A Hybrid Approach for Solving the 3D Helmholtz Equation on Heterogeneous Platforms

Gloria Ortega[1], Inmaculada García[2], and G. Ester Martín Garzón[1]

[1] Informatics Dpt., Univ. of Almería, Agrifood Campus of Int. Excell., ceiA3, 04120 Almería, Spain
{gloriaortega,gmartin}@ual.es
[2] Dpt. Computer Architecture, University of Málaga, 29080 Málaga, Spain
igarciaf@uma.es

Abstract. We are interested in the resolution of the 3D Helmholtz equation for real applications. Solving this problem numerically is a computational challenge due to the large memory requirements of the matrices and vectors involved. For these cases, the massive parallelism of GPU architectures and the high performance at lower energy of the multicores can be exploited. To do a fair comparison between the benefit of accelerating the three-dimensional Helmholtz equation using GPU architectures and multicore platforms, this paper describes three different parallelization schemes on a multi-GPU cluster and also includes an evaluation of their performance. The three parallel schemes consist of: (1) using the multicore processors (CPU version), (2) using the GPU devices (GPU version); and (3) using a hybrid implementation which combines CPU cores and GPU devices simultaneously (hybrid version). Experimental results show that our hybrid implementation outperforms the other approaches in terms of performance.

Keywords: Helmholtz equation, heterogeneous platforms, MPI, multi-GPUs.

1 Introduction

Modern high performance architectures are characterized by the heterogeneity of their resources. Modern supercomputers are distributed memory platforms based on a network of multicore processors which additionally incorporate accelerator devices such as GPUs [1]. They supply different types of resources to take advantage of the different kinds of parallelism appearing in real world applications (instructions level parallelism, data parallelism, task parallelism and so on). However, the development of software to efficiently exploit the full set of heterogeneous resources of modern computers is still a challenge. Several programming interfaces have to be used to develop software in this context, among them we highlight: MPI to develop applications on distributed platforms [20] and CUDA to accelerate programs through GPUs [13]. It means that programmers must combine several kinds of data structures and programming models to efficiently exploit the resources of the current heterogeneous computers. Development of

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 198–207, 2014.

software based on several parallel interfaces exhibits additional drawbacks due to its portability can be limited so, parallel implementations have to be tuned for each particular heterogeneous architecture. In this context, the main challenge is to determine an appropriate workload distribution because the computational burden assigned to every processing element should be related to its computational power. Therefore, it is necessary to have a detailed knowledge of both the computing resources and the algorithm [11].

PETSc is a well-known parallel library to solve PDEs, which contains several parallel versions supporting MPI, shared memory pthreads, and NVIDIA GPUs, as well as hybrid parallelism [3]. However, currently the PETSc routines based on CUDA are in progress. There are no benchmarking results published for the multi-GPU implementation using complex numbers.

This paper investigates a parallel approach for the solution of the 3D Helmholtz equation which combines the exploitation of the high regularity of the matrices involved in the numerical methods and the massive parallelism supplied by heterogeneous architecture of modern multi-GPU clusters [1].

In [17] we developed a CUDA version for solving the Helmholtz equation which takes advantage of the regularity of the matrices. It was used to solve Optical Diffraction Tomography (ODT) problems for 2D prototypes. From this experience we realized that the 3D extensions of these models need the support of modern multi-GPU clusters because of their high computational requirements.

The hybrid implementation is based on MPI and CUDA interfaces [7]. Due to the fact that CUDA and MPI produce portable code capable of exploiting both distributed architectures and multicores (if so many MPI processes as cores are launched), none specific interface to exploit the multicore structure has been included in the implementation. Our hybrid version launches MPI processes that exploit the CPU cores or the GPU devices of the multi-GPU cluster.

This way, both types of MPI processes can collaborate exploiting all the resources of the considered heterogeneous architecture. The main advantages of the hybrid implementation are related to two issues. On one hand, a hybrid version is capable of exploiting the different kinds of resources in multi-GPU clusters. This is due to the fact that CPU cores, despite being slower than the GPU devices, will be able to collaborate to the GPU device and accelerate the final computation by the overlapping the computation on CPUs and GPUs. Note that the workload of CPU processes will be significantly less than the workload of the GPU processes. On the other hand, our hybrid approach will overcome the memory limitation of the GPU devices because it can exploit the larger memory resources of multicore nodes. This way, GPUs will accelerate the computation but the size of the GPU workload is limited due to their smaller memories. CPU processes can collaborate with the GPU processes to exploit the large memory of multicore nodes.

The rest of this document is organized as follows. Section 2 describes the Helmholtz Equation. Section 3 is dedicated to explain the characteristics of our hybrid implementation of the 3D Helmholtz equation. Moreover the Biconjugate Gradient Method used as solver of the linear system of equation is shown. Section

4 analyzes the results of our hybrid version. Finally, in Section 5 the main results are outlined and future lines of research are presented.

2 The Problem

The Helmholtz equation can be applied to solve many problems related to steady-state oscillations (acoustical, thermal, electromagnetic, mechanical) [5,8]. The scalar approximation of the Helmholtz equation is defined by the following Partial Differential Equation (PDE):

$$(\nabla^2(\mathbf{r}) + k(\mathbf{r})^2)E(\mathbf{r}) = 0 \tag{1}$$

where ∇^2 is the Laplace operator; E is a complex scalar function defined at a spatial point $r = (x, y, z) \in \mathbb{R}^3$ and $k(\mathbf{r})$ can take real or complex values. This equation naturally appears from general conservation laws of physics and can be interpreted as a wave equation for monochromatic waves (wave equation in the frequency domain) where $k(\mathbf{r})^2$ is related to the wavenumber [19]. This equation can be numerically solved by means of the appropriated transformation based on Green's functions and the spatial discretization [12,19].

The Helmholtz equation is an example of linear elliptic Partial Differential Equation (PDE) which has been extensively studied [19]. Numerical solutions based on Finite Elements Method (FEM) have been developed for the discretization of this kind of differential operator [6]. It is well known that the "pollution effect" limits the reliability of the FEM solution of Helmholtz equation for high wavenumbers [2]. This numerical instability can be avoided only when a very fine mesh is used [4]. So, Helmholtz PDE leads to a very large linear system of equations specially for applications defined in a 3D space. Moreover, the use of FEM makes the matrix system sparse and regular [19,12] (see [19] for more details related to the boundary conditions of FEM).

3 Parallel Implementation of the Iterative Method

For high wave numbers the discretization results in a very large sparse system of linear equations $Ax = b$, where b indicates the independent term, x is the unknown vector and matrix $A \in \mathbb{C}^{N \times N}$. This linear system of equation cannot be solved with direct methods on current computers within reasonable time. The linear system is symmetric but indefinite, non-Hermitian and ill-conditioned which brings difficulties when solving with basic iterative methods [9]. As a result, iterative methods such as the Biconjugate Gradient method (BCG) can be used to effectively solve the Helmholtz Equation [21,10].

The BCG algorithm is an iterative method, which at every iteration includes two Sparse Matrix Vector products (SpMVs) and several dot products and saxpy operations (to analyze BCG in depth read [18,16]).

The SpMV is the operation with the highest computational cost. In [15], a specific storage format (named Regular Format), which takes advantage of the

regularity of the sparse matrix of Helmholtz Equation (A), has shown to considerably reduce the memory requirements of the SpMV operation and consequently to improve its performance. The regularities of A can be summarized as follows: A is a symmetric matrix; maximum of seven non-zero elements per row; non-zero elements are located at seven diagonals, where one is the main diagonal, two of them are the first lower and upper diagonals and four of them are located at $\pm D1$-th and $\pm D2$-th diagonals; all the elements of every lateral diagonal are equal (a, b and c); and the main diagonal is defined by a set of complex numbers. Bearing in mind these characteristics, the Regular Format proposed in this work consists of: one array, $A[]$ (complex) of dimension N (where N represents the dimension matrix); three integer values (a, b and c) to store all remaining entries; and two additional integer values ($D1$ and $D2$) which specify the displacements of the lateral diagonals with respect to the main diagonal.

Because of the large dimension of the problems in which the resolution of the Helmholtz equation is required, High Performance Computing is an indispensable tool. This is the reason we have considered the MPI paradigm in this work. MPI provides all the routines that are needed to break the tasks involved in the massive computational process into subtasks that can be distributed to the set of available cores for processing.

As aforementioned, in this work three different strategies for solving the 3D Helmholtz equation are discussed: CPU version to balance the workload among the multiprocessors, GPU version to balance among the GPU devices of the cluster and a heterogeneous implementation which exploits both CPU and GPU processes located into the multi-GPU cluster.

1. CPU version is based on the distribution of the problem among several cores located into different nodes. So, MPI paradigm is used for processing tasks on cores which can belong to nodes with shared or distributed memory. Note that for solving the SpMV operations, we account the regularity of the matrix A for establishing a strategy that allows the reduction of the number and size of the messages among the available processors. So, to compute the SpMV, each MPI process adds redundant elements to the chunk of the vector (at the beginning and at the end). The number of additional elements is fixed ($2{\cdot}D2$), hereinafter referred as "halo". These "halos" will be exchanged among the processors for updating the value of the vector before the computation of every SpMV. So, the use of redundant halos reduces the size of the messages and the number of communications among processes to compute the SpMV. This fact allows to exploit the regularity of A by means of the optimization of the communication pattern.
2. GPU version is based on the distribution of the problem among several GPU devices that can be located in the same or in a different node. Then, MPI paradigm is also used. In order to develop cluster applications that take advantage of GPU accelerators, it is necessary to select a GPU programming language and toolkit among the existing. In this work, a high abstraction subroutine library (CUBLAS [14]), which provides commonly used algorithms, has been considered. As it is well known, the key requirement for obtaining

effective acceleration from applications on GPU devices is the minimization of I/O between the host and the GPU. So, CPU-GPU intercommunications are strongly reduced using the strategy based on "halos" also applied in the CPU version.

3. Hybrid version has been designed to exploit all the computing resources of heterogeneous architectures. The idea behind this last implementation is the collaboration among CPU and GPU processes to accelerate the BCG algorithm by the exploitation of the full variety of available resources of multi-GPU clusters using also the "halos" strategy. Here it is important to highlight that a static workload balance scheduling has been considered because the workload of the application is known at compile time and constant during the execution. So, the distribution between the different processes can be done at compile time. This hybrid version presents two main advantages: (1) the parallel computation of GPU and CPU processes, despite CPU is slower than GPU, contributes to accelerate the algorithm execution. We have implemented an efficient collaboration between CPU and GPU by establishing an appropriate balance of the workload assigned to each device; and (2) the hybrid version can exploit the large memory resources of multicore nodes. This way, the memory limitation of the GPUs is overcome by the inclusion of CPU processes in the hybrid strategy.

4 Evaluation

This section shows the experimental results carried out to evaluate the performance of the hybrid version for solving the 3D Helmholtz equation. The hybrid approach is also compared to two different implementations: CPU version and GPU version.

Our experiments were performed on a cluster which is comprised of four compute nodes (Bullx R424-E3. Intel Xeon E5 2650 (16 cores every node) and 64 GB de RAM). Figure 1 depicts the platform. Each node is equipped with two Tesla M2075 GPUs managed by the CUDA driver included in the 5.0 toolkit. Note that each GPU device has 5.24 Gbytes of global memory (total $5.24 \times 8 = 41.92$ Gbytes), and that the size of the memory is one of the main factors which determines to what extend real world problems can be solved when only GPU platforms are used. Figure 1 depicts the structure of the cluster used in our experimental evaluations. Programs were compiled with mpicc and -O2 as optimization option.

The set of test sparse matrices is described in Table 2, where N represents the dimension of the sparse matrix, $D2$ identifies the number of elements of every "halo", and nnz the number of nonzero elements. Note that all the matrices have double-complex datatype and the average number of entries per row is 7 (for more details about the particular structure of the sparse matrix A, related to the discretization of the Helmholtz equation read [15]).

Firstly, a study of the runtime of the CPU and GPU approaches was carried out. Tables 1 and 2 show the experimental results where *Runt* identifies the

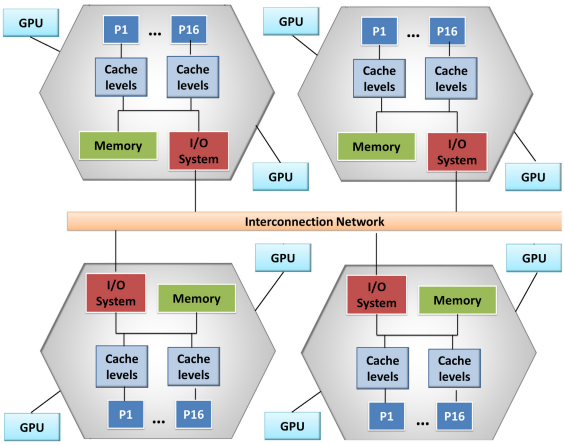

Fig. 1. Structure of the cluster, which is comprised of four compute nodes (16 cores every node) and a total of eight Tesla M2075 GPU devices

Matrix	N	$D2$	nnz
m_160^3	160^3	160^2	2.87E+07
m_200^3	200^3	200^2	5.60E+07
m_240^3	240^3	240^2	9.68E+07
m_280^3	280^3	280^2	1.54E+08
m_320^3	320^3	320^2	2.29E+08
m_360^3	360^3	360^2	3.27E+08
m_400^3	400^3	400^2	4.48E+08
m_440^3	440^3	440^2	5.96E+08
m_480^3	480^3	480^2	7.74E+08
m_520^3	520^3	520^2	9.84E+08

Fig. 2. Set of test matrices

total runtime in seconds of the BCG execution and $\%C$ represents the percentage of the communications penalties with respect to the total runtime (the remaining time is associated to the computation time). Note that CPU and GPU processes have been launched in different nodes since the number of MPI processes has been selected less or equal to the number of available cores on the cluster. To be more specific, the implementation using 2CPUs (2GPUs) has been launched using two nodes of the cluster and the implementations using 4CPUs (4GPUs), 8CPUs (8GPUs) and 16CPUs, using the four nodes of the cluster. Focusing our attention in Table 1, up to 16CPUs have been considered for the parallelization because using more CPU processors (e. g. 32CPUs) the size of the halos are relevant with respect to the local data and consequently the communication penalties deteriorate the performance.

Tables 1 and 2 clearly shown that the runtime ($Runt$) decreases as the number of CPU and GPU processes increases. It is due to the fact that the resolution of the Helmholtz equation using BCG is dominated by the computation. This fact is more evident for the CPU version with the low value of $\%C$ column with values which range from 0.43 to 11.32. In the GPU version, the communication penalties are larger (range from 4.48 to 33.94) and correspond to the additional communication between the CPU and the GPU and to the runtime reduction of the GPU. The value of $\%C$ is directly related to the dimension of the problem to solve, therefore, the bigger is the problem, the less impact of the communication time respect to the total runtime. One issue that can be seen in Table 2 is that the 3D Helmholtz equation could not be solved for the matrices with the largest size due to the limitations related to the global memory resources of the GPUs.

Table 1. Runtimes(s) of 1000 iterations of the Helmholtz BCG using the CPU version with 1, 2, 4, 8 and 16 processors (1CPU, 2CPUs, 4CPUs, 8CPUs and 16CPUs) (Column *Runt*). Column *%C* identifies the percentage of the total runtime that consume the communication processes for every execution.

	1CPU	2CPUs		4CPUs		8CPUs		16CPUs	
	Runt	Runt	%C	Runt	%C	Runt	%C	Runt	%C
m_160^3	214.32	108.12	1.34	54.64	1.95	28.18	3.84	16.34	11.32
m_200^3	377.98	190.80	1.45	98.30	2.31	50.67	4.66	31.44	6.50
m_240^3	719.37	357.22	1.15	179.84	4.08	92.96	2.88	54.38	4.84
m_280^3	1038.38	520.67	1.10	264.55	0.83	140.53	6.22	85.24	4.77
m_320^3	1750.79	920.56	0.68	458.92	2.02	230.57	3.59	127.79	4.93
m_360^3	2217.68	1111.75	1.53	560.56	3.74	295.42	3.70	177.96	4.23
m_400^3	3411.10	1687.70	0.85	847.62	1.69	436.52	2.01	250.03	3.26
m_440^3	4124.25	2095.97	2.56	1035.28	0.43	538.26	2.74	324.65	3.84
m_480^3	5929.36	2963.20	0.49	1489.04	1.20	752.24	1.93	429.89	4.16
m_520^3	-	3410.84	0.61	1743.64	0.81	878.22	2.07	538.24	2.89

Table 2. Runtimes(s) of 1000 iterations of the Helmholtz BCG using the GPU version with 2, 4 and 8 GPU devices (2GPUs, 4GPUs and 8GPUs) (Column *Runt*). Column *%C* identifies the percentage of the total runtime consumed by the communication processes for every execution.

	2GPUs		4GPUs		8GPUs	
	Runt	%C	Runt	%C	Runt	%C
m_160^3	13.89	9.50	7.90	19.40	5.03	33.94
m_200^3	26.60	7.55	14.80	15.84	9.12	30.22
m_240^3	45.73	6.45	24.85	12.28	14.60	25.38
m_280^3	72.49	5.28	38.43	10.44	22.18	22.68
m_320^3	108.15	4.48	55.57	9.32	31.90	19.23
m_360^3	-	-	78.95	8.38	44.22	16.90
m_400^3	-	-	107.42	6.99	58.04	14.85
m_440^3	-	-	-	-	76.73	13.79
m_480^3	-	-	-	-	97.86	11.66
m_520^3	-	-	-	-	-	-

The best results in terms of performance are always obtained by the GPU approach thanks to massive parallelism that GPU devices offer. So, the execution of the resolutions of the 3D Helmholtz Equation based on 1000 iterations of the BCG method using 2, 4 and 8 GPUs are faster than the execution using 2, 4, 8 and 16 CPU processors, with acceleration factors which range from 5.6 to 8.6 when we compare the same number of CPU and GPU processes (2CPUs-2GPUs, 4CPUs-4GPUs and 8CPUs-8GPUs). Moreover, the communications penalties of the GPU version are higher due to the additional communication GPU-CPU and viceversa in the "halos" swap.

In the hybrid version we have considered the executions with 4 and 8 GPU processes as these resources reached the best results in terms of performance

for the size of the considered test problems in the evaluation. Table 3 shows the runtime of 1000 iterations of the BCG method based on the 3D Helmholtz Equation using two hybrid configurations: 4GPUs+8CPUs and 8GPUs+8CPUs. The following notation is used, *Runt* identifies the total runtime in seconds of the execution and $GPU(s)$ and $CPU(s)$ represent the runtime in seconds of the GPU and CPU processes, respectively. Column F identifies the ratio between the workload assigned the CPU and the GPU. The workload assigned to a GPU process is F times higher compared to the workload of the CPU process. Different benchmarking processes have been carried out for this cluster and this algorithm to determine the computational burden associated to the hybrid version which provides the best workload balance. The value of F has been estimated by a preliminary benchmarking (not automatized) using values which range from 8 to 12 (these values are related to the acceleration factor of solving the same 3D Helmholtz problem in one GPU with respect to one GPU). Column $\%I$ represents the acceleration factor of the hybrid implementation versus the GPU version with 4 GPUs (for 4GPUs+8CPUs implementation) and 8 GPUs (for 8GPUs+8CPUs implementation). As aforementioned, the workload of the GPU version has to be larger than the workload of the CPU version but taking into account the memory limitation of the GPUs. Therefore, for problems of dimension 440^3, 480^3 and 520^3 with 4GPUs+8CPUs, the memory requirements of the four GPUs are not enough and the value of F is less than 8.

Table 3. Profiling of the resolution of the 3D Helmholtz Equation based on 1000 iterations of the BCG method using the hybrid version and two different configurations: 4GPUs+8CPUs and 8GPUs+8CPUs. Column *Runt* identifies the total runtime in seconds of the execution, $GPU(s)$ and $CPU(s)$ show the runtime of the GPU and the CPU, respectively, F column denotes the factors to balance the workload in the hybrid approach and, finally, $\%I$ represents the acceleration factor of the hybrid implementation versus the GPU version with 4 GPUs (for 4GPUs+8CPUs implementation) and 8 GPUs (for 8GPUs+8CPUs implementation).

	4GPUs+8CPUs					8GPUs+8CPUs				
	Runt	GPU(s)	CPU(s)	F	%I	Runt	GPU(s)	CPU(s)	F	%I
m_160^3	7.24	5.29	3.31	11	8.45	4.81	3.03	2.48	10	4.24
m_200^3	13.66	10.74	7.49	12	7.73	8.62	5.80	4.22	11	5.45
m_240^3	21.63	17.47	16.17	10	12.96	14.4	9.66	10.09	11	1.40
m_280^3	33.44	27.36	26.75	8	12.98	21.42	15.00	13.67	10	3.43
m_320^3	49.13	41.74	37.95	12	11.58	30.52	22.18	18.58	11	4.35
m_360^3	69.73	60.10	50.61	10	11.67	41.73	29.80	31.36	10	5.61
m_400^3	93.75	83.41	68.05	11	12.73	55.80	43.70	42.81	9	3.87
m_440^3	131.99	101.85	123.08	7	-	73.35	56.62	56.58	11	4.41
m_480^3	329.82	102.76	306.90	3	-	93.38	75.65	73.67	9	4.58
m_520^3	473.82	15.45	459.82	2	-	123.49	97.83	98.14	9	-

As can be observed in Table 3, the runtimes of the CPU and GPU processes are well balanced. Note that in all experiments, 4 GPU or 8 GPU cards are

exploited and the runtime decreases as the number of GPU processes increases. It is due to the fact that CPU processes collaborate with the GPU processes to accelerate the computation. Note that the workload balance is not optimal for the matrices m_440^3, m_480^3 and m_520^3 due to the memory limitations of the GPU. So, although the workload of the CPU processes is high, the CPU memory resources allow to extend the dimension of problem with light penalties of performance. For the considered dimension of the problem to solve (from $N = 160^3$ to $N = 520^3$), the approach with 4GPUs+8CPUs reaches the best results in terms of performance compared to the 4GPUs version (acceleration factors range from 8.45 to 12.73). It results evident that this kind of platforms can only be efficiently exploited for very large problems, even larger than the instances considered in this work. It is also clear that the hybrid platform is appropriate to overcome the GPU memory limitations.

5 Conclusions

This paper has described the implementation of a parallel solution for the 3D Helmholtz equation which combines the exploitation of the high regularity of the matrices involved in the numerical methods and the massive parallelism supplied by heterogeneous architecture of modern multi-GPU cluster. Experimental results have shown that this approach outperforms both, the CPU and the GPU approaches when several CPU cores collaborate with the GPU devices. However, we only obtain large acceleration factors using more GPUs when the size of the problem is large enough.

As aforementioned, the difficulty having the workload well-balanced is related to the appropriate distribution of the workload between the CPU and GPU processes, so our current on-going work involves to automatize the benchmarking process to determine the most suitable factor F to exploit the specific architecture considered. Furthermore, another future work will be focused on the development of a hybrid version MPI-OpenMP on the multi-GPU cluster.

Acknowledgments. G. Ortega is a fellow of the Spanish FPU programme. Work partially supported by the Spanish Ministry of Science (TIN2008-01117, TIN2012-37483) and J. Andalucía (P10-TIC-6002, P011-TIC7176), partially financed by the European Reg. Dev. Fund (ERDF).

References

1. Top 500 List (June 2013), `http://www.top500.org/lists/2013/06/`
2. Babuska, I.M., Sauter, S.A.: Is the pollution effect of the FEM Avoidable for the Helmholtz Equation considering high wave numbers? SIAM Rev. 42(3), 451–484 (2000)
3. Balay, S., et al.: PETSc Users Manual. Revision 3.3
4. Bao, G., Wei, G.W., Zhao, S.: Numerical solution of the Helmholtz equation with high wave numbers. Int. J. Numer. Meth. Eng. 59, 389–408 (2004)

5. Dautray, R., Lions, J.L.: Mathematical analysis and numerical methods for science and technology. Springer, Berlin (1990)
6. Ihlenburg, F., Babuska, I.: Solution of Helmholtz problems by knowledge-based FEM. CAMES 4, 397–415 (1997)
7. Jacobsen, D.A., Thibault, J.C., Senocak, I.: An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters. In: Proc. of the 48th American Institute of Aeronautics and Astronautics (AIAA) Aerospace Science Meeting, Floria, USA (2010)
8. Junger, M.C., Feit, D.: Sound, Structures, and Their Interaction, 2nd edn. The MIT Press (1986)
9. Knibbe, H., Oosterlee, C.W., Vuik, C.: GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. J. Comput. Appl. Math. 236(3), 281–293 (2011)
10. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Nat. Bur. Stand. 45, 255–282 (1950)
11. Lastovetsky, A.L.: Special issue of journal of parallel and distributed computing: Heterogeneity in parallel and distributed computing. J. Parallel Distrib. Comput. 72(10), 1397 (2012)
12. Duraiswami, R., Gumerov, N.A.: Fast Multipole Methods for the Helmholtz Equation in Three Dimensions. Elsevier Science (2004)
13. NVIDIA. CUDA Programming guide. Version 4 (2011)
14. NVIDIA. Du-06702-001_v5.5 CUBLAS user guide. Technical report (July 2013)
15. Ortega, G., Garzón, E.M., Vázquez, F., García, I.: Exploiting the regularity of differential operators to accelerate solutions of PDEs on GPUs. In: Proc. of CMMSE, Benidorm, Spain, June 26-30, pp. 908–917 (2011)
16. Ortega, G., Garzón, E.M., Vázquez, F., García, I.: The BiConjugate gradient method on GPUs. J. Supercomput. 64, 49–58 (2013)
17. Ortega, G., Lobera, J., Arroyo, M.P., García, I., Garzón, E.M.: High Performance Computing for Optical Diffraction Tomography. In: Proc. of HPCS, Madrid, Spain, July 2-6, pp. 195–201 (2012)
18. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM (April 2003)
19. Sadiku, M.N.O.: Numerical Techniques in Electromagnetics, 2nd edn. CRC Press (July 2000)
20. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI-The Complete Reference, 2nd revised edn. The MPI Core, vol.1. MIT Press, Cambridge (1998)
21. Van der Vorst, H.A., Ciarlet, P.G., Iserles, A., Kohn, R.V., Wright, M.H.: Iterative Krylov Methods for Large Linear Systems. Cambridge Univ. Press (2003)