J Supercomput (2017) 73:4931–4956
DOI 10.1007/s11227-017-2066-y

# A parallel multigrid solver for incompressible flows on computing architectures with accelerators

**Vassilios G. Mandikas**[1] · **Emmanuel N. Mathioudakis**[1]

Published online: 10 May 2017

**Abstract** An efficient parallel multigrid pressure correction algorithm is proposed for the solution of the incompressible Navier–Stokes equations on computing architectures with acceleration devices. The pressure correction procedure is based on the numerical solution of a Poisson-type problem, which is discretized using a fourth-order finite difference compact scheme. Since this is the most time-consuming part of the solver, we propose a parallel pressure correction algorithm using an iterative method based on a block cyclic reduction solution method combined with a multigrid technique. The grid points are numbered with respect to the red–black ordering scheme for the parallel Gauss–Seidel smoother. These parallelization techniques allow the execution of the entire simulation computations on the acceleration device, minimizing memory communication costs. The realization is developed using the OpenACC API, and the numerical method is demonstrated for the solution of two classical incompressible flow test problems. The first is the two-dimensional lid-driven cavity problem over equal mesh sizes while the other is the Stokes boundary layer, which is a decent benchmark problem for unequal mesh spacing. The effect of several multigrid components on modern and legacy acceleration architectures is examined. Eventually the performance investigation demonstrates that the proposed parallel multigrid solver achieves an acceleration of more than $10\times$ over the sequential solver and more than $4\times$ over multi-core CPU only realizations for all tested accelerators.

✉ Emmanuel N. Mathioudakis
manolis@amcl.tuc.gr

Vassilios G. Mandikas
bmandikas@science.tuc.gr

1 Applied Mathematics and Computers Laboratory, School of Mineral Resources Engineering, Technical University of Crete, University Campus, 73132 Chania, Greece

**Keywords** Incompressible Navier–Stokes equations · Compact finite difference schemes · Geometric multigrid techniques (GMG) · Zebra line Gauss–Seidel relaxation · Block cyclic reduction (BCR) · Computing accelerators

## 1 Introduction

The development of high-order accurate methods for the numerical solution of incompressible Navier–Stokes equations is long-term, noteworthy issue in engineering. Applications modeling low-speed aerodynamics, biomechanics, direct and large eddy simulations for turbulent flows, aerodynamics design with detailed information of the near wall flowfield, are among realistic simulation problems that demand the development of a high-order numerical scheme.

In [17], in the solution of the unsteady incompressible Navier–Stokes equation, an explicit Runge–Kutta, high-order finite difference solver is presented employing pressure–velocity coupling, using a solution technique [19,22] which is based on fourth-order compact schemes [13,22]. The results indicate that high-resolution simulations require fine discretizations increasing the computational cost of the numerical solution of the pressure equation. A performance improvement in the aforementioned method can be achieved applying of Geometric Multigrid techniques (GMG) [2,3,34,36]. The study in [23,26] concludes that in case studies where the problem dimensions size up to $256 \times 256$ the incorporation of the multigrid scheme improve the overall efficiency of the implementation in terms of time. However, problems where the discretization is finer than those above computational cost become intolerable despite the high convergence rates of GMG. This outcome motivates us to design a parallel algorithm of the numerical solver for modern computing architectures with accelerators.

Over the last decade, the rapid evolution of Graphics Processing Units (GPUs) into powerful, cost-efficient, programmable computing accelerators for general purpose computations, allowed researchers to deploy them in the numerical solution of Partial Differential Equations (PDE) in several scientific fields [8,12]. Recent efforts to accelerate Computational Fluid Dynamics (CFD) simulations using GPUs can be found in [10,27] and especially in implementations employed GMG in [35].

In the last few years, several multigrid-based Navier–Stokes solvers have been successfully implemented on architectures with accelerators. Shinn and Vanka [31] implemented a multigrid Full Approximation Scheme (FAS) method using the semi-implicit method for the SIMPLE algorithm to simulate incompressible flows. Thibault and Senocak [33] developed a second order of accuracy CUDA-based GPU solver for the incompressible Navier–Stokes equations. In a domain of 8,388,608 computational nodes, the double precision simulation algorithm over a single GPU for the lid-driven cavity problem achieved roughly a $13\times$ acceleration compared to the single-core CPU only implementation. Later, Cohen and Molemaker [7] presented a second-order finite volume algorithm implementation solving the incompressible flow equations with the Boussinesq approximation. Using double precision arithmetic, the GPU realizations were approximately eight times faster over the eight-core CPU only simulations. However, it is worth mentioning that the majority of the parallel implementations on computing architectures with accelerators are based on low order of discretization schemes. Feng et al. [11] have studied the performance of GMG for a low-order difference scheme on CPU–GPU computing architectures. The results indicate, that, when the problem size is large enough, it is more effective to perform computations on the accelerator. Several fluid dynamics simulations are implemented using the OpenFoam software [28]. It is an open-source CFD toolbox based on the parallel finite volume method, where the incompressibility condition is enforced with the pressure-implicit split-operator (PISO) or semi-implicit method for pressure-linked equations (SIMPLE) algorithms.

In many simulation applications, such as the temperature distribution in a thin rod and the Stokes oscillating plate, the distribution of the physical quantity can vary in each direction. In this type of problems, multigrid can use a cost-effective strategy. This is called semi-coarsening strategy, i.e., a mesh coarsening procedure performed only along the dominant spatial direction [21,37]. There is insufficient literature on concerning performance investigations of multigrid methods using the semi-coarsening against the full-coarsening strategy [5] on computing architectures with accelerators. In [24], a massively parallel version of the semi-coarsening multigrid 3D solver is developed and multi-GPU implementations are found to be faster than the multi-core CPU only implementations.
The contributions of this paper are as follows:

– A parallel algorithm of a high-order Navier–stokes numerical solver based on finite difference compact schemes [17] and GMG techniques [15,26] is designed for computing architectures with accelerators.
– The implementation of the algorithm using OpenACC directives [32].
– Attention is focused on featuring the differences between the parallel implementation over grids with equal and unequal mesh spacing.
– A performance investigation of cell-centered GMG techniques on parallel architectures is obtained.
– Multigrid's relaxation phase leads to series of block tridiagonal linear systems. These linear systems are effectively solved with the block cyclic reduction method [16] which is appropriate for vector computers [18,20].
– Taking the structure of the matrices into account, the matrix-free storage method is chosen, allowing the entire computation to be executed on the accelerator device.
– The coarse-grid operators are determined by the discretization of the pressure correction equation on the coarse grids; therefore, the parallel attributes of the algorithm are retained.
– Performance investigations for several acceleration device types are presented.

This work is organized as follows: Sect. 2 presents the governing equations and the fourth-order discretization procedures in space and time for the Navier–Stokes and the pressure correction equations. In Sect. 3, the multigrid technique with its components is presented accelerating the pressure correction procedure. In Sect. 4, the parallel Navier–Stokes algorithm is presented in detail. The parallel performance results of the Navier–Stokes solver and the influence of the parallel pressure correction algorithm at each time step for high-resolution simulations are the scope of Sect. 5.

## 2 The numerical scheme

In order to improve the multigrid incompressible Navier–Stokes solver's performance, modern computing architectures with accelerator devices can be used in realization. The proposed high-order parallel solver is based on the work in [17], as it is extended using multigrid acceleration in [26].

### 2.1 Governing equations

The conservative form of the incompressible Navier–Stokes equations in Cartesian coordinates is:

$$\partial_i \, u_i = 0 \tag{1}$$

$$\partial_t \, u_i + \partial_j \, \mathfrak{F}_{ij}^c = -\nabla p + \frac{1}{Re} \partial_j \, \mathfrak{F}_{ij}^{\upsilon} \, , \tag{2}$$

where $u_i = \mathbf{u} = [u, v, w]^{\mathrm{T}}$ is the velocity vector, $\mathfrak{F}_{ij}^c = [\mathbf{F}_x^c, \mathbf{F}_y^c, \mathbf{F}_z^c]^{\mathrm{T}}$ and $\mathfrak{F}_{ij}^{\upsilon} = [\mathbf{F}_x^{\upsilon}, \mathbf{F}_y^{\upsilon}, \mathbf{F}_z^{\upsilon}]^{\mathrm{T}}$ are vector-valued functions containing the convective and viscous flux vectors, respectively, $p$ is the pressure, and $Re = UL/\nu$ is the nondimensional Reynolds number based on the characteristic velocity $U$, the characteristic length-scale $L$ and the kinematic viscosity $\nu$. In this work, the numerical method is demonstrated for two-dimensional flows, where the convective flux vectors are $\mathbf{F}_x^c = [u^2, uv]^{\mathrm{T}}$ and $\mathbf{F}_y^c = [uv, v^2]^{\mathrm{T}}$ and the viscous flux vectors are $\mathbf{F}_x^{\upsilon} = [\partial_x u, \partial_x v]^{\mathrm{T}}$ and $\mathbf{F}_y^{\upsilon} = [\partial_y u, \partial_y v]^{\mathrm{T}}$.

### 2.2 Numerical approach

The physical domain is subdivided into rectangular cells $C_{i,j}$ for $i = 1, \ldots, N_x$ and $j = 1, \ldots, N_y$ of width $\Delta x$ and height $\Delta y$, discretizing each spatial dimension into $N_x$ and $N_y$ subintervals. Fourth-order compact finite difference discretization method is applied for solving incompressible Navier–Stokes equations, formulated on a staggered grid arrangement, as illustrated in Fig. 1. The dependent variables are discretized on each computational cell $C_{i,j}$: The pressure $p$ denoted by $p_{i,j}$ at the cell center; $u$ velocity component denoted by $u_{i+1/2,j}$ at the vertical edges midpoints; and $v$ velocity component denoted by $v_{i,j+1/2}$ at the horizontal edges midpoints.

Incompressibility is enforced using a globally defined pressure correction equation, which is discretized to a fourth-order nine-point compact scheme. A cell-centered geometric multigrid (CCMG) technique is applied for accelerating the numerical solution of the pressure correction equation [26]. The temporal discretization is carried out by an explicit fourth-order Runge–Kutta method [4]. The main structure of the solver is outlined in Algorithm 1. A more detailed description of the method can be found in [17].

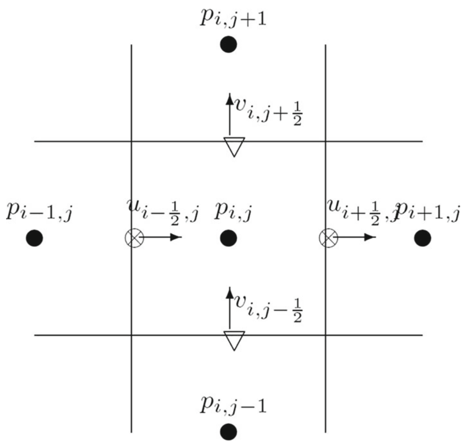

**Fig. 1** Schematic details for the computational cell $C_{ij}$

**Algorithm 1** Multigrid Navier-Stokes solver

```
Step 1. Guess an initial pressure field
        for all time steps do
            for all RK4 stages do
                while (not converge) do
Step 2.                 - Solve pressure correction equation
Step 3.                 - Correct pressure and velocity fields
                end
            endfor
Step 4.     Update velocities and pressure
Step 5.     Set current pressure field as initial pressure
        endfor
```

Consider now the momentum Eq. (2) written in the following compact form, as

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{R}(\mathbf{u}, p; t), \tag{3}$$

where $\mathbf{R}(\mathbf{u}, p; t) = -\nabla p + A(\mathbf{u}; t)$.

Assuming $\gamma = \Delta x/\Delta y$ and $a = 1+\gamma^2,\ b = 5-\gamma^2,\ c = 5\gamma^2 - 1$, each time step of the discretization scheme for both momentum and the pressure correction equations can be written in the form

**Stage 1** :

$$u^{(n,1)}_{\tilde{i}+1/2,j} = u^{(n)}_{\tilde{i}+1/2,j} \tag{4}$$

$$v^{(n,1)}_{i,\tilde{j}+1/2} = v^{(n)}_{i,\tilde{j}+1/2} \tag{5}$$

$$p^{(n,1)}_{i,j} = p^{(n)}_{i,j} \tag{6}$$

**Stage $\boldsymbol{\ell}$ :**

$$u^{(n,\ell)}_{\tilde{i}+1/2,j} = u^{(n)}_{\tilde{i}+1/2,j} + a_{\ell,\ell-1}\Delta t\ F^{(n,\ell-1)}_{\tilde{i}+1/2,j} \tag{7}$$

$$v^{(n,\ell)}_{i,\tilde{j}+1/2} = v^{(n)}_{i,\tilde{j}+1/2} + a_{\ell,\ell-1}\Delta t\ G^{(n,\ell-1)}_{i,\tilde{j}+1/2} \tag{8}$$

**Pressure Correction Equation** :

$$\begin{aligned}&a(\Delta p_{i+1,j+1} + \Delta p_{i+1,j-1} + \Delta p_{i-1,j+1} + \Delta p_{i-1,j-1}) + 2b(\Delta p_{i,j+1} + \Delta p_{i,j-1})\\ &+2c(\Delta p_{i+1,j} + \Delta p_{i-1,j}) - 20a\Delta p_{i,j} \ = \ \Delta x^2[8(\nabla\cdot\mathbf{u}^{(n,\ell)})_{i,j}\\ &+(\nabla\cdot\mathbf{u}^{(n,\ell)})_{i-1,j} + (\nabla\cdot\mathbf{u}^{(n,\ell)})_{i+1,j} + (\nabla\cdot\mathbf{u}^{(n,\ell)})_{i,j-1} + (\nabla\cdot\mathbf{u}^{(n,\ell})_{)i,j+1}]\end{aligned} \tag{9}$$

**Correct** :

$$p^{(n,\ell)}_{i,j} = p^{(n,\ell-1)}_{i,j} + \Delta p_{i,j} \tag{10}$$

$$u^{(n,\ell)}_{\tilde{i}+1/2,j} = u^{(n,\ell)}_{\tilde{i}+1/2,j} - a_{\ell,\ell-1}\Delta t\ \left(\frac{\partial\Delta p}{\partial x}\right)_{\tilde{i}+1/2,j} \tag{11}$$

$$v^{(n,\ell)}_{i,\tilde{j}+1/2} = v^{(n,l)}_{i,\tilde{j}+1/2} - a_{\ell,\ell-1}\Delta t\ \left(\frac{\partial\Delta p}{\partial y}\right)_{i,\tilde{j}+1/2}, \qquad \text{for } \boldsymbol{\ell = 2, 3, 4} \tag{12}$$

**Update** :

$$u^{(n+1)}_{\tilde{i}+1/2,j} = u^{(n)}_{\tilde{i}+1/2,j} + \frac{\Delta t}{6}\left(F^{(n,1)}_{\tilde{i}+1/2,j} + 2F^{(n,2)}_{\tilde{i}+1/2,j} + 2F^{(n,3)}_{\tilde{i}+1/2,j} + F^{(n,4)}_{\tilde{i}+1/2,j}\right) \tag{13}$$

$$v^{(n+1)}_{i,\tilde{j}+1/2} = v^{(n)}_{i,\tilde{j}+1/2} + \frac{\Delta t}{6}\left(G^{(n,1)}_{i,\tilde{j}+1/2} + 2G^{(n,2)}_{i,\tilde{j}+1/2} + 2G^{(n,3)}_{i,\tilde{j}+1/2} + G^{(n,4)}_{i,\tilde{j}+1/2}\right) \tag{14}$$

$$p^{(n+1)}_{i,j} = p^{(n,4)}_{i,j} \tag{15}$$

for $i = 1, \ldots, N_x, \quad j = 1, \ldots, N_y,$ and $\tilde{i} = 1, \ldots, N_x - 1, \quad \tilde{j} = 1, \ldots, N_y - 1.$ The first argument $n$ in index pair $(n, \ell)$ indicates the time step (i.e., corresponding to time $t^n = n\Delta t$). The parameter $\ell$ corresponds to a particular stage of RK4 method, with $t^{n,1} = t^n$, $t^{n,2} = t^{n,3} = t^n + \Delta t/2$, $t^{n,4} = t^n + \Delta t$, where $\Delta t$ is the time-step size, and term $(F^{(n,\ell)}, G^{(n,\ell)})$ denotes the discrete vector of $\mathbf{R}(\mathbf{u}^{(n,\ell)}, p^{(n,\ell)}; t^{n,\ell})$ in space, for stages $\ell = 2, 3, 4$.

The discrete pressure correction Eq. (9) needs to be solved at each stage of the RK4 method. Discretizing Eq. (9) with Dirichlet, Neumann, Robin or/and mixed boundary conditions using fourth-order compact difference schemes, a sparse linear system

$$M\boldsymbol{\Delta p} = \boldsymbol{b}. \tag{16}$$

is arising, where the coefficient matrix $M$ has order $N_xN_y \times N_xN_y$. The application of a lexicographic ordering scheme for unknowns and equations results the coefficient matrix of form

$$M = \begin{bmatrix} A_1 & A_2 & A_3 & A_4 & O & \dots & O & O \\ A_5 & A_6 & A_5 & O & O & \dots & O & O \\ O & A_5 & A_6 & A_5 & O & \dots & O & O \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & \dots & O & A_5 & A_6 & A_5 & O \\ O & O & \dots & O & O & A_5 & A_6 & A_5 \\ O & O & \dots & O & \widehat{A_4} & \widehat{A_3} & \widehat{A_2} & \widehat{A_1} \end{bmatrix}. \tag{17}$$

Basic matrices $A_i$ for $i = 1, \dots, 6$ and $\widehat{A}_j$ for $j = 1, \dots, 4$ have the following structure

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & 0 & \dots & 0 & 0 \\ a_5 & a_6 & a_5 & 0 & 0 & \dots & 0 & 0 \\ 0 & a_5 & a_6 & a_5 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_5 & a_6 & a_5 & 0 \\ 0 & 0 & \dots & 0 & 0 & a_5 & a_6 & a_5 \\ 0 & 0 & \dots & 0 & \widehat{a_4} & \widehat{a_3} & \widehat{a_2} & \widehat{a_1} \end{bmatrix} \in \mathbb{R}^{N_x \times N_x} \quad . \tag{18}$$

It can be shown, that, in case of all sides of the computational domain possess the same boundary conditions, then matrices $\widehat{A_j}$ are equal to $A_j$ for $j = 1, \dots, 4$.

## 3 Multigrid-based solver

The pressure correction algebraic system (16) is sparse and large for realistic applications, where fine discretizations are required. Its efficient solution on parallel computing architectures suggests an iterative solution method that can be vectorized [9,25,30]. Multigrid methods are commonly used to accelerate convergence for solving PDEs using hierarchy grids. Furthermore, the convergence rates of multigrid methods are independent of the grid size. In multigrid iterations or cycles, coarser discretizations are being used in order to eliminate the lower frequency components of error by an iterative method called smoother. All calculations are performed on the coarse grids and significant computation time-saving can be achieved. The GMG technique comprises of the smoothing procedure and two grid-transfer operators. The first is the restriction operator, mapping the residual vector to a coarser grid, and the other called prolongation, interpolates the error from the coarser to a finer grid.

### 3.1 Multigrid smoother

Smoothing schemes that use red–black ordering of grid nodes are well suited for parallel computations. Block red–black Gauss–Seidel smoother (often called zebra relaxation) calculations can be carried out in parallel, since groups of red and black unknowns are decoupled. Another advantage of using zebra relaxation is that its smoothing factors and thus the multigrid convergence rates are better than those of lexicographic line Gauss–Seidel relaxations [34]. Therefore, we consider the horizontal zebra coloring scheme, resulting the pressure correction linear system, which can be written in the following partitioned form:

$$\begin{bmatrix} D_R & H_B \\ H_R & D_B \end{bmatrix} \begin{bmatrix} \Delta p_R \\ \Delta p_B \end{bmatrix} = \begin{bmatrix} \mathbf{b}_R \\ \mathbf{b}_B \end{bmatrix} \tag{19}$$

where

$$H_R = \begin{bmatrix} A_5 & A_5 & O & \dots & O & O & O \\ O & A_5 & A_5 & \dots & O & O & O \\ O & O & A_5 & \dots & O & O & O \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & O & \dots & A_5 & A_5 & O \\ O & O & O & \dots & O & A_5 & A_5 \\ O & O & O & \dots & O & \widehat{A}_4 & \widehat{A}_2 \end{bmatrix}, \quad H_B = \begin{bmatrix} A_2 & A_4 & O & \dots & O & O & O \\ A_5 & A_5 & O & \dots & O & O & O \\ O & A_5 & A_5 & \dots & O & O & O \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ O & O & O & \dots & A_5 & O & O \\ O & O & O & \dots & A_5 & A_5 & O \\ O & O & O & \dots & O & A_5 & A_5 \end{bmatrix}$$

and

$$D_W = \text{diag}[\tilde{A}_1 \ \underbrace{A_6 \ \cdots \ A_6}_{\frac{N_y}{2}-2}] \quad , \quad D_B = \text{diag}[\underbrace{A_6 \ \cdots \ A_6}_{\frac{N_y}{2}-2} \ \tilde{A}_2]$$

with

$$\tilde{A}_1 = \begin{bmatrix} A_1 & A_3 \\ O & A_6 \end{bmatrix} \quad \text{and} \quad \tilde{A}_2 = \begin{bmatrix} A_6 & O \\ \widehat{A}_3 & \widehat{A}_1 \end{bmatrix} .$$

The linear system's form (19) owns of increased scalability and parallelism properties [9,25,30], since the groups of unknowns can be processed on parallel. Each block of unknowns corresponds to an horizontal grid line. Their evaluation is possible with a linear system solution having the basic matrices $A_1$, $A_6$ or $\hat{A}_1$ as the coefficient matrix. Thus, choosing a parallel solver for them can further increase the degree of parallelism of our numerical method.

*Block cyclic reduction* The coefficient matrices $A_1$, $A_6$ or $\hat{A}_1$ of the basic linear systems have the nonzero entries $a_3$, $a_4$ and $\widehat{a_3}$, $\widehat{a_4}$ in the first and last lines. Their presence does not allow one to apply the Cyclic Reduction algorithm directly when solving a linear system with such coefficient matrices.
Assuming that $N_x = 2^q$, each basic linear system has the following block tridiagonal form

$$\begin{bmatrix} \tilde{D} & \tilde{C} & & & \\ B & D & C & & \\ & \ddots & \ddots & \ddots & \\ & & B & D & C \\ & & & \widehat{B} & \widehat{D} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{2^{q-1}-1} \\ \mathbf{u}_{2^{q-1}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{2^{q-1}-1} \\ \mathbf{b}_{2^{q-1}} \end{bmatrix} \tag{20}$$

where

$$\begin{aligned}
&\tilde{D} = \begin{bmatrix} a_1 & a_2 \\ a_5 & a_6 \end{bmatrix}, \quad \tilde{C} = \begin{bmatrix} a_3 & a_4 \\ a_5 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & a_5 \\ 0 & 0 \end{bmatrix} \\
&D = \begin{bmatrix} a_6 & a_5 \\ a_5 & a_6 \end{bmatrix}, \quad \widehat{D} = \begin{bmatrix} a_6 & a_5 \\ \widehat{a_2} & \widehat{a_1} \end{bmatrix}, \quad \widehat{B} = \begin{bmatrix} 0 & a_5 \\ \widehat{a_4} & \widehat{a_3} \end{bmatrix}, \quad C = B^{\mathsf{T}}.
\end{aligned} \tag{21}$$

Block Cyclic Reduction algorithm is chosen for its efficient parallel solution. The parallel algorithm proceeds in two phases: reduction and back substitution. During each step of the reduction stage, an elimination of half of the unknowns is performed. Based on the above matrix partition, after $q-1$ reductions an one block or a $2\times 2$ linear system solution is carried out. All previously eliminated unknowns are computed by the back-substitution procedure.
The reduction process generates the sequence of systems:

$$\begin{bmatrix} \tilde{D}^{(k)} & \tilde{C}^{(k)} & & \\ B^{(k)} & D^{(k)} & C^{(k)} & \\ & \ddots & \ddots & \ddots \\ & & \widehat{B}^{(k)} & \widehat{D}^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{1\cdot 2^k} \\ \mathbf{u}_{2\cdot 2^k} \\ \vdots \\ \mathbf{u}_{(2^{q-1-k}-1)\cdot 2^k} \\ \mathbf{u}_{(2^{q-1-k})\cdot 2^k} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1^{(k)} \\ \mathbf{b}_2^{(k)} \\ \vdots \\ \mathbf{b}_{(2^{q-1-k}-1)}^{(k)} \\ \mathbf{b}_{(2^{q-1-k})}^{(k)} \end{bmatrix} \tag{22}$$

for $k = 0, 1, \ldots, q-1$. The matrices and vectors are defined by means of the following recursions

$$\begin{aligned}
&B^{(k+1)} = -B^{(k)}(D^{(k)})^{-1}B^{(k)} \\
&D^{(k+1)} = D^{(k)} - B^{(k)}(D^{(k)})^{-1}C^{(k)} - C^{(k)}(D^{(k)})^{-1}B^{(k)} \\
&C^{(k+1)} = \tilde{C}^{(k+1)} = -C^{(k)}(D^{(k)})^{-1}C^{(k)} \\
&\tilde{D}^{(k+1)} = \tilde{D}^{(k)} - B^{(k)}(\tilde{D}^{(k)})^{-1}\tilde{C}^{(k)} - C^{(k)}(\tilde{D}^{(k)})^{-1}B^{(k)} \\
&\widehat{D}^{(k+1)} = \widehat{D}^{(k)} - \widehat{B}^{(k)}(D^{(k)})^{-1}C^{(k)} \\
&\widehat{B}^{(k+1)} = -\widehat{B}D^{-1}B \\
&\mathbf{b}_1^{(k+1)} = \mathbf{b}_2^{(k)} - B^{(k)}(\tilde{D}^{(k)})^{-1}\mathbf{b}_1^{(k)} - C^{(k)}(\tilde{D}^{(k)})^{-1}\mathbf{b}_3^{(k)} \\
&\mathbf{b}_j^{(k+1)} = \mathbf{b}_{2j}^{(k)} - B^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2j-1}^{(k)} - C^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2j+1}^{(k)} \quad j = 2, \ldots, 2^{q-1-k}-1 \\
&\mathbf{b}_{2^{q-1-k}}^{(k+1)} = \mathbf{b}_{2^{q-k}}^{(k)} - \widehat{B}^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2^{q-k}-1}^{(k)}
\end{aligned} \tag{23}$$

for $k = 0, 1, \ldots, q-2$. At the end of this process, the $2\times 2$ system $D^{(q-1)}\mathbf{u}_{2^{q-1}} = \mathbf{b}_1^{q-1}$ is solved.

Taking into account the structure of the matrices, one may easily verify that

$$\tilde{D}^{(k)} = \begin{bmatrix} a_1^{(k)} & a_2^{(k)} \\ a_5 & a_6^{(k)} \end{bmatrix}, \quad B^{(k)} = \begin{bmatrix} 0 & a_3^{(k)} \\ 0 & 0 \end{bmatrix}, \quad D^{(k)} = \begin{bmatrix} a_6^{(k)} & a_5 \\ a_5 & a_6^{(k)} \end{bmatrix},$$
$$\widehat{D}^{(k)} = \begin{bmatrix} a_6^{(k)} & a_5 \\ \widehat{a_2}^{(k)} & \widehat{a_1} \end{bmatrix}, \quad \widehat{B}^{(k)} = \begin{bmatrix} 0 & a_3^{(k)} \\ 0 & \widehat{a_3}^{(k)} \end{bmatrix}, \quad C^{(k)} = B^{(k)^{\mathsf{T}}}, \tag{24}$$

for $k = 1, \ldots, q-2$. It may also become clear that the structure for all above matrices is retained at every cyclic reduction level. Specifically, only 6 of elements need to be stored in each reduction step. Back substitution is performed evaluating the solution of the following linear systems

$$\begin{aligned} \tilde{D}^{(k)}\mathbf{u}_{2^k} &= b_1^{(k)} - \tilde{C}^{(k)}\mathbf{u}_{2^{k+1}} \\ D^{(k)}\mathbf{u}_{(2j-1)2^k} &= b_{2j-1}^{(k)} - B^{(k)}\mathbf{u}_{(2j-2)2^k} - C^{(k)}\mathbf{u}_{(2j)2^k} \end{aligned} \tag{25}$$

for $j = 2, \ldots, 2^{q-k-2}$ and $k = q-2, \ldots, 0$, starting with vector $\mathbf{u}_{2^{q-1}}$.

## 3.2 Multigrid strategy

The choice of multigrid cycling strategy (i.e., V, W or F-cycling) affects convergence rate and parallel performance of the pressure correction numerical scheme. Coarsest and middle grids, corresponding to small size problems, are frequently visited in each F- and W-cycle, making these strategies less effective on parallel environments. Further, each cycle includes more computation effort than the one in V-cycle. As shown in [26], the implemented intergrids transfer operators influence the convergence rates of the V-cycle algorithm. However, the authors propose a novel intergrid pair ($BCP$, $MR$), which ensures the good convergence rates of the V-cycle. Their application between two grids, $\Omega^h$ (fine) and $\Omega^H$ (coarse), will be presented below, under the consideration that the ratio of coarse to fine grid is two ($H = 2h$) as shown in Fig. 2.

*Bicubic prolongation operator and correction $\boldsymbol{e}_h = \boldsymbol{e}_h + P_H^h \boldsymbol{e}_H$* In the ascent phase of a multigrid technique, an interpolation (prolongation) formula is applied to represent the error correction computed on the coarse grid to a finer grid. In [6,26], a high-order interpolation (BCP) has been derived, using the fourth-order formula based on Lagrange's interpolation polynomial. The formulation is given by

$$\begin{aligned} e_{2i,2j}^h = e_{2i,2j}^h + \frac{1}{128^2}(&49e_{i-1,j-1}^H - 735e_{i,j-1}^H - 235e_{i+1,j-1}^H + 35e_{i+2,j-1}^H \\ &-735e_{i-1,j}^H + 11025e_{i,j}^H + 3675e_{i+1,j}^H - 525e_{i+2,j}^H \\ &-245e_{i-1,j+1}^H + 3675e_{i,j+1}^H + 1225e_{i+1,j+1}^H - 175e_{i+2,j+1}^H \\ &+35e_{i-1,j+2}^H - 525e_{i,j+2}^H - 175e_{i+1,j+2}^H + 25e_{i+2,j+2}^H \end{aligned} \tag{26}$$

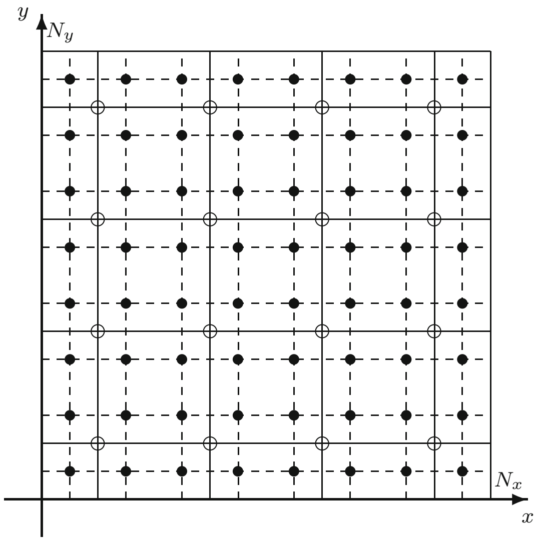

**Fig. 2** Coarse (*open circle*) and fine (*filled circle*) grid nodes values for a two-grid V-cycle

valid for all interior nodes. Error node values $e^h_{2i+1,2j}$, $e^h_{2i,2j+1}$, $e^h_{2i+1,2j+1}$ and nodes close to the boundary are similarly treated.

*Restriction operator* $\boldsymbol{r}_H = R^H_h \boldsymbol{r}_h = R^H_h(\boldsymbol{b}_h - M^h \Delta p_h)$ The reverse intergrid operator of prolongation is called restriction and involves the transfer of residual vector values from the fine grid $\boldsymbol{r_h}$ to a coarser grid $\boldsymbol{r_H}$ according to the scheme

$$r^h_{i,j} = \frac{1}{16}\Big(r^h_{2i-1,2j-1} + r^h_{2i+2,2j-1} + r^h_{2i-1,2j+2} + r^h_{2i+2,2j+2} \\ 3(r^h_{2i,2j} + r^h_{2i+1,2j} + r^h_{2i,2j+1} + r^h_{2i+1,2j+1})\Big) \tag{27}$$

valid for all interior nodes. In [6,26], one can find a more detail description of the restriction formula for all nodes including those close to the boundary.

*Unequal mesh size approach* The application of the multigrid technique with the partial semi-coarsening strategy [21,37] is described below. Considering that the predominant direction of the problem is on the $x$-axis, then only the dominant direction is coarsened till the mesh aspect ratio $\gamma$ is equal to 1 (i.e., $\Delta x = \Delta y$). Starting from this grid size, a multigrid technique is conducted with full coarsening (mesh coarsening performed in both directions). Multigrid with partial semi-coarsening strategy, use one-way restriction ($\widetilde{R}^H_h$) and interpolation ($\widetilde{P}^h_H$) operators, which are the analogous one dimensional of the above full-coarsening operators. Once the grid is reduced to an equal mesh size in both directions, the full coarsening operators can be applied. It is important to say that the aforementioned transfer operators are straightforward to parallelize, since the symbolic formulae for those operators are available.

**Algorithm 2** Parallel Multigrid Navier Stokes solver

```
Step 1.  CPU: Initialization of $u$, $v$ and $p$ vectors
Step 2.  CPU: Construction of $M^{(k)}$ matrices for all $k$ multigrid levels
Step 3.  CPU: Factorization of basic-matrices $A_1^{(k)}$, $A_6^{(k)}$, $\widehat{A}_1^{(k)}$ for all $k$ levels (stored
             in $fA_1^{(k)}$, $fA_6^{(k)}$, $f\widehat{A}_1^{(k)}$) based on the block cyclic reduction process
Step 4.  CPU to Accelerator Device Memory transfer: Copy vectors $u$, $v$, $p$
             and $M^{(k)}$, $fA_1^{(k)}$, $fA_6^{(k)}$, $f\widehat{A}_1^{(k)}$ for all $k$ levels
         while ($t^n < t_{final}$) do
Step 5.      acc_kernel: Evaluate the RHS vectors $\boldsymbol{F}(u, v, p; t^n)$, $\boldsymbol{G}(u, v, p; t^n)$
             for $\ell = 2$ to $\ell = 4$ do
                 while ($\nabla \cdot \mathbf{u}^{(\ell)} > tol$) do
Step 6.              acc_kernel: Evaluate vectors $u^{(\ell)}$, $v^{(\ell)}$ according to (7), (8)
Step 7.              acc_kernel: Evaluate the RHS vector for the Pressure Correction
                       Equation (9)
                     while ($\| \boldsymbol{b} - M\Delta p \| > tol$) do
Step 8.                  acc_kernel: $\Delta p = MG\_V(k, \Delta p, M^{(k)}, fA_1^{(k)}, fA_6^{(k)}, f\widehat{A}_1^{(k)}, \boldsymbol{b})$
                     end
Step 9.              acc_kernel: Correct $p^{(\ell)}$, $u^{(\ell)}$, $v^{(\ell)}$ according to (10), (11), (12)
             endfor
Step 10.     acc_kernel: Update $u$, $v$, $p$ at $t^n + \Delta t$ according to (13), (14), (15)
         end
Step 11. Accelerator Device to CPU Memory transfer: Copy vectors $u$, $v$, $p$
```

## 4 Parallel solution of incompressible Navier–Stokes equations

In the previous section, a multigrid-based iterative solver has been presented for solving the pressure correction equation, which is the most computationally intense part of the Navier–Stokes numerical scheme. This iterative solver consists of several numerical procedures with high degrees of parallelism. They compose a parallel algorithm (Algorithm 2) for an efficient solution of the incompressible Navier–Stokes equations on computing architectures with accelerators. The solver's algorithm is being ported to the OpenACC API.

It is important to point out that all computations are performed on the accelerator device. Only the initialization procedures are performed on the CPU (Steps 1–3), and their data are being send to the accelerator device performing the iterative solution procedure. When an acceptable approximation of pressure and velocities has been evaluated on the device, their values are being send back to the host's main memory. Steps 1–3 include initialization of velocity and pressure vectors and construction and cyclic reduction factorization of the basic matrices arising from the discretization of the pressure correction equation. These matrices are constructed for all $k$ levels of the multigrid cycle. Matrix-free storage method is preferred due to the block structure of all matrices (Eqs. 17–18). Their computations only involve 10 value entries for each matrix regardless of the multigrid level, which establishes this algorithm as very efficient in terms of storage. All basic linear algebra operations are suitably modified considering this matrix-free storage type.

**Algorithm 3** Parallel Multigrid V-cycle algorithm

```
acc_kernel :  $\Delta p^{(\lambda)} = MG\_V(\lambda, \Delta p^{(\lambda)}, M^{(\lambda)}, fA_1^{(\lambda)}, fA_6^{(\lambda)}, f\widehat{A}_1^{(\lambda)}, \boldsymbol{b}^{(\lambda)})$
  Presmoothing:
Step 1.  $\Delta p^{(k)} = ZebraGS(\Delta p^{(m)}, M^{(\lambda)}, \boldsymbol{b}^{(\lambda)}, fA_1^{(\lambda)}, fA_6^{(\lambda)}, f\widehat{A}_1^{(\lambda)}, v_1, tol)$
  Restriction:
         if $\gamma = 1$ then
Step 2.      $\boldsymbol{b}^{(\lambda-1)} = R_h^H(\boldsymbol{b}^{(\lambda)} - M^{(\lambda)}\Delta p^{(\lambda)})$
         elseif $\gamma < 1$ then
Step 3.      $\boldsymbol{b}^{(\lambda-1)} = \widetilde{R}_H^h(\boldsymbol{b}^{(\lambda)} - M^{\lambda}\Delta p^{(\lambda)})$
         end
  Recursion:
         if $\lambda = 1$ then
Step 4.      $\Delta p^{(1)} = ZebraGS(\Delta p^{(1)}, M^{(1)}, \boldsymbol{b}^{(1)}, fA_1^{(1)}, fA_6^{(1)}, f\widehat{A}_1^{(1)}, maxstep, tol)$
         elseif $k > 1$ then
Step 5.      $\Delta p^{(\lambda-1)} = MG\_V(\lambda-1, \boldsymbol{O}, M^{(\lambda-1)}, fA_1^{(\lambda-1)}, fA_6^{(\lambda-1)}, f\widehat{A}_1^{(\lambda-1)}, \boldsymbol{b}^{(\lambda-1)})$
         end
  Interpolation:
         if $\gamma = 1$ then
Step 6.      $\Delta p^{(\lambda)} = \Delta p^{(\lambda)} + P_H^h\Delta p^{(\lambda-1)}$
         elseif $\gamma < 1$ then
Step 7.      $\Delta p^{(\lambda)} = \Delta p^{(\lambda)} + \widetilde{P}_H^h\Delta p^{(\lambda-1)}$
         end
  Postsmoothing:
Step 8.  $\Delta p^{(\lambda)} = ZebraGS(\Delta p^{(\lambda)}, M^{(\lambda)}, \boldsymbol{b}^{(\lambda)}, fA_1^{(\lambda)}, fA_6^{(\lambda)}, f\widehat{A}_1^{(\lambda)}, v_2, tol)$
```

All iterative computations are performed on the accelerator device, including the RK4 time steps and the multigrid pressure correction technique. Steps 5–10 describe the RK4 computations with the solution of the pressure correction problem at each stage (Steps 5–9). There are also calculations (Step 10) for the update of pressure and velocities at each time step.

The parallel V-cycle multigrid technique for the pressure correction procedure can be described with the recursive algorithm (see Algorithm 3). Grid nodes are continually reduced in half, performing restriction, smoothing, prolongation and error correction phases calculating the new pressure approximation $\Delta p^{(\lambda)}$. Index $\lambda$ corresponds to the grid level, with $\lambda = 1$ implying coarsest grid. Values $v_1$ and $v_2$ correspond to the number of smoothing steps applied to every discrete problem of the form $M^{(\lambda)}\Delta p^{(\lambda)} = b^{(\lambda)}$ during the pre-smoothing and post-smoothing procedure, respectively. The fully parallelizable restriction and interpolation multigrid operators are applied in Steps 2, 3, 6 and 7. These transfer operators are not being constructed explicitly, instead a matrix-free scheme using matrix–vector multiplication is being implemented. The recursive application of multigrid technique at the coarser grid is applied in Step 5. The multigrid algorithm is concluded with the application of the smoother. This parallel procedure is applied twice (Steps 1 and 8), before the restriction and after the interpolation procedure, for every multigrid grid level $k$. At the coarsest grid level

$k = 1$ the smoother is also applied (Step 4) for the solution of the error pressure linear system.

Algorithm 4 presents the zebra $x$-line Gauss–Seidel smoothing procedure for the multigrid technique applied to the pressure correction linear system. In this parallel procedure a new approximation of the pressure error $\Delta p$ is calculated using an initial value $\Delta p^{\text{old}}$. The algorithm consists of two main computing phases evaluating the new pressure error approximation, based on the red–black Gauss–Seidel smoothing process. In the red phase (Steps 3a–4c), approximations that correspond to red pressure grid nodes are being calculated, involving the matrix–vector multiplication of Steps 3a–d and the direct block linear system solution of Steps 4a–c. The matrix–vector multiplication that involves matrix $H_B$ is a fully block parallelizable procedure. The computation involves matrix–vector multiplications with the basic matrices $A2$, $A_4$ and $A_5$ and some vector additions. All these basic linear computations are independent, and they can be performed efficiently on parallel architectures. In Steps 4a–c, a parallel block cyclic reduction is applied for the direct block linear system solution, increasing the parallelism of the algorithm. The second part of the algorithm has a similar structure that involves the calculated red approximations, matrix–vector multiplication with matrix $H_R$ and again a block cyclic reduction linear system solution. The total amount of these computations is parallelizable on block form. Moreover, matrix–vector multiplications where the basic matrices $A_5$, $\widehat{A_2}$ and $\widehat{A_4}$ are involved and vector additions include a second level of parallelism. This additional level also exists on the cyclic reduction solving procedure.

The block cyclic reduction direct solver's algorithm is presented in Algorithm 5. It comprises two parallel phases, the forward reduction of the right-hand side vector (Steps 2a–c) and the backward substitution (Steps 3a–c). Parallel processes appear at each reduction level $k$ (Steps 2b,3c). These processes involve $2 \times 2$ block matrix–vector multiplications, vector additions and linear system solution for specific matrix forms (see Eq. 24). This computation can be easily performed with closed formulas, optimizing the parallel computation in block level.

The computation in the above parallel Algorithms 2–5 entails two levels of parallelization. The first exploits the block structure of the data while the second takes advantage of the basic linear algebra operations within each block. For instant, the parallel computations of Step 6a in Algorithm 4 can performed as illustrated in Fig. 3. The first level of parallelization is on the block level for the evaluation of vector $s_i = A_5 \Delta p_i$. This parallel matrix–vector multiplication is partitioned in $\frac{N_y}{2}$ blocks of $N_x$ size. Each block operation is assigned into a $N_x$ size of group cores (Block Thread). Taking into account the structure of $A_5$ matrix, each core of every block thread evaluates a single element of vector $s_i$ using matrix values $a_1, a_2, a_3, a_4, a_5$ and $a_6$ (assuming Neumann boundary conditions). For this particular matrix–vector operation $\frac{N}{2} = \frac{N_y N_x}{2}$ core threads are needed. The work assigned to each core thread is represented in the oblong core boxes in Fig. 3. All other parallel block computations are assigned in a similar way to the acceleration cores.

The incompressible Navier–Stokes parallel solver's algorithm, designed in this section, has been implemented on parallel architectures with accelerators as present in the following section.

**Algorithm 4** Horizontal Zebra Gauss-Seidel smoother
`acc_kernel` : $\Delta p = ZebraGS(\Delta p^{old}, M, \boldsymbol{b}, fA_1, fA_6, \widehat{fA}_1, maxstep, tol)$

```
Step 1.  $\Delta p = \Delta p^{old}$ ; $istep = 0$
         do {
Step 2.      $\Delta p^{old} = \Delta p$ ; $istep = istep + 1$
  Red Phase
             Parallel Matrix-Vector Multiplication: $\Delta p_R = H_B \Delta p_R^{old}$
             for $i = 1$ to $i = N_y/2 - 1$ do in parallel
Step 3a.         $s_i = A_5 \Delta p_i^{old}$
             endfor
             for $i = 1$ to $i = N_y/2 - 1$ do in parallel
Step 3b.         $\Delta p_{i+1} = s_i + s_{i+1}$
             endfor
Step 3c.     $\Delta p_1 = A_2 \Delta p_1^{old} + A_4 \Delta p_2^{old}$
             for $i = 1$ to $i = N_y/2$ do in parallel
Step. 3d         $\Delta p_i = -\Delta p_i + \boldsymbol{b}_i$
             endfor
             Parallel Solution of $D_R \Delta p_R = \Delta p_R$
             for $i = 1$ to $i = N_y/2 - 1$ do in parallel
Step 4a.         $\Delta p_{i+1} = BCR(\Delta p_{i+1}, A_6, fA_6)$
             endfor
Step 4b.     $\Delta p_1 = -A_3 \Delta p_2 + \Delta p_1$
Step 4c.     $\Delta p_1 = BCR(\Delta p_1, A_1, fA_1)$
  Black phase
             for $i = N_y/2 + 1$ to $i = N_y$ do in parallel
Step. 5          $\Delta p_i = \boldsymbol{b}_i$
             endfor
             Matrix-Vector Operation: $\Delta p_B = -H_R \Delta p_R + b_B$
             for $i = 1$ to $i = N_y/2$ do in parallel
Step 6a.         $s_i = A_5 * \Delta p_i$
             endfor
             for $i = 1$ to $i = N_y/2 - 1$ do in parallel
Step 6b.         $\Delta p_{N_y/2+i} = \boldsymbol{b}_{N_y/2+i} - s_i - s_{i+1}$
             endfor
Step 6c.     $\Delta p_{N_y} = \boldsymbol{b}_{N_y} - \widehat{A_4} \Delta p_{N_y/2-1} - \widehat{A_2} \Delta p_{N_y/2}$
             Parallel Solution of $D_B \Delta p_B = \Delta p_B$
             for $i = N_y/2 + 1$ to $i = N_y$ do in parallel
Step 7a.         $\Delta p_i = BCR(\Delta p_i, A_6, fA_6)$
             endfor
Step 7b.     $\Delta p^{N_y} = -\widehat{A_3} \Delta p_{N_y-1} + \Delta p_{N_y}$
Step 7c.     $\Delta p_{N_y} = BCR(\Delta p_{N_y}, \widehat{A_1}, f\widehat{A_6})$
         }while ($\|\Delta p - \Delta p^{old}\| > tol$ or $istep < maxstep$)
```

**Algorithm 5** Parallel Block Cyclic Reduction

```
acc_kernel : $\mathbf{u} = BCR(\boldsymbol{b}, A, fA)$
Step 1.  $q = log_2(N_x)$
           Parallel Factorization of right hand side vector $\boldsymbol{b}$
         for $k = 0$ to $k = q - 2$ do
Step 2a.     $\mathbf{b}_1^{(k+1)} = \mathbf{b}_2^{(k)} - B^{(k)}(\tilde{D}^{(k)})^{-1}\mathbf{b}_1^{(k)} - C^{(k)}(\tilde{D}^{(k)})^{-1}\mathbf{b}_3^{(k)}$
             for $j = 2$ to $j = 2^{q-1-k} - 1$ do in parallel
Step 2b.         $\mathbf{b}_j^{(k+1)} = \mathbf{b}_{2j}^{(k)} - B^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2j-1}^{(k)} - C^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2j+1}^{(k)}$
             endfor
Step 2c.     $\mathbf{b}_{2^{q-1-k}}^{(k+1)} = \mathbf{b}_{2^{q-k}}^{(k)} - \widehat{B}^{(k)}(D^{(k)})^{-1}\mathbf{b}_{2^{q-k}-1}^{(k)}$
         endfor
           Parallel Backward-Substitution
Step 3a. Solve $D^{(q-1)}\mathbf{u}_{2^{q-1}} = \boldsymbol{b}_1^{(q-1)}$
         for $k = q - 2$ to $k = 0$ do
Step 3b.     Solve $\tilde{D}\mathbf{u}_{2^k} = \boldsymbol{b}_1^{(k)} - \tilde{C}^{(k)}\mathbf{u}_{2^{k+1}}$
             for $j = 2$ to $j = 2^{q-k-2}$ do in parallel
Step 3c.         Solve $D\mathbf{u}_{(2j-1)2^k} = \boldsymbol{b}_{2j-1}^{(k)} - B^{(k)}\mathbf{u}_{(2j-2)2^k} - C^{(k)}\mathbf{u}_{(2j)2^k}$
             endfor
         endfor
```

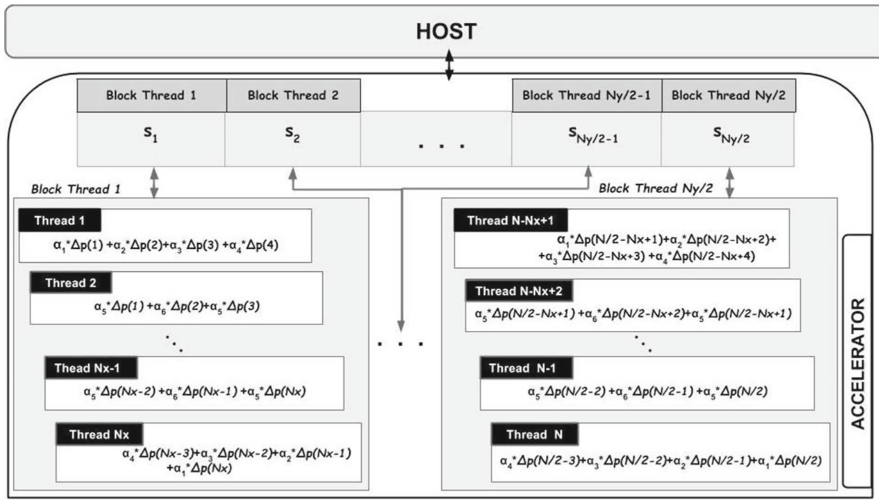
HOST
Block Thread 1
Block Thread 2
Block Thread Ny/2-1
Block Thread Ny/2
S1
S2
. . .
SNy/2-1
SNy/2
Block Thread 1
Block Thread Ny/2
Thread 1
α1*Δp(1) +α2*Δp(2)+α3*Δp(3) +α4*Δp(4)
Thread 2
α5*Δp(1) +α6*Δp(2)+α5*Δp(3)
Thead Nx-1
α5*Δp(Nx-2) +α6*Δp(Nx-1) +α5*Δp(Nx)
Thread Nx
α4*Δp(Nx-3)+α3*Δp(Nx-2)+α2*Δp(Nx-1) +α1*Δp(Nx)
Thread N-Nx+1
α1*Δp(N/2-Nx+1)+α2*Δp(N/2-Nx+2)+ +α3*Δp(N/2-Nx+3) +α4*Δp(N/2-Nx+4)
Thread N-Nx+2
α5*Δp(N/2-Nx+1) +α6*Δp(N/2-Nx+2)+α5*Δp(N/2-Nx+1)
Thread N-1
α5*Δp(N/2-2) +α6*Δp(N/2-1) +α5*Δp(N/2)
Thread N
α4*Δp(N/2-3)+α3*Δp(N/2-2)+α2*Δp(N/2-1)+α1*Δp(N/2)
ACCELERATOR

**Fig. 3** Parallelization approach for executing the Step 6a in Algorithm 4 on accelerator device. Each oblong *box* is handled solely by GPU core thread

## 5 Parallel implementation

The parallel performance of the Navier–Stokes solver for equal and unequal mesh size discretization problems is being investigating in this section. The application is developed in double precision Fortran code using the OpenMP and OpenACC APIs support for PGI-CDK 16.7 compiler's suite. In order to demonstrate the computational efficiency of the solver, realizations of the sequential single CPU implementations are compared to OpenMP multi-core CPU only and OpenACC device accelerator implementations. Compiler's option `-fast` is used, enabling compiler's optimization features, including the vectorization option. The basic linear algebra operations were performed using the BLAS scientific library for CPU implementations [1]. Three types of parallel machines with different accelerator device architectural design were selected using CUDA's version 7.5.

- The first machine is the HP SL390s server features two 6-core Xeon X5660@2.8 GHz processors, with 24 GB of memory and a Tesla M2070 GPU. The Nvidia's Fermi type M2070 GPU accelerator has 6 GB GDDR5 of memory and 448 cores organized in 14 multiprocessors attached on the host via an PCI-E Gen2 slot connection. The operating system is Oracle Linux 6.1.
- The second machine is the Dell R730 server features two 8-core Xeon E5-2630@2.4 GHz processors, with 16 GB of memory and a Tesla K40 GPU. The Nvidia's Kepler type K40 GPU accelerator has 12 GB GDDR5 of memory and 2880 cores organized on 15 multiprocessors attached on the host via an PCI-E Gen3 slot connection. The operating system is Ubuntu Linux 16.10.
- The third machine is the Dell R730 server features two 8-core Xeon E5-2695@2.3 GHz processors, with 64 GB of memory and a Tesla K80 GPU. The Nvidia's Kepler type K80 GPU accelerator has 24 GB GDDR5 of memory and 4992 cores organized on 26 multiprocessors attached on the host via an PCI-E Gen3 slot connection. The operating system is Ubuntu Linux 16.10.

Two classical incompressible flow test problems are solved. All reported measurements are the average values of 35 numerical solutions, using machines in standalone mode. The first is the driven-cavity flow steady-state simulation problem [14], and the other is the unsteady Stokes Oscillatory Plate [29].

In all realizations the time-step size is determined by CFL $= 0.75$, and the size of the coarsest mesh ($\lambda = 0$) of the GMG pressure correction process consists of $4 \times 4$ computing cells. The number of pre- and post-smoothing iterations in the descent and ascent phase of every multigrid cycle is $v_1 = 2$ and $v_2 = 1$, respectively.

### 5.1 Driven-cavity flow problem

The steady-state driven-cavity flow problem [14] has been widely used as a validation example in CFD field. The flow is contained within a unit square cavity, and no-slip boundary conditions are imposed on all walls, except on the upper wall, where $u = 1$ and $v = 0$. The boundary condition for pressure for all walls is obtained by assuming zero normal gradient ($\partial p/\partial n = 0$). The results obtained from the implementation are compared to the numerical data from Ghia et al. [14], in Fig. 4. The agreement with the benchmark data for $Re = 3200$ is acceptable. It is noteworthy that the fourth-order accuracy of the parallel solver is verified.

The performance of the Navier–Stokes solver on multi-core CPU only architectures yields no significant acceleration (see Fig. 5) for an increased number of CPU cores, since adding more than eight CPU cores leads to negligible performance improvement (see Fig. 5).

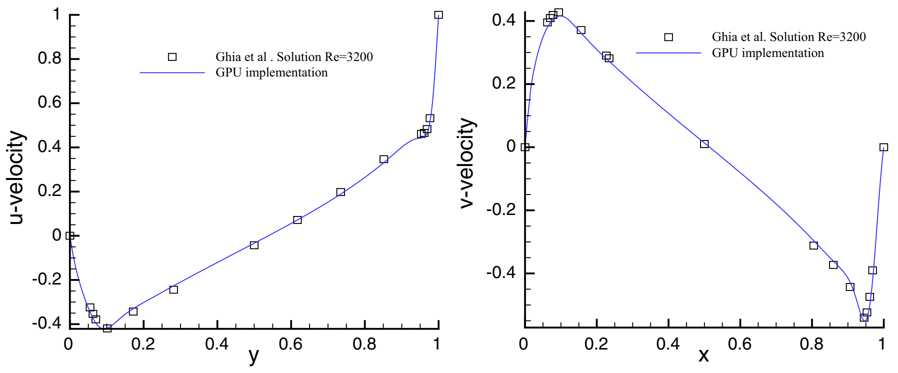

**Fig. 4** Comparison of velocity approximations with reference solutions by Ghia et al. [14] for $Re = 3200$ over a $1024 \times 1024$ mesh at $T = 60$; $\Delta t = 1/1500$

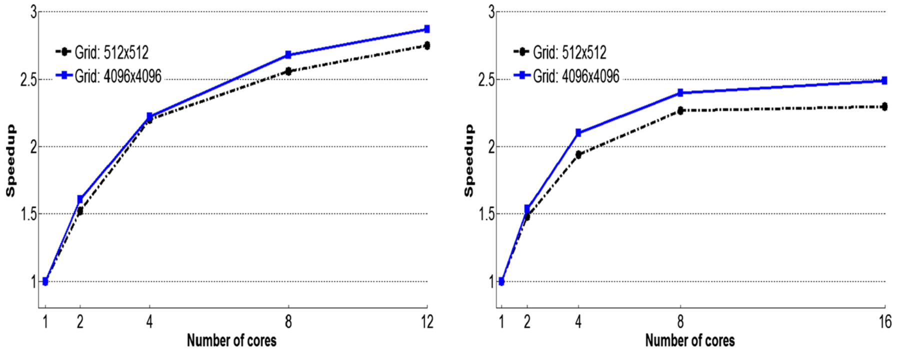

**Fig. 5** Speedup measurements for multi-core CPU only realizations. *Left* for the first computer choice. *Right* for the second computer choice; driven cavity flow ($Re = 3200$)

With respect to the GPU implementations, speedup plot measurements in Fig. 6 for the three types of computer realizations indicate that acceleration is accomplished for the pressure correction problem and for the entire Navier Stokes solver. The acceleration factor in GPU implementations over multi-core CPU only implementations is also measured for the first two computer choices.

The best GPU acceleration of the Navier–Stokes solver is about $11\times$ for the Tesla M2070 GPU, $15\times$ for the Tesla K40 GPU and $14\times$ for the Tesla K80 GPU types versus a single-core CPU. It is noticeable that the speedup measurements increase for finer discretization problems, where the computation is also increased. Accelerators' technical specifications differences (PCI bus connection type, number of cores and memory size) yield speedup differences among architecture implementations.

Speedup performance in case of GPU over multi-core CPU only implementations is observed to be less efficient than those over the single CPU core implementations as expected.

It can also noted that the pressure correction procedure has similar performance with the entire Navier–Stokes solver for all grid choices. This outcome may be explained

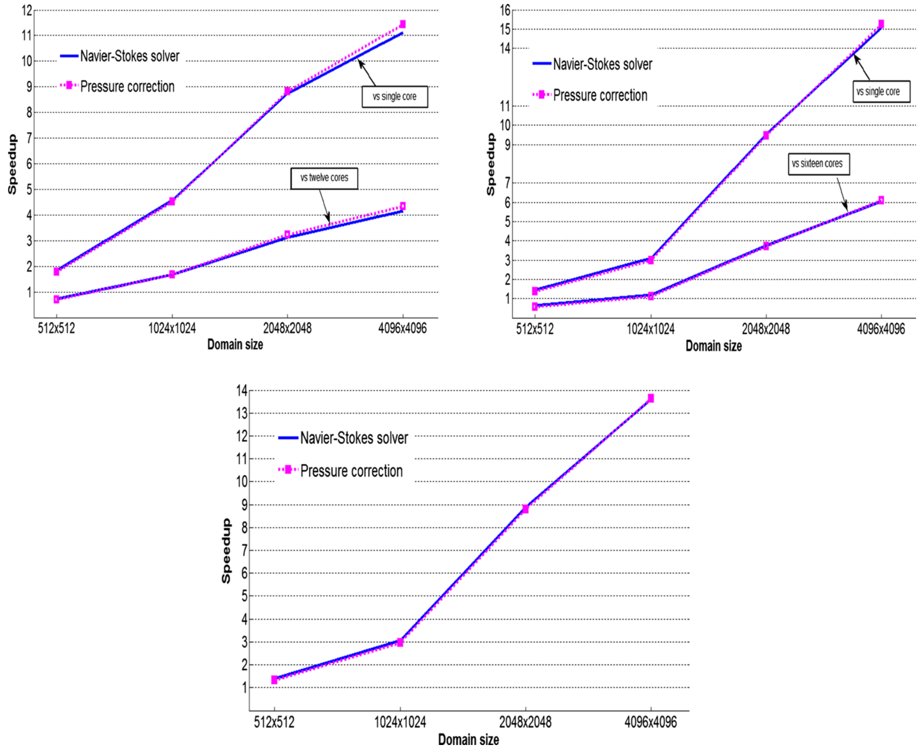

**Fig. 6** Pressure correction and entire Navier–Stokes solver speedups for GPU over single and multi-core CPU only realizations. *Top left* GPU type is M2070. *Top right* GPU type is Tesla K40. *Bottom* GPU type is Tesla K80; driven cavity flow ($Re = 3200$)

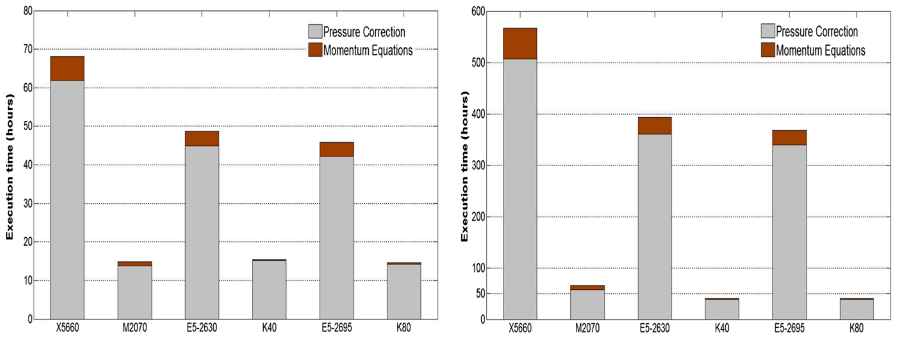

**Fig. 7** Execution time distribution for 1024 × 1024 (*right figure*) and 2048 × 2048 (*left figure*) discretization problems for single CPU and CPUGPU implementations; driven cavity flow ($Re = 3200$)

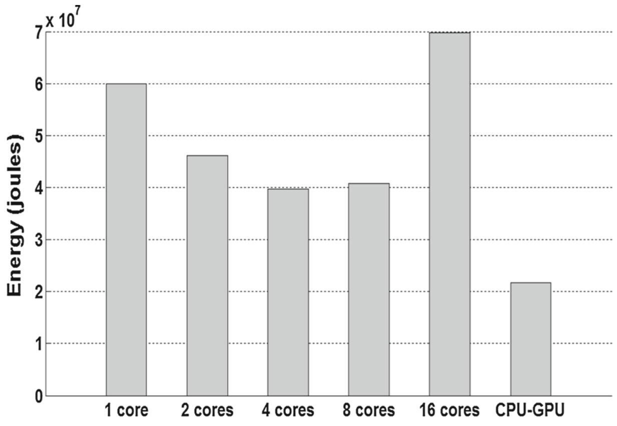

**Fig. 8** Energy measurements in Joules for the second machine choice; driven cavity flow 2048 × 2048 discretization ($Re = 3200$)

by estimating the time measurements (see Fig. 7). Execution time for solving the momentum equation is relatively small compared to perform the pressure correction procedure. This time difference becomes less for larger problem sizes. However, the pressure correction computation is still the dominant portion of the total runtime.

The energy consumption measurements of the algorithm is also investigated for the case of the driven cavity flow test problem. The energy monitoring tool likwid is used to measure the processor and memory energy consumption for CPU-only implementations, while for CPUGPU realizations the Nvidia-smi software is used. Figure 8 presents the energy cost for the second computer choice, solving the 2048 × 2048 problem size with all available computing resources. It seems that, enabling all 8 cores of the first processor, is the most energy efficient choice for the CPU-only implementation. Adding the second's processor cores the energy consumption is significantly increased. CPUGPU computation choice is the most energy efficient one among all available algorithm realizations.

**Table 1** V-cycle's subroutines time percentage (%) for the Tesla M2070 multigrid realization

| Mesh size | Residual (%) | Smoother (%) | Restriction (%) | Prolongation and correction (%) | Norms (%) |
|---|---|---|---|---|---|
| 512 × 512 | 3.72 | 79.18 | 1.75 | 1.94 | 13.41 |
| 1024 × 1024 | 5.96 | 77.53 | 2.36 | 2.19 | 11.96 |
| 2048 × 2048 | 9.42 | 74.05 | 3.3 | 2.51 | 10.72 |
| 4096 × 4096 | 12.02 | 72.07 | 3.92 | 2.74 | 9.25 |

**Table 2** V-cycle's subroutines time percentage (%) for the Tesla K40 multigrid realization

| Mesh size | Residual (%) | Smoother (%) | Restriction (%) | Prolongation and correction (%) | Norms (%) |
|---|---|---|---|---|---|
| 512 × 512 | 3.02 | 79.58 | 1.89 | 2.09 | 13.42 |
| 1024 × 1024 | 4.33 | 78.80 | 2.36 | 2.13 | 12.38 |
| 2048 × 2048 | 6.93 | 76.94 | 3.11 | 2.14 | 10.88 |
| 4096 × 4096 | 9.79 | 74.41 | 3.91 | 2.15 | 9.74 |

**Table 3** V-cycle's subroutines time percentage (%) for the Tesla K80 multigrid realization

| Mesh size | Residual (%) | Smoother (%) | Restriction (%) | Prolongation and correction (%) | Norms (%) |
|---|---|---|---|---|---|
| 512 × 512 | 2.96 | 78.55 | 2.00 | 2.11 | 14.38 |
| 1024 × 1024 | 4.17 | 78.95 | 2.37 | 2.06 | 12.45 |
| 2048 × 2048 | 6.64 | 76.17 | 3.3 | 2.10 | 11.79 |
| 4096 × 4096 | 9.35 | 74.79 | 4.01 | 2.07 | 9.78 |

Next, the performance of the pressure correction procedure is been investigated thoroughly. Tables 1, 2 and 3 present the percentage execution time measurements for every parallel computation component. The proportion data indicate that the most computationally intense part of the algorithm is the smoother process. The smoothing procedure takes about 70% of the computation in all cases. On the contrary, the intergrid operators go through $<7\%$ of the total kernel time. Among the V-cycle's subroutines, the residual function varies widely as the problem size increases due to the involved matrix–vector multiplication.

### 5.2 Stokes oscillating plate

The flow over an infinite oscillating plate (Stokes solution) is an unsteady incompressible flow problem, which fulfills the terms of anisotropy. The flow over the plate is

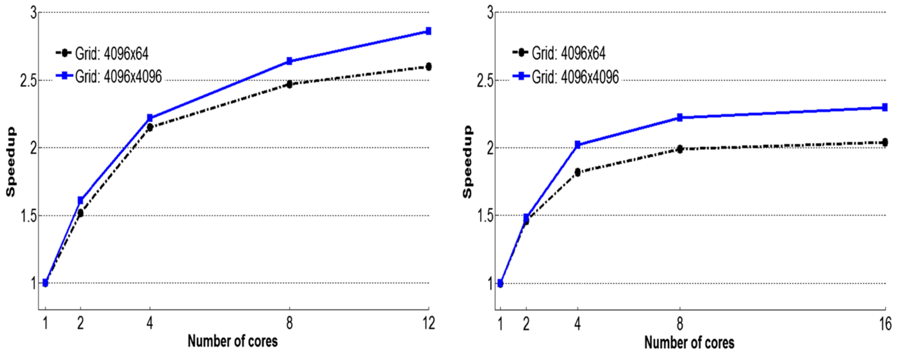

**Fig. 9** Speedup measurements for multi-core CPU only realizations. *Left* for the first computer choice. *Right* for the second computer choice; Stokes oscillatory plate ($Re = 5000$)

established after the plate starts an oscillatory motion (on the plate $x = 0$) with speed $v(0, y, t) = v_0 \cos \Omega t$.

The exact solution [29] for the time-varying velocity is:

$$v(x, y, t) = \mathrm{e}^{-x\sqrt{\frac{\Omega Re}{2}}} \cos\left(\Omega t - x\sqrt{\frac{\Omega Re}{2}}\right). \tag{28}$$

The parameter values $v_0 = 1$ and $\Omega = 2\pi$ are used. Periodic boundary conditions are imposed in the streamwise direction. On the plate $u = \partial p/\partial n = 0$ is considered. For $Re = 5000$, a fixed time interval was used ($\Delta t = 10^{-4}$) for all spatial discretizations. We discretize with mesh ratio $\gamma \leq 1$ ($\Delta x \leq \Delta y$), because the exact solution changes only in the $x$-direction.

This flow problem assesses the performance efficiency of the parallel solver over extremely anisotropic problems. Two multigrid strategies were tested, since the modeled physical quantity has uneven distribution in different directions. These are the full-coarsening and the partial semi-coarsening approaches.

As Fig. 9 shows, speedup measurements for CPU only multi-core realizations are close to 2.6× and 2.3×. Further, it seems that the partial semi-coarsening strategy (solid graph lines in Fig. 9) has similar parallel performance and scalability as the full coarsening (dash-dot graph lines in Fig. 9) for CPU only multi-core architectures.

The parallel performance of the CPUGPU partial semi-coarsening and full-coarsening multigrid Navier–Stokes solver is shown in Fig. 10. The speedup measurements for the three types of computer implementations are demonstrated against single CPU only and multi-core CPU only architectures. These measurements include both the pressure correction problem and the entire Navier–Stokes solver for the available parallel computer architectures. In particular, the overall speedup for the finest grid size (4096 × 4096) is about 10× for the Tesla M2070 GPU, 14.5× for the Tesla K40 GPU and 12.2× for the Tesla K80 GPU type over the single-core CPU solver realization. At first glance, it can be seen that the GPU performance implementation of the anisotropic algorithm is similar to the isotropic one for the cavity flow

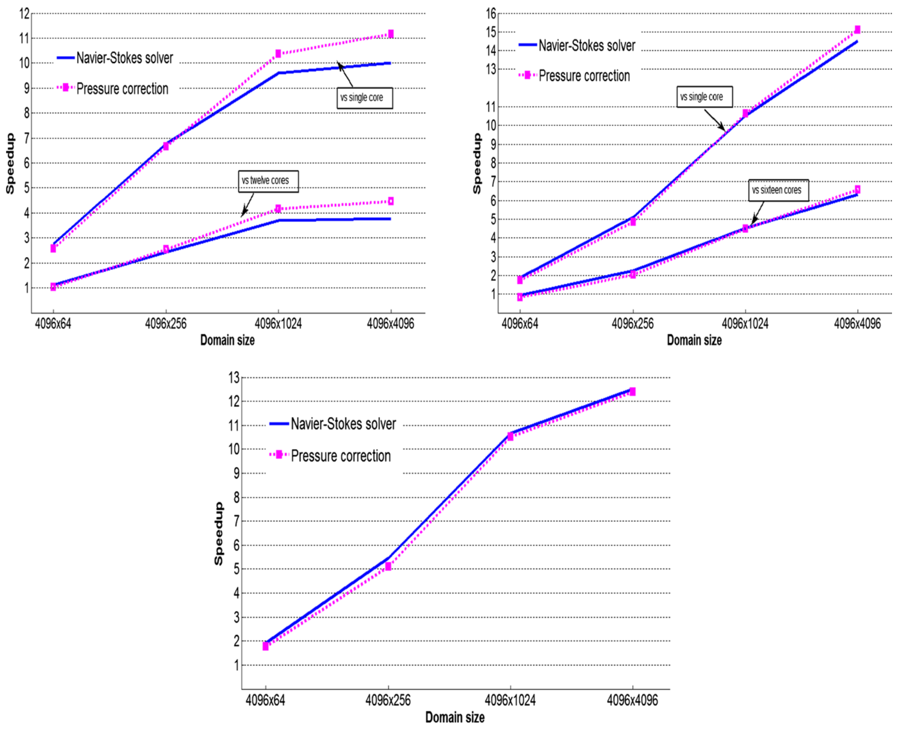

**Fig. 10** Pressure correction and entire Navier–Stokes solver speedups for GPU over single and multi-core CPU only realizations. *Top left* GPU type is M2070. *Top right* GPU type is Tesla K40. *Bottom* GPU type is Tesla K80; Stokes oscillatory plate ($Re = 5000$)

test problem. However, it is noticeable that in the first test case, the CPUGPU solver achieves a speedup slightly less than two in case of a $512 \times 512$ discretization, in the implementation that involves the first computer machine, and close to one for the second and third computers. In the anisotropic problem, in case of $4096 \times 64$ discretization size, a speedup slightly $<3$ for the first computer and close to two for the other computer types is, respectively, observed. These two different dimension problem cases have the same number of nodes (262,144), but the division of labor by the GPU in the anisotropic case leads to improved computation efficiency. In this anisotropic case less but larger basic matrices are involved in the computation, regarding the isotropic case.

It is also observed that the gap between speedup lines of pressure correction and the entire Navier–Stokes solver is larger for fine grid problems. To identify this discrepancy, the distribution of the execution time between pressure correction and momentum subroutines is presented in Fig. 11. The simulation execution time is measured in hours for two size problems, single CPU only and CPUGPU implementations for all computer choices. It is obvious that in the anisotropic case the momentum procedure has enlarged time percentage compared with the one in the isotropic case. This is due to the reduced number of multigrid cycles needed for the pressure correction subproblem. For example, one V-cycle is required at each Runge–Kutta stage

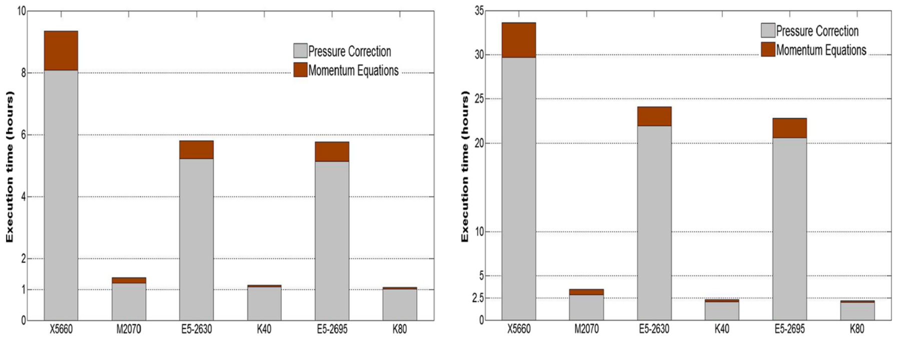

**Fig. 11** Execution time distribution of the CPUGPU Navier–Stokes solver for the Stokes oscillatory plate over 4096 × 64 and 4096 × 1024 mesh sizes at $T = 2.25$

for the anisotropic case, while for the isotropic problem at least three cycles are needed.

## 6 Conclusions

An efficient parallel algorithm of Navier–Stokes solver is designed and realized in this work. The solver is based on a high order of accuracy compact finite difference scheme, using the global pressure correction method. This highly efficient solver includes a multigrid technique, which is not suitable for implementations on parallel architectures. In order to increase parallelism the zebra numbering scheme is chosen, and the block cyclic reduction method is also applied for solving the arising linear subsystems. Taking account the structure of the finite difference matrix and the involved basic linear algebra operations, an extra level of parallelization appears in the parallel algorithm. All sub-matrices of the finite difference matrix are manipulated with their appropriate block entries, avoiding the entire matrix storage in the computation. This minimizes data storage and communication between the main computer and accelerator memories. Since accelerator devices have lower memory limitations, the proposed computation handling permits performing the whole computation on the acceleration device. The performance investigation of the parallel implementation for equal and unequal mesh discretizations resulted that multi-CPU only realizations are less efficient than realizations on computing architectures with acceleration devices. The performance of the parallel solver has been investigated on three types of GPU acceleration architectures, including the modern Kepler type and the legacy Fermi one. An acceleration of more than 10x is obtained for fine discretization problems in all test cases and computing architectures. These performance results of the solver encourage in the direction of extending the Navier–Stokes solver in the three-dimensional case on parallel architectures with accelerators. This extension may include problems defined on more complicated geometries, such as curvilinear coordinates.

**Acknowledgements** Thanks are due to Prof. C. Antonopoulos for his support providing access to the Tesla K80 GPU system of the University of Thessaly. We, also, gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

## References

1. BLAS (2017) http://www.netlib.org/blas/
2. Brandt A (1982) A guide to multigrid development in multigrid methods. Springer, Berlin
3. Briggs WL, Henson VE, McCormick S (2000) A multigrid tutorial. SIAM, Philadelphia
4. Butcher J (1987) The numerical analysis of ordinary differential equations: Runge–Kutta methods and general linear methods. Wiley, Chichester
5. Charalampaki NE, Mathioudakis EN (2014) CPU–GPU computations for MultiGrid techniques coupled with Fourth-Order Compact Discretizations for Isotropic and Anisotropic Poisson problems. In: Proceedings of the 6th International Conference on Numerical Analysis http://lib.amcl.tuc.gr/handle/triton/70
6. Christara C, Smitht B (1997) Multigrid and multilevel methods for quadratic spline collocation. BIT 37(4):781–803
7. Cohen J, Molemaker M (2009) A fast double precision CFD code using CUDA. In: Parallel CFD 2009
8. Dang DM, Christara C, Jackson KR (2009) A parallel implementation on GPUs of ADI finite difference methods for parabolic PDEs with applications in finance. Can Appl Math Q CAMQ 17(4):627–660
9. Dongarra J, Duff I, Sorensen D, van der Vorst H (1998) Numerical linear algebra for high-performance computers. SIAM, Philadelphia
10. Elsen E, LeGresley P, Darve E (2008) Large calculation of the flow over a hypersonic vehicle using a GPU. JCP 227(24):10148–10161
11. Feng C, Shu S, Xu J, Zhang CS (2014) Numerical study of geometric multigrid methods on CPU–GPU heterogeneous computers. Adv Appl Math Mech 6:1–23
12. Filelis-Papadopoulos CK, Gravvanis GA (2014) Parallel multigrid algorithms based on generic approximate sparse inverses: an SMP approach. J Supercomput 67(2):384–407
13. Gaitonde D, Visbal M (1998) High order schemes for Navier–Stokes equations: algorithm and implementation into FDL3DI. In: NASA, AFRL-VA-WP-TR-1998–3060
14. Ghia U, Ghia KN, Shin CT (1982) High-re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method. JCP 48:387–411
15. Gravvanis GA, Filelis-Papadopoulos CK (2014) On the multigrid cycle strategy with approximate inverse smoothing. Eng Comput 31(1):110–122
16. Hockney RW (1965) A fast direct solution of Poisson's equation using Fourier analysis. J ACM 12(1):95–113
17. Kampanis N, Ekaterinaris J (2006) A staggered grid, high-order accurate method for the incompressible Navier–Stokes equations. JCP 215:589–613
18. Karniadakis GE, Kirby RM (2003) Parallel scientific computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation. Cambridge University Press, New York
19. Karniadakis GE, Sherwin SJ (2005) Spectral/hp element methods for computational fluid dynamics. Oxford University Press, Oxford
20. Kass M, Lefohn A, Owens JD (2006) Interactive depth of field using simulated diffusion. Technical Report 06-01, Pixar Animation Studios
21. Larsson JF, Lien S, Yee E (2005) Conditional semicoarsening multigrid algorithm for the Poisson equation on anisotropic grids. JCP 208:368–383
22. Lele SK (1992) Compact finite difference schemes with spectral-like resolution. JCP 103:16–42
23. Mandikas V, Mathioudakis E, Papadopoulou E, Kampanis N (2013) In: Proceedings of the World Congress on Engineering 2013, vol 1, pp 74–79
24. Manea AM (2015) Parallel multigrid and multiscale flow solvers for high-performance-computing architectures. PhD thesis, Stanford University
25. Mathioudakis E, Papadopoulou E, Saridakis Y (2004) Iterative solution of elliptic collocation systems on a cognitive parallel computer. Comput Math Appl 48:951–970
26. Mathioudakis EN, Mandikas VG, Kozyrakis GV, Kampanis NA, Ekaterinaris JA (2017) Multigrid cell-centered techniques for high-order incompressible flow numerical solutions. Aerosp Sci Technol 64:85–101. doi:10.1016/j.ast.2017.01.015
27. Niemeyer KE, Sung CJ (2014) Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. J Supercomput 67(2):528–564
28. OpenFOAM (2017) http://www.openfoam.com
29. Panton RL (1984) Incompressible flow. Wiley, Chichester
30. Saad Y (2003) Iterative methods for sparse linear systems. SIAM, Philadelphia

31. Shinn A, Vanka S (2009) Implementation of a semi-implicit pressure-based multigrid fluid flow algorithm on a graphics processing unit. In: ASME Conference Proceedings 2009(43864), pp 125–133
32. The OpenACC Application Programming Interface (2017) http://www.openacc.org
33. Thibault JC, Senocak I (2009) CUDA implementation of a Navier–Stokes solver on multi-GPU desktop platforms for incompressible flows. In: 47th AIAA Aerospace Sciences Meeting doi:10.2514/6.2009-758
34. Trottenberg U, Oosterlee C, Schhüller A (2001) Multigrid. Elsevier Academic Press, New York
35. Vanka S, Shinn A, Sahu K (2011) Computational fluid dynamics using graphics processing units: challenges and opportunities. In: ASME Conference Proceedings 2011(54921), pp 429–437
36. Wesseling P (1992) An introduction to multigrid methods. Wiley, Chichester
37. Zhang J (2002) Multigrid method and fourth-order compact scheme for 2D poisson equation with unequal mesh-size discretization. JCP 179:170–179