Theoretical and Applied Mechanics Letters 13 (2023) 100474

Contents lists available at ScienceDirect

Theoretical and Applied Mechanics Letters

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

Full Article

An incompressible flow solver on a GPU/CPU heterogeneous architecture parallel computing platform

Qianqian Li[a,b], Rong Li[a,b], Zixuan Yang[a,b,*]

[a] The State Key Laboratory of Nonlinear Mechanics, Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China
[b] School of Engineering Sciences, University of Chinese Academy of Sciences, Beijing 100049, China

ARTICLE INFO

Keywords:
GPU Acceleration
Parallel computing
Poisson equation
Preconditioner

ABSTRACT

A computational fluid dynamics (CFD) solver for a GPU/CPU heterogeneous architecture parallel computing platform is developed to simulate incompressible flows on billion-level grid points. To solve the Poisson equation, the conjugate gradient method is used as a basic solver, and a Chebyshev method in combination with a Jacobi sub-preconditioner is used as a preconditioner. The developed CFD solver shows good performance on parallel efficiency, which exceeds 90% in the weak-scalability test when the number of grid points allocated to each GPU card is greater than 208^3. In the acceleration test, it is found that running a simulation with 1040^3 grid points on 125 GPU cards accelerates by 203.6x over the same number of CPU cores. The developed solver is then tested in the context of a two-dimensional lid-driven cavity flow and three-dimensional Taylor-Green vortex flow. The results are consistent with previous results in the literature.

1. Introduction

Computational Fluid Dynamics (CFD) is used in many engineering fields. To increase computational speed and size, parallel computing is becoming a regular technique used in almost all commercial and open-source CFD software. Along with the rapid development of hardware, parallel computing is experiencing the transition from the multi-core era to the many-core era. As a representative of the many-core system, the graphic processing unit (GPU) is used in an increasing number of CFD applications [1].

According to their governing equations, CFD solvers can be categorized into those for compressible and incompressible flows. For a compressible flow solver with an explicit time-marching scheme, the application of a GPU/CPU heterogeneous architecture parallel computing platform, hereinafter abbreviated as 'GPU/CPU platform', is relatively mature [2–5]. In contrast, the development of CFD solvers for incompressible flows on the GPU/CPU platform is limited by the frequent exchange of data during the solution of the pressure Poisson equation, which typically consumes over 95% of the total computational cost when the number of grid points is large. In this regard, the ability to simulate incompressible flows on the GPU/CPU platform is bounded by the performance of the Poisson equation solver, including its convergence performance, parallel efficiency, and computational cost. Although there are many iteration methods for solving or preconditioning linear equations, some algorithms, such as the Successive Over Relaxation (SOR) method, are not friendly to the GPU/CPU platform because of the sequential dependency of the solutions at different grid points in the same iteration step. Therefore, to build a CFD solver for incompressible flow, it is crucial to find a GPU-friendly algorithm to solve the Poisson equation.

There are some previous studies on linear equation solvers and preconditioners on the GPU/CPU platform. The Jacobi iteration method is a GPU-friendly algorithm, which was implemented by Thibault and Senocak [6] to build a CFD solver. They showed that running the code on four GPU cards accelerated the simulation by 100x over four CPU cores. However, the convergence performance of the Jacobi iteration method is usually unsatisfactory when the dimension of the linear system is large. The conjugate gradient (CG) algorithm [7], which is also GPU-friendly, often shows better convergence performance than the Jacobi iteration algorithm. The CG method was adopted by Zaspel and Griebel [8] to solve the Poisson equation, using the Jacobi iteration method as a preconditioner. Conducted on up to 48 GPU cards, the parallel efficiency of their method was reported as 41.3% and 95.3% in weak-scalability and strong-scalability tests, respectively. Oyarzun et al. [9] also used a Jacobi-preconditioned CG method to perform direct numerical simulations and large-eddy simulations of incompressible turbulence on unstructured mixed meshes. The simulation speed of their code on the GPU platform was 8 times faster than on the CPU platform. The multigrid algorithm is another GPU-friendly preconditioning method. Cohen and Molemaker [10] used a multigrid method to solve

[*] Corresponding author.
E-mail address: yangzx@imech.ac.cn (Z. Yang).

https://doi.org/10.1016/j.taml.2023.100474
Received 10 July 2023; Received in revised form 21 September 2023; Accepted 7 October 2023
Available online 10 October 2023

the Poisson equation on the GPU platform. Tested on one GPU card, their solver was found to accelerate the simulation with 28 million grid points by 8x versus an eight-core CPU. Molemaker et al. [11] compared the performance of the multigrid solver to the CG solver preconditioned by an incomplete Cholesky method. They reported that the preconditioned CG solver was unstable when the number of grid points exceeded 16 million. Jacobsen and Senocak developed an amalgamated parallel 3D geometric multigrid linear equation solver [12] and nested it into a CFD solver [13]. They conducted a weak-scalability test of the parallel efficiency of their CFD solver. In their test, they increased the number of grid points from 71 million to 18 billion, and correspondingly increased the number of GPU cards from 1 to 256. They found that when 256 GPU cards were used, the parallel efficiency dropped to 22%.

From the above overview of the existing incompressible flow CFD solvers on the GPU/CPU platform, it is understood that the multigrid method is efficient to solve the Poisson equation. An advantage of the multigrid method is that it provides excellent convergence performance in a small number of iteration steps. For example, in the test case of A 3D lid-driven cavity flow on 257^3 grid points [12], the residual divergence decreases by 10 to 12 orders of magnitude within 26 iteration steps using the multigrid method, while 6000 Jacobi iterations only reduce the residuals by less than one order of magnitude. On the other hand, many CFD solvers can tolerate certain magnitude of residual divergence without significantly changing the simulation results. In this sense, there is an opportunity to choose an alternative preconditioner to reduce computational cost and improve parallel efficiency. Although a different preconditioner is likely to result in a larger residual divergence than the multigrid method, it is possibly tolerable to the CFD solver. Therefore, we attempt to find an algorithm that is different from the multigrid method and meanwhile is friendly to the GPU/CPU heterogeneous-architecture platform to construct the complete solver for incompressible flow.

In the present study, we attempt to find out whether there are other preconditioner that can provide tolerable residual divergence with relatively low computational cost. The optimal preconditioner is then embedded into the incompressible flow solver to support large-size parallel computing on billion-level grid points. The remainder of this paper is organized as follows. In Section 2, we present the numerical method and tests of the Poisson equation solvers. In Section 3, numerical examples are given. In Section 4, the main finding of the present study is summarized.

2. Numerical methods and tests

2.1. Governing equations and discretization

The continuity and momentum equations for incompressible flows are expressed as

$$\frac{\partial u_j}{\partial x_j} = 0, \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}\left(u_j u_i\right) = -\frac{\partial P}{\partial x_i} + \boldsymbol{\nu}\frac{\partial^2 u_i}{\partial x_j \partial x_j}, \tag{2}$$

where x_i for $i = 1, 2, 3$ represent Cartesian coordinates, with u_i being the velocity components in the corresponding direction, t is the time, P is the pressure divided by the fluid density, and $\boldsymbol{\nu}$ is the kinematic viscosity. A finite difference scheme is used for spatial discretization. A second-order central-difference scheme is used to calculate the convection and diffusion terms in the momentum equation. A first-order explicit Euler scheme is used for time advancement. The projection algorithm [14] is adopted to satisfy the divergence-free condition. Specifically, to evolve the velocity from step s to $s+1$, the velocity field u^* is predicted using the momentum equation without the pressure gradient term as

$$u_i^* = u_i^s + \Delta t\left(-\frac{\partial}{\partial x_j}\left(u_j^s u_i^s\right) + \boldsymbol{\nu}\frac{\partial^2 u_i^s}{\partial x_j \partial x_j}\right), \tag{3}$$

Table 1
Parameters of preconditioners.

PC ID	**PC**	N_{PC}	**subPC**	N_{subPC}
01	-	-	-	-
02	Jacobi	1	-	-
03	Jacobi	5	-	-
04	Jacobi	10	-	-
05	Jacobi	15	-	-
06	Jacobi	20	-	-
07	Chebyshev	2	-	-
08	Chebyshev	2	Jacobi	1
09	Chebyshev	2	Jacobi	5
10	Chebyshev	2	Jacobi	10
11	Chebyshev	2	Jacobi	15
12	Chebyshev	2	Jacobi	20

where Δt represents the time step. The pressure is gained by solving the following Poisson equation.

$$\frac{\partial^2 P^{s+1}}{\partial x_j \partial x_j} = \frac{1}{\Delta t}\frac{\partial u_i^*}{\partial x_j}. \tag{4}$$

The velocity is then projected as

$$u_i^{s+1} = u_i^* - \Delta t\frac{\partial P^{s+1}}{\partial x_j}. \tag{5}$$

2.2. Poisson equation solver

To choose an optimal Poisson equation solver for the follow-up tests of the complete CFD solver, we conduct the simulation of the Taylor-Green Vortex (TGV) case (see Section 3 for details of case setup) for 1000 time steps to compare the computational time and residual divergence obtained from different preconditioners. The number of grid points is 1040^3, and 5^3 GPU cards are used for parallel computing.

The discretized Poisson equation is a large sparse linear equation system. The condition number of the coefficient matrix is typically large when the number of grid points is large. In this situation, a basic solver incorporated with a preconditioner is usually used to solve the linear equation system. In this study, we have implemented two basic solvers, namely the CG [7,15,16] and Generalized Minimum RESidual (GMRES) methods [16,17], and two preconditioners, namely the Jacobi and Chebyshev methods [16]. Table 1 lists all preconditioners tested in the present study. As shown, we have tested in total 12 preconditioners. **PC** 01 corresponds to a none preconditioner condition. **PC** 02–06 uses the Jacobi method to build the preconditioner with the iteration step N_{PC} ranging from 1 to 20. **PC** 07 uses the Chebyshev method. We note that in the Chebyshev iteration method, a sub-preconditioner (denoted by **subPC** in Table 1) can be used. Jacobi iteration method is chosen as the sub-preconditioner of Chebyshev iteration method, where the iteration step of the sub-preconditioner N_{SubPC} ranges from 1 to 20, yielding **PC** 08–12. Each preconditioner is nested into both the CG and GMRES solvers. Therefore, in total 24 Poisson equation solvers are tested.

The major consideration for choosing the above basic solvers and preconditioners is their implementation feasibility on a GPU platform. Some algorithms, such as the Gauss-Seidel (GS) and successive overrelaxation (SOR) methods, are efficiency on pure CPU architecture but cannot be implemented on a GPU/CPU heterogeneous architecture. In contrast, the Chebyshev iteration method is a GPU-friendly algorithm. To be specific, in the GS and SOR methods, the solution at each iteration step is updated sequentially from the first grid point to the last. This means that when different grid points are assigned to different processors, some processors need to wait the solution given by other processors. In the CG, GMRES, Chebyshev and Jacobi methods, the solution at step $(n+1)$ is only dependent on the solution before step $(n+1)$. When the solution before step $(n+1)$ is known, the solution at different flow regions can be assigned to different GPU cards to conduct the iteration from step n to step $(n+1)$ simultaneously. Therefore, these algorithms are friendly to parallel computing.

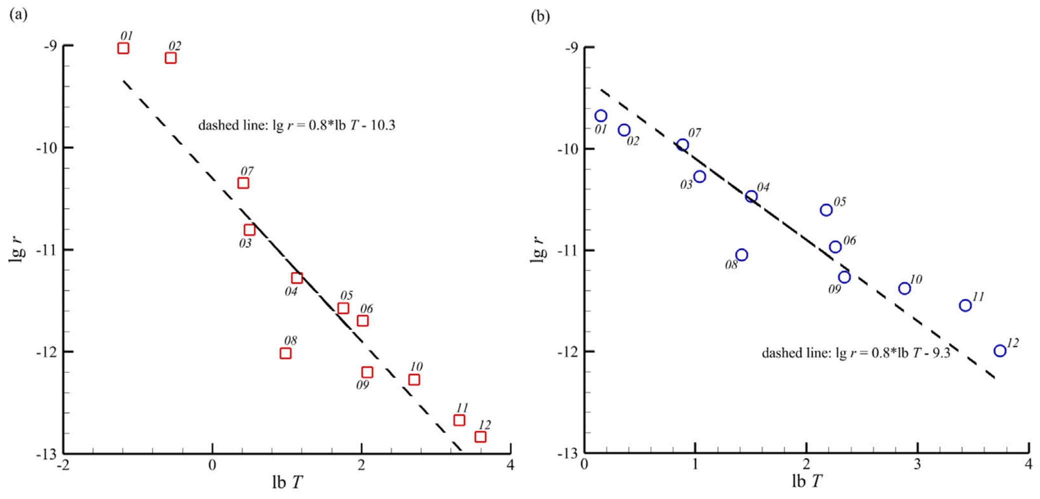

Fig. 1. Performance of different preconditioners shown in a $\lg r$ - lb T chart. Each symbol with an index number represents a preconditioner listed in Table 1. The dashed lines correspond to linear relationships between $\lg r$ and lb T. The basic solver applies the (a) CG and (b) GMRES iteration methods. The tests are conducted in the context of the TGV flow for 1000 time steps.

Figure 1 compares the performance of different preconditioners. The iteration steps for both CG and GMRES methods are set to 30 in these tests of the preconditioners. In the figures, each point corresponds to one preconditioner, marked by the **PC** numbers listed in Table 1. The horizontal and vertical axis values of each point give the time-consuming and convergence indices of the preconditioner, respectively. The time-consuming index is defined as lb T, with T being the averaging time used for solving a Poisson equation. The unit of T is second. The convergence index is defined as $\lg r$, where r is the non-dimensionalized residual divergence of the velocity field after the correction step given by Eq. (5). The characteristic velocity and length scales for performing the non-dimensionalization are given in Section 3. It is desired that a Poisson equation solver can use shorter time to provide smaller residual divergence. In this regard, if a preconditioner is closer than other ones to the left-bottom corner of the scatter plot shown in Fig. 1, it should be regarded as the optimal choice for constructing the Poisson equation solver in the problem under test.

As shown in Fig. 1(a), when the CG method is used as the basic solver, most preconditioners are located around a dashed line, expressed by $\lg r = 0.8$*lb T - 10.3. This means that if we change the preconditioner from one to the other among these ones, we can expect to reduce the residual divergence by $10^{0.8} \approx 6.3$ times by doubling the computational time. However, there is an exception, that is, **PC** 08, which locates to the left-bottom of the dashed line. To construct this preconditioner, the Chebyshev iteration method is conducted for two steps, while the Jacobi method is conducted for one iteration as the sub-preconditioner of the Chebyshev method. The observation from Fig. 1(b) gives a similar conclusion. Therefore, **PC** 08 is chosen as one of the preconditioners for constructing the complete Poisson equation solver. Another preconditioner that deserves some attention is **PC** 03. Compared with **PC** 02, it increases the iteration step of the Jacobi preconditioner from one to five, yielding significant reduction of the residual divergence when it is combined to the CG method. Therefore, **PC** 03 is also further considered in the follow-up tests of the complete Poisson equation solvers.

Thus far, we have chosen two preconditioners based on the tests in the context of the TGV flow on 1040^3 grid points. Combining these two preconditioners with CG and GMRES methods as the basic solvers yields four complete Poisson equation solvers, which are summarized in Table 2. These Poisson equation solvers are tested again in the context of the TGV flow by running the simulation for 1000 steps. In these tests, we change the iteration step of the basic solvers from 30 to 150 to further examine the possibility of reducing the residual divergence by increasing the iteration step of the basic solver. The GMRES method is restarted every 30 iteration steps. The test results are depicted in a $\lg r$ - lb T chart in Fig. 2. The four lines with different symbol patterns represent four Poisson equation solvers. The numbers along with the symbols give the number of iteration steps N_{iter} of the basic solvers.

Table 2
Parameters of complete Poisson equation solvers.

PES ID	Basic solver	**PC**	N_{PC}	**subPC**	N_{subPC}
01	CG	Jacobi	5	-	-
02	CG	Chebyshev	2	Jacobi	1
03	GMRES(m)	Jacobi	5	-	-
04	GMRES(m)	Chebyshev	2	Jacobi	1

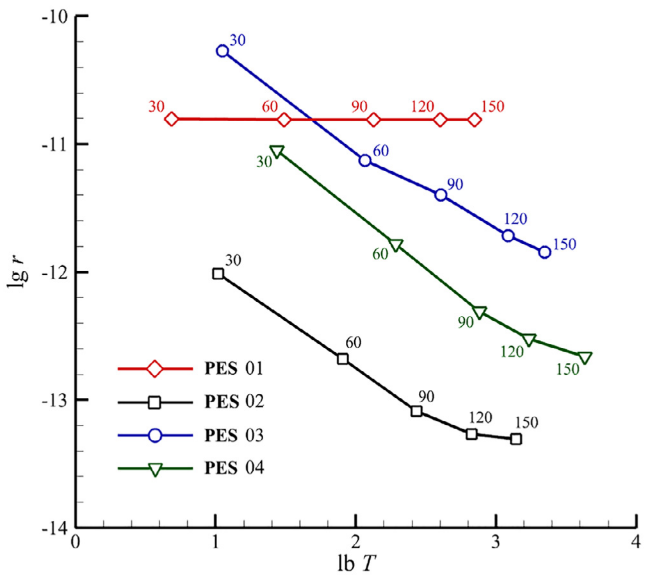

Fig. 2. Performance of different Poisson equation solvers shown in a $\lg r$ - lb T chart. Lines with different symbol patterns represent different Poisson equation solvers listed in Table 2. The numbers along with the symbols give the number of iteration steps per time step of the basic solvers. The tests are conducted in the context of the TGV flow for 1000 time steps.

As shown in Fig. 2, increasing the iteration step of **PES** 01 does not reduce the residual divergence. Therefore **PES** 01 is not strongly recommended unless the number of grid points is not very large. For the other three solvers, the residual divergence shows a similar trend of decrease by increasing the number of iteration step. The residual divergence given by **PES** 02 is smaller than that given by other Poisson

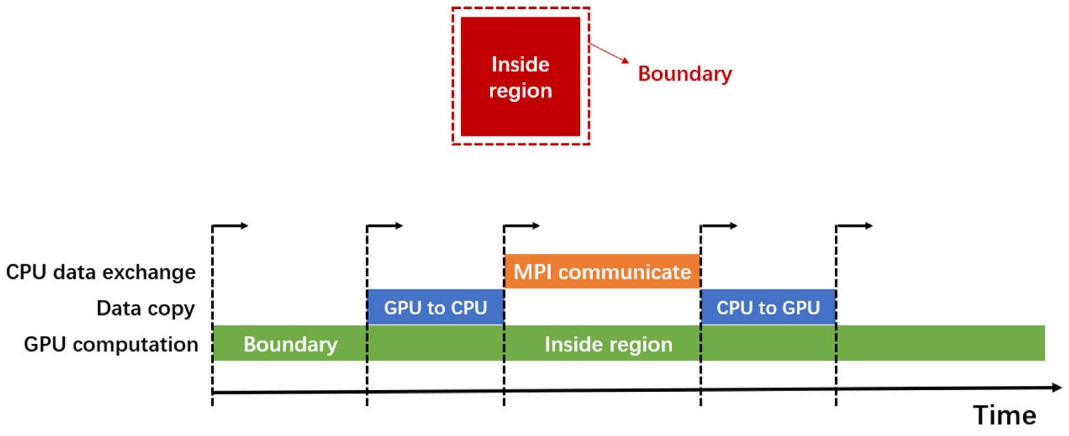

Fig. 3. Diagram of communication and calculation scheme.

equation solvers by using the same computational time. It is also noticed that increasing the iteration step from 120 to 150 does not continue to reduce the residual divergence. Therefore, **PES** 02 with 120 iteration steps is chosen to conduct the follow-up tests of the parallel efficiency and the acceleration ratio over a GPU/CPU platform.

2.3. Parallel programming

The code is developed using Compute Unified Device Architecture (CUDA) [18], a development environment and software architecture for general-purpose computing using the C programming language without requiring application programming interface of graphics. Many parallel machines employ the GPU/CPU heterogeneous architecture and the CUDA programming environment, containing both CPU and GPU components working together. The CPU and its memory are called the host and host memory, respectively, and the GPU and its memory are called the device and device memory, respectively. The CPU control unit and the cache unit are mainly responsible for performing complex logical processing and data communication between nodes, while the GPU mainly consists of processing units responsible for massive computing tasks. The MPI-GPU hierarchical parallel programming framework is usually used for this hardward architecture. Each MPI process is responsible for a subdomain and assigns computational tasks to the corresponding device for fine-grained parallel computing on a grid basis, while the information exchange tasks of the boundary grid during computation are executed at the host side via MPI.

In our code, the domain data are stored on GPU entirely. While CUDA streams provide an access to transfer data between multiple GPUs in parallel, it also adds complexity to the code and incurs runtime overheads. Since the size of our dataset is relatively small, the CUDA streams are not used. It should be noted that there is no connection channel between the GPUs, which means data exchange must be bridged through the host memory. Although this method may be slower than direct GPU-to-GPU communication, it is still sufficiently fast for our dataset size.

In the present parallel strategy, the computational domain is divided into sub-domains with the same number of grid points by 3D partitioning. This approach can effectively leverage the parallel computing power of GPUs. Each sub-domain is assigned to a GPU card to compute. To enable communication between different sub-domains, we use MPI processes that are responsible for data exchange and boundary grid information transfer. Each MPI process is connected to a single GPU card that is responsible for computing its associated sub-domain, and data exchange between MPI processes is conducted through the host memory. We also use the synchronous communication calculation scheme, as shown in Fig. 3.

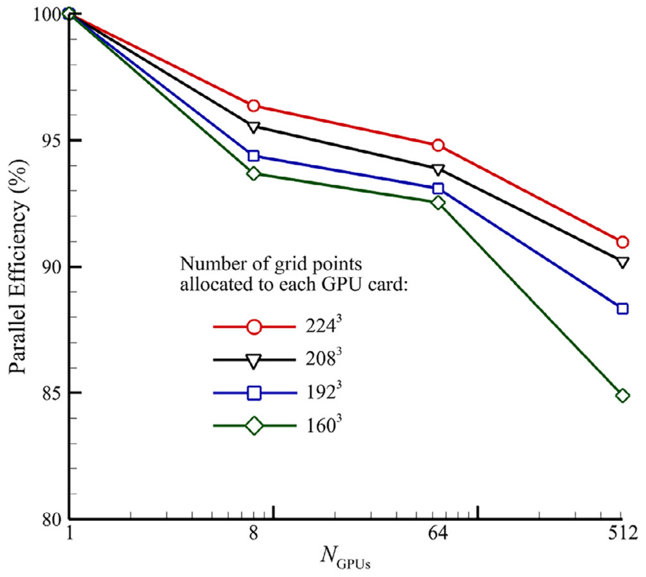

Fig. 4. Parallel efficiency in weak scalability tests. The scalability tests are conducted in four groups. the number of grid points allocated to each GPU card ranges from 160^3 to 224^3. All tests are conducted in the context of TGV flow.

2.4. Parallel efficiency

To further examine the parallel efficiency of the Poisson equation solver, we conduct four groups of weak scalability tests. In the weak scalability tests, the number of grid points allocated to each GPU card ranges from 160^3 to 224^3 in different groups. In each group of tests, the number of GPU cards varies from 1 to 8^3.

As shown in Fig. 4, the weak scalability test shows that when the number of grid points allocated to each GPU card is larger than 208^3, the parallel efficiency exceeds 90% when 512 GPU cards are used to conduct the parallel computing. From the parallel efficiency obtained from the weak scalability tests, it is evident that the computing load allocated to each GPU card needs to be sufficiently large to maximize the computing power of GPU.

2.5. Acceleration over a CPU platform

To further evaluate the performance of the Poisson equation solver on the GPU/CPU platform, we conduct a comparative test of the Poisson equation solver on a pure CPU platform. The tests are also conducted in the context of TGV flow on 1040 grid points. The test on the GPU/CPU platform uses 5^3 GPU cards. The tests on CPU platform use different number of CPU cores, ranging from 5^3 to 20^3. The values of averaging

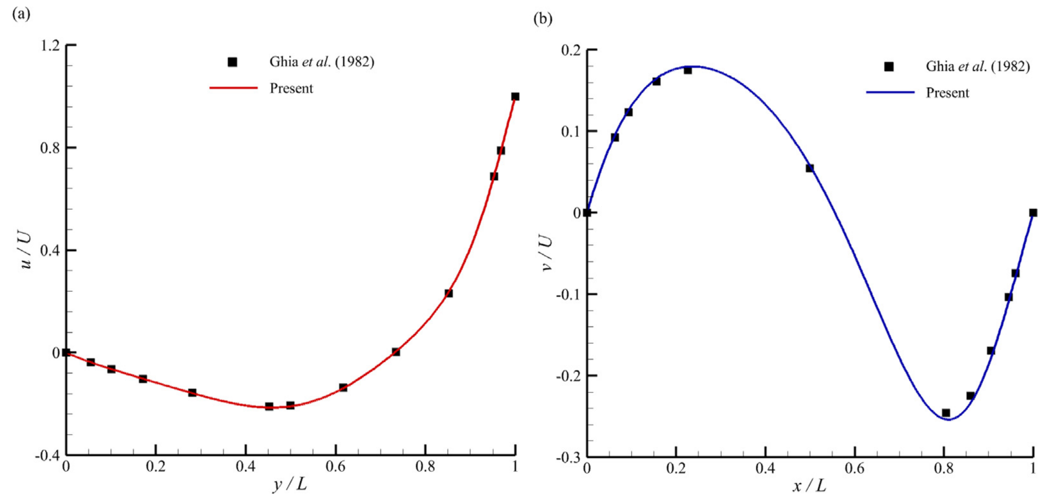

Fig. 5. Profiles of mean velocity in the lid-driven cavity flow. (a) Profiles of horizontal velocity u along the vertical direction y for $x = 0.5$, and (b) profiles of vertical velocity v along the horizontal direction x for $y = 0.5$.

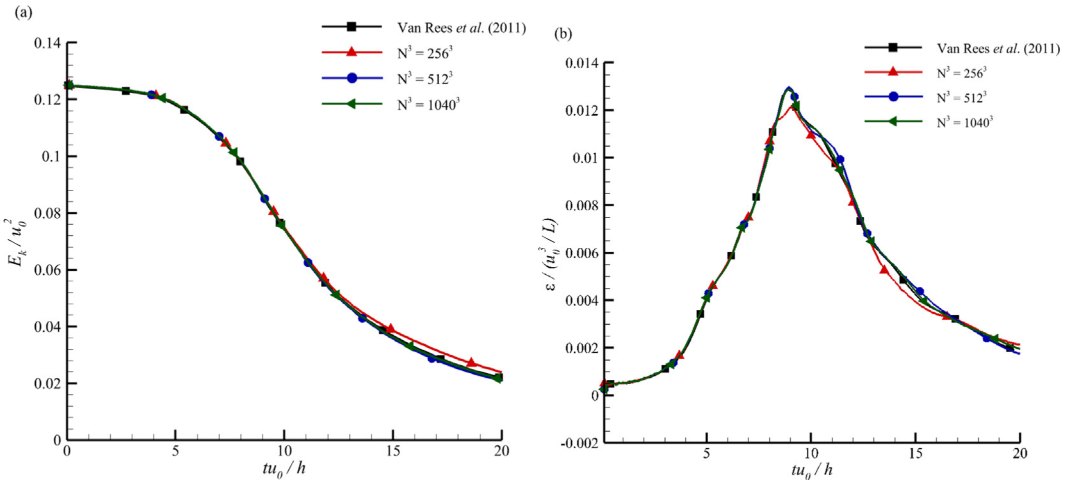

Fig. 6. Evolution history of (a) total kinetic energy and (b) dissipation rate obtained from the developed CFD solver using different number of grid points. The results of Van Rees et al. [20] are superimposed for validation.

Table 3
Acceleration ratio of a GPU/CPU platform over a pure CPU platform for running a case with 1040^3 grid points.

	Cores/Cards	Time (s)	Acceleration ratio
CPU	5x5x5	431.65	1.00
CPU	10x10x10	18.94	22.79
CPU	20x20x20	5.28	81.71
GPU	5x5x5	2.12	203.61

time used for running the simulation are listed in Table 3. It is seen that when the number of GPU cards of the GPU/CPU platform is identical to the number of CPU cores of the pure CPU platform, the GPU/CPU platform accelerates the simulation by approximately 203.6x. When the number of CPU cores increases to $20^3 = 8000$, the time cost on the CPU platform is still approximately $5.28/2.12 = 2.5$x that on the GPU/CPU platform. This test evidently shows the feasibility of accelerating the numerical simulations of incompressible flow with a large number of grid points using a GPU/CPU platform.

3. Numerical examples

Based on the above tests of the Poisson equation solvers, we choose **PES** 02 to conduct the following tests of the complete CFD solver for incompressible flow. The first test case is a two-dimensional lid-driven flow in a square cavity. In this test case, the computational domain is a square cavity with $[0, L] \times [0, L]$. The flow is driven by the lid at $y = L$ at a constant velocity U. At the other three boundaries, i.e., $x = 0$, $x = L$, and $y = 0$, a no slip condition is prescribed. Its convergence performance is sufficient for this problem with a small number of grid points. Figure 5 compares the horizontal velocity u/U at the vertical centerline $x = 0.5L$ and vertical velocity v/U at the horizontal centerline $y = 0.5L$ of the square cavity with the previous results of Ghia et al. [19]. A good agreement is observed from the figures. This test shows the correctness of the complete CFD solver.

The second test is the TGV flow. The computational domain is a cubic with $[0, 2\pi L]^3$. Periodic boundary condition is applied in all three directions. The initial velocity is given as

$$u_x = u_0 \cos(x/L)\sin(y/L)\sin(z/L), \tag{6}$$

$$u_y = -u_0 \sin(x/L)\cos(y/L)\sin(z/L), \tag{7}$$

$$u_z = 0, \tag{8}$$

where u_0 is the characteristic velocity scale. The Reynolds number is $Re = u_0 L/\nu = 1600$, where ν denotes the kinematic viscosity. The number of grid points varies from $N^3 = 256^3$ to 1040^3 to test the resolution convergence performance.

Figure 6 depicts the time history of the bulk kinetic energy E_k and dissipation rate ε obtained from the present solver. Here, E_k and ε are

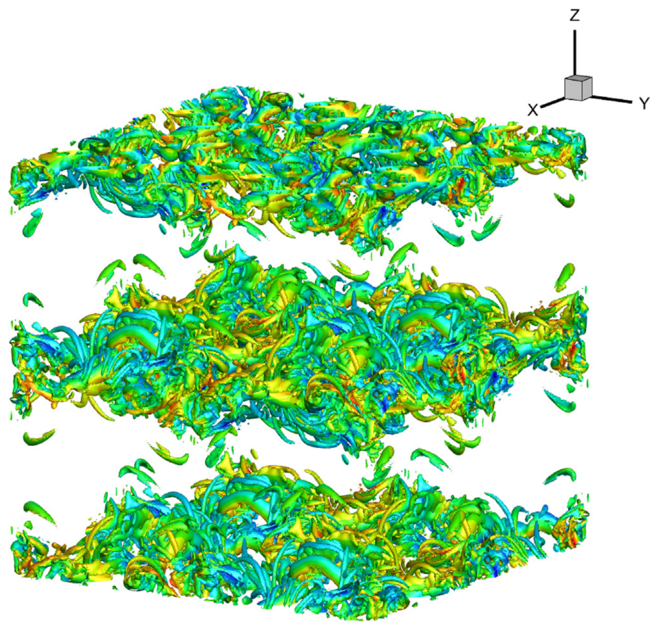

Fig. 7. Instantaneous vortex structure shown using the isosurface of $Q = 7.7$ at $t^* = 9$ of Taylor-Green vortex problem.

defined as

$$E_k = \frac{1}{|\Omega|}\int_\Omega \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2}\mathrm{d}\Omega, \tag{9}$$

and

$$\varepsilon = -\frac{\mathrm{d}E_k}{\mathrm{d}t}, \tag{10}$$

respectively. The results of Van Rees et al. [20] using 512^3 grid points are superimposed for validation. As shown in Fig. 6, both bulk kinetic energy E_k and dissipation ε are underestimated when 256^3 grid points are used to conduct the simulaiton. As the number of grid points increases to 512^3, the present results reach an agreement with previous DNS results. Figure 7 shows the instantaneous vortex structures visualized by the isosurface of $Q = 7.7$ at $t^* = 9$. The distribution of vortex structures is in agreement with the result of DeBonis [21].

4. Conclusion

In this paper, we describe a developed CFD solver for GPU/CPU heterogeneous architecture parallel computing platform, with specific focus on the choice of Poisson equation solvers. From a series of tests, the CG basic solver incorporated with a Chebyshev preconditioner with a one-step Jacobi sub-preconditioner is recognized as a high-efficiency Poisson equation solver. Its convergence performance is better than other solvers under test. Although the residual divergence given by the chosen Poisson equation solver is slightly larger than that given by a multi-grid preconditioner, it is tolerable to the CFD solver. Therefore, it is chosen for its simplicity in implementation and its good performance on the parallel efficiency. The weak-scalability tests show that the parallel efficiency of the chosen solver on 512 GPU cards with respect to one GPU card is higher than 90% when the number of grid points allocated to each GPU card is larger than 208^3. The present study shows a desirable future of simulating incompressible flows on GPU/CPU platform.

As a final remark of this paper, we note that the present study aims to verify the feasibility of solving the incompressible equations on GPU/CPU platform to accelerate the simulation. This point is supported by the test results shown in this paper. However, we fully recognize that the current solver only supports Cartesian grids and does not support complex boundaries. This is a limitation of the current work, and test of complex cases is assumed to be an important part of future work. Testing the parallel efficiency on a larger number of grid points is also desired in the future.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The numerical calculations in this study were carried out on the ORISE Supercomputer. This research is supported by the National Natural Science Foundation of China (NSFC). Basic Science Center Program for Multiscale Problems in Nonlinear Mechanics'(Grant No. 11988102) and NSFC project (Grant No. 11972038).

Appendix

This appendix provides the algorithms tested in the present study.

Algorithm A: preconditioned CG Method

To solve a linear equation $Ax = b$ with an initial guess x_0, the preconditioned CG method [7,15,22] is given below.

1. Initialization

$$r_0 = b - Ax_0, \tag{A1}$$

$$Az_0 = r_0, \tag{A2}$$

$$p_0 = z_0. \tag{A3}$$

2. Iteration for $k = 0, 1, 2,...$

$$\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}, \tag{A4}$$

$$x_{k+1} = x_k + \alpha_k p_k, \tag{A5}$$

$$r_{k+1} = r_k - \alpha_k A p_k. \tag{A6}$$

If r_{k+1} is smaller than the tolerance, then stop the iteration. Otherwise, continue the following calculations

$$Az_{k+1} = r_{k+1}, \tag{A7}$$

$$\beta_k = \frac{z_{k+1}^T r_{k+1}}{z_k^T r_k}, \tag{A8}$$

$$p_{k+1} = z_{k+1} + \beta_{k+1} p_k. \tag{A9}$$

The solution is given by x_{k+1}. Equations (A2) and (A7) are known as the preconditioning, solved using the Jacobi or Chebyshev preconditioners in the present study.

Algorithm B: GMRES Method with precondition

The preconditioned GMRES method [16,17] utilized in the present study is given below.

1. Initialization

$$r_0 = b - Ax_0, \tag{B1}$$

$$Ar_0{}' = r_0, \tag{B2}$$

$$v_1{}' = r_0{}'/\|r_0{}'\|. \tag{B3}$$

2. Iteration for $j = 1, 2, \dots, m$,

$$h_{i,j} = \left(A v_j{}', v_i{}' \right), i = 1, 2, \dots, j, \tag{B4}$$

$$\hat{v}_{j+1} = A v_j{}' - \sum_{i=1}^{j} h_{i,j} v_i, \tag{B5}$$

$$A \hat{v}_{j+1}{}' = \hat{v}_{j+1}, \tag{B6}$$

$$h_{j+1,j} = \|\hat{v}'_{j+1}\|, v_{j+1} = \hat{v}'_{j+1} / h_{j+1,j}. \tag{B7}$$

3. Calculate the approximate solution

$$x_m = x_0 + V_m y_m, \quad \text{where } y_m \text{ minimizes } J(y) = \|\beta e_1 - \overline{H}_m y\|. \tag{B8}$$

4. Restart

Calculate the residual

$$r_m = b - A x_m. \tag{B9}$$

If $||r_m||$ is smaller than the tolerance, then stop the algorithm. Otherwise let $x_0 = x_m$ and go to step 2. The solution is given by $x = x_m$. The preconditioning given by Eqs. (B2) and (B6) is solved using Jacobi or Chebyshev preconditioners in the present study.

Algorithm C: Chebyshev Method with sub-precondition

In the present study, the Chebyshev method [16] is only used as a preconditioner to solve Eqs. (A2), (A7), (B2), and (B6). It is not used as a basic solver to solve the Poisson equation. However, to express the Chebyshev iteration algorithm, we continue using the general form of a linear equation $Ax = b$. The Chebyshev iteration method is given below.

1.Start: Select x_0 and calculate

$$r_0 = b - A x_0, \tag{C1}$$

$$A r_0{}' = r_0, \tag{C2}$$

$$\sigma_1 = \frac{\delta}{\theta}, \quad \rho_0 = \frac{1}{\sigma_1}, \quad d_0 = \frac{1}{\theta} r_0{}'. \tag{C3}$$

The iteration parameters δ and θ are determined as

$$\delta = \frac{\lambda_1 - \lambda_n}{2}, \quad \theta = \frac{\lambda_1 + \lambda_n}{2}, \tag{C4}$$

where λ_1 and λ_n are the maximum and minimum eigenvalues of matrix A, respectively. The eigenvalues of the matrix are obtained approximately from a GMRES method with a Jacobi preconditioner [6]. The GMRES iteration is conducted for ten steps to obtain ten eigenvalues, from which the maximum and minimum are chosen.

2.Iteration: For $k = 0, 1, \dots,$

$$x_{k+1} = x_k + d_k, \tag{C5}$$

$$r_{k+1} = r_k - A d_k. \tag{C6}$$

If $||r_{k+1}||$ is smaller than the tolerance, then stop the algorithm. Otherwise, continue the following calculations

$$A r_{k+1}{}' = r_{k+1}, \tag{C7}$$

$$\rho_{k+1} = \left(2\sigma_1 - \rho_k \right)^{-1}, \tag{C8}$$

$$d_{k+1} = \rho_{k+1} \rho_k d_k - \frac{2\rho_{k+1}}{\delta} r_{k+1}{}'. \tag{C9}$$

Equations (C.2) and (C.6) are solved by a Jacobi sub-preconditioner. The solution is given by x_{k+1}.

References

[1] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, in: Computer graphics forum, volume 26, Wiley Online Library, 2007, pp. 80–113.
[2] E. Phillips, Y. Zhang, R. Davis, J. Owens, Rapid aerodynamic performance prediction on a cluster of graphics processing units, in: 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, 2009, p. 565.
[3] A. Corrigan, F.F. Camelli, R. Löhner, J. Wallin, Running unstructured grid-based CFD solvers on modern graphics hardware, Int. J. Numer. Methods Fluids 66 (2) (2011) 221–229.
[4] L. Fu, Z. Gao, K. Xu, F. Xu, A multi-block viscous flow solver based on GPU parallel methodology, Comput. Fluids 95 (2014) 19–39.
[5] J. Lai, Z. Tian, H. Li, S. Pan, A CFD heterogeneous parallel solver based on collaborating CPU and GPU, in: IOP Conference Series: Materials Science and Engineering, volume 326, IOP Publishing, 2018, p. 012012.
[6] J. Thibault, I. Senocak, Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows, in: 47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition, 2009, p. 758.
[7] M.R. Hestenes, E. Stiefel, et al., Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand. (1934) 49 (6) (1952) 409–436.
[8] P. Zaspel, M. Griebel, Solving incompressible two-phase flows on multi-GPU clusters, Comput. Fluids 80 (2013) 356–364.
[9] G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, Portable implementation model for CFD simulations. application to hybrid CPU/GPU supercomputers, Int. J. Comut. Fluid Dyn. 31 (9) (2017) 396–411.
[10] J. Cohen, M.J. Molemaker, A fast double precision CFD code using CUDA, Parallel Comput. Fluid Dyn.: Recent Adv. Future Direct. (2009) 414–429.
[11] J. Molemaker, J.M. Cohen, S. Patel, J. Noh, et al., Low viscosity flow simulations for animation, Symposium on Computer Animation, volume 2008, 2008.
[12] D. Jacobsen, I. Senocak, A full-depth amalgamated parallel 3d geometric multigrid solver for GPU clusters, in: 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2011, p. 946.
[13] D.A. Jacobsen, I. Senocak, Multi-level parallelism for incompressible flow computations on GPU clusters, Parallel Comput. 39 (1) (2013) 1–20.
[14] A.J. Chorin, Numerical solution of the navier-stokes equations, Math. Comput. 22 (104) (1968) 745–762.
[15] D.M. Young, K.C. Jea, T.-Z. Mai, Preconditioned Conjugate Gradient Algorithms and Software for Solving Large Sparse Linear Systems, Technical Report, Texas Univ., Austin (USA). Center for Numerical Analysis, 1987.
[16] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[17] Y. Saad, M.H. Schultz, Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.
[18] Cuda c programming guide [eb/ol], 2012.12/2013.06, http://docs.nvidia.com/cuda/cuda-c-programming-guide/.
[19] U. Ghia, K.N. Ghia, C.T. Shin, High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method, J. Comput. Phys. 48 (3) (1982) 387–411.
[20] W.M. Van Rees, A. Leonard, D.I. Pullin, P. Koumoutsakos, A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high reynolds numbers, J. Comput. Phys. 230 (8) (2011) 2794–2805.
[21] J. DeBonis, Solutions of the taylor-green vortex problem using high-resolution explicit finite difference methods, in: 51st AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition, 2013, p. 382.
[22] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-GPU platform, in: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE, 2010, pp. 583–592.