Uma abordagem de alto desempenho para multiplicação de matrizes densas em sistemas reconfiguráveis

Viviane Lucy S. Souza, Victor W. C. de Medeiros, Derci de O. Lima e Manoel E. de Lima
Universidade Federal de Pernambuco
Centro de Informática
{vlss, vwcm, dol, mel}@cin.ufpe.br

Resumo

A demanda por máquinas de alto desempenho e por novas estratégias que buscam melhorar o processamento de dados em aplicações de computação científica tem crescido muito nos últimos anos. Algumas novas arquiteturas baseadas em GPU, processadores Cell e FPGA ou ainda plataformas híbridas aparecem como soluções para esses problemas.

Neste trabalho nós apresentamos uma arquitetura de alto desempenho para implementação de multiplicação de matrizes densas em uma plataforma comercial híbrida, o RASC (Reconfigurable Application-Specific Computing). O RASC foi desenvolvido pela Silicon Graphics e consiste em uma plataforma composta por um processador de propósito geral acoplado a co-processadores baseados em FPGA.

A arquitetura proposta investiga como a solução do problema de multiplicação de matrizes pode tirar proveito das características de uma plataforma com alto grau de paralelismo. Nós também investigamos a escalabilidade do algoritmo e os mecanismos de reuso de dados. Baseado nessas investigações um estudo de caso é sugerido e discutido me detalhes.

1. Introdução

Os FPGAs tornaram-se uma boa opção para aceleração de aplicações científicas devido a grandes avanços em sua arquitetura. Dentre estes avanços podemos citar, o maior número de blocos lógicos (CLBs), blocos de memória internos (BRAMs) e blocos específicos para processamento digital de sinais (DSP blocks) [1][2][3][4]. Todos estes avanços resultaram no surgimento de uma nova tecnologia para computação de alto desempenho [5][6][7].

Em geral, a maioria das plataformas que utilizam esta tecnologia adotam uma abordagem de *hardware software codesign,* onde um ou mais processadores de propósito geral estão conectados ao hardware reconfigurável através de um barramento de alta velocidade. Estas arquiteturas permitem a exploração de dois níveis de paralelismo: o paralelismo de granularidade grossa, implementado em múltiplos nós convencionais e, o paralelismo de granularidade fina obtido através da implementação em FPGAs.

Alguns fabricantes de supercomputadores, como a Cray e a SGI, têm desenvolvido estes sistemas reconfiguráveis de alto desempenho [8][9][10]. Resultados experimentais têm mostrado que estas arquiteturas híbridas podem alcançar melhor desempenho do que soluções baseadas unicamente em processadores de propósito geral. [11][12][13].

Neste trabalho, um sistema reconfigurável de alto desempenho, a plataforma híbrida RASC RC100, desenvolvida pela SGI, é utilizada como uma plataforma para processamento de multiplicação de matrizes em problemas de computação científica [14][15].

A arquitetura proposta visa explorar o paralelismo, a escalabilidade e o reuso de dados para usar de forma eficiente as propriedades da hierarquia de memória da plataforma RASC.

O artigo está organizado da seguinte forma: a Seção 2 apresenta uma visão geral da plataforma RASC. A Seção 3 apresenta alguns trabalhos relacionados que envolvem multiplicação de matrizes. A Seção 4 aborda a operação de multiplicação de matrizes. A Seção 5 apresenta um estudo de caso. Os resultados experimentais são apresentados na Seção 6. Por fim, as conclusões são apresentadas na Seção 7.

2. Plataforma SGI RASC RC100

A plataforma RASC [10], utilizada neste trabalho, é composta por um servidor SGI Altix 350 conectada a uma placa aceleradora baseada em FPGA, a RC100. A conexão entre os dois módulos é feita via interface NUMAlink que provê uma largura de banda de 3,2GB/s, como ilustra a Figura 1.

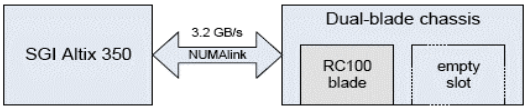

Figura 1. Conexão entre Sistema Altix e a RC100

A plataforma SGI Altix 350 consiste em dois processadores Itanium 2 dual-core, com 4GB de memória física cada. O módulo acelerador de hardware, o RC100, contém dois FPGAs Xilinx Virtex-4 LX200, dois chips de conexão ponto a ponto (TIO) e um FPGA extra responsável pela carga de *bitstreams* nos FPGAs de computação. Esses FPGAs operam a uma frequência máxima de 200MHz.

A Figura 2 mostra um diagrama do módulo de hardware. Os dois FPGAs computacionais são conectados aos chips de I/O através de uma interface de alta largura de banda e baixa latência, a SSP(Scalable System Ports) [19].Cada um dos FPGAs é acoplado a até 5 bancos de memória QDR SRAM de 8MB cada, com uma largura de banda de 1,6GB/s.

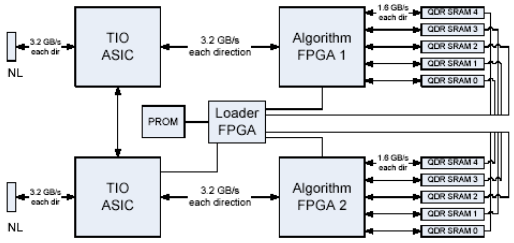

Figura 2. Módulo de Hardware RC100

Para facilitar a criação de novos projetos baseados no RASC, a SGI disponibiliza todos os módulos controladores e interfaces através do *RASC Core Services*. Estes módulos ocupam uma fração da área dos FPGAs e permite, entre outras coisas, a comunicação entre os algoritmos do usuário, a memória e as portas SSP.

Assim, um algoritmo desenvolvido para executar na plataforma RASC deve ser integrado ao *RASC Core Services* para acessar todas as funcionalidades do sistema.

3. Trabalhos Relacionados

Devido a importância da multiplicação de grandes matrizes na solução de problemas de computação científica, existe uma quantidade razoável de trabalhos focados em acelerar esta implementação. Desta forma, podemos destacar três abordagens na literatura que propõem como neste trabalho, arquiteturas especiais que buscam diminuir a latência da operação explorando o paralelismo e o reuso de dados das matrizes. Em [20] é apresentada uma arquitetura que contém elementos de processamento que não se comunicam entre si. Esta abordagem provê uma melhor utilização dos recursos reduzindo a lógica e roteamentos necessários para a comunicação entre os elementos. A arquitetura é baseada no máximo reuso de dados considerando que os dados das matrizes que estão sendo multiplicadas são lidos uma única vez e armazenado completamente nas memórias internas do FPGA (BRAMs). Esta arquitetura não explora o problema de capacidade de armazenamento, nem explora o fato que a busca e o processamento dos dados podem ser realizadas em *pipeline*, minimizando os requisitos de memória.

Em [21], três algoritmos são apresentados para a implementação da multiplicação de matrizes em FPGA. Nestes algoritmos, os elementos de processamento (PEs) são organizados em uma arquitetura de *array* linear. Os algoritmos propostos são baseados apenas na quantidade de recursos disponíveis (slices configuráveis, BRAMs e largura de banda) nos FPGAs, limitando o paralelismo completo da aplicação. Os dois primeiros algoritmos propostos têm a melhor condição de latência para execução paralela, mas requer a implementação de um grande número de elementos de processamento (PEs), o que na maioria das vezes é inviável. Contudo, o terceiro algoritmo, provê uma arquitetura mais realística e escalável, a partição do problema é realizada objetivando um casamento perfeito com as restrições impostas pelo hardware.

Em [22] é apresentado um algoritmo para multiplicação de matrizes de tamanhos arbitrários. A arquitetura é representada por um *array* linear de PEs e o algoritmo proposto divide o problema da multiplicação em duas partes que são tratadas por dois algoritmos diferentes, chamados algoritmos Master e Slave respectivamente. O algoritmo Master é responsável pela distribuição e o Slave provê as operações de multiplicação e adição. Neste trabalho, a distribuição dos dados e o processamento são realizados em paralelo, minimizando a latência e tornando a aplicação mais eficiente.

4. Multiplicação de matrizes

Nas próximas subseções, nós descrevemos uma arquitetura de alto desempenho que implementa a multiplicação de matrizes na plataforma RASC, a arquitetura é baseada na exploração do paralelismo e do reuso de dados das matrizes.

4.1. Estratégia para exploração do paralelismo

A estratégia básica para melhorar o desempenho da plataforma é utilizar os módulos RC100 em problemas de exploração de paralelismo real, como no caso da multiplicação de matrizes.

O algoritmo de multiplicação de matrizes, dado por $C_{n*n} = A_{n*n} \bullet B_{n*n}$ é um exemplo típico de exploração de paralelismo.

Cada elemento da matriz C é o resultado da operação $c_{ij} = \sum_{k=1}^{n} a_{ik} \bullet b_{kj}$, que inclui n - 1 somas de n produtos independentes.

Assim, se pudermos instanciar n multiplicadores e $n - 1$ somadores em um recurso reconfigurável, e se os módulos estão operando em *pipeline*, após a latência inicial necessária para necessária para encher o *pipeline*, nós podemos computar um elemento da matriz resultante em cada ciclo de relógio. E, considerando que os dados de entrada estão sempre disponíveis, a matriz C poderá ser calculada em n^2 ciclos de relógio.

Entretanto, a instância de um grande número de multiplicadores (n) nem sempre é possível, devido as limitações nos recursos do FPGA. Assim, se somente k multiplicadores e $k-1$ somadores podem ser instanciados, com $n = k \bullet r$, um elemento da matriz resultante pode ser calculado em r ciclos de relógio. Os r elementos parciais devem ser acumulados em registradores de saída durante o processo e multiplicação. Neste caso, a operação total consumiria aproximadamente $r \bullet n^2$ ciclos de relógio [17].

Neste trabalho, cada multiplicação é executada utilizando unidades multiplicadoras acumuladoras (MACs). Cada MAC calcula elementos independentes da matriz de saída.

Se é possível instanciar k MACs no RC100, cada um destes MACs, num dado instante, será responsável pelo calculo de um elemento da matriz resultante. Desta forma, considerando MACs operando em *pipeline*, a computação de um componente será realizada após n ciclos de relógio. Após esse tempo, já que temos k MACs, k elementos da matriz de saída estão disponíveis. Assim, a matriz resultante será calculada após aproximadamente $\frac{n^3}{k}$ ciclos de relógio. A Figura 3 ilustra essa arquitetura.

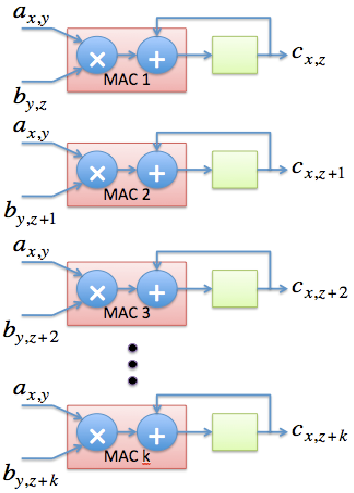

Figura 3. Arquitetura do multiplicador de matrizes com k MACs

4.2. Estratégia para exploração do reuso de dados

A arquitetura descrita acima considera uma largura de banda capaz de alimentar os MACs a cada ciclo de relógio. Entretanto, esta largura de banda nem sempre está disponível e, na maioria dos casos, representa o gargalo do sistema.

Para uma arquitetura composta por k MACs, operações com palavras de tamanho $N_{bits\ by\ word}$, um FPGA operando a uma frequência de relógio f, e desconsiderando a latência inicial de acesso aos dados na memória, uma largura de banda de $2 \bullet k \bullet f \bullet N_{bits\ by\ word}$, em bits por segundo, é necessária.

Assim, para um exemplo de aplicação na plataforma RASC, operando com palavras de 64 bits, e considerando a instância de 10 MACs, será necessário uma largura de banda de 32GB/s. Entretanto, a atual configuração do RASC apresenta uma largura de banda de somente 3,2GB/s.

Para melhorar os requisitos de largura de banda, nós devemos explorar as características do problema. No caso da multiplicação de matrizes, sabe-se que cada um dos elementos da matriz é reusado n vezes. Assim, é apropriado manter esses elementos armazenados próximos às unidades de processamento, nos FPGAs,

especificamente nas BRAMs, permitindo que esses dados sejam reusados mais rapidamente.

Neste caso, se uma linha de uma dada matriz A é armazenada nas BRAMs, ela pode ser reusada *n* vezes e a largura de banda necessária será reduzida a aproximadamente $k \bullet f \bullet N_{bits\ by\ word}$.

Agora, um MAC computa $r \bullet n$ elementos da matriz resultante, onde $r = \frac{n}{k}$. O $MAC_{i(\,i=1\ldots k\,)}$ calcula os elementos $c_{l,i}, c_{l,k+i}, c_{l,2\bullet k+i}, \ldots, c_{l,k\bullet(\,r-1\,)+i}$ da matriz resultante, onde $l = 1 \ldots n$.

Como esta solução foi desenvolvida para uma plataforma baseada em FPGAs, duas observações devem ser consideradas: primeiramente, o número de BRAMs disponíveis no FPGA deve ser suficiente para armazenar uma linha da matriz A (estratégia de reuso) e uma linha da matriz resultante. Desta forma, a ordem das matrizes operadas deve obedecer a condição $n \le \frac{N_{BRAMs}}{2}$, onde N_{BRAMS} é o número de palavras que podem ser armazenadas nas BRAMs; A segunda consideração é o compromisso que deve existir entre o número de MACs (que é limitada pela quantidade de blocos de DSP) [18] e a largura de banda disponível na arquitetura, esta relação é dada por $k = min\left(\frac{bw}{f \bullet N_{bits\ by\ word}}, N_{MAC\ DSP}\right)$, onde bw é a largura de banda da memória em bits por segundo, *k* é o número de MACs, , *f* é a freqüência de operação do FPGA e N_{MAC_DSP} é o número máximo de MACs que podem ser instanciados no FPGA usando os DSPs disponíveis.

Na plataforma RASC, com FPGAs Virtex-4 LX200, operando a uma freqüência de 200MHz, com 6.048 Kbits de memória BRAM , 96 blocos de DSP e uma largura de banda com a memória de 3,2GB/s, é possível armazenar até 96.768 palavras de 64 bits. Entretanto, devido as limitações de largura de banda, o número de MACs será limitado a apenas duas (2) unidades.

Assim, apesar do reuso dos dados da matriz A ter provido uma redução nos requisitos de largura de banda de 50%, ainda assim, o número de MACs permitidos foi menor que a capacidade do FPGA. Esse fato ocasiona um desperdício de recursos e uma diminuição na exploração do paralelismo da aplicação.

Para resolver esse problema, devemos considerar as seguintes características da operação de multiplicação de matrizes: o resultado da operação $C_{n*n} = A_{n*n} \bullet B_{n*n}$ é a soma das multiplicações de *n* vetores coluna da matriz A, representados aqui por, A^i, pelos correspondentes vetores linha da matriz B, representados aqui por B^i. Matematicamente, nós podemos representar esta operação por: $C_{n*n} = \sum_{i=1}^{n} A^i_{n*1} \bullet B^i_{1*n}$.

Uma vez que podemos particionar o problema em *n* multiplicações vetoriais, cada multiplicação de um vetor $n * 1$ por um vetor $1 * n$ produz uma matriz $n * n$ como resultado. Assim, com a leitura de $2 \bullet n$ elementos podemos executar n^2 multiplicações. Esta abordagem elimina o gargalo no acesso aos dados, já que o tempo de processamento de n^2 operações é, geralmente, maior do que o tempo necessário para leitura dos próximos $2 \bullet n$ elementos de dados.

Ao fim de *n* operações vetoriais, a multiplicação será completada. Nesta estratégia, resultados parciais devem ser acumulados em FIFOs na saída dos MACs As FIFOs devem ser suficientes para armazenar todos os n^2 elementos parciais gerados em cada operação vetorial. Assim, se *k* MACs são instanciados, *k* FIFOs de tamanho $\frac{n^2}{k}$ são necessários.

Além disso, há a necessidade de armazenar temporariamente os elementos das matrizes de entrada para serem processados, para tanto, FIFOs são inseridas na entrada dos MACs. A FIFO_A armazena os elementos da matriz A que são consumidos por todos os MACs. As FIFO_B_i armazenam os elementos da matriz B. Um elemento nesta FIFO é consumido somente pelo MAC *i* correspondente. A Figura 4 ilustra esta arquitetura.

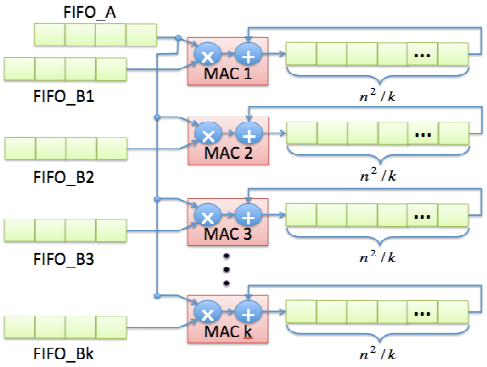

Figura 4. Arquitetura para multiplicação de matrizes com *K* MACs utilizando a estratégia de multiplicação vetorial.

Para implementar esta arquitetura, a quantidade de BRAMs disponíveis deve ser suficiente para armazenar os n^2 elementos da matriz resultante, assim, $n \le \sqrt{N_{BRAMs}}$, onde N_{BRAMs} é o número total de palavras que podem ser armazenadas nas BRAMs.

A consideração relacionada ao número de MACs foi relaxada, uma vez que, o acesso aos dados não é mais um gargalo do sistema. No caso extremo, devemos ter um número de MACs que satisfaça a condição, $k = min\left(\frac{bw \bullet n}{f \bullet 2 \bullet N_{bits\ by\ word}}, N_{MACDSP}\right)$.

Em um exemplo prático, utilizando a plataforma RASC e a abordagem descrita acima e, considerando palavras de 64 bits (ponto flutuante de dupla precisão), podemos operar matrizes da ordem de até 311, já que o FPGA provê 6048 Kbits de BRAM. E o número de MACs instanciados é limitado a 10 (dez), este limite é dado pela disponibilidade de blocos de DSP no FPGA e não mais pela largura de banda com a memória.

4.3. Escalabilidade da arquitetura

As condições para explorar a máxima capacidade de paralelismo de um FPGA e o uso da hierarquia de memória de forma mais eficiente foram discutidas na seção anterior. Para matrizes de ordens maiores onde essas condições não são satisfeitas, observamos que o problema pode ser particionado de forma a manter as condições estudadas e garantir o melhor desempenho do recurso reconfigurável.

Se *N* é a ordem da matriz, *n* é a máxima ordem que pode ser processada no FPGA para alcançar melhor desempenho e *K* é o número de recursos reconfiguráveis disponíveis, podemos particionar as matrizes de entrada em matrizes de ordem *n*. Cada particionamento da matriz A é chamado $A^{ij}\ (i = 1...\frac{N}{n}), (j = 1...\frac{N}{n})$, o mesmo é válido para a matriz B. Assim, uma sub-matriz da matriz resultante C pode ser calculado pela expressão $C^{ij} = \sum_{k=1}^{N/n} A^{ik} \cdot B^{kj}$ que corresponde a uma soma de produtos de matrizes de ordem *n*. Cada sub-matriz resultante pode ser calculada em um recurso reconfigurável e a adição pode ser feita em software ou na própria arquitetura, providos os devidos ajustes.

Desta forma, cada recurso reconfigurável será responsável pela computação de $\frac{\left(\frac{N}{n}\right)^3}{k}$ multiplicações de sub-matrizes .

Assim, garantiremos o máximo de exploração do paralelismo de granularidade grossa, dado que temos um conjunto de FPGAs operando em paralelo, e também de granularidade fina, com as operações internas a cada um dos recursos reconfiguráveis.

5. Estudo de caso – Multiplicação de Matrizes no RASC

Este estudo de caso consiste na implementação de um multiplicador de matrizes quadradas de ordem 100, baseado na arquitetura descrita anteriormente. Vamos considerar A, B e C, matrizes 100 * 100. Neste caso, um FPGA numa plataforma RC100 instancia 10 MACs de ponto-flutuante de dupla precisão e FIFOs de tamanho 1000 nas saídas de cada um desses MACs.

5.1. Estudo de caso - Arquitetura de memória

A memória da plataforma RC100 é dividida em dois bancos lógicos. Cada um desses bancos lógicos é composto pela combinação de dois bancos físicos de 1M palavras * 64 bits, resultando num banco lógico de 1M palavras * 128 bits.

Os dois bancos lógicos são organizados como segue: o primeiro é utilizado como uma memória de entrada e armazena as duas matrizes A e B; o segundo é utilizado como uma memória de saída e armazena a matriz resultante C.

A memória de entrada (primeiro banco lógico) é dividido em dois segmentos. O primeiro segmento armazena os elementos da matriz A ordenados por coluna, já que está é a forma padrão de leitura. O segundo segmento armazena a matriz B ordenada por linhas. A memória de saída (segundo banco lógico) tem apenas um segmento que armazena a matriz resultante C. A Figura 5 apresenta a distribuição da memória.

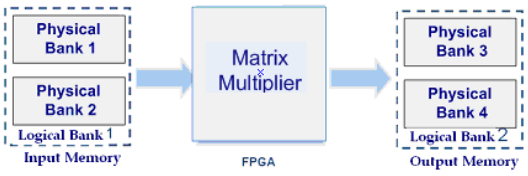

Figura 5. Distribuição lógica da memória para o multiplicador de matrizes

5.2. Estudo de caso - Arquitetura de processamento

O multiplicador é dividido em três blocos básicos:

- O bloco *memory control* é responsável por ler os dados de entrada da memória, escrevê-los nas FIFOs de entrada, lê-los nas FIFOs de saída e escrever os dados de saída na memória.
- O bloco *processing control* gerencia todo o fluxo de processamento, lê os dados das FIFOs de entrada, controla a arquitetura apresentada na Figura 5 e escreve os dados nas FIFOs de saída.
- O bloco de processamento é similar ao apresentado na Figura 5 e utiliza multiplicadores acumuladores (MACs) de ponto-flutuante precisão dupla, de acordo com o padrão IEEE-754 [18].

A Figura 6 apresenta um diagrama em blocos da arquitetura desenvolvida.

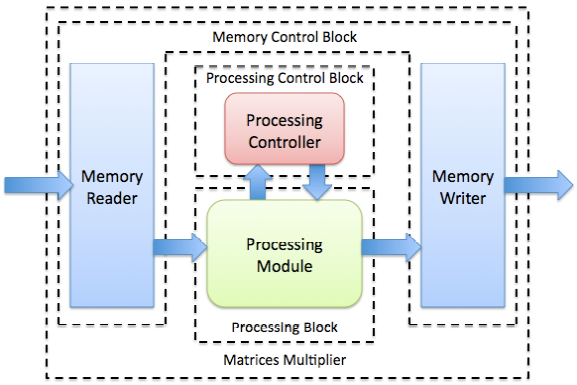

Figura 6. Diagrama em blocos do multiplicador de matrizes

5.3. Estudo de caso – Estimativa de desempenho

Embora este método de multiplicação tenha sido desenvolvido para a plataforma RASC, sua arquitetura parece ser viável para outras plataformas de alto desempenho.

Deste modo, nós podemos definir um modelo teórico para estimativa de performance do multiplicador de matrizes em algumas arquiteturas, baseado do número de ciclos de relógio necessários para executar as operações.

O modelo proposto é focado nos seguintes parâmetros relacionados ao hardware reconfigurável: a frequência de operação do FPGA, a largura de banda, e a latência da memória no acesso aos dados.

Neste caso particular, a multiplicação é particionada em 100 multiplicações de vetores. Deste modo, nós podemos definir o tempo de execução, em ciclos de relógio, do algoritmo como (T_{exec}), o tempo de execução de cada uma das 100 multiplicações vetoriais como (T_{mult}), mais o tempo de acesso aos dados necessários para a inicialização do algoritmo (T_{init}) e o tempo de escrita na memória de saída (T_{write}). Então, T_{exec} é dado por, $T_{exec} = T_{init} + 100 \bullet T_{mult} + T_{write}$.

O tempo de acesso ao primeiro dado (T_{init}), depende da quantidade de dados necessários para iniciar o processamento. Na nossa abordagem nós utilizamos um valor empírico de vinte palavras da matriz B e uma palavra da matriz A. Então, $T_{init} = T_{latency} + \left(\frac{20 \bullet 64}{bw}\right) \bullet f + T_{trans} + \left(\frac{1 \bullet 64}{bw}\right) \bullet f$, onde $T_{latency}$ corresponde a latência da memória, em ciclos de relógio, bw a largura de banda em bits/s., T_{trans} é o número de ciclos de relógio necessários para a transição entre os estados na máquina de estados, f corresponde a frequência do relógio e as expressões $\left(\frac{20 \bullet 64}{bw}\right) \bullet f$ e $\left(\frac{1 \bullet 64}{bw}\right) \bullet f$ estão relacionados ao número de ciclos de relógio necessários para ler vinte palavras da matriz B e uma palavra da matriz A respectivamente.

Deste modo, o tempo total, em ciclos de relógio, para a multiplicação de um vetor nesta arquitetura é de aproximadamente 1070 ciclos de relógio.

No final do processamento os dados são lidos das 10 FIFOs de saída e gravados na memória de saída. Contudo, as memórias de saída recebem em sua interface de entrada, palavras de 128 bits então, nós temos que agrupar a saída de 64 bits de duas FIFOs em uma única palavra de 128 bits que será gravada na memória. Como resultado, o tempo para escrever 10.000 elementos de 64 bits da matriz resultante é $T_{write} = 1000 \bullet T_{write\_5}$, onde T_{write_5} é o tempo necessário para escrever um bloco de cinco palavras de 128 bits (ou dez de 64 bits) na memória de saída, o que corresponde a pegar um elemento de cada uma das saídas dos MACs.

Para a plataforma RASC, o modelo de estimativa de performance determinou um valor de 114.020 ciclos de relógios para a execução da operação completa. O modelo pode ser aplicado em outras plataformas reconfiguráveis de alto desempenho, desde que suas frequências de operação, largura de banda e latência no acesso aos dados sejam conhecidos previamente.

6. Resultados Experimentais

O projeto foi simulado através do SSP Stub [19], uma ferramenta de verificação para projetos desenvolvidos na plataforma RASC. Esta ferramenta permite a simulação da aplicação do usuário integrada a transmissão e recepção de pacotes através do SSP.

Os resultados da simulação permitiram a validação e a estimativa de desempenho da solução desenvolvida na plataforma RASC. A estimativa, como na Seção anterior, é baseada na contagem do número de ciclos de relógio transcorridos do início ao fim da multiplicação.

A simulação na ferramenta SSP Stub consumiu 116.132 ciclos de relógio para a multiplicação da matriz quadrada 100*100. Na estimativa apresentada anteriormente, baseada numa abordagem teórica, o tempo de execução consumiu 114.020 ciclos de relógio, um erro de apenas 2%.

Com a finalidade de comparar a utilização da solução de hardware, na plataforma RC100, em diferentes freqüências, e a solução de software utilizando um processador convencional, foi gerada uma tabela. A Tabela 1 apresenta o tempo de execução em hardware para as freqüências 50 MHz, 100 MHz e 200 MHz e a execução em software em uma máquina com um processador Intel Xeon de 1.6 GHz e 4 GB de memória RAM.

Tabela 1- Comparação do tempo de execução em software e no RASC para diferentes frequências

Plataforma	Tempo de execução (ms)
RASC (50 MHz)	2,32263
RASC (100 MHz)	1,16132
RASC (200 MHz)	0,580657
Intel Xeon 1.6 GHz	10,748

7. Conclusões

O uso de FPGAs associados a processadores de propósito geral tem alcançado consideráveis níveis de aceleração na computação científica. Entretanto, esses resultados dependem, principalmente, de um particionamento *hardware/software* eficiente. A porção da aplicação escolhida para executar no recurso reconfigurável deve permitir a exploração do paralelismo inerente ao hardware, e o apropriado uso da hierarquia de memória.

Durante este trabalho, uma arquitetura otimizada para multiplicação de matrizes em sistemas reconfiguráveis de alto desempenho foi apresentada. Os resultados mostraram que uma análise prévia do problema permite o desenvolvimento de uma arquitetura que explora o paralelismo e elimina, através do reuso de dados e da utilização das memórias BRAMs internas ao FPGA, o gargalo no acesso aos dados.

Os resultados de simulação mostraram que com esta arquitetura, operando a 200MHz, podemos alcançar uma aceleração de aproximadamente 20 vezes quando comparado a uma implementação em software executada em um processador Xeon de 1,6 GHz com 4GB de memória RAM. Neste resultado não foi considerado o tempo de carga inicial dos dados nas memórias do RC100.

Os resultados também mostraram que, como o gargalo no acesso aos dados foi eliminado, o tempo de execução da aplicação é quase que inteiramente determinado pelo tempo de processamento das multiplicações. Desta forma, assumindo um FPGA hipotético onde 100 MACs podem ser instanciados, podemos alcançar uma aceleração de 200 vezes mantendo as mesmas condições de largura de banda.

8. Referências

[1] Laurenz Christian Buri, Studies of Classical HPC Problems on fine-grained and massively parallel computing enviromnment based on reconfigurable hardware, Msc. Thesis, Department of Microelectronics and Information Technology IMIT KTH, 2006.

[2] Ronald Scrofano, Jr. Accelerating Scientific Computing Applications with reconfigurable hardware, Ph.D.Thesis, Faculty of the Graduate School University of Southern California, 2006.

[3] Aiichiro Nakano. Class notes for CSCI 599: High performance scientific computing University of Southern California, Fall semester, 2003.

[4] D.C. Rapaport. The Art of Molecular Dynamics Simulation. Cambridge University Press, Cambridge, 2004.

[5] Maya B. Gokhale and Paul S. Graham. Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays. Springer, Dordrecht, The Netherlands, 2005.

[6] Ling Zhuo, Viktor K. Prasanna, Scalable and Modular Algorithms for Floating-Point Matrix

Multiplication on Reconfigurable Computing Systems, IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 18, No. 4, pp. 433-448, April 2007.

[7] Ling Zhuo , Viktor K. Prasanna, Scalable Hybrid Designs for Linear Algebra on Reconfigurable Computing Systems, Proceedings of the 12th International Conference on Parallel and Distributed Systems, p.87-95, July 12-15, 2006.

[8] L. Zhuo and V. K. Prasanna. Design Tradeoffs for BLAS Operations on Reconfigurable Hardware. In Proc. 34th Int'l Conf. Parallel Processing (ICPP'05), Oslo, Norway, June 2005.

[9] SRC Computers, Inc., http://www.srccomp.com/. Accesed in: March/2009.

[10] SGI RASC, www.sgi.com/products/rasc/. Accessed in: March/2009.

[11] Ling Zhuo , Viktor K. Prasanna, Scalable Hybrid Designs for Linear Algebra on Reconfigurable Computing Systems, Proceedings of the 12th International Conference on Parallel and Distributed Systems, p.87-95, July 12-15, 2006.

[12] L. Zhuo and V.K. Prasanna, "High-Performance Linear Algebra Operations on Reconfigurable Systems," Proc. Supercomputing 2005, IEEE CS Press, 2005, p. 2.

[13] R. Scrofano and V. K. Prasanna. Computing Lennard-Jones Potentials and Forces with Reconfigurable Hardware. In Proc. Int'l Conf. Eng. of Reconfigurable Systems and Algorithms (ERSA'04), pages 284–290, June 2004

[14] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Ed.. SIAM, 1994.

[15] R. Scrofano and V. K. Prasanna. Computing Lennard-Jones Potentials and Forces with Reconfigurable Hardware. In Proc. Int'l Conf. Eng. of Reconfigurable Systems and Algorithms (ERSA'04), pages 284–290, June 2004.

[16] NUMALink. http://www.nasi.com/sgi_NUMAlink.php. Accessed in: March/2009.

[17] Laurenz Christian Buri, Studies of Cassicals HPC Problems on fine-grained and massively parallel computing enviromnment based on reconfigurable hardware, Msc. Thesis, Department of Microelectronics and Information Technology IMIT KTH, 2006.

[18] Barros, A. C., Medeiros, V. W., Souza, V. L., Nascimento, P. S., Mazer, Â., Barbosa, J. P., Neves, B. P., Santos, I., and de Lima, M. E. 2008. Implementation of a double-precision multiplier accumulator with exception treatment to a dense matrix multiplier module in FPGA. In Proceedings of the 21st Annual Symposium on integrated Circuits and System Design (Gramado, Brazil, September 01 - 04, 2008). SBCCI '08. ACM, New York, NY, 40-45.

[19] SSP Stub Users Guide http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=bks&fname=/SGI_EndUser/RASC_UG/apb.html. Accessed in: March/2009.SSP Stub Users Guide http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=bks&fname=/SGI_EndUser/RASC_UG/apb.html. Accessed in: March/2009.

[20] Campbell, S. J. and Khatri, S. P. 2006. Resource and delay efficient matrix multiplication using newer FPGA devices. In Proceedings of the 16th ACM Great Lakes Symposium on VLSI (Philadelphia, PA, USA, April 30 - May 01, 2006).GLSVLSI '06. ACM, New York, NY, 308-311.

[21] Zhuo, L. 2007. Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on Reconfigurable Computing Systems. IEEE Trans. Parallel Distrib. Syst. 18, 4 (Apr. 2007), 433-448.

[22] Dou, Y., Vassiliadis, S., Kuzmanov, G. K., and Gaydadjiev, G. N. 2005. 64-bit floating-point FPGA matrix multiplication. In Proceedings of the 2005 ACM/SIGDA 13th international Symposium on Field-Programmable Gate Arrays (Monterey, California, USA, February 20 - 22, 2005). FPGA '05. ACM, New York, NY, 86-95.