Análise de Performance de *Crossbar Switch* com a Utilização de HDL

Nelson A. Gonçalves Junior[1], Renata C. Lopes da Silva[2], Ronaldo A. L. Gonçalves[3], João Angelo Martini[4]
Departamento de Informática
Universidade Estadual de Maringá – Maringá, PR - Brasil
{njunior[1], renatals[2], ronaldo[3], jangelo[4]}@din.uem.br

Resumo

Uma forma de aumentar o desempenho de sistemas computacionais é a utilização de diversas unidades de processamento interconectadas. Os sistemas multiprocessadores, como são denominados, têm sua performance influenciada por diversos fatores, entre eles a rede de interconexão, responsável pela comunicação entre as unidades de processamento. Tal rede pode possuir diversas topologias, sendo uma das mais utilizadas a rede multiestágio, que possui diversas fases de chaves roteadoras, responsáveis pelo direcionamento das mensagens. Tais chaves, as Crossbar Switches, também são responsáveis pelo armazenamento temporário de pacotes que não podem ser imediatamente enviados. Neste contexto, o presente trabalho se propõe a investigar o elemento básico de roteamento de redes de interconexão, a crossbar switch, investigando-o e submetendo-o a diferentes análises utilizando a Linguagem de Descrição de Hardware VHDL.

1. Introdução

Ao se falar de sistemas computacionais, sempre se evidencia o desempenho destes em relação à velocidade de execução dos serviços requisitados. Buscando uma constante melhora no desempenho dos sistemas, diversas pesquisas são realizadas no meio acadêmico e na indústria. Uma direção que essas investigações tomaram nos últimos anos foi a exploração de sistemas multiprocessadores, que consistem em modelos computacionais que exploram o paralelismo para possibilitar a execução de diversas instruções simultâneas.

Tais sistemas são dotados de diversas unidades de processamento, vários processadores e módulos de memória interligados de forma a trocarem dados entre si e chegar aos resultados desejados de maneira mais rápida e eficiente. Para que essas unidades de processamento se comuniquem é necessária uma rede de interconexão.

Existem diversas topologias para as redes de interconexão, partindo de redes estáticas (rede completamente conectada, redes de *array* linear, redes cubo, entre outras), que mantêm uma conexão ponto a ponto fixa entre os nós da rede, até redes dinâmicas (múltiplos barramentos, redes *crossbar* e redes multiestágio) onde o caminho da comunicação não é fixo. Uma das topologias mais empregadas é a rede dinâmica multiestágio [1], que emprega pequenas chaves roteadoras utilizadas para facilitar as comunicações entre as unidades de processamento.

Denominadas *crossbar switches* ou *switching elements* (SE), essas chaves são estruturas responsáveis por rotear as mensagens que estão sendo enviadas de uma unidade de processamento para outra. Elas reduzem o custo da rede de interconexão e melhoram o desempenho do sistema quando comparadas com outras que apresentam um maior atraso na propagação de mensagens [2] [3].

São justamente essas chaves que o presente trabalho busca investigar. E para realizar essa investigação foi utilizada uma linguagem de descrição de hardware. As Linguagens de Descrição de Hardware (HDLs – *Hardware Description Languages*) surgiram inicialmente para documentação de circuitos integrados, mas logo foram aperfeiçoadas e ganharam características que lhes permitiram projetar e simular circuitos lógicos.

São linguagens de alto nível que permitem a construção, simulação e análise de circuitos lógicos digitais, facilitando o projeto e a realização de testes. Diferente das linguagens de programação convencionais, as HDLs possuem estruturas para a representação de certos comportamentos de circuitos e sistemas eletrônicos como tempo e concorrência,

fundamentais para a descrição da representação do comportamento de componentes de hardware.

A linguagem escolhida para a realização deste trabalho foi a VHDL (***V****ery High-Speed Integrated Circuits* ***H****ardware* ***D****escription* ***L****anguage* - Linguagem de Descrição de Hardware de Circuitos Integrados de Altíssima Velocidade) [4], uma linguagem bem documentada e padronizada, que facilita a comunicação entre os diversos módulos que podem existir em um projeto, garantindo a reusabilidade e portabilidade dos códigos desenvolvidos.

Dentro deste contexto, o presente trabalho investiga esse componente essencial das redes de interconexão, a *crossbar switch*, sob a perspectiva de uma linguagem de descrição de hardware, simulando a chave sob condições reais e trazendo análises comportamentais das diversas configurações que a chave pode possuir.

A organização dessas chaves pode acontecer de diferentes formas, as quais foram implementadas buscando uma comparação de desempenho de cada um desses modelos. Os testes foram realizados com as implementações de chaves 2x2 (duas portas de entrada e duas de saída) e chaves 4x4. Além dos testes isolados de cada um dos tipos de chaves, as chaves 2x2 foram conectadas de forma a simular uma rede de interconexão 8x8 de três estágios, buscando a análise do comportamento das chaves quando interligam várias unidades de processamento.

As simulações foram realizadas considerando a tecnologia FPGA (*Field Programmable Gate Array*) [5], de forma a gerar códigos sintetizáveis para tal dispositivo caso os projetos venham a ser implementados em hardware no futuro.

2. Trabalhos Relacionados

A utilização de linguagens de descrição de hardware vem aumentando com o passar dos anos. Isso ocorre porque cada vez mais a indústria está deixando de lado o modelo gráfico de projeto de hardware e adotando projetos baseados em HDL [6]. Seguindo a indústria, o meio acadêmico também começa utilizar essas linguagens, utilizando-as no auxílio às disciplinas correlatas a área de Arquitetura de Computadores e na validação de modelos sugeridos ou protótipos de circuitos lógicos e integrados.

A utilização de HDLs como ferramenta de apoio ao estudo de Arquiteturas de Computadores é proposta em [7] e [8]. São propostos modelos arquiteturais e exemplos práticos em HDLs para que os alunos compreendam de forma mais simples seus respectivos funcionamentos e preparem-se melhor para o mercado de trabalho.

Em relação às redes de interconexão, as HDLs também têm se mostrado eficientes pois permitem a simulação das redes ou de componentes delas, viabilizando estudos sobre seu desempenho e propostas de melhorias para aumentar a performance.

Em [9] foi realizada uma análise de uma rede Banyan [10] composta por *switches* 2x2 de forma a comparar o comportamento da rede durante a realização de *broadcasts*.

Outra análise de rede foi realizada em [11], em que se utilizou VHDL para projetar e analisar uma rede Torus para o computador BlueGene/L, da IBM. A utilização da linguagem de descrição de hardware ajudou na validação da rede, uma vez que graças às simulações pôde-se propor alterações que melhoraram o desempenho do sistema.

Redes de interconexão optoeletrônicas também foram propostas em [14] de forma a descrever a modelagem de sinais ópticos em VHDL e permitir a integração de simulações eletrônicas/optoeletrônicas.

A proposta de chaves *crossbar* para melhorar o desempenho das redes também tem sido constante. Alguns exemplos são a utilização de uma chave reconfigurável para a implementação dinâmica de topologias independente da tecnologia utilizada [15], além de alguns modelos de *crossbar switches* propostos em VHDL, como o uso de uma memória a ser utilizada como *buffer* compartilhado pelas portas de entrada e saída [12] e o projeto de um *crossbar switch* 32x32 com *buffers* capazes de armazenar pacotes de tamanhos variáveis [13].

3. Implementações

Ao todo foram implementados cinco modelos de chaves *crossbar* com a utilização da linguagem de descrição de hardware VHDL. Para o desenvolvimento do trabalho foi utilizado o simulador VHDL Simili [16], versão 3.0 *free*, ideal para fins acadêmicos e projetos de pequena e média escala. O simulador possui interface gráfica, editor de código e visualizador de gráficos de onda, o que permitiu a execução dos testes e análise dos resultados de forma objetiva.

Para simular a comunicação entre as unidades de processamento foram definidos pacotes de 128 bytes, tamanho médio de um pacote em uma rede de interconexão [17]. Tais pacotes são divididos em três campos: *Routing Information* (RI), *Sequence Number* (SN) e *Data*.

O campo RI trata das informações necessárias para o roteamento do pacote. As simulações foram

realizadas tratando as chaves isoladas e também simulando uma rede de três estágios para as chaves 2x2. Dessa forma, para o roteamento de um pacote através da rede que utiliza estes *switches* são necessários três bits de RI. A chave irá direcionar o pacote recebido de acordo com esse bit. Se o bit for '1', o pacote será direcionado para a saída superior, enquanto que se o bit for '0', o pacote será enviado à saída inferior da chave. Cada bit representa um estágio da rede, sendo que o bit mais a direita é lido no primeiro estágio, o bit do meio no segundo e o bit mais a esquerda no último estágio da rede. Já para os *switches* 4x4 a direção que um pacote irá tomar ao entrar em uma chave é indicada por pares de bits, sendo necessários então, seis bits para a simulação do RI nessas chaves, considerando uma rede de três estágios.

O SN trata do número de seqüência utilizado para remontar a mensagem quando todos os pacotes que a compõe chegam ao destino. Esse campo é necessário uma vez que os pacotes podem ser entregues fora de ordem, para aumentar o desempenho da rede de interconexão. Foram reservados seis bits para esse campo, de forma a dividir uma mensagem em até 63 pacotes. O restante dos bits é ocupado por dados.

As subseções seguintes descrevem cada um dos modelos de chaves implementados e analisados.

3.1. *Unbuffered Crossbar Switch*

A chave *unbuffered* se caracteriza por não possuir *buffers* em suas entradas e saídas. Serve apenas como roteador dos pacotes através da rede de interconexão. Além das portas de entrada e saída, a chave também recebe um sinal *clk*, que representa o *clock* interno dessa *crossbar*. Esse *clock* é de 50 MHz, freqüência utilizada como forma de comparar o desempenho entre as diferentes chaves [5].

O *datapath* interno desta chave é composto por quatro ou dois multiplexadores, dependendo da quantidade de portas de entrada e saída. Os multiplexadores associados a cada saída foram colocados em processos diferentes na implementação, garantindo que eles serão executados de forma concorrente, assim como em uma *crossbar switch* real.

Durante a chegada de novos pacotes nas entradas da *crossbar*, pode ocorrer de dois desses pacotes terem o mesmo endereço de destino. Isso é tratado de duas formas diferentes neste modelo de chave. Na primeira, as portas recebem prioridades fixas, sendo que a porta de entrada 0 possui prioridade maior, seguida pela 1, e assim por diante. Já no segundo caso, as prioridades são alternadas a cada ciclo de *clock*, fazendo com que a cada ciclo, uma das portas possua maior prioridade. Neste caso, por não possuir *buffers* de armazenamento nas *crossbar*, um dos pacotes terá de ser descartado.

3.2. Buffered *Crossbar* Switch FIFO

Uma das formas de evitar, ou pelo menos diminuir, o desperdício de pacotes é através da utilização de *buffers* nas entradas da *crossbar switch*. Dessa forma, se um pacote não pode ser imediatamente redirecionado à sua respectiva saída, pois um outro pacote com prioridade maior necessita ir para esta mesma saída, ele pode ficar aguardando em um buffer, no próprio *switch*, até que em um ciclo de *clock* posterior ele possa ser enviado. A forma mais simples de implementar esses *buffers* é através da organização denominada FIFO (*first-in first-out*), em que somente a primeira posição do *buffer* será analisada. Caso o pacote nessa posição esteja bloqueado, nenhum outro pacote pode ser enviado, mas novos pacotes podem chegar à chave, sendo armazenados no *buffer*.

O comprimento dos *buffers* foram variados de forma a analisar o desempenho em relação ao custo deles, variando de 3 até 12 posições para cada buffer.

3.3. *Buffered Crossbar Switch* Não-Bloqueante

Outro modelo de c*rossbar* s*witch* que utiliza *buffers* tem como característica a análise não só da primeira posição do *buffer*, mas de qualquer pacote que possa ser redirecionado para uma das saídas disponíveis. Dessa forma, se o pacote na primeira posição estiver bloqueado, a *crossbar* analisa a possibilidade de enviar outro pacote não-bloqueado para uma das saídas disponíveis. A figura 1 apresenta a organização desse buffer em uma chave 2x2, onde o pacote A deseja ir para a saída superior da chave, ocupada pelo pacote C e o pacote B deseja ir para a saída inferior, que está disponível.

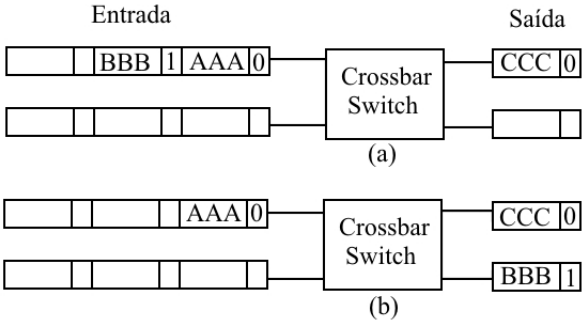

Figura 1. Simulação de Configuração dos *Buffers* Não-Bloqueantes

3.4. *Output Queued Crossbar Switch*

Outra forma de utilizar *buffers* para o armazenamento de pacotes é a colocação de filas nas portas de saída. Dessa forma, um decodificador é utilizado para logo que o pacote chegar à *crossbar switch*, ser imediatamente alocado no buffer correspondente a saída indicada por seu endereçamento.

Com a *Output Queued Crossbar Switch* [18] não é necessário analisar se a saída que o pacote irá utilizar está bloqueada ou não, pois ele já está associado a sua saída correspondente. Assim, os *buffers* podem ser uma simples FIFO, que analisa somente a primeira posição, não sendo necessário hardware complementar para analisar as outras posições do buffer. Isso também garante que pacotes mais antigos não tenham um atraso muito grande durante o congestionamento de uma das saídas. A organização de tal chave com duas entradas e duas saídas é apresentada na figura 2.

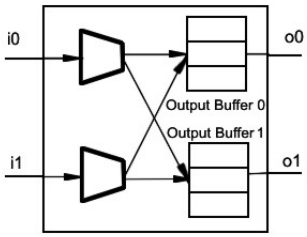

Figura 2. Organização da *Output Queued Crossbar Switch*

3.5. SRAM *Crossbar Switch*

Outra técnica de implementação da rede interna da *crossbar switch* é através da utilização de uma memória estática SRAM (*Static Random Access Memory*) [12]. Assim é possível que as entradas escrevam na memória e as saídas leiam os dados dela, baseadas no endereço de destino da mensagem.

Foram utilizadas memórias de 2 Kbytes para uma chave 4x4 e de 2 Kbytes e 1 Kbytes para a *crossbar* 2x2, buscando a análise do comportamento com tamanhos de memória diferentes. Com a utilização de uma memória para realizar o roteamento e armazenamento de pacotes, são necessários hardwares complementares, já que deve ocorrer escrita na memória, através das portas de entrada, e leitura, através das portas de saída.

Quando um pacote chega a uma porta de entrada ele é gravado na memória na primeira posição vaga que encontrar. A leitura ocorre através de decodificadores que ligam cada posição da memória com as portas de saída. Assim, sempre que houver pelo menos um pacote endereçado para uma determinada saída ele será enviado, independente de sua posição na memória. O endereçamento ocorre da mesma forma que nas chaves apresentadas nas seções anteriores, sendo realizado pelo campo RI no pacote.

A decodificação também possui suas prioridades. Assim, entre dois pacotes endereçados para a mesma saída, o que estiver na posição menor da memória será enviado. Isto ocorre porque o buffer é atualizado sempre que os pacotes deixam-no, sendo que os pacotes mais velhos são sempre colocados no topo da memória, possuindo maior prioridade sobre os pacotes mais novos.

3.6. Rede *Shuffle-Exchange*

Para a análise das diferentes chaves foi também implementada e analisada uma rede 8x8 com três estágios. A topologia adotada foi a *Shuffle-Exchange* [19], que segue o modelo de interconexão apresentado na figura 3.

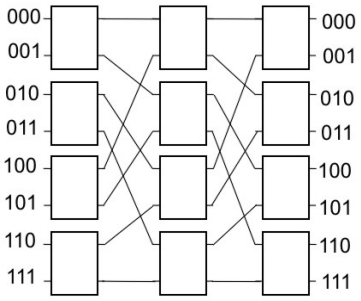

Figura 3. Rede *Shuffle-Exchange* 8x8

É possível notar através da figura 4 que as conexões são realizadas através da operação *Shuffle*, em que os bits são deslocados sempre à esquerda, conforme a seguinte fórmula:

$$S\ (P_i\ P_{i-1} \ldots P_1) = P_{i-1}\ P_{i-2} \ldots P_i$$

Dessa forma, a saída de 001 é interligada à entrada de 010 do próximo estágio, a saída de 010 a entrada de 100 e assim por diante.

Para análise da rede, foram alternados os tipos de *crossbar* para comparar o desempenho da rede com cada um dos modelos implementados.

Outra análise foi realizada em relação à estratégia de *switching* da rede para chaves *unbuffered*. Foi analisado o *circuit swithing* [20], onde o caminho de uma mensagem saindo de uma fonte até seu destino é estabelecido antes da comunicação ser iniciada. O caminho inicialmente é estabelecido e só então a

comunicação acontece. Tal técnica foi implementada somente nos *switches* do tipo *unbuffered* uma vez que não é necessário o armazenamento temporário dos pacotes em nós intermediários da rede. Uma vez que uma mensagem saia do nó fonte, o *circuit switching* garante que ela será entregue a seu destino sem qualquer interrupção.

A outra técnica de *switching* analisada foi o *virtual cut-through* [21], onde a mensagem é dividida em pacotes, que vão sendo enviados através da rede. Caso dois pacotes cheguem em um nó e desejem utilizar a mesma saída, um deles será imediatamente enviado, enquanto outro é armazenado em um buffer. Se não existir *buffers* nas chaves *crossbar* o pacote deve ser descartado e reenviado a partir de sua fonte.

4. Testes e Resultados

Buscando a comparação do tempo de envio de diversos pacotes e da perda causada pelo reenvio deles, foram introduzidos cinco *testbenchs* diferentes para cada conjunto de testes. Os *testbenchs* são códigos em VHDL utilizados para associar dados às entradas do programa.

Essa associação de dados foi realizada de forma pseudo-randômica, variando o fluxo de pacotes de acordo com o *testbench* gerado. Em todos os casos, a freqüência de recebimento de pacotes nas entradas é grande, de forma a observar o comportamento das chaves durante o maior fluxo de pacotes possível.

Para a associação dos pacotes ou mensagens às entradas utilizou-se circuitos LFSRs (*linear feedback shift register*) [22] cuja entrada é uma função de seu próprio estado anterior, gerando pacotes diferentes a cada ciclo de clock. Os pacotes criados dão a impressão de serem aleatórios, porém os mesmos pacotes são gerados a cada execução. Dessa forma é possível comparar os diferentes tipos de *crossbar switch* em relação às mesmas entradas de dados. Foram utilizadas cinco diferentes variações dos circuitos LFSRs, de forma a gerar cinco conjuntos de dados diferentes. Os resultados apresentados são baseados na média desses cinco conjuntos de dados diferentes.

4.1. *Crossbar Switches* 2x2

Inicialmente foram testadas as chaves 2x2 isolamente. Para cada teste, foram associados 1.198 pacotes, o que equivale a um fluxo de 149 Kbytes passando pelas *crossbars*. A cada ciclo de *clock*, as entradas recebem dois novos pacotes. As análises foram realizadas comparando a quantidade de pacotes perdidos por não poderem ser roteados imediatamente e não existir espaço suficiente para que fiquem armazenados na chave, sendo reenviados pela fonte e o tempo total para o envio de todos os pacotes.

As chaves *unbuffered* mostraram os piores rendimentos, despachando os pacotes em 16,8 us e 16,22 us para chaves com prioridade fixa e prioridade alternada, respectivamente. Com a prioridade fixa, a porta de entrada 0 sempre possui preferência no momento de enviar os pacotes. Assim, se existir conflito entre os pacotes nas duas portas, o pacote que estiver na porta 0 é enviado. Já com a prioridade alternada, essa preferência vai sendo trocada a cada ciclo de *clock*. Assim, em um primeiro ciclo, a porta 0 possui prioridade, no segundo a porta 1, e assim por diante.

Em relação às chaves com *buffers* FIFO foram testadas oito configurações diferentes, variando o tamanho dos *buffers* e as prioridades. Inicialmente a chave foi configurada com *buffers* capazes de armazenar 3 pacotes e então foi sendo aumentada a capacidade de armazenamento em cada chave. Os melhores resultados foram obtidos com a utilização de *buffers* com capacidade de armazenar 12 pacotes e prioridade variável, que enviou todos os pacotes em 14,79 us. É importante ressaltar que chaves com capacidade de armazenamento de 5 pacotes não tiveram um resultado tão inferior, enviando todos os pacotes em 14,99 us.

Pôde-se observar com as simulações das chaves FIFO que muitos pacotes prontos para serem enviados sofrem atrasos desnecessários por não estarem na primeira posição do *buffer*. Uma forma de minimizar tal problema é a utilização de *buffers* não-bloqueantes, que despacha o primeiro pacote pronto para o envio, independente da posição que ele ocupe. Com tais *buffers* houve um pequeno ganho de desempenho, com os dados sendo enviados em 14,57 us. Tais resultados correspondem a *buffers* com capacidade de armazenamento de 5 pacotes. O aumento na capacidade dos *buffers* também foi testado, porém não produziu diferenças satisfatórias nos resultados.

Um ponto que deve ser analisado para se decidir entre *buffers* FIFO ou não-bloqueantes no projeto de uma rede de interconexão é o custo destes últimos que é maior, além de serem mais complexos de se implementar, uma vez que qualquer posição do *buffer* pode ser acessada para o envio de algum pacote.

Outra alternativa é a utilização de *buffers* nas portas de saída. Na denominada *Output Queued Crossbar Switch* os pacotes que chegam são diretamente colocados em *buffers* nas portas de saídas. Se um pacote está em determinado *buffer* significa que ele já pode ser enviado através da porta de saída correspondente. Dessa maneira não é necessário

nenhum *hardware* complementar para manipular o *buffer*, que pode ser uma simples fila FIFO. Além de simples tal técnica mostrou-se ser eficiente, enviando os pacotes em 12,81 us. Com a *OQ crossbar* foram perdidos em média apenas 79 dos 1.198 pacotes que passavam pela chave, ou seja, apenas 6,59% do total de pacotes tiveram que ser reenviados pela fonte.

A última chave avaliada foi a *crossbar* que conta com uma memória SRAM em sua rede interna. O desempenho de tal chave também foi muito bom em relação às outras 2x2, ficando pouco abaixo da *Output Queued Crossbar Switch*. Foram testadas memórias de 1 e 2 Kbytes, que são capazes de armazenar 8 e 16 pacotes.

A chave com memória de 1 KB enviou os pacotes em 14,34 us enquanto a chave com memória de 2 KB obteve um tempo médio de 13,27 us, mostrando que o aumento no tamanho da memória teve influência no desempenho. A tabela 1 apresenta os resultados do tempo de execução em microsegundos (us) e percentagem de pacotes perdidos nos testes realizados com os *switches* 2x2.

Tabela 1. Resultados das Simulações de Chaves 2x2

Chave	Tempo Total (us)	% Perda
Unbuffered	16,22	34,22
FIFO (5 posições)	14,79	25,79
FIFO (12 posições)	14,59	22,61
Não-Bloqueante (5 posições)	14,57	21,45
Output Queued	12,81	6,59
SRAM 1KB	14,34	17,52
SRAM 2KB	13,27	7,92

Como pode ser observado na tabela 1, os melhores resultados foram obtidos com a *Output Queued Crossbar Switch* com *buffers* de cinco posições para chaves 2x2. Algumas chaves mostraram-se pouco eficientes, tendo perdido mais de 20% dos pacotes.

4.2. Rede *Shuffle-Exchange* 8x8

Outro teste procurou comparar o desempenho de uma rede *Shuffle-Exchange* 8x8 de forma a avaliar a importância das *crossbar switches* em uma rede de interconexão. A rede foi implementada em VHDL e então simulada com cada uma das chaves 2x2 desenvolvidas. Os testes foram realizados com a simulação do tráfego de cerca de 1,2 MB pela rede.

Uma técnica que foi possível simular através dessa rede, foi a de *circuit switching*, onde todo o caminho da mensagem é inicialmente estabelecido, e então a mensagem pode ser enviada. Essa técnica foi simulada apenas com *switches unbuffered*, uma vez que não é necessário o armazenamento de pacotes em estágios intermediários da rede. O tempo total para o envio dos dados foi de 34,24 us.

O desempenho com esse tipo de chave foi melhor do que utilizando o método *virtual cut-through*, que divide as mensagens em diversos pacotes enviando a mensagem aos poucos. As mensagens foram divididas em 9.584 pacotes e tal método, com *switches unbuffereds*, obteve um tempo total de 41,66 us. O tempo de execução da rede com a utilização de outras chaves é apresentado na tabela 2.

Tabela 2. Resultados da Simulação da Rede *Shuffle-Exchange*

Chave	Tempo (us)
Unbuffered Virtual Cut-Through	41,66
Unbuffered Circuit Switching	34,24
FIFO (5 posições)	35,24
FIFO (12 posições)	30,48
Não-Bloqueante (5 posições)	29,97
Output Queued	27,17
SRAM 1KB	27,58
SRAM 2KB	26,98

É importante notar que as simulações com chaves FIFO de 5 e 12 posições isoladas tiveram apenas 0,2 us de diferença. Já quando colocadas em uma rede, essa diferença acaba aumentando para quase 5 us. Vale ressaltar também que os dados aqui simulados correspondem a apenas 1,2 MB e que as diferenças se acentuariam com um conjunto de dados maiores.

Como pode ser observado na tabela 2, o melhor desempenho foi obtido pela rede utilizando memória compartilhada entre as portas de entrada e saída. Porém, a diferença foi pequena em relação à *Output Queued Crossbar Switch*. Dessa forma, o que deve ser analisado é a relação custo/beneficio que a chave traria, uma vez que o custo de uma memória SRAM é maior que o da utilização de buffers. Além do espaço para a alocação dos pacotes, a utilização de uma memória como *datapath* interno requer o uso de multiplexadores nas entradas da chave *crossbar* para a escrita e de decodificadores para a leitura de dados da memória e o envio em sua respectiva porta.

4.3. *Crossbar Switches* 4x4

Buscando uma análise mais profunda dos modelos de *crossbar switch* foram implementadas chaves 4x4, com quatro portas de entrada e quatro portas de saída, de forma a analisar se os resultados seriam os mesmos

para chaves com um número maior de portas, que conseqüentemente receberiam mais dados ao mesmo tempo. Os testes de cada modelo de chave foram realizados isoladamente.

Nestes testes, a cada ciclo de *clock* são lidos 512 bytes pela *crossbar switch*, o que equivale à chegada de quatro pacotes simultâneos. Ao todo, são cerca de 300 Kbytes que devem ser roteados pelas chaves. Os resultados obtidos podem ser observados na tabela 3.

Tabela 3. Resultados das Simulações de Chaves 4x4

Chave	Tempo Total (us)
Unbuffered	26,97
FIFO (5 posições)	19,97
FIFO (12 posições)	18,82
Não-Bloqueante (5 posições)	15,32
Output Queued (3 posições)	13,62
Output Queued (5 posições)	12,95
SRAM 2KB	12,82

Conforme mostra a tabela 3 é possível notar que as chaves *unbuffered* e FIFO tiveram resultados muito aquém do esperado. Em relação às chaves FIFO, o grande volume de dados que entra na chave faz com que as posições dos *buffers* sejam ocupadas rapidamente. E como os pacotes só são enviados se não estiverem bloqueados e estiverem na primeira posição do buffer, muitos acabam sendo perdidos.

Novamente os melhores resultados acabaram sendo obtidos pela chave SRAM, com desempenho pouco superior à chave *Output Queued* com *buffers* de cinco posições.

Os testes realizados mostraram o impacto de diferentes modelos de *crossbar switch* sobre o comportamento de uma rede de interconexão. Por não possuírem unidade de armazenamento, as *Unbuffered Crossbar Switch* têm seu desempenho melhorado apenas quando utiliza o *circuit switching* como estratégia de *switching*. Isso porque elas apenas roteiam os pacotes que chegam sem possuir unidades de armazenamento para o caso de conflito.

A *crossbar switch* com *buffers* FIFO também não apresentou resultados satisfatórios. Isso porque, mesmo com as unidades de armazenamento, apenas o pacote que está na primeira posição do buffer pode ser enviado. Com muitos conflitos de endereços ocorrendo, os *buffers* ficam cheios rapidamente, e muitos pacotes são perdidos.

Esse atraso causado por um pacote bloqueado na primeira posição é amenizado com a utilização de *buffers* não-bloqueantes, que analisam todas as posições dos *buffers* para o envio de um pacote à saída. Porém, isso requer *hardware* mais complexo, uma vez que cada posição do *buffer* deve estar associada às saídas existentes na chave.

Um método mais eficiente e menos complexo é a colocação de *buffers* diretamente nas saídas da chave, como na *Output Queued Crossbar Switch*. Dessa forma, cada novo dado que chega é alocado diretamente no seu buffer de saída, que pode ser uma simples FIFO. Se houver pacotes no *buffer*, ele é imediatamente enviado por sua respectiva saída, causando pouco desperdício de pacotes.

Uma maneira mais complexa de implementar, mas que gerencia melhor a alocação de espaço para o armazenamento de pacotes bloqueados é a utilização de um buffer compartilhado entre todas entradas e saídas, utilizando uma memória RAM estática. Dessa forma um pacote em qualquer posição da memória pode ser enviado, desde que não esteja bloqueado, o que traz ganho de desempenho para a rede.

5. Conclusão

Responsáveis por direcionar as mensagens de dados pela rede, as *crossbar switches* são elementos fundamentais nas redes de multiprocessadores. Nesse contexto, o presente estudo abordou as *crossbar switches* buscando analisá-las sob a perspectiva de uma linguagem de descrição de hardware. Isto permitiu a simulação e análise dos diferentes tipos de *crossbar switches*, comparando-as em condições reais de execução.

Para um fluxo contínuo de dados algumas chaves mostraram-se menos eficientes, tendo alta taxa de perda de pacotes e tempo de execução, como foi o caso da *Unbuffered Crossbar Switch* e da *Crossbar Switch* com *buffers* FIFO. Mantendo uma média melhor, a *Crossbar Switch* com *buffers* não-bloqueantes teve resultados razoavelmente bons. Porém, as menores perdas foram percebidas com a *Output Queued Crossbar Switch* e com a chave SRAM, que mantiveram um excelente índice de envio de pacotes por ciclo de *clock* em chaves 2x2 e 4x4.

Os estudos realizados mostraram que, apesar de ser um componente básico, a *crossbar switch* tem muita influência sob o desempenho de uma rede de interconexão. Os atrasos causados por modelos de chave pouco eficientes acabam afetando o tempo de processamento com pacotes perdidos que precisam ser reenviados. Isso tem um peso maior em uma rede grande, onde o reenvio de pacotes diretamente da fonte ocasiona atrasos elevados, ou quando há uma grande taxa de dados passando pelas chaves.

O simulador permitiu a execução e o teste do projeto sobre diversas circunstâncias, tornando possível a análise da *crossbar switch* sobre diferentes condições de teste. A utilização da linguagem de descrição de hardware VHDL foi fundamental para uma análise mais precisa das chaves, graças ao seu alto nível de reusabilidade, facilidade de testes e, principalmente, a capacidade de simulação de tempo, que trouxe maior precisão às validações dos resultados.

Na continuidade desta pesquisa, pretende-se investigar outros elementos que influenciam o desempenho de uma rede de interconexão, além de análise de outras topologias e redes intra-chip.

6. Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ) e à Fundação Araucária pelo apoio.

7. Referências

[1] C. Wu, M. J. Lee, "Performance Analysis of Multistage Interconnection Network Configuration and Operation", IEEE Transaction On Computers, Vol. 41, 1992, pp.18-27.

[2] H. El-Rewini, M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing*, John Wiley and Sons, 2ª Edição, 2005.

[3] J. Kitowski, *Review of Parallel Computer Architectures*. Institute of Computer Science. Polônia, 1997. Disponível em <http://www.icsr.agh.edu.pl/publications/html/ppam97prof/> Acesso em abril de 2006.

[4] J. P. Uyemura, *Sistemas Digitais: Uma Abordagem Integrada*, Pioneira Thomson Learning, 2002.

[5] V. Singhal, R. Lê, *High-Speed Buffered Crossbar Switch Design Using Virtex-EM devices*, Xilinx, 2000. 7 pp.

[6] A. A. Sagahyroon, "From AHPL to VHDL: A Course in Hardware Description Languages", IEEE Transactions on Education, Volume 43, Edição 4, 2000, pp. 449-454.

[7] V. A. Pedroni, "Teaching Design-Oriented VHDL", IEEE International Conference on Microeletronic Systems Education, Edição 1, 2003. pp. 6-7.

[8] J. O. Hanblem, "A VHDL Synthesis Model of The MIPS Processor For Use In Computer Architecture Laboratories", IEEE Transactions On Education, Volume 40, Edição 4, 1997. 10 pp.

[9] D. Tutsh, G. Hommel, "Performance of Buffered Multistage Interconnection Networks in Case of Packet Multicasting", Proceedings of Advances in Parallel and Distributed Computing, 1997. pp. 50-57.

[10] C.P. Kruskal, M. Snir, "The Performance of Multistage networks for Multiprocessors", IEEE Transactions on Computer, Volume C-32, Edição 12, 1983. pp. 1091-1098.

[11] D. Blumrich, et al, "Design and Analysis of the BlueGene/L Torus Interconnection Network", IBM Research Report, 2003.

[12] G. Kornaros, "A Buffered Crossbar Switch Fabric Utilizing Shared Memory", Proceedings of the 9th EUROMICRO Conference on Digital System Design, 2006. pp.180-188.

[13] D. G. Simos, "Design of a 32x32 Variable-Packet-Size Buffered Crossbar Switch Chip", Tese de Mestrado. Departamento de Ciência da Computação, Universidade de Creta, Grécia, 2004. 102 pp.

[14] S. Koh, "VHDL Modeling of Optoeletronic Interconnect Networks", Analog Integrated Circuits and Signal Processing, Volume 16, Edição 2, 1998. pp. 111-119.

[15] H. C. Freitas, C. A. Martins, "Chave Crossbar Reconfigurável para Implementação Dinâmica de Topologias em Redes de Interconexão, V Workshop em Sistemas Computacionais de Alto Desempenho, 2004. pp. 74-81.

[16] VHDL Simili Tool Set, versão 3.0. Fabricado por SymphonyEDA, abril de 2005. Disponível em <www.symphnyeda.com>. Acesso em abril de 2006.

[17] W. J. Dally, B. Towles, *Principals and Practices of Interconnection Networks*, Morgan Kaufmann, 3ª Edição, 2003. 550 pp.

[18] S. S. Mneimneh, et al, "On Scheduling Using Parallel Input-Output Queued 11111111111 Switches with no Speedup", Proceedings of IEEE Workshop on High Performance Switching and Routing, 2001. pp. 317-323.

[19] C. L. Wu, T. Y. Feng, "The Universality of The Shuffle-Exchange Network", IEEE Transactions on Computers, Volume C-30, Edição 5, 1981. pp. 324-332.

[20] M. Ould-Khaoua, G. Min, "Circuit Switching: An Analysis for K-Ary N-Cubes With Virtual Channels", IEEE Proceedings – Computers and Digital Techniques, Volume 148, Edição 6, Novembro de 2001, pp. 215-219.

[21] P. Kermani, L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique", Computer Networks, Volume 3, 1979. pp.267-286.

[22] P. Kitsos, et al, "A Reconfigurable Linear Feedback Shift Register (LFSR) for the Blootooth System", Proceedings of the 8th IEEE International Conference on Eletronics, Circuits and System, 2000. pp. 991-994.