Avaliação de Performance de Arquiteturas para Computação de Alto Desempenho

Karin Strauss
kstrauss@larc.usp.br

Wilson V. Ruggiero
wilson@larc.usp.br

julho de 2002

Resumo

Este trabalho apresenta a avaliação de desempenho de três componentes de arquitetura inovadora criados no Projeto Blue Gene, da IBM.

Com a crescente demanda por poder computacional e a incapacidade de computadores baseados no modelo seqüencial em atender essa demanda, surgiram os computadores paralelos. A evolução natural, então, foi o aumento do paralelismo, de alguns processadores para milhares deles. Assim nasceram os chamados computadores maciçamente paralelos.

Uma arquitetura surgida dessa evolução é conhecida como arquitetura celular. Ela baseia-se na simplicidade dos elementos de computação e no vasto paralelismo, sendo que o trabalho conjunto de muitas células resulta em grandes computações. Esse campo, porém, é relativamente novo e pouco explorado.

Um dos projetos que pretendem estudar e construir máquinas de arquitetura celular é o Projeto Blue Gene, sendo desenvolvido no IBM T.J.Watson Research Center. Além da inovação na arquitetura da máquina como um todo, também foram propostas inovações específicas como a arquitetura dos próprios processadores, unidades de ponto flutuante e redes de interconexão.

Foram escolhidos três dos componentes de arquitetura mais inovadora para o estudo, com o objetivo de avaliar sua capacidade antes mesmo de sua fabricação. Assim, é possível, se necessário, alterar o projeto dos componentes sem grande impacto no orçamento, já que os custos de refazer o processo de fabricação são evitados.

1 Introdução

Poder computacional é cada vez mais necessário nos dias de hoje, tanto em aplicações comerciais quanto em aplicações científicas. Por isso, as grandes empresas de computação têm investido muito esforço no desenvolvimento de supercomputadores de grande poder computacional e de altíssimo desempenho. Projetos ambiciosos têm surgido a partir dos investimentos destas empresas, alguns até pretendendo produzir máquinas com poder maior do que o dos maiores supercomputadores atuais somados.

Um desses projetos surgiu no centro de pesquisas T.J.Watson, da IBM. Trata-se do projeto *Blue Gene* [1] [2] [3], que visa desenvolver uma série de supercomputadores de arquitetura inovadora. O projeto baseia-se na idéia de se construir máquinas paralelas em forma celular, ou seja, compostas de blocos simples e de mesma estrutura, ligados apenas a alguns outros blocos vizinhos semelhantes [4]. Através da ligação desses blocos seria possível construir uma máquina paralela de alto desempenho e grande escalabilidade, já que, caso se necessite de mais poder computacional, basta adicionar mais células.

Além da inovação do ponto de vista de ligação entre os componentes do sistema paralelo e das redes de interconexão, pretende-se, no projeto *Blue Gene*, inovar em aspectos arquiteturais mais específicos, como na arquitetura do próprio processador e de unidades auxiliares.

Avaliar o sistema como um todo, de uma só vez, é uma tarefa bastante complicada: por isso, divide-se a avaliação em tarefas menores, que enfocam subsistemas mais restritos, tornando a análise mais simples.

Os primeiros subsistemas a passar por avaliação são, normalmente, os de arquitetura mais inovadora, justamente porque seu comportamento ainda não é dominado pelos projetistas. Em seguida, passa-se a fazer avaliações mais abrangentes, examinando a interação entre os diversos componentes do sistema.

Avaliar e validar o ganho de desempenho com certos aspectos arquiteturais do sistema é de grande importância para o sucesso do projeto e, para que isso seja possível antes mesmo de a máquina real existir, são usados simuladores [5] [6].

Num projeto de grande porte como o *Blue Gene*, há vários grupos de pessoas distintos envolvidos. Cada um desses grupos está interessado em aspectos diferentes das máquinas.

Como exemplo, pode-se citar um engenheiro de *hardware* verificando se o *timing* projetado para o circuito propaga-se de forma adequada. O nível de detalhamento necessário nessa verificação é bastante diverso do nível de detalhamento em que um engenheiro de *software* está interessado quando avalia o número de instruções gastas com comunicação em uma aplicação paralela.

Enquanto o engenheiro de *hardware* observa o sistema no nível de portas lógicas e transistores por um período relativamente curto de tempo (da ordem de ps) e em uma região bastante restrita do *hardware*, o engenheiro de *software* observa o sistema em um nível muito mais alto, de duração maior, e está interessado no comportamento do sistema como um todo.

Como o simulador usado pelo engenheiro de *hardware* precisa ser muito detalhado, seu alcance é limitado. São necessários grandes períodos de tempo para simular pequenas frações de segundos. Por esse motivo, é necessário que haja diversos simuladores, cada qual com finalidade, velocidade e nível de detalhamento distintos.

Pode-se, também, obter resultados dos simuladores mais detalhados e mais lentos para calibrar os simuladores mais rápidos e menos detalhados. Dessa forma, aumenta-se a precisão desses últimos, tornando possível estimar o desempenho do sistema com *workloads* reais em tempos razoavelmente pequenos.

As avaliações de desempenho em um projeto como esse são numerosas e freqüentemente utilizam-se de um ou mais simuladores, com diferentes níveis de detalhes.

O objetivo principal da pesquisa foi avaliar o desempenho de três componentes diferentes das máquinas em desenvolvimento no projeto Blue Gene.

Os três componentes estudados foram cuidadosamente escolhidos por serem três fatores significativos para a estimativa do desempenho geral dos supercomputadores *Blue Gene*. Analisar o impacto de cada um dos componentes no desempenho do sistema, e os custos associados a eles, em momento prévio à fabricação das máquinas é importante pois, nessa fase, ainda é possível fazer alterações no projeto.

O primeiro componente é o processador Cyclops. Esse processador é composto por um número grande de unidades de execução, também chamadas de *threads*. Essas unidades de execução compartilham memória *cache*, memória principal e unidades de ponto flutuante. A forma como as *threads* compartilham o *cache* de dados é configurável através da forma de endereçamento.

O Cyclops tem uma arquitetura bastante inovadora, cujo comportamento ainda não é completamente conhecido e dominado. Além disso, é uma das opções de processador que podem ser empregados em supercomputadores *Blue Gene*. Algumas das grandes preocupações dos projetistas são qual banda de memória pode ser obtida com a utilização do *chip*, se é possível atingir o limite teórico calculado e quais políticas devem ser usadas para que isso aconteça. Assim, a avaliação de desempenho da banda de memória do *chip* é um tópico relevante de pesquisa, que pode ajudar os projetistas a estimar o desempenho geral de uma máquina construída com blocos básicos desse tipo.

O objetivo é, então, estudar o comportamento da taxa de transferência de dados entre as *threads* e a memória principal e verificar se é possível atingir o limite teórico calculado, com várias configurações de *cache* de dados e de distribuição de dados na memória principal.

O segundo componente é uma unidade de ponto flutuante dupla. Essa é uma unidade denominada pela própria IBM como SIMOMD (*Single Instruction, Multiple Operation, Multiple Data*), por ter instruções paralelas, cruzadas, assimétricas e complexas.

A unidade de ponto flutuante dupla também é um projeto bastante novo e pouco explorado. Sua arquitetura apresenta diferenças relevantes em relação a outras unidades de ponto flutuante paralelas, já que possibilita não só operações paralelas como também operações cruzadas, assimétricas e complexas. Isso facilita muitas operações freqüentemente utilizadas em aplicações científicas, incluindo diversas computações de álgebra linear (multiplicação de matrizes, por exemplo) e com números complexos. Para que seja possível aproveitar todo o potencial que a unidade pode oferecer, é necessário que haja suporte de compiladores, e que as aplicações sejam codificadas de forma adequada. Assim, a avaliação é feita sobre o conjunto arquitetura-compilador-aplicação. Como o projeto *Blue Gene* é bastante voltado à construção de máquinas de alto desempenho para aplicações científicas, e esse componente tem bastante influência no desempenho geral, é importante ter dados a seu respeito.

O objetivo é, então, estudar os ganhos de se ter uma unidade de ponto flutuante dupla como esta em relação a uma unidade simples similar. Nesse caso, a avaliação de desempenho não se restringe apenas à arquitetura. Devem ser também levados em conta o compilador e os algoritmos utilizados.

O terceiro componente é a forma de interconexão entre as células do sistema. Essas células farão sua comunicação através de várias redes distintas. A rede a ser estudada apresenta uma topologia toroidal e constitui a principal rede para aplicações do *Blue Gene*.

A arquitetura altamente paralela da máquina a ser construída implica em uma comunicação entre células muito intensa, já que a computação e os dados estão distribuídos entre eles. Esse comportamento de comunicação pode influenciar negativamente o desempenho da computação. Assim, o desempenho da rede de interconexão é, novamente, um fator importante para o desempenho geral da máquina, e deve ser estudado.

O objetivo é, então, avaliar o desempenho de um algoritmo distribuído bastante utilizado em computação científica, verificando a eficácia dessa rede de interconexão.

As demais seções deste artigo estão organizadas da seguinte forma: na seção 2 é apresentada a avaliação de desempenho do processador Cyclops; na seção 3 a avaliação de desempenho da unidade de ponto flutuante dupla é explicada; a seção 4 mostra a avaliação de desempenho da rede de interconexão toroidal estudada; na seção 5 são apresentadas as conclusões e comentados os trabalhos futuros.

2 Cyclops

O *Cyclops* é uma unidade básica que pode ser utilizada na construção de uma das máquinas *Blue Gene*. Trata-se de uma célula *SMP (Symmetric Multi-Processor)* com múltiplas linhas de execução (daqui em diante chamadas de *threads*), memória embutida e dispositivos de comunicação integrados.

O *Cyclops* é ainda uma arquitetura em desenvolvimento. A configuração do *chip* estudada neste trabalho será descrita a seguir.

O *chip* consiste em 128 *threads* com 64 registradores de 32 *bits* cada. Cada registrador tem precisão simples (32 *bits*), mas é possível agrupar dois registradores em um par para prover precisão dupla (64 *bits*). Além disso, cada *thread* tem um contador de instruções (PC) e uma unidade lógico-aritmética de ponto fixo.

Essas *threads* são organizadas em grupos de 4, chamados de *quads*. Cada *quad*, portanto, consiste nessas 4 *threads* que compartilham uma unidade de ponto flutuante e uma unidade de *cache* de dados de 16 KB. Cada dois *quads* compartilham um *cache* de instruções de 32 KB, de associatividade *8-way* e linha de 64 *bytes*. A memória principal, compatilhada por todas as *threads*, consiste em 8 MB, divididos em 16 bancos de 512 KB. Essa configuração pode ser observada na figura 1.

Se duas ou mais *threads* tentam utilizar o mesmo recurso em um mesmo ciclo, uma delas é selecionada como a vencedora e aloca o recurso. As *threads* restantes esperam até que o recurso

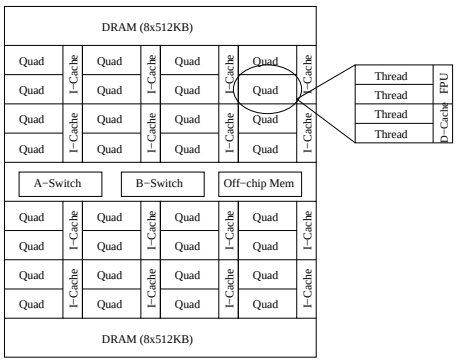


Figura 1: Diagrama de blocos do chip analisado - Um chip contém 32 quads, unidades de 4 linhas de execução (threads), que compartilham uma unidade de ponto flutuante (FPU) e um *cache* de dados (D-Cache). Pares de quads compatilham um *cache* de instruções (I-Cache). O chip também contém 16 bancos de 512 KB de memória principal embutida, hardware de interconexão (A-switch e B-switch) e um controlador para acessar memória externa (opcional).

esteja disponível. A decisão de qual *thread* tem prioridade é feita utilizando-se a técnica de *round-robin*, para garantir que todas as *threads* terão a mesma chance de utilização de um recurso.

O *chip* Cyclops ainda possui um mecanismo de sincronização entre *threads* por *hardware*. Esse mecanismo permite que se faça sincronizações muito rápidas, da ordem de alguns ciclos após a última *thread* chegar à barreira.

Como já explicado, a memória principal do *chip* consiste em 16 bancos de 512 KB, num total de 8 MB. São utilizados apenas 24 *bits* para o endereçamento físico da memória. Os 8 *bits* restantes são utilizados para definir no *cache* de qual *quad* os dados serão colocados. Os dados podem ser colocados em um *cache* pertencente ao mesmo *quad* da *thread* que os resquisitaram ou em outro *cache* definido por esses 8 *bits*.

Como os dados podem ser colocados em qualquer um dos *caches*, dependendo do *software*, aparece um problema de consistência de dados. Esse problema só pode ser evitado com uma programação cuidadosa, já que não há suporte de *hardware* para isso.

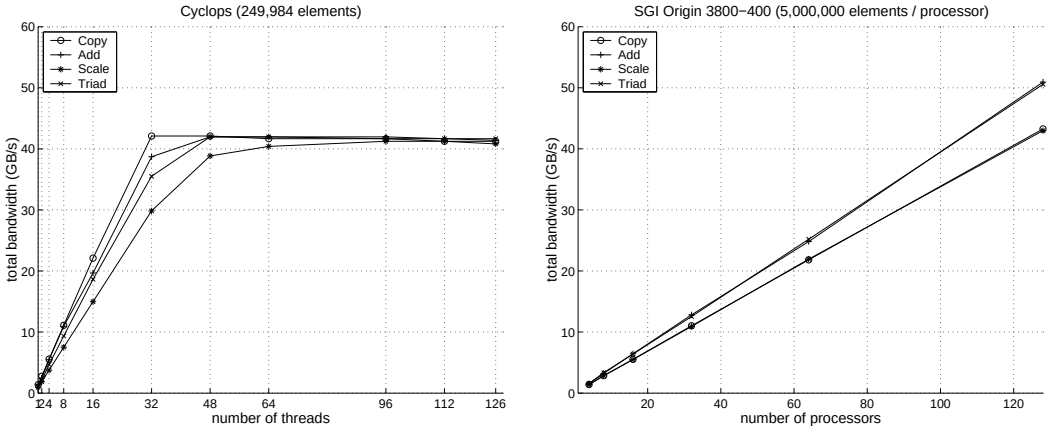
O tempo de acesso em modo *burst* para dois blocos de 32 *bytes* subseqüentes em um banco de memória, ou seja, 64 bytes, é de 12 ciclos. Pode-se fazer acessos simultâneos a todos os bancos de memória. Assim, supondo uma freqüência de excitação de 500 MHz, temos uma banda teórica de 42GB/s (64 B*16 bancos*500 MHz/12 ciclos).

Para o estudo, foi utilizado o STREAM *benchmark* [7][8] executado sobre um simulador de instruções específico para o *chip*, que inclui o modelo de *threads* e compartilhamento de recursos. O STREAM foi desenvolvido especialmente para a medição de banda de memória e consiste em quatro operações com vetores: cópia (Copy), adição (Add), multiplicação por escalar (Scale) e multiplicação-e-soma (Triad).

Os experimentos indicaram que é possível atingir a banda máxima teórica quando se traz os dados para o *cache* pertencente ao mesmo *quad* da *thread* que os acessaram. Pode-se comparar o desempenho do *chip* com o de uma máquina comercial, como mostra a figura 2.

Pode-se notar que os vetores usados com o SGI Origin 3800/400 são muito maiores, e que o Cyclops não tem memória suficiente para experimentos com vetores deste tamanho.

De qualquer forma, é importante notar que um único chip do Cyclops pode atingir banda de memória sustentável comparável à atingida por uma máquina comercial topo-de-linha.



(a) (b)

Figura 2: Comparação da performance de (a) Cyclops (tamanho do vetor: 249.984 elementos) *vs.* performance de (b) SGI Origin 3800-400 (tamanho do vetor: 5.000.000 elementos por processador)

3 Unidade de Ponto Flutuante Dupla

A unidade de ponto flutuante dupla estudada, chamada *PowerPC 440 FP2* é uma extensão da unidade de ponto flutuante simples projetada para o processador *PowerPC 440*, chamada *PowerPC 440 FPU*.

A arquitetura projetada baseia-se na duplicação da unidade original. Uma das unidades é, então, chamada de primária e a outra é chamada de secundária. As instruções originais, ou seja, as instruções normais da unidade de ponto flutuante simples, destinam-se à unidade primária. Foram adicionadas novas instruções ao *instruction-set*, tanto para operações simultâneas nas duas unidades quanto para operações na unidade secundária, estas similares às instruções primárias. Cada uma das unidades tem seu próprio banco de registradores, que podem ser endereçados separadamente. Também é possível endereçar os dois registradores de uma só vez. Pode-se, ainda, mover dados entre os bancos de registradores.

As novas instruções com as duas unidades incluem operações paralelas, cruzadas, assimétricas e complexas. A tabela 3 mostra alguns exemplos. As instruções assimétricas realizam operações diferentes mas relacionadas, nos dois caminhos de dados, enquanto que as instruções complexas podem ser simétricas ou assimétricas, mas são especificamente voltadas para computações com números complexos. O nome dado ao tipo de operações que a unidade é capaz de executar é SIMOMD, ou seja, *Single Instruction Multiple Operation Multiple Data*.

Classe	Exemplo	Operação na Unid. Primária	Operação na Unid. Secundária
Paralela	fpmadd fT, fA, fB, fC	$P_T = P_A * P_C + P_B$	$S_T = S_A * S_C + S_B$
Cruzada	fxcpmadd fT, fA, fB, fC	$P_T = P_A * P_C + P_B$	$S_T = P_A * S_C + S_B$
	fxmadd fT, fA, fB, fC	$P_T = P_A * S_C + P_B$	$S_T = S_A * P_C + S_B$
Assimétrica	fxcpnpma fT, fA, fB, fC	$P_T = -P_A * P_C + P_B$	$S_T = P_A * S_C + S_B$
Complexa	fxcxnpma fT, fA, fB, fC	$P_T = -S_A * S_C + P_B$	$S_T = S_A * P_C + S_B$

Tabela 1: Instruções de exemplo da *PowerPC 440 FP2*

É possível, com essa unidade, fazer transferências de dados de palavras de 128 *bits*. Uma palavra de 64 *bits* é colocada no registrador primário selecionado na instrução e a outra é colocada no

registrador secundário correspondente. Nesse caso, os registradores são encarados como pares. Se um dado é colocado no terceiro registrador primário, o dado subseqüente será colocado no terceiro registrador secundário. O equivalente pode acontecer se a operação for um *store*.

As operações de movimentação de dados podem ser paralelas, onde a palavra endereçada na instrução é colocada em um registrador primário, também selecionado na instrução, e a palavra consecutiva a ela é colocada no registrador secundário correspondente.

Outro modo de movimentação de dados é a operação cruzada. Nesse caso, a palavra endereçada é colocada em um registrador secundário selecionado na instrução e a palavra seguinte é colocada no registrador primário.

Para que esse tipo de operação seja eficiente, porém, os dados devem estar alinhados em relação a seu tamanho (dados de 32 *bits* alinhados em posições com endereço múltiplo de 4 *bytes*, dados de 64 *bits* alinhados em endereço múltiplo de 8 *bytes*) e precisa se encaixar completamente em uma região alinhada de 256 *bits*, ou 16 *bytes*, que é o tamanho de uma linha do *cache* do *PowerPC 440*.

As restrições podem ser mais bem ilustradas pela figura 3. Elas aparecem porque a arquitetura de *cache* do processador força que transferências de dados só possam ser feitas a partir de uma das metades de uma linha de *cache*.

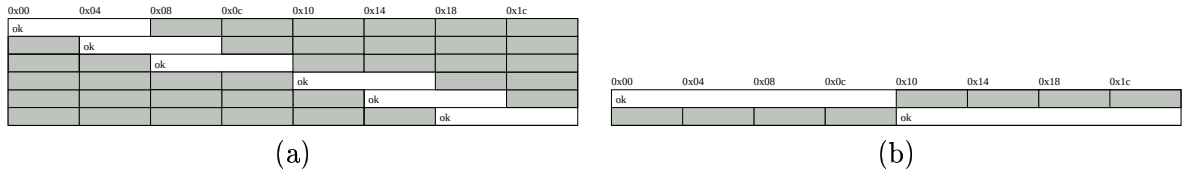


Figura 3: Restrições de alinhamento em uma linha de *cache* para o *PowerPC 440 FP2*. (a) mostra acessos de 64 *bits* (1 palavra de 64 *bits* ou duas palavras de 32 *bits* e (b) mostra acessos de 128 *bits* (2 palavras de 64 *bits*).

Se as restrições não forem seguidas, o tempo para que a operação termine aumenta muito (fica da ordem de milhares de ciclos, contra alguns ciclos se as restrições forem atendidas), já que é gerada uma interrupção de alinhamento e o *software* deve forçar o alinhamento dos dados de forma a atender às restrições.

O conjunto de *kernels* utilizados para a avaliação de desempenho da tríade arquitetura-compilador-algoritmo faz parte do BLAS (*Basic Linear Algebra Subprograms*) [9] que, como o próprio nome diz, é um conjunto de rotinas básicas de álgebra linear. Os *kernels* utilizados foram o daxpy ($y[n] = a * x[n] + y[n]$, onde x e y são vetores), dgemv ($y[m] = y[m] + A[m][n] * x[n]$, onde x e y são vetores e A é uma matriz) e dgemm ($C[m][n] = C[m][n] + A[m][k] * B[k][n]$, onde A, B e C são matrizes). Esses *kernels* foram executados sobre um simulador baseado em descrição de *hardware* e o modelo simulado inclui processador e unidade de ponto flutuante.

Os experimentos mostraram que a unidade pode atingir ganhos significativos em desempenho se devidamente programada.

Os resultados indicam que o compilador é capaz de aproveitar os novos recursos da unidade de ponto flutuante quando alimentado com dados sobre alinhamento e sobreposição de vetores e matrizes, principalmente para *kernels* mais simples, com poucos aninhamentos.

O compilador ainda encontra dificuldades em otimizar códigos mais complicados, com vários níveis de aninhamento. É possível, porém, gerar código otimizado para a unidade com o auxílio do recurso de *intrinsics* do compilador, onde o programador tem um maior controle sobre o código de baixo nível gerado, podendo escolher diretamente a seqüência de instruções *assembly* a serem utilizadas em determinados trechos do programa.

4 Rede de Interconexão Toroidal

Numa das máquinas *Blue Gene*, a rede de interconexão normalmente utilizada para troca de dados de aplicações é a rede toroidal. Assim, cada uma das células é ligada a suas vizinhas através de seis *links*, como mostrado na figura 4. Todas as células responsáveis pela computação de aplicações serão interligadas em uma topologia toroidal de três dimensões.

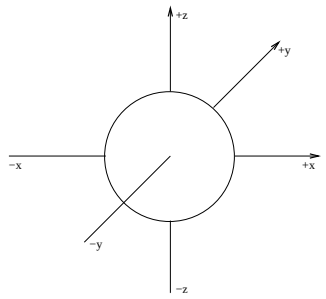


Figura 4: *Links* (com direção e sentido) de uma célula

Os pacotes têm formato proprietário e seu roteamento é feito por *hardware*. Cada pacote tem tamanho fixo de 256 *bytes* e cabeçalho de *hardware* de 8 *bytes*, incluído no pacote. O dispositivo de comunicação da rede toroidal é mapeado em memória.

Todas as transmissões e recepções entre processador e dispositivo de comunicação devem ser feitos em operações de 128 *bits*, ou 16 *bytes*. Assim, toda a interface entre processador e dispositivo é feita através da unidade de ponto flutuante dupla, a única unidade capaz de fazer transferências de dados tão largas.

Como as transmissões são feitas em blocos de 16 *bytes* e o cabeçalho de *hardware* só ocupa 8 *bytes*, decidiu-se utilizar os 8 *bytes* seguintes, já na área de dados, como cabeçalho de *software*. Assim, a primeira parte transmitida de um pacote inclui apenas cabeçalhos e através do cabeçalho de *software* pode-se passar informações a respeito dos dados sendo transmitidos.

O roteamento pode ser feito em modo determinístico, ou seja, é possível determinar qual o caminho seguido pelo pacote da origem para o destino. Outro modo de roteamento é o dinâmico e, nesse caso, não se pode determinar qual o caminho seguido por um pacote: ele depende do tráfego nos *links* de cada uma das células por onde o pacote passa. O modo de roteamento é determinado através de configurações feitas via *software* no cabeçalho do pacote.

Podem-se configurar outras características de roteamento através do cabeçalho do pacote como, por exemplo, os chamados *hint bits*. Os *hint bits* são destinados a instruir o *hardware* a enviar o pacote em um determinado sentido preferencial.

Outra característica interessante disponível através dessa rede de interconexão é o *multicast* em linha. De acordo com um *bit* configurável do cabeçalho, o chamado *deposit bit*, pode-se instruir o *hardware* a depositar o pacote em cada uma das células por onde o pacote passa até alcançar seu destino.

Esse tipo de operação, porém, só pode ser feita se o caminho que o pacote segue é uma linha, sem mudança de direção. Assim, só se pode enviar pacotes *multicast* se a origem e o destino estiverem em uma mesma linha do toróide.

Como para cada par origem-destino em uma direção do toróide há duas formas de atingir o destino saindo da origem (sentido positivo ou negativo), usa-se os *hint bits* para especificar qual dos caminhos devem ser seguidos.

Como cada direção do toróide está fechada em anel, pode-se, também, enviar pacotes para todas as células em uma linha. Basta que uma célula envie um pacote para si mesma. O *hardware* envia o pacote à célula vizinha, que copia o pacote para si e também o envia à proxima célula. Quando o

pacote chega a um extremo, ele é enviado ao outro extremo, dando a volta no anel. O processo se repete até que o pacote atinja a célula originadora (que é o próprio destino). Assim, pode-se fazer um *multicast* total na linha, como mostra a figura 5.

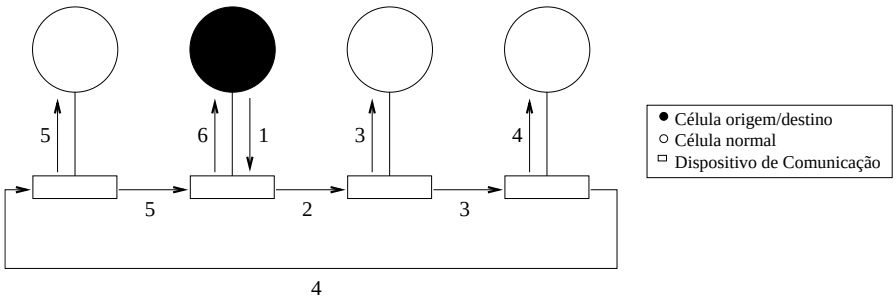


Figura 5: *Multicast* completo em linha: os números mostram o caminho do pacote, partindo da célula origem, sendo depositada e reenviada em cada uma das outras células e voltando para a célula inicial, que também é o destino.

Apesar da limitação de manter o pacote em uma linha quando se usa os *deposit bits*, ainda é possível fazer *broadcasts* que atinjam todas as células, com algum suporte de *software*. Basta para isso que se marque o pacote como *broadcast* em algum campo da área de dados e se envie esse pacote com um *multicast* em linha em uma direção (suponhamos a direção x). Assim, todas as células daquela linha recebem esse pacote. Em seguida, o *software* verifica que aquele é um pacote de *broadcast* e o envia como *multicast* em linha em uma outra direção (digamos que seja a direção y). Assim, todos os pacotes ao longo de todos os anéis da direção inicial (x) recebem uma cópia do pacote. O *software* deve entrar novamente em ação e enviar agora um pacote de *multicast* em linha na direção restante (z, nesse caso). Dessa forma, cada uma das células do sistema recebe uma cópia do pacote inicial. O processo de *broadcast* descrito acima está ilustrado na figura 6.

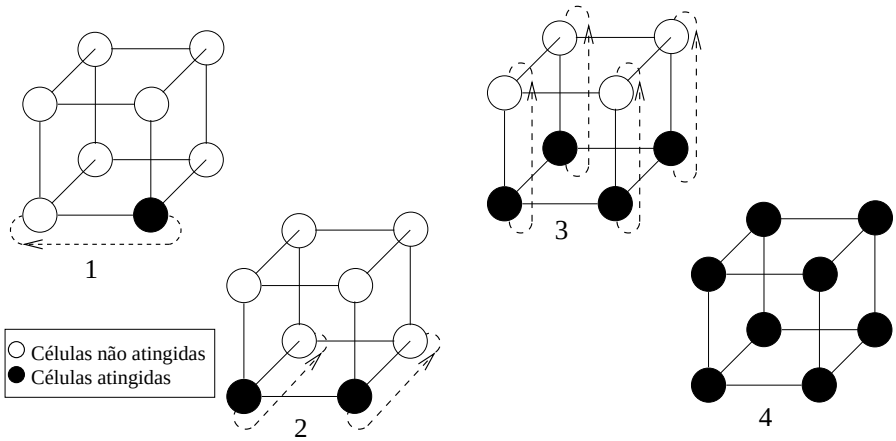


Figura 6: *Broadcast* com o auxílio de *software*: quatro etapas. (1) dados saem da origem, (2) dados terminam de percorrer eixo x, (3) dados terminam de percorrer eixo y e (4) dados terminam de percorrer eixo z.

O desempenho dessa rede de interconexão foi avaliado utilizando-se um simulador de sistema completo especialmente desenvolvido para a máquina. Ele modela várias células e a interconexão que as liga e é capaz de efetivamente simular a troca de dados entre elas pela rede. Assim, é possível

executar aplicações paralelas que se utilizem da rede de interconexão para troca de dados e colher informações sobre elas com o simulador.

A aplicação selecionada para a avaliação foi a multiplicação tridimensional de matrizes, adaptada de [10].

Foi possível observar pelos experimentos que o *overhead* causado pelo cabeçalho do pacote de dados é relativamente baixo, não tendo um impacto significativo no desempenho do sistema.

Outro resultado importante é que o *multicast* em linha é uma grande ferramenta para algoritmos tridimensionais onde uma informação precisa ser replicada em um subconjunto de células, pois evita que seja necessário transmitir essa informação várias vezes. Vale ressaltar que o *multicast* em linha diminui a quantidade de comunicação necessária para a resolução do problema. Diminuindo a quantidade de informações enviadas, ganha-se tempo de processamento para a computação propriamente dita, tempo esse que estaria sendo utilizado para o envio das informações. É importante perceber que não necessariamente se diminui a ocupação nos *links* de comunicação, já que a informação transmitida por *multicast* trafega da mesma forma pelos *links*, ao longo de todas as células envolvidas.

5 Conclusões e Trabalhos Futuros

Os simuladores mostraram-se ferramentas muito valiosas para estudos de avaliação de desempenho.

Para este estudo foram utilizados diferentes simuladores para diferentes propósitos. Cada simulador tem seus pontos fortes e fracos, sendo úteis para obter resultados específicos diferentes.

Foi possível verificar que o Cyclops é capaz, na prática, de atingir os níveis teóricos de banda sustentável de memória, podendo ser comparado a máquinas comerciais topo-de-linha. O único fator limitante para o uso do *chip* individualmente é a quantidade de memória disponível. Assim, o uso do *chip* para aplicações paralelas que utilizam poucos dados em cada célula parece ser indicado.

A unidade de ponto flutuante dupla estudada pode ser bastante utilizada para aplicações científicas. Ela provê uma maneira eficiente de aumentar o desempenho desse tipo de aplicação, sem aumentar muito a complexidade do *hardware*.

Porém, é muito importante que haja suporte de ferramentas para a geração de código para essa unidade. O compilador estudado é bastante eficiente para *kernels* simples. Conforme a complexidade dos *kernels* aumenta, o compilador apresenta mais dificuldades para gerar código eficiente.

Cabe, então, ao programador otimizar o código manualmente para a unidade, tarefa suportada pelo compilador através do recurso chamado de *intrinsics*.

A topologia toroidal mostrou-se adequada para a computação tridimensional de multiplicação de matrizes. Outras aplicações adequadas ao conceito de computação celular também parecem se adequar à topologia. O *overhead* observado não causa impacto significativo no desempenho como um todo.

O recurso de *multicast* em linha mostrou-se bastante eficiente e útil para aplicações onde é necessário transmitir a mesma informação para diversas células.

No caso do Cyclops, ainda é necessário estudar outros aspectos que não apenas a banda de memória atingida pelo sistema.

Outros benchmarks relacionados a aplicações científicas podem ser utilizados, como o BLAS, para a avaliação do compartilhamento das unidades de ponto flutuante entre as *threads*.

Além disso, o recurso de barreiras rápidas implementadas em *hardware* pode ser mais explorado utilizando aplicações altamente paralelizáveis que necessitem de sincronização.

No caso da unidade de ponto flutuante dupla, pode-se continuar o estudo com *kernels* que utilizem operações complexas, como por exemplo o zdgemm.

Kernels de álgebra linear esparsa, ou mesmo outros *kernels* bastante utilizados em aplicações científicas, como por exemplo o FFT *(Fast Fourier Transforms)*, podem também ser utilizados para aprofundar o estudo.

O estudo da rede de topologia toroidal, pode ser aprofundado utilizando outros *kernels* científicos paralelizáveis.

Se uma biblioteca MPI for desenvolvida para uso sobre essa rede, um estudo interessante é a implementação do dgemm tridimensional sobre essa nova biblioteca, para avaliar o *overhead* introduzido por ela. Ainda se pode utilizar o nível 2 do *benchmark* NAS [11], criado pela NASA, para avaliar os ganhos com a biblioteca MPI sobre a rede de topologia toroidal em relação a uma rede Ethernet.

Para avaliar o desempenho do conjunto unidade de ponto flutuante e rede de topologia toroidal, ainda se pode utilizar aplicações de processamento sísmico, química computacional e modelamento de clima do SPEChpg96 [12] (*Standard Performance Evaluation Corporation: High Performance Group*), *kernels* adaptados como a FFT complexa unidimensional e a decomposição LU blocada do SPLASH [13] (*Stanford Parallel Applications for Shared Memory*) e aplicações adaptadas também do SPLASH como simulação de oceanos e simulação de água com ou sem estruturas de dados especiais.

Referências

[1] F. Allen et al. Blue Gene: A vision for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40(2):310–328, 2001.

[2] Eric J.Lerner. Cellular architecture builds next generation supercomputers. *Think Research*, 1, June 2002.

[3] G.Almasi et al. Cellular supercomputing with system-on-a-chip. In *ISSCC 2002 Proceedings*, San Francisco, California, USA, 2002. IBM T.J.Watson Research and IBM Enterprise Server Group.

[4] Moshe Sipper. The emergence of cellular computing. *Computer Magazine*, 1, Julho 1999.

[5] Steve Alan Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, 1998.

[6] Shubhendu S. Mukherjee, Sarita V. Adve, Todd Austin, Joel Emer, and Peter S. Magnusson. Performance simulation tools. *Computer Magazine*, 1, February 2002.

[7] John D. McCalpin. Sustainable memory bandwidth in current high performance computers, 1995. http://home.austin.rr.com/mccalpin/papers/bandwidth/.

[8] Memory bandwidth: STREAM benchmark performance results. http://www.cs.virginia.edu/stream/ref.html.

[9] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. Society for Industrial and Applied Mathematics, 1991.

[10] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5), 1995.

[11] NAS parallel benchmarks. `http://www.nas.nasa.gov/Software/NPB`.

[12] Spec benchmarks. `http://www.specbench.org`.

[13] Splash-2: Stanford parallel applications for shared memory. `http://www-flash.stanford.edu/apps/SPLASH/`.