletrônica vs. ótica) no momento que o multiprocessador seja construído. Ao longo deste capítulo foi selecionada a implementação de NetCache por duas razões: a) o custo decrescente dos componentes óticos sugere que no futuro a ótica pode prover uma melhor relação custo/desempenho que a eletrônica, e o mais importante pelo momento, b) existe a intenção de extrapo-

Operação	Latência
Acerto na cache compartilhada	
1. Verificação da cache primária	1
2. Verificação da cache secundária	4
3. Retardo médio na cache compartilhada	25
4. Transferência da NI para a cache secundária	16
Total	46
Falha na cache compartilhada	
1. Verificação da cache primária	1
2. Verificação da cache secundária	4
3. Retardo médio TDMA	8
4. Pedido à memória	1*
5. Retardo de propagação	1
6. Leitura de memória	76 ⁺
7. Transferência do bloco	11
8. Retardo de propagação	1
9. Transferência da NI para a cache secundária	16
Total	119

Tabela 5.1: Tempos de Leitura para NetCache em Ciclos de Processador

lar o uso de NetCache aos blocos do disco, o que requereria uma grande quantidade de memória eletrônica para a sua implementação alternativa. A arquitetura de NetCache pode ser aplicada ao caso de blocos de disco com um mínimo custo adicional: o de uma fibra ótica maior.

5.2 Metodologia

Para avaliar o desempenho de NetCache e compará-lo em relação a propostas de multiprocessadores baseados em redes óticas previamente estudadas, utilizamos simulações de aplicações paralelas reais.

5.2.1 Simulação

Simulamos multiprocessadores de 16 nós com as redes de interconexão NetCache, DMON e LambdaNet. Os simuladores são baseados no MINT [42]. Cada nó das máquinas simuladas contém um processador de 200 MHz, um *write-buffer* de 16 entradas, uma cache primária de 4 KBytes diretamente mapeada e com blocos de 32 Bytes, uma cache secundária de 16 KBytes diretamente mapeada e com blocos de 64 Bytes,

memória local, e uma interface de rede. Note que as caches simuladas são pequenas porque as limitações no tempo de simulação nos impedem de usar entradas reais. Na verdade, a capacidade das caches primária e secundária foram reduzidas por um fator de 32. O objetivo destas reduções é produzir, aproximadamente, o mesmo tráfego de acessos à memória que nos sistemas reais. Assim, a capacidade de armazenamento de NetCache deverá também aumentar por um fator próximo a 32 no caso de sistemas reais. O aumento da capacidade de armazenamento de NetCache pode ser obtido aumentando o comprimento do anel ótico, aumentando a taxa de transmissão e/ou usando mais *cache-channels*. Entretanto, aumentar a capacidade do anel ótico por um fator de 32 pode não ser prático com a tecnologia ótica atual. Espera-se que em um futuro próximo este aumento de tamanho seja possível. Considerando-se esses avanços futuros, nossas suposições de capacidade se tornam bastante conservadoras, especialmente se forem consideradas técnicas de multiplexação tais como OTDM, a qual deverá suportar até 5000 canais [21].

Os dados compartilhados são intercalados a nível de bloco entre as diferentes memórias. Assumimos que todas as instruções e acertos de leitura na cache primária demoram um ciclo de processador. Falhas de leitura na cache primária bloqueiam o processador até que o pedido de leitura é satisfeito. Um acerto de leitura na cache secundária demora 12 ciclos para completar. As escritas vão ao *write-buffer* e demoram um ciclo, exceto quando o *write-buffer* está cheio. Neste último caso, o processador é bloqueado até que uma entrada seja liberada. As leituras podem ser escalonadas antes que as escritas no *write-buffer*. Um módulo de memória pode prover as primeiras duas palavras 12 ciclos após a emissão do pedido. As outras palavras são entregues a uma taxa de duas palavras por cada 8 ciclos. A contenção na memória e na rede são totalmente modeladas.

No protocolo de coerência baseado em atualizações, somente o bloco na cache secundária é atualizado quando uma atualização chega ao nó. A cópia desse bloco na cache primária é invalidada. Além disso, para reduzir o tráfego de escritas, o *write-buffer* agrupa todas as escritas para um mesmo bloco em todos os protocolos implementados. Uma atualização somente leva as palavras que foram modificadas em cada bloco. As implementações dos protocolos assumem um modelo de memória *release-consistency* [41].

A taxa de transmissão ótica simulada foi de 10 Gbits/s. Nas simulações de NetCache

Operação	Latência	
	LambdaNet	DMON
Falha na cache secundária		
1. Verificação da cache primária	1	1
2. Verificação da cache secundária	4	4
3. Retardo médio TDMA	—	8
4. Reserva do canal	—	1*
5. Retardo de sintonização	—	4
6. Pedido à memória	1*	2
7. Retardo de propagação	1	1
8. Leitura de memória	76 ⁺	76 ⁺
9. Retardo médio TDMA	—	8
10. Reserva do canal	—	1*
11. Transferência de bloco	11*	12
12. Retardo de propagação	1	1
13. Transferência da NI para a cache secundária	16	16
Total	111	135

Tabela 5.2: Tempos de Leitura para LambdaNet e DMON em Ciclos de Processador

assumimos 128 *cache-channels*. O comprimento da fibra é aproximadamente 45 metros. Estes parâmetros produzem uma latência de 40 ciclos para uma circulação completa dos dados e uma capacidade de armazenamento de 32 KBytes. O tamanho do bloco de cache compartilhada é 64 Bytes.

O protocolo de coerência implementado acima da cache compartilhada foi descrito anteriormente. A latência de leitura na cache compartilhada de NetCache foi dividida nos seus componentes base na tabela 5.1. As latências de uma falha de leitura na cache secundária para LambdaNet e DMON são apresentadas na tabela 5.2. As latências de uma transação de coerência em NetCache, LambdaNet, DMON com coerência baseada em atualizações (DMON-U), e DMON com I-SPEED (DMON-I) são mostradas na tabela 5.3, assumindo 8 palavras escritas em cada bloco de cache³. Todos os números nas tabelas estão em ciclos de processador e assumem um cenário livre de contenção nos canais de comunicação e na memória. Os valores marcados com '*' e '+' são os que podem ser aumentados pela contenção e/ou serialização na rede e memória, respectivamente.

Note que durante as simulações o mínimo *slot* TDMA é de um ciclo de processador

³Note que a maior parte de todas as transações de coerência são normalmente escondidas do processador pelo *write-buffer*.

Operação	Latência (em ciclos)			
	NetCache	LambdaNet	DMON-U	DMON-I
1. Verificação da cache secundária	4	4	4	4
2. Escrita na NI	10	10	10	2
3. Retardo médio TDMA	8*	—	8	8
4. Reserva do canal	—	—	1*	1*
5. Atualização/Invalidação	8	7	8	2
6. Retardo de propagação	1	1	1	1
7. Retardo médio TDMA	8	—	8	8
8. Reserva do canal	—	—	1*	1*
9. Reconhecimento	1*	1*	1	1
10. Retardo de propagação	1	1	1	1
11. Escrita na cache secundária	—	—	—	8
Total	41	24	43	37

Tabela 5.3: Tempos de uma Transação de Coerência para NetCache, LambdaNet, DMON-U e DMON-I em Ciclos de Processador

tanto para DMON como para NetCache. Assim, cada *slot* do canal de controle em DMON e do canal de pedidos em NetCache é de um ciclo. Cada *slot* dos canais de coerência em NetCache é de pelo menos 2 ciclos. A duração real de cada *slot* neste caso depende do número de palavras atualizadas.

Os parâmetros de simulação assumidos representam uma percepção pessoal do que é razoável para os multiprocessadores atuais e em um futuro próximo. O estudo do espaço de parâmetros apresentado na seção de resultados permitirá investigar o impacto das mais importantes hipóteses arquiteturais.

5.2.2 Aplicações

O conjunto de aplicações usadas nesta avaliação é composto pelas 10 aplicações usadas na avaliação de OPTNET, além de FFT e LU. Essas duas novas aplicações são parte do conjunto de *benchmarks* SPLASH-2 [43] e foram incluídas por apresentar características interessantes de reuso de dados: FFT quase não tem reuso e LU apresenta um reuso significativo. A tabela 5.4 mostra todas as aplicações e os seus parâmetros de entrada.

Programa	Descrição	Tamanho da Entrada
CG	Gradiente Conjugado (kernel)	1400 × 1400 doubles, 78148 <> 0
Em3d	Propagação de ondas eletromagnéticas	8 K nós, 5% remotos, 10 iter.
FFT	Transformada Rápida de Fourier 1D	16 K pontos
Gauss	Eliminação de Gauss sem blocos	256 × 256 floats
LU	Fatorização LU por blocos	512 × 512 floats
Mg	Poisson 3D usando técnicas de multigrid	24 × 24 × 64 floats, 6 iterações
Ocean	Simulação da movimentação de oceanos	Grade 66 × 66
Radix	Ordenação de inteiros	512 K chaves, radix 1024
Raytrace	Traçador paralelo de raios luminosos	teapot
SOR	Relaxamento progressivo	256 × 256 floats, 100 iterações
Water	Simulação de moléculas de água (spatial)	512 moléculas, 4 passos
WF	Algoritmo do caminho mais curto	384 vértices, conectividade 50%

Tabela 5.4: Descrição das Aplicações e Principais Parâmetros de Entrada

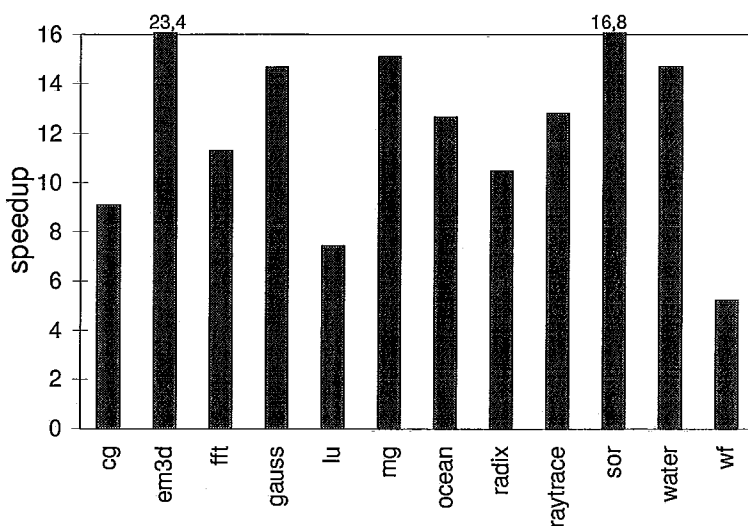


Figura 5.2: Ganho de Desempenho em um Multiprocessador de 16 Nós com NetCache

5.3 Resultados

Nesta seção será avaliado o desempenho de um multiprocessador baseado em NetCache e comparado com multiprocessadores baseados nas redes LambdaNet e DMON. Inicialmente são mostrados os resultados de tempos de execução e de eficiência da arquitetura como uma cache de dados. Posteriormente serão avaliadas diferentes organizações da cache e políticas de substituição. Finalmente, o efeito de algumas das suposições feitas durante as simulações base serão avaliadas.

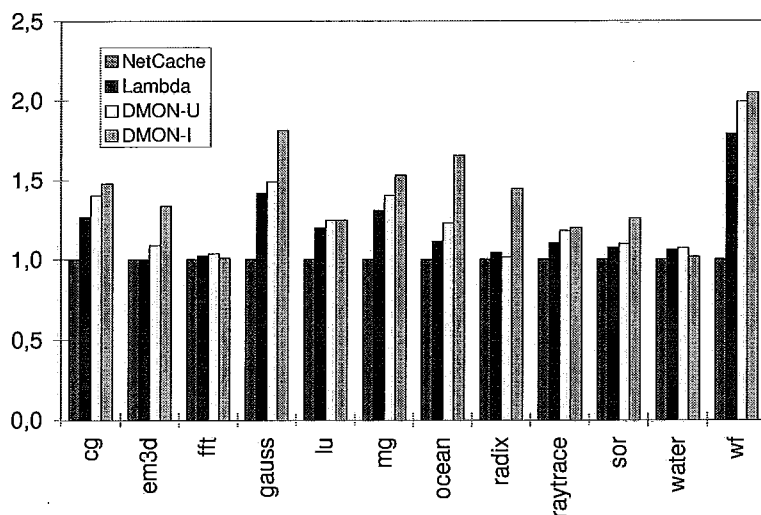


Figura 5.3: Tempos de Execução (com Relação ao MP NetCache) de NetCache, LambdaNet, DMON-U e DMON-I

5.3.1 Desempenho Geral

A figura 5.2 mostra o ganho de desempenho das aplicações executando em um multiprocessador baseado em NetCache com 16 nós e uma cache compartilhada de 32 KBytes. A figura demonstra que, exceto para CG, LU e WF, as aplicações exibem ganhos bons para 16 nós. Os dois extremos, Em3d e WF, foram explicados no capítulo anterior.

A figura 5.3 mostra os tempos de execução das 12 aplicações em um multiprocessador de 16 nós e 32 KBytes de cache compartilhada. Para cada aplicação são mostrados, da esquerda para a direita, os desempenhos de NetCache, LambdaNet, DMON-U e DMON-I, normalizados aos resultados de NetCache. Esta figura demonstra que o desempenho de DMON-U é igual ou melhor que o de DMON-I. A vantagem de DMON-U sobre DMON-I é, em média, 12%. Os dois sistemas apresentam o mesmo desempenho para FFT, LU, Raytrace, Water e WF, mas para as outras aplicações a vantagem de DMON-U está entre 5% (CG) e 42% (Radix), com uma média de 21%. Estes resultados podem ser explicados em parte pelo fato de que os sistemas baseados em atualizações exibem uma taxa de falha na cache secundária menor que em DMON-I. No entanto, as diferenças na taxa de falha não são extremamente significativas, pois as aplicações simuladas são dominadas por falhas de substituição. O fator mais importante nesta comparação é o fato de que DMON-I sofre mais de contenção na rede e memória que os outros sistemas. A contenção é mais pronunciada no sistema DMON-I devido às escritas dos blocos de cache modificados que

são jogados fora das caches secundárias, aos acessos de diretório requeridos em todos os pedidos a memória, e às mensagens extras requeridas para o re-envio de um pedido ao dono atual do bloco.

Uma comparação entre os sistemas LambdaNet e DMON-U mostra que o primeiro sistema se comporta ao menos tão bem como o segundo para todas as aplicações, mas as diferenças de desempenho são sempre pequenas, em média 6%. Os dois sistemas mostram quase o mesmo desempenho para FFT, LU, Radix, SOR e Water. Para as outras 7 aplicações, a vantagem de LambdaNet está entre 5% (Gauss) e 11% (CG e WF), com uma média de 9%. Esta diferença relativamente pequena pode parecer surpreendente devido à latência de leitura ser o maior *overhead* em todas as aplicações (exceto WF), e a latência de uma falha na cache secundária sem contenção no sistema DMON-U ser 22% maior que no sistema LambdaNet. No entanto, o sistema LambdaNet é normalmente mais propenso à contenção que DMON-U ou NetCache devido a duas características: a) as transações de leitura e escrita não estão desacopladas em LambdaNet, e b) a ausência de pontos de serialização para as atualizações dos diferentes nós gera um tráfego de atualizações enorme para LambdaNet. Desta forma, os efeitos de contenção explicam porque as diferenças de desempenho em favor de LambdaNet não são tão significativas.

Uma comparação entre os sistemas NetCache e DMON-I é claramente favorável a NetCache em todos os casos. A vantagem de desempenho de NetCache é em média 42% para as aplicações utilizadas. Para FFT e Water os dois sistemas são similares, mas para as outras 10 aplicações, a vantagem de NetCache está entre 20% (Raytrace) e 105% (WF), dando em média 50%. Os motivos principais para a imensa disparidade entre NetCache e DMON-I são dois: a) uma fração significativa das falhas de leitura na cache secundária são acertos na cache compartilhada de NetCache, o que reduz tremendamente a latência dessas operações, e b) mesmo quando a cache compartilhada é relativamente ineficiente, as latências de uma falha de leitura na cache secundária são maiores em DMON-I que em NetCache, especialmente quando a quantidade de tráfego gerado pela aplicação na rede é significativa.

NetCache também apresenta um melhor desempenho que DMON-U. A vantagem do sistema NetCache tem uma média de 27% para as aplicações simuladas. Exceto para FFT e Radix, onde as diferenças de desempenho são desprezíveis, o ganho de NetCache varia

de 7% (Water) a 99% (WF), dando uma média de 32%. Esta diferença de desempenho significativa pode ser atribuída em grande parte à habilidade da cache compartilhada para reduzir a latência média das leituras, pois as latências sem contenção de uma falha de leitura na cache secundária somente diferem 13%, e a contenção afeta os dois sistemas de forma similar.

A figura 5.3 demonstra que uma comparação entre NetCache e LambdaNet é claramente favorável a NetCache. Os tempos de execução de NetCache são em média 20% menores para as aplicações utilizadas. O desempenho dos dois sistemas é equivalente para 3 aplicações: Em3d, FFT e Radix. Para as outras 9 aplicações, a vantagem de NetCache é em média 26%, variando de 7% para SOR e Water a 41 e 79% para Gauss e WF, respectivamente. Sem considerar as 5 aplicações para as quais as diferenças de desempenho são inferiores a 10%, a vantagem média de NetCache sobre LambdaNet aumenta para 31%. Novamente, o principal motivo para estes resultados é a habilidade de NetCache para reduzir a latência média de leituras quando uma porcentagem não desprezível de falhas na cache secundária acertam na cache compartilhada. Os experimentos realizados para NetCache, sem a cache compartilhada, confirmam esta afirmação. Em geral, o desempenho de OPTNET é um pouco pior que LambdaNet (em média 1%), um pouco melhor que DMON-U (em média 4%), e bastante superior a DMON-I (em média 17%). Para avaliar a arquitetura de NetCache mais detalhadamente, a próxima subseção trata da eficiência da cache ótica. Posteriormente, serão avaliadas organizações e políticas alternativas para a cache compartilhada.

5.3.2 Eficiência da Cache Compartilhada

A figura 5.4 mostra a eficiência de NetCache como uma cache de dados, assumindo, novamente, um multiprocessador de 16 nós. Para cada aplicação é mostrado um grupo de 4 barras. A barra mais à esquerda representa a latência de leitura como uma porcentagem do tempo total de execução em NetCache sem a cache compartilhada. As outras barras apresentam resultados de NetCache com 32 KBytes de cache compartilhada, da esquerda para a direita: a taxa de acerto na cache compartilhada, a porcentagem de redução na latência de uma falha na cache secundária, e a porcentagem de redução na latência de leitura total.

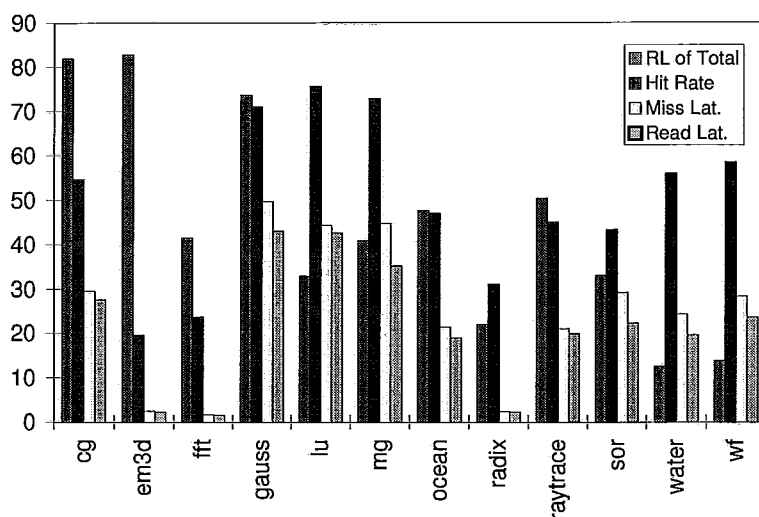


Figura 5.4: Porcentagem da Latência de Leitura no Tempo de Execução, Taxa de Acerto (em Porcentagem) na Cache Compartilhada, Redução da Latência de Leitura em uma Falha e Redução da Latência de Leitura Total

A figura demonstra que, para unicamente 3 aplicações (Radix, Water e WF), a latência de leitura é uma fração pequena do tempo total de execução em um multiprocessador com NetCache, mas sem a cache compartilhada. A latência de leitura é uma fração significativa nas outras 9 aplicações. É justamente nessas aplicações com maior fração de latência de leitura que a cache compartilhada apresenta uma maior contribuição. No entanto, como se observa na figura, nem todas as aplicações exibem taxas de acerto elevadas quando é assumida uma cache de 32 KBytes.

Resumindo, os resultados das taxas de acerto dividem as aplicações em 3 grupos. O primeiro (chamado de *pouco-reuso*) inclui as aplicações com uma insignificante reutilização de dados na cache compartilhada. Neste grupo estão Em3d, FFT e Radix, para as quais menos de 32% das falhas na cache secundária acertam na cache compartilhada. Como resultado disso, as reduções na latência de uma falha na cache secundária e na latência total de leitura são quase desprezíveis para estas aplicações. O segundo grupo de aplicações (*muito-reuso*) exibe uma significativa reutilização dos dados na cache compartilhada. Três aplicações pertencem a este grupo: Gauss, LU e Mg. As suas taxas de acerto são bastante altas, aproximadamente 70%, o que produz uma redução na latência de uma falha na cache secundária e na latência total de leitura de pelo menos 35%. As outras 6 aplicações (CG, Ocean, Raytrace, SOR, Water e WF) formam o terceiro grupo (*moderado-reuso*) com taxas de acerto intermédias e reduções nas latências de leitura na

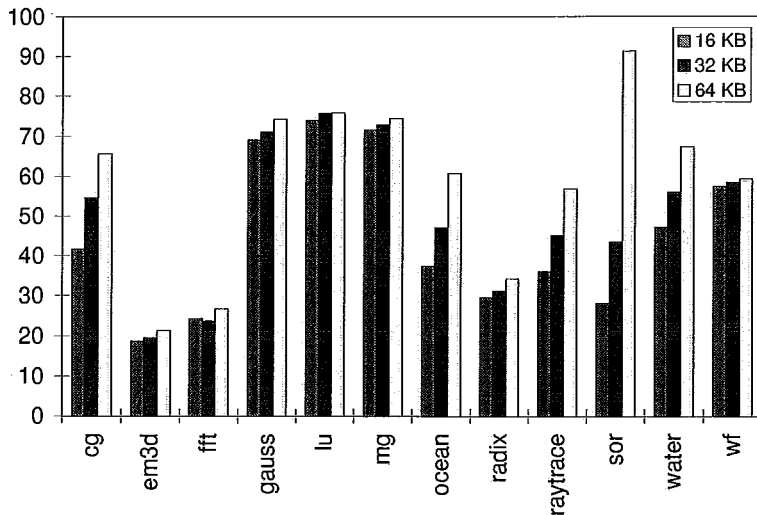


Figura 5.5: Taxas de Acerto (em Porcentagem) para uma Cache Compartilhada de 16, 32 e 64 KBytes

ordem de 20 a 30%. Estes resultados sugerem que NetCache não poderia melhorar o desempenho de LambdaNet para Em3d, FFT, Radix e Water. Estas aplicações simplesmente não se beneficiam da cache compartilhada, seja porque as suas taxas de acerto são baixas ou porque a latência das leituras não é um parâmetro importante no desempenho destas aplicações. Por outro lado, estes resultados também sugerem que WF não devia se beneficiar de NetCache, o que as simulações não confirmam. O motivo para este resultado (aparentemente) surpreendente é que a redução na latência de leitura conseguido por NetCache tem um importante efeito colateral: melhora o comportamento das sincronizações em 7 das aplicações executadas (CG, LU, Mg, Ocean, Radix, Water e WF). Para WF, em particular, uma cache compartilhada de 32 KBytes melhora o *overhead* das sincronizações em 56%, aliviando a falta de balanceamento exposta durante as barreiras da aplicação.

5.3.3 Avaliação de Diferentes Organizações e Políticas de Substituição

A seguir serão avaliadas diversas organizações da cache compartilhada e diferentes políticas de substituição. Em particular, será estudado o tamanho da cache compartilhada, o tamanho do bloco da cache ótica, a sua associatividade, e a política de substituição da cache compartilhada.

Tamanho da Cache Compartilhada. A figura 5.5 apresenta as taxas de acerto em NetCache para 16, 32 e 64 KBytes de cache compartilhada em um multiprocessador com

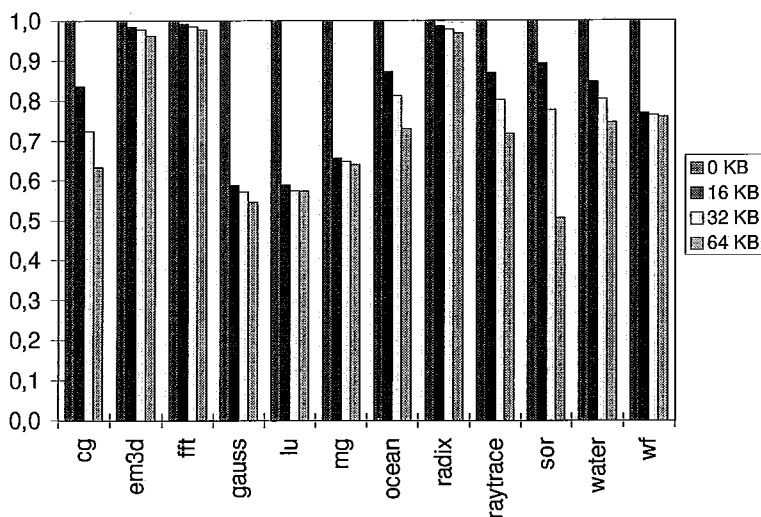


Figura 5.6: Latências de Leitura (com Relação ao MP OPTNET) em uma Cache Compartilhada de 16, 32 e 64 KBytes

16 nós. O tamanho da cache compartilhada foi alterado mudando o número de *cache-channels*. Mais especificamente, uma cache compartilhada de 16 KBytes é implementada com 64 *cache-channels*, e uma cache de 64 KBytes é implementada com 256 *cache-channels*.

A figura 5.5 mostra que as taxas de acerto das aplicações nos grupos *pouco-reuso* e *muito-reuso* não são afetadas pelo tamanho da cache. No caso das aplicações com pouco reuso, o motivo para este comportamento é que a cache compartilhada é muito pequena para manter o conjunto de trabalho de todos os nós. A insensibilidade das aplicações com muito reuso ao tamanho da cache tem um motivo diferente: uma cache compartilhada de 16 KBytes já é suficiente para manter a maior parte do conjunto de trabalho, assim, os aumentos de tamanho produzem apenas impactos pequenos nas taxas de acerto. As aplicações no grupo *moderado-reuso* mostram os resultados mais interessantes. As taxas de acerto destas aplicações melhoram significativamente com os aumentos no tamanho da cache compartilhada, exceto para WF. O motivo desta exceção é que o tamanho da cache compartilhada é insignificante comparado com o tamanho do conjunto de trabalho de WF. Esta aplicação não se comporta exatamente como as do grupo *pouco-reuso* devido às suas excelentes propriedades de localidade espacial.

As figuras 5.6 e 5.7 mostram as latências de leitura e os tempos de execução, respectivamente, para cada tamanho de cache compartilhada. Note que as figuras também incluem uma barra (a mais à esquerda) correspondente a um multiprocessador com Net-

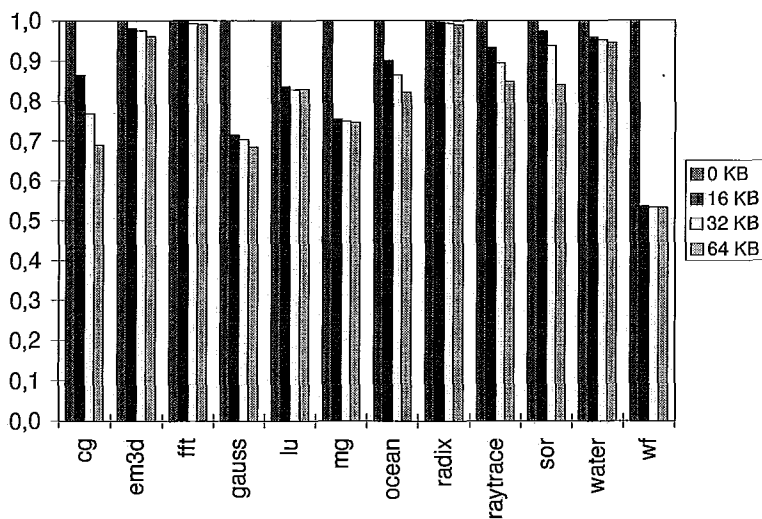


Figura 5.7: Tempos de Execução (com Relação ao MP OPTNET) em uma Cache Compartilhada de 16, 32 e 64 KBytes

Cache sem a cache compartilhada. Os dados da figura estão normalizados aos resultados do multiprocessador sem a cache compartilhada. A mais importante observação que pode ser feita da figura 5.6 é que o uso da cache compartilhada reduz significativamente a latência de leitura para a maioria das aplicações (grupos *moderado-reuso* e *muito-reuso*). As reduções na latência de leitura para estas aplicações pode chegar a 50%, como é o caso de SOR com uma cache compartilhada de 64 KBytes, sendo em média 28% para uma cache compartilhada de 32 KBytes. Além disso, como era esperado, as reduções na latência de leitura conseguidas pelos diferentes tamanhos de cache têm as mesmas tendências da figura 5.5.

O efeito desses benefícios também é notório no desempenho total do sistema. A figura 5.7 demonstra que o uso de uma cache compartilhada melhora os tempos de execução significativamente, exceto para Em3d, FFT, Radix e Water. Os ganhos de desempenho alcançados por WF são particularmente significativos, 47%. Uma comparação dos resultados para os diferentes tamanhos de cache justifica a seleção de uma cache compartilhada de 32 KBytes como a nossa arquitetura base, pois este tamanho apresenta uma melhor relação custo/desempenho que os outros sistemas.

Tamanho de Bloco da Cache Compartilhada. Também avaliamos o tamanho do bloco da cache compartilhada, mantendo constante o tamanho da cache em 32 KB. Lembre-se que o tamanho base é 64 Bytes, o menor possível considerando que o bloco da cache

secundária é também de 64 Bytes. Os experimentos mostram que um aumento no tamanho do bloco, para tirar vantagem da localidade dos programas, é uma má decisão. Para as aplicações com pouca localidade espacial, tal aumento de tamanho simplesmente aumenta a quantidade de poluição da cache compartilhada. Por exemplo, o *overhead* no desempenho associado a um bloco de 128 Bytes alcança 33% para Em3d e 12% para CG. Para algumas aplicações estes *overheads* não são muito acentuados, mas não existe nenhuma vantagem na utilização de blocos maiores que 64 Bytes.

O tamanho de bloco da cache compartilhada segue as mesmas tendências observadas nas caches tradicionais: um aumento no tamanho do bloco normalmente degrada o desempenho quando a cache é relativamente pequena, ou quando, a diminuição da taxa de falhas obtida por blocos maiores não compensa a sua maior latência de procura.

Associatividade da Cache Compartilhada. Atualmente, a arquitetura de NetCache determina que um certo bloco só pode ser encontrado em um canal específico, i.e., os blocos são diretamente mapeados aos canais. Além disso, a arquitetura determina que um bloco pode estar em qualquer lugar dentro do canal, i.e., os *cache-channels* são totalmente associativos. Esta organização simplifica bastante o *hardware* da subrede em anel por duas razões. A primeira é que o nó que falha no acesso a um bloco não requer mais de um receptor sintonizável. A segunda razão é que o *home* não precisa se preocupar com endereços de linhas de cache específicos dentro do canal, o *home* simplesmente insere o bloco na primeira linha de cache utilizável.

Para avaliar como seria possível melhorar o desempenho de NetCache mudando a associatividade da cache compartilhada, realizamos experimentos onde os *cache-channels* não são diretamente mapeados, i.e., um certo bloco deve ser colocado em uma linha da cache compartilhada específica dentro do seu *cache-channel*. Mudar a alocação dos blocos aos canais não é uma opção, pois isso envolve um *hardware* mais complexo.

A figura 5.8 mostra as taxas de acerto de uma cache compartilhada de 32 KBytes como função da associatividade de cada *cache-channel*. A barra à esquerda de cada grupo representa a organização padrão de NetCache (“Fully”), enquanto que a barra à direita representa a opção de ter *cache-channels* diretamente mapeados (“Direct”). A figura mostra que os *cache-channels* diretamente mapeados alcançam taxas de acerto muito pe-

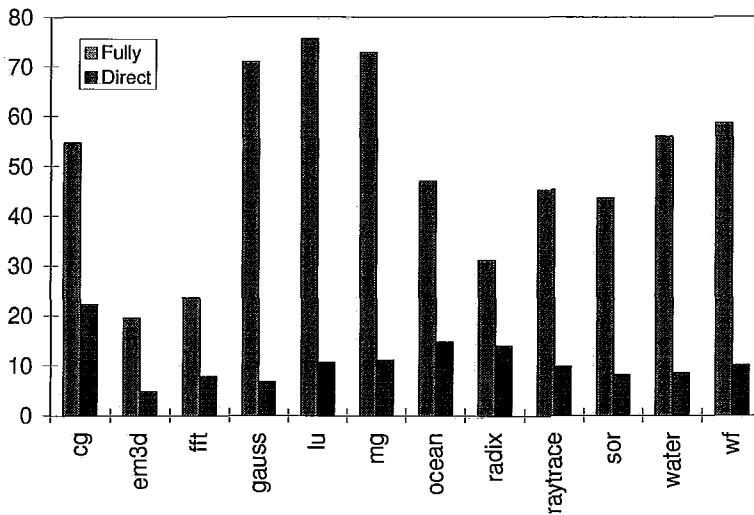


Figura 5.8: Taxas de Acerto (em Porcentagem) para a Cache Compartilhada como Função da Associatividade de Cada Canal

quenas em todos os casos. As taxas de acerto nunca são maiores que 25%. Assim, estes resultados justificam claramente a seleção feita para a arquitetura de NetCache.

A associatividade da cache compartilhada também segue as mesmas tendências observadas nas caches tradicionais, onde as caches totalmente associativas apresentam taxas de acerto maiores que as diretamente mapeadas.

Política de Substituição na Cache Compartilhada. Outro aspecto importante na arquitetura de NetCache é a política de substituição de blocos na cache compartilhada. A arquitetura determina que um bloco aleatório (o bloco contido na próxima linha de cache que passe na frente do nó) no correspondente *cache-channel* deve ser substituído quando um novo bloco vai ser inserido na cache compartilhada. Novamente, esta política simplifica o *hardware* da subrede em anel, pois a interface da subrede não precisa manter nenhuma informação adicional. Além disso, a política padrão otimiza a operação de substituição, já que um novo bloco é inserido na cache compartilhada tão logo quanto for possível.

Para avaliar como seria possível melhorar o desempenho de NetCache usando uma melhor política de substituição, realizamos experimentos assumindo políticas LFU (*Least Frequently Used*), LRU (*Least Recently Used*) e FIFO (*First In, First Out*).

A figure 5.9 apresenta as taxas de acerto em uma cache compartilhada de 32 KBytes como função da política de substituição. Da esquerda para a direita, as barras em cada

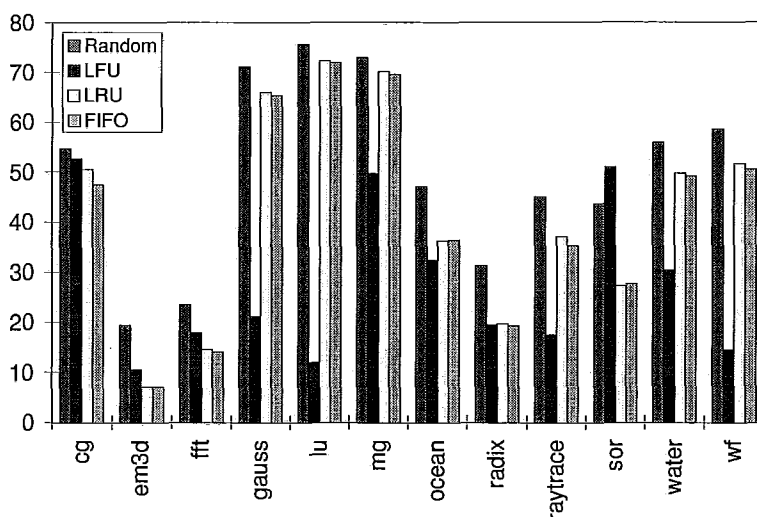


Figura 5.9: Taxas de Acerto (em Porcentagem) para a Cache Compartilhada como Função da Política de Substituição

grupo representam a política aleatória, atualmente aplicada por NetCache, e as políticas LFU, LRU e FIFO. A figura mostra que a política aleatória alcança taxas de acerto maiores em todos os casos, exceto SOR. Para certas aplicações, como Em3d, Ocean e Radix, a superioridade da política aleatória sobre as outras é substancial. As políticas LRU e FIFO atingem um desempenho comparável, mas as vezes significativamente pior que a política aleatória. Embora LFU apresente o melhor resultado para SOR, ela se comporta pobremente em relação às outras políticas em alguns casos (Gauss, LU, Mg, Raytrace, Water e WF). Novamente, estes resultados justificam a seleção da política de substituição para NetCache.

Estes resultados das políticas de substituição na cache compartilhada são um tanto surpreendentes, pois estratégias mais sofisticadas apresentam taxas de acerto menores que a estratégia aleatória. A política LRU normalmente atinge bons resultados nas caches tradicionais, a cache compartilhada se comporta de modo diferente por duas razões: a) cada *cache-channel* só pode armazenar 4 blocos, assim a estratégia aleatória tem uma probabilidade razoável de selecionar o melhor, e o mais importante, b) todos os processadores do sistema inserem blocos na cache compartilhada, não unicamente o processador local conectado à cache tradicional. Com estas características e as diferentes temporizações dos acessos a memória, uma política do tipo LRU deixa de ter sentido.

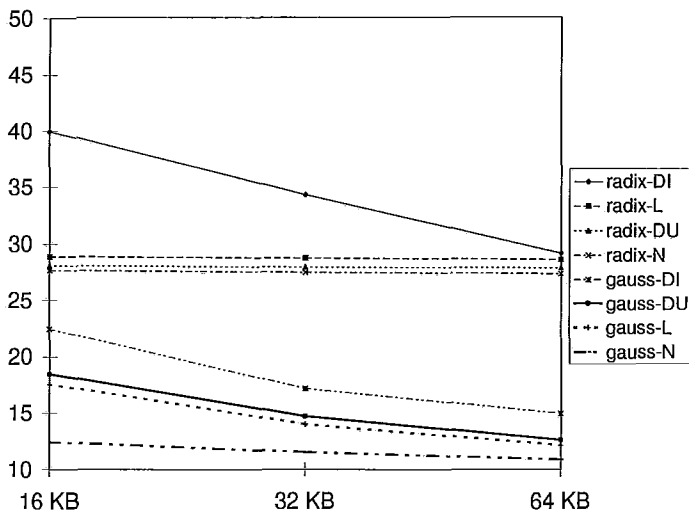


Figura 5.10: Tempos de Execução (em Milhões de Ciclos) como Função do Tamanho da Cache Secundária

5.3.4 Impacto dos Parâmetros Arquiteturais

Nesta subseção será avaliado o impacto de algumas das suposições feitas nas simulações para poder entender melhor o comportamento de NetCache. Inicialmente será feito um estudo com diferentes tamanhos da cache secundária, logo será realizada uma avaliação do impacto da taxa de transmissão, e finalmente será tratado o efeito da latência de leitura das memórias. Para simplificar a análise, os resultados estarão concentrados em uma aplicação representativa de cada um dos grupos *muito-reuso* e *pouco-reuso*, Gauss e Radix, respectivamente.

Tamanho da Cache Secundária. A figura 5.10 apresenta os tempos de execução dos sistemas NetCache (“N”), LambdaNet (“L”), DMON-U (“DU”) e DMON-I (“DI”), com 16 nós, para Gauss e Radix como função do tamanho da cache secundária. Lembre-se que os experimentos base assumem uma cache secundária de 16 KBytes. Todos os resultados assumem uma cache compartilhada de 32 KBytes.

Como era esperado, um aumento no tamanho da cache secundária reduz os benefícios potenciais da cache compartilhada, pois este aumento de tamanho produz uma redução na taxa de falhas de leitura. Isto é exatamente o que sucede com Gauss, uma aplicação com boa localidade. No entanto, a vantagem de NetCache permanece considerável em todos os casos, ainda para caches secundárias 4 vezes maiores que as do experimento base.

Por outro lado, os aumentos do tamanho da cache secundária não afetam significativa-

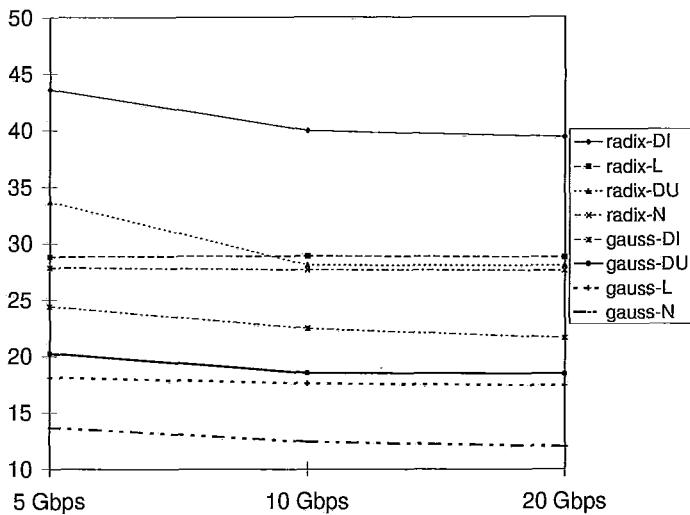


Figura 5.11: Tempos de Execução (em Milhões de Ciclos) como Função da Taxa de Transmissão

mente a taxa de falhas em Radix, já que esta aplicação exibe uma localidade extremamente baixa. Desta forma, somente DMON-I melhora o tempo de execução de Radix com esse aumento de tamanho. Esse ganho de desempenho é uma consequência da redução na contenção quando um relativamente pequeno número de falhas e escritas a memória são evitadas.

Finalmente, note que tanto para Gauss como para Radix, um aumento de 4 vezes na quantidade da cache secundária dos multiprocessadores baseados em LambdaNet e DMON não é suficiente para ganhar de NetCache com uma cache compartilhada de 32 KBytes e uma cache secundária de 16 KBytes. Este resultado mostra o fato de que esses outros sistemas requerem uma grande quantidade de cache eletrônica adicional para atingir os benefícios de uma cache compartilhada de 32 KBytes.

Taxas de Transmissão. A figura 5.11 apresenta o tempo de execução dos sistemas NetCache, LambdaNet, DMON-U e DMON-I como função da taxa de transmissão ótica. Lembre-se que os experimentos base assumem uma taxa de 10 Gbits/s. Todos os resultados de NetCache assumem uma cache compartilhada de 32 KBytes, i.e., o comprimento da subrede em anel é ajustado inversamente com as variações da taxa de transmissão. Por exemplo, duplicando a taxa de transmissão é necessário diminuir pela metade o comprimento do anel.

A figura mostra que as aplicações sofrem uma perda significativa de desempenho nos

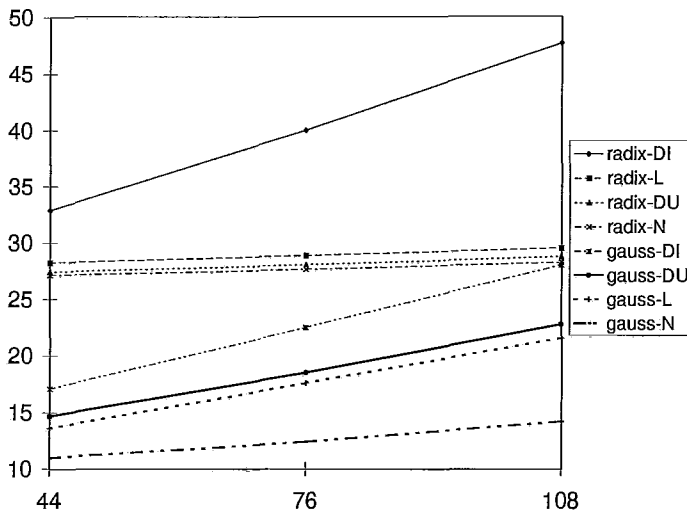


Figura 5.12: Tempos de Execução (em Milhões de Ciclos) como Função da Latência de Leitura na Memória (em Ciclos de Processador)

multiprocessadores baseados em DMON com uma taxa de transmissão de 5 Gbits/s, especialmente quando as aplicações exibem pouca localidade. As perdas experimentadas por NetCache e LambdaNet não são significativas. No entanto, decrementos na taxa de transmissão têm um impacto negativo nos benefícios potenciais da cache compartilhada, especialmente para aplicações do grupo *muito-reuso*, já que a diferença percentual entre uma falha e um acerto na cache compartilhada é reduzida. Por exemplo, a 5 Gbits/s um acerto de leitura na cache compartilhada demora 68 ciclos, enquanto que uma falha demora 140 ciclos, um fator de 2. A 10 Gbits/s, uma falha de leitura na cache compartilhada demora 2,6 vezes mais do que um acerto.

No entanto, a evolução tecnológica tende a aumentar as taxas de transmissão e não a diminuí-las. Aumentos nas taxas de transmissão têm um impacto maior nos tempos de execução de NetCache que nos outros sistemas, especialmente para aplicações com uma significativa reutilização dos dados na cache compartilhada.

Latência de Leitura de Memória. A figura 5.12 apresenta os tempos de execução dos multiprocessadores baseados em NetCache, LambdaNet e DMON como função da latência de leitura de um bloco na memória. Lembre-se que a latência de leitura de um bloco na memória assumida para os experimentos base foi de 76 ciclos de processador. Os resultados de NetCache assumem uma cache compartilhada de 32 KBytes.

A figura mostra alguns resultados interessantes, o mais importante é que os aumentos

na latência elevam os tempos de execução de NetCache muito menos que nos outros sistemas. Este comportamento é, na verdade, uma das grandes vantagens do sistema NetCache, já que os aumentos na latência aumentam os benefícios potenciais alcançáveis pela cache compartilhada.

5.3.5 Comparação com Outros Sistemas

Para finalizar a seção de resultados é importante comparar a arquitetura de NetCache com outros tipos de sistemas, pelo menos em termos qualitativos.

Uma comparação direta entre NetCache e um multiprocessador com uma rede eletrônica tradicional não seria necessária devido à significativa disparidade de desempenho existente entre este último e um multiprocessador com uma rede ótica. A disparidade é produzida por alguns fatores como a largura de faixa da rede, a possibilidade de ter canais independentes entre os diferentes conjuntos de nós, e a possibilidade de usar uma coerência baseada em *snooping*.

Poderia se pensar que, ao invés de NetCache, seria mais benéfico usar uma rede ótica simples (como OPTNET) e aumentar o tamanho da cache secundária. Mesmo que caches secundárias maiores possam melhorar o desempenho de certas aplicações, o suficiente para ganhar de NetCache com caches secundárias pequenas, a quantidade de cache adicional, necessária para isto acontecer, pode ser bastante significativa, como demonstrado na subseção anterior. Além disso, ainda em sistemas com caches secundárias grandes, a maioria de aplicações deve ainda se beneficiar de uma cache compartilhada de baixo nível, especialmente quando os custos da tecnologia ótica diminuem e as taxas de transmissão aumentem.

Da mesma forma, é possível conceber uma organização onde uma cache eletrônica adicional é colocada do lado da memória com uma rede ótica mais simples. Estas caches de memória seriam compartilhadas por todos os nós da máquina e adicionariam um novo nível na hierarquia. Esse novo nível retardaria os acessos a memória que não possam ser satisfeitos pelas caches adicionais. NetCache também adiciona um novo nível na hierarquia de memória, mas este nível não retarda os acessos que falham nele.

5.4 Trabalhos Relacionados

A arquitetura de NetCache é a única do seu gênero que se conhece até o momento. Uma mistura de rede e cache nunca foi proposta e/ou avaliada para multiprocessadores. No entanto, algumas poucas áreas estão relacionadas com esta pesquisa, por exemplo a utilização de redes óticas em máquinas paralelas, o uso de linhas de retardo óticas e a utilização de caches compartilhadas em multiprocessadores.

Memórias de linhas de retardo têm sido implementadas em sistemas de comunicação ótica [34] e em computadores totalmente óticos [36]. Estas memórias temporais apresentam um tempo de acesso e capacidade de armazenamento proporcional ao comprimento do canal e à taxa de transmissão dos dados [4]. As memórias de linha de retardo ótica não foram utilizadas como caches nem aplicadas a multiprocessadores, embora as redes óticas sejam parte de alguns projetos de computação paralela [17, 10], como mostrado nos capítulos 2 e 4. As únicas características da comunicação ótica que estes projetos exploram são a sua grande largura de faixa e a capacidade de disseminação a uma grande quantidade de nós. Além destas características, NetCache também explora o potencial armazenamento de dados nas redes óticas.

Alguns estudos têm proposto a utilização de caches compartilhadas para multiprocessadores formados de um ou vários *chips*, e.g., [46, 47, 48, 49, 50, 51]. Em termos de multiprocessadores de um único *chip*, Nayfeh e Olukotun [46] têm proposto a construção de grupos de processadores compartilhando uma grande cache, mas sem incluir caches individuais para cada processador. Em [47] é discutido o projeto de um grupo de processadores em um único *chip* com caches locais e uma cache secundária compartilhada. [48] mostra resultados para uma cache compartilhada infinita e totalmente associativa, e compara este sistema com um outro que tenha caches individuais para cada processador.

Outros estudos consideraram caches de rede para um multiprocessador de vários *chips*. Cada nó do sistema ASURA [49] é um grupo de processadores com uma interface de rede que contém parte do espaço de endereçamento global e cacheia dados remotos. Da mesma forma, Bennett *et al.* [50] avaliam os benefícios de adicionar uma cache compartilhada à interface de rede de um grupo de processadores como um meio de melhorar o desempenho de estações de trabalho configuradas como um multiprocessador. Final-

mente, Stache [51] implementa uma alternativa totalmente associativa e controlada por *software* para uma cache de rede.

Embora NetCache evite os acessos a memória principal como uma cache compartilhada ou cache de rede, este trabalho se diferencia dos anteriores em alguns pontos: a) NetCache armazena dados de todos os nós, não só dos que podem caber em um único *chip*, b) NetCache armazena dados sem redundância, enquanto que caches de rede isoladas podem gastar espaço armazenando múltiplas cópias dos dados, c) além de armazenar dados, NetCache serve também como uma rede de comunicação, e d) NetCache é baseado em ótica, não em eletrônica.

5.5 Conclusões

Neste capítulo foi proposto um novo conceito arquitetural: uma mistura rede/cache ótica chamada de NetCache. Através de um grande conjunto de simulações mostramos que um multiprocessador baseado em NetCache supera facilmente a outros sistemas baseados em redes óticas, especialmente quando as aplicações têm uma grande quantidade de reutilização dos dados. Adicionalmente, as decisões de projeto de NetCache foram justificadas mediante uma extensa avaliação da organização e políticas usadas pela cache compartilhada. Finalmente, o estudo da variação de parâmetros mostra que os benefícios de NetCache podem ser aumentados no futuro. Baseados nesses resultados e nos decrescentes custos dos componentes óticos, a principal conclusão é que a arquitetura de NetCache é altamente eficiente e deve ser definitivamente considerada pelos projetistas de multiprocessadores.

Capítulo 6

OWCache

Algumas aplicações acessam mais dados do que cabe na memória principal. Nessas aplicações, chamadas de *out-of-core*, a redução dos custos de acesso ao disco é um dos principais problemas de desempenho a serem resolvidos. Por este motivo, os programadores desse tipo de aplicações normalmente as codificam com chamadas explícitas de entrada/saída. No entanto, escrever aplicações com chamadas explícitas de entrada/saída apresenta algumas desvantagens [52]: a programação se converte frequentemente em uma tarefa bastante difícil [53]; as chamadas ao sistema para entrada/saída envolvem custos de cópia entre as áreas do usuário e do sistema operacional; e o código resultante não é sempre portátil (do ponto de vista de desempenho) entre máquinas com diferentes configurações de memória e/ou recursos, por exemplo diferentes quantidades de memória ou latências de entrada/saída. Em contraste com o estilo de programação com entrada/saída explícita, acreditamos que as aplicações *out-of-core* devem se basear exclusivamente nos mecanismos de memória virtual e que os custos de acesso ao disco devem ser atenuados pelo mesmo sistema que executa tais aplicações. A nossa preferência pela entrada/saída baseada em memória virtual é análoga a favorecer memória compartilhada ao invés de troca de mensagens como um modelo de programação mais apropriado para a programação paralela.

Basicamente, a entrada/saída baseada em memória virtual envolve leitura de páginas da memória e escrita de páginas para o disco (*swap-outs*). As leituras de páginas podem geralmente ser tratadas com eficiência através do *prefetching* dinâmico dos dados para a memória principal (ou para a cache do controlador de disco), conforme mostra [54, 55]. Nos casos onde o *prefetching* dinâmico não é muito eficiente por si só, ele pode ser refi-

nado através do envolvimento do compilador [52] ou através de indicações dos possíveis acessos futuros [56]. As escritas ao disco, no entanto, são mais difíceis de se otimizar, mesmo quando elas acontecem fora do caminho crítico da computação. O problema de desempenho das escritas ao disco está relacionado com a sua concentração ao longo de pequenos intervalos de tempo. Como resultado deste comportamento, o sistema operacional deve sempre reservar um número relativamente grande de *frames* livres para evitar que o processador pare, esperando que as operações de escrita anteriores terminem. Na verdade, quanto mais eficiente é a técnica de *prefetching*, maior é o número de *frames* livres que o sistema operacional deve reservar. Esta situação é especialmente problemática nos multiprocessadores escaláveis, onde nem todos os nós são capazes de realizar operações de entrada/saída (e.g., [57, 58]), pois tanto as limitações de latência como as de largura de faixa do disco retardam estas escritas.

Assim, este capítulo propõe uma extensão simples à rede de interconexão ótica de um multiprocessador escalável para otimizar as escritas ao disco (*swap-outs*). Mais especificamente, esta proposta estende a rede de interconexão com um anel ótico que não somente transfere páginas entre as memórias locais e os discos do multiprocessador, como também atua como uma grande cache compartilhada para as escritas ao disco. Quando existe espaço na cache de disco, as páginas a serem escritas são copiadas do anel ótico à cache, de forma que as páginas escritas por um mesmo nó são copiadas em conjunto. A rede estendida proporciona vários benefícios de desempenho ao sistema: ela provê uma área adicional onde as páginas a serem escritas podem permanecer até que o disco esteja livre; ela aumenta a possibilidade de combinar várias escritas ao disco; e ela atua como uma *victim-cache* para as páginas que saem da memória e posteriormente são acessadas pelo mesmo processador ou por um processador diferente.

Com a finalidade de determinar como estes benefícios afetam o desempenho do sistema, o anel ótico foi avaliado como uma extensão da rede OPTNET [9]. A extensão adiciona basicamente um anel ótico com vários canais WDM, chamados de *cache-channels*, à rede OPTNET. Estes *cache-channels* são utilizados para armazenar os *swap-outs*. A combinação de OPTNET com o anel ótico é chamada de OWCACHE [8].

Para mostrar que o anel ótico pode também ser aplicado com sucesso a multiprocessadores tradicionais, onde os processadores são interconectados através de redes ele-

trônicas, determinamos quais seriam os seus benefícios em um multiprocessador conectado por uma grade eletrônica. Neste caso, o anel ótico é projetado como uma simples extensão ao subsistema de entrada/saída do multiprocessador. A diferença entre OWCACHE e esta extensão, chamada de NWCACHE [12], está no fato de que NWCACHE não requer modificações no *hardware* do multiprocessador e é mais flexível e modular, pois a interface NWCACHE pode ser inserida no barramento de entrada/saída dos nós de qualquer multiprocessador.

6.1 Uma Cache Ótica para Escritas

Nesta seção descrevemos a arquitetura básica do multiprocessador e o sistema operacional considerados, assim como a arquitetura e uso do anel ótico que implementa OWCACHE.

6.1.1 Arquitetura do Multiprocessador e Gerência da Memória Virtual

A arquitetura básica do multiprocessador considerado corresponde a um multiprocessador escalável com coerência de caches, onde os processadores são interconectados através de uma rede OPTNET ligeiramente modificada. A ligeira modificação melhora o desempenho de OPTNET na presença de tráfego gerado pelos *swap-outs*. Esta modificação se resume ao incremento de um quarto canal de disseminação, chamado de *swap-channel*, que cuida de todo o tráfego produzido pelos *swap-outs*. Da mesma forma que os canais de coerência em OPTNET, o *swap-channel* usa TDMA com intervalos de tempo variável para controle de acesso ao meio. A inclusão de um canal extra requer a adição de um outro receptor fixo e de um outro transmissor fixo por nó, resultando em uma complexidade total de *hardware* de $9 \times p$ componentes óticos.

Como se observa na figura 6.1, cada nó do sistema inclui um processador (“ μP ”), uma TLB (*Translation Lookaside Buffer*), um *write-buffer* (“WB”), caches de primeiro (“L1”) e segundo (“L2”) níveis, memória local (“LM”) e uma interface de rede (“NI”). Cada nó capaz de realizar operações de entrada/saída também inclui um disco e o seu controlador conectado ao barramento de entrada/saída.

A única parte do sistema operacional que consideraremos é a referente ao código para gerência da memória virtual. Novamente, aqui será assumida uma estratégia convencio-

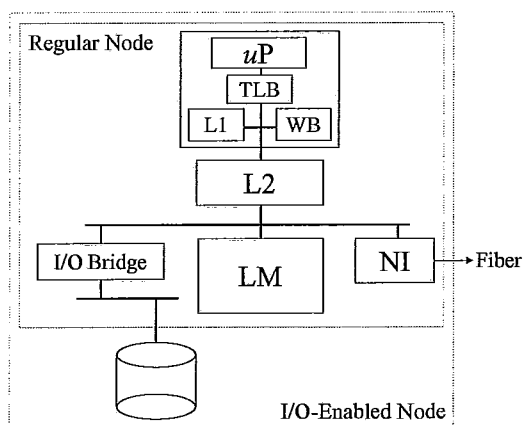


Figura 6.1: Arquitetura dos Nós

nal. Mais especificamente, o sistema base implementa uma tabela de páginas única para toda a máquina, onde cada uma de suas entradas é acessada pelos diferentes processadores com exclusão mútua. Cada vez que as permissões de acesso a uma página são restringidas, uma operação *TLB-shutdown* é executada, sendo todos os processadores interrompidos para apagar as entradas dessa página na sua TLB.

O sistema operacional mantém um conjunto mínimo de *frames* livres por nó dentro do multiprocessador. Em uma falta de página, o sistema operacional envia um pedido para essa página¹ ao disco correspondente através do canal de pedidos de OPTNET (assume-se que as páginas estão armazenadas em grupos de 32 páginas consecutivas e que o sistema de arquivos paralelo associa cada um destes grupos a um disco diferente de modo *round-robin*.) Para cada pedido, o controlador de disco lê a página da sua cache (acerto na cache) ou disco (falha na cache) e a envia ao solicitante através do seu correspondente *home-channel*. Isto é, uma operação de pedido de página é similar ao pedido de um bloco de memória no caso da rede OPTNET.

Quando a página que produziu a falta chega ao nó, a tabela de páginas global é atualizada, permitindo que outros processadores acessem remotamente os dados dessa página. Se a chegada desta página reduz o número de *frames* livres no nó para menos de um valor mínimo, o sistema operacional usa LRU (*Least Recently Used*) para selecionar a página a ser substituída. Se a página foi modificada, uma operação de *swap-out* é iniciada. Caso contrário, o *frame* da página é simplesmente liberado.

¹Por simplicidade de apresentação, daqui em diante não haverá nenhuma distinção entre uma página de memória virtual e um bloco de disco

Uma página que é retirada da memória é enviada à cache de disco correspondente através do *swap-channel*. O controlador de disco responde a essa mensagem com um ACK, se ele foi capaz de colocar a página na sua cache. As escritas têm preferência sobre os *prefetches* na cache. O ACK permite que o nó que realizou o *swap-out* reutilize o espaço ocupado pela página na memória. O controlador de disco responde com um NACK, se não existe espaço livre na sua cache (i.e., a cache do controlador de disco está cheia de páginas a serem escritas ao disco). O controlador armazena os NACKs em uma fila FIFO. Quando é liberado algum espaço na cache do controlador, o controlador envia uma mensagem OK ao nó na cabeça da fila, o que permite que o nó correspondente reenvie a página. As mensagens de ACK, NACK, e OK são enviadas através do canal de pedidos de OPTNET.

O *prefetching* de páginas não está dentro do âmbito deste estudo. Assim, serão considerados os dois extremos de *prefetching*: *prefetching* ótimo e *prefetching* básico. O *prefetching* ótimo tenta se aproximar do desempenho atingido por compiladores altamente sofisticados [52] ou indicações dadas pelas aplicações [56] que podem buscar antecipadamente os dados do disco para as caches ou memórias locais. A técnica ideal assume que todos os pedidos de páginas podem ser satisfeitos diretamente da cache de disco, i.e., todos os acessos de leitura ao disco são executados fora do caminho crítico dos pedidos de leitura.

Sob o cenário do *prefetching* básico, só ocorre *prefetching* durante uma falha na cache de disco. Nesse caso, o controlador preenche a sua cache com as páginas que se seguem seqüencialmente à página que produziu a falta. Esta técnica é básica por três razões: a) os arquivos estão distribuídos através de vários discos; b) blocos distintos, localizados em um mesmo disco, podem ser acessados concorrentemente por vários nós; e c) algumas aplicações não acessam as suas páginas seqüencialmente.

6.1.2 OWCACHE = OPTNET + Anel Ótico

Sistema Base. A figura 6.2 mostra o sistema OWCACHE: uma rede OPTNET básica² e um anel ótico. O anel é utilizado somente para transferir as páginas que foram recentemente ejetadas da memória pelos diferentes nós do multiprocessador e também para armazená-

²O anel elimina a necessidade do *swap-channel*

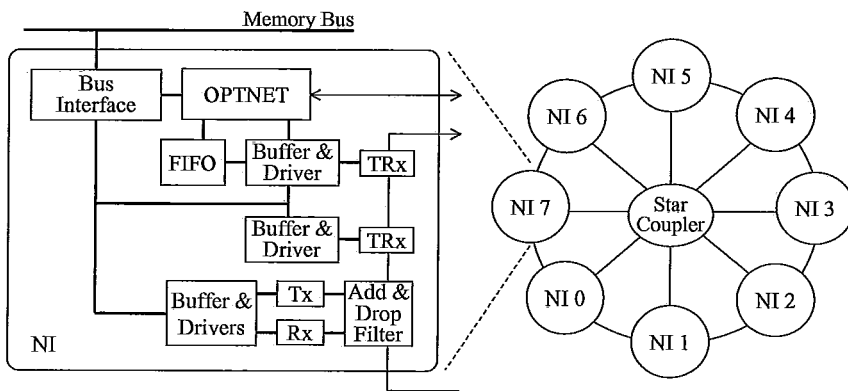


Figura 6.2: Arquitetura de OWCACHE

las na própria rede. Qualquer outro tráfego flui através dos canais regulares de OPTNET como foi descrito na seção 6.1.1.

Os *swap-outs* são continuamente enviados em volta dos *cache-channels* em uma única direção. Ou seja, o anel atua como uma cache de escritas, mantendo as páginas ejetadas da memória, até que exista espaço suficiente na cache do controlador de disco. Se a página é solicitada novamente enquanto está armazenada no anel ótico, ela pode ser remapeada na memória principal e retirada do anel.

A capacidade de armazenamento do anel é totalmente independente dos tamanhos individuais ou combinados das memórias locais. Ela é simplesmente proporcional ao número de canais utilizáveis e à largura de faixa e comprimento dos canais.

Gerência do Anel. Cada *cache-channel* transfere e armazena as páginas ejetadas por um único nó. Uma página pode ser enviada ao anel se existe espaço livre no *cache-channel* associado ao nó. Um *swap-out* para o anel ótico permite que o nó correspondente reutilize o espaço ocupado pela página em memória. No momento do *swap-out*, o nó deve ligar um bit (o bit *Ring*) na entrada correspondente da tabela de páginas, indicando que essa página está armazenada no anel ótico. O nó que iniciou a operação de *swap-out* deve também enviar uma mensagem à interface OWCACHE do nó com capacidade de entrada/saída correspondente. Esta mensagem é enviada através do canal de pedidos de OPTNET e inclui apenas o número da página ejetada da memória. A interface OWCACHE remota, então, armazena o número do nó que ejetou a página da sua memória (obtido implicitamente da mensagem) e o número de página em uma fila FIFO associada ao correspondente *cache-channel*.

Cada vez que o controlador de disco ligado ao nó tem espaço para uma outra página na sua cache, a interface de rede procura os *cache-channels* mais carregados e copia tantas páginas quanto possível do anel para a cache de disco. Depois que a página é enviada à cache do controlador de disco, um ACK é enviado ao nó que originalmente ejetou essa página. O ACK é usado pelo nó para que este possa reutilizar o espaço ocupado pela página no anel e apagar o bit *Ring* associado à página.

Existem duas características importantes na cópia de páginas do anel ótico para o disco que aumentam a localidade espacial das diferentes escritas na cache: a) páginas são normalmente copiadas na mesma ordem em que foram ejetadas da memória; e b) a interface realiza a busca em um outro canal após terminar com as páginas do canal atual. Quando um nó envia páginas consecutivas ao anel, estas duas características permitem que várias escritas sejam enviadas ao disco em uma única operação.

As páginas podem ser remapeadas na memória facilmente a partir do anel ótico. Durante uma falta de página, o nó verifica se o bit *Ring* para essa página está ligado. Se estiver desligado, a falta de página procede como descrito na seção anterior. Caso contrário, o nó usa a última tradução virtual-para-física dessa página para determinar qual foi o último nó que ejetou essa página. Assim o nó pode simplesmente procurar a página no correspondente *cache-channel*. Adicionalmente, o nó que sofreu a falta de página deve enviar uma mensagem, através do canal de pedidos, à interface OWCACHE do nó que possui o disco correspondente informando o número da página. Esta mensagem indica à interface OWCACHE que a página não precisa ser escrita no disco, pois existe novamente uma cópia dela na memória principal. Assim, a interface remota elimina o número da página da fila FIFO dos *cache-channels* e envia o ACK ao nó que originalmente ejetou a página da sua memória.

Note que OWCACHE não sofre de problemas de coerência, pois não é permitida mais de uma cópia da página fora dos limites do controlador de disco. A única cópia da página pode estar na memória ou no anel ótico.

Custo de Software. O custo de *software* de OWCACHE é ínfimo (supondo que o código do sistema operacional está disponível). As únicas alterações necessárias no código do sistema operacional são a inclusão dos bits *Ring* e do controle da interface OWCACHE.

Custo de Hardware. O custo do *hardware* eletrônico do anel ótico compreende a interface do barramento de memória, as filas FIFO, e as memórias temporárias e controladores que comunicam as partes eletrônicas e óticas da interface OWCACHE. Os requerimentos de *hardware* ótico do anel são também mínimos. A interface OWCACHE de cada nó pode ler de qualquer *cache-channel*, mas só pode escrever no *cache-channel* associado ao nó e, assim, não requer nenhum tipo de mecanismo de arbitragem. A interface OWCACHE rege e resincroniza este *cache-channel* com capacidade para escrita. Para realizar esta funcionalidade, a interface requer dois receptores sintonizáveis, um transmissor fixo e um receptor fixo, como mostrado na figura 6.2. Um dos receptores sintonizáveis é responsável por ler as páginas escritas no disco a partir do anel, enquanto que o outro receptor sintonizável é usado para procurar no anel uma página que produziu uma falta local. O transmissor fixo é usado para inserir novos dados no *cache-channel* com capacidade para escrita. Em conjunto com este transmissor, o receptor fixo é usado para recircular os dados no *cache-channel* com capacidade para escrita. Assim, o custo de *hardware* ótico para o anel é de só $4 \times p$ componentes óticos, onde p é o número de nós e *cache-channels* no multiprocessador. Assim, o custo total da interface OWCACHE (OPTNET + anel ótico) é de $11 \times p$ componentes óticos. Este custo é um pouco maior que OPTNET, mas é ainda aceitável mesmo com o custo atual da tecnologia ótica. A produção em grandes quantidades dos componentes óticos e os avanços da tecnologia ótica reduziriam estes custos ainda mais.

Note que, embora o anel ótico atue como uma cache para dados de disco, ele não garante não-volatilidade (como alguns – não todos – controladores de disco fazem), de forma similar ao uso da memória de nós livres para armazenar *swap-outs* (e.g., [59]). No entanto, este não é um problema sério, pois os dois esquemas otimizam a gerência de memória virtual para aplicações que não têm requerimentos de confiabilidade, como é o caso das aplicações científicas.

Note também que, embora este estudo tenha estendido a rede OPTNET com o anel ótico, a mesma idéia pode ser aplicada a qualquer rede ótica. OPTNET foi selecionada porque a combinação desta rede com o seu protocolo de coerência atinge a melhor relação custo/desempenho de todos os sistemas similares sob várias hipóteses arquiteturais e para um conjunto grande de aplicações [9], mas nenhuma característica do projeto depende da rede ótica base.

Parâmetro	Valor
Número de Nós	8
Número de Discos	4
Tamanho da Página	4 KBytes
Latência de uma falha na TLB	100 ciclos
Latência de um <i>TLB-Shutdown</i>	500 ciclos
Latência de uma Interrupção	400 ciclos
Tamanho do <i>Write-Buffer</i>	16 entradas
Tamanho da Cache Primária	4 KBytes
Bloco da Cache Primária	32 Bytes
Latência da Cache Primária	1 ciclo
Tamanho da Cache Secundária	16 KBytes
Bloco da Cache Secundária	64 Bytes
Latência da Cache Secundária	12 ciclos
Tamanho da Memória por Nó	256 KBytes
Latência do Barramento de Memória	12 ciclos
Taxa do Barramento de Memória	800 MBytes/s
Latência do Barramento de E/S	8 ciclos
Taxa do Barramento de E/S	300 MBytes/s
Taxa de Transmissão Ótica	10 Gbits/s
Tempo de Vôo	1 ciclo
Canais WDM no Anel Ótico	8
Latência Round-Trip do Anel	52 μ seg
Capacidade de Armazenamento do Anel	512 KBytes
Capacidade de Armazenamento por Canal	64 KBytes
Cache do Controlador de Disco	16 KBytes
Latência Mínima de Busca	2 mseg
Latência Máxima de Busca	22 mseg
Latência Rotacional	4 mseg
Taxa de Transferência do Disco	20 MBytes/s

Tabela 6.1: Parâmetros Base Principais e os seus Valores – 1 ciclo = 5 η seg

6.2 Metodologia

Como estamos interessados em avaliar o desempenho de um multiprocessador baseado em OWCACHE com coerência de caches e sob diversas suposições arquiteturais, neste estudo usamos simulações de aplicações paralelas reais.

6.2.1 Simulação

Foram utilizadas simulações detalhadas, dirigidas por eventos (baseadas no MINT [42]), de multiprocessadores com coerência de cache baseados em OPTNET e OWCACHE. A

contenção gerada pela memória, rede e entrada/saída é totalmente modelada. A simulação do sistema operacional do multiprocessador está limitada à parte que verdadeiramente nos interessa, a gerência de memória virtual.

Os parâmetros da simulação e os seus valores base são mostrados na tabela 6.1. Os tamanhos das caches e memória principal foram mantidos pequenos propositalmente, pois limitações nos tempos de simulação nos impedem de usar tamanhos de entrada reais. Na verdade, as capacidades das caches primárias e secundárias, o anel ótico, e a cache de disco foram reduzidas por um fator de 32, enquanto que a capacidade da memória principal foi reduzida por um fator de 256 com relação aos tamanhos de um sistema real. O objetivo destas reduções é produzir aproximadamente o mesmo tráfego de memória virtual que nos sistemas reais.

A seleção da quantidade de armazenamento ótico requer observações adicionais. O aumento do tamanho do armazenamento de OWCACHE pode ser realizado aumentando-se o comprimento do anel ótico (e aumentando-se assim a sua latência de *round-trip*), aumentando a taxa de transmissão e/ou usando mais *cache-channels*. De qualquer forma, aumentar a capacidade do anel ótico simulado por um fator de 32 pode não ser prático com a tecnologia ótica atual. No entanto, espera-se que em um futuro próximo este aumento de tamanho seja possível. Na verdade, as suposições de capacidade podem ser consideradas altamente conservadoras com relação ao potencial da ótica, especialmente se são consideradas técnicas de multiplexação tais como OTDM a qual suportaria até 5000 canais [21].

Os valores base da tabela 6.1 representam nossa percepção do que é “razoável” para os multiprocessadores atuais. O estudo do espaço de parâmetros mostrado na seção 6.3 permitirá investigar o impacto de outras importantes hipóteses arquiteturais.

6.2.2 Aplicações

Uma aplicação *out-of-core* é definida como sendo aquela que trabalha sobre um conjunto de dados extremamente grande que não cabe na memória principal. Assim, o nosso conjunto de aplicações consiste de 7 programas paralelos: Em3d, FFT, Gauss, LU, Mg, Radix e SOR, onde os seus parâmetros de entrada foram modificados para produzir um conjunto de dados (na verdade, um *working set* por processador) maior que a memória

Prog.	Descrição	Tamanho de Entrada	Total (MB)
Em3d	Propagação de ondas eletromagnéticas	32 K nós, 5% remotos, 10 iter.	2,5
FFT	Transformada Rápida de Fourier 1D	64 K pontos	3,1
Gauss	Eliminação de Gauss sem blocos	570 × 512 doubles	2,3
LU	Fatorização LU por blocos	576 × 576 doubles	2,7
Mg	Poisson 3D usando técnicas de multigrid	32 × 32 × 64 floats, 10 iter.	2,4
Radix	Ordenação de inteiros	320 K chaves, radix 1024	2,6
SOR	Relaxamento progressivo	640 × 512 floats, 10 iterações	2,6

Tabela 6.2: Descrição das Aplicações e os seus Principais Parâmetros de Entrada

principal de cada processador simulado. Todas essas aplicações já foram descritas nos capítulos anteriores. A tabela 6.2 mostra as aplicações, os seus parâmetros principais de entrada e o tamanho total dos seus dados (em MBytes). Todas as aplicações mapeiam os seus arquivos em memória, tanto para leitura como para escrita³, e os acessam através dos mecanismos de memória virtual padrão. Desta forma, os diferentes padrões de acesso à memória apresentados pelas aplicações consideradas permitem generalizar os nossos resultados para um amplo conjunto de padrões de *swap-out*.

6.3 Resultados Experimentais

Nesta seção será avaliado o desempenho de um multiprocessador baseado em OWCACHE, comparando-o com um multiprocessador baseado em OPTNET.

6.3.1 Benefícios de Desempenho

Primeiramente, é importante determinar qual é o melhor número mínimo de *frames* livres para cada combinação de multiprocessador com técnica de *prefetching*. Realizamos experimentos variando este número mínimo de *frames* para cada uma das aplicações estudadas. Na presença de OWCACHE, a maioria das aplicações atinge o seu melhor desempenho com um mínimo de só 2 *frames* livres, independente da estratégia de *prefetching* utilizada.

A melhor configuração para o multiprocessador baseado em OPTNET, entretanto, não é óbvia. Sob *prefetching* ótimo, 3 aplicações (Gauss, LU, e SOR) se beneficiam de gran-

³A chamada *mmap* do UNIX força o usuário a especificar um possível tamanho máximo do arquivo. Isto não foi um problema nos casos estudados, pois sempre foi possível determinar o tamanho exato de todos os arquivos usados pelas aplicações. No entanto, nosso sentimento é que a chamada *mmap* do UNIX é bastante restritiva para um estilo de programação baseado exclusivamente em memória virtual.

Aplicação	OPTNET	OWCache
em3d	49,1	1,8
fft	70,6	3,1
gauss	30,8	1,0
lu	40,2	1,9
mg	29,8	0,5
radix	47,1	2,4
sor	31,6	1,2

Tabela 6.3: Tempos Médios de *Swap-Out* (em Milhões de Ciclos de Processador) sob *Prefetching* Ótimo

de quantidade (≥ 16) de *frames* livres, enquanto duas delas (Em3d e FFT) atingem o seu melhor desempenho com apenas 2 ou 4 *frames* livres. As outras duas aplicações, Mg e Radix, requerem 8 e 12 *frames* livres, respectivamente, para obter o seu melhor desempenho. Por outro lado, sob *prefetching* básico, todas as aplicações, exceto SOR, se beneficiam de pequena quantidade (< 4) de *frames* livres. Assim, selecionamos 12 e 4 *frames* como o número mínimo de *frames* livres sob *prefetching* ótimo e básico, respectivamente. Todos os resultados apresentados a seguir, correspondem a estas configurações.

Um dos principais interesses deste estudo é determinar como os benefícios providos por OWCache se relacionam com os seus ganhos de desempenho. Como foi mencionado, OWCache possui os seguintes benefícios de desempenho: provê uma área temporal onde as páginas ejetadas da memória podem residir até que o disco esteja livre; aumenta a possibilidade de combinar várias escritas ao disco; e atua como uma *victim-cache* para as páginas que saem da memória e posteriormente são acessadas pelo mesmo ou por um processador diferente. A seguir revisamos algumas estatísticas relacionadas com cada um destes benefícios.

Área Temporal de Escrita. As tabelas 6.3 e 6.4 mostram os tempos médios (em ciclos de processador) para ejetar da memória uma página sob *prefetching* ótimo e básico respectivamente. As tabelas mostram que os tempos de *swap-out* são de 1 a 3 ordens de magnitude menores quando OWCache é utilizado. A razão principal deste resultado é que OWCache efetivamente aumenta a quantidade de cache de disco observada pela memória. Um *swap-out* é somente retardado, na presença de OWCache, quando o *cache-channel* correspondente a esse nó está cheio. Por outro lado, quando OWCache não é assumido, os

Aplicação	OPTNET	OWCache
em3d	192,7	2,1
fft	382,1	43,6
gauss	762,3	78,0
lu	393,3	41,2
mg	89,4	6,1
radix	1223,1	2,1
sor	661,3	2,1

Tabela 6.4: Tempos Médios de *Swap-Out* (em Milhares de Ciclos de Processador) sob *Prefetching* Básico

Aplicação	OPTNET	OWCache	Aumento
em3d	1,21	1,24	2%
fft	1,50	2,06	37%
gauss	1,06	1,07	1%
lu	1,15	1,25	9%
mg	1,20	1,27	6%
radix	1,17	1,37	17%
sor	1,64	2,90	77%

Tabela 6.5: Número Médio de Páginas Escritas sob *Prefetching* Ótimo

swap-outs são muito mais frequentemente retardados devido à falta de espaço na cache de disco. Como esperado, as tabelas também mostram que os tempos de *swap-out* são muito maiores sob a técnica de *prefetching* ótimo do que sob *prefetching* básico. Este resultado se explica pelo fato de que sob *prefetching* ótimo os reduzidos tempos de leitura de uma página agrupam os *swap-outs* no tempo, aumentando a contenção no disco.

Combinação de Escritas. Devido à forma em que as páginas são copiadas a partir do anel óptico para a cache de disco, a localidade das escritas na cache de disco é normalmente aumentada. Quando páginas consecutivas podem ser encontradas sequencialmente na cache do controlador do disco, as escritas destas páginas podem ser combinadas em um único acesso de escrita ao disco. Os dados nas tabelas 6.5 e 6.6 confirmam esta afirmação. As tabelas mostram o número médio de páginas que são combinadas em cada operação de escrita ao disco; o máximo valor possível de combinação é 4, que é o número máximo de páginas que podem caber na cache do controlador de disco. Os resultados mostram que os ganhos na combinação de escritas são moderados sob a estratégia de *prefetching*

Aplicação	OPTNET	OWCache	Aumento
em3d	1,16	1,17	1%
fft	1,28	1,45	13%
gauss	1,03	1,04	1%
lu	1,04	1,05	1%
mg	1,04	1,19	14%
radix	1,08	1,12	4%
sor	1,17	1,50	28%

Tabela 6.6: Número Médio de Páginas Escritas sob *Prefetching* Básico

Aplicação	Ótimo	Básico
em3d	9,2	7,1
fft	12,8	8,4
gauss	57,6	60,9
lu	18,9	14,6
mg	55,8	46,2
radix	20,6	18,0
sor	18,6	30,7

Tabela 6.7: Taxas de Acerto (em Porcentagem) para OWCache sob Diferentes Técnicas de *Prefetching*

básico ($\leq 28\%$), mas podem ser significativos sob a estratégia de *prefetching* ótimo ($\leq 77\%$). Novamente, o agrupamento temporal dos *swap-outs* sob *prefetching* ótimo é o responsável por este resultado, pois é mais comum para o controlador de disco encontrar escritas consecutivas ao mesmo tempo na sua cache.

Victim-Cache. A tabela 6.7 apresenta as taxas de acerto em uma leitura de página em OWCache sob as técnicas de *prefetching* ótimo e básico. A tabela mostra que as taxas de acerto são ligeiramente maiores sob *prefetching* ótimo que sob *prefetching* básico, exceto para Gauss e SOR, novamente devido às características temporais dos *swap-outs* sob estas duas técnicas. Adicionalmente, estes resultados mostram que as taxas de acerto podem ser tão altas quanto 61% (Gauss) ou tão baixas quanto 7% (Em3d). Estes resultados se devem à combinação de dois fatores: o tamanho do conjunto de trabalho da memória e o grau de compartilhamento de dados das aplicações. Gauss, MG, e Em3d exibem uma significativa quantidade de compartilhamento, mas só Gauss e MG possuem conjuntos de trabalho que (quase) podem caber no tamanho combinado de memória e OWCache. As

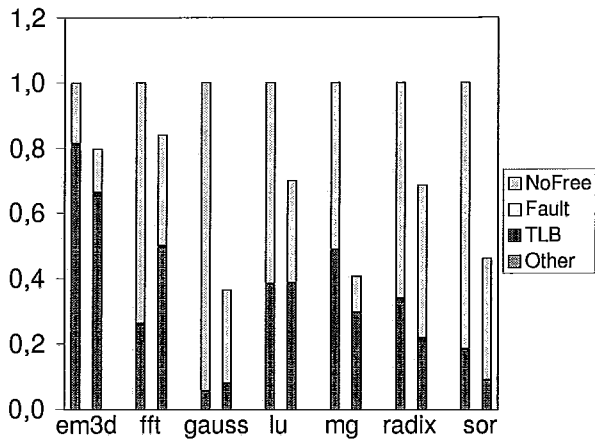


Figura 6.3: Tempo de Execução (com Relação ao MP OPTNET) de OPTNET e OWCACHE sob *Prefetching* Ótimo

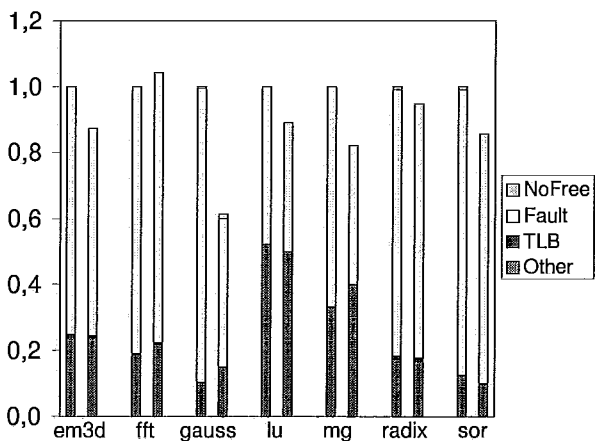


Figura 6.4: Tempo de Execução (com Relação ao MP OPTNET) de OPTNET e OWCACHE sob *Prefetching* Básico

outras aplicações atingem taxas de acerto no intervalo de 8 a 31%. Os efeitos benéficos da *victim-cache* são ainda mais pronunciados sob *prefetching* básico, pois os custos de uma falta de leitura representam uma fração significativa dos tempos de execução da aplicação. Além disso, as páginas podem ser lidas somente um pouco mais rápido do anel óptico do que da cache do controlador de disco, reduzindo os ganhos potenciais da *victim-cache* sob *prefetching* ótimo.

Os resultados apresentados aqui confirmam que OWCACHE produz benefícios de desempenho significativos. Os maiores ganhos são produzidos pelos rápidos *swap-outs* e pela capacidade de atuar como uma *victim-cache*.

Desempenho Geral. As figuras 6.3 e 6.4 mostram os tempos de execução normalizados de cada uma das aplicações sob *prefetching* ótimo e básico, respectivamente. De cima

para abaixo, cada barra nos gráficos está dividida em: o tempo parado como resultado da falta de *frames* livres (“NoFree”); o custo das faltas de página (“Fault”); o custo das falhas na TLB e *TLB-shutdowns* (“TLB”); e os componentes do tempo de execução que não são relacionados com a gerência da memória virtual (“Other”), incluindo tempo de processador, falhas nas caches e tempos de sincronização.

A figura 6.3 mostra que sob a estratégia de *prefetching* ótimo, os tempos “NoFree” são sempre muito significativos para o multiprocessador baseado em OPTNET, especialmente para Gauss e SOR. A frequência com que o sistema operacional sofre de falta de *frames* livres no multiprocessador baseado em OPTNET se deve ao fato de que as leituras de páginas terminam rapidamente, enquanto que os *swap-outs* consomem muito tempo. Quando o multiprocessador inclui OWCACHE, os tempos “NoFree” são reduzidos bastante como resultado dos *swap-outs* muito mais rápidos.

A figura também demonstra que, para algumas aplicações, o tempo que levam as operações não relacionadas com memória virtual é significativamente reduzido na presença de OWCACHE. Estas reduções são resultado, principalmente, da melhor sincronização produzida pela grande redução no desbalanceamento de carga. Desta forma, pode-se observar que OWCACHE produz ganhos de desempenho que variam de 16% (FFT) até 60 e 64% (MG e Gauss), 39% em média, quando é assumido um *prefetching* ótimo. Na verdade, as melhoras de desempenho são maiores que 30% em todos os casos, exceto para Em3d e FFT.

Os resultados de desempenho, quando o *prefetching* básico é assumido, são totalmente diferentes. Sob esta técnica, os tempos de execução são dominados pelas latências de falta de página, pois as taxas de acerto na cache de disco nunca são maiores que 15%. Assim, as latências de falta de página produzem o tempo necessário para que os *swap-outs* terminem. Como resultado, os tempos “NoFree” quase desaparecem, diminuindo a importância de *swap-outs* rápidos na arquitetura OWCACHE.

Sob *prefetching* básico, a adição de OWCACHE ao multiprocessador melhora o seu desempenho de 5% (Radix) a 39% (Gauss) para todas as aplicações, exceto FFT, a qual degrada o seu desempenho por 4%. Os ganhos relacionados com OWCACHE são produzidos pelas reduções razoáveis das latências de falta de página, as mesmas que são resultado da leitura de páginas a partir da cache ótica e da diminuição da contenção no disco.

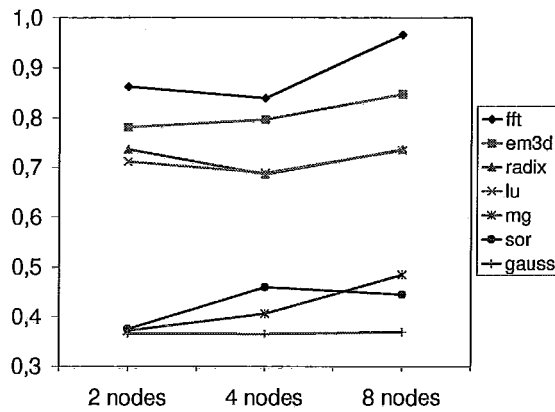


Figura 6.5: Tempo de Execução do MP OWCACHE (com Relação ao MP OPTNET) para 2, 4, e 8 Nós de Entrada/Saída sob *Prefetching* Ótimo

Discussão. Em resumo, mostramos que OWCACHE é extremamente útil quando o *prefetching* é eficiente, principalmente como resultado dos *swap-outs* rápidos. OWCACHE não é tão eficiente quando o *prefetching* é ineficiente ou inexistente, mesmo quando a sua característica de *victim-cache* melhora o desempenho de muitas aplicações significativamente. Espera-se que os resultados para técnicas de *prefetching* reais e sofisticadas [52, 56] estejam entre esses dois extremos. Além disso, quando as técnicas de *prefetching* melhorarem e a tecnologia ótica se desenvolver, ganhos maiores virão da arquitetura OWCACHE.

6.3.2 Impacto dos Parâmetros Arquiteturais

Nesta subseção será estudado o efeito dos parâmetros mais importantes usados nas simulações: o número de nós com capacidade de entrada/saída, o tamanho de OWCACHE, as caches do controlador de disco e a memória principal, e a largura de faixa usada nos diferentes multiprocessadores.

Número de Nós com Entrada/Saída Habilitada. Variamos o número de nós com entrada/saída habilitada de 2 a 8 em um sistema com 8 nós. As figuras 6.5 and 6.6 apresentam os resultados destes experimentos para as 7 aplicações sob *prefetching* ótimo e básico, respectivamente. Cada ponto das figuras representa o tempo de execução do multiprocessador baseado em OWCACHE, normalizado com o tempo de execução correspondente do multiprocessador baseado exclusivamente em OPTNET.

Os resultados mostram que os ganhos de desempenho produzidos por OWCACHE sob *prefetching* ótimo decrescem quando o número de nós com entrada/saída habilitada (i.e.,

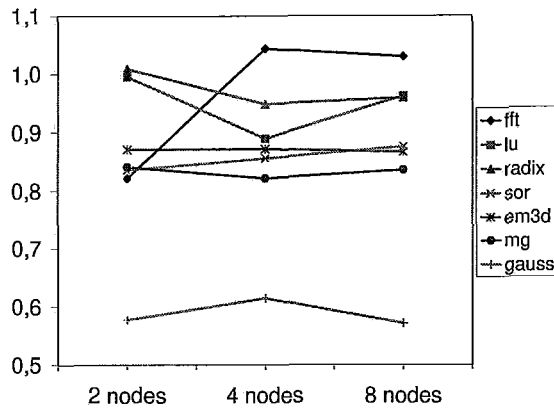


Figura 6.6: Tempo de Execução do MP OWCACHE (com Relação ao MP OPTNET) para 2, 4, e 8 Nós de Entrada/Saída sob *Prefetching* Básico

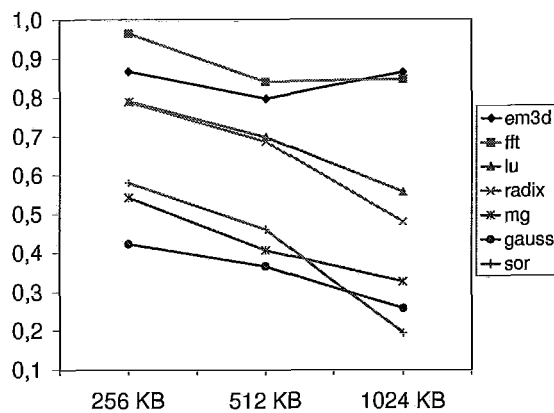


Figura 6.7: Tempo de Execução do MP OWCACHE (com Relação ao MP OPTNET) com 256, 512, e 1024-KB sob *Prefetching* Ótimo

o *throughput* de entrada/saída do sistema) é aumentado, pois os tempos “NoFree” decrescem em porcentagem com relação ao tempo total de execução do multiprocessador baseado em OPTNET. De qualquer forma, os ganhos de desempenho de OWCACHE permanecem significativos, com todas as aplicações beneficiando-se em média 35% quando são utilizados 8 nós de entrada/saída.

Por outro lado, o aumento do número de nós com capacidade de entrada/saída normalmente aumenta o desempenho atingível por OWCACHE sob *prefetching* básico, pois a contenção no disco deixa de ser um problema para o multiprocessador baseado em OWCACHE.

Tamanho de OWCACHE. Ajustando o número de nós com capacidade de entrada/saída para 4, variamos a capacidade de armazenamento de OWCACHE de 256 KBytes a 1 MByte de dados através do ajuste do comprimento do anel ótico. As figuras 6.7 e 6.8 mostram

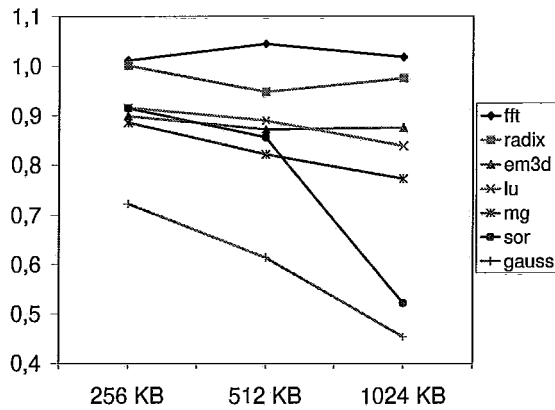


Figura 6.8: Tempo de Execução do MP OWCACHE (com Relação ao MP OPTNET) com 256, 512, e 1024-KB sob *Prefetching* Básico

estes resultados sob *prefetching* ótimo e básico. Novamente, cada ponto das figuras representa o desempenho de OWCACHE normalizado com relação ao desempenho de OPTNET. Sob *prefetching* ótimo, observamos que aumentando a capacidade do anel, o desempenho de quase todas as aplicações melhora, pois os *swap-outs* terminam mais rápido em média. Na verdade, *swap-outs* mais rápidos apresentam um impacto substancial no desempenho sob *prefetching* ótimo, já que os tempos “NoFree” representam invariavelmente uma grande fração do tempo total de execução.

De forma similar, aumentos na capacidade de armazenamento melhoram o desempenho de várias aplicações sob *prefetching* básico. Neste caso, o desempenho melhora nas aplicações que podem se beneficiar substancialmente da *victim-cache* (i.e., Gauss, LU, MG, e SOR). Gauss e SOR são as aplicações que se beneficiam mais destes aumentos de capacidade. A razão para esse fato é que o espaço adicional e a temporização dos *swap-outs* permite um aumento significativo no número de faltas de página que podem ser satisfeitas por OWCACHE. Mais especificamente, as taxas de acerto em OWCACHE para Gauss variam de 45 a 61 e a 79% quando a capacidade de OWCACHE aumenta de 256 KBytes a 512 KBytes e a 1 MByte, enquanto que as taxas de acerto de SOR variam de 12 a 31 e a 72% para os mesmos aumentos de capacidade.

Note que em OWCACHE, a latência média de acesso aos dados deixa de ser crítica, o que permite diminuir custos sem degradar os ganhos de desempenho obtidos pela memória de rede. Isso se deve a que as latências de acesso aos discos são extremamente grandes, em média, 6 ordens de magnitude maiores que as latências de acesso à memória.

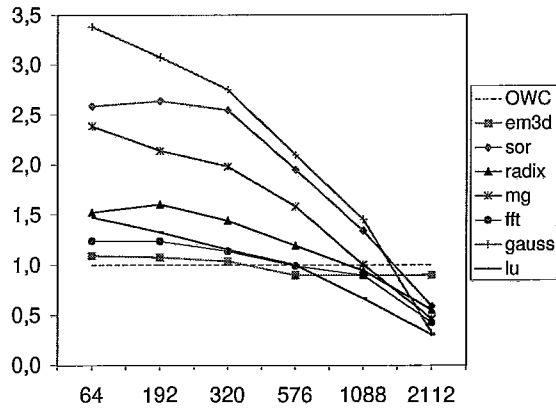


Figura 6.9: Tempo de Execução do MP OPTNET (com Relação ao MP OWCACHE) para Vários Tamanhos de Cache de Disco Combinada (em KBytes) sob *Prefetching* Ótimo

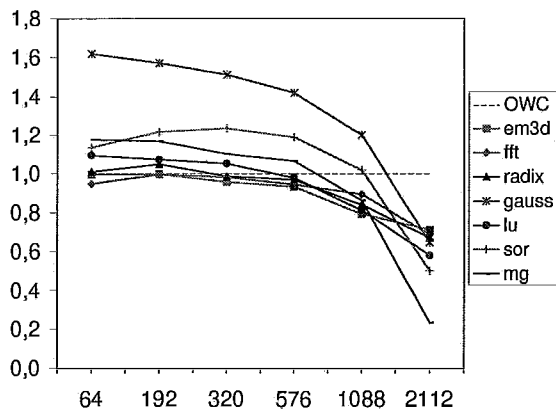


Figura 6.10: Tempo de Execução do MP OPTNET (com Relação ao MP OWCACHE) para Vários Tamanhos de Cache de Disco Combinada (em KBytes) sob *Prefetching* Básico

Este fato permite armazenar uma quantidade maior de dados em um número menor de canais através do simples aumento do comprimento da fibra.

Tamanho das Caches dos Controladores de Disco. Também executamos experimentos onde os tamanhos das caches do controlador de disco na arquitetura baseada em OPTNET foram variados. As figuras 6.9 e 6.10 mostram estes resultados. Cada ponto nas figuras representa o tempo de execução de OPTNET normalizado com relação ao tempo de execução de OWCACHE.

Estas figuras nos permitem responder duas perguntas importantes: (1) Um multiprocessador baseado em OWCACHE permitiria obter um melhor desempenho que um sistema baseado em OPTNET com a mesma capacidade total de armazenamento em caches? e (2) Qual capacidade de armazenamento em caches seria necessária para que o multiprocessador baseado em OPTNET ultrapasse o sistema baseado em OWCACHE?

Para responder à primeira pergunta comparamos o desempenho de dois multiprocessadores com 4 nós de entrada/saída e um espaço de cache total de 576 KBytes. Um dos multiprocessadores está equipado com um anel ótico de 512 KBytes e caches de 16 KBytes nos controladores de disco, enquanto que o outro tem somente caches de disco de 144 KBytes. Os resultados destes experimentos mostram que o multiprocessador baseado em OWCACHE apresenta um melhor desempenho para a maioria de aplicações, mesmo sob estas condições.

Sob *prefetching* ótimo, o sistema proposto é minimamente ultrapassado nas aplicações Em3d (10%), FFT (1%) e LU (1%), somente. Os ganhos de desempenho resultam das significativas reduções nos tempos “NoFree”, geradas pelos *swap-outs* mais rápidos. Os *swap-outs* terminam mais rápido com OWCACHE como uma consequência da maior flexibilidade do armazenamento ótico. Mais especificamente, OWCACHE pode tratar com distribuições especiais de *swap-outs* ao longo dos nós de entrada/saída mediante o cacheamento de mais dados dirigidos a certos nós que a outros em cada instante de tempo. Na verdade, OWCACHE pode inclusive, se necessário, estar temporariamente preenchido com dados dirigidos a um único nó. Grandes caches de disco, entretanto, não são tão flexíveis, pois a quantidade de dados que pode ser dirigida a um nó, sem causar retardos, é equivalente ao tamanho de uma única cache de disco. Este fato ilustra a principal vantagem da cache compartilhada (OWCACHE) em comparação a múltiplas caches locais (caches dos controladores de disco) com os mesmos tamanhos combinados.

Sob *prefetching* básico, a vantagem de desempenho de OWCACHE não é clara, pois os *swap-outs* mais rápidos são irrelevantes em termos do desempenho geral. O multiprocessador baseado em OWCACHE com 576 KBytes de armazenamento total só apresenta um melhor desempenho para 3 aplicações (Gauss, MG, e SOR). No entanto, OWCACHE é ultrapassado por não mais que 8%. Para Gauss, MG, e SOR, OWCACHE obtém vantagem da sua capacidade de atuar como uma *victim-cache*. A política da *victim-cache* efetivamente produz taxas de acerto maiores que as taxas combinadas das caches dos controladores de disco do multiprocessador baseado em OPTNET para estas três aplicações.

Assim, estes resultados demonstram que OWCACHE é claramente mais eficiente que o OPTNET sob *prefetching* ótimo quando ambos os sistemas envolvem a mesma quantidade de armazenamento. Por outro lado, o *prefetching* básico leva a uma mistura de resultados.

Para responder à segunda pergunta comparamos o desempenho de um multiprocessador baseado em OWCACHE e com a mesma configuração dos experimentos anteriores, com um multiprocessador baseado em OPTNET com 64, 192, 320, 576, 1088 e 2112 KBytes de cache de disco combinada (Figuras 6.9 e 6.10). Novamente, cada ponto nas figuras representa o desempenho de OPTNET normalizado com relação ao desempenho de OWCACHE.

Sob *prefetching* ótimo observamos que o multiprocessador baseado em OPTNET requer 2112 KBytes para ultrapassar OWCACHE em Gauss e SOR, enquanto que para MG e Radix requer 1088 KBytes. As outras 3 aplicações requerem somente 576 KBytes. Sob *prefetching* básico, por outro lado, o multiprocessador baseado em OPTNET requer 2112 KBytes para ultrapassar OWCACHE em Gauss e SOR, 1088 KBytes para MG e 576 KBytes para LU. As outras 3 aplicações requerem 320 KBytes ou menos para ultrapassar OWCACHE. Embora estas caches de disco não sejam extremamente grandes para sistemas reais, note que as simulações assumem uma memória principal de somente 2048 KBytes para todo o multiprocessador.

Estes últimos resultados mostram que o multiprocessador baseado em OPTNET normalmente requer uma grande capacidade de caches de disco para atingir o mesmo desempenho do sistema OWCACHE. Devido à simplicidade do *hardware* de OWCACHE, ao fato de que ela é somente uma extensão à uma rede existente, ao rápido decréscimo do custo dos componentes óticos, e ao fato de que aumentar significativamente a quantidade de memória nos controladores de disco para simplesmente satisfazer às aplicações *out-of-core* é claramente exagerado, a solução ótica a este problema parece ser melhor que a posição eletrônica contrária.

Tamanho da Memória Principal. Também variamos o tamanho da memória principal de cada nó. Com a finalidade de manter as características *out-of-core* das aplicações, variamos também os seus tamanhos de entrada na mesma proporção. Mais especificamente, a memória principal de cada nó foi aumentada de 256 a 320 KBytes. Desta forma, observamos que o desempenho do multiprocessador baseado em OWCACHE com relação ao baseado em OPTNET é o mesmo em ambos os casos sob *prefetching* básico. Por outro lado, sob *prefetching* ótimo, as variações de desempenho dependem da aplicação.

Por exemplo, os ganhos do sistema baseado em OWCACHE com relação ao baseado em OPTNET crescem de 64 a 68% em Gauss, enquanto que em SOR, os ganhos caem de 53 a 35%. A redução nos ganhos obtidos por SOR é devida ao decremento na taxa de *swap-outs*. Esta taxa cai de 5,5 a 3,8 *swap-outs* a cada milhão de ciclos.

Assim, pode-se concluir que os ganhos obtidos pelo uso de OWCACHE são significativos para a maioria de aplicações *out-of-core*, mesmo que o tamanho da memória principal em cada nó seja maior, pois esses ganhos dependem basicamente da taxa de *swap-outs*. Em nossos experimentos observamos ganhos significativos quando a taxa de *swap-outs* é maior que 2 *swap-outs* a cada milhão de ciclos.

Largura de Faixa para Swap-Outs. A diferença significativa da largura de faixa para *swap-outs* entre os dois multiprocessadores considerados pode influir consideravelmente nas comparações. Para mostrar que este não é o caso, coletamos informação sobre a quantidade de contenção existente no *swap-channel* da arquitetura baseada em OPTNET. Observamos que a contenção no *swap-channel* praticamente é inexistente. Além disso, quando se configura o multiprocessador baseado em OPTNET para usar um *swap-channel* por nó (e assim evitar inclusive a arbitragem dos canais), as únicas aplicações que melhoram o seu desempenho são MG (4%) e Radix (9%) sob *prefetching* ótimo. As outras 5 aplicações, sob *prefetching* ótimo, e todas as aplicações, sob *prefetching* básico, apresentam os mesmos resultados de desempenho de um único *swap-channel*.

Pode-se concluir então que a grande largura de faixa usada pelos *cache-channels* não têm influência nenhuma sobre os ganhos de desempenho obtidos por OWCACHE. O verdadeiro motivo para estes ganhos está na habilidade dos *cache-channels* de atuar como uma área extra para escritas e como uma *victim-cache*.

Resumo. Estes resultados mostram que OWCACHE é extremamente eficiente sob ambos os tipos de *prefetching*, mesmo quando comparado com grandes caches de disco. Além disso, os resultados mostram que variações no tamanho de OWCACHE têm efeitos benéficos sob um *prefetching* ótimo, mas não necessariamente sob um *prefetching* básico.

6.4 Estendendo um Multiprocessador Tradicional

O estudo anterior mostra que um anel ótico pode ser benéfico para um multiprocessador óticamente interconectado. No entanto, estes multiprocessadores não são ainda amplamente utilizáveis. Com o propósito de mostrar que o anel ótico pode ser aplicado com sucesso em arquiteturas de multiprocessadores mais tradicionais e amplamente utilizáveis, nesta seção apresentamos a extensão de um multiprocessador conectado por uma grade eletrônica com uma cache ótica para escritas ao disco. Esta extensão, chamada de NWCACHE, é mais modular e flexível que OWCACHE, pois a interface NWCACHE pode ser inserida nos barramentos de entrada/saída de qualquer multiprocessador. A arquitetura de NWCACHE e os seus mais importantes resultados de desempenho serão discutidos abaixo.

6.4.1 Arquitetura Básica

Como a base para a implementação de NWCACHE, utilizamos a arquitetura de um multiprocessador tradicional com coerência de caches, onde os processadores são interconectados através de uma rede tradicional em grade e com roteamento *wormhole*. A estrutura de cada nó do sistema é a mesma da figura 6.1. Este multiprocessador foi estendido com o anel ótico simplesmente inserindo a interface NWCACHE no barramento de entrada/saída de cada nó. O controlador de disco dos nós com capacidade de entrada/saída pode ser conectado à interface NWCACHE. Assim, a interface NWCACHE une o nó ao anel ótico e filtra alguns dos acessos ao disco de forma similar à interface OWCACHE. Alguns destes acessos podem ser satisfeitos pela mesma interface NWCACHE.

Com a finalidade de implementar NWCACHE, o *hardware* padrão do multiprocessador não requer nenhuma modificação. O código para a gerência da memória virtual é bastante similar ao descrito anteriormente na seção 6.1.1 para OWCACHE.

A interface NWCACHE é similar à interface OWCACHE apresentada na figura 6.2. As únicas diferenças entre estas interfaces são: a) a interface do barramento de memória no projeto OWCACHE se transforma em uma interface para o barramento de entrada/saída; e b) a lógica para a interface de OPTNET no projeto OWCACHE é substituída por uma interface de disco. Como resultado, o custo do *hardware* eletrônico de NWCACHE está restrito às interfaces do barramento de entrada/saída e de disco, às filas FIFO, e às memórias e

aos controladores que ligam as partes eletrônica e ótica da interface NWCACHE. O custo do *hardware* ótico de NWCACHE é também mínimo, somente $4 \times p$ componentes óticos, onde p é o número de nós e *cache-channels* do multiprocessador.

Novamente, o código de gerência do anel é quase o mesmo discutido na seção 6.1.2. As mensagens que não estão relacionadas com a gerência de memória virtual ainda fluem através da rede eletrônica. No entanto, diferente de OWCACHE, as mensagens enviadas aos nós com capacidade de entrada/saída após o *swap-out* de uma página devem também incluir o número do nó que realiza a operação, pois a interface NWCACHE não tem meios para determinar a fonte dos *swap-outs*.

6.4.2 Resultados Experimentais

Para avaliar o desempenho do multiprocessador baseado em NWCACHE, utilizamos simulações dirigidas por eventos de um multiprocessador com coerência de caches DASH-like [60] com e sem NWCACHE. Os parâmetros das simulações são os mesmos listados na tabela 6.1. A rede eletrônica assume uma latência de roteamento de 4 ciclos do processador e uma taxa de transferência de 200 MBytes/s. Além disso, utilizamos o mesmo conjunto de aplicações e parâmetros de entrada mostrados na tabela 6.2.

Benefícios de Desempenho. Os experimentos realizados para determinar o número mínimo de *frames* livres mostra que, na presença de NWCACHE, a maioria das aplicações atinge o seu melhor desempenho com um mínimo de somente 2 *frames* livres, independente da estratégia de *prefetching*. Por outro lado, a melhor configuração para o multiprocessador tradicional não é óbvia. Sob *prefetching* ótimo, 4 aplicações (Gauss, LU, Radix, e SOR) favorecem números grandes (> 12) de *frames* livres, enquanto duas delas (Em3d e MG) atingem o seu melhor desempenho com somente 2 *frames* livres. A outra aplicação, Radix, requer 8 *frames* livres para obter o seu melhor desempenho. Sob *prefetching* básico, por outro lado, todas as aplicações exceto SOR favorecem números pequenos (2 ou 4) de *frames* livres. Assim, foram selecionados 12 e 4 *frames* como os números mínimos de *frames* livres sob *prefetching* ótimo e básico, respectivamente. Todos os resultados a seguir correspondem a estas configurações.

Da mesma forma que OWCACHE, NWCACHE melhora o desempenho devido ao fato de prover uma área extra onde as páginas podem residir até o disco estar livre; aumentando a

probabilidade de combinar várias escritas ao disco; e atuando como uma *victim-cache* para as páginas que foram ejetadas da memória e posteriormente acessadas pelo mesmo ou por um processador diferente. Estas características de NWCACHE melhoram o desempenho de forma similar a OWCACHE. Em resumo, os tempos de *swap-out* são de 1 a 3 ordens de magnitude menores quando a extensão NWCACHE é usada; os ganhos na combinação de escritas são moderados ($\leq 16\%$) sob *prefetching* básico, mas podem ser significativos ($\leq 58\%$) sob *prefetching* ótimo; e as taxas de acerto no anel ótico são ligeiramente maiores sob *prefetching* ótimo que sob *prefetching* básico, variando de 9 (Em3d) a 60% (Gauss e MG).

Além desses três benefícios de desempenho, que são compartilhados com OWCACHE, NWCACHE também reduz o tráfego de dados através da rede de interconexão do multiprocessador e dos barramentos de memória, pois: a) as páginas ejetadas da memória não são transferidas pela rede de interconexão; e b) as leituras de página que acertam em NWCACHE não são transferidas pela rede, nem pelos barramentos de memória dos nós de entrada/saída (quando o pedido para o nó de entrada/saída correspondente pode ser abortado a tempo). Esta redução no tráfego de dados produz uma redução na contenção observada pela rede.

Para avaliar os benefícios de NWCACHE em termos da redução de contenção, coletamos estatísticas da latência média de uma leitura de página a partir da cache do controlador de disco. A comparação destas estatísticas entre o multiprocessador tradicional e o multiprocessador baseado em NWCACHE provê uma estimativa aproximada da quantidade de contenção que é eliminada. Sob *prefetching* básico, estes resultados mostram que NWCACHE reduz as latências de um acerto na cache de disco em até 63%. Para a maioria de aplicações, as reduções variam de 24 a 38%. Levando em consideração que a leitura de uma página a partir da cache de disco demora 6K ciclos aproximadamente na ausência de contenção, pode-se estabelecer que as reduções de contenção geradas por NWCACHE são sempre significativas. Por exemplo, um acerto na cache de disco demora 21K ciclos em média para LU executando no multiprocessador tradicional. Isto significa que quase 15K ciclos são devidos à contenção de várias formas. Na presença de NWCACHE, o número de ciclos devido à contenção em LU é reduzido a 14K ciclos, indicando uma redução de 7%. No outro extremo, considere a redução da latência de acerto na cache de disco atingida

por MG, 63%. Dos 19K ciclos que MG leva para ler uma página das caches de disco no multiprocessador tradicional, aproximadamente 13K ciclos são devidos à contenção. Na presença de NWCACHE, o número de ciclos devidos à contenção em MG é reduzido a quase 700, ou seja 95% de redução na contenção. Por outro lado, sob *prefetching* ótimo, NWCACHE não é bem sucedido em atenuar a contenção, pois não existe tempo suficiente para prevenir a transferência de uma página através da rede e do barramento de entrada/saída quando o pedido de uma página produz um acerto no anel ótico.

Desempenho Geral. Sob *prefetching* ótimo, os tempos “NoFree” são reduzidos significativamente como resultado dos *swap-outs* muito mais rápidos permitidos por NWCACHE. Além disso, o tempo gasto pelas operações não relacionadas com memória virtual é significativamente reduzido na presença de NWCACHE. Estas reduções são resultado dos ganhos nos custos de acesso aos dados remotos e do melhor comportamento das sincronizações, que é, por sua vez, um resultado da redução significativa do tráfego através do sistema de memória (rede e memórias). Assim, observa-se que NWCACHE produz ganhos de desempenho de 41% em média, variando de 23% (Em3d) a 60 e 64% (MG e Gauss) quando um *prefetching* ótimo é assumido. Na verdade, os ganhos são maiores que 28% em todos os casos, exceto Em3d.

Sob *prefetching* básico, por outro lado, a adição de NWCACHE ao multiprocessador melhora o seu desempenho de 3 (Radix) a 42% (Gauss) para todas as aplicações exceto FFT, a qual degrada o seu desempenho em 3%. Os ganhos relacionados com NWCACHE são produzidos pela redução razoável das latências de faltas de página, as mesmas que são resultado das leituras que acertam na cache ótica e da diminuição de contenção.

Estes resultados confirmam que NWCACHE produz benefícios de desempenho significativos em várias formas. Os maiores ganhos de desempenho vêm dos rápidos *swap-outs*, da *victim-cache* e da redução de contenção. Lembre que OWCACHE produz ganhos de desempenho que são resultado dos rápidos *swap-outs* e da *victim-cache* somente; a contenção na rede ótica de OWCACHE é praticamente nula. O tempo de execução de cada aplicação mostra que as duas implementações da cache de escritas, NWCACHE e OWCACHE, produzem ganhos de desempenho similares. O impacto das variações arquiteturais é também similar em ambos os sistemas. Estes resultados não são uma coincidência, ob-

viamente. Para várias aplicações *out-of-core*, a maioria do tempo de execução é gasto em operações de memória virtual, as mesmas que se beneficiam similarmente das duas implementações. Para outras aplicações, NWCACHE é capaz de reduzir a contenção na rede eletrônica o suficiente para fazê-la ter um comportamento tão bom como o de uma rede ótica.

6.5 Trabalhos Relacionados

Um pouco poucas áreas são relacionadas com esta proposta, por exemplo, o uso de redes WDM em computadores, o uso da ótica na implementação de memórias de linha de retardo e as otimizações de operações de escrita ao disco.

As memórias de linha de retardo foram implementadas em sistemas de comunicação ótica [34] e em computadores totalmente óticos [36]. Do conhecido até agora, o único sistema que explora o potencial da ótica para armazenar dados no projeto de multiprocessadores é NetCache. No sistema NetCache um anel ótico é usado para armazenar os blocos de memória como se fosse uma cache de terceiro nível compartilhada por todos os processadores. Tanto NetCache como OWCACHE são extensões simples à rede OPTNET, mas NWCACHE é uma cache de escritas ao disco ligada aos barramentos de entrada/saída de um multiprocessador tradicional. Uma vantagem arquitetural dos sistemas OWCACHE e NWCACHE, sobre NetCache, é o seu reduzido custo de *hardware* ótico; o número de componentes óticos do sistema NetCache é $25 \times p$, onde p é o número de processadores.

Alguns pesquisadores também têm se preocupado em melhorar o desempenho das operações de escrita em vários tipos de subsistemas de discos. Estes esforços incluem trabalhos para melhorar o desempenho de escritas pequenas em RAIDs (e.g., [61]), usar RAM não volátil como cache de escritas (e.g., [62]), fazer *logs* das escritas e posteriormente escrevê-las no disco seqüencialmente (e.g., [63]), usar a memória dos nós desocupados ou com pouca carga para armazenar as páginas ejetadas das outras memórias [59], e usar um disco de *logs* para cachear as escritas dirigidas ao disco principal [64]. Os dois últimos tipos de trabalhos são os mais parecidos com o nosso.

O armazenamento dos *swap-outs* na memória de outro nó é somente apropriado para redes de estações de trabalho onde um ou mais nós podem estar desocupados ou com

carga baixa em qualquer instante. Esta mesma técnica não pode ser aplicada ao ambiente computacional considerado pela nossa proposta, pois todos os processadores são sempre parte da computação e não possuem memória livre para ajudar-se entre si.

A arquitetura de armazenamento proposta em [64], chamada de DCD (Disk Caching Disk), localiza um disco de *logs* entre a cache de disco baseada em RAM e o disco de dados verdadeiro, criando um nível extra de cache para escritas. Novos dados a serem escritos ao disco são armazenados na cache de RAM e posteriormente escritos seqüencialmente no disco de *logs*. Sobre escrever ou ler um bloco requer uma busca no disco de *logs* para encontrar o bloco correspondente. Quando o disco de dados está livre, os dados são copiados do disco de *logs* ao disco de dados. Este esquema melhora o desempenho devido a reduzir significativamente as latências de busca e rotacional quando são escritos dados novos ao disco de *logs*, resultando em uma liberação de espaço mais rápida na cache de RAM. Sobre escrever ou ler um bloco envolve latências de busca e rotacional comparáveis aos acessos ao disco de dados principal.

Da mesma forma que DCD, o anel ótico também tenta melhorar o desempenho das escritas criando um nível extra de cache. No entanto, o anel ótico localiza esta cache entre a memória principal e as caches de disco, não requerendo nenhuma modificação nos controladores de disco padrão. Além disso, sobre escrever ou ler dados da cache ótica é tão eficiente como escrever dados novos no anel ótico. Outra vantagem da proposta da memória de rede é que ela cria um caminho exclusivo para que as escritas cheguem aos controladores de disco. No entanto, a tecnologia usada para implementar a cache adicional em DCD permite maior espaço de armazenamento que a cache ótica.

6.6 Conclusões

Neste capítulo foi proposto OWCACHE: uma extensão simples à rede ótica de um multiprocessador com coerência de cache que melhora significativamente o desempenho das aplicações *out-of-core* através da otimização dos *swap-outs* de páginas. As mais importantes vantagens de OWCACHE são os seus *swap-outs* mais rápidos e o seu comportamento como uma *victim-cache*. Através de um conjunto grande de simulações detalhadas, mostramos que um multiprocessador baseado em OWCACHE pode facilmente ultrapassar o

desempenho de um multiprocessador baseado em OPTNET para a maioria das aplicações *out-of-core*; as diferenças de desempenho em favor de OWCACHE podem ser tão grandes quanto 64% e dependem do tipo de *prefetching* de disco utilizado.

Demonstramos também que o anel ótico pode ser aplicado com sucesso a um multiprocessador tradicional, conectado por uma grade eletrônica, de forma modular e flexível. Com essa finalidade, propomos NWCACHE: um dispositivo baseado no anel ótico que pode ser inserido no subsistema de entrada/saída desses multiprocessadores. As vantagens mais importantes de NWCACHE são os seus rápidos *swap-outs*, *victim-cache* e a redução da contenção. Os resultados do tempo de execução mostram que as implementações NWCACHE e OWCACHE produzem ganhos de desempenho similares, mesmo quando essas implementações otimizam o desempenho de formas ligeiramente diferentes.

Com base nestes resultados, no estudo de espaço de parâmetros e no custo continuamente decrescente dos componentes óticos, a nossa principal conclusão é que o anel ótico é altamente eficiente sob várias hipóteses arquiteturais e para a maioria de aplicações paralelas *out-of-core*. Note no entanto que, embora este estudo esteja focado na otimização de *swap-outs* de páginas, o cacheamento de dados com um anel ótico pode ser também benéfico para outros tipos de tráfego com escritas ao disco.

Capítulo 7

Implementação de uma Memória de Rede na Internet

A crescente popularidade da Internet e o aparecimento de aplicações distribuídas que demandam características específicas de serviço (e.g., grandes quantidades de dados, informações constantemente atualizadas) estão fazendo com que os servidores de rede se convertam em gargalos. Embora a utilização de servidores de rede escaláveis é uma opção para atenuar este problema [65], a grande largura de faixa existente nos enlaces entre os roteadores de alta velocidade [66] e o surgimento de tecnologias como as redes ativas [5, 6] oferecem outras alternativas para aumentar o desempenho desse tipo de aplicações de forma ortogonal ao desenvolvimento de servidores de rede mais poderosos.

Assim, neste capítulo propomos a utilização da grande largura de faixa dos enlaces entre roteadores para a construção de uma memória de rede que, junto com a tecnologia de redes ativas, permitirá eliminar os gargalos produzidos nos servidores de rede que tratam dados dinâmicos, melhorando notavelmente o desempenho das aplicações distribuídas que executam sobre essas redes. Mais especificamente, a nossa idéia é manter circulando, ao longo dos enlaces entre roteadores, os dados dinâmicos mais acessados pela aplicação. A atualização e manutenção da coerência desses dados será realizada através de roteadores ativos como os propostos em [67, 68, 69]. Basicamente, um roteador ativo é um dispositivo que permite realizar um processamento personalizado sobre as diversas mensagens que circulam através dele. Desta forma, parte do processamento feito pelos servidores pode ser realizado em um número potencialmente grande de roteadores por onde passam as mensagens das aplicações. Esta proposta está baseada na observação de que existem certas aplicações distribuídas, tais como os leilões eletrônicos, que traba-

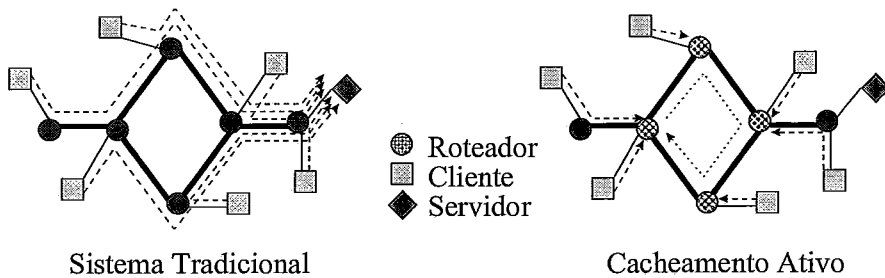


Figura 7.1: Tráfego para um Leilão de Mercadorias

lham com um conjunto relativamente pequeno de dados, normalmente acessado por um grande número de nós, que não pode ser cacheado devido a sua natureza dinâmica. Assim, o armazenamento desses dados dentro da própria rede pode melhorar o desempenho dessas aplicações, principalmente através da diminuição da contenção nos servidores e da redução da latência observada pelos clientes.

Para entender melhor esta proposta considere, por exemplo, a aplicação que realiza o leilão de um determinado conjunto de mercadorias através da Internet. As páginas para esse leilão podem ser acessadas por um número extremamente grande de possíveis compradores, sendo que a página com as informações sobre as últimas ofertas para cada mercadoria vai mudar constantemente e não poderá ser cacheada pelos clientes nem pelos *proxies* da rede. Desta forma, o acesso a essa página vai gerar uma contenção enorme no servidor que ainda vai ter que processar as ofertas geradas pelos diferentes clientes. Além disso, nesta aplicação pode existir um número bastante expressivo de ofertas que não são mais válidas, pois levam consigo valores inferiores ao maior até então recebido (a característica distribuída da aplicação e os retardos na comunicação podem gerar este tipo de inconsistência). Sob este cenário, a nossa proposta é manter a página com as últimas ofertas para cada mercadoria circulando entre os diferentes roteadores da rede (figura 7.1), sendo que a atualização dessa informação é feita através dos roteadores ativos. Esses roteadores interceptariam as mensagens enviadas pelos clientes, tanto pedindo essa página como fazendo novas ofertas, e eles mesmos retornariam uma resposta baseando-se na informação armazenada na memória de rede. Desta forma, a contenção gerada no servidor e as latências observadas pelos clientes seriam drasticamente diminuídas, fazendo com que o número de transações por unidade de tempo possa ser aumentado substancialmente.

7.1 Fundamentos

Nesta seção são apresentados alguns fundamentos sobre a Internet e as redes ativas. Alguns exemplos de aplicações distribuídas que podem se beneficiar da nossa proposta são também discutidos.

7.1.1 A Internet

A Internet pode ser vista como uma coleção de domínios de roteamento interconectados, onde cada um desses domínios é um conjunto de nós (e.g., roteadores, *gateways*, elementos de processamento) que compartilham informações e políticas de roteamento sob uma única administração [70]. Cada domínio de roteamento pode ser classificado como um domínio de extremo ou como um domínio de trânsito. Um domínio de extremo só transporta o tráfego que se origina ou termina naquele domínio. Um domínio de trânsito, por outro lado, não apresenta essa restrição e a sua principal função é interconectar eficientemente os domínios de extremo.

Os domínios de extremo normalmente correspondem às redes de instituições ou algumas outras coleções de LANs (*Local Area Networks*), enquanto que os domínios de trânsito são as denominadas WANs (*Wide Area Networks*) ou MANs (*Metropolitan Area Networks*). Assim, os domínios de trânsito estão formados por um conjunto de roteadores de alta velocidade, os quais podem estar conectados a um número determinado de domínios de extremo através de nós especiais, localizados nos domínios de extremo, chamados *gateways*. Em geral, um domínio de extremo pode estar conectado a um ou mais domínios de trânsito, e estes últimos podem ser organizados em hierarquias, como é o caso das MANs e WANs.

Desta forma, a estrutura topológica da Internet pode ser modelada por um grafo, onde os seus nós representam roteadores ou *gateways*, e as suas arestas representam as diferentes conexões entre esses elementos (figura 7.2). Assim também, os elementos de processamento podem ser modelados como folhas conectadas a um único nó roteador.

Informações adicionais referentes à estrutura e características da rede podem também ser adicionadas ao grafo. Por exemplo, a rede Abilene (primeiro protótipo da Internet-2) [66] possui enlaces OC-48 (2,48 Gbits/s) para interconectar os seus roteadores princi-

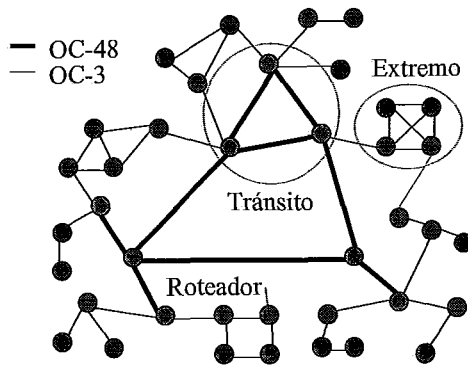


Figura 7.2: Estrutura Topológica da Internet

país (roteadores Cisco 12000 *Series* com uma capacidade de roteamento de 60 Gbits/s) e enlaces OC-3 (155 Mbits/s) para conectar os *gateways* dos domínios de extremo aos roteadores dos domínios de trânsito principais. Note, no entanto, que a rede Abilene prevê a atualização dos enlaces entre os seus roteadores principais a OC-192 (9,6 Gbits/s) ou superior nos próximos anos.

7.1.2 Redes Ativas

As redes ativas definem um novo tipo de arquitetura onde os nós internos da rede (i.e., os elementos de roteamento da rede) são capazes de realizar computações específicas sobre os pacotes que circulam através deles [6]. O processamento realizado por esses nós pode estar baseado no seu estado interno e/ou nas informações de controle transportadas nos pacotes. Como resultado desse processamento, os elementos de roteamento podem escalonar zero ou mais pacotes para serem transmitidos a outros nós e, inclusive, mudar o seu estado interno não-transiente. É também importante mencionar que o processamento feito pelos nós pode ser específico para os pacotes de cada usuário e/ou aplicação. Desta forma, as redes ativas permitem modificar dinamicamente o comportamento da rede observado pelos usuários.

Atualmente, existem várias propostas para a implementação de uma rede ativa. Calvert *et al.* [5] têm caracterizado estas propostas com base em três atributos: o grau de programabilidade da rede, a capacidade de manter ou não um estado interno nos nós, e a granularidade de controle. A programabilidade da rede pode variar desde um conjunto fixo de parâmetros configuráveis até uma linguagem capaz de descrever qualquer computação efetiva. A capacidade para manter um estado no interior dos nós se refere, por

sua vez, ao estado não-transiente que poderia ser instalado por alguns pacotes e acessado por outros. Finalmente, a granularidade de controle pode permitir que um único pacote modifique o comportamento do nós roteadores até que essa mudança seja explicitamente sobrescrita ou, em um outro extremo, um pacote pode modificar o comportamento dos roteadores apenas no seu próprio tratamento. De acordo com esta última característica, os dois modelos básicos para redes ativas são o uso de roteadores programáveis [68] e o uso de pacotes ativos especiais chamados de cápsulas [67].

O uso de roteadores programáveis permite manter o formato dos pacotes existentes mediante a provisão de um mecanismo que carrega, ao longo dos diferentes nós da rede e previamente ao envio dos pacotes, as rotinas ou programas a serem executados pelos diferentes nós. Assim, o envio dos pacotes é feito da mesma forma que nas redes convencionais, sendo que quando um pacote chega a um nó ativo, o seu cabeçalho é examinado primeiro e, só então, a rotina correspondente é executada para operar sobre o conteúdo desse pacote. O uso de pacotes ativos (ou cápsulas), por outro lado, substitui o pacote passivo das arquiteturas convencionais por rotinas pequenas que podem ser encapsuladas nos mesmos pacotes e executadas em cada um dos nós ao longo do seu caminho. Esses pacotes ativos podem inclusive conter dados de usuário.

Tanto no caso dos roteadores programáveis como dos pacotes ativos, os elementos de roteamento ativos podem interoperar com os elementos de roteamento tradicionais, os quais simplesmente passam para frente, de forma transparente, os pacotes enviados pelas diferentes aplicações.

7.1.3 Exemplos de Aplicações Distribuídas

Entre as aplicações que podem se beneficiar da nossa proposta podemos citar o cacheamento ativo de informação, como é o caso do exemplo descrito na introdução deste capítulo, e a sincronização de aplicações distribuídas.

No caso do cacheamento ativo de informação, além dos leilões que são aplicações que envolvem milhões de dólares na atualidade [71], bancos de dados que mantêm informações que mudam constantemente (e.g., listagens de estoque, notícias atualizadas sobre eventos determinados, etc.) podem também se beneficiar da nossa memória de rede. Da mesma forma que nos leilões, essas aplicações podem armazenar o seu conjunto de

dados mais acessado na memória de rede e atualizá-lo conforme às políticas definidas no servidor.

Além de diminuir a contenção no servidor e de reduzir a latência observada pelos clientes, esta estratégia também facilita a interação das aplicações com os dados compartilhados, pois a nossa memória de rede provê a imagem de uma memória centralizada única baseada em um modelo de consistência sequencial. Esta última característica, por sua vez, simplifica a programação das aplicações. Assim, a nossa memória de rede pode também ser utilizada na sincronização de aplicações iterativas distribuídas. Exemplos de tais aplicações são os jogos de realidade virtual entre usuários remotos (e.g., Doom, War-Birds, Duke 3D, Heretic), trabalho cooperativo suportado por computador (e.g., controle de acesso aos objetos, gerenciamento de grupos), entre outras.

7.2 Memória de Rede

A implementação da memória de rede não envolve nenhuma mudança na infraestrutura da rede, supondo a existência de um conjunto mínimo de roteadores ativos ao longo dos domínios de trânsito usados pela aplicação. Neste caso, vamos supor, unicamente por facilidade de apresentação, que o nosso modelo de redes ativas corresponde ao dos roteadores programáveis. Da mesma forma, também suporemos que os roteadores são capazes de manter um estado interno não-transiente e que eles suportam uma linguagem de programação relativamente flexível.

Assim sendo, a nossa memória de rede pode ser implementada em qualquer um dos domínios de trânsito usados pela aplicação que possua largura de faixa suficiente para não se tornar um gargalo de comunicação. Para a implementação da memória de rede, as aplicações deverão primeiro carregar, durante a sua fase de inicialização, as rotinas de processamento respectivas em cada um dos roteadores ativos do domínio de trânsito selecionado como linha de retardo. A seguir, alguns dos elementos de processamento usados pela aplicação (e.g., o servidor da aplicação) podem enviar um ou mais pacotes, contendo as informações a serem mantidas pela memória de rede, a esses roteadores ativos. A partir desse instante então, a aplicação pode começar a sua execução normal, considerando que os pacotes enviados através da rede deverão ser associados às rotinas corresponden-

tes mediante etiquetas específicas no cabeçalho das mensagens. Adicionalmente, antes ou durante a fase de terminação da aplicação, os elementos de processamento devem remover as rotinas previamente carregadas nos roteadores ativos, destruindo desta forma a memória de rede implementada durante a fase de inicialização.

Rotinas de Processamento Ativo. As rotinas carregadas pela aplicação nos roteadores ativos devem ser capazes de manter circulando de forma coerente as informações contidas na memória de rede. Além disso, essas rotinas devem também implementar algumas outras funções básicas dependentes da aplicação (e.g., a atualização e leitura de informações), assim como a terminação da memória de rede.

A rotina que mantém circulando os dados na memória de rede é fundamental na nossa proposta. Ela, basicamente, após receber um pacote identificado por uma etiqueta especial, envia uma mensagem de reconhecimento (*acknowledgement*) à fonte do pacote original e uma cópia do pacote ao próximo roteador no domínio de trânsito escolhido pela aplicação para a implementação da memória de rede. Após enviada a cópia do pacote, o roteador inicializa um alarme que só é desativado quando uma mensagem de reconhecimento para esse pacote é recebida. Se nenhuma mensagem de reconhecimento é recebida até o instante em que expira o tempo do alarme, o pacote é novamente enviado. Isto garante a perpetuação da informação na memória de rede mesmo ante falhas na transmissão dos dados.

As outras rotinas carregadas nos roteadores ativos vão depender da funcionalidade requerida pela aplicação, mas elas devem prover, fundamentalmente, mecanismos que permitam a atualização, leitura e terminação da memória de rede. As rotinas de atualização devem facilitar a modificação parcial ou total dos dados contidos nos pacotes, podendo inclusive enviar uma resposta, dependente dos dados existentes na memória, ao nó que originou o pedido. As rotinas de leitura, por outro lado, devem permitir o acesso parcial ou total às informações mantidas na memória de rede. Neste caso, obviamente, é indispensável o envio de uma resposta ao nó fonte do pedido. Finalmente, as rotinas de terminação devem permitir a destruição da memória de rede garantindo, ou não, a coerência e validade das informações armazenadas nela.

Por exemplo, no caso do leilão de mercadorias através da Internet, a rotina de

atualização deve alterar o valor de uma determinada mercadoria ante uma oferta maior que o seu valor atual. No caso da oferta ser aceita, uma mensagem indicando o êxito da submissão deve ser retornada. Caso contrário, uma mensagem de falha deverá ser enviada ao respectivo nó cliente. A rotina de leitura, por outro lado, deve facilitar que tanto os clientes como o servidor possam atualizar as suas informações locais periodicamente com os dados mantidos na memória de rede. Os clientes, por exemplo, podem querer ler o último valor associado a uma determinada mercadoria antes de fazer uma outra oferta. Finalmente, a rotina de terminação deve desativar as rotinas de atualização de todos os roteadores ativos envolvidos na aplicação e, depois de um tempo determinado, enviar os valores finais para o servidor de rede. Desta forma, a largura de faixa utilizada pela aplicação é liberada e o servidor termina possuindo os resultados finais do leilão.

Um aspecto importante em aplicações do tipo leilão eletrônico é a confiabilidade do sistema distribuído. Como a nossa memória de rede só garante a tolerância a falhas de enlaces de comunicação, técnicas de tolerância a falhas mais elaboradas devem ser implementadas a nível dos servidores e clientes para garantir a coerência e a perpetuação dos dados. Um exemplo de tais técnicas poderia ser o cruzamento periódico de informações diretamente entre os clientes e servidores. Desta forma, a própria aplicação pode garantir a confiabilidade dos dados transferidos através da rede [72].

Características Desejadas nas Redes Ativas. Como a proposta de redes ativas é relativamente nova e ainda é uma área de pesquisa, existem várias alternativas e soluções para a sua implementação. A seguir serão discutidas as nossas principais suposições e/ou requerimentos para a implementação da memória de rede.

Como foi mencionado anteriormente, a nossa proposta se baseia no modelo de roteadores programáveis e, de forma similar à proposta feita para NetScript [68], assume que a rede de comunicação pode ser observada como uma coleção de nós ativos interconectados por enlaces virtuais formando uma rede, também virtual, de mais alto nível. Para isso, uma nova camada de comunicação deve ser implementada sobre a arquitetura de rede existente (e.g., sobre a camada de rede IP). Esta nova camada de *software* deve prover suporte para a instalação, execução e controle das rotinas ativas, para a gerência e transmissão de pacotes entre os nós virtuais, e para a administração e alocação dos recursos de

rede. Em resumo, esta nova camada deve prover a abstração de uma máquina virtual onde as aplicações são uma coleção de *threads*, distribuídas sobre os diferentes nós virtuais, e encarregadas de processar os pacotes que circulam através da rede.

Assim, a instalação de rotinas nos roteadores ativos pode ser feita através de portas traseiras (*back-doors*) que só são acessíveis ao administrador da rede e/ou a usuários cadastrados com autenticação e verificação prévia dos mesmos. Este esquema, além de ser simples e facilitar o projeto dos roteadores ativos, permite garantir certo nível de proteção e confiabilidade na execução das rotinas por parte dos elementos ativos. A associação dos pacotes às rotinas, por outro lado, pode ser feita através de etiquetas no cabeçalho ou em posições específicas do corpo do pacote. Particularmente, achamos que a inclusão de uma etiqueta no campo opcional dos pacotes IP é a alternativa mais simples. Finalmente, no que se refere à alocação de recursos, o problema é similar ao enfrentado pelos sistemas distribuídos multiusuário. Assim, esquemas empregados pelos sistemas operacionais distribuídos para a gerência, alocação e proteção dos recursos podem também ser utilizados por estas máquinas virtuais.

Infelizmente, ainda não existe um padrão definitivo para as redes ativas, mas esforços estão sendo feitos para encontrar um modelo de programação comum, uma forma de acesso rápida e eficiente aos recursos oferecidos pelos roteadores ativos, um mecanismo de proteção para os recursos da rede, e um esquema de instalação que permita aos usuários colocar uma determinada funcionalidade no lugar certo dentro da rede [5, 6]. Espera-se que em um futuro não muito distante, esta tecnologia esteja à disposição dos usuários convencionais.

Cientes e Servidores. A utilização da memória de rede por parte das aplicações distribuídas requer algumas mudanças no código dos seus clientes e servidores. Do lado dos clientes essas mudanças são mínimas, pois só é preciso incluir o identificador das rotinas que trataram esses pacotes no cabeçalho dos mesmos. Isso pode ser feito facilmente sem o conhecimento explícito do usuário e mesmo usando navegadores convencionais através de *plug-ins* ou *Java scripts* fornecidos pelo servidor.

Do lado do servidor, essas mudanças são mais radicais e dependem do tipo de aplicação. No caso do leilão, por exemplo, o servidor tem que inicializar a memória de

rede através da API (*Application Programming Interface*) fornecida pela camada de redes ativas. Após essa inicialização, ele deve também implementar algum mecanismo que permita processar as ofertas recebidas diretamente (ofertas que não passaram por nenhum roteador ativo). Uma alternativa para o processamento correto de tais ofertas é re-enviar todos esses pacotes ao roteador ativo mais próximo, pois só a memória de rede possui os valores atualizados do leilão. No entanto, uma solução mais eficiente pode ser retornar ao cliente um novo endereço de serviço que obriga aos pacotes dos clientes a passar pelo menos por um roteador ativo. O *forwarding* de um endereço para um outro servidor é uma característica suportada inclusive a nível da linguagem HTML (*Hyper-Text Markup Language*).

Um outro tópico de interesse na implementação dos clientes e servidores é a utilização de TCP (*Transport Control Protocol*) ou UDP (*User Datagram Protocol*) como o seu protocolo de comunicação. TCP tem a vantagem de permitir uma comunicação confiável e ser a base do protocolo HTTP (*Hyper-Text Transport Protocol*), mas o seu uso requer o estabelecimento de uma conexão entre o cliente e o servidor antes de qualquer troca de informação. Como a nossa proposta intercepta as mensagens dirigidas ao servidor, a comunicação entre os pontos terminais da conexão não poderia ser garantida.

Por este motivo e já que a maioria de aplicações alvo da nossa proposta requerem uma comunicação do tipo pedido-resposta, nosso estudo vai estar restrito às aplicações que podem se beneficiar do uso de UDP como o seu protocolo básico de transporte. Isso simplifica bastante o código executado nos roteadores ativos e, como será mostrado na seção experimental, não influencia no desempenho nem na correção das aplicações.

Note, no entanto, que aplicações que só podem usar TCP como o seu protocolo de transporte também se beneficiariam da memória de rede. Para isso, seria necessário manter o estado das conexões TCP em cada um dos roteadores ativos. Obviamente, isso geraria complicações adicionais ao processamento que tem que ser feito nos roteadores ativos, mas pode ser um tema para estudos futuros.

Consumo de Largura de Faixa. Um outro ponto importante na nossa proposta é o consumo de largura de faixa por parte da memória de rede. Matematicamente, esse consumo de largura de faixa pode ser expresso como:

Parâmetros	Valor
Taxa de transmissão dos roteadores	2,48 Gbits/s
Taxa de transmissão dos <i>gateways</i>	155 Mbits/s
Tempo de serviço no servidor	200 μ seg
Tempo de <i>time-out</i> nos clientes	10 seg
Tempo de <i>time-out</i> nos roteadores	10 seg
Tempo de roteamento	800 η seg
Tempo de <i>forwarding</i>	100 + 50*R μ seg
Tempo de atualização	100 μ seg
Tempo de leitura	100 μ seg
Número de unidades de roteamento	4
Tamanho das filas de mensagens	1024

Tabela 7.1: Parâmetros Usados pelos Simuladores

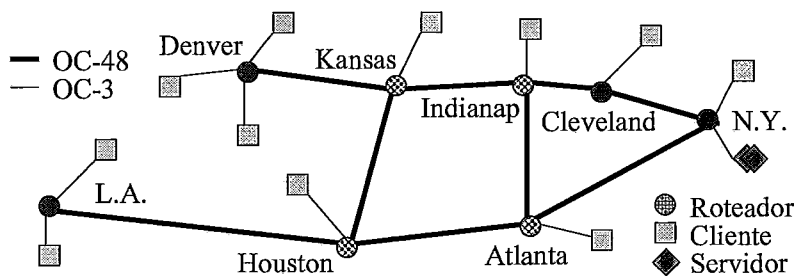


Figura 7.3: Estrutura Topológica da Internet-2

$$\text{número_de_pacotes} \times \text{tamanho_do_pacote} \times \text{round-trips_por_segundo}$$

onde o número de *round-trips* por segundo é determinado pelas distâncias entre os roteadores, pelo tempo médio de cada roteamento e pelo número de roteadores que implementam a memória de rede. Normalmente, as distâncias entre roteadores e o número de roteadores usados pela memória de rede são valores fixos e conhecidos. Já o tempo médio de roteamento depende da quantidade de contenção encontrada pelos pacotes que implementam a memória de rede. Esta característica é muito importante, pois como veremos na seção 7.4.3, a nossa memória de rede é capaz de se adaptar a mudanças na largura de faixa à disposição da aplicação.

7.3 Metodologia

Com a finalidade de avaliar a nossa proposta para diferentes parâmetros de rede e ter estatísticas mais detalhadas sobre as aplicações que executam nessas redes, simulações

dirigidas por eventos de várias configurações e topologias foram implementadas. A tabela 7.1 apresenta os parâmetros e valores base supostos nas simulações das diferentes configurações. Tanto a taxa de transmissão dos roteadores como a dos *gateways* e o tempo de roteamento são os mesmos encontrados na Internet-2. Os tempos de serviço e processamento (i.e., leitura, atualização e *forwarding*), por outro lado, correspondem aos tempos empregados por uma estação de trabalho Ultra-SPARC para essas mesmas tarefas. O tempo de *forwarding*, especificamente, se refere ao tempo requerido pelo roteador para rotear um pacote com informações da memória de rede. Note que esse tempo depende do número de mensagens que o roteador envia aos clientes como resultado do processamento feito sobre esse pacote (R). Por sua vez, a figura 7.3 mostra a topologia base usada em nossos experimentos. A topologia base é a mesma implementada pela rede Abilene. De acordo com a expressão apresentada na seção anterior e supondo uma rede livre de contenção (i.e., um tempo de *round-trip* igual a 15,7 msec), a quantidade de informação que pode ser armazenada nesta rede é 4,7 MBytes. O efeito da variação dos parâmetros de rede mais importantes e da topologia usada pelas aplicações será analisado na seção 7.4.3.

7.3.1 A Aplicação

A aplicação que será avaliada com detalhe na próxima seção é o leilão de mercadorias através da Internet. Um leilão pode ser definido como uma instituição de mercado com regras explícitas que determinam a alocação e preços de um determinado conjunto de recursos com base nas ofertas provenientes dos participantes do mercado. Assim, existem vários tipos de leilões, sendo que o mais popular, e no qual vai se concentrar nosso estudo, é o denominado leilão EOO (*English Open Outcry*) [71]. Este tipo de leilão corresponde ao caso onde se têm conjuntos de vendedores e compradores mutuamente exclusivos, sempre é possível saber qual é a última oferta para uma mercadoria determinada, e os compradores vão incrementando os valores das suas ofertas com a finalidade de adquirir as mercadorias.

Uma característica importante dos compradores que participam de um leilão através da Internet é que eles podem ser pessoas ofertando através de um navegador convencional ou agentes de *software* autônomos configurados para interagir com um servidor de leilões

Parâmetros	Valor
Número de nós cliente	11
Número de compradores por nó cliente	512
Número de ofertas de cada comprador	1024
Número de mercadorias leiloadas	64
Área de dados de cada mercadoria	32 Bytes
Tamanho das mensagens de pedido e resposta	256 Bytes

Tabela 7.2: Parâmetros da Aplicação

determinado. Uma das vantagens de usar agentes autônomos é a sua capacidade para fazer ofertas simultâneas por várias mercadorias e com um tempo de resposta menor. A popularidade de tais agentes faz com que o tráfego gerado nos servidores seja cada vez maior. Assim, nossas simulações consideram que os clientes do leilão são agentes de *software* autônomos e que tanto o servidor como os clientes implementam unicamente as operações básicas requeridas pela aplicação, i.e., consulta das últimas ofertas válidas e submissão de novas ofertas.

Tanto a consulta como a submissão de ofertas geram uma resposta por parte do servidor aos clientes do leilão. No caso da submissão de uma nova oferta, a mensagem de resposta carrega, além do resultado da transação (êxito ou fracasso na submissão), a última maior oferta válida para essa mercadoria. A finalidade desta otimização é permitir desempenhos maiores nos servidores de leilão tradicionais quando estes estão sujeitos a grandes cargas de trabalho, como acontece quando se utilizam agentes de *software* autônomos. Os parâmetros mais importantes da aplicação usada em nossas simulações são apresentados na tabela 7.2.

7.3.2 Configurações Básicas

Para poder estabelecer as verdadeiras vantagens da nossa proposta em relação a sistemas similares, simulamos, além da configuração tradicional e da nossa proposta, outras duas configurações que permitem o leilão de mercadorias usando exclusivamente roteadores ativos. A seguir são descritas cada uma destas configurações.

Sistema Tradicional. A configuração tradicional consiste de um único servidor de leilões ao qual todos os clientes enviam as suas ofertas. A rede não precisa ter nenhuma capaci-

dade de processamento, além dos serviços básicos da camada de rede, já que o servidor faz todo o processamento requerido pela aplicação. Este sistema será analisado em duas variantes: com um servidor seqüencial e com um servidor distribuído (e.g., um *cluster* de estações de trabalho). No caso do servidor distribuído, cada um dos processadores é responsável por leiloar um conjunto diferente de mercadorias. Assim, este esquema elimina a necessidade de sincronização entre as diferentes unidades de processamento. Note que, embora o desempenho do sistema distribuído seja melhor do que o do sistema seqüencial, o sistema distribuído apresenta a desvantagem de perder desempenho quando ocorrem desbalanceamentos nas taxas de acesso às diferentes mercadorias.

Roteadores Ativos – Filtro de Ofertas. Esta configuração é bastante similar ao sistema anterior que se baseia unicamente em um servidor seqüencial. A sua principal diferença é que os roteadores ativos presentes na rede são usados como filtros de ofertas válidas [73]. Mais especificamente, o servidor atualiza nos roteadores ativos, de tempos em tempos, o valor mínimo requerido pelas ofertas para que elas passem a ser tratadas no servidor. As ofertas que não ultrapassam esse valor mínimo são respondidas diretamente pelos roteadores ativos com uma mensagem de falha. Neste caso, o tempo entre as atualizações desses valores mínimos tem um papel importante no desempenho do sistema. Desta forma, analisaremos duas variantes: (a) uma atualização é enviada a todos os roteadores ativos cada vez que uma oferta válida chega ao servidor, e (b) uma atualização é enviada a todos os roteadores ativos a cada certo número de ofertas inválidas (no nosso simulador esse valor é 200) que chegam ao servidor.

Roteadores Ativos – Processamento Distribuído. Em contraste com o sistema anterior, esta configuração permite que o processamento das ofertas seja feito totalmente nos roteadores ativos. Para isso, o servidor distribui, durante a inicialização da aplicação, o conjunto de mercadorias pelas quais cada roteador ativo é responsável. Cada roteador se encarrega de um conjunto distinto de mercadorias. Durante a execução do leilão, o servidor só redireciona os pedidos que chegam a ele aos roteadores correspondentes. Os roteadores ativos, além de responder às diferentes mensagens de oferta dirigidas a eles, também se encarregam de rotear as mensagens dirigidas aos outros roteadores. Assim como a configuração tradicional com um servidor distribuído, este sistema também

Parâmetros	Valor
Taxa de transmissão	100 Mbits/s
Tempo de serviço	300 μ seg
Tempo de <i>time-out</i>	10 seg
Tempo de roteamento	150 μ seg
Tempo de <i>forwarding</i>	$260 + 140 * R$ μ seg
Tempo de atualização	260 μ seg
Tempo de leitura	260 μ seg
Unidades de roteamento	1
Tamanho da fila de mensagens	1024

Tabela 7.3: Parâmetros da Rede Virtual

apresenta degradação de desempenho frente a desbalanceamentos nas taxas de acesso às diferentes mercadorias.

Roteadores Ativos mais Memória de Rede. Como descrita anteriormente, a nossa proposta permite que qualquer roteador ativo seja responsável por qualquer mercadoria, pois a memória compartilhada implementada através da linha de retardo permite manter coerentes as informações armazenadas de forma simples e elegante. Esse comportamento ocorre porque os pacotes que implementam a memória de rede atuam como *tokens* seqüencializadores dos acesso à memória compartilhada, garantindo desta forma exclusão mútua nas atualizações dos valores mínimos das ofertas. É também importante mencionar que, nesta configuração, as ofertas que chegam diretamente ao servidor (sem passar por nenhum roteador ativo) são redirigidas a um novo endereço que obriga as mensagens a passar por pelo menos um roteador ativo.

7.4 Resultados

Antes de apresentar os resultados para as diversas configurações analisadas na seção anterior, vamos mostrar os resultados para uma implementação da nossa proposta sobre um *cluster* de estações de trabalho que valida as nossas simulações.

7.4.1 Validação do Simulador

Com a finalidade de validar os resultados obtidos através das nossas simulações, vamos comparar esses resultados com os de uma implementação real do leilão de mercadorias. A nossa implementação se baseia em uma rede virtual, com a mesma estrutu-

ra topológica da figura 7.3, implementada sobre um *cluster* de 23 estações de trabalho Ultra-SPARC que executam Solaris e estão interconectadas por uma rede *Fast-Ethernet*. Os parâmetros de rede da nossa implementação, que também são usados pelas simulações nesta comparação, são apresentados na tabela 7.3. Por sua vez, os parâmetros da aplicação executada sobre esta rede são os mesmos mostrados na tabela 7.2.

A implementação considera tanto uma configuração tradicional (com um servidor seqüencial ou distribuído) como uma baseada na nossa proposta. Para isso, cada roteador (ativo ou não) da rede virtual foi implementado em uma estação de trabalho independente. O servidor de leilões também foi implementado sobre estações de trabalho independentes, para desta forma, isolar os efeitos da contenção gerada nele. Finalmente, vários agentes clientes, que fazem ofertas indistintamente para todas as mercadorias, foram implementados sobre cada uma das estações restantes. Os nós clientes fazem ofertas à sua capacidade máxima de processamento, sendo que cada comprador espera pela resposta correspondente antes de emitir uma nova oferta.

No que se refere à implementação dos roteadores, eles só fornecem a funcionalidade básica requerida pela nossa proposta, usando para isso os serviços básicos da camada IP já existente. Especificamente, as tabelas de roteamento são carregadas estaticamente durante a fase de inicialização do roteador. No caso dos roteadores ativos, a associação dos pacotes IP às rotinas correspondentes é feita através de uma etiqueta localizada nas primeiras posições do frame de dados. As rotinas de processamento, por sua vez, são também carregadas na inicialização do roteador.

Comparando os resultados desta implementação com os do nosso simulador observamos que, para o caso de uma configuração tradicional com um servidor seqüencial, o número de roteamentos realizados pela implementação é 0,8% inferior ao número de roteamentos contabilizados na simulação. Esta diferença se deve, basicamente, aos números distintos de mensagens perdidas na rede devido a falhas de comunicação. Na implementação são perdidas apenas 70K mensagens, enquanto que na simulação são perdidas 147K mensagens de um total de 3,3M mensagens. Por outro lado, na nossa implementação o número de ofertas atendidas por segundo chega a 3251, enquanto que na simulação esse valor é 3213; uma diferença de apenas 1,2% no desempenho destes sistemas.

Da mesma forma, observamos que para o caso de uma configuração tradicional com um servidor distribuído que inclui 4 elementos de processamento, a diferença entre o número de roteamentos realizados pela implementação e os contabilizados na simulação é de apenas 1,6%. Novamente, essa diferença aparece devido às diferentes taxas de perda de mensagens na rede de comunicação. O número de ofertas atendidas, por outro lado, chega a 3192 por segundo na nossa implementação, enquanto que na simulação esse mesmo valor é 3595; uma diferença de 11% entre esses dois valores.

No caso da nossa proposta, as diferenças no número de roteamentos são em média 7,9% quando comparamos os resultados da nossa implementação com os da simulação. A disparidade destes valores não se deve exclusivamente à perda de mensagens, como nos casos anteriores, mas também ao diferente número de *round-trips* que os pacotes da memória de rede realizam na nossa implementação e na simulação. De qualquer forma, o número de *round-trips* por segundo de um pacote na implementação e na simulação são bastante próximos, 81 e 91 *round-trips* por segundo, respectivamente. Por outro lado, o número de ofertas atendidas por segundo chega a 8,8K na implementação, enquanto que na simulação é 10,9K ofertas por segundo; uma diferença de 19%.

Nos resultados anteriores podemos observar que as diferenças no desempenho dos sistemas aumentam de 1,2 para 11 e finalmente para 19%, conforme aumenta o paralelismo no processamento das ofertas (i.e., de um servidor seqüencial para um servidor distribuído e finalmente para o cacheamento ativo). A causa para o aumento dessas diferenças é que a rede *Fast-Ethernet* não provê necessariamente uma ligação exclusiva para a comunicação entre dois nós quaisquer. Assim, quanto maior o paralelismo no processamento das ofertas, maior a concorrência pelo acesso à rede, ocasionando demoras no envio das mensagens por parte da interface de rede. Este fenômeno não acontece na nossa simulação, pois nela cada par de nós possui um canal exclusivo de comunicação, da mesma forma que ocorre na rede Abilene.

Assim, podemos concluir, a partir desses resultados, que nos casos onde a interferência entre os diferentes enlaces de comunicação devida ao envio de mensagens entre pares distintos de nós não é significativa, o simulador e a nossa implementação apresentam comportamentos bastante similares. Quando existe interferência entre os enlaces de comunicação, os resultados da nossa simulação diferem um pouco (menos de 20%) da

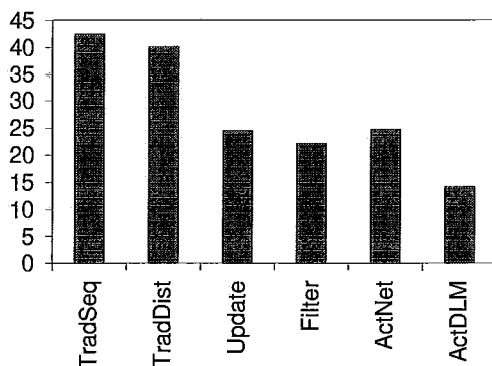


Figura 7.4: Número de Roteamentos (em Milhões) para Cada Configuração

implementação nas configurações avaliadas. Este fato nos permite afirmar que os resultados obtidos através do nosso simulador são uma boa indicação do desempenho esperado para implementações reais das nossas diferentes configurações.

7.4.2 Resultados Base

A figura 7.4 mostra o número de roteamentos realizados pelas diferentes configurações quando cada um dos 11 nós clientes gera 512K ofertas. A figura mostra os resultados para o sistema tradicional com um servidor seqüencial (“TradSeq”), o sistema tradicional com um servidor distribuído (“TradDist”), o sistema que usa roteadores ativos como filtros de ofertas com uma atualização a cada oferta válida (“Update”), o sistema que usa roteadores ativos como filtros de ofertas com uma atualização a cada 200 ofertas inválidas (“Filter”), o sistema que usa roteadores ativos para implementar um processamento distribuído (“ActNet”), e a nossa proposta (“ActDLM”).

Note que, como era esperado, o fato de usar um servidor com várias unidades de processamento (“TradDist”) não reduz significativamente o número de mensagens roteadas com relação ao caso de um servidor seqüencial (“TradSeq”). As diferenças existentes entre os sistemas “TradSeq” e “TradDist” (aproximadamente 6%) são produzidas exclusivamente pela menor taxa de perda de mensagens do sistema distribuído. Esta menor taxa de perda de mensagens, por sua vez, se deve à menor contenção gerada nos servidores. Por outro lado, o uso de roteadores ativos como filtros de ofertas ou para implementar um processamento distribuído das ofertas reduz o número de roteamentos em relação aos casos tradicionais, mas não tanto como a nossa proposta. Isto se deve às mensagens não filtradas nos sistemas “Update” e “Filter”, ou às mensagens que chegam inicialmente ao

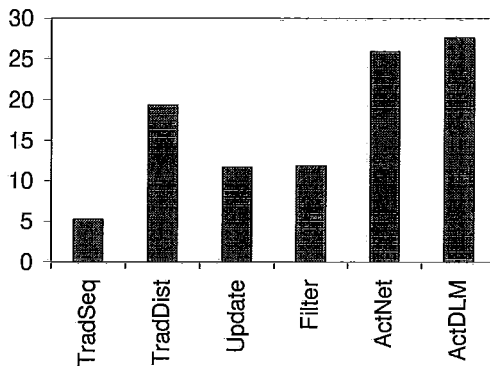


Figura 7.5: Número de Ofertas Atendidas por Segundo (em Milhares) para Cada Configuração

nó que não é o encarregado de processar essas ofertas no sistema “ActNet”. Além disso, os sistemas que usam os roteadores ativos como filtros de ofertas requerem o envio de atualizações por parte do servidor aos distintos roteadores, 30K atualizações no caso do sistema “Filter” e 3,9M atualizações no caso do sistema “Update”.

É também importante notar que mesmo usando uma memória de rede, onde os dados compartilhados estão continuamente circulando entre os roteadores ativos, o número de roteamentos necessários pela aplicação é muito menor na nossa proposta que nos outros sistemas. Este fato se deve a que somente 0,04% dos roteamentos correspondem ao *forwarding* de pacotes da memória de rede, pois devido às distâncias consideradas na topologia da figura 7.3, o número de *round-trips* realizados pelos pacotes da memória de rede é de apenas 5,6 por segundo durante todo o tempo simulado. Além disso, como todas as ofertas são tratadas no primeiro roteador ativo que encontram no seu caminho até o servidor, o número de *hops* percorridos pelas mensagens é muito menor que nos outros casos. Desta forma, a nossa proposta resulta em 66% menos roteamentos que nos casos tradicionais e aproximadamente 40% menos roteamentos que nos casos onde são usados exclusivamente roteadores ativos.

No que se refere ao desempenho dos sistemas, a figura 7.5 mostra o número de ofertas processadas por unidade de tempo para os mesmos sistemas descritos anteriormente. Como pode-se observar, a utilização de um servidor distribuído, com 4 unidades de processamento, aumenta notavelmente (273%) o desempenho do sistema com relação a um servidor seqüencial. Os ganhos com o uso de filtros (independente da estratégia de atualização) são menores, pois o servidor continua sendo um ponto de contenção devi-

do ao processamento das ofertas válidas (mais de 40% das ofertas não são filtradas nos sistemas “Update” e “Filter”). De qualquer forma, o ganho de desempenho dos sistemas que usam o filtro de ofertas com atualizações a cada oferta válida ou a cada 200 ofertas inválidas é de 125 e 129%, respectivamente, em relação ao desempenho de um sistema tradicional com um servidor seqüencial.

O sistema que mais se aproxima do desempenho atingido pela nossa proposta é aquele que usa os roteadores ativos como elementos de processamento distribuído. Note que mesmo com um número considerável de roteamentos, o sistema “ActNet” apresenta um ganho de desempenho de 398% em relação a um sistema tradicional com um servidor seqüencial. Este ganho significativo se deve ao processamento distribuído que é realizado pelos 4 roteadores ativos presentes na nossa topologia e à diminuição da contenção na rede pelo envio de um número menor de pacotes entre os roteadores.

Vemos também que os ganhos de desempenho da nossa proposta em relação ao sistema “TradSeq” podem chegar a 436% com apenas quatro roteadores ativos. Com relação aos sistemas que usam os roteadores ativos como filtros, os nossos ganhos são de 136 e 132% quando se utilizam atualizações a cada oferta válida ou a cada 200 ofertas inválidas, respectivamente. Com relação aos sistemas “TradDist” e “ActNet” o nosso sistema somente apresenta ganhos de 42 e 7%, respectivamente. No entanto, como veremos na próxima seção, estes sistemas sofrem tremendamente com desbalanceamentos de carga, coisa que não acontece com o nosso sistema.

Em resumo, devido à memória de rede se apresentar como uma memória compartilhada baseada em um modelo de consistência seqüencial, garantindo exclusão mútua entre os diferentes acessos, e ao fato de usar a capacidade de processamento dos roteadores ativos, esta proposta permite a implementação de aplicações distribuídas facilmente escaláveis e com um poder de processamento maior.

7.4.3 Variação de Parâmetros

Alguns dos parâmetros mais importantes da seção anterior serão analisados a seguir. Entre os principais temos o tipo de distribuição das ofertas geradas pelos clientes, a largura de faixa dos enlaces entre os roteadores disponível para a aplicação e o efeito da topologia utilizada.

Distribuição Não-Uniforme das Ofertas. Quando os acessos às diferentes mercadorias não produzem uma distribuição uniforme de carga nos diferentes elementos encarregados do processamento das ofertas, as configurações “TradDist” e “ActNet” sofrem de degradações no seu desempenho. Especificamente, quando dois dos 4 elementos de processamento recebem 50% mais ofertas, o sistema “TradDist” degrada o seu desempenho em aproximadamente 25%, enquanto que o sistema “ActNet” degrada o seu desempenho em quase 15%. Assim, os ganhos da nossa proposta com relação aos sistemas “TradDist” e “ActNet” passam de 42 e 7% a 89 e 25%, respectivamente.

Todos os outros sistemas, como era esperado, não alteram o seu desempenho, exceto pela configuração “Filter”, onde o desbalanceamento de carga melhora o desempenho do sistema em 7%. Este fenômeno acontece devido a uma melhor filtragem das ofertas, o que, por sua vez, se deve à menor quantidade efetiva de mercadorias acessadas por unidade de tempo.

Taxa de Transmissão dos Enlaces Entre os Roteadores. Mantendo a distribuição das ofertas nos diferentes elementos de processamento uniforme, variamos a taxa de transmissão entre os roteadores principais de OC-3 (155 Mbits/s) até OC-192 (9,6 Gbits/s). Nestes experimentos, podemos observar que o desempenho de todas as configurações praticamente não depende desta taxa de transmissão, pois em nenhum dos casos houve mudanças relevantes no número de ofertas processadas por unidade de tempo. Este fato se deve a que o tempo gasto pela aplicação com o envio de mensagens através da fibra é pouco significativo mesmo quando essas distâncias são consideráveis (centenas de quilômetros). Isto também demonstra que a contenção no servidor e nos roteadores é a causa maior para a perda de desempenho neste tipo de aplicações, pois até mesmo redes relativamente lentas não comprometem o desempenho das aplicações nas configurações consideradas.

Em particular, no caso da nossa proposta, as diferenças no número de ofertas processadas por unidade de tempo chegam a 0,6% quando variamos a taxa de transmissão entre os roteadores principais de OC-3 até OC-192. Um outro dado importante é que o número de *round-trips* dos pacotes da memória de rede cai de 5,7 para 5,3 por segundo (uma diferença de 7%) quando a taxa de transmissão é diminuída de OC-192 para OC-3.

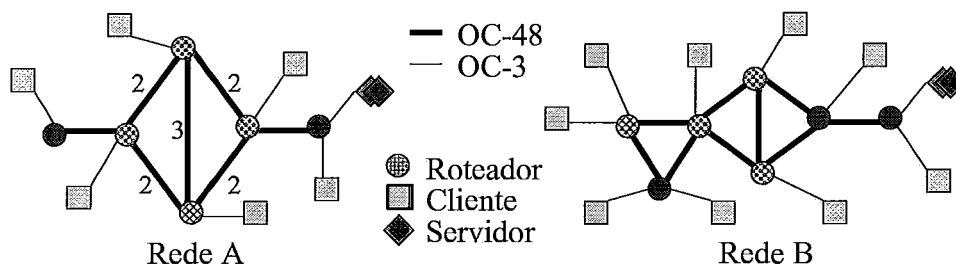


Figura 7.6: Exemplos de Topologias de Rede

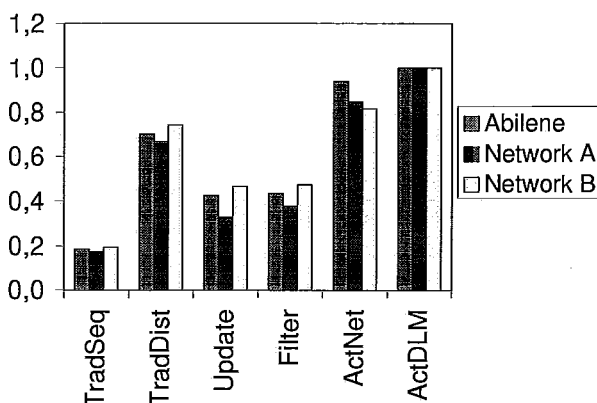


Figura 7.7: Desempenho de Cada Configuração para Diferentes Topologias

Desta forma comprovamos que a nossa memória de rede é capaz de se adaptar a diferentes larguras de faixa sem perdas consideráveis de desempenho.

Diferentes Topologias. A figura 7.6 mostra outras duas topologias arbitrárias utilizadas para a avaliação das configurações estudadas. Nestas topologias, todas as distâncias são 100 quilômetros, exceto onde estão marcados, explicitamente, valores diferentes. Nesses casos, o valor indica o número de centenas de quilômetros entre os roteadores correspondentes. Por sua vez, a figura 7.7 mostra o número de ofertas atendidas por segundo pelas diferentes configurações, tanto na topologia base (figura 7.3) quanto nas duas outras topologias mostradas na figura 7.6. Os resultados estão normalizados com relação ao desempenho atingido pela nossa proposta (“ActDSM”) para cada uma das topologias.

Como pode-se observar, a utilização de diferentes topologias na implementação das configurações analisadas anteriormente produz pequenas variações com relação aos resultados de desempenho obtidos para a topologia base. As diferenças maiores aparecem para os sistemas que usam os roteadores como filtros de ofertas, mas mesmo assim, elas não ultrapassam 23%. Em termos gerais, a tendência apresentada na topologia base é mantida em todas as outras topologias.

7.5 Trabalhos Relacionados

Existem três áreas de pesquisa que são bastante relacionadas com esta proposta: (a) o uso de memórias de linhas de retardo no projeto de sistemas de computação, (b) a utilização de redes ativas para a otimização de um determinado conjunto de aplicações e (c) o cacheamento de dados na Internet.

Como foi mencionado, as memórias de linha de retardo óticas foram implementadas com sucesso em sistemas de comunicação [34] e em computadores totalmente óticos [36]. Trabalhos anteriores como NetCache, OWCACHE e NWCACHE são também outros exemplos da utilização da grande largura de faixa das comunicações óticas no armazenamento de dados. Estes últimos trabalhos estão orientados, basicamente, à otimização do desempenho de multiprocessadores escaláveis. No entanto, a presente proposta não precisa de um anel ótico dedicado nem de *hardware* ótico adicional para a implementação da memória de rede. A nossa proposta usa a infraestrutura de rede que estará a disposição de qualquer usuário, em um futuro próximo, para a implementação da memória de rede.

A tecnologia de redes ativas, por sua vez, nasceu faz pouco tempo como uma alternativa aos vários problemas que foram identificados nas redes atuais. Assim, os seus objetivos principais são acelerar a renovação da infraestrutura de rede existente e tornar possível o desenvolvimento de novas aplicações. Entre as aplicações que poderiam se beneficiar desta tecnologia estão a distribuição e fusão de informação [74, 75], a provisão de qualidade de serviço [69], o gerenciamento de redes [76, 68], cacheamento de informação [77, 75], computação móvel [78, 67], proteção de redes (e.g., *firewalls*) [67], entre outras.

No entanto, a proposta descrita neste capítulo funde o conceito das redes ativas com a idéia de usar a própria rede para o armazenamento de informações compartilhadas. Desta forma, melhora-se ainda mais o desempenho de determinadas aplicações distribuídas através de uma redução na contenção gerada nos servidores e na rede, e de uma diminuição no tempo de resposta observado pelos clientes. Além disso, devido ao esquema proposto para manter as informações circulando entre os roteadores ativos, a nossa proposta se adapta automaticamente a mudanças na largura de faixa disponível para a aplicação. Este último fator é muito importante em sistemas como os assumidos, onde várias aplicações e usuários compartilham diversos recursos.

Finalmente, existem muitos trabalhos relacionados com o cacheamento de dados na Internet [79, 80, 81, 82], no entanto, a maioria deles trata somente do cacheamento de arquivos (ou dados) estáticos [79, 83]. O cacheamento dos dados estáticos pode ser realizado pelos diferentes elementos de processamento envolvidos na aplicação (e.g., navegadores, agentes autônomos, servidores, etc.) [82] ou pelos *proxies* existentes na rede [84]. Propostas para usar roteadores ativos no cacheamento de dados estáticos foram também feitas em [77, 75].

Nossa proposta se diferencia destes trabalhos uma vez que dados (ou arquivos) dinâmicos são armazenados na própria rede. Além disso, a nossa proposta usa os roteadores ativos como elementos de processamento capazes de desenvolver tarefas úteis na execução de aplicações distribuídas, melhorando o desempenho e a escalabilidade de tais aplicações.

7.6 Conclusões

Este capítulo apresentou a utilização da grande largura de faixa dos enlaces entre os roteadores para a construção de uma memória de rede que, junto com a tecnologia de redes ativas, permitirá melhorar o desempenho de várias aplicações distribuídas. Os resultados das nossas simulações mostraram que, para o caso do cacheamento de informações dinâmicas, o ganho no número de transações realizadas por unidade de tempo pode chegar a 436% com relação a um sistema convencional, quando são usados 4 roteadores ativos. Como também mostrou o nosso conjunto de simulações, estes ganhos são dificilmente atingíveis através de propostas baseadas exclusivamente em redes ativas ou em algum outro tipo de processamento distribuído. Com base nestes resultados, na tendência de aumento da largura de faixa dos enlaces entre roteadores, e considerando que as redes ativas são uma evolução natural da infraestrutura de redes atual, a nossa principal conclusão é que os desenvolvedores de aplicações distribuídas devem levar em consideração esta proposta como uma alternativa para atingir melhores desempenhos.

Capítulo 8

Conclusões e Trabalhos Futuros

O objetivo desta Tese foi explorar a capacidade das fibras óticas de atuar como memórias de linha de retardo no projeto de sistemas de computação paralelos e distribuídos. Com esse propósito, projetamos e avaliamos quatro redes óticas, três das quais são capazes de armazenar informação na própria rede. Especificamente, foram desenvolvidas: a) OPT-NET, uma rede de interconexão ótica para multiprocessadores, b) NetCache, uma rede de interconexão que também atua como uma cache de terceiro nível para os dados compartilhados de aplicações paralelas, c) OWCACHE, uma rede de interconexão ótica para multiprocessadores que também atua como uma cache compartilhada para escritas ao disco, e d) um sistema que permite o cacheamento ativo de dados dinâmicos acessados por aplicações distribuídas que executam sobre a Internet.

Através de simulações detalhadas de várias aplicações paralelas e distribuídas executando sobre esses sistemas, demonstramos que os ganhos de desempenho obtidos pela nossa proposta são plenamente satisfatórios. Em particular, NetCache supera facilmente os outros sistemas baseados em redes óticas, especialmente quando as aplicações têm uma grande quantidade de reutilização dos dados. Este fato se deve à redução efetiva do tempo gasto com falhas nas caches locais. Da mesma forma, pudemos observar que, no caso de OWCACHE, as características mais importantes que contribuem para melhorar o desempenho dos sistemas baseados nesta rede são os seus *swap-outs* mais rápidos e a sua capacidade de se comportar como uma *victim-cache*. Tanto NetCache quanto OWCACHE se baseiam em OPTNET, a qual apresenta uma excelente relação custo/desempenho.

Finalmente, para o caso do cacheamento ativo de dados dinâmicos na Internet, os ganhos de desempenho são produzidos principalmente pela diminuição da contenção no

servidor e na rede, e pela redução do tempo de resposta observado pelos clientes. Estes ganhos são dificilmente atingíveis através de propostas baseadas exclusivamente em redes ativas ou em algum outro tipo de processamento distribuído.

Em resumo, as principais vantagens de usar a própria rede de comunicação como uma cache compartilhada são: a) o tamanho da memória de rede não precisa ser extremamente grande, o que permite tempos de acesso relativamente baixos; b) a cache ótica permite evitar acessos aos níveis mais baixos do sistema de memória, podendo diminuir o tráfego de informação no restante da rede e nos barramentos dos níveis inferiores do sistema de memória; c) a cache ótica pode ser compartilhada por todos os processadores sem contenção; d) a cache ótica pode reduzir qualquer problema de acesso não uniforme aos níveis mais baixos do sistema de memória; e e) a memória de rede garante exclusão mútua no acesso a dados compartilhados sem necessidade de *hardware* ou *software* adicional. Além disso, outras características das redes óticas, tais como a sua grande largura de faixa e a sua capacidade de disseminação, permitem otimizar os protocolos de coerência e sincronização necessários na maior parte dos sistemas paralelos e distribuídos.

Desta forma, o estudo realizado nesta Tese demonstra que a idéia de utilizar uma rede ótica como meio de armazenamento, além de como meio de comunicação, pode melhorar bastante o desempenho alcançado pelos sistemas de computação paralelos e distribuídos.

Futuramente, pretendemos melhorar a rede de interconexão proposta como cache de terceiro nível para os blocos de memória de multiprocessadores (NetCache), procurando encontrar alternativas que permitam reduzir e/ou eliminar as suas desvantagens, especialmente com relação aos seus custos de implementação. Também pretendemos buscar novas alternativas e sistemas para a utilização das redes óticas como meios de armazenamento de dados, procurando oferecer uma relação custo/desempenho satisfatória sob as mais diversas variações arquiteturais. Uma possível alternativa seria o uso dessas idéias na implementação de servidores de rede escaláveis ou algum outro sistema de computação.

Um outro trabalho bastante interessante para dar continuidade à nossa pesquisa é a exploração de técnicas de tolerância à latência baseadas na memória de rede. Por não precisar consumir largura de faixa adicional da rede ótica, técnicas de tolerância à latência como o *prefetching* de dados podem melhorar significativamente o desempenho de sistemas paralelos.

Finalmente, estamos interessados em continuar a nossa pesquisa com a aplicação da memória de rede no contexto da Internet. Devido à popularidade dessa rede e a possibilidade de implementar a nossa proposta sem a necessidade de *hardware* adicional, a utilização da memória de rede na otimização de aplicações distribuídas sobre a Internet tem um grande interesse para a comunidade científica. Nesta direção, os nossos próximos passos serão a avaliação da memória de rede em um ambiente real de grande porte e a utilização da memória de rede para a sincronização entre processos, além do cacheamento de informações, em aplicações comerciais.

Referências Bibliográficas

- [1] P. E. GREEN, “Optical Networking Update”, *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 5, pp. 764–779, June 1996.
- [2] C. PARTRIDGE, *Gigabit Networking*. Reading, MA: Addison-Wesley Publishers, 2nd ed., April 1995.
- [3] P. W. DOWD, J. CHU, “Photonic Architectures for Distributed Shared Memory Multiprocessors”, In: *Proceedings of the 1st International Conference on Massively Parallel Processing using Optical Interconnections*, Cancun, Mexico, pp. 151–161, IEEE Computer Society Press, April 1994.
- [4] D. B. SARRAZIN, H. F. JORDAN, V. P. HEURING, “Fiber Optic Delay Line Memory”, *Applied Optics*, vol. 29, no. 5, pp. 627–637, February 1990.
- [5] K. L. CALVERT, S. BHATTACHARJEE, E. W. ZEGURA, J. STERBENZ, “Directions in Active Networks”, *IEEE Communications—Special Issue on Programmable Networks*, vol. 36, no. 10, pp. 72–78, October 1998.
- [6] D. L. TENNENHOUSE, J. M. SMITH, W. D. SINCOCKIE, D. J. WETHERALL, G. J. MINDEN, “A Survey of Active Network Research”, *IEEE Communications*, vol. 35, no. 1, pp. 80–86, January 1997.
- [7] E. V. CARRERA, R. BIANCHINI, “NetCache: A Network/Cache Hybrid for Multiprocessors”, In: *Proceedings of the III Workshop on Optics and Computer Science*, San Juan, Puerto Rico, pp. 859–872, Springer-Verlag, April 1999.
- [8] E. V. CARRERA, R. BIANCHINI, “Disk Write Caching with an Optical Network”, In: *Proceedings of the VI International Conference on Parallel Interconnects*, Anchorage, AK, pp. 452–460, IEEE Computer Society Press, October 1999.

- [9] E. V. CARRERA, R. BIANCHINI, “OPTNET: A Cost-Effective Optical Network for Multiprocessors”, In: *Proceedings of the International Conference on Supercomputing*, Melbourne, Australia, pp. 401–408, July 1998.
- [10] J.-H. HA, T. M. PINKSTON, “SPEED DMON: Cache Coherence on an Optical Multichannel Interconnect Architecture”, *Journal of Parallel and Distributed Computing*, vol. 41, no. 1, pp. 78–91, January 1997.
- [11] M. S. GOODMAN, H. KOBRINSKI, M. P. VECCHI, R. M. BULLEY, J. L. GIMLETT, “The LAMBDANET Multiwavelength Network: Architecture, Applications, and Demonstrations”, *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 6, pp. 995–1004, August 1990.
- [12] E. V. CARRERA, R. BIANCHINI, “NWCACHE: Optimizing Disk Accesses via an Optical Network/Write cache Hybrid”, In: *Proceedings of the III Workshop on Optics and Computer Science*, San Juan, Puerto Rico, pp. 845–858, Springer-Verlag, April 1999.
- [13] C. A. BRACKETT, “Dense Wavelength Division Networks: Principles and Applications”, *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 6, pp. 948–964, August 1990.
- [14] B. S. GLANCE, J. M. WIESENFELD, U. KOREN, R. W. WILSON, “New Advances on Optical Components Needed for FDM Optical Networks”, *IEEE Photonics Technical Letters*, vol. 5, no. 10, pp. 1222–1224, October 1993.
- [15] L. G. KASOVSKY, T. K. FONG, T. HOFMEISTER, “Optical Local Area Network Technologies”, *IEEE Communications*, vol. 32, no. 12, pp. 50–54, December 1994.
- [16] B. MUKHERJEE, “WDM-Based Local Lightware Networks – Part I: Single-hop Systems”, *IEEE Network*, vol. 6, no. 3, pp. 12–27, May 1992.
- [17] K. GHOSE, R. K. HORSELL, N. SINGHVI, “Hybrid Multiprocessing in OPTIMUL: A Multiprocessor for Distributed and Shared Memory Multiprocessing with WDM Optical Fiber Interconnections”, In: *Proceedings of the International Conference on Parallel Processing*, pp. 196–199, August 1994.

- [18] E. HALL, J. KRAVITZ, R. RAMASWAMI, M. HALVORSON, S. TENBRINK, R. THOMSEN, “The Rainbow-II Gigabit Optical Network”, *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 5, pp. 814–823, June 1996.
- [19] B. MUKHERJEE, “WDM-Based Local Lightware Networks – Part II: Multihop Systems”, *IEEE Network*, vol. 6, no. 4, pp. 20–32, July 1992.
- [20] D. M. SPIRIT, A. D. ELLIS, P. E. BARNSLEY, “Optical Time Division Multiplexing: Systems and Networks”, *IEEE Communications*, vol. 32, no. 12, pp. 56–62, December 1994.
- [21] A. G. NOWATZYK, P. R. PRUCNAL, “Are Crossbars Really Dead? The Case for Optical Multiprocessor Interconnect Systems”, In: *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pp. 106–115, June 1995.
- [22] J. P. SOKOLOFF, P. R. PRUCNAL, I. GLESK, M. KANE, “A Terahertz Optical Asymmetric Demultiplexer”, *IEEE Photonics Technology Letters*, vol. 5, no. 7, pp. 106–117, July 1993.
- [23] ANSI, “Standard X3T11 (Fibre Channel)”, Technical Report, American National Standards Institute, April 1998.
- [24] D. B. SCHWARTZ, C. K. Y. CHUN, J. GRULA, S. PLANER, G. RASKIN, S. SHOOK, “OPTOBUS I: Performance of a 4 Gb/s Optical Interconnect”, In: *Proceedings of the 3rd International Conference on Massively Parallel Processing using Optical Interconnections*, Maui, Hawaii, pp. 256–263, IEEE Computer Society Press, October 1996.
- [25] ANSI, “Standard X3T12 (FDDI)”, Technical Report, American National Standards Institute, May 1997.
- [26] C. HUANG, P. K. MCKINLEY, “Communication Issues in Parallel Computing Across ATM Networks”, *IEEE Parallel and Distributed Technology*, vol. 2, no. 4, pp. 73–86, October 1994.

- [27] T. V. EICKEN, A. BASU, V. BUCH, “Low-Latency Communication Over ATM Networks Using Active Messages”, *IEEE Micro*, vol. 15, no. 1, pp. 46–53, February 1995.
- [28] Y. HUANG, P. K. MCKINLEY, “Efficient Collective Operations with ATM Network Interface Support”, In: *Proceedings of the International Conference on Parallel Processing*, vol. 1, Bloomington, IL, pp. 34–43, August 1996.
- [29] C. A. THEKKATH, H. M. LEVY, E. D. LAZOWSKA, “Efficient Support for Multicomputing on ATM Networks”, Technical Report 93-04-03, Department of Computer Science and Engineering, University of Washington, Seattle, WA, April 1993.
- [30] T. V. EICKEN, D. E. CULLER, S. C. GOLDSTEIN, K. E. SCHAUSER, “Active Messages: A Mechanism for Integrated Communication and Computation”, In: *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Gold Coast, Australia, pp. 256–266, May 1992.
- [31] T. V. EICKEN, A. BASU, V. BUCH, W. VOGELS, “U-Net: A User-Level Network Interface for Parallel and Distributed Computing”, In: *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, Colorado, pp. 40–53, December 1995.
- [32] FOUNDRY NETWORKS, “High Performance LAN Alternatives”, Technical Report <http://www.foundrynet.com/wpvol1.html>, Foundry Networks, November 1998.
- [33] GIGABIT ETHERNET ALLIANCE, “Gigabit Ethernet – Overview”, Technical Report <http://www.gigabit-ethernet.org/technology/whitepapers/>, Gigabit Ethernet Alliance, May 1999.
- [34] R. LANGENHORST, M. EISELT, W. PIEPER, G. GROßKOPF, R. LUDWIG, L. KÜLLER, E. DIETRICH, H. G. WEBER, “Fiber Loop Optical Buffer”, *Journal of Lightwave Technology*, vol. 14, no. 3, pp. 324–335, March 1996.
- [35] V. P. HEURING, H. F. JORDAN, J. P. PRATT, “Bit-serial Architecture for Optical Computing”, *Applied Optics*, vol. 31, no. 17, pp. 3213–3224, June 1992.

- [36] H. F. JORDAN, V. P. HEURING, R. J. FEUERSTEIN, “Optoelectronic Time-of-Flight Design and the Demonstration of an All-Optical, Stored Program, Digital Computer”, *Proceedings of IEEE – Special Issue on Optical Computing*, vol. 82, no. 11, pp. 1678–1689, November 1994.
- [37] J. P. PRATT, V. P. HEURING, “Delay Synchronization in Time-of-Flight Optical Systems”, *Applied Optics*, vol. 31, no. 14, pp. 2430–2437, May 1992.
- [38] C. E. LOVE, H. F. JORDAN, “SPOC – A Stored Program Optical Computer”, *IEEE Potentials*, vol. 13, no. 4, pp. 11-15, November 1994.
- [39] A. HUANG, N. WHITAKER, H. AVRAMOPOULOS, P. FRENCH, H. HOUH, I. CHUANG, “A System’s Perspective of the Sagnac Fiber Logic Gates and Their Possible Applications”, *Applied Optics*, vol. 33, no. 26, p. 6254, August 1994.
- [40] T. MIKI, “The Potencial of Photonic Networks”, *IEEE Communications*, vol. 32, no. 12, pp. 23–27, December 1994.
- [41] K. GHARACHORLOO, D. LENOSKI, J. LAUDON, P. GIBBONS, A. GUPTA, J. L. HENNESSY, “Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors”, In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*, Seattle, WA, pp. 15–26, May 1990.
- [42] J. E. VEENSTRA, R. J. FOWLER, “MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors”, In: *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Durham, NC, pp. 201–207, January 1994.
- [43] S. C. WOO, M. OHARA, E. TORRIE, J. P. SINGH, A. GUPTA, “The SPLASH-2 Programs: Characterization and Methodological Considerations”, In: *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pp. 24–36, May 1995.
- [44] D. BAILEY *et al.*, “The NAS Parallel Benchmarks”, Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

- [45] D. CULLER, A. DUSSEAU, S. C. GOLDSTEIN, A. KRISHNAMURTHY, S. LUMETTA, T. V. EICKEN, K. YELICK, “Parallel Programming in Split-C”, In: *Proceedings of Supercomputing 93*, pp. 262–273, November 1993.
- [46] B. A. NAYFEH, K. OLUKOTUN, “Exploring the Design Space for a Shared-Cache Multiprocessor”, In: *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, IL, pp. 166–175, April 1994.
- [47] M. FARRENS, G. TYSON, A. R. PLESZKUN, “A Study of Single Chip Processor/Cache Organizations for Large Numbers of Transistors”, In: *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, IL, pp. 338–347, April 1994.
- [48] A. ERLICHSON, B. A. NAYFEH, J. P. SINGH, K. OLUKOTUN, “The Benefits of Clustering in Shared Address Space Multiprocessors: An Applications-Driven Investigation”, In: *Proceedings of the 1st International Conference on Massively Parallel Processing using Optical Interconnections*, Cancun, Mexico, pp. 83–91, IEEE Computer Society Press, April 1994.
- [49] S. MORI, H. SAITO, M. GOSHIMA, M. YANAGIHARA, T. TANAKA, D. FRASER, K. JOE, H. NITTA, S. TOMITA, “A Distributed Shared Memory Multiprocessor: ASURA – Memory and Cache Architectures”, In: *Proceedings of Supercomputing 93*, pp. 740–749, 1993.
- [50] J. K. BENNETT, K. E. FLETCHER, W. E. SPEIGHT, “The Performance Value of Shared Network Caches in Clustered Multiprocessor Workstations”, In: *Proceedings of the 16th International Conference on Distributed Computing Systems*, pp. 113–120, May 1996.
- [51] S. K. REINHARDT, J. R. LARUS, D. A. WOOD, “Tempest and Typhoon: User-Level Shared Memory”, In: *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, IL, pp. 325–337, April 1994.
- [52] T. C. MOWRY, A. K. DEMKE, O. KRIEGER, “Automatic Compiler-Inserted I/O Prefetching for Out-Of-Core Applications”, In: *Proceedings of the 2nd USENIX*

- Symposium on Operating Systems Design and Implementation*, Seattle, WA, pp. 3–17, October 1996.
- [53] D. E. WOMBLE, D. S. GREENBERG, R. E. RIESEN, S. R. WHEAT, “Out of Core, Out of Mind: Practical Parallel I/O”, In: *Proceedings of the Scalable Parallel Libraries Conference*, Mississippi State University, pp. 10–16, October 1993.
- [54] K. MCKUSICK, W. JOY, S. LEFFLER, R. FABRY, “A Fast File System for UNIX”, *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 181–197, August 1984.
- [55] D. KOTZ, C. ELLIS, “Practical Prefetching Techniques for Multiprocessor File Systems”, *Journal of Distributed and Parallel Databases*, vol. 1, no. 1, pp. 33–51, January 1993.
- [56] T. KIMBREL, A. TOMKINS, R. H. PATTERSON, B. BERSHAD, P. CAO, E. FELTEN, G. A. GIBSON, A. R. KARLIN, K. LI, “A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching”, In: *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, Seattle, WA, pp. 19–34, October 1996.
- [57] A. AGARWAL, R. BIANCHINI, D. CHAIKEN, K. JOHNSON, D. KRANZ, J. KUBIATOWICZ, B.-H. LIM, K. MACKENZIE, D. YEUNG, “The MIT Alewife Machine: Architecture and Performance”, In: *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pp. 2–13, June 1995.
- [58] R. ALVERSON, D. CALLAHAN, D. CUMMINGS, B. KOBLENZ, A. PORTERFIELD, B. SMITH, “The Tera Computer System”, In: *Proceedings of the International Conference on Supercomputing*, pp. 1–16, June 1990.
- [59] E. FELTEN, J. ZAHORJAN, “Issues in the Implementation of a Remote Memory Paging System”, Technical Report 91-03-09, Department of Computer Science and Engineering, University of Washington, Seattle, WA, March 1991.
- [60] D. LENOSKI, J. LAUDON, T. JOE, D. NAKAHIRA, L. STEVENS, A. GUPTA, J. HENNESSY, “The DASH Prototype: Logic Overhead and Performance”, *IEEE*

Transactions on Parallel and Distributed Systems, vol. 4, no. 1, pp. 41–61, January 1993.

- [61] D. STODOLSKY, M. HOLLAND, W. COURTRIGHT II, G. GIBSON, “Parity Logging Disk Arrays”, *ACM Transactions on Computer Systems*, vol. 12, no. 3, pp. 206–235, August 1994.
- [62] C. RUEMMLER, J. WILKES, “UNIX Disk Access Patterns”, In: *Proceedings of the USENIX 93 Annual Technical Conference*, San Diego, CA, pp. 405–420, January 1993.
- [63] M. ROSENBLUM, J. K. OUSTERHOUT, “The Design and Implementation of a Log-Structured File System”, *ACM Transactions on Computer Systems*, vol. 10, no. 2, pp. 26–52, February 1992.
- [64] Y. HU, Q. YANG, “DCD – Disk Caching Disk: A New Approach for Boosting I/O Performance”, In: *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Philadelphia, PA, pp. 169–178, May 1996.
- [65] E. V. CARRERA, R. BIANCHINI, “Analytical and Experimental Evaluation of Cluster-Based Network Servers”, Technical Report 718, Department of Computer Science, University of Rochester, Rochester, NY, August 1999.
- [66] UCAID, “Abilene: Project Summary”, Technical Report http://www.ucaid.edu/abilene/html/project_summary.html, University Corporation for Advanced Internet Development, March 1999.
- [67] D. L. TENNENHOUSE, D. J. WETHERALL, “Towards an Active Network Architecture”, *ACM Computer Communication Review*, vol. 26, no. 2, pp. 5–17, April 1996.
- [68] Y. YEMINI, S. D. SILVA, “Towards Programmable Networks”, In: *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L’Aquila, Italy, pp. 87–96, October 1996.

- [69] S. BHATTACHARJEE, K. L. CALVERT, E. W. ZEGURA, “An Architecture for Active Networking”, In: *High Performance Networking 97*, pp. 213–220, April 1997.
- [70] K. L. CALVERT, M. DOAR, E. W. ZEGURA, “Modeling Internet Topology”, *IEEE Communications*, vol. 35, no. 6, pp. 160–163, June 1997.
- [71] P. R. WURMAN, M. P. WELLMAN, W. E. WALSH, “The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents”, In: *Proceedings of the 2nd International Conference on Autonomous Agents*, Minneapolis, MN, pp. 301–308, May 1998.
- [72] J. H. SALTZER, D. P. REED, D. D. CLARK, “End-to-End Arguments in System Design”, *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, November 1984.
- [73] D. J. WETHERALL, U. LEGEDZA, J. GUTTAG, “Introducing New Internet Services: Why and How”, *IEEE Network – Special Issue on Active and Controllable Networks*, vol. 12, no. 3, pp. 12–19, July 1998.
- [74] L.-W. LEHMAN, S. J. GARLAND, D. L. TENNENHOUSE, “Active Reliable Multicast”, In: *Proceedings of IEEE INFOCOM 98*, San Francisco, CA, pp. 813–820, April 1998.
- [75] U. LEGEDZA, D. J. WETHERALL, J. GUTTAG, “Improving The Performance of Distributed Applications Using Active Networks”, In: *Proceedings of IEEE INFOCOM 98*, San Francisco, CA, pp. 828–936, April 1998.
- [76] B. SCHWARTZ, A. W. JACKSON, W. T. STRAYER, W. ZHOU, D. ROCKWELL, C. PARTRIDGE, “Smart Packets for Active Networks”, In: *OpenArch 99*, pp. 786–895, March 1999.
- [77] S. BHATTACHARJEE, K. L. CALVERT, E. W. ZEGURA, “Self-organizing Wide-area Network Caches”, In: *Proceedings of IEEE INFOCOM 98*, vol. 2, San Francisco, CA, pp. 600–608, April 1998.

- [78] J. HARTMAN, U. MANBER, L. PETERSON, T. PROEBSTING, “Liquid Software: A New Paradigm for Networked Systems”, Technical Report 96-11, Department of Computer Science, University of Arizona, Tucson, AZ, June 1996.
- [79] P. CAO, S. IRANI, “Cost-Aware WWW Proxy Caching Algorithms”, In: *Proceedings of USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, pp. 193–206, December 1997.
- [80] L. FAN, P. CAO, J. ALMEIDA, A. BRODER, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol”, In: *Proceedings of ACM SIGCOMM 98*, pp. 254–265, 1998.
- [81] V. HOLMEDAHL, B. SMITH, T. YANG, “Cooperative Caching of Dynamic Content on a Distributed Web Server”, In: *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pp. 243–250, July 1998.
- [82] E. P. MARKATOS, “Main Memory Caching of Web Documents”, In: *Proceedings of 5th International World Wide Web Conference*, Paris, France, pp. 893–906, May 1996.
- [83] A. CHANKHUNTHOD, P. DANZIG, C. NEERDAELS, M. F. SCHWARTZ, K. J. WORRELL, “A Hierarchical Internet Object Cache”, In: *Proceedings of the USENIX 96 Annual Technical Conference*, San Diego, CA, pp. 153–163, January 1996.
- [84] M. ABRAMS, C. R. STANDRIDGE, G. ABDULLA, S. WILLIAMS, E. A. FOX, “Caching Proxies: Limitations and Potentials”, In: *Proceedings of 4th International World Wide Web Conference*, Boston, MA, pp. 537–545, December 1995.