IMPLEMENTAÇÃO PARALELA DO MÉTODO BICGSTAB(2) EM GPU USANDO CUDA E MATLAB PARA SOLUÇÃO DE SISTEMAS LINEARES

PARALLEL IMPLEMENTATION OF THE BICGSTAB(2) METHOD IN GPU USING CUDA AND MATLAB FOR SOLUTION OF LINEAR SYSTEMS

Lauro Cássio Martins de Paula
Instituto de Informática - Universidade Federal de Goiás/Brazil
laurocassio21@gmail.com

Resumo: **Este artigo apresenta uma implementação paralela do método iterativo Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)) em *Graphics Processing Unit* (GPU) para solução de sistemas lineares grandes e esparsos. Tal implementação faz uso da integração CUDA-Matlab, em que as operações do método são executadas nos núcleos de uma GPU por meio de funções padrão do Matlab. O objetivo é mostrar que a exploração de paralelismo utilizando essa nova tecnologia pode fornecer um desempenho computacional significante. Para a validação do trabalho, comparou-se a implementação proposta com uma implementação sequencial e outra paralelizada do BiCGStab(2) nas linguagens C e CUDA-C, respectivamente. Os resultados mostraram que a implementação proposta é mais eficiente e pode ser indispensável para que simulações sejam realizadas com qualidade e em um tempo hábil. Os ganhos de eficiência computacional foram de, respectivamente, 76x e 6x em relação à implementação em C e CUDA-C.**

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

Abstract: **This paper presents a parallel implementation of the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) iterative method in Graphics Processing Unit (GPU) for solution of large and sparse linear systems. This implementation uses the CUDA-Matlab integration, in which the method operations are performed in a GPU cores using Matlab built-in functions. The goal is to show that the exploitation of parallelism by using this new technology can provide a significant computational performance. For the validation of the work we compared the proposed implementation with a BiCGStab(2) sequential and parallelized implementation in the C and CUDA-C languages, respectively. The results showed that the proposed implementation is more efficient and can be indispensable for simulations being carried out with quality and in a timely manner. The gains in computational efficiency were, respectively, 76x and 6x compared to the implementation in C and CUDA-C.**

Keyword: Matlab; GPU; CUDA; BiCGStab(2).

I. INTRODUÇÃO

Diversos métodos podem ser utilizados para solução de sistemas lineares. Alguns deles são considerados ótimos em relação ao custo computacional. Entretanto, dependendo do tamanho do sistema a ser solucionado, o desempenho computacional pode ser afetado. Em alguns casos, em que os sistemas lineares a serem solucionados podem ser muito grandes, o processamento computacional pode durar vários dias e as diferenças de velocidade de solução dos métodos são significantes. Consequentemente, a implementação de métodos robustos e eficientes se torna importante e muitas vezes indispensável para que simulações sejam realizadas com qualidade e em um espaço curto de tempo [16].

Um sistema linear pode ser considerado como um conjunto finito de equações lineares aplicadas em um conjunto finito de variáveis. Sistemas lineares esparsos e de grande porte podem aparecer como resultado da modelagem de vários problemas na ciência da computação e nas engenharias. Para solucionar tais sistemas, métodos iterativos de solução podem ser considerados mais indicados e eficientes do que métodos exato [18]. Os métodos iterativos podem utilizar menos memória e reduzir erros de arredondamento nas operações do computador [1]. Tais métodos realizam aproximações sucessivas, em cada iteração, para obter uma solução mais precisa para o sistema. Os métodos iterativos clássicos como o de Jacobi [2] e de Gauss-Seidel [3] são considerados fáceis para se implementar e utilizar. No entanto, apesar dessa característica, ambos podem apresentar uma convergência lenta ou mesmo não convergirem para grandes sistemas[4]. Outra desvantagem é que, quando a matriz dos coeficientes não é quadrada (número de linhas igual ao número de colunas), esses dois métodos podem não garantir uma convergência para o sistema linear [9]. Como consequência, a pesquisa e a implementação de métodos computacionais podem ser consideradas tarefas importantes em várias áreas da ciência, principalmente as que envolvem a solução de sistemas de equações lineares de grande porte [16].

Diversos trabalhos têm utilizado os recursos computacionais das *Graphics Processing Units* (GPU) para solucionar sistemas lineares grandes e esparsos. Por exemplo, Bowins [5] apresentou uma

comparação de desempenho computacional entre o método de Jacobi e o método Gradiente Bi-Conjugado Estabilizado (BiCGStab). Em seu trabalho, ambos os métodos foram implementados em duas versões: uma sequencial e outra paralelizada. Com base nos resultados obtidos, ele mostrou que, conforme o tamanho do sistema aumenta, a implementação paralela supera a sequencial em termos de eficiência computacional. Rodriguez *et al.*[6] explorou o potencial computacional de múltiplas GPUs, utilizando *Open Computing Language* (OpenCL), com a finalidade de resolver sistemas de equações lineares de grande porte. Paula *et al.* [16] apresentaram uma paralelização do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)) para solução de sistemas lineares, utilizando *Compute Unified Device Architecture* (CUDA), e compararam o desempenho computacional entre as versões sequencial e paralelizada do método. Eles mostraram que, do ponto de vista computacional, a versão paralelizada do método BiCGStab(2) é mais eficiente. Paula [18] apresentou uma comparação de desempenho computacional entre o BiCGStab(2) proposto em [16] e alguns métodos iterativos na solução de sistemas lineares grandes e esparsos, evidenciando que o BiCGStab(2) paralelizado também supera tais métodos em termos de eficiência computacional.

Este trabalho apresenta uma implementação paralela do método BiCGStab(2), que utiliza a tecnologia CUDA-Matlab em GPU, para solução de sistemas lineares. O objetivo foi mostrar que a implementação proposta pode ser mais apropriada e, por meio de sua utilização, é possível viabilizar a solução eficaz de sistemas lineares grandes e esparsos para que sistemas cada vez mais complexos (maiores) possam ser solucionados em um tempo hábil. Para atingir esse objetivo, realizou-se uma comparação com a implementação do método BiCGStab(2) apresentada por Paula *et al.* [16] na solução de sistemas lineares de tamanhos variados. Os resultados obtidos mostraram que o tempo computacional pode ser reduzido significativamente com a implementação proposta neste artigo. Foi possível obter ganhos de eficiência computacional de 76x e 6x, respectivamente, em relação à implementação sequencial e paralelizada proposta em [16].

O restante deste artigo está organizado da seguinte forma. Na Seção 2 é detalhado o método iterativo BiCGStab(2). A Seção 3 descreve a arquitetura CUDA e sua integração com o Matlab. Os materiais e os métodos utilizados para se alcançar o objetivo do trabalho são descritos na Seção 4. Os resultados são apresentados e discutidos na Seção 5. Por fim, a Seção 6 traz as conclusões do trabalho.

II. MÉTODO BICGSTAB(2)

A solução de um sistema de equações lineares $Ax = b$, onde A_{nxn}é a matriz dos coeficientes e b_{nx1} o vetor dos termos independentes, pode exigir um esforço computacional enorme, principalmente quando A é muito grande. Por exemplo, para solucionar um sistema linear, pode-se utilizar algum método iterativo.Os métodos iterativos realizam aproximações sucessivas, em cada iteração, para obter uma solução mais precisa e são recomendados para sistemas lineares grandes com matrizes esparsas [8].

Os métodos iterativos são classificados em dois grupos: os métodos estacionários e os não-estacionários [16]. Os métodos estacionários utilizam a mesma informação em cada iteração, ou seja, os resultados de uma iteração são utilizados pelas próximas iterações. Nos métodos não-estacionários, a informação utilizada pode mudar a cada iteração. Os métodos não-estacionários são mais difíceis de implementar, porém podem fornecer uma convergência mais rápida para o sistema e se mostram mais adequados mesmo quando a matriz dos coeficientes é densa (não-esparsa) [11].

O Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)) é um método iterativo não-estacionário desenvolvido por van der Vorst e Sleijpen [12]. Tal método reúne as vantagens do BiCGStab e do método *Generalized Minimum Residual* (GMRES), o qual tem a propriedade de minimizar, a cada passo, a norma do vetor residual sobre um subespaço Krylov [29]. Com isso, o BiCGStab(2) é considerado um método robusto e com garantia de convergência superior à do BiCGStab, adequado para solução de sistemas

lineares gerados na solução das equações diferenciais de escoamentos de fluidos [18].

A Figura 1 representa o algoritmo do método BiCGStab(2). Algumas adaptações de nomenclatura foram feitas com relação ao algoritmo original. 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.

No passo 38, 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 como, 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. $\widehat{\mathbf{r_0}} = \mathbf{r_0}$
3. $\rho = \alpha = \omega_1 = \omega_2 = 1$
4. $\mathbf{w} = \mathbf{v} = \mathbf{p} = \mathbf{0}$
5. **para** i = 0, 2, 4, 6, ... **faça**
6. $\rho' = -\omega_2\,\rho$

Passo par BiCGStab

7. $\rho = (\mathbf{r_i}, \widehat{\mathbf{r_0}})$
8. $\beta = \alpha\,\rho/\rho'$
9. $\rho' = \rho$
10. $\mathbf{p} = \mathbf{r_i} - \beta(\mathbf{p} - \omega_1\mathbf{v} - \omega_2\mathbf{w})$
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. $\mathbf{x} = \mathbf{x_i} + \alpha\mathbf{p}$ $\omega_2\mathbf{s}$

Passo ímpar

17. $\rho = (\mathbf{s}, \widehat{\mathbf{r_0}})$
18. $\beta = \alpha\,\rho/\rho'$
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. $\mathbf{t} = \mathbf{A}\mathbf{s}$

Etapa GMRES

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. $\omega_2 = (\omega_2 - \nu\omega_1/\mu)/\tau$
35. $\omega_1 = (\omega_1 - \nu\omega_2)/\mu$
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 preciso, parar

Figura 1 - Algoritmo do método iterativo BiCGStab(2) [16].

III. CUDA

Compute Unified Device Architecture (CUDA) foi a primeira arquitetura e interface para programação de aplicação (API), criada pela NVIDIA® em 2006, a permitir que a GPU pudesse ser utilizada para uma ampla variedade de aplicações [10]. CUDA é suportada por todas as placas gráficas da NVIDIA®, que são extremamente paralelas, possuindo muitos núcleos com diversas memórias *cache* e uma memória compartilhada por todos os núcleos. O código em CUDA é uma extensão da linguagem computacional C (CUDA-C), onde algumas palavras-chave são utilizadas para rotular as funções paralelas (*kernels*) e suas estruturas de dados [10].

Desde o seu surgimento, diversos trabalhos têm utilizado CUDA para a paralelização de vários tipos de problemas. Por exemplo, Yldirim e Ozdogan [20] apresentaram um algoritmo como uma abordagem de agrupamento baseado em transformada *wavelet* para paralelização em GPU usando CUDA-C. Fabris e Krohling [21] propuseram um algoritmo de evolução implementado em CUDA-C para solução de problemas de otimização.Atasoy *et al.* [22] apresentaram um método de eliminação implementado em CUDA-C usando Gauss-Jordan para solução de sistemas de equações lineares. Paula *et al.* [16] utilizaram CUDA-C para paralelizar o método BiCGStab(2), utilizado para solução de sistemas lineares. Paula *et al.* [17] propuseram uma estratégia de paralelização para a fase 2 do Algoritmo das Projeções Sucessivas utilizando CUDA-C. Gaioso *et al.* [15], utilizando CUDA-C, apresentaram uma paralelização para o algoritmo Floyd-Warshall, utilizado para encontrar os caminhos mínimos entre todos os pares de vértices de um grafo.

Com o intuito de ajudar os programadores, a MathWorks® desenvolveu um *plugin* capaz de fazer a integração entre CUDA e Matlab. Fazer uso do Matlab para computação em GPU pode permitir que aplicações sejam aceleradas mais facilmente. As GPUs podem ser utilizadas com Matlab por meio do *Parallel Computing Toolbox* (PCT). O PCT fornece uma maneira eficiente para acelerar códigos na linguagem Matlab, executando-os em uma GPU. Para isso, o programador deve alterar o tipo de dado para entrada de uma função para utilizar os comandos (funções) do Matlab que foram sobrecarregados (*GPUArray*). Por meio da função *GPUArray* é possível alocar dados na memória da GPU e fazer chamadas a várias funções do Matlab, que são executadas nos núcleos de processamento da GPU. Além disso, os desenvolvedores podem fazer uso da interface *CUDAKernel* no PCT para

integrar seus códigos em CUDA-C com o Matlab [23].

O desenvolvimento de aplicações a serem executadas na GPU utilizando o PCT é, geralmente, mais fácil e rápido do que utilizar a linguagem CUDA-C [24]. De acordo com Little e Moler [25], isso ocorre porque os aspectos de exploração de paralelismo são realizados implicitamente pelo próprio PCT, livrando o programador de muitas inconveniências. Entretanto, a organização e o número de *threads* a serem executadas nos núcleos da GPU não podem ser gerenciados manualmente pelo programador. Ainda, é importante ressaltar que, para poder ser utilizado, o PCT requer uma placa gráfica da NVIDIA®.

Após a integração CUDA-Matlab, alguns trabalhos têm utilizado essa tecnologia. Por exemplo, a NVIDIA® [26] lançou um livro que demonstra como programas desenvolvidos em Matlab podem ser acelerados usando suas GPUs. Simek e Asn [27] apresentaram uma implementação em Matlab com CUDA para compressão de imagens médicas. Kong *et al.* [28] aceleraram algumas funções do Matlab para processamento de imagens em GPUs. Reese e Zaranek [23] desenvolveram um manual de programação em GPUs usando Matlab. Little e Moler [25] mostraram vários detalhes de como realizar computações do Matlab em GPUs usando CUDA. Mais recentemente, Liu *et al.* [24] apresentaram uma pesquisa e comparação de programação usando GPUs em Matlab.

IV. MATERIAIS 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 [17]. Diferentemente de uma *Central Processing Unit* (CPU), a qual executa apenas algumas *threads* em paralelo, a GPU foi projetada para executar milhares delas.

Como citado anteriormente, pode-se explorar paralelismo em GPUs utilizando o *plugin* PCT, que fornece uma forma eficiente para acelerar códigos na linguagem Matlab invocando funções que foram sobrecarregadas para serem executadas nos núcleos de uma GPU da NVIDIA®. Sendo assim, este artigo apresenta uma implementação do método BiCGStab(2) em Matlab, que utiliza essa tecnologia. A implementação proposta é análoga ao algoritmo na Figura 1. Inicialmente, os dados são transferidos para a memória da GPU. Logo após, o método inicia sua execução e todas as operações são executadas nos núcleos de processamento da GPU pelas *threads*, que são criadas e gerenciadas implicitamente pelo PCT.

Assim como em [18] e [16], todos os sistemas lineares utilizados neste trabalho foram gerados utilizando-se funções padrão do software Matlab (versão R2013a). A matriz dos coeficientes de cada sistema foi gerada de forma aleatória utilizando a função *gallery('dorr', n)*, que retorna uma matriz quadrada, de dimensão *n*, esparsa e diagonal dominante. A característica diagonal dominante indica que a soma de todos os elementos em uma linha não é maior que o elemento da diagonal principal da matriz. O vetor das incógnitas foi gerado de forma aleatória por meio da função *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*. Para cada um dos sistemas gerados, foi repassado para o método BiCGStab(2) apenas a matriz *A* e o vetor *b*, que, após a tentativa de convergência do sistema, retornou o vetor *x*.

Para avaliar o ganho computacional obtido por meio da implementação paralelizada do método, registrou-se o tempo gasto em cada iteração do algoritmo do BiCGStab(2) [16]. Todos os testes foram realizados em um computador com processador Intel Core i7 2600 3,4GHz, 8 GB de memória RAM, com placa de vídeo NVIDIA® GeForce GTX 550Ti, dispondo de 2 GB de memória e 192 núcleos de processamento operando a 900MHz.

V. RESULTADOS E DISCUSSÃO

Os resultados obtidos com o método BiCGStab(2) paralelizado foram confrontados com sua implementação sequencial, com o intuito de verificar o ganho computacional obtido com a implementação paralelizada. Além disso, realizou-se uma comparação com as implementações

(sequencial e paralelizada) do BiCGStab(2) propostas por Paula *et al.*[16].

Os gráficos comparativos do tempo de processamento (em segundos) dos diversos sistemas lineares resolvidos com o método BiCGStab(2) (sequencial e paralelizado) em Matlab são apresentados nas Figuras 2 e 3.

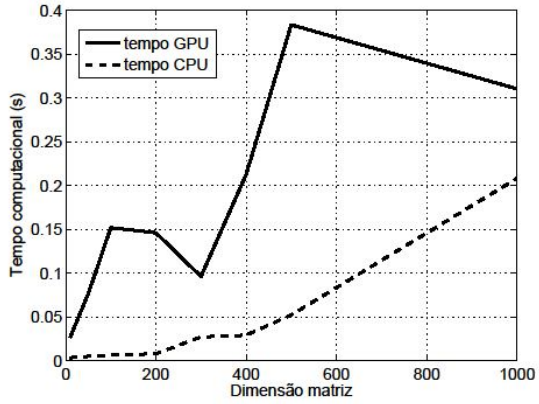

Figura 2 - Comparação da velocidade de cálculo para sistemas com dimensão entre 10 e 1000.

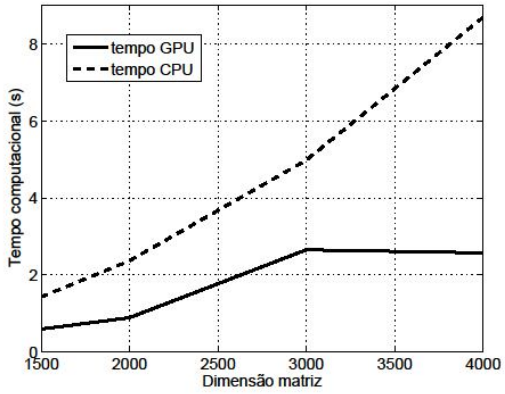

Figura 3 - Comparação da velocidade de cálculo para sistemas com dimensão entre 1500 e 4000.

A Figura 2 mostra que a implementação sequencial pode ser mais eficiente para sistemas lineares com dimensão entre 10 e 1000. Isso ocorre devido ao fato de o algoritmo do método conter operações intrinsecamente sequenciais. Por exemplo, os produtos escalares executados sequencialmente na CPU, dependendo da dimensão do sistema, podem apresentar um tempo computacional significativamente reduzido em comparação ao tempo da mesma execução nos núcleos da GPU. Da mesma forma, as operações entre os escalares (passos 8, 13 e 34, por exemplo) não podem ser divididas entre múltiplas *threads* e, consequentemente, isso pode resultar em um baixo desempenho quando executadas por uma única *thread* na GPU. Além disso, devido à existência de um *overhead* associado à paralelização das tarefas na GPU, o tamanho do sistema a ser solucionado deve ser levado em consideração [13], [16], [17].

Por outro lado, a Figura 3 mostra que, para sistemas com dimensão maior que 1500, o BiCGStab(2) paralelizado supera a implementação sequencial. Nesse caso, em comparação de eficiência computacional, o ganho de *speedup* obtido foi de aproximadamente 2,59x. Logo, a implementação que utiliza a GPU seria uma implementação mais apropriada desde que o tamanho do sistema empregado seja maior que 1500x1500.

A Figura 4 mostra uma comparação entre a implementação sequencial proposta e a implementação sequencial proposta por Paula *et al.*[16]. O BiCGStab(2) implementado em Matlab se mostra muito superior em relação à mesma implementação na linguagem C. Observa-se que o tempo para a implementação proposta por Paula *et al.* [16] requer um esforço computacional que aumenta de forma aproximadamente exponencial com o tamanho do sistema, enquanto que o tempo para a implementação em Matlab é menos acentuado. O ganho de *speedup* proporcionado pela implementação sequencial em Matlab foi de aproximadamente 76,75x. Consequentemente, a utilização da implementação do método em Matlab pode proporcionar um ganho de desempenho computacional mais significativo.

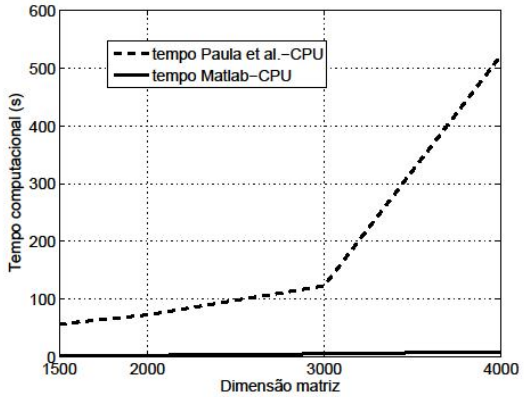

Figura 4 - Comparação da velocidade de cálculo para sistemas com dimensão entre 1500 e 4000 entre as implementações sequenciais do BiCGStab(2) em Matlab e em C.

A Figura 5 mostra uma comparação entre a implementação paralelizada proposta e a implementação paralelizada proposta por Paula *et al.*[16]. Assim como no caso anterior, nota-se a superioridade do BiCGStab(2) paralelizado, utilizando a integração CUDA-Matlab, na solução dos sistemas tratados. É possível observar que o tempo para a implementação em CUDA-C também exige um esforço computacional de forma aproximadamente exponencial na medida em que a dimensão do sistema aumenta. Nesse caso, o *speedup* obtido foi de aproximadamente 6,12x. Por isso, em comparação com a implementação paralelizada proposta por Paula *et al.*[16], o BiCGStab(2) paralelizado em Matlab pode ser uma escolha mais apropriada do ponto de vista computacional.

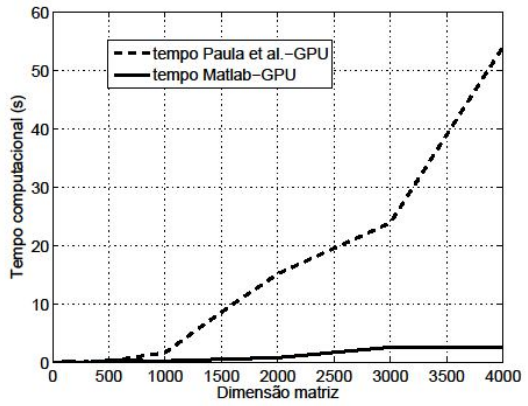

Figura 5 - Comparação da velocidade de cálculo para sistemas com dimensão entre 1500 e 4000 entre as implementações sequenciais do BiCGStab(2) em Matlab e em CUDA-C.

VI. CONCLUSÃO

A solução de sistemas de equações lineares está envolvida em diversas áreas da ciência e das engenharias. O esforço computacional exigido para a solução dos sistemas lineares aumenta de forma proporcional com o tamanho do sistema. Para casos em que os sistemas a serem resolvidos são muito grandes, o tempo de processamento pode durar muitas horas, ou até mesmo dias, e as diferenças de velocidade de solução dos métodos utilizados são, definitivamente, significantes. Consequentemente, a utilização de métodos robustos se torna uma tarefa importante nas áreas que envolvem a solução de sistemas lineares de grande porte. Além do mais, a exploração de paralelismo em tais métodos pode ser um fator importante para uma redução significante do tempo computacional gasto na solução dos sistemas tratados.

Nesse contexto, foi implementado e utilizado neste trabalho um código computacional em Matlab do método iterativo BiCGStab(2) para solução de sistemas lineares grandes e esparsos. O método foi implementado em uma versão inteiramente sequencial, bem como em uma versão paralelizada utilizando uma GPU por meio da integração CUDA-Matlab. O objetivo deste artigo foi apresentar uma nova implementação do BiCGStab(2) para viabilizar a solução rápida de sistemas lineares e comparar o desempenho computacional com a implementação sequencial. Adicionalmente, realizou-se uma comparação com a implementação sequencial e paralelizada proposta por Paula *et al.*[16].

Para os sistemas aqui avaliados, constatou-se uma superioridade da implementação paralelizada com CUDA-Matlab em relação ao tempo computacional gasto no cálculo de cada sistema. Foi possível obter um ganho de *speedup* em torno de 76x e 6x em comparação com a implementação sequencial e paralelizada apresentada em [16], respectivamente. Em comparação com a implementação sequencial em Matlab, o BiCGStab(2) paralelizado se mostrou superior apenas para sistemas com dimensão maior que 1500 e o *speedup* foi de aproximadamente 2,5x. Portanto, foi possível concluir que a implementação do método que executa na GPU, em comparação com as implementações propostas por Paula *et al.*[16], seria uma implementação mais adequada e apropriada para a obtenção de um desempenho computacional significativo.

Os trabalhos futuros que optarem por seguir nessa mesma linha de pesquisa poderão solucionar sistemas lineares com dimensões maiores que as deste trabalho.Os sistemas gerados nas simulações de escoamentos de fluidos em problemas estudados pela Dinâmica dos Fluidos Computacional poderão ser solucionados. Técnicas para uma exploração eficiente de paralelismo nas operações de produtos escalares entre vetores também poderão ser aplicadas na tentativa de aumentar ainda mais o desempenho computacional. Ainda, alternativas à

integração CUDA-Matlab, como OpenCL, poderão ser investigadas para uma realização de estudos comparativos.

.

REFERÊNCIAS

[1] N. B. Franco. Cálculo Numérico. São Paulo: Pearson Prentice Hall, 2006.

[2] J. J. Judice *et al*., Sistemas de Equações Lineares. Departamento de Matemática da Universidade de Coimbra, 1996.

[3] D. M. Claudio, and J. M. Marins, Cálculo Numérico Computacional. São Paulo: Atlas, 1989.

[4] H. K. Versteeg, and W. Malalasekera, An introduction to computational fluid dynamics: the finite volume method. Londres: Longman Scientific & Technical, 1995.

[5] E. C. Bowins, A comparison of sequential and GPU implementations of iterative methods to compute reachability probabilities. In: First workshop on graph inspection and traversal engineering, 2012, pp. 20-34.

[6] N. R. Rodrigues *et al.*, Resolução de sistemas de equações lineares de grande porte em clusters multi-GPU utilizando o método do gradiente conjugado em OpenCL. Dissertação de Mestrado. 2013.

[7] B. Goerl *et al.*, Bibliotecas e Ferramentas para Computação Numérica de alto desempenho. In: Escola Regional de Alto Desempenho (ERAD). 2011.

[8] C. R. Milani. Computação verificada aplicada a resolução de sistemas lineares intervalares densos em arquiteturas multicore. Dissertação de Mestrado. 2010.

[9] H. Anton, and R. C. Busby. Álgebra linear contemporânea. Bookman, 2006.

[10] CUDA. Nvidia CUDA C Programming Guide. Nvidia Corporation, 2012.

[11] H. A. Vorst. A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM, v. 13, 1992, pp. 631-644.

[12] H. A. Vorst *et al.* Hybrid Bi-Conjugate Gradient Methods for CFD problems. Dynamic review, 1995.

[13] CUDA. Nvidia CUDA C Best practices guide. Nvidia corporation, 2009.

[14] D. Kincaid, and C. Ward. Mathematics of Scientific Programming. Pacific Grove, 1996.

[15] R. A. Gaioso, and L. C. M. Paula *et al.* Paralelização do Algoritmo Floyd-Warshall usando GPU. In: WSCAD-SSC, 2013, pp. 19-25.

[16] L. C. M. Paula, L. B. S. Souza, L. B. S. Souza, and W. S. Martins, Aplicação de Processamento Paralelo em Método Iterativo para Solução de Sistemas Lineares. In: X Encontro Anual de Computação, 2013, pp. 129-136.

[17] L. C. M. Paula, A. S. Soares, T. W. Soares, W. S. Martins, A. R. G. Filho, and C. J. Coelho. Partial Parallelization of the Successive Projections Algorithm using Compute Unified Device Architecture. In: International Conference on Parallel and Distributed Processing Techniques and Applications, 2013, pp. 737-741.

[18] L. C. M. Paula. Paralelização e Comparação de Métodos Iterativos na Solução de Sistemas Lineares Grandes e Esparsos. ForScience: Revista Científica do IFMG, v. 1, p. 1-12, 2013.

[19] L. C. M. Paula, A. S. Soares, T. W. Soares, A. C. B. Delbem, C. J. Coelho, and A. R. G. Filho. Parallelization of a Modified Firefly Algorithm using GPU for Variable Selection in a Multivariate Calibration Problem. International Journal of Natural Computing Research, (In press), 2014.

[20] A. Yldirim, and C. Ozdogan. Parallel wavelet-based clustering algorithm on GPUs using CUDA. Procedia Computer Science, v. 3, p. 396-400, 2011.

[21] F. Fabris, and R. A. Krohling. A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on GPU using C-CUDA. Expert Systems with Applications, v. 39, p. 10324-10333, 2012.

[22] Atasoy *et al.* Using gauss-Jordan elimination method with CUDA for linear circuit equation systems. Procedia Technology, v. 1, p. 31-35, 2012.

[23] J. Reese, and S. Zaranek. GPU Programming in MATLAB. The MathWorks, Inc.http://www.mathworks.com/company/newsletters/articles/gpu-programming-in-matlab.html, 2011.

[24] X. Liu. *et al* .Research and Comparison of CUDA GPU Programming in MATLAB and Mathematica. In: Proceedings of 2013 Chinese Intelligent Automation Conference, 2013, pp. 251-257.

[25] J. Little, and C. Moler. MATLAB GPU Computing Support for NVIDIA CUDA-Enabled GPUs. The MathWorks, Inc.http://www.mathworks.com/discovery/matlab-gpu.html, 2013.

[26] CUDA. Accelerating MATLAB with CUDA. NVIDIA Corporation, v. 1, 2007.

[27] V. Simek, and R. R. Asn. GPU acceleration of 20d-dwt image compression in matlab with cuda. In: Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on, pp. 274-277. IEEE, 2008.

[28] J. Kong *et al*. Accelerating matlab image processing toolbox functions on gpus. In: Proceedings of the 3rd Workshop on General-Purpose Computation onGraphics Processing Units, 2010, pp. 75–85.

[29] Y. Saad, and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on scientifc and statistical computing, v. 7, p. 856-869.