DOI: http://dx.doi.org/10.29069/forscience.2013v1n1.e18

PARALELIZAÇÃO E COMPARAÇÃO DE MÉTODOS ITERATIVOS NA SOLUÇÃO DE SISTEMAS LINEARES GRANDES E ESPARSOS

Lauro Cássio Martins de Paula[1]

RESUMO

Apresenta-se neste trabalho uma comparação de desempenho computacional entre métodos iterativos utilizados para solução de sistemas lineares. O objetivo é mostrar que a utilização de processamento paralelo fornecido por uma *Graphics Processing Unit* (GPU) pode ser viável, por viabilizar a solução rápida de sistemas de equações lineares, para que sistemas grandes e esparsos possam ser solucionados em um espaço curto de tempo. Para a validação do trabalho, utilizou-se uma GPU, por meio da arquitetura *Compute Unified Device Architecture* (CUDA), e comparou-se o desempenho computacional dos métodos iterativos de Jacobi, Gauss-Seidel, BiCGStab e BiCGStab(2) paralelizado na solução de sistemas lineares de tamanhos variados. Foi possível observar uma aceleração significativa nos testes com o método paralelizado, que se acentua consideravelmente na medida em que os sistemas aumentam. Os resultados mostraram que a aplicação de processamento paralelo em um método robusto e eficiente, tal como o BiCGStab(2), se torna muitas vezes indispensável, para que simulações sejam realizadas com qualidade e em tempo não proibitivo.

Palavras-chave: CUDA. GPU. BiCGStab(2).

PARALLELIZATION AND COMPARISON OF ITERATIVE METHODS IN SOLVING LARGE AND SPARSE LINEAR SYSTEMS

ABSTRACT

This paper presents a computational performance comparison between some iterative methods used for linear systems solution. The goal is to show that the use of parallel processing provided by a Graphics Processing Unit (GPU) may be more feasible, for making possible the fast solution of linear equations systems in order that complex and sparse problems can be solved in a short time. To validate the paper a GPU through the NVIDIA's Compute Unified Device Architecture (CUDA) was employed and the computational performance was compared with Jacobi, Gauss-Seidel, BiCGStab iterative methods and BiCGStab(2) parallelized in the solution of linear systems of varying sizes. There was a significant acceleration in tests with the parallelized code, which increases considerably as much as systems increase. The results showed that the application of parallel processing in a robust and efficient method, as BiCGStab(2), it is often necessary for the simulations be performed with quality and in not prohibitive time.

Keywords: CUDA. GPU. BiCGStab(2).

[1] Mestrando em Ciência da Computação pela Universidade Federal de Goiás (UFG), bacharel em Ciência da Computação, com ênfase em Matemática Computacional, pela Pontifícia Universidade Católica de Goiás (PUC Goiás). E-mail: laurocassio21@gmail.com

1 INTRODUÇÃO

Existem na literatura diversos métodos utilizados para solução de sistemas lineares. Alguns desses métodos são considerados ótimos em relação ao custo computacional, dependendo do tamanho do sistema a ser solucionado. Para trabalhos reais da ciência, nos quais os sistemas lineares a serem resolvidos podem ser muito grandes, o processamento computacional pode durar vários dias e as diferenças de velocidade de solução dos métodos são, definitivamente, significantes. Com isso, a implementação de métodos robustos e eficientes é importante e muitas vezes indispensável, para que simulações sejam realizadas com qualidade e em tempo não proibitivo.

Um sistema linear é um conjunto finito de equações lineares aplicadas em um mesmo conjunto finito de variáveis. Geralmente, sistemas lineares esparsos e de grande porte aparecem como resultado da modelagem de vários problemas na ciência da computação e nas engenharias. Para solucionar tais sistemas, métodos iterativos de solução são mais indicados do que métodos exatos, por usarem menos memória e reduzirem erros de arredondamento do computador (FRANCO, 2006). Métodos iterativos realizam aproximações sucessivas, em cada iteração, para obter uma solução mais precisa para o sistema. Os Métodos clássicos como o de Jacobi (JUDICE et al., 1996) e de Gauss-Seidel (CLAUDIO; MARINS, 1989), apesar de sua fácil implementação, podem apresentar convergência lenta ou mesmo não convergirem para grandes sistemas (VERSTEEG; MALALASEKERA, 1995). Portanto, a pesquisa e a comparação de métodos iterativos a serem implementados em códigos computacionais podem ser consideradas tarefas importantes em várias áreas da ciência que envolvam a solução de grandes sistemas de equações lineares (PAULA et al., 2013a).

Trabalhos recentes têm utilizado *Graphics Processing Unit* (GPUs) para solucionar sistemas lineares. Por exemplo, Bowins (2012) apresentou uma comparação de desempenho computacional entre o método de Jacobi e o método Gradiente Bi-Conjugado Estabilizado (BiCGStab). Ambos os métodos foram implementados em uma versão sequencial e outra paralelizada, e foi possível concluir que, conforme o tamanho do sistema aumenta, a implemetação paralela supera a sequencial. Rodriguez et al. (2013) explorou o potencial computacional de múltiplas GPUs, utilizando OpenCL, com a finalidade de resolver sistemas de equações lineares de grande porte. Recentemente, Paula et al. (2013a) apresentou uma comparação de desempenho computacional entre as versões sequencial e paralelizada do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)) na solução de sistemas

lineares. Apesar de os resultados mostrarem que a versão paralelizada supera a implementação sequencial, não é realizada uma comparação com outros métodos. Nesse contexto, este trabalho utiliza a implementação paralela do método BiCGStab(2), proposta pelo autor deste trabalho em Paula et al. (2013a), e apresenta uma comparação com a implementação sequencial dos métodos de Jacobi, Gauss-Seidel e BiCGStab. O objetivo foi mostrar que o BiCGStab(2) paralelizado, do ponto de vista computacional, é uma implementação mais apropriada e por meio de sua utilização é possível viabilizar a solução rápida de sistemas lineares para que problemas cada vez mais complexos (sistemas maiores) possam ser solucionados em um tempo curto. Para atingir tal objetivo, utilizou-se uma GPU, por meio da arquitetura *Compute Unified Device Architecture* (CUDA), e comparou-se o desempenho computacional dos métodos na solução de sistemas lineares de tamanhos variados.

Este artigo está organizado da seguinte forma. Na Seção 2, são detalhados os métodos iterativos de Jacobi, Gauss-Seidel, BiCGstab e BiCGStab(2). Na Seção 3, descrevem-se os materiais e os métodos utilizados para se alcançar o objetivo do trabalho. Os resultados são discutidos na Seção 4. A Seção 5 traz as conclusões do trabalho.

2 MÉTODOS PARA SOLUÇÃO DE SISTEMAS LINEARES

Métodos iterativos podem ser classificados em dois grupos: métodos estacionários e não-estacionários. Nos estacionários, a mesma informação é usada em cada iteração. Como consequência, tais métodos são, normalmente, mais fáceis de implementar. Nos não-estacionários, a informação usada pode mudar a cada iteração. Esses métodos são considerados mais complexos para se implementar, porém fornecem uma convergência mais rápida para o sistema (PAULA et al., 2013a). Nesta seção são detalhados os principais aspectos sobre quatro métodos iterativos: Jacobi, Gauss-Seidel, BiCGStab e BiCGStab(2), sendo os dois primeiros considerados clássicos ainda largamente aplicados na solução de sistemas de equações lineares.

2.1 Jacobi

O método de Jacobi é um algoritmo utilizado para determinar a solução de sistemas de equações lineares com os maiores valores absolutos em cada linha e coluna dominados pelo elemento da sua diagonal (FRANCO, 2006). Tal método herdou o nome do matemático Carl

Gustav Jakob Jacobi. Uma das principais vantagens do método de Jacobi é o fato de ele ser simples e fácil de se implementar em um código computacional. Por outro lado, possui a desvantagem de não garantir o funcionamento em todos os casos. O método de Jacobi é considerado razoável para sistemas lineares pequenos e sua convergência tende a ficar muito lenta para sistemas grandes (JUDICE et al., 1996).

Considerando-se um sistema linear $Ax = b$ de ordem n, o comportamento do método de Jacobi ocorre da seguinte maneira: isola-se, em cada equação, uma variável e aplica-se a todas as outras o valor inicial fornecido inicialmente como, por exemplo, (0, 0, ..., 0). Inicialmente, isola-se x_1 na primeira equação, x_2 na segunda, ... e x_n na enésima equação:

$$\mathrm{x}_1 = \frac{1}{A_{11}}[b_1 - A_{12}x_2 - A_{13}x_3 - \ldots - A_{1n}x_n] \quad (1)$$

$$\mathrm{x}_1 = \frac{1}{A_{22}}[b_2 - A_{21}x_1 - A_{23}x_3 - \ldots - A_{2n}x_n] \quad (2)$$

.

.

.

$$\mathrm{x}_1 = \frac{1}{A_{nn}}\left[b_n - A_{n1}x_1 - A_{n2}x_2 - \ldots - A_{n(n-1)}x_{n-1}\right]. \quad (3)$$

A ideia básica do método consiste em fornecer um valor inicial às variáveis x_1^0, x_2^0, ..., x_n^0, colocar esses valores no lado direito das equações, obter x_1^1, x_2^1, ..., x_n^1, colocar esses novos valores nas equações, obter x_1^2, x_2^2, ..., x_n^2, e assim sucessivamente. O critério de parada pode ser um número máximo de iterações previamente definido.

O método garante convergência do sistema apenas quando um critério, chamado critério das linhas, é satisfeito (ANTON; BUSBY, 2006). Isso implica que o valor absoluto do termo diagonal na linha i deve ser maior do que a soma dos valores absolutos de todos os outros termos na mesma linha. Em outras palavras, isso equivale a dizer que, para o método garantir convergência, a matriz (A) dos coeficientes deverá ser diagonal dominante (PAULA et al., 2013a).

2.2 Gauss-Seidel

O método de Gauss-Seidel é um método iterativo utilizado para solução de sistemas lineares. Tal método possui esse nome em homenagem aos matemáticos Carl Friedrich Gauss

e Philipp Ludwig Seidel. O método de Gauss-Seidel pode ser considerado uma otimização do método de Jacobi. Também considerado um método clássico, ainda é bastante aplicado na solução de sistemas lineares. No método, os novos valores de x, em cada iteração, podem ser obtidos pela substituição dos valores anteriores de x no lado direito das equações (1), (2) e (3).

Os novos valores de x não são todos calculados simultaneamente. Primeiramente, x_1 é obtido. Logo após, x_2 é obtido, e assim por diante. Como os novos valores de x deveriam ser mais precisos do que seus predecessores, parece razoável esperar que uma maior precisão seja obtida se os novos valores de x forem utilizados assim que se tornarem disponíveis (ANTON; BUSBY, 2006). Devido a essa característica, nota-se que uma tentativa de aplicação de processamento paralelo em seu procedimento torna-se consideravelmente ineficaz. Isso ocorre devido à dependência de dados. Por exemplo, x_3 não poderá ser calculado antes de x_2 ser calculado, e x_2 não poderá ser calculado antes de x_1 ser calculado.

Assim como ocorre com o método de Jacobi, existem situações em que as iterações produzidas pelo método de Gauss-Seidel deixam de convergir para a solução do sistema (ANTON; BUSBY, 2006). No caso em que a matriz dos coeficientes é diagonal dominante, a taxa de convergência é determinada por quanto o lado esquerdo da equação (4) domina o lado direito (quanto mais dominante o lado esquerdo, mais rápida será a convergência) (ANTON; BUSBY, 2006):

$$|A_{ii}| > \sum_{j=1}^{n} |A_{ij}|, \forall\, i = 1, \ldots, n \; e \; j \neq i. \qquad (4)$$

2.3 BiCGStab

O método Gradiente Bi-conjugado Estabilizado (BiCGStab) foi desenvolvido por Van der Vorst (VORST, 1992). Segundo o *Institute for Scientific Information* (ISI), este trabalho foi o mais citado na área de Matemática Aplicada durante a década de 1990.

A figura 1 mostra a sequência de passos do BiCGStab. Algumas adaptações de nomenclatura foram feitas com relação ao algoritmo original, na busca de melhor entendimento do método. No algoritmo, as letras gregas representam escalares, letras minúsculas representam vetores expressos na forma matricial, as letras maiúsculas representam matrizes e os parênteses com vetores separados por vírgula representam produtos escalares entre os vetores.

1. $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
2. Escolha um vetor $\hat{\mathbf{r}}_0$ tal que $(\hat{\mathbf{r}}_0, \mathbf{r}_0) \neq 0$, como $\hat{\mathbf{r}}_0 = \mathbf{r}_0$
3. $\rho_0 = \alpha_0 = \omega_0 = 1$
4. $\mathbf{v}_0 = \mathbf{p}_0 = \mathbf{0}$
5. Para n = 1, 2, 3, ...
    6. $\rho_n = (\hat{\mathbf{r}}_0^T, \mathbf{r}_{n-1})$
    7. $\beta_n = (\rho_n/\rho_{n-1})(\alpha_{n-1}/\omega_{n-1})$
    8. $\mathbf{p}_n = \mathbf{r}_{n-1} + \beta_n(\mathbf{p}_{n-1} - \omega_{n-1}\mathbf{v}_{n-1})$
    9. $\mathbf{v}_n = \mathbf{A}\mathbf{p}_n$
    10. $\alpha_n = \rho_n/(\hat{\mathbf{r}}_0^T, \mathbf{v}_n)$
    11. $\mathbf{s}_n = \mathbf{r}_{n-1} - \alpha_n\mathbf{v}_n$
    12. $\mathbf{t}_n = \mathbf{A}\mathbf{s}_n$
    13. $\omega_n = (\mathbf{t}_n^T, \mathbf{s}_n)/(\mathbf{t}_n^T, \mathbf{t}_n)$
    14. $\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n\mathbf{p}_n + \omega_n\mathbf{s}_n$
    15. Se x_n for suficientemente preciso, Pare; senão
    16. $\mathbf{r}_n = \mathbf{s}_n - \omega_n\mathbf{t}_n$

Figura 1 - Algoritmo do método iterativo BiCGStab (PAULA et al., 2013)

Como mostra a figura 1, o BiCGStab é um método iterativo em que o número máximo de iterações pode ser previamente definido. A partir do passo 6 no algoritmo, cada passo é dependente do anterior, ou seja, dois passos não podem ser executados em paralelo. Em consequência disso, para uma possível paralelização do método, a exploração de paralelismo deve ocorrer em cada passo separadamente.

2.4 BiCGStab(2)

O método BiCGStab(2) foi desenvolvido por Van der Vorst e Sleijpen (VORST; SLEIJPEN, 1995). De acordo com seus criadores, tal método reúne as vantagens do BiCGStab, resultando, normalmente, em um método com garantia de convergência superior e adequado, por exemplo, para solução de sistemas lineares gerados na solução das equações diferenciais de escoamentos de fluidos.

A figura 2 representa o algoritmo do método BiCGStab(2). Assim como na figura 1, algumas adaptações de nomenclatura foram feitas com relação ao algoritmo original. No passo 38 do método, para que o vetor x_{i+2} seja suficientemente preciso, o maior valor referente à diferença entre os resultados de cada termo do vetor x em duas iterações consecutivas, dividida pelo resultado do termo na iteração atual, deverá ser menor do que uma dada precisão, por exemplo, $maior\left(\frac{x_i - (x_i - 1)}{x_i}\right) < 10^{-5}$.

1. $\mathbf{r_0} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$
2. Escolha um vetor arbitrário $\widehat{\mathbf{r_0}}$ tal que $(\mathbf{r_0}, \widehat{\mathbf{r_0}}) \neq 0$, como $\widehat{\mathbf{r_0}} = \mathbf{r_0}$
3. $\rho = \alpha = \omega_1 = \omega_2 = 1$
4. $\mathbf{w} = \mathbf{v} = \mathbf{p} = \mathbf{0}$
5. For $i = 0, 2, 4, 6, \ldots$
6. $\rho' = -\omega_2\, \rho$

Even Bi-CG step

7. $\rho = (\mathbf{r_i}, \widehat{\mathbf{r_0}})$
8. $\beta = \alpha\, \rho/\rho'$
9. $\rho' = \rho$
10. $\mathbf{p} = \mathbf{r_i} - \beta(\mathbf{p} - \omega_1 \cdot \mathbf{v} - \omega_2 \cdot \mathbf{w})$
11. $\mathbf{v} = \mathbf{A}\mathbf{p}$
12. $\gamma = (\mathbf{v}, \widehat{\mathbf{r_0}})$
13. $\alpha = \rho/\gamma$
14. $\mathbf{r} = \mathbf{r_i} - \alpha\, \mathbf{v}$
15. $\mathbf{s} = \mathbf{A}\mathbf{r}$
16. $\mathbf{x} = \mathbf{x_i} + \alpha\mathbf{p}$

Odd Bi-CG step

17. $\rho = (\mathbf{s}, \widehat{\mathbf{r_0}})$
18. $\beta = \alpha\, \rho/\rho'$
19. $\rho' = \rho$
20. $\mathbf{v} = \mathbf{s} - \beta\mathbf{v}$
21. $\mathbf{w} = \mathbf{A}\mathbf{v}$
22. $\gamma = (\mathbf{w}, \widehat{\mathbf{r_0}})$
23. $\alpha = \rho/\gamma$
24. $\mathbf{p} = \mathbf{r} - \beta\, \mathbf{p}$
25. $\mathbf{r} = \mathbf{r} - \alpha\mathbf{v}$
26. $\mathbf{s} = \mathbf{s} - \alpha\mathbf{w}$
27. $\mathbf{t} = \mathbf{A}\mathbf{s}$

GMRES(2)-part

28. $\omega_1 = (\mathbf{r}, \mathbf{s})$
29. $\mu = (\mathbf{s}, \mathbf{s})$
30. $\nu = (\mathbf{s}, \mathbf{t})$
31. $\tau = (\mathbf{t}, \mathbf{t})$
32. $\omega_2 = (\mathbf{r}, \mathbf{t})$
33. $\tau = \tau - \nu^2/\mu$
34. $\omega_2 = (\omega_2 - \nu \cdot \omega_1/\mu)/\tau$
35. $\omega_1 = (\omega_1 - \nu \cdot \omega_2)/\mu$
36. $\mathbf{x_{i+2}} = \mathbf{x} + \alpha\mathbf{p} + \omega_1\mathbf{r} + \omega_2\mathbf{s}$
37. $\mathbf{r_{i+2}} = \mathbf{r} - \omega_1\mathbf{s} - \omega_2\mathbf{t}$
38. Se $\mathbf{x_{i+2}}$ for suficientemente preciso: parar.
39. End For

Figura 2 - Algoritmo do método iterativo BiCGStab(2) (PAULA et al., 2013)

3 MATERIAL E MÉTODOS

A GPU foi inicialmente desenvolvida como uma tecnologia orientada à vazão, otimizada para cálculos de uso intensivo de dados, onde muitas operações idênticas podem ser realizadas em paralelo sobre diferentes dados. Diferente de uma *Central Processing Unit* (CPU) multicore atual, a GPU foi projetada para executar milhares de *threads* em paralelo (PAULA et. al, 2013b).

De forma correspondente ao *hardware*, modelos de programação tal como CUDA (NVIDIA CUDA, 2012) e OpenCL (TSUCHIYAMA et al., 2010) permitem que as aplicações sejam executadas mais facilmente na GPU. CUDA foi a primeira *Application Programming Interface* (API) a permitir que a GPU pudesse ser usada para uma ampla variedade de aplicações. CUDA pode ser definida em duas partes: *hardware* e *software*. Quanto ao *hardware*, a arquitetura básica da GPU consiste em um conjunto de multiprocessadores (SMs), cada um contendo vários processadores (SPs). Todos os SPs dentro de um SM executam as mesmas instruções sobre dados diferentes, conforme o modelo *Single Instruction Multiple Data* (SIMD). Quanto à parte de software, CUDA é construída com base em um conjunto de bibliotecas implementadas na linguagem C. Elas permitem explorar integralmente os recursos da GPU. Em CUDA, a CPU é chamada *host* e a GPU é chamada *device*. No *host*,

toda a parte sequencial do código é executada e as funções *kernel* são chamadas para serem executadas no *device*, onde todas as partes aritmeticamente intensivas do código podem ser executadas nos núcleos de processamento da GPU (NVIDIA CUDA, 2012).

Assim como o algoritmo do método BiCGStab, o algoritmo do método BiCGStab(2) apresenta uma forte dependência de dados. Logo, alguns passos do método não podem ser executados antes que outros sejam executados anteriormente. Como mostrado em Paula et al. (2013a), devido a essa restrição, apenas alguns passos do método foram paralelizados. Explorou-se paralelismo nas operações de multiplicação entre matriz e vetor, soma de vetores, subtração de vetores e multiplicação de vetor por escalar. Por exemplo, o passo 36 do algoritmo do BiCGStab(2) é dividido entre n *threads*, onde n é o tamanho do vetor x e para cada $0 \leq i <$ n uma *thread* executa a seguinte operação:

$$x_i = x_i + \alpha p_i + \omega_1 r_i + \omega_2 s_i. \qquad (5)$$

Portanto, cada elemento do vetor x é calculado em paralelo. A seguir, mostra-se um trecho do código utilizado na chamada da função *kernel* que implementa o passo 36 do algoritmo:

```
calculaVetorX<<<nBlocos, nThreadsPorBloco>>>(x, p, r, s, alpha, omega1, omega2, n);
```

Na chamada da função, *nBlocos* = *nThreadsPorBloco* = a raiz quadrada de n, onde n é o número de linhas do vetor x. Se a raiz quadrada de n não é um número inteiro, o teto desse número é obtido. Segue-se o algoritmo da função *calculaVetorX*. Na função, a variável $0 \leq id <$ n representa o identificador global de cada *thread*, que executa uma cópia da função independentemente.

```
__global__ void calculaVetorX(float *x, float *p, float *r, float *s, float alpha, float omega1, float omega2, int n) {
    int id = blockIdx.x * blockDim.x + threadIdx.x;
    if(id < n) {
        x[id] = x[id] + alpha * p[id] + omega1 * r[id] + omega2 * s[id];
    }
}
```

Neste trabalho, os resultados obtidos com o método BiCGStab(2) paralelizado foram confrontados com os de Jacobi, Gauss-Seidel e BiCGStab, com o intuito de verificar o ganho

computacional obtido com a implementação paralelizada. Para avaliar o ganho computacional obtido por meio da implementação em CUDA, registrou-se o tempo dispendido em cada iteração do algoritmo.

Todos os testes foram realizados em um computador com processador Intel Core i7 2600 (3,4GHz), 8 GB de memória RAM, com placa gráfica NVIDIA GeForce GTX 550Ti dispondo de 2 GB de memória e 192 processadores operando a 900MHz.

4 RESULTADOS E DISCUSSÃO

Os sistemas lineares solucionados neste trabalho foram gerados por meio de funções do software Matlab R2013a. A matriz dos coeficientes de cada sistema foi gerada de forma aleatória utilizando a função padrão *gallery('dorr', n)*, que retorna uma matriz quadrada, de dimensão n, esparsa e diagonal dominante. O vetor das incógnitas foi gerado de forma aleatória por meio da função padrão *randn(n, 1)*, que retorna um vetor de n linhas e 1 coluna. O vetor dos termos independentes foi gerado multiplicando-se a matriz A e o vetor x ($b = A * x$). Então, para cada um dos sistemas gerados, foi repassado para os métodos apenas a matriz A e o vetor b, que, após a tentativa de convergência do sistema, retornaram o vetor x.

O intuito deste artigo não é comparar as diferenças de solução entre o Matlab e os métodos, mas apenas utilizar o Matlab para gerar os sistemas aleatoriamente e comparar a velocidade de cálculo dos métodos de Jacobi, Gauss-Seidel, BiCGStab e BiCGStab(2) paralelizado na solução de diversos sistemas lineares.

O gráfico comparativo do tempo de processamento dos métodos é apresentado no gráfico 1:

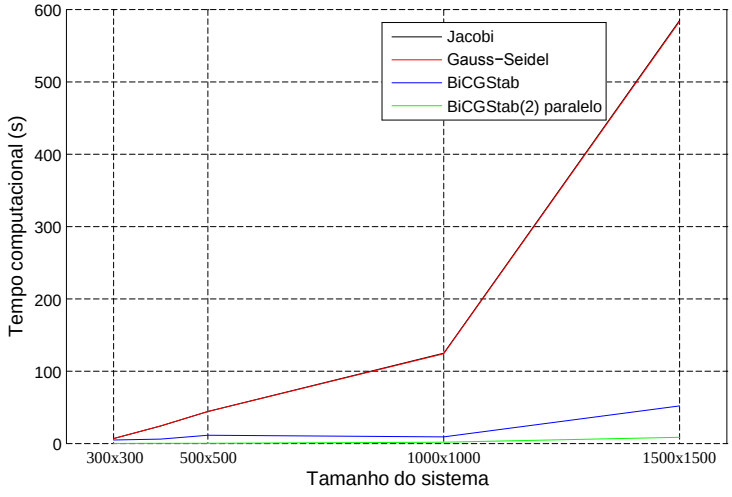

Gráfico 1 - Comparação da velocidade de cálculo para sistemas lineares de dimensão entre 300x300 e 1500x1500

4.1 Avaliação do ganho computacional obtido com o BiCGStab(2) paralelizado

Analisando o gráfico 1 nota-se a superioridade do método BiCGStab(2) paralelizado na solução dos sistemas tratados. É possível verificar que os métodos de Jacobi e Gauss-Seidel requerem um esforço computacional que aumenta de forma aproximadamente exponencial com o tamanho do sistema. Observa-se também que o tempo para o BiCGStab e BiCGStab(2) paralelizado também aumenta nesse caso, porém com uma taxa de crescimento menos expressiva. Em comparação de eficiência computacional, o BiCGStab(2) paralelizado se mostrou 52 vezes mais rápido que os métodos de Jacobi e Gauss-Seidel na solução dos sistemas. Já em comparação com o método BiCGStab, a implementação paralela foi 12 vezes mais rápida.

Para sistemas lineares com dimensão menor que aproximadamente 100x100, os outros métodos poderão se mostrar mais rápidos que o método paralelizado. Isso ocorre devido à existência de um *overhead* associado à paralelização das tarefas na GPU (PAULA et al., 2013b). Isso implica que o tamanho do sistema a ser solucionado na GPU deve ser levado em consideração. Fatores em relação ao tempo de acesso à memória podem influenciar no desempenho computacional. Por exemplo, o acesso à memória global da GPU, geralmente, apresenta uma alta latência e está sujeito a um acesso aglutinado aos dados em memória. Entretanto, observa-se uma aceleração significativa nos testes com o BiCGStab(2) paralelizado, que se acentua consideravelmente na medida em que os sistemas aumentam.

5 CONCLUSÃO

Foi utilizado neste trabalho um código computacional do algoritmo do método iterativo BiCGStab(2) para solução de sistemas lineares. O método foi implementado em uma versão paralelizada, utilizando a linguagem CUDA-C. Tal implementação foi proposta em Paula et al. (2013a), onde se explorou o uso de uma GPU para a paralelização de algumas etapas do algoritmo do BiCGStab(2) e mostrou-se que a versão paralelizada do método supera a implementação sequencial.

O objetivo deste trabalho foi utilizar o método BiCGStab(2) paralelizado para viabilizar a solução rápida de sistemas lineares e comparar o desempenho computacional com outros métodos na solução de sistemas de tamanhos variados. Para os sistemas lineares avaliados neste artigo, constatou-se uma superioridade da versão paralelizada do BiCGStab(2) em relação ao tempo computacional gasto no cálculo de cada sistema. Foi possível obter

ganhos (*speedup*) de desempenho para diferentes tamanhos de sistema linear. Observou-se que a média de *speedup*, quando comparado com os métodos de Jacobi e Gauss-Seidel, foi de aproximadamente 52. Quando comparado com o método BiCGStab, a média de *speedup* foi de aproximadamente 12. Portanto, concluiu-se que, em comparação com os métodos de Jacobi, Gauss-Seidel e BiCGStab, a versão paralelizada do BiCGStab(2) seria uma implementação mais apropriada, do ponto de vista computacional, desde que o tamanho do sistema empregado seja suficientemente grande para justificar o *overhead* gasto na paralelização das tarefas na GPU.

Trabalhos futuros poderão solucionar sistemas lineares ainda maiores. Os sistemas gerados nas simulações de escoamentos de fluidos, amplamente estudados na Dinâmica dos Fluidos Computacional, poderão ser utilizados. Existe também a possibilidade de se aplicar uma otimização no código computacional da versão paralelizada do método BiCGStab(2). Uma possível otimização seria a utilização de uma técnica conhecida como *Persistent Threads*, na qual uma única função *kernel* é executada na GPU durante todo o processamento do algoritmo. Espera-se que nesse caso os ganhos computacionais proporcionados sejam ainda mais expressivos. Ainda, alternativas à arquitetura CUDA, tal como OpenCL, poderão ser investigadas para a realização de estudos comparativos.

REFERÊNCIAS

ANTON, H; BUSBY, R. C. **Álgebra linear contemporânea.** Porto Alegre: Bookman, 2006.

BOWINS, E. C. A comparison of sequential and GPU implementations of iterative methods to compute reachability probabilities. In: WORKSHOP ON GRAPH INSPECTION AND TRAVERSAL ENGINEERING, I., 2012, Tallim. **Proceedings...** Tallinn [s.n.], 2012. p. 20-34.

CLAUDIO, D. M; MARINS, J. M. **Cálculo numérico computacional**. São Paulo: Atlas, 1989.

FRANCO, N. B. **Cálculo numérico.** São Paulo: Pearson Prentice Hall, 2006.

GAIOSO, R. R.; PAULA, L. C. M. Paralelização do algoritmo Floyd-Warshall usando GPU. In: SIMPÓSIO EM SISTEMAS COMPUTACIONAIS. XIV., 2013. **Anais...** [S.l.: s.n.], 2013.

JUDICE, J. J. et. al. **Sistemas de equações lineares**. Coimbra: Departamento de Matemática da Universidade de Coimbra, 1996.

KINCAID, D.; WARD, C**. Mathematics of scientific computing**. [S.l.]: Pacific Grove, 1996.

NVIDIA CUDA. **NVIDIA CUDA C Programming best practices guide**. [S.l.]: NVIDIA Corporation, 2009.

NVIDIA CUDA. **NVIDIA CUDA C Programming Guide 4.2**. [S.l.]: NVIDIA Corporation, 2012.

PAULA, L. C. M et al. Aplicação de processamento paralelo em método iterativo para solução de sistemas lineares. In: ENCONTRO ANUAL DE COMPUTAÇÃO (ENACOMP), X., 2013a, Catalão. **Anais...** Catalão: UFG, 2013a. p. 129-136.

PAULA, L. C. M. et al. Partial parallelization of the successive projections algorithm using compute unified device architecture. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS (PDPTA). 19., 2013b, Las Vegas. **Proceedings...** Las Vegas: [s.n.], 2013b.

RODRIGUEZ, N. R. et al. **Resolução de sistemas de equações lineares de grande porte em clusters multi-GPU utilizando o método do gradiente conjugado em OpenCL**. Rio de Janeiro: Departamento de Informática da PUC-Rio, 2013.

TSUCHIYAMA, R. et. al. C. **The openCL programming book**. [S.l.] Fixstars, 2010.

VERSTEEG, H. K; MALALASEKERA, W. **An introduction to computational fluid dynamics**: the finite volume method. Londres: Longman Scientific & Technical, 1995.

VORST, H. A. V. Bi-CGStab: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. **SIAM Journal of Scientific and Statistical Computing**, v. 13, p. 631-644, 1992.

VORST, H. A. V; SLEIJPEN, G. L. G. Hybrid Bi-Conjugate Gradient Methods for CFD Problems. In: HAFEZ, M.; OSHIMA K. **Computational fluid dynamics REVIEW 1995**. Chicester: Wiley, 1995. p 457-476.

Recebido em: 11/07/2013

Aprovado em: 30/09/2013