Computers & Fluids 81 (2013) 134–144

Contents lists available at SciVerse ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

A portable OpenCL-based unstructured edge-based finite element Navier–Stokes solver on graphics hardware

R. Rossi [a,b], F. Mossaiby [c,*], S.R. Idelsohn [a,d]

[a] Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Barcelona, Spain
[b] UPC, BarcelonaTech, Campus Norte UPC, 08034 Barcelona, Spain
[c] Department of Civil Engineering, Faculty of Engineering, University of Isfahan, 81744-73441 Isfahan, Iran
[d] Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain

ARTICLE INFO

Article history:
Received 18 December 2012
Received in revised form 27 March 2013
Accepted 16 April 2013
Available online 2 May 2013

Keywords:
Unstructured grids
Navier–Stokes
Edge-based
GPU
OpenCL
OpenMP

ABSTRACT

The rise of GPUs in modern high-performance systems increases the interest in porting portion of codes to such hardware. The current paper aims to explore the performance of a portable state-of-the-art FE solver on GPU accelerators. Performance evaluation is done by comparing with an existing highly-optimized OpenMP version of the solver. Code portability is ensured by writing the program using the OpenCL 1.1 specifications, while performance portability is sought through an optimization step performed at the beginning of the calculations to find out the optimal parameter set for the solver. The results show that the new implementation can be several times faster than the OpenMP version.

1. Introduction

The solution of the incompressible Navier–Stokes problem, which describes the motion of Newtonian fluids, represents an open challenge for the numerical community. Given the importance of the problem, a large effort was spent over the years in the development of dedicated numerical techniques to improve the speed and the accuracy of the solution process. The use of lattice-based approaches such as the Finite Differences or Lattice-Boltzmann schemes lead to the development of highly-efficient schemes, which can deal effectively with very large computational meshes. The relative simplicity of the computational kernels together with the highly regular structure of the computations were found to fit perfectly to the architectural needs of modern accelerators. Various authors (see e.g. [1]) were able to achieve performance boosts by developing optimized kernels with respect to their single-CPU counterparts. Although local adaptivity was shown to be very effective, any modification with respect to the simple approach of using regular meshes typically leads to a decrease in the computational efficiency and a drop in the performance boost for hardware accelerated algorithms. Furthermore, all of such techniques find major limitations in dealing with complex geometries and curved boundaries.

Unstructured discretization of the space, typically based on tetrahedral meshes, represent a possible solution for such problems. The strength of such approaches is the possibility of using body-fitted meshes and spatially adapted discretizations of the space. The price paid to achieve this advantage is an increasingly irregular computational pattern which reflects in a variable number of edges surrounding the nodes of the computational mesh. Such a situation does not represent a major problem for cache-based processors used in conjunction with OpenMP or MPI programming paradigms, since computations can be organized so to take full advantage of the cache, while the number of parallel OpenMP threads is typically kept low. However, prior experiences of the authors [2], confirmed by the reports of others [3], seem to suggest that only low speedups can be achieved with respect to the CPU-only solutions.

The current paper aims to examine the OpenCL porting of an OpenMP edge-based solver so as to identify and discuss the performance bottlenecks. The paper starts with a brief description of the solution algorithm used, followed by the presentation of the data structure employed and of the structure of the parallel computations. An effort is performed to make the discussion as implementation-independent as possible. The impact of using black-box solvers (such as the ViennaCL library [4]) in comparison to in-house optimized implementations for the solution of the implicit pressure step is evaluated. Benchmarking data over platforms with

* Corresponding author. Tel.: +98 (311) 793 4015; fax: +98 (311) 793 2089.
E-mail addresses: rrossi@cimne.upc.edu (R. Rossi), mossaiby@eng.ui.ac.ir (F. Mossaiby), sergio@cimne.upc.edu (S.R. Idelsohn).

different software and hardware configurations are presented so as to allow a broader comparison. Finally the impact of a run-time optimization step will be evaluated. The proposed implementation is open-source and freely available as a part of the Kratos framework [5]. More information can be found in the Wiki page for Kratos [6].

2. Finite element formulation

Between the many existing possibilities for the solution of Navier–Stokes problem, we use a fractional-step approach. In this scheme we solve explicitly the momentum equation (using a 4-step Runge–Kutta scheme) and implicitly the pressure correction step. A detailed discussion of the algorithm is presented in [7] for the incompressible case and in [8] for the solution of Low-Mach compressible problems.

In order to understand the basic concepts of the method, we may start by considering the strong form of the Navier–Stokes equations, written for the constant–density case

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nu \Delta \mathbf{u} + \nabla p = \mathbf{b} \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

where $\mathbf{u}$ is the velocity, ν is the kinematic viscosity, p is the pressure and $\mathbf{b}$ is the applied body force. By applying the standard Galerkin approach, using the test functions $\mathbf{v}$ and q, we obtain

$$(\mathbf{v}, \mathbf{b}) - \left(\mathbf{v}, \frac{\partial \mathbf{u}}{\partial t}\right) - (\mathbf{v}, \mathbf{u} \cdot \nabla \mathbf{u}) + (\mathbf{v}, \nu \Delta \mathbf{u}) - (\mathbf{v}, \nabla p) = \mathbf{0} \tag{3}$$

$$(q, \nabla \cdot \mathbf{u}) = 0 \tag{4}$$

Eqs. (3) and (4) define the discrete equivalent of the original continuous problem. Since our goal is to use low-order simplicial meshes and equal order velocity–pressure pairs, a stabilized formulation is needed to allow the solution of the resulting mixed $(\mathbf{u}, \mathbf{p})$ problem. Different possibilities exist for this purpose [9]. We favor here the use of the so-called split-OSS approach [10], which is known to work properly in a wide range of applications. FIC stabilization [11,12] could be used as an alternative since it leads to very similar discrete forms. Since a discussion of the properties of the chosen stabilization method falls outside the scope of this work, we refer the reader to the literature for a detailed description of the properties of such technique. In the following we express the stabilization terms as the non linear operators $\mathbf{S}(\mathbf{u}) := (\mathbf{u} \cdot \nabla \mathbf{v},\ \tau \Pi_\perp(\mathbf{u} \cdot \nabla \mathbf{u}))$ and $\mathbf{S}_p(\mathbf{p}) := (q,\ \tau \Pi_\perp(\nabla p))$ where $\Pi_\perp$ represents the orthogonal projection operator and τ is a suitably defined scalar. Reader should check [10] or [14] for a detailed description of such terms. The resulting final form of the discrete equations is

$$\mathbf{M}\frac{\partial \mathbf{u}}{\partial t} + [\mathbf{C}\mathbf{u}) - \nu \mathbf{L} + \mathbf{S}(\mathbf{u})]\mathbf{u} + \mathbf{\nabla p} = \mathbf{F} \tag{5}$$

$$\mathbf{D}\mathbf{u} + \mathbf{S}_p \mathbf{p} = \mathbf{0} \tag{6}$$

where $\mathbf{M}$ is the lumped mass matrix and

$$\mathbf{\nabla}_{IJ} := \int_\Omega N_I \nabla N_J \mathrm{d}\Omega \tag{7}$$

$$\mathbf{G}_{IJ} := \int_\Omega \nabla N_I N_J \mathrm{d}\Omega \tag{8}$$

$$\mathbf{D}_{IJ} := \int_\Omega N_I \nabla N_J^T \mathrm{d}\Omega = \mathbf{\nabla}_{IJ}^T \tag{9}$$

$$\mathbf{L}_{IJ} := \int_\Omega \nabla N_I \cdot \nabla N_J \mathrm{d}\Omega \tag{10}$$

are linear operators obtained by integrating over the domain the finite element shape functions indicated with N. The convection operator $\mathbf{C}(\mathbf{u})$ is a non-linear term which depends on the velocity. It can be defined as

$$\mathbf{C}_{IJ} := \int_\Omega N_I \mathbf{u} \cdot \nabla N_J^T \mathrm{d}\Omega \tag{11}$$

Eq. (5) has three components in 3D. This is reflected in $\mathbf{G}$ and $\mathbf{D}$ being respectively 3×1 and 1×3 matrices for each couple of indices IJ of the FE mesh. The description is completed by the exact definition of the stabilization terms. A detailed description of such terms, particularized to the Split-OSS case, and using the same notation used here can be found in [13]. Choosing a fractional-step approach implies approximating the original system of equations as

$$\mathbf{M}\frac{\partial \hat{\mathbf{u}}}{\partial t} + [\mathbf{C}(\hat{\mathbf{u}}) - \nu \mathbf{L} + \mathbf{S}(\hat{\mathbf{u}})]\hat{\mathbf{u}} + \mathbf{\nabla p}_n = \mathbf{F} \tag{12}$$

$$\mathbf{M}\frac{\partial (\mathbf{u} - \hat{\mathbf{u}})}{\partial t} + \frac{\Delta t}{2}\mathbf{\nabla}(\mathbf{p}_{n+1} - \mathbf{p}_n) = \mathbf{0} \tag{13}$$

$$\mathbf{D}\mathbf{u} + \mathbf{S}_p \mathbf{p}_{n+1} = \mathbf{0} \tag{14}$$

where $\hat{\mathbf{u}}$ is the so-called fractional-step velocity. The fractional-step velocity is modified at each step so that its value in the past coincides with the velocity $\mathbf{u}_n$, that is, $\hat{\mathbf{u}}_n = \mathbf{u}_n$.

Since Eq. (12) is still continuous in time, we have different options in choosing a time integration scheme to properly advance in time the momentum equation (the first of the three equations above). As we stated previously, our choice in the current work is the use of a 4th order Runge–Kutta scheme. As observed in [7,14], by making the fundamental approximation of considering the end-of-step velocity to *depend linearly* on the pressure despite the non-linearity of the convection terms, we may conclude that the end-of-step velocity can be expressed as

$$\mathbf{u} = \hat{\mathbf{u}} + \frac{\Delta t}{2}\mathbf{M}^{-1}\mathbf{\nabla}(\mathbf{p}_{n+1} - \mathbf{p}_n) \tag{15}$$

where $\mathbf{p}_{n+1}$ and $\mathbf{p}_n$ indicate respectively the *new* value of the pressure (at time t_{n+1}), and its latest known value (at time t_n). Substituting symbolically this expression into the mass conservation equation (Eq. (14)) and replacing the discrete Laplacian $\mathbf{DM}^{-1}\mathbf{G}$ with the continuous one $\mathbf{L}$ we obtain the equation

$$\frac{\Delta t}{2}\mathbf{L}(\mathbf{p}_{n+1} - \mathbf{p}_n) + \mathbf{S}_p \mathbf{p}_{n+1} = \mathbf{D}\hat{\mathbf{u}} \tag{16}$$

The above equation needs to be solved for the pressure. This ultimately leads to a solution strategy articulated in the following steps:

1. Solve Eq. (12) for the fractional-step velocity, $\hat{\mathbf{u}}$.
2. Solve Eq. (16) for the pressure, p_{n+1}.
3. Solve Eq. (15) for the end-of-step velocity, $\mathbf{u}$.

From a practical point of view, the equation to be solved in Step 1 is mathematically equivalent to the solution of a convection–diffusion equation for each of the velocity components. The pressure gradient and the external pressure term act here as source terms. The idea we leverage in this work is to take advantage of the efficient edge-based data structure described in [13,14] to allow the efficient computation of the fractional-step velocity.

The essential idea is that, basing on a slight approximation of the viscous and convective term, it is possible to *approximate* all of the operators described in terms of constant operators. Such operators can be computed at the beginning of the calculations and stored for each edge IJ of the finite element mesh, making the approach very appealing for use on GPU since the operators can be computed on the CPU and transferred to the GPU memory for later usage.