Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 61 (2013) 87 – 93

Procedia Engineering

www.elsevier.com/locate/procedia

Parallel Computational Fluid Dynamics Conference (ParCFD2013)

Direct numerical simulation of incompressible flows on unstructured meshes using hybrid CPU/GPU supercomputers

G.Oyarzun[a], R.Borrell[b], A.Gorobets[a,c], O. Lehmkuhl[a,b], A.Oliva[a,*]

[a]*Heat and Mass Transfer Technological Center, ETSEIAT, Technical University of Catalonia, C/ Colom 11, 08222, Terrassa, Spain*
[b]*Termo Fluids, S.L., c/ Mag Colet 8, 08204, Sabadell (Barcelona), Spain*
[c]*Keldysh Institute of Applied Mathematics RAS, 4A, Miusskaya Sq., Moscow, 125047, Russia*

Abstract

This paper describes a hybrid MPI-CUDA parallelization strategy for the direct numerical simulation of incompressible flows using unstructured meshes. Our in-house MPI-based unstructured CFD code has been extended in order to increase its performance by means of GPU co-processors. Therefore, the main goal of this work is to take advantage of the current hybrid supercomputers to increase our computing capabilities. CUDA is used to perform the calculations on the GPU devices and MPI to handle the communications between them. The main drawback for the performance is the slowdown produced by the MPI communication episodes. Consequently, overlapping strategies, to hide MPI communication costs under GPU computations, are studied in detail with the aim to achieve scalability when executing the code on multiple nodes.

Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)
Keywords: CPU/GPU hybrid supercomputers, direct numerical simulation, Navier-Stokes equations, MPI, CUDA.

Nomenclature

$\boldsymbol{u}$	velocity vector
Re	dimensionless Reynolds number
$\boldsymbol{f}$	body force vector
p	pressure

Subscripts

k	refers to the control volume itself
nb	refers to the control volume face-neighbor

Superscripts

n	refers to discrete time level
*	refers to predictor

1. Introduction

The advent of Graphic Processing Units (GPUs) as tools capable to solve large amounts of calculations is now a consolidated reality. Good results attained on solving problems that exhibit a fine-grained parallelism have captured

* Corresponding author. Tel.: +34-93-739-81-92 ; fax: +34-93-739-89-20.
E-mail address: cttc@cttc.upc.edu

Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)
doi:10.1016/j.proeng.2013.07.098

the interest of the High Performance Computing (HPC) community prompting the research in this field. Meanwhile, new programming models such as the Compute Unified Device Architecture (CUDA) [1] developed by NVIDIA, or the Open Computing Language (OpenCL) [2] developed by the Khronos group have facilitated exploiting the computational power of the GPU devices. However, achieving good performance is not trivial, and strongly depends on the nature of the problem being solved and a proper understanding of the GPU architecture.

On the other hand, direct numerical simulation (DNS) constantly demands more and more computing power, arising the interest of using the most recent technologies. Over the last years, several authors have attempted to simulate the Navier-Stokes equations using GPUs. The first Navier-Stokes code for incompressible flows on structured grids was presented in [3] using a high-level shading language, previous to the creation of CUDA. A more recent 3D extension of this work can be found in [4], focusing on a simplified real-time simulation of the flows for rendering processes. With the introduction of CUDA, a 3D Finite Difference Computation was first described in [5], structured meshes were used creating a stencil that can exploit the characteristics of new GPU devices, also a first overlapping approach to utilize multi-GPUs was presented. A detailed implementation of these concepts applied to incompressible flows was proposed in [6]. Meanwhile, similar approaches have been applied to unstructured meshes using a single-GPU in [7, 8, 9]. Finally, a novel DNS simulation using unstructured meshes and multi-GPU platforms is explained in [10, 11].

In our previous work we focused on solving the Poisson equation, because it is the most computing intensive part of the algorithm, this fact has been claimed by other authors as well [10]. However, the rest of the time-integration sums up still 30-40% of the total execution time so, in order to achieve a good overall acceleration, is also desirable to obtain performance on these parts. In this paper we focus on porting to GPUs the whole time integration step of a DNS simulation code for incompressible flows on unstructured meshes. MPI is used to perform the communication between nodes, while CUDA is adopted as the programming model to perform calculations in the GPU. Scalability is our big objective, and to achieve it overlapping techniques are implemented.

The rest of this paper is organized as follows: Section 2 describes the governing equations and the math model to solve the incompressible Navier-Stokes equations. In Section 3 the CUDA programming model and the hybrid parallelization strategy are explained. In Section 4 are shown some preliminary results. Finally, further work is outlined in Section 5.

2. Governing equations and Math Model

The dimensionless form of the Navier-Stokes equations for incompressible flows reads:

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0, \tag{1}$$

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = Re^{-1}\boldsymbol{\nabla}^2\boldsymbol{u} - \boldsymbol{\nabla} p + \boldsymbol{f}, \tag{2}$$

TermoFluids code [13] has been used as a software platform to perform our studies. The equations are discretized on an unstructured grid with the finite-volume symmetry-preserving method [14]. For the temporal discretization, a second-order explicit one-leg scheme is used. Finally, the pressure-velocity coupling is solved by means of a fractional step projection method [12]. In short, a predictor velocity, $\boldsymbol{u}^p$, is explicitly evaluated without considering the contribution of the pressure gradient in Equation 2, and then the Poisson pressure correction equation is solved in order to meet the incompressibility constraint:

$$\Delta \boldsymbol{p}^{n+1} = \nabla \boldsymbol{u}^p, \tag{3}$$

where $\boldsymbol{p}^{n+1}$ is the pressure correction field at time-step $n+1$. Once the pressure correction field is obtained, is used to calculate the velocities at the time-step $n+1$ by means of the equation:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^p - \nabla p^{n+1} \tag{4}$$

The time integration process is outlined in Algorithm 1. Further details on the numerical algorithm, the math model and the code functionalities can be found in [13, 15, 16, 17].

Algorithm 1 Time integration step

1: Evaluate the predictor velocities
2: Solve the Poisson equation
3: Correct the velocities

Considering a discretization over a control volume k with a set of faces defined as $F(k)$, then the predictor velocities stands as:

$$u_k^p = u_k^n - \Delta t \left[\sum_{f \in F(k)} (u_{nb}^n + u_k^n) \frac{M_f}{2V_k} - \nu \sum_{f \in F(k)} (u_{nb}^n - u_k^n) \frac{A_f}{\delta d_f V_k} \right] \tag{5}$$

where the subscript nb refers to the control volume of the face-neighbor, V_k is the volume of the control volume k, A_f refers to the surface of the face f, $\hat{n}_f$ is the outward-unit normal to the face f, M_f is the mass flow on the face f and is calculated as $M_f = u_f \cdot \hat{n}_f \cdot A_f$, and δd_f is the normal-projected distance between the centroids of the control volumes k and nb. Excepting the mass flow, all the coefficients multiplying the velocities remain constant during the overall execution. Since we work with unstructured meshes, the stencil generated per each control volume lacks of the structure needed to use an efficient 3D stencil as the one explained in [5]. Then, we consider the following matricial approach of the Equation 5:

$$\boldsymbol{u}^p = \boldsymbol{u}^n - \Delta t \, (\boldsymbol{C}(\boldsymbol{u}^n)\boldsymbol{u}^n + \boldsymbol{D}\boldsymbol{u}^n) \tag{6}$$

where D is a constant sparse matrix containing the diffusion coefficients and $\boldsymbol{C}(\boldsymbol{u}^n)$ is a variable sparse matrix containing the convection coefficients, which is re-evaluated at each iteration according to the mass flow. Thus the calculation of the predictor velocities can be reduced to two Sparse Matrix Vector multiplications (SpMV).

Applying the same discretization to the Equation 3 yields into a discrete Poisson equation that reads:

$$\sum_{f \in F(k)} (p_{nb}^{n+1} - p_k^{n+1}) \frac{A_f}{\delta d_f} = \frac{1}{\Delta t} \sum_{f \in F(k)} \rho \frac{(u_k^p + u_{nb}^p)}{2} \cdot \hat{n}_f A_f \tag{7}$$

Considering the matricial notation, the equation reads:

$$\boldsymbol{A}\boldsymbol{p}^{n+1} = \frac{1}{\Delta t} (\boldsymbol{B}\boldsymbol{u}^p) \tag{8}$$

where $\boldsymbol{B}$ is a constant sparse matrix.

The characteristics of the Poisson system, defined by a symmetric and positive definite matrix $\boldsymbol{A}$, makes suitable the use of the Preconditioned Conjugate Gradient (PCG) algorithm to solve it.

Finally, the discrete velocity correction equation over the control volume k is written as:

$$u_k^{n+1} = u_k^p - \Delta t \sum_{f \in F(k)} (p_k^{n+1} + p_{nb}^{n+1}) \frac{\hat{n}_f A_f}{2\rho V_k} \tag{9}$$

With the matricial approach is transform into:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^p - \Delta t (\boldsymbol{E}\boldsymbol{p}^{n+1}) \tag{10}$$

being $\boldsymbol{E}$ a constant sparse matrix.

It is important to notice that matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ and $\boldsymbol{E}$ share the same structure (columns and rows where the non-zero coefficients are located are the same), so its storage can be compressed. From our previous work[11], we concluded that the Preconditioned Conjugate Gradient time execution is dominated by the Sparse Matrix Vector multiplication. Moreover from the above discretization, it is possible to conclude that the time integration step is dominated by the SpMV as well. Therefore, the performance of our CPU/GPU algorithm is mostly determined by the performance of the SpMV when using multiple GPUs.

3. GPU architecture and Parallelization Model

A GPU device consist of a set of streaming multiprocessors (SMs) that employ a Single Instruction Multiple Threads (SIMT) model to create, schedule, manage and execute hundreds of threads during the execution of a program. The threads are grouped into warps, each thread of a warp runs the same instructions concurrently on a SM. A CUDA program is composed of a CPU code, so called host code, that launches tasks on a GPU device and a GPU code, so called kernel code, that runs on the device. The kernel functions are executed on the GPU by a large set of parallel threads. GPU devices have no common RAM memory space with CPUs nor with each other. Therefore, data transfer operations via PCI-Express are required to communicate between the devices. NVIDIA devices of compute capability 1.1 and higher can overlap the copies between GPU and CPU with kernel execution. Concurrent kernels can be executed as well, the concurrency is managed through streams. A stream is defined as a sequence of commands executed in order. Then multiple streams, executed in parallel, are used to achieve the concurrency. Further details about the GPU architecture and the CUDA programming model can be found in the official documentation [1].

The execution scheme adopted for DNS on hybrid clusters is an extension of the flowchart proposed by [8] for single-GPUs. Our particular flowchart is depicted in Figure 1. The oval boxes correspond to the GPU kernels, while the rest of the functions are executed in the CPU.

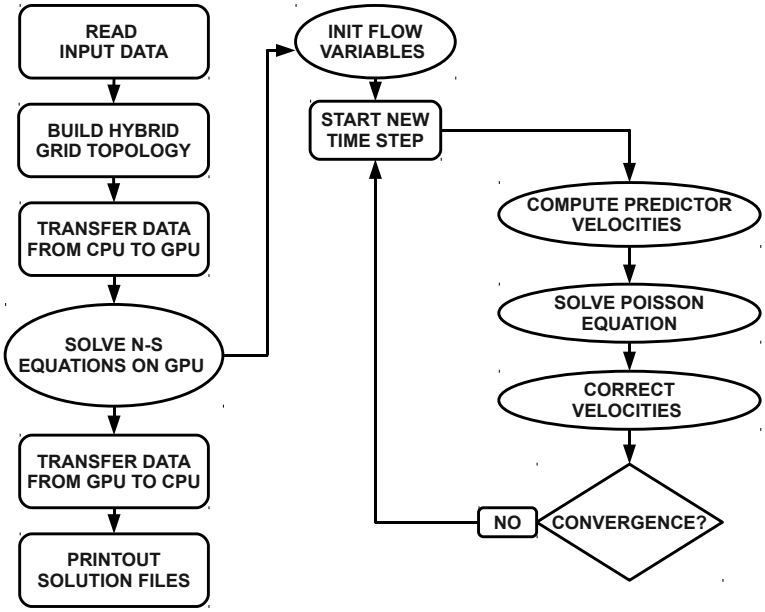

Fig. 1. Flowchart proposed to perform the time integration step.

In our approach, the MPI group of processes consists of as many processes as GPU-devices in the computing nodes engaged. The underlying MPI parallelization is based on a domain decomposition approach. Therefore, the initial mesh $\mathcal{M}$, that is used for the discretization, is divided into P non-overlapping sub-meshes $\mathcal{M}_0, ..., \mathcal{M}_{P-1}$ by means of a graph partitioning tool such as METIS library [18]. For each MPI process, the corresponding unknowns of the

system can be divided into different sub-sets: (1) *Owned* unknowns are those associated to the nodes of $\mathcal{M}_i$; (2) *Inner* unknowns are the owned unknowns that are coupled only with other owned unknowns; (3) *Interface* unknowns are those owned unknowns which are coupled with non-owned (external) unknowns; and finally (4) *Halo* unknowns are those external unknowns (that belong to other processes) which are coupled with owned unknowns of the MPI process considered.

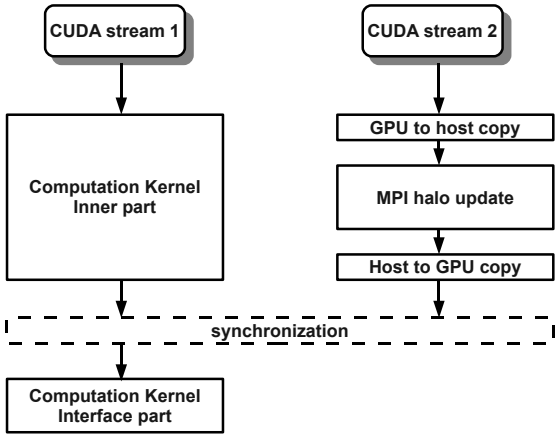

Fig. 2. Two stream concurrent execution model

Considering our parallelization model, each MPI communication process requires three steps: i) download data from the GPUs to the host memory; ii) transfer data between MPI processes; iii) upload data back to the GPUs. In order to make the code scalable when using hybrid supercomputers, a two stream execution model is used to overlap, whenever is possible, the MPI communications and data transfers with GPU computations. This computing scheme is represented in Figure 2.

4. Preliminary Results

Special attention was paid in the optimal computation of the sparse matrix vector multiplication because, as was shown in Section 2, is the dominant part of the iterations. Details of this implementation are explained in [11]. Numerical experiments were carried out on CURIE hybrid nodes [19] consisting of: two Intel Xeon E5640 (Westmere) 2.66 GHz quad-core CPUs and two Nvidia M2090 GPUs.Interconnection between nodes are performed via an Infiniband QDR network. CUDA Toolkit 4.0 was used to implement the GPU kernels. Unstructured tetrahedral meshes of different sizes were used to discretize the Poisson equation. For all the meshes, the matrix resulting from the discretization has in average 4.9 entries per row, being tremendously sparse.

In Figure 4 is shown the acceleration attained with the MPI-CUDA implementation of the SpMV algorithm when is executed in only one GPU respect to the execution in only one CPU-core. These tests were executed for meshes ranging from 100,000 to 400,000 control volumes (CV). It is possible to observe a positive trend of the speedup, i.e. increasing the mesh size provokes a raise of the speedup as well. Up to an 12.5X speedup and 7.3 Gflops are obtained for the largest case. This results are consistent with those reported in the works [20, 21] for similar matrices.

Additionally, a week speedup test is shown in Figure 4 for both the multi-CPU and multi-GPU implementations of the PCG. Recalling the cluster configuration, each hybrid node consist of 8 CPU-cores and 2 GPUs, the mesh size is kept constant at 400,000 CV per GPU (i.e. 100,000 CV per CPU-core). The Non-overlapped (NSpMV) and Overlapped (OSpMV) implementation of the multi-GPU SpMV are depicted as well. The plots shows nearly ideal scaling in all the implementations when increasing the number of nodes from 2 to 8. The communication hiding

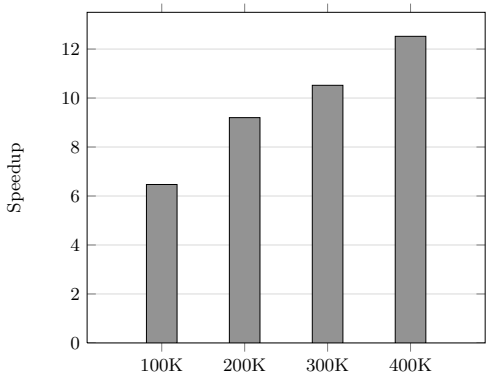

Fig. 3. Speedup of the single GPU respect to the single CPU execution for the SpMV considering different mesh sizes.

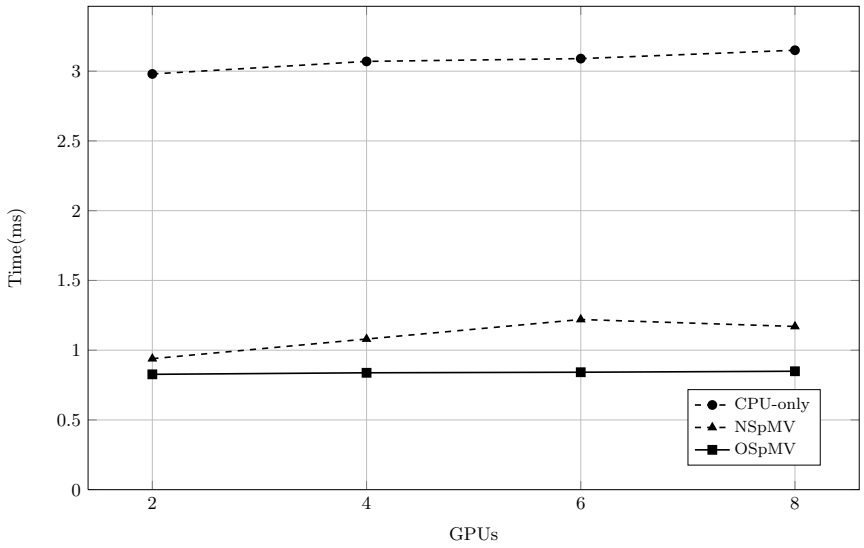

Fig. 4. Illustration of the weak speedup for a SpMV iteration loading 400,000 CV per GPU.

can be observed as the OSpMV outperforms the rest of implementations in all the cases. The difference between the execution times of the OSpMV and the pure-CPU implementation is kept constant at around 3.6X.

5. Further work

The hybrid parallelization of the SpMV can be considered accomplished. Reasonable results have been obtained, with an overall speedup up to 3.6 when comparing 8 GPUs against 32 CPU-cores. This results encourage us to develop a DNS implementation for unstructured meshes fully running on GPUs. To achieve this objective the hybrid implementation of the SpMV must be applied to all the steps of the Algorithm 1.

Another challenge relays on developing an specific storage format in order to achieve coalesced data access, and thus increasing the acceleration obtained from the GPU co-processors.

Acknowledgments

The computations presented in the paper have been carried out using the PRACE Research Infrastructure supercomputer CURIE, based in France at the Tres Grand Centre de Calcul (TGCC). The work has been partially financially supported by the RFBR, research project No. 12-01-33022 and by the Ministerio de Ciencia e Innovacin, Spain (ENE2010-17801). The authors thankfully acknowledge these institutions.

References

[1] NVIDIA-Corporation, NVIDIA CUDA C Programming Guide (2007).
[2] Khronos Group, OpenCL 1.2 Specification (2009).
[3] Harris M.J.,2004, Fast Fluid Dynamics Simulation on the GPU,in: GPU Gems, Addison Wesley, 2004, pp. 637-665 (Chapter 38).
[4] Crane K.,Llamas I., Tariq S.,Real-Time Simulation and Rendering of 3D Fluids,in: GPU Gems 3, Pearson Education, Inc.: Boston, 2007,pp. 633-675 (Chapter 30).
[5] Micikevicius P., 3D Finite Difference Computation on GPUs using CUDA, in: Proceeding of 2nd Workshop on General Purpose Processing on Graphics Processing Units,2009.
[6] Thibault J., Senocak I., CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows, in: 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition,No. AIAA 2009-758, 2009.
[7] Corrigan A., Camelli F., Lhner R., Running Unstructured Grif Based CFD Solvers on Modern Graphics Hardware, in: 19th AiAA Computational Fluid Dynamics, San Antonio TX, 2009.
[8] Kampolis I., Trompoukis X., Asouti V., Giannakoglou K., CFD-based analysis and two-level aerodynamic optimization on graphic processing units, in: Computer Methods in Applied Mechanics and Engineering,2010, vol 199, pp.712-722.
[9] Asouti V., Trompoukis X., Kampolis I., Giannakoglou K., Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphic Processing Units, in: International Journal for Numerical Methods in Fluids,2010, vol 67, pp. 232-246.
[10] Khakeh-Saeed A., Perot J.B., Direct numerical simulation of turbulence using GPU accelerated supercomputers, in: Journal of Computational Physics, 2013,vol 235, pp. 241-257.
[11] Oyarzun G.,Borrell R., Gorobets A., Oliva A., MPI-CUDA Sparse Matrix-Vector Muliplication for the Conjugate Gradient with an Approximate Inverse Preconditioner, in: submitted under revision Computer and Fluids Journal, Elsvier,2013.
[12] Chorin A.J., Numerical Solution of the Navier-Stoles Equations, in: Journal of Computational Physics, 1968, vol 22, pp. 745-762
[13] O. Lehmkuhl, C. Pérez-Segarra, R. Borrell, M. Soria, A. Oliva, TERMOFLUIDS: A new Parallel unstructured CFD code for the simulation of turbulent industrial problems on low cost PC Clusters, in: Parallel Computational Fluid Dynamics, Elsevier, Antayla (Turkey), 2007.
[14] R. W. C. P. Verstappen, A. E. P. Veldman, Symmetry-Preserving Discretization of Turbulent Flow, Journal of Computational Physics 187 (2003) 343–368.
[15] O. Lehmkuhl, R. Borrell, I. Rodríguez, C. Pérez-Segarra, A. Oliva, Assessment of the symmetry-preserving regularization model on complex flows using unstructured grids, Computers and Fluids 60 (2012) 108–116.
[16] R. Borrell, O. Lehmkuhl, F. X. Trias, A. Oliva, Parallel direct Poisson solver for discretisations with one Fourier diagonalisable direction, Journal of Computational Physics 230 (12) (2011) 4723–4741.
[17] G. Colomer, R. Borrell F. X. Trias, I. Rodrguez, Parallel algorithms for Sn transport sweeps on unstructured meshes, Journal of Computational Physics 232 (1) (2013) 118–135.
[18] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[19] Tgcc curie supercomputer, 2011, http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
[20] Bolz J., Farmer I., Grinspun E., Schrder P., Sparse matrix solvers on the GPU: conjugate gradients and multigrid, in: ACM Transactions on Graphics, 2003, vol 22, pp. 917-924.
[21] Michels D., Sparse-Matrix-CG-Solver in CUDA, in: Proceedings of CESCG 2011, The 15th Central European Seminar on Computer Graphics, 2011.