Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 61 (2013) 81 – 86

Procedia Engineering

www.elsevier.com/locate/procedia

Parallel Computational Fluid Dynamics Conference (ParCFD2013)

An OpenCL-based parallel CFD code for simulations on hybrid systems with massively-parallel accelerators

Andrey Gorobets[a,b], F. Xavier Trias[a], Assensi Oliva[a,*]

[a] *University of Catalonia, Heat and Mass Transfer Technological Center, ETSEIAT, C/Colom 11, Terrassa, 08222, Spain*
[b] *Keldysh institute of applied mathematics of RAS, 4A, Miusskaya Sq., Moscow,125047,Russia*

Abstract

A parallel finite-volume CFD algorithm for modeling of incompressible flows on hybrid supercomputers is presented. It is based on a symmetry-preserving high-order numerical scheme for structured meshes. A multilevel approach that combines different parallel models is used for large-scale simulations on computing systems with massively-parallel accelerators. MPI is used on the first level within the distributed memory model to couple computing nodes of a supercomputer. On the second level OpenMP is used to engage multiple CPU cores of a computing node. The third level exploits the computing potential of massively-parallel accelerators such as GPU (Graphics Processing Units) of AMD and NVIDIA, or Intel Xeon Phi accelerators of the MIC (Many Integrated Core) architecture. The hardware independent OpenCL standard is used to compute on accelerators of different architectures within a general model for a combination of a central processor and a math co-processor.

Keywords: OpenCL, MPI, OpenMP, GPU, Parallel CFD, finite-volume, structured mesh

Nomenclature

$\boldsymbol{u}$	velocity vector
p	pressure
$\boldsymbol{f}$	body force vector
$\mathcal{D}$	diffusion operator
D	discrete diffusion operator
$\mathcal{C}$	convection operator
C	discrete convection operator
M	discrete divergence operator
L	discrete Laplacian perator
Pr	Prandtl number
Ra	Rayleigh number
Re	Reynolds number
Greek symbols	
θ	temperature
ϕ	a scalar field
Ω	matrix of the control volume sizes
Subscripts	
h	discrete mesh function

* Corresponding author.
E-mail address: oliva@cttc.upc.edu

1. Introduction

The CFD code considered in this work is designed for modelling of incompressible flows on hybrid supercomputers using structured meshes. It is a multi-purpose software framework for development of new models for simulations of incompressible turbulent flows with conjugated heat transfer. On the one hand, it serves as a playground for testing and tuning of new turbulence and regularization models. Examples of models that have been developed on a base of this computing software can be found in [1].

On the other hand, it can be used as a DNS code that can run on tens of thousands of CPU cores [2] constructing a reliable validation basis for the development of new modeling approaches. The underlying algorithm is based on the finite-volume symmetry-preserving spatial discretization [3]. Its implementation allows whatever order of accuracy, however, the schemes of 2-nd and 4-th orders are in use. The core of the algorithm is a Fourier-based Poisson solver for cases with one periodic direction that combines direct and iterative approaches.

Achieving maximal performance with modern HPC systems is of high priority for the both aforementioned purposes. A large-scale DNS requires huge computing power and demands higher levels of scalability and parallel efficiency. For the development of models the performance issues are also important since the process of validation and tuning requires performing plenty of simulations with different configurations and parameters. For this reason the algorithm is subjected to the never-ending upgrade chasing the rapid evolution of the supercomputer architectures.

Today there is a twofold trend in the evolution of HPC systems. Firstly, the number of cores in systems tends to increase. This requires highly scalable algorithms and motivates the use of a more complex MPI+OpenMP parallel model. Secondly, the high computing potential of new hybrid systems relies on massively-parallel accelerators that demand even more complex parallel and computing models that combine principally different kinds of parallelism, namely, MIMD (multiple instruction multiple data) and SIMD (single instruction multiple data).

This work is focused on using modern hybrid systems based on graphics processing units (GPU) of AMD and NVIDIA or different kinds of accelerators, for instance, Intel Xeon Phi accelerators. The hardware independent computing standard OpenCL (Open Computing Language) [4] is employed to compute on accelerators of different architectures.

2. Overview of the algorithm and the mathematical model

2.1. Governing equations

The velocity field, $\boldsymbol{u}$, and the temperature, θ, are governed by the dimensionless incompressible Navier-Stokes (NS) equations coupled with the temperature transport equation:

$$\begin{aligned} \boldsymbol{\nabla} \cdot \boldsymbol{u} &= 0, & (1)\\ \partial_t \boldsymbol{u} + \mathcal{C}(\boldsymbol{u}, \boldsymbol{u}) &= \mathcal{D}\boldsymbol{u} - \boldsymbol{\nabla} p + \boldsymbol{f}, & (2)\\ \partial_t \theta + \mathcal{C}(\boldsymbol{u}, \theta) &= Pr^{-1}\mathcal{D}\theta, & (3) \end{aligned}$$

where the convective term is defined by $\mathcal{C}(\boldsymbol{u}, \phi) = (\boldsymbol{u} \cdot \nabla)\phi$. In the case of natural convection the density variations are not neglected and the diffusive term is defined as $\mathcal{D}\boldsymbol{u} = PrRa^{-1/2}\boldsymbol{\nabla}^2\boldsymbol{u}$ and the body force vector is given by $\boldsymbol{f} = (0, 0, Pr\theta)$ (Boussinesq approximation), where Ra is the Rayleigh number. In the forced convection case (density variations neglected) the temperature becomes a passive scalar, the body force vector $\boldsymbol{f} = 0$, and the diffusive term is $\mathcal{D}\boldsymbol{u} = Re^{-1}\boldsymbol{\nabla}^2\boldsymbol{u}$, where Re is the dimensionless Reynolds number.

The incompressible NS equations (2) are discretized on a staggered Cartesian grid using symmetry-preserving discretizations [3]. The operator-based finite-volume spatial discretization of these equations reads

$$\Omega \frac{d\boldsymbol{u}_h}{dt} + \mathsf{C}(\boldsymbol{u}_h)\boldsymbol{u}_h + \mathsf{D}\boldsymbol{u}_h - \mathsf{M}^t\boldsymbol{p}_h = \boldsymbol{0}_h, \quad (4)$$

where M is the discrete divergence operator. The discrete incompressibility constraint is given by $\mathsf{M}\boldsymbol{u}_h = \boldsymbol{0}_h$. The diffusion matrix, D, represents the integral of the diffusive flux through the faces of control volumes. The diagonal matrix, Ω, is the scaling by the sizes of the control volumes. The convection matrix, $\mathsf{C}(\boldsymbol{u}_h)$, is skew-symmetric, *i.e.* $\mathsf{C}(\boldsymbol{u}_h) = -\mathsf{C}^t(\boldsymbol{u}_h)$. The time evolution of the temperature, θ_h, is discretized in the same way. Then, for the temporal discretization, a second-order explicit one-leg scheme is used for both the convective and diffusive terms. Further details on the time integration method can be found in [5]. Finally, the pressure-velocity coupling is then solved by means of a classical fractional step projection method [6]: a predictor velocity, $\boldsymbol{u}_h^p$, is explicitly evaluated without considering the contribution of the pressure gradient. Then, by imposing the incompressibility constraint, $\mathsf{M}\boldsymbol{u}_h^{n+1} = \boldsymbol{0}_h$, it leads to a Poisson equation for $\boldsymbol{p}_h^{n+1}$ to be solved once at each time-step, $\mathsf{L}\boldsymbol{p}_h^{n+1} = \mathsf{M}\boldsymbol{u}_h^p$, where the discrete Laplacian operator, $\mathsf{L} = -\mathsf{M}\Omega^{-1}\mathsf{M}^t$, is represented by a symmetric negative semi-definite matrix.

2.2. Outline of the algorithm

The discretization of the Poisson equation results in a linear system to be solved: $\mathbf{A}^{3D}\mathbf{x}^{3D} = \mathbf{b}^{3D}$, where $\mathbf{A}^{3D} = -\mathsf{L} \in \mathbb{R}^{N\times N}$ and $N = N_x \times N_y \times N_z$ is the total number of nodes ($3D$ means each unknown is coupled with its neighbors in the three spatial directions). The original 3D system is decomposed using FFT into the set of independent 2D systems: $\hat{\mathbf{A}}_i^{2D}\hat{\mathbf{x}}_i^{2D} = \hat{\mathbf{b}}_i^{2D}$, where $i = 1, \cdots, N_x$.

The 2D systems that correspond to lower Fourier frequencies are worse conditioned. The strategy of the Poisson solver is to use the direct parallel Schur complement based method [7] to solve one or several worst conditioned systems. A Preconditioned Conjugate Gradient (PCG) method is used for the remaining better conditioned systems since the matrices $\hat{\mathbf{A}}_i^{2D}$ are symmetric and positive-definite. The 2D systems are ordered descending conditioning number, the direct method is used for the first D systems, where D is a delimiting parameter. The resulting algorithm of the time step is following:

1. explicit solution of the momentum equation;
2. explicit solution of the temperature transport equation;
3. calculate predictor fields, boundary conditions, and the divergence field (r.h.s. for the Poisson eq.);
4. solution of the Poisson equation:
   (a) the change-of-basis from physical to spectral space for the right-hand-side with a standard FFT;
   (b) solve the D lower-frequency 2D systems with the direct Schur-complement based method;
   (c) solve the remaining 2D systems with the iterative PCG method;
   (d) restore in the physical space the solution sub-vectors with inverse FFT;
5. calculation of gradient, calculation the resulting velocity field;
6. time integration step to the new temporal layer;
7. communication stage for halo update.

2.3. Parallelization

The first two levels of parallelization are based on a ”classical” MPI+OpenMP approach. MPI is used on the first level within the distributed memory model to couple computing nodes of a supercomputer. On the second level OpenMP is used to engage multiple CPU cores of a computing node. The underlying CPU parallelization is described in detail in [2] where the code was tested on up to 12800 CPU cores for meshes with up to 10^9 grid points and the estimations of the parallel potential showed limitations beyond 10^5 cores.

3. Extension for the hybrid computing model

3.1. Basic operations of the algorithm

To summarize the composition of the algorithm, the main numerical operations are categorized into the following six types:

- linear operator;
- nonlinear operator;
- vector algebraic operations of form $\mathbf{y} = \sum_{i=1}^{n} a_i * \mathbf{x}_i + c$, where $\mathbf{y}$ and $\mathbf{x}_i$ are mesh functions (fields), a_i are scalar coefficients, and the number of summands, n, may vary from 2 to 4;

- FFT transform;
- sparse matrix-vector product (SpMV);
- reduction operations – dot products and norms.

The first three types form the explicit part of the code delivering the discrete convection, diffusion and other operators in use. The last three form the PCG solver with an incomplete inverse preconditioner [8]. These six computing kernels (with few more minor operations) form the overall OpenCL-based CFD algorithm for hybrid systems.

The use of accelerators is considered within a higher-level domain decomposition. The computing domain of each accelerator is a subdomain inside a bigger domain. Fields of physical variables (mesh functions) are allocated for the extended subdomain that is a junction of a local subdomain with its halo nodes from neighbor subdomains.

The general strategy in the implementation is to avoid wherever possible loops and `if` branchings by unrolling the loops and replacing `if` statements with preprocessor definitions - luckily the kernel code compilation in runtime of the host code allows this. These constructions can have significant overhead on such accelerators and removing it may notably improve performance.

3.2. Linear operator

A linear operator is stored in a structured matrix with 7 diagonals for the 2-nd order discretization, with 19 diagonals for the 4-th order, etc. [3]. The coefficients of the linear operator stencil stay constant during the time integration.

The important feature is that for the periodic 3D case the mesh is uniform in the x direction and the periodic boundary conditions are prescribed in this direction. The resulting 3D matrix of the stencil coefficients consists of N_x equal "2D" blocks, where N_x is the number of nodes in x direction. The stencil matrix is stored as a set of diagonals for one "2D" block. Each diagonal in the set is stored as a mesh function defined in nodes of a 2D y-z slice of the subdomain. Such a data structure shown in Fig. 1 is preferred for the purpose of memory access coalescing over the block structure when values of all nonzero entries in each row are grouped together. The size of the OpenCL work-

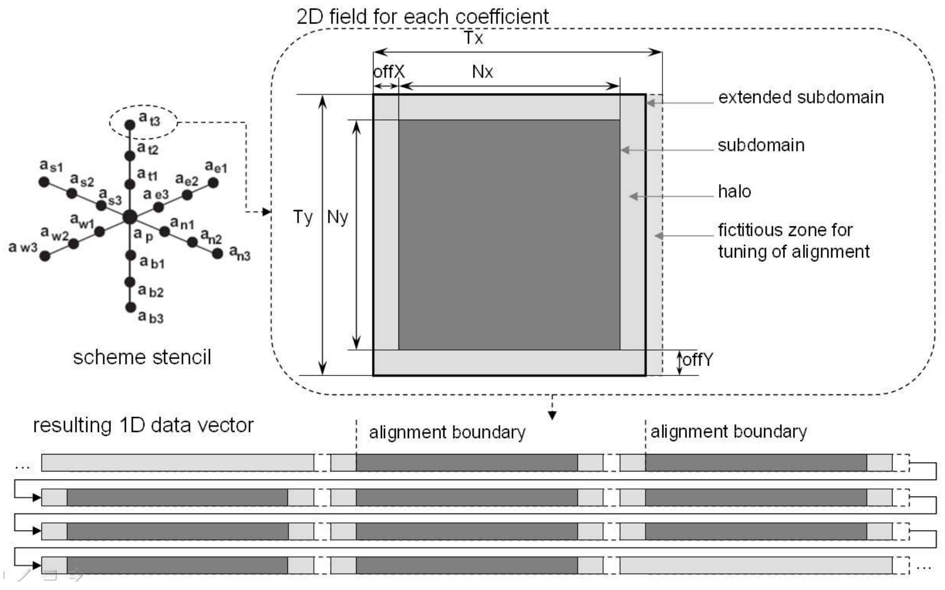

Fig. 1. Scheme of the space stencil data structure.

group is bigger or equal to the number of nodes in the local subdomain. Each thread processes independently its node. The threads are gathered in local work-groups in x direction - a block of threads processes the nodes with the same position in y and z directions. If the optimal local work-group size is bigger than the N_x, then one local work-group can process several neighboring lines. If N_x is bigger than the optimal block size, the operation is partitioned into several calls. This appears to be more efficient then to have a loop in the kernel.

3.3. Nonlinear operator

Representation of a nonlinear operator is more complicated, since stencil coefficients depend on the current velocity (advection) field. Firstly, coefficients of the stencil in each grid point are generated from the advection field. Then the stencil is applied to the input field. A set of ”pre-coefficients” is used for each stencil node to accelerate generation of the stencil coefficients. Pre-coefficients are calculated once at the beginning for the given mesh geometry. The number of pre-coefficients per stencil node may vary from zero to the width of the stencil leg. Since the mesh is staggered a heavy logic is needed to calculate the position of the corresponding velocity value depending if the resulting field is staggered in a particular direction and if the velocity component is staggered in the direction of the leg. It appeared to be more efficient to store the positions in a general CSR-like format (Compressed Sparse Row) for space matrices when the corresponding index of the velocity field value is stored for each pre-coefficient.

3.4. Poisson solver operations

The solver includes operations of the four basic types - FFT, SpMV, reduction, and vector algebraic operations. FFT is parallelized in a straightforward way: each thread independently process its subvector of N_x size (for one y-z position). The remaining three kinds of operations are forming the PCG solver with a block incomplete inverse preconditioner. The PCG method is applied to the group of independent 2D systems. Each of the systems may require different number of iterations. Each operation of the iterative method is applied to all the systems remaining to solve in order to group the communications. For instance, the SpMV is performed for multiple systems in order to commit the halo update for all the systems in a one message. In these conditions the SpMV, dot product, vector algebraic operations, have a common feature. The threads process all the remaining systems and if the convergence for a system is already achieved, then it gets a convergence flag and the treads responsible for that system are skipped. Reduction operations (dot product, norms) are implemented in a standard way except for dealing with a set of multiple vectors. Each local work-group performs reduction in the shared memory and then one thread performs reduction of results of local work-groups in the global memory.

It must be noted, that the direct Schur complement based solver is executed only on CPU. Porting it to GPU would hardly result in any considerable gain in performance since its time consumption is to a great extent due to the MPI data transfer. However, this approach requires two additional host-accelerator communications to transfer the r.h.s. vectors of the 2D systems to solve to the CPU and to copy solution vectors back to the accelerator.

4. Performance of separate operations

The tests have been carried out on example of a real application, a simulation of a differentially heated cavity, using the 4-th order scheme, double precision floating point format and a mesh with 1.28 million grid points made up of 64 slices in the periodic direction. The equipment involved in the tests includes:

- CPU Intel Xeon X5670 2.93GHz, 12 GFLOPS per core;
- GPU NVIDIA Tesla C2050, 515 GFLOPS, 144 Gb/s;
- GPU AMD Radeon 7970, 947 GFLOPS, 264Gb/s.

The GNU C++ compiler was used to build the CPU code with full optimization enabled. OpenCL 1.1 [4] was used for the kernel code, compilation with full optimization. It appeared that performance may notably change for different driver versions on AMD GPU. The version of OpenCL driver for AMD in these tests was 898.1. The performance results are summarized in Table 1. It shows speedups for different types of operations comparing one CPU core against one GPU accelerator. In addition it reports the achieved net performance in GFlops and the corresponding percent of the peak performance. It can be seen from the results that the net performance extracted from the GPU devices is rather far from the peak performance in terms of floating point operations. However, the main limiting factor for operations with such a low FLOP per byte ratio is the memory bandwidth. For the NVIDIA C2050 GPU the peak of 515 GFlops divided by the 144Gb/s peak memory bandwidth gives ratio around 3.6. The same relation holds for the AMD 7970 GPU device. Since the algorithm operates with 8-byte values this ratio must be multiplied by a factor of 8. Hence, there must be at least around 30 floating point operations per one double argument from the global memory of a GPU device to be near the peak performance. In most of the operations considered, the actual ratio is

Operation	1 CPU core			NVIDIA C2050			AMD 7970		
	time, sec.	GFlops	% of peak	speedup, times	GFlops	% of peak	speedup, times	GFlops	% of peak
diffusion	0.08	2.4	20	22.5	42	8.1	45.5	85	9
convection	0.25	2.0	17	17.6	36	7	43.0	88	9.3
type 3 op.	0.01	1.1	9	18.0	19.3	3.7	96	143	15.1
SpMV	0.042	0.76	6.3	19.2	14.6	2.8	45	34	3.6
FFT	0.035	1.5	12	7.7	11.5	2.2	12.7	19	2.0
Dot product	0.002	1.2	10	7.6	9	1.7	11.6	14	1.5

Table 1. Speedup of different operations comparing one GPU against one Intel Xeon CPU core (mesh size 1.28 millions of nodes, 4-th order scheme), achieved net performance in GFlops and the corresponding % of the peak performance.

at least one order of magnitude lower. This explains why the achieved computing performance can hardly overcome 10%. Finally, to compare GPU against the 6-core Xeon the speedups reported in Table 1 can be divided by a factor of around 4.5 (which is a typical OpenMP speedup observed on the 6-core CPU considered).

5. Conclusion

An OpenCL implementation of the parallel finite-volume CFD algorithm for modeling of incompressible flows on hybrid supercomputers has been presented. It is based on the six computing kernels that include linear and nonlinear operators, sparse matrix vector product, dot product, FFT. Results for a single-GPU performance is demonstrated for GPU devices of both AMD and NVIDIA. Multi-GPU performance and the two-level communication scheme with MPI communications and host-to-accelerator data transfer are to be presented. An experience with Intel Xeon Phi accelerators is also to be reported.

Acknowledgements

The K-100 hybrid supercomputer, Moscow, Russia, has been used for the tests. The work has been financially supported by RFBR, Russia, project No. 12-01-33022, and by the *Ministerio de Ciencia e Innovación*, Spain, project ENE2010-17801 and a Juan de la Cierva contract JCI-2009-04910. The authors thankfully acknowledge these institutions.

References

[1] F. X. Trias and A. Gorobets and C. D. Pérez-Segarra and A. Oliva", DNS and regularization modeling of a turbulent differentially heated cavity of aspect ratio 5, International Journal of Heat and Mass Transfer, Volume 57, Issue 1, 2013, pp. 171-182.

[2] A. Gorobets, F. X. Trias, R. Borrell, O. Lehmkuhl, A. Oliva, Hybrid MPI+OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction, Elsevier, Computers & Fluids 49 (2011), pp. 101-109.

[3] R. W. C. P. Verstappen, A. E. P. Veldman, Symmetry-Preserving Discretization of Turbulent Flow, Journal of Computational Physics 187 (2003) 343–368.

[4] Khronos OpenCL Working Group, The OpenCL Specification, Version: 1.1, 2010, http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[5] F. X. Trias and O. Lehmkuhl, A self-adaptive strategy for the time-integration of Navier-Stokes equations, Numerical Heat Transfer, part B, Volume 60, Number 2, 2011, pp. 116-134.

[6] A. J. Chorin, Numerical Solution of the Navier-Stokes Equations, Journal of Computational Physics 22 (1968) 745–762.

[7] M. Soria, C. D. Pérez-Segarra, A.Oliva, A Direct Schur-Fourier Decomposition for the Solution of the Three-Dimensional Poisson Equation of Incompressible Flow Problems Using Loosely Parallel Computers, Numerical Heat Transfer, Part B 43 (2003) 467–488.

[8] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS, 1996.