Aplicação de Processamento Paralelo em método iterativo para solução de sistemas lineares

Lauro C. M. de Paula[1], **Leonardo B. S. de Souza**[2], **Leandro B. S. de Souza**[3], **Wellington S. Martins**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.001-970 – Goiânia – GO – Brazil

[2]Escola de Engenharia Civil – Universidade Federal de Goiás (UFG)
Caixa Postal – 74.605-220 – Goiânia – GO – Brazil

[3]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Caixa Postal – 31.270-901 – Belo Horizonte – MG – Brazil

{lauropaula, wellington}@inf.ufg.br, leobarra@eec.ufg.br, lbss@ufmg.br

Abstract. *This paper presents a C++ source code with parallel programming of the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) method, used for solving linear systems. The BiCGStab(2) method combines the Bi-Conjugate Gradient Stabilized (BiCGStab) method advantages, resulting in a convergence superior guarantee. The goal is to enable the rapid solution of linear systems to more complex problems (larger systems) can be solved in a short time. To validate the paper we used the parallel processing capacity of a GPU through the* $NVIDIA^{®}$ *Compute Unified Device Architecture (CUDA) and compared the sequential and parallelized code computational performance of the BiCGStab(2) applied to solving linear systems of varying sizes. There was a significant acceleration in tests with the parallelized code, which increases considerably as much as systems increase. It was possible to obtain performance gains for different systems size. The results allowed us to evaluate the developed code as a useful tool for solving large linear systems applied in various areas of science.*

Resumo. *Apresenta-se neste trabalho um código computacional em linguagem C++ com programação paralela do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)), utilizado para solução de sistemas lineares. O método em questão reúne as vantagens do método Gradiente Bi-Conjugado Estabilizado (BiCGStab), resultando em garantia superior de convergência. O objetivo é viabilizar a solução rápida de sistemas de equações lineares para que problemas cada vez mais complexos possam ser solucionados em um tempo curto. Para a validação do trabalho, utilizou-se a capacidade de processamento paralelo de uma GPU, por meio da arquitetura CUDA, e comparou-se o desempenho computacional dos códigos sequencial e paralelizado do BiCGStab(2) na solução de sistemas lineares de tamanhos variados. Observou-se uma aceleração significativa nos testes com o código paralelizado, que se acentua consideravelmente na medida em que os sistemas aumentam. Foi possível obter ganhos (*$speedup$*) de desempenho para diferentes tamanhos de sistema. Os resultados permitiram avaliar o código desenvolvido como uma ferramenta útil para solução de grandes sistemas lineares aplicados nas diversas áreas da ciência.*

1. Introdução

No campo de investigação da Dinâmica dos Fluidos Computacional (*Computational Fluid Dynamics* - CFD), existem diversos métodos utilizados para solução de sistemas lineares. Alguns deles são considerados ótimos em relação ao custo computacional, dependendo do tamanho do sistema a ser resolvido. 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 as simulações de escoamentos de fluidos sejam realizadas com qualidade e em tempo não proibitivo.

Como os sistemas lineares encontrados na CFD são, normalmente, grandes e esparsos, métodos iterativos de solução são mais indicados do que métodos exatos, também 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 iterativos clássicos como o de Jacobi e de Gauss-Seidel, apesar de sua fácil implementação, podem apresentar convergência lenta ou mesmo não convergirem para grandes sistemas [Versteeg and Malalasekera 1995]. Portanto, a pesquisa e a comparação de métodos iterativos a serem implementados em códigos computacionais podem ser consideradas tarefas importantes tanto na CFD quanto em outras áreas da ciência que envolvam a solução de grandes sistemas de equações lineares.

Trabalhos recentes fizeram uso de GPU (*Graphics Processing Unit*), por meio da arquitetura CUDA (*Compute Unified Device Architecture*), desenvolvida e mantida pela $NVIDIA^{®}$, para solucionar sistemas lineares. Elise Bowins [Bowins 2012], por exemplo, apresenta uma comparação de desempenho computacional entre o método de Jacobi e o método Gradiente Bi-Conjugado Estabilizado (BiCGStab), desenvolvido por Van der Vorst [Vorst 1992]. 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 usando CUDA supera a implementação sequencial.

Nesse contexto, este trabalho apresenta um código computacional em linguagem C++ com programação paralela do método Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)). Tal método foi implementado em uma versão sequencial e outra paralelizada. Para a versão sequencial, foi utilizada a linguagem computacional C++, e para a paralelizada foi utilizada uma extensão da linguagem C, CUDA-C. O objetivo foi viabilizar a solução rápida de sistemas lineares para que problemas cada vez mais complexos (sistemas maiores) possam ser solucionados em um espaço curto de tempo. Para tal, foi utilizado a capacidade de processamento paralelo de uma unidade de processamento gráfico (GPU), por meio da arquitetura CUDA, e comparou-se o desempenho computacional dos códigos sequencial e paralelizado do BiCGStab(2) na solução de sistemas lineares de tamanhos variados. Tanto quanto sabemos, não se tem conhecimento de outros trabalhos publicados com programação paralela do método BiCGStab(2). Portanto, este artigo não faz comparação da versão paralelizada do BiCGStab(2) com implementações de outros trabalhos.

Este artigo está organizado da seguinte forma. Na Seção 2, é detalhado o método iterativo BiCGStab(2). Na Seção 3, descreve-se o material e os métodos utilizados para

se alcançar o objetivo do trabalho. Os resultados são discutidos na Seção 4. A Seção 5 traz as conclusões e algumas propostas para trabalhos futuros.

2. Método BiCGStab(2)

Resolver um sistema linear **Ax = b** para o vetor **x**, onde **A** é uma matriz de n linhas e n colunas e **b** um vetor de n linhas e 1 coluna, pode consumir bastante tempo, principalmente quando **A** é grande. Existem diversos métodos iterativos para resolver esse sistema linear. Tais métodos realizam aproximações sucessivas, em cada iteração, para obter uma solução mais precisa para o sistema [Bowins 2012]. Ainda, métodos iterativos são recomendados para sistemas lineares grandes com matrizes esparsas [Molnarka and Miletics 2005].

Os métodos iterativos podem ser classificados em dois grupos: os métodos estacionários e os métodos não-estacionários. Nos métodos estacionários, a mesma informação é usada em cada iteração. Como consequência, estes métodos são, normalmente, mais fáceis de implementar. Nos métodos não-estacionários, a informação usada pode mudar a cada iteração. Esses métodos são mais difíceis de implementar, mas, geralmente, fornecem uma convergência mais rápida para o sistema. Além disso, métodos iterativos se mostram mais adequados mesmo quando a matriz dos coeficientes é densa [Vorst 1995].

Neste trabalho, foca-se em um método iterativo não-estacionário, chamado método do Gradiente Bi-Conjugado Estabilizado Híbrido (BiCGStab(2)). O método BiCGStab(2) foi desenvolvido por Van der Vorst e Sleijpen [Sleijpen and Vorst 1995]. De acordo com seus criadores, tal método reúne as vantagens do BiCGStab e do GMRES (*Generalized Minimum Residual*), resultando, normalmente, em um método com garantia de convergência superior à do BiCGStab e adequado, por exemplo, para solução de sistemas lineares gerados na solução das equações diferenciais de escoamentos de fluidos.

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, ...
   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 1. Algoritmo do método iterativo BiCGStab(2), adaptado de [Sleijpen and Vorst 1995].

É representado na Figura 1 o algoritmo do método BiCGStab(2). 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.

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, fornecida inicialmente como, por exemplo, 10^{-5}. Ou seja, $maior(\frac{x_i-(x_{i-1})}{x_i}) < 0.00001$.

3. Material e Métodos

A Unidade de Processamento Gráfico de Propósito Geral (ou GPGPU) utiliza a GPU não apenas para renderização gráfica, mas também para processamento de imagem, visão computacional, cálculo numérico, dentre outras aplicações. 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.

Diferentemente de uma CPU (*Central Processing Unit*) Multicore, a qual executa, normalmente, algumas $threads$ em paralelo, a GPU foi projetada para executar milhares de $threads$ em paralelo. Modelos de programação, tal como CUDA [$CUDA^{TM}$ 2011] e OpenCL [Tsuchiyama et al. 2010], permitem que as aplicações sejam executadas mais facilmente na GPU [Xiao and Feng 2009]. CUDA foi a primeira arquitetura e interface para progamação de aplicação (*Application Programming Interface* - API), criada pela $NVIDIA^{®}$ em 2006, a permitir que a GPU pudesse ser usada para uma ampla variedade de aplicações [Bowins 2012]. Em CUDA, a CPU é chamada $host$, onde toda a parte sequencial do código é executada e as funções $kernel$ são chamadas para serem executadas na GPU. A GPU, por sua vez, é chamada $device$, a qual pode executar todas as partes aritmeticamente intensivas do código [Gravvanis and Papadopoulos 2011]. Neste trabalho, o algoritmo do método BiCGStab(2), descrito na seção 2, foi codificado em linguagem C++ empregando uma implementação inteiramente sequencial, bem como uma implementação parcialmente paralelizada empregando CUDA-C [$CUDA^{TM}$ 2009].

Ao analisar o algoritmo do método BiCGStab(2), é possível verificar uma forte dependência de dados, isto é, alguns passos não podem ser executados antes que outros sejam executados anteriormente. Devido a essa característica, apenas algumas partes do método foram paralelizadas, como: 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:

$$x_i = x_i + \alpha p_i + \omega_1 r_i + \omega_2 s_i. \quad (1)$$

Portanto, cada elemento do vetor x é calculado em paralelo. O passo 11 (v = Ap) somente poderá ser executado após o passo 10 ter sido totalmente executado. Para

garantir essa condição, utiliza-se uma barreira de sincronização, por meio de uma função padrão ($cudaDeviceSynchronize()$) da arquitetura CUDA. De forma análoga, o mesmo ocorre para outras etapas do algoritmo.

Os produtos escalares são realizados sequencialmente. Poderia ser possível aumentar o paralelismo dividindo os produtos entre múltiplas somas feitas em paralelo, mas isso pode exigir muita sobrecarga ($overhead$) para resultar em um ganho de desempenho significante [Bowins 2012].

A seguir, mostra-se um pequeno trecho do código em CUDA-C usado para a implementação do BiCGStab(2). Inicialmente, o código é executado na CPU, onde as funções $kernel$ são chamadas para serem executadas na GPU. A multiplicação entre matriz (A) e vetor (x), por exemplo, acontece em duas etapas, por meio de duas chamadas consecutivas à duas funções $kernel$ distintas. Na primeira função $kernel$, uma matriz auxiliar (C) recebe a multiplicação de cada elemento da matriz pela respectiva posição do vetor. Para isso, $n\ threads$ executam em paralelo, onde cada $thread$ percorre uma linha da matriz. Na segunda função, um vetor auxiliar (aux) recebe a soma dos elementos de cada linha da matriz C, obtendo o vetor resultante da multiplicação.

$$multiplicacao1 <<< nBlocos, nThreadsPorBloco >>> (A, x, C, n); \quad (2)$$

$$multiplicacao2 <<< nBlocos, nThreadsPorBloco >>> (C, aux, n); \quad (3)$$

Para cada uma das funções $kernel$, $nBlocos$ = $nThreadsPorBloco$ = $\sqrt{n}$. Ou seja, o número de blocos e o número de $threads$ por bloco, especificados na chamada da função, são iguais a raiz quadrada do número de linhas (n) da matriz A. Se $\sqrt{n}$ não é um número inteiro, então esse valor é arredondado para cima.

Existem alguns passos separados do método BiCGStab(2) que poderiam ser executados em paralelo, porém isso não é possível utilizando uma única GPU. Uma possível solução seria utilizar múltiplas GPUs ou, provavelmente, uma próxima geração de GPUs. Entretanto, a maioria dos passos do algoritmo devem ser realizados em sequência.

Os resultados obtidos com o método BiCGStab(2) paralelizado foram confrontados com os do BiCGStab(2) sequencial, 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 de vídeo $NVIDIA^{®}$ GeForce GTX 550Ti dispondo de 2 GB de memória e 192 processadores operando a 900MHz.

4. Resultados

Todos os sistemas lineares utilizados neste trabalho foram gerados através de funções do software MATLAB (versão R2011a). A matriz (A) 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 (x) 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 (b) 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 o método BiCGStab(2), sequencial e paralelizado, apenas a matriz A e o vetor b, que, após a tentativa de convergência do sistema, retornou o vetor x.

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

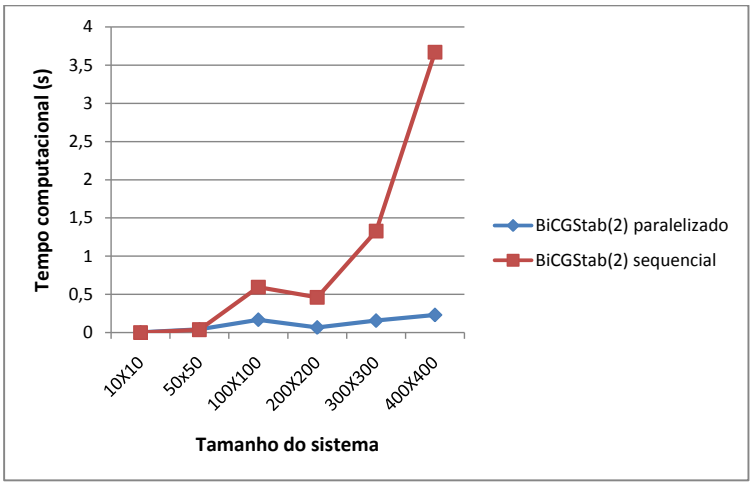

Figura 2. Comparação da velocidade de cálculo para os sistemas lineares de dimensão entre 10x10 e 400x400.

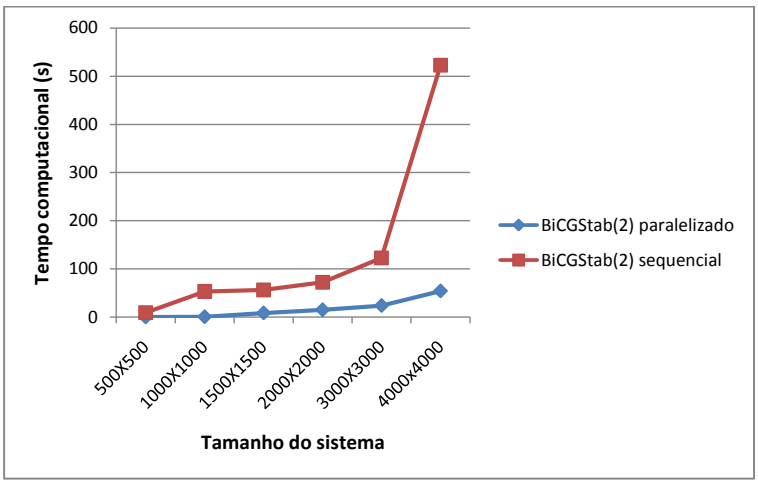

Figura 3. Comparação da velocidade de cálculo para os sistemas lineares de dimensão entre 500x500 e 4000x4000.

4.1. Avaliação do ganho computacional proporcionado pela versão paralelizada do BiCGStab(2)

Analisando os gráficos nas figuras 2 e 3, nota-se a superioridade do método BiCGStab(2) paralelizado na solução dos sistemas tratados. É possível verificar que a versão sequencial do método requer um esforço computacional que aumenta rapidamente com o tamanho

do sistema. Observa-se também que o tempo para a versão paralelizada também aumenta nesse caso, porém com taxa de crescimento menos expressiva.

Para sistemas lineares com dimensão menor que aproximadamente 50x50, a versão sequencial poderá se mostrar equivalente ou mais rápida que à paralelizada. Isso ocorre devido à existência de um $overhead$ associado à paralelização das tarefas na GPU [Filho et al. 2011]. Ou seja, 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 [$CUDA^{TM}$ 2011]. 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. Conclusões

Foi implementado e utilizado neste trabalho um código computacional do algoritmo do método iterativo BiCGStab(2), para solução de sistemas lineares de tamanhos variados. Tal método foi implementado em duas versões, uma sequencial e outra paralelizada, utilizando as linguagens C++ e CUDA-C, respectivamente. Na versão paralelizada, explorou-se o uso de uma GPU, por meio da arquitetura CUDA, para a paralelização de algumas etapas do algoritmo do BiCGStab(2).

Para os sistemas lineares avaliados neste artigo, constatou-se uma superioridade da versão paralelizada em relação ao tempo computacional gasto no cálculo de cada sistema. Foi possível obter ganhos ($speedup$) de desempenho, com a implementação usando CUDA, para diferentes tamanhos de sistema. Observou-se que a média de $speedup$ ficou em torno de $\approx$10. Portanto, concluiu-se que 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 seguintes nessa linha de pesquisa poderão envolver solução de sistemas lineares ainda maiores, como, por exemplo, os sistemas gerados nas simulações de escoamentos de fluidos, amplamente estudados na Dinâmica dos Fluidos Computacional. Além disso, poderá ser possível aplicar uma otimização no código computacional da versão paralelizada do método BiCGStab(2), de tal forma que etapas que ainda são resolvidas sequencialmente poderão ser solucionadas em paralelo. Uma possível otimização seria a utilização de *Persistent Threads* (PT), onde 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, devido à uma melhor utilização dos recursos de processamento paralelo da GPU. Adicionalmente, alternativas à arquitetura CUDA poderão ser investigadas para realização de estudos comparativos.

Agradecimentos

Os autores agradecem à CAPES pelo apoio fornecido à pesquisa.

Referências

Bowins, E. C. (2012). A comparison of sequential and gpu implementations of iterative methods to compute reachability probabilities. *Proceedings First Workshop on GRAPH Inspection and Traversal Engineering*, pages 20–34.

$CUDA^{TM}$, N. (2009). *NVIDIA CUDA C Programming Best Practices Guide*. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050.

$CUDA^{TM}$, N. (2011). *NVIDIA CUDA C Programming Guide*. NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 4.0 edition.

Filho, A. R. G., Arruda, F. D. B., Harrop, R. K., and Yoneyama, T. (2011). Programacao paralela cuda para simulacao de modelos epidemiologicos baseados em individuos. *Simposio Brasileiro de Automacao Inteligente*.

Franco, N. B. (2006). *Calculo Numerico*. Sao Paulo: Pearson Prentice Hall.

Gravvanis, G. A. and Papadopoulos, C. K. F. (2011). Solving finite difference linear systems on gpus: Cuda based parallel explicit preconditioned biconjugate conjugate gradient type methods. *Journal of Supercomputing*, 61:590–604.

Molnarka, G. and Miletics, E. (2005). A genetic algorithm for solving general system of equations. *Third Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence*.

Sleijpen, G. L. G. and Vorst, H. A. V. (1995). Hybrid bi-conjugate gradient methods for cfd problems. *Computational Fluid Dynamics REVIEW*, (902).

Tsuchiyama, R., Nakamura, T., Iizuka, T., Asahara, A., Son, J., and Miki, S. (2010). *The OpenCL Programming Book*. Fixstars.

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

Vorst, H. A. V. (1992). 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*, 13(2):631–644.

Vorst, H. A. V. (1995). Parallel iterative solution methods for linear systems arising from discretized pdes. *France Workshop Lecture Notes*.

Xiao, S. and Feng, C. (2009). Inter-block gpu communication via fast barrier synchronization. *23rd IEEE International Parallel & Distributed Processing Symposium, ISSN: 15302075*.