Computers and Fluids 121 (2015) 37–43

Contents lists available at ScienceDirect

Computers and Fluids

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

Performance of a projection method for incompressible flows on heterogeneous hardware

Tobias Kempe[a,*], Alvaro Aguilera[b], Wolfgang Nagel[b], Jochen Fröhlich[a]

[a] Institute of Fluid Mechanics, Technische Universität Dresden, George-Bähr Str. 3c, 01062 Dresden, Germany
[b] Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden, Zellescher Weg 12, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 31 March 2015
Revised 29 July 2015
Accepted 31 July 2015
Available online 18 August 2015

Keywords:
Incompressible flow
Projection method
Poisson equation
GPU acceleration
PETSc
Hypre

ABSTRACT

Practical experiments on the flow in a lid-driven cavity are carried out to compare the performance of a second-order finite volume Navier–Stokes solver for incompressible fluids employing a projection method, when using various linear solver libraries on central processing units (CPUs) and graphical processing units (GPUs). The goal of the paper is to identify the potential of GPU acceleration using the built-in GPU solvers of PETSc and to provide information if the usage of GPUs is beneficial for this type of fluid solver in terms of performance and implementation effort. Additionally, energy consumption having emerged as another important goal of optimization in high-performance computing is addressed as well. In this study, the solvers available in the PETSc library, which are running on CPU as well as with GPU support, are compared with the solvers provided by the hypre library in a systematic way. The power consumption of the CPU and the GPU during the solution is measured to assess the energy efficiency in terms of the performance-per-Watt ratio. It is found that for the considered numerical scheme the usage of iterative solvers on current GPU systems is not necessarily beneficial, neither in terms of performance nor in terms of energy consumption, since both libraries, PETSc and hypre, provide highly-optimized solvers for massively parallel CPU systems.

1. Introduction

Graphical processing units (GPUs) originally developed for computer games nowadays are widely used to perform scientific and engineering simulations. In particular, they are strictly used as computing core in some of the most performant supercomputing facilities [4]. Along with performance and implementation issues, the total energy consumption required for numerical solution of a physical problem is a topic that has gained notoriety over the last decade since the size of new high-performance computing facilities is being increasingly constrained by the energy requirements of both the computing and the cooling infrastructure [20]. The Tianhe-2 system at the National Super Computer Center in Guangzhou, China, the fastest system created until now, is an example. Its power consumption is almost 18MW when working at full capacity [4]. Heterogeneous hardware platforms, combining central processing units (CPUs) and GPUs, allow significant savings of computing time and power consumption for some applications, e.g. [13,19,27] to name but a few.

Collange *et al.* [11], for example, investigated how computations and memory accesses impact the power consumption of Nvidia GPUs. In the study of Huang *et al.* [19] the energy efficiency of GPUs for scientific computing was investigated for the case of the electrostatic potential generated by charges inside a molecule and significant advantages of the GPU over the CPU were found with respect to energy consumption and computing time. Ma *et al.* [27] present a model to dynamically estimate the power consumption of the GPU and to increase the performance per Watt ratio by an automated selection approach of choosing near-optimal energy efficient GPU programming configurations. Enos *et al.* [13] investigated the performance per Watt of four scientific applications that were ported on GPUs. The results indicate that although GPUs significantly increase power consumption, the provided acceleration results in a reduction of the overall power consumption.

In computational fluid dynamics (CFD) the direct numerical simulation (DNS) of turbulent single phase and multiphase flows in general is very expensive, even at low to moderate Reynolds numbers. Various authors address energy consumption [31] and report on significant acceleration of the their CFD codes using GPUs [7,12,16,28,30,35]. However, the authors of [31] state that “existing codes needs to be ported and optimized, a procedure which is not yet standardized and may require non trivial efforts, even to high-performance computing specialists”.

* Corresponding author. Tel.: +4935146336651.
E-mail addresses: tobias.kempe@tu-dresden.de (T. Kempe), alvaro.aguilera@tu-dresden.de (A. Aguilera), wolfgang.nagel@tu-dresden.de (W. Nagel), jochen.froehlich@tu-dresden.de (J. Fröhlich).

http://dx.doi.org/10.1016/j.compfluid.2015.07.028
0045-7930/© 2015 Elsevier Ltd. All rights reserved.

The purpose of the present study is to investigate the potential for efficiency increase of a CFD code with the help of GPUs, when employing a discretization scheme widely used for incompressible flows. It is based on the solution of an elliptic equation for the pressure correction. The chosen configuration is a three-dimensional driven cavity which is representative for a wide range of applications. The cost functions investigated are computational speed, energy consumption and implementation effort.

2. Numerical method and problem definition

2.1. Description of the flow solver

The solver employed in the present study is the in-house code PRIME (Phase-Resolving sIMulation Environment), an immersed boundary method (IBM) finite-volume solver for multiphase flows [21–24]. It is routinely used for high-performance computing of unsteady multiphase flow problems on billions of grid points and thousands of processors. A physical problem addressed in such simulations, for example, is the transport of sediment in riverbeds [25,32–34]. An Eulerian–Lagrangian IBM is constituted of three components, a fluid-solver on a generally regular Cartesian – Eulerian background grid covering the entire computational domain, a description of the Lagrangian motion of the disperse phase, and a coupling mechanism by an IBM.

In the present study, only the Eulerian solver for incompressible flow on the regular grid is employed. The governing equations are the unsteady three-dimensional Navier–Stokes equations for Newtonian fluids of constant density

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

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) = -\frac{1}{\rho_f} \nabla p + \nu_f \nabla^2 \mathbf{u}. \tag{2}$$

The nomenclature is as usual, with $\mathbf{u} = (u, v, w)^T$ designating the velocity vector in Cartesian components along the Cartesian coordinates x, y, z, while t represents time, p pressure, ρ_f the fluid density, and ν_f the kinematic viscosity of the fluid. An efficient approach for the numerical solution of problems involving incompressible fluids is a projection method, as proposed by Chorin [10]. In algorithms of this class, the solution of the nonlinear system of Eqs. (1) and (2) is split into the following two consecutive steps. First, a non divergence-free intermediate velocity field is computed neglecting the pressure term or using the pressure field of the previous time level. In the second step, the pressure at the new time level is determined and the intermediate velocity field is projected onto a divergence free field. The projection step requires the solution of an elliptic equation - the pressure correction Poisson equation. It is derived by applying the divergence operator to the momentum equation (2) and using the incompressibility condition (1).

To simplify the study, the basic fluid solver of [23] is used here in a slightly modified version. The implicit treatment of the viscous terms in (2) yields a linear system of Helmholtz type. This is omitted here and the time advancement is accomplished by an explicit three-step third-order low-storage Runge–Kutta scheme for the convective terms and the viscous terms as well. The temporal discretization scheme can be summarized as follows

$$\begin{aligned}\frac{\tilde{\mathbf{u}} - \mathbf{u}^{k-1}}{\Delta t} &= -2\,\alpha^k \frac{1}{\rho_f} \nabla p^{k-1} - \gamma^k \left(\nabla \cdot (\mathbf{u}\mathbf{u})^{k-1} + \nu_f \nabla^2 \mathbf{u}^{k-1}\right) \\ &\quad - \zeta^k \left(\nabla \cdot (\mathbf{u}\mathbf{u})^{k-2} + \nu_f \nabla^2 \mathbf{u}^{k-2}\right)\end{aligned} \tag{3a}$$

$$\nabla^2 \sigma^k = \nabla \cdot \tilde{\mathbf{u}} \tag{3b}$$

$$\mathbf{u}^k = \tilde{\mathbf{u}} - \nabla \sigma^k \tag{3c}$$

$$p^k = p^{k-1} + \frac{\sigma^k}{2\,\alpha_k\,\Delta t}, \tag{3d}$$

where the superscript k denotes the Runge–Kutta sub-step, with the corresponding coefficients α^k, γ^k and ζ^k given in [23]. First, an intermediate velocity field $\tilde{\mathbf{u}}$ according to (3a) is computed using the pressure of the previous sub-step, p^{k-1}. The solution of the Poisson equation (3b) to obtain the pressure correction σ and the projection step (3c) yields the divergence-free velocity field $\mathbf{u}^k$. The pressure field subsequently is updated by (3d).

The spatial discretization of (1) and (2) is performed by a central second-order finite-volume scheme on a staggered Cartesian grid [18] for all terms. The parallelization of the code is accomplished by a domain decomposition approach with explicit MPI communications employing ghost cells in the PETSc framework.

2.2. Solution of linear systems of equations

At various stages of the numerical scheme, the solution of linear systems of equations is required. Usually, this is the Poisson equation for the pressure (3b). In case the viscous terms are treated implicitly, (3a) is modified and requires the solution of a linear equation of Helmholtz type [23]. This is not performed here, as this equation is similar to the Poisson equation for the pressure correction and would make the assessment more complicated here by introducing further algorithmic degrees of freedom without changing the conclusions. Indeed the solution of the Poisson equation (3b) for single as well as for multiphase applications, usually is the most time consuming part of the overall solution procedure. It requires up to 80 to 95 % of the total computing time, and hence efficient solvers for this part are essential for a good performance of the entire code.

Currently, the linear solvers provided by the hypre library [14] are used with great success on massively parallel systems and are routinely employed by the authors. To explore possible improvements of the code in terms of performance and energy consumption, practical experiments with an alternative solver library are conducted in this paper. While hypre does not provide GPU support, the PETSc library [9], in addition to a variety of linear solvers running on CPU, in recent time, also contains options for GPUs [29].

The key idea of the paper is to compare the performance and energy efficiency of the solvers provided by hypre with the solvers of PETSc, without or with GPU support. The strategy is to compare identical solvers on single and multiple-cores of a CPU and on a GPU as well as the best available solver on CPU versus the best available solver on GPU. This provides first of all information for practitioners. Second, it allows to obtain an assessment of the considered method in a more general sense and to inform on the performance for similar solvers, e.g. treating the Helmholz equation. The following solvers and corresponding labels are used in the paper:

hypre

- H1: Conjugate gradient (CG) without preconditioner
- H2: CG with parallel semicoarsening multigrid (PFMG) preconditioner
- H3: Biconjugate gradient stabilized method (BiCGSTAB) with PFMG preconditioner

PETSc

- P1: CG without preconditioner
- P2: CG with algebraic multigrid (PCML) preconditioner

PETSc with GPU support

- PGPU1: CG without preconditioner
- PGPU2: CG with NVIDIA smoothed aggregation (CUSP) preconditioner