# PROGRAMAÇÃO PARALELA DE UM MÉTODO ITERATIVO PARA SOLUÇÃO DE GRANDES SISTEMAS DE EQUAÇÕES LINEARES USANDO A INTEGRAÇÃO CUDA-MATLAB

**Lauro Cássio Martins de Paula**
Universidade Federal de Goiás – UFG/Brazil
lauro_cassio@hotmail.com

***Resumo:*** **Sistemas de equações lineares podem aparecer como resultado da modelagem de diversos problemas da área de matemática, engenharia e ciência da computação. O método Gradiente Bi-Conjugado Estabilizado (BiCGStab) é um método iterativo utilizado para solucionar sistemas lineares, principalmente sistemas esparsos e de grande porte. Nesse contexto, este artigo propõe uma implementação paralela do método BiCGStab para solução de grandes sistemas lineares. A implementação proposta faz uso de uma *Graphics Processing Unit* (GPU) por meio da integração CUDA-Matlab, onde as operações do método são executadas nos núcleos de processamento da GPU pelas próprias funções do Matlab. Tal implementação visa proporcionar um desempenho computacional superior em relação à sua implementação sequencial. Adicionalmente, comparou-se o desempenho computacional do BiCGStab com uma implementação do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)), proposta recentemente pelo autor, na solução de sistemas lineares aleatórios e com tamanhos variados. Os resultados mostraram que o BiCGStab paralelizado é mais eficiente na solução dos sistemas tratados. Foi possível obter ganhos de eficiência computacional de aproximadamente 5x em relação à implementação sequencial do BiCGStab. Em comparação com o BiCGStab(2), o BiCGStab paralelizado se mostrou, em média, 2x mais rápido.**

***Palavras-chave:*** **Matlab; BiCGStab; Sistemas Lineares; GPU; CUDA.**

***Abstract:*** **Linear Equations Systems may appear as modeling result of many problems in mathematics, engineering and computer science. The Bi-Conjugate Gradient Stabilized (BiCGStab) method is an iterative method used for solving linear systems, specially the sparse and large ones. In this context, this paper proposes a parallel implementation of the BiCGStab method for solving large linear systems. The proposed implementation uses a Graphics Processing Unit (GPU) through the CUDA-Matlab integration, in which the method operations are performed in the processing cores of the GPU by the Matlab built-in functions. Such implementation aims to provide a high computational performance in relation to its sequential implementation. In addition, we compare the BiCGStab computational performance with an implementation of the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) method, recently proposed by the author in the solution of random linear systems with varying sizes. The results showed that the parallelized BiCGStab is more efficient in solving the treated systems. It was possible to obtain gains of computational efficiency of approximately 5x in relation to the sequential implementation of the BiCGStab. Compared with the BiCGStab(2) the parallelized BiCGStab was on average 2x faster.**

***Keywords:*** **Matlab; BiCGStab; Linear Systems; GPU; CUDA.**

## I. INTRODUÇÃO

A solução de sistemas de equações lineares pode significar em resultados para diversos problemas do mundo real, tais como problemas da matemática, ciência da computação e engenharia hidráulica. Um sistema linear é formado por um conjunto finito de equações aplicadas a um conjunto finito de variáveis, normalmente representado por:

$$Ax = b, \quad (1)$$

em que $A$ é a matriz dos coeficientes, $b$ é o vetor dos termos independentes e $x$ o vetor das incógnitas (ou variáveis).

Para solucionar o sistema da Equação (1), pode-se utilizar algum método de solução. Diversos métodos disponíveis na literatura podem ser utilizados e alguns deles são considerados ótimos em relação ao custo computacional [18]. Métodos iterativos clássicos como Jacobi [28] e Gauss-Seidel [20], apesar de sua fácil utilização, são lentos e não conseguem solucionar qualquer tipo de sistema linear [15]. Em alguns casos, o tamanho dos sistemas lineares pode ser consideravelmente grande e exigir um tempo computacional significativo para a solução. Com isso, a implementação e utilização de métodos eficientes se torna importante e indispensável para que as soluções dos sistemas sejam realizadas com qualidade e em um tempo hábil [17].

Os métodos iterativos são considerados mais eficientes para solucionar sistemas esparsos (onde a maioria dos coeficientes são nulos) e de grande porte [1]. O método Gradiente Bi-Conjugado Estabilizado (BiCGStab), proposto por van der Vorst [10], é um método iterativo que se destaca por sua robustez e eficiência. Além disso, o BiCGStab tem sido um dos métodos mais citados e utilizados na área de Matemática Aplicada desde a década de 1990 [17].

Nesse contexto, este trabalho apresenta uma paralelização do método BiCGStab para solução de sistemas lineares grandes e esparsos. Para a paralelização, utilizou-se uma *Graphics Processing Unit* (GPU) por meio da integração *Compute Unified*

*Device Architecture* (CUDA)-Matlab, em que todas as operações são executadas implicitamente nos núcleos da GPU por funções padrão do software Matlab®.

O objetivo da implementação proposta foi proporcionar um desempenho computacional eficiente na solução de diversos sistemas com tamanhos variados. Para a validação do trabalho, comparou-se o desempenho do BiCGStab com sua implementação sequencial e com uma implementação do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)), recentemente proposta pelo autor em [18]. Foi possível observar uma redução significativa do tempo computacional nos testes com a aplicação do BiCGStab paralelizado. Ganhos de eficiência computacional de aproximadamente 5x e 2x foram obtidos, respectivamente, em relação à implementação sequencial do BiCGStab e BiCGStab(2) paralelizado.

O restante deste artigo está organizado da seguinte forma. A Seção II descreve os trabalhos mais recentes na literatura que fizeram uso de GPU para solucionar sistemas lineares. Na Seção III, introduz-se o método iterativo BiCGStab. A Seção IV descreve o modelo de programação CUDA e sua integração com o Matlab. Os materiais e os métodos utilizados para se alcançar o objetivo do trabalho são descritos na Seção V. Os resultados são apresentados na Seção VI. Por fim, a Seção VII destaca as conclusões do trabalho e aponta sugestões para trabalhos futuros.

## II. TRABALHOS RELACIONADOS

Muitos trabalhos na literatura têm apresentado resultados relevantes para solucionar problemas que envolvem sistemas de equações lineares. Alguns deles fizeram uso de GPUs na tentativa de reduzir o tempo computacional. Entre eles, pode-se citar o trabalho de Bowins [4], que propôs uma paralelização para os métodos BiCGStab e Jacobi, evidenciando que o BiCGStab supera o método de Jacobi para matrizes densas. Entretanto, uma comparação com outros métodos iterativos, tal como BiCGStab(2), não foi realizada.

Paula *et al.* [15] propuseram uma paralelização para o método BiCGStab(2), destacando uma superioridade em relação à implementação sequencial do método. No entanto, não foi realizada uma comparação com outros métodos de solução.

Bueno [5], em sua dissertação, fez uso de múltiplas GPUs com *Open Computing Language* (OpenCL) para solucionar sistemas lineares de grande porte utilizando o método Gradiente Conjugado (CG). Porém, o método CG é considerado o mais simples entre todas suas posteriores versões, além de não ser tão eficiente quanto métodos como o BiCGStab e BiCGStab(2) [11].

Paula [17], utilizando a implementação proposta em [15], apresentou uma comparação de desempenho computacional entre as implementações sequenciais dos métodos Jacobi, Gauss-Seidel e BiCGStab, evidenciando que o BiCGStab(2) paralelizado também supera tais métodos.

Mais recentemente, Paula [18] propôs uma paralelização para o método BiCGStab(2) utilizando uma GPU por meio da integração CUDA-Matlab, onde ganhos computacionais de mais de 70x puderam ser obtidos em relação à implementação proposta em [15]. No entanto, uma comparação com algum outro método clássico ou robusto não foi realizada. Sendo assim, este artigo, além de propor uma paralelização para o método BiCGStab, realiza uma comparação com a implementação do método BiCGStab(2) (o qual é uma adaptação do BiCGStab) proposta por Paula [18].

## III. BICGSTAB

O Gradiente Bi-Conjugado Estabilizado (BiCGStab) é um método iterativo desenvolvido por van der Vorst [11]. Esse método é uma extensão do método CG e, além de ser considerado robusto e eficiente, tem sido bastante utilizado na solução das equações diferenciais de escoamentos de fluidos [17].

A Figura 1 representa o algoritmo do BiCGStab. Assim como em [15], [17] e [18], 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 $x_n$ 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, $maior\left(\frac{x_n - (x_n - 1)}{x_n}\right) < 10^{-5}$ [18].

1. $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{a}_0$
2. $\hat{\mathbf{r}}_0 = \mathbf{r}_0$
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 $\mathbf{x}_n$ for suficientemente preciso, pare.
    16. $\mathbf{r}_n = \mathbf{s}_n - \omega_n\mathbf{t}_n$

Figura 1: Algoritmo do método iterativo BiCGStab [15].

.

## IV. INTEGRAÇÃO CUDA-MATLAB

*Compute Unified Device Architecture* (CUDA) é um modelo de programação em GPU desenvolvido pela NVIDIA® em 2006. CUDA permitiu que a GPU pudesse ser utilizada para uma ampla variedade de aplicações [9]. O código em CUDA é visto como uma extensão da linguagem computacional C (CUDA-C), onde algumas palavras-chave são utilizadas para rotular as funções paralelas (*kernels*) [9]. Como mostra a Figura 2, os *kernels* são executados nos núcleos da GPU pelas *threads*, que são organizadas em blocos, e os blocos são organizados em um *grid*.

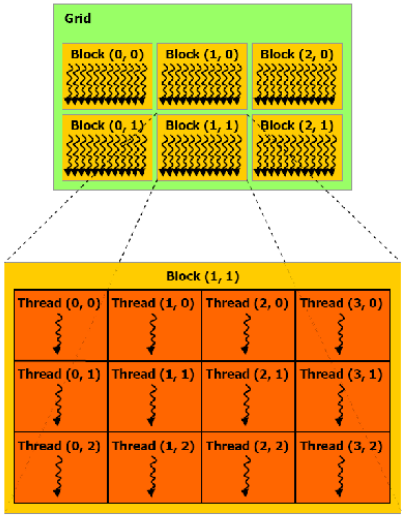


Figura 2: Organização das *threads* em CUDA [9].

Recentemente, a MathWorks® desenvolveu um *plugin* capaz de fazer a integração entre CUDA e Matlab [27]. 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 [19].

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 [25]. Entretanto, para poder ser utilizado, é necessário que a máquina tenha uma placa gráfica da NVIDIA®. Ainda, apesar de os aspectos de exploração de paralelismo serem realizados implicitamente pelo próprio PCT, o número e a organização das *threads* nos blocos não podem ser gerenciados manualmente pelo programador [18]. Por outro lado, o PCT é capaz de livrar o programador de inconveniências e otimiza, implicitamente, a execução do código nos núcleos da GPU [24].

## V. MATERIAIS E MÉTODOS

Todos os sistemas lineares solucionados neste trabalho foram gerados aleatoriamente utilizando-se funções padrão do software Matlab® (versão R2013a). A matriz dos coeficientes (*A)* 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 [18].

O vetor das incógnitas (*x*) 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 (*b*) foi gerado multiplicando-se a matriz *A* e o vetor *x*.

Para cada um dos sistemas gerados, foi repassado para o método apenas a matriz *A* e o vetor *b*, que, após a tentativa de convergência do sistema, retornou o vetor *x*.

No algoritmo do método, cada passo é realizado sequencialmente e a paralelização ocorre apenas nas operações de multiplicação, adição e subtração de cada passo do algoritmo. É importante destacar que, nas operações matemáticas apenas entre os escalares, o desempenho computacional na CPU pode ser mais eficaz do que na GPU devido à existência de uma característica intrinsecamente sequencial. No entanto, nos passos em que ocorrem operações entre matrizes e vetores, o paralelismo é eficientemente explorado pela integração CUDA-Matlab, proporcionando um desempenho computacional mais significativo.

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, e 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.

## VI. RESULTADOS

Com o intuito de verificar o ganho computacional obtido com a implementação paralelizada, os resultados obtidos com o método BiCGStab foram confrontados com sua implementação sequencial. Adicionalmente, realizou-se uma comparação com as implementações do método BiCGStab(2) propostas em [15].

O gráfico comparativo do tempo de processamento em segundos dos sistemas lineares solucionados com o método BiCGStab sequencial e paralelizado é apresentado na Figura 3. É possível verificar que o BiCGStab paralelizado é, em média, 5,2x mais rápido que o BiCGStab sequencial na solução dos sistemas tratados.

A Figura 4 mostra a média dos ganhos de eficiência computacional (*speedup*) obtidos na solução de cada sistema. Por exemplo, para a solução de um sistema com dimensão igual a 10.000, o

BiCGStab paralelizado se mostrou aproximadamente 11x mais rápido que a sua versão sequencial.

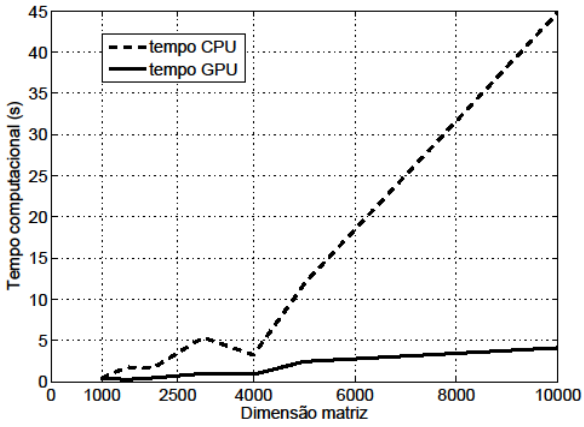


Figura 3: Comparação da velocidade de cálculo para sistemas com dimensão entre 1000 e 10000.

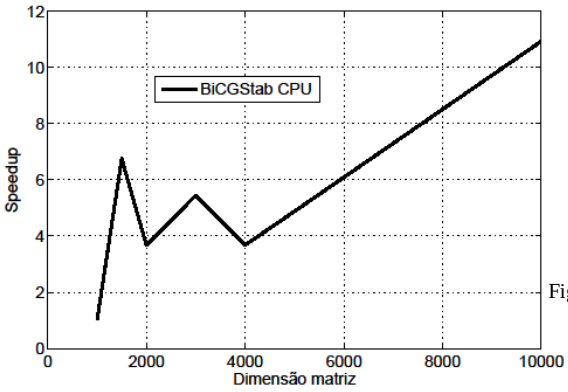


Figura 4: Ganhos de *speedup* do BiCGStab paralelizado em relação à sua implementação sequencial.

A Figura 5 mostra uma comparação entre o BiCGStab sequencial e a implementação sequencial do BiCGStab(2) proposta por Paula [15]. Observa-se que o tempo para o BiCGStab é relativamente inferior apenas para sistemas lineares com dimensão maior que aproximadamente 1.800. Com isso, nota-se que a implementação sequencial do BiCGStab seria uma implementação mais apropriada, do ponto de vista computacional, para sistemas com dimensão relativamente grande.

Como mostra a Figura 6, a média do ganho de *speedup* proporcionado pelo BiCGStab foi de aproximadamente 1,3x. O BiCGStab sequencial apresentou um melhor desempenho para o sistema com dimensão igual a 4.000, sendo aproximadamente 2,7x mais rápido que o BiCGStab(2) sequencial.

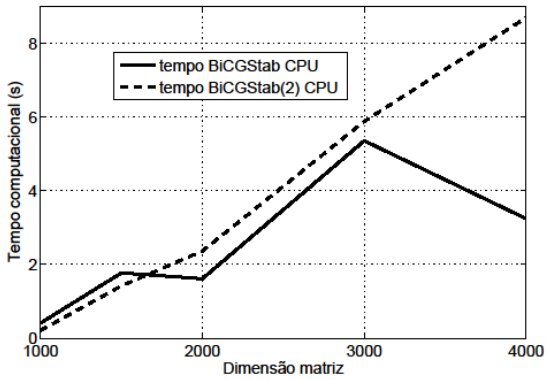


Figura 5: Comparação da velocidade de cálculo para sistemas com dimensão entre 1000 e 4000 entre as implementações sequenciais do BiCGStab e BiCGStab(2).

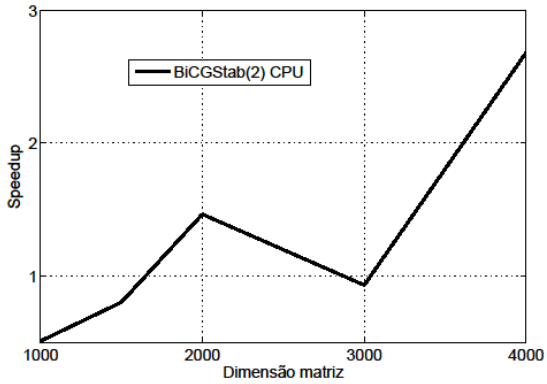


Figura 6: Ganhos de *speedup* do BiCGStab sequencial em relação à implementação sequencial do BiCGStab(2).

A Figura 7 mostra uma comparação entre o BiCGStab paralelizado e a implementação paralelizada do BiCGStab(2) proposta em [18]. Assim como no caso anterior, nota-se a superioridade do BiCGStab na solução dos sistemas tratados. É possível observar que o BiCGStab paralelizado é, em média, 2,2x mais rápido que o BiCGStab(2).

A Figura 8 representa a média do *speedup* obtido na solução de cada sistema. Nota-se que, em comparação com o BiCGStab(2) paralelizado, proposto por Paula [18], o BiCGStab pode ser uma aplicação mais apropriada para solucionar sistemas lineares esparsos e de grande porte em um tempo computacional reduzido.

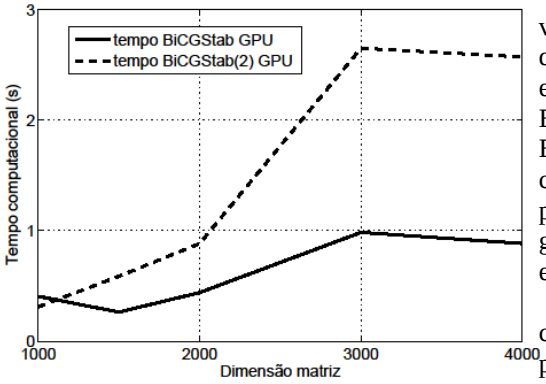


Figura 7: Comparação da velocidade de cálculo para sistemas com dimensão entre 1000 e 4000 entre as implementações paralelizadas do BiCGStab e BiCGStab(2).

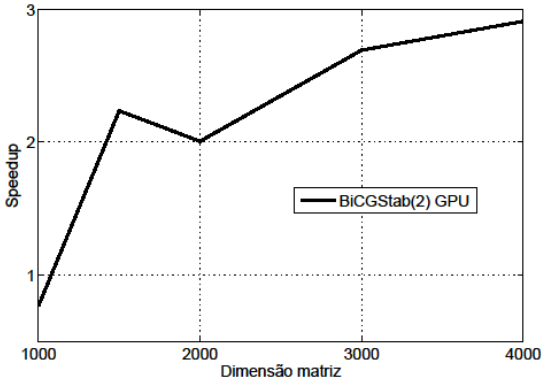


Figura 8: Ganhos de *speedup* do BiCGStab paralelizado em relação à implementação paralelizada do BiCGStab(2).

## VII. CONCLUSÃO

A utilização de métodos iterativos eficientes se torna uma tarefa importante nas áreas que envolvem a solução de grandes sistemas de equações lineares. Por outro lado, o tempo computacional cresce de forma proporcional com o tamanho do sistema. A utilização de métodos robustos e uma exploração eficiente de paralelismo pode ser um fator importante para a redução do tempo de processamento.

Foi implementado e utilizado neste trabalho um código computacional em Matlab do método iterativo BiCGStab para solução de sistemas lineares de tamanhos variados. 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 foi viabilizar a solução rápida de sistemas lineares e comparar o desempenho computacional com a implementação sequencial. Adicionalmente, realizou-se uma comparação com uma implementação sequencial e outra paralelizada do método BiCGStab(2), proposta em [18].

Após uma extensa revisão bibliográfica realizada, verificou-se que Paula [18] foi o único autor até então que apresentou uma paralelização para o BiCGStab(2) em GPU. De acordo com Paula [17] [18], o método BiCGStab(2), em geral, 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.

Para os sistemas solucionados neste artigo, constatou-se uma superioridade do BiCGStab paralelizado em relação ao tempo computacional gasto no cálculo de cada sistema. Foi possível obter um ganho de *speedup* em torno de 5x e 2x em comparação com a implementação sequencial do BiCGStab e paralelizada do BiCGStab(2) [15], respectivamente. Portanto, foi possível concluir que o BiCGStab proposto, em comparação com o BiCGStab(2) proposto por Paula [15], seria uma implementação mais apropriada para a obtenção de um desempenho computacional reduzido.

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

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

[4] 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.

[5] A. L. C. Bueno. 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.

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

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

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

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

[10] 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.

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

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

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

[14] R. A. Gaioso, and L. C. M. Paula *et al.* Paralelização do Algoritmo Floyd-Warshall usando GPU. In: XIV Simpósio em Sistemas Computacionais, 2013, pp. 19-25.

[15] 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.

[16] 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.

[17] 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.

[18] L. C. M. Paula. Implementação Paralela do Método BiCGStab(2) em GPU usando CUDA e Matlab para Solução de Sistemas Lineares. Revista de Sistemas e Computação, v. 3, p. 125-131, 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, v. 4, p. (In press), 2014.

[20] W. Li, and W. Sun. Modified Gauss-Seidel type methods and Jacobi type methods for Z-matrices. Linear Algebra and its Applications, v. 317, p. 227-240, 2000.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] G. L. G. Sleijpen, and H. A. Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. SIAM Review, v. 42, p. 267-293, 2000.

[29] 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.

[30] 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.