Parallelization of an Iterative Method for Solving Large and Sparse Linear Systems using the CUDA-Matlab Integration

Lauro Cássio Martins de Paula,
Anderson da Silva Soares
Institute of Informatics
Federal University of Goiás
Goiânia, Brazil
{lauropaula, anderson}@inf.ufg.br

***Abstract*—This paper presents a parallel implementation of the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) iterative method in a 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. The results showed that the proposed implementation is more efficient and can be viable for simulations being carried out with quality and in a timely manner. The gains in computational efficiency were 76x and 6x compared to the implementation in C and CUDA-C, respectively.**

Keywords: Matlab, GPU, CUDA, BiCGStab(2).

I. Introduction

A linear system is a linear equations finite set applied in a variable finite set. Sparse and large linear systems may appear as result of the modeling of various computer science and engineer problems [18]. To solve such systems, iterative methods are more indicated and efficient than exact methods [20]. Iterative methods use less memory space and reduce rouding errors in computer operations [15]. Such methods perform successive approximations in each iteration to obtain a more precise solution for the system.

Classical iterative methods such as Jacobi and Gauss-Seidel are considered easy to deploy and use [17]. Nevertheless, despite this feature both may have a slow convergence or even not converge for large systems [20]. Another disadvantage is that when the coefficient matrix is not square (number of rows equal to the number of columns), these two methods can not guarantee the linear system convergence. As a consequence, the research and implementation of computational methods are considered important tasks in various areas of science, particularly those that involve the solution of large linear equations systems [6].

There are several methods for solution of linear systems. Some of them are considered good in relation to the computational cost. However, the computational performance may be affected if the size of the system is large. In some cases in which the linear systems to be solved are very large, the computational processing may last too many days and the methods solution speed difference are significant. Consequently, the implementation of efficient and robust methods such as the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) becomes important and often necessary for the simulations are performed with quality and in a short time [2]. The BiCGStab(2) is an iterative method developed for solving large and sparse linear systems and is considered a good one [6].

Several studies have used the computational resources of Graphics Processing Units (GPU) to solve large and sparse linear systems. For instance, Bowins [2] presented a comparison of computational performance between the Jacobi method and the Bi-Conjugate Gradient Stabilized (BiCGStab) method. In that work, both methods were implemented in two versions: sequential and parallelized. Based on the results obtained, he showed that as the size of the system increases, the parallel implementation outperforms the sequential in terms of computational efficiency.

Weber *et al.* [21] presented graphics processing unit (GPU) data structures and algorithms to efficiently solve sparse linear systems that are typically required in simulations of multibody systems and deformable bodies. Their solving method results in a speedup factor of up to 13 in comparison to other sequential and GPU methods.

More recently, Paula *et al.* [6] proposed a parallelization of the BiCGStab(2) method for solving linear systems using Compute Unified Device Architecture (CUDA) and compared the computational performance between the sequential and parallelized versions of the method. They showed that from the computational point of view, the parallel version of BiCGStab(2) method is more efficient.

In this context, this paper presents a parallel implementation of the BiCGstab(2) method, which uses the CUDA-Matlab technology in a GPU for solving linear systems. The goal was showing that the proposed implementation can be more appropriate and, through its use, it is possible to enable the efficient solution of large and sparse linear systems for increasingly complex (larger) systems can be solved in a timely manner. To achieve this goal, we performed a comparison with the implementation of the BiCGStab(2) method proposed by Paula *et al.* [6] in the solution of linear systems of varying sizes. The results showed that the computation time can be significantly reduced with the implementation proposed in this paper. It was possible to obtain speedup gains of 76x and 6x compared with the sequential and parallelized implementation proposed in [6], respectively.

The remainder of this paper is organized as follows. It is detailed in Section II the BiCGStab(2) iterative method. Section III describes the CUDA and its integration with Matlab. The materials and methods used to achieve the objective of the work are described in Section IV. The results are presented and discussed in Section V. Finally, Section VI contains the conclusions.

II. BiCGStab(2) Method

The solution of a linear equations system $Ax = b$, where $A_{n \times n}$ is the coefficient matrix and $b_{n \times 1}$ the vector of independent terms, may require a huge computational effort especially when A is very large. For example, to solve a linear system one can use an iterative method. Iterative methods perform successive approximations in each iteration to obtain a more accurate solution and are recommended for large linear systems with sparse matrices [1].

Iterative methods are classified into two groups: stationary and non-stationary methods [6]. The stationary methods use the same information at each iteration, *i.e.*, the results of one iteration are used for the next iterations [18]. In non-stationary methods, the information used may change with each iteration. The non-stationary methods are difficult to implement but may provide a faster convergence for the system and are more suitable even when the coefficient matrix is dense (non-sparse) [20].

The BiCGStab(2) is a non-stationary iterative method developed by van der Vorst and Sleijpen [18]. This method combines the advantages of BiCGStab and Generalized Minimum Residual (GMRES) method [14]. Consequently, the BiCGStab(2) is considered a robust method and with convergence guarantee superior to BiCGStab, suitable for solution of linear systems generated in the solution of differential equations of fluid flow [18].

Algorithm 1 shows a snippet of pseudocode for the algorithm of BiCGStab(2) method. A full pseudocode can be obtained in [6]. Some adjustments were made naming comparing with the original algorithm. In the Algorithm 1, the Greek letters represent scalars, lowercase letters represent vectors expressed in matrix form, capital letters represent matrices, and parentheses with comma separated vectors represent scalar products between vectors.

In step 38 of the method, so that the vector x_{i+2} is sufficiently precise, the higher value corresponding to the difference between the results of each term of the vector x in two consecutive iterations, divided by the result of the term in the current iteration, should be less than a given accuracy as, for example, $max(\frac{x_i-(x_{i-1})}{x_i}) < 10^{-5}$.

Algorithm 1: Snippet of pseudocode for the algorithm of BiCGStab(2) method.

```
1.  $r_0$ = b - A$x_0$
2.  $\hat{r}_0$ = $r_0$
3.  $\rho$ = $\alpha$ = $\omega_1$ = $\omega_2$ = 1
4.  v = w = p = 0
5.  **for** $i$ = 0, 2, 4, ... **do**
6.      $\hat{\rho}$ = -$\omega_2\rho$
        Even BiCGStab step: from step 7 to 16
7.      $\rho$ = $r_i^T\hat{r}_0$ ...
16.     $x_i$ = $x_i$ + $\alpha$p
        Odd BiCGStab step: from step 17 to 27
17.     $\rho$ = $s^T\hat{r}_0$ ...
27.     t = As
        GMRES(2)-part: from step 28 to 38
28.     $\omega_1$ = $r^T$s ...
36.     $x_{i+2}$ = $x_i$ + $\alpha$p + $\omega_1$r + $\omega_2$s
37.     $r_{i+2}$ = $r_i$ - $\omega_1$s - $\omega_2$t
38.     If $x_{i+2}$ is accurate, stop.
39. **end for**
```

III. CUDA

Compute Unified Device Architecture (CUDA) was the first Application Programming Interface (API), created by $NVIDIA^{®}$ in 2006, to allow the GPU could be used for a wide variety of applications [4]. CUDA is supported by all graphics cards from $NVIDIA^{®}$, which are extremely parallel, having many cores with many memories and a memory cache shared by all cores. The CUDA code is an extension of the C computer language (CUDA-C), where a few keywords are used to label the parallel functions (kernels) and their data structures [3].

Since its inception, several studies have used CUDA to parallelization of various types of problems. For instance, Yldirim and Ozdogan [22] presented an algorithm as a clustering approach based on wavelet transform for parallelization on GPU using CUDA-C. Fabris and Krohling [9] proposed an algorithm of evolution implemented in CUDA-C for solving optimization problems. Atasoy *et al.* [1] presented a eliminating method implemented in CUDA-C using Gauss-Jordan to solve systems of linear equations. Paula *et al.* [6] used CUDA-C to parallelize the BiCGStab(2) method for solving linear systems of varying sizes. Finally, Paula *et al.* [8] proposed a parallelization strategy for phase 2 of the Successive Projections Algorithm using CUDA-C.

In order to help programmers, the $MathWorks^{®}$ has developed a plugin able to do integration between CUDA and Matlab. Make use of Matlab to GPU computing can enable applications to be more easily accelerated. GPUs can be used with Matlab using the Parallel Computing Toolbox (PCT). The PCT provides an efficient way to speedup codes in Matlab language, running them on a GPU [11], [7]. For this, the programmer must change the data type to input a function to use the commands (functions) in Matlab that were overloaded ($GPUArray$). Through $GPUArray$ function one can allocate memory in the GPU and make calls to various functions of Matlab, which are performed on the GPU's processing cores. Additionally, developers can make use of the PCT $CUDAKernel$ interface to integrate their code in CUDA-C with Matlab [13].

The development of applications running on the GPU using the PCT is usually easier and faster than using CUDA-C language [12]. According to Little and Moler [11], this is because aspects of exploitation of parallelism are implicitly performed by the PCT itself, freeing the programmer from many inconveniences. However, the organization and the number of threads to be executed on the GPU cores can not be managed manually by the programmer. Still, it is important to emphasize that in order to be used, the PCT requires a graphics card from $NVIDIA^{®}$.

After CUDA-Matlab integration, few studies have used this technology. For example, the $NVIDIA^{®}$ [5] released a book that demonstrates how programs developed in Matlab can be accelerated using GPUs. Simek and Asn [16] presented an implementation in MATLAB with CUDA for compression of medical images. Kong *et al.* [10] accelerated some functions in Matlab for image processing on GPUs. Reese and Zaranek [13] developed a manual programming GPUs using Matlab. More recently, Paula *et al.* [7] proposed a parallel implementation of the Firefly Algoritm using CUDA-Matlab for variable selection in a multivariate calibration problem. Based on the results of these works, we note that, in future, the PCT may be more used due to the fact of allowing a code in Matlab can be easily parallelized. Therefore, instead of implementing a kernel function and set the amount and organization of threads blocks, the programmer must only identify which parts of your code are parallelizable and make use of the built-in Matlab functions.

IV. Experimental

The GPU was initially developed as a flow-oriented technology, optimized for calculations of data-intensive applications , where many identical operations can be performed in parallel on different data [4]. Unlike a Central Processing Unit (CPU), which executes only a few threads in parallel, the GPU was designed to run thousands of them [8].

As previously mentioned, one can explore parallelism in GPUs using the PCT plugin, which provides an efficient way to speedup codes in Matlab language invoking functions that are overloaded to run in the cores of a GPU from $NVIDIA^{®}$. Thus, this paper presents an implementation of the BiCGStab(2) method in Matlab, which uses this technology. The proposed implementation is analogous to Algorithm 1. Initially, the data are transferred to the GPU memory. Soon after, the method begins execution and all operations are performed in the GPU's processing cores for threads that are created and managed implicitly by the PCT.

All the linear systems used in this paper were generated using Matlab (version R2013a) built-in functions. The coefficient matrix (A) of each system was generated randomly using the function $gallery('dorr', n)$, which returns a square matrix of dimension n, sparse and diagonally dominant. The diagonal dominant characteristic indicates that the sum of all elements in a row is not greater than the main diagonal element of the matrix. The vector of unknowns (x) was randomly generated by $randn(n, 1)$ function, which returns a vector of n rows and 1 column. The vector of independent terms (b) was generated by multiplying the matrix A and vector x. For each system generated was passed to BiCGStab(2) only the matrix A and the vector b which, after attempting convergence system, returned vector x.

To evaluate the computational gain obtained by implementing the parallelized method, it was recorded the time spent on each iteration of the BiCGStab(2) algorithm.

The purpose of this paper was not to compare the differences between Matlab and solution methods, but only use Matlab to generate the random systems and compare the speed of calculation of the methods in the solution of several linear systems.

A. Computational setup

All calculations were carried out by using a desktop computer with an Intel Core i7 2600 (3.40 GHz), 8 GB of RAM memory and a $NVIDIA^{®}$ GeForce GTX 550Ti graphics card with 192 CUDA cores and 2 GB of memory config. The Matlab R2013a software platform was employed throughout.

V. Results and Discussion

The results obtained with the BiCGStab(2) parallelized method were compared with its sequential implementation, in order to verify the computational gain obtained with parallelized implementation. Additionally, a comparison was made with implementations (sequential and parallelized) of BiCGStab(2) proposed by Paula *et al.* [6]. The comparative graphs of processing time (in seconds) of different linear systems solved with the BiCGStab(2) method (sequential and parallelized) in Matlab are shown in Figures 1 and 2.

Figure 1 shows that the sequential implementation may be more efficient for linear systems with dimensions ranging from 10 to 1000. This is due to the fact the algorithm of the method contain inherently sequential operations. For example, the scalar products running sequentially on the CPU, depending on the size of the system, may have a significantly reduced computational time compared to the same time of execution in cores of the GPU. Likewise, the operations between scalars (steps 8, 13 and 34, for example) can not be divided between multiple threads and, consequently, this may result in poor performance when executed by a single GPU thread. Furthermore, due to the existence of an overhead associated with the parallelization of tasks in GPU, the size of the system to be solved must be taken into consideration [3], [6], [8].

On the other hand, Figure 2 shows that for systems with dimension greater than 1500, the parallelized BiCGStab(2)

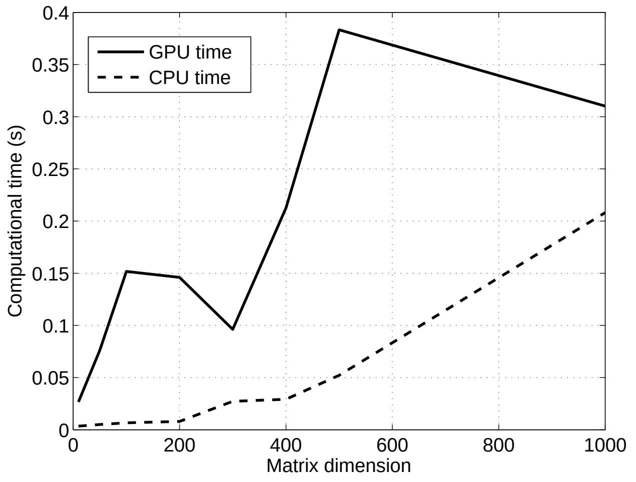

Fig. 1. Comparison of calculation speed for systems with dimension between 10 and 1000.

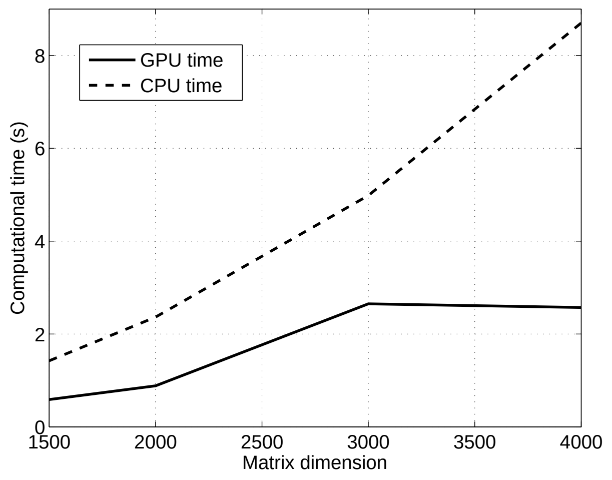

Fig. 2. Comparison of calculation speed for systems with dimension between 1500 and 4000.

exceeds the sequential implementation. In this case, in comparison of computational efficiency, the speedup gain obtained was approximately 2.59x. Therefore, the implementation that uses the GPU would be more appropriate since the size of the system used is greater than 1500×1500.

Figure 3 shows a comparison between the proposed sequential implementation and the sequential implementation proposed by Paula *et al.* [6]. The BiCGStab(2) implemented in Matlab is much higher compared to the same implementation in C language. It is observed that the time for implementation proposed by Paula *et al.* [6] requires a computational effort which increases approximately exponentially with the size of the system, while the time for implementation in Matlab is less pronounced. The speedup gain provided by the sequential implementation in Matlab was approximately 76.75x. Consequently, the use of the method implementation in Matlab can provide a more significant gain of computational performance.

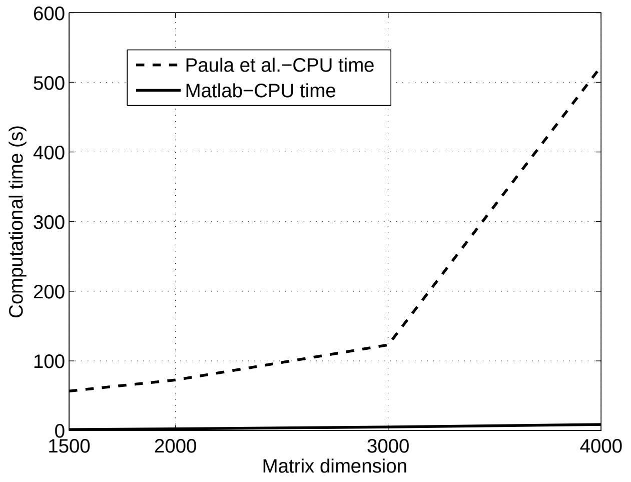

Fig. 3. Comparison of calculation speed for systems with dimension between 1500 and 4000 between sequential implementations of the BiCGStab(2) in Matlab and C.

Figure 4 shows a comparison between the proposed parallelized implementation and the parallelized implementation proposed by Paula *et al.* [6]. As in the previous case, it is possible to note the superiority of the parallelized BiCGStab(2) using CUDA-Matlab integration in the solution of the treated systems. It can be seen that the time for implementation in CUDA-C also requires a computational effort approximately exponentially in that the size of the system increases. In this case, the speedup obtained was approximately 6.12x. Therefore, compared to the parallelized implementation proposed by in [6], the parallelized BiCGStab(2) in Matlab can be a more appropriate choice of the computational point of view.

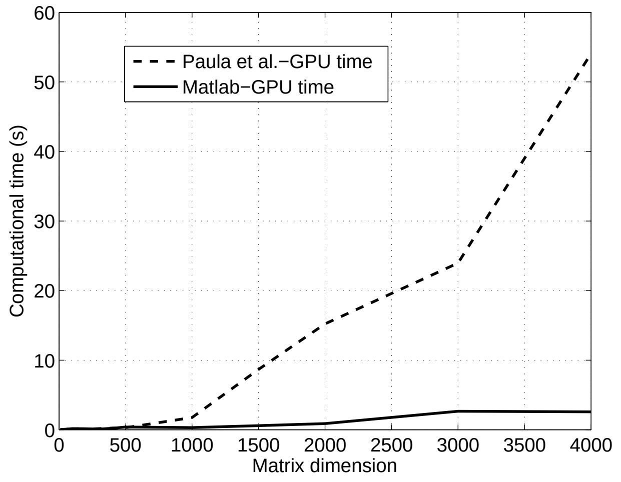

Fig. 4. Comparison of calculation speed for systems with dimension between 1500 and 4000 between parallelized implementations of the BiCGStab(2) in CUDA-Matlab and CUDA-C.

VI. Conclusion

We have implemented and used in this work a computer code in Matlab of the BiCGStab(2) iterative method for solution of large and sparse linear systems. The method was implemented on a fully sequential version as well as in a parallelized version using a GPU with CUDA-Matlab integration. The purpose of this paper was to present a new implementation of BiCGStab(2) to enable the rapid solution of linear systems and compare the computational performance with the sequential implementation. Additionally, a comparison was made with the sequential and parallelized implementation proposed in [6].

For the systems evaluated here, it was found a superiority of the parallelized implementation with CUDA-Matlab regarding the computational time spent in the calculation of each system. It was possible to obtain a speedup gain of around 76x and 6x compared to the sequential and parallelized implementation presented in [6], respectively. Compared to the sequential implementation in Matlab, the parallelized BiCGstab(2) was faster only for systems with dimension greater than 1500, and the speedup was approximately 2.5x. Therefore, it was concluded that the implementation of the method that performs in the GPU, compared to implementations proposed by Paula *et al.* [6], would be a more suitable and appropriate implementation to obtain a significant computational performance.

Future works in this same line of research may solve linear systems with larger dimensions than this paper. The systems generated in the simulations of fluid flow problems studied in the Computational Fluid Dynamics may be solved. Techniques for efficient exploitation of parallelism in scalar product between vectors operations can also be applied in an attempt to further increase the computational performance. Furthermore, alternatives to CUDA-Matlab integration such as OpenCL [19] may be investigated for comparative studies.

Acknowledgment

The authors thank the research agencies CAPES and FAPEG for the support provided to this research.

References

[1] Nesrin Aydin Atasoy, Baha Sen, and Burhan Selcuk, *Using gauss-jordan elimination method with cuda for linear circuit equation systems*, Procedia Technology **1** (2012), no. 0, 31–35.

[2] Elise Cormie Bowins, *A comparison of sequential and gpu implementations of iterative methods to compute reachability probabilities*, Proceedings First Workshop on GRAPH Inspection and Traversal Engineering (2012), 20–34.

[3] N. $CUDA^{TM}$, *Nvidia cuda c programming best practices guide*, NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2009.

[4] NVIDIA $CUDA^{TM}$, *Nvidia cuda c programming guide*, 5.0 ed., NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2013.

[5] NVIDIA Corp. $CUDA^{TM}$, *Accelerating matlab with cuda*, vol. 1, NVIDIA Corporation, 2007.

[6] Lauro Cássio Martins de Paula, Leonardo Barra Santana de Souza, Leandro Barra Santana de Souza, and Wellington Santos Martins, *Aplicação de processamento paralelo em método iterativo para solução de sistemas lineares*, X Encontro Anual de Computação, 2013, pp. 129–136.

[7] Lauro Cássio Martins de Paula, Anderson S. Soares, Telma W. Soares, Alexandre C. B. Delbem, Clarimar J. Coelho, and Arlindo 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 **4** (2014), no. 1, 31–42.

[8] Lauro Cássio Martins de Paula, Anderson Silva Soares, Telma W. Soares, Wellington Santos Martins, Arlindo Rodrigues Galvo Filho, and Clarimar Jos Coelho, *Partial parallelization of the successive projections algorithm using compute unified device architecture*, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2013, pp. 737–741.

[9] Fabio Fabris and Renato A. Krohling, *A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on gpu using c-cuda*, Expert Systems with Applications **39** (2012), no. 12, 10324–10333.

[10] Jingfei Kong, Martin Dimitrov, Yi Yang, Janaka Liyanage, Lin Cao, Jacob Staples, Mike Mantor, and Huiyang Zhou, *Accelerating matlab image processing toolbox functions on gpus*, Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010, pp. 75–85.

[11] J. Little and C. Moler, *Matlab gpu computing support for nvidia cuda-enabled gpus*, http://www.mathworks.com/discovery/matlab-gpu.html, 2013.

[12] Xiongwei Liu, Lizhi Cheng, and Qun Zhou, *Research and comparison of cuda gpu programming in matlab and mathematica*, Proceedings of 2013 Chinese Intelligent Automation Conference, Springer, 2013, pp. 251–257.

[13] J. Reese and S. Zaranek, *Gpu programming in matlab*, http://www.mathworks.com/company/newsletters/articles/gpu-programming-in-matlab.html, 2011.

[14] Youcef Saad and Martin Schultz, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing **7** (1986), no. 3, 856–869.

[15] Yousef Saad, *Iterative methods for sparse linear systems*, Siam, 2003.

[16] Vaclav Simek and Ram Rakesh Asn, *Gpu acceleration of 2d-dwt image compression in matlab with cuda*, Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on, IEEE, 2008, pp. 274–277.

[17] Gerard Sleijpen and Henk A. Van der Vorst, *A jacobi-davidson iteration method for linear eigenvalue problems*, SIAM Review **42** (2000), no. 2, 267–293.

[18] Gerard L. G. Sleijpen and Henk A. Van Vorst, *Hybrid bi-conjugate gradient methods for cfd problems*, Computational Fluid Dynamics REVIEW (1995), no. 902.

[19] Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Jeongdo Son, and Satoshi Miki, *The opencl programming book*, Fixstars, 2010.

[20] Henk A. Van Vorst, *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** (1992), no. 2, 631–644.

[21] Daniel Weber, Jan Bender, and Markus Schnoes, *Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications*, Computer Graphics Forum, Wiley Online Library, 2012.

[22] Ahmet Artu Yldirim and Cem Ozdogan, *Parallel wavelet-based clustering algorithm on gpus using cuda*, Procedia Computer Science **3** (2011), no. 0, 396–400.