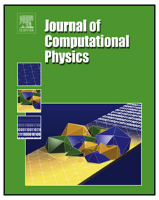
Journal of Computational Physics 414 (2020) 109447

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp



A parallel nonlinear multigrid solver for unsteady incompressible flow simulation on multi-GPU cluster

Xiaolei Shi [a], Tanmay Agrawal [a], Chao-An Lin [a,*], Feng-Nan Hwang [b], Tzu-Hsuan Chiu [a]

[a] Department of Power Mechanical Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan
[b] Department of Mathematics, National Central University, Taoyuan 32001, Taiwan

A R T I C L E   I N F O

Article history:
Received 19 March 2019
Received in revised form 6 February 2020
Accepted 29 March 2020
Available online 18 April 2020

Keywords:
FAS V-cycle scheme
Artificial compressibility method
Dual-time stepping
Multi-GPU
Navier-Stokes equations
Incompressible flow

A B S T R A C T

A nonlinear multigrid solver for solutions of unsteady three-dimensional incompressible viscous flow working on multi-GPU cluster is developed. The solver consists of a full approximation scheme (FAS) V-cycle scheme to accelerate the computation, in which the artificial compressibility method based Navier-Stokes solver is used as a smoother. Multi-stream overlapping strategies are designed to assist multi-GPU computations. The numerical procedure is validated by computing 3D laminar and turbulent flows within a lid-driven cubic cavity. The predicted results compare favorably with previous benchmark solutions and measurements, both in mean and turbulent quantities. For the performance of the FAS V-cycle scheme, up to two orders of magnitude speedups are reported, and the relationship between work unit (WU) and total grid number N is $O\left(N^{0.3}\right)$ under the deepest FAS V-cycle. A detailed evaluation of the GPU implementation is carried out employing the Roofline model and the scalability analysis.

1. Introduction

The graphics processing unit (GPU) is a highly parallel programmable processor featuring peak arithmetic functions and a much larger memory bandwidth than the central processing unit (CPU) [1]. Therefore, it is ideal for applications with high computation demand, such as computational fluid dynamics (CFD). During the last decade, GPUs have been widely used for CFD. Several researchers [2–7] have used the GPU to achieve order-of-magnitude performance gains over its CPU counterpart.

The classical numerical solutions of the incompressible flows encounter decoupling between both pressure and velocity. To advance the pressure field to a steady-state solution, numerical schemes involving solving a new equation such as a Poisson equation for the pressure correction may be used [8]. Currently, there exist several approaches to dealing with the linear system for the pressure Poisson equation in multi-GPU implementation, such as the point iterative method, i.e., the Jacobi [9] or Gauss-Seidel [10] method; Krylov subspace methods, e.g., the Conjugate Gradient (CG) method [3,11,12]; and Fast Fourier transform (FFT) for cases with a periodic boundary [6,13].

* Corresponding author.
E-mail addresses: xiaoleishi.th@gmail.com (X. Shi), tanmayagrawal7@gmail.com (T. Agrawal), calin@pme.nthu.edu.tw (C.-A. Lin), hwangf@math.ncu.edu.tw (F.-N. Hwang), nemovten608@gmail.com (T.-H. Chiu).

https://doi.org/10.1016/j.jcp.2020.109447
0021-9991/© 2020 Elsevier Inc. All rights reserved.

The pointwise iterative method is suitable for GPU computing, but an unacceptable low convergence rate may be experienced. For the CG method, evaluating the matrix-vector products (SpMV) is necessary, and requires all-to-all communication and low GPU utilization (less than 10% for the SpMV operations) due to the memory-bounded operations with unstructured memory access [11]. Additional acceleration, such as a multigrid method can be adopted. For example, Senocak et al. [9] developed the geometric multigrid solver for the pressure Poisson equation, in which the weighted Jacobi or red-black Gauss-Seidel method is used as a smoother. For cases with a periodic boundary condition [6], one can solve the Poisson equation using FFT, which usually produces a higher performance, but the periodic domain constraint limits the broader application.

An alternative approach to avoid solving the pressure Poisson equation is to introduce a pseudo-time pressure-derivative term to the continuity equation. This method, formally called the artificial compressibility method (ACM), was first proposed by Chorin [14] and has been widely used for the simulation of incompressible flows [15]. Rogers et al. [16] proposed an unsteady ACM formulation, such that for every physical time-step, the divergence free can be ensured by performing iterations in pseudo time. Based on this concept, Louda et al. [17] formulated explicit numerical schemes for the incompressible flow calculation, which merely requires the information of the nearest neighbors such that the locality is especially suitable for parallel computing.

However, the numerical scheme mentioned above usually shows a good convergence rate in the first few iterations, generating poorer convergence rates subsequently. The multigrid method has proven itself as a powerful tool to address this problem [18–21]. The plethora of numerical schemes for incompressible flows has been accelerated using the multigrid method [22–24]. Moreover, the multigrid method has been proven efficient for both computing and memory usage, and the additional memory requirement is merely $O(\log N)$, where N is the total number of points in the finest grid [21]. Drikakis et al. [25] developed one of the earliest explicit artificial compressibility-based solvers with a multigrid accelerator, albeit for steady-state flows. A full multigrid full approximation scheme (FMG-FAS) scheme was adopted and up to 32 times speedups, as compared to the single-grid counterpart were reported.

To the authors' knowledge, the implementation of a parallel nonlinear multigrid with the ACM-type smoother on multi-GPU for unsteady flow simulation is not available. Thus, this study primarily aims to develop an explicit scheme for unsteady incompressible flows accelerated by the FAS V-cycle scheme, suitable for multiple GPUs. Subsequently, the study has the following objectives: first, to develop an explicit scheme within the framework of ACM; second, to develop optimized algorithms for the discretized system and to perform a multigrid process on GPUs; and thereafter, to investigate the performance of the multi-GPU implementation. Furthermore, the convergence property of the FAS V-cycle scheme is addressed. Lastly, lid-driven cavity flow problems are solved to demonstrate the efficiency and accuracy of this multi-GPU solver.

The remainder of this paper is organized as follows: In section 2, the discretization of unsteady artificial compressibility equations and the methodology of the FAS V-cycle scheme are presented. In section 3, details of the multi-GPU implementation are presented. In section 4, numerical results of the steady and transitioning 3D lid-driven cavity flows are presented for validation. The performances of the FAS V-cycle scheme and the multi-GPU implementation are shown in section 5. Section 6 provides a conclusion for the study.

2. Methodology

2.1. Governing equations and their discretization

In this section, we describe the numerical method for solving the governing equations. Consider incompressible Navier-Stokes equations for unsteady flows given by

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_i} - \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} = 0 \tag{2}$$

where u, p and ν are the velocity, pressure and kinematic viscosity, respectively. The continuity and momentum equations (Eqns. (1) and (2)) are discretized on a staggered grid using the finite-volume approach, and spatial derivatives for the pressure and velocity components are approximated using a second-order central difference scheme. The discrete equations are integrated in time using an implicit second-order time accurate scheme, i.e.,

$$\frac{\partial u_i^{n+1}}{\partial x_i} = 0 \tag{3}$$

$$\frac{3u_i^{n+1} - 4u_i^n + u_i^{n-1}}{2\Delta t} + \frac{\partial}{\partial x_j}(u_i^{n+1}u_j^{n+1}) + \frac{1}{\rho}\frac{\partial p^{n+1}}{\partial x_i} - \nu \frac{\partial^2 u_i^{n+1}}{\partial x_j \partial x_j} = 0 \tag{4}$$

where Δt is the physical time step, and the superscripts, $n+1$, n, and $n-1$ represent different physical time levels. To facilitate the efficient usage of GPU, here the velocity and pressure coupling is performed using the artificial compressibility method [16,14,15] by introducing pseudo-time terms as described in [15,26]. To construct an explicit scheme, the Euler forward difference method is adopted to integrate the equations in pseudo time. The discretized equations are expressed as

$$\frac{u_i^{l+1}-u_i^l}{\Delta\tau}+\frac{3u_i^{l+1}-4u_i^n+u_i^{n-1}}{2\Delta t}+\frac{\partial}{\partial x_j}(u_i^l u_j^l)+\frac{1}{\rho}\frac{\partial p^l}{\partial x_i}-\nu\frac{\partial^2 u_i^l}{\partial x_j\partial x_j}=0 \tag{5}$$

$$\frac{1}{\beta}\frac{p^{l+1}-p^l}{\Delta\tau}+\frac{\partial u_i^{l+1}}{\partial x_i}=0 \tag{6}$$

where $\Delta\tau$ and l are the pseudo-time step and pseudo time level. β is an numerical parameter in the artificial compressibility scheme. The velocity u_i and pressure p at the physical time step $n+1$ can be obtained through pseudo-time stepping via Eqns. (6) and (5) until the solution reaches a steady state, i.e., $p^{l+1}\approx p^l=p^{n+1}$ and $u_i^{l+1}\approx u_i^l=u_i^{n+1}$ as l is sufficiently large. In this case, the divergence-free condition will be satisfied at each physical time step. The stopping criterion of each pseudo time iteration is the satisfaction of continuity, $\sum|\nabla\cdot\mathbf{u}|\le 10^{-8}$. A typical flowchart of this numerical scheme is demonstrated in Fig. 1. The time steps $\Delta\tau$ and Δt are both constrained by the standard CFL condition [27], whose definition is given by $\Delta t, \Delta\tau \le \text{CFL}\times\min\left(1/\sum|u_i|/\Delta x_i\right)$. One advantage of Eqns. (5) and (6) is that this formulation is in an explicit form. Therefore, no linear equation solver is needed and it is natural to implement on a GPU, and especially on multiple GPUs [28]. Also, we will show in the numerical results section that Eqs. (5) and (6) implemented on multiple GPUs can serve as an efficient smoother for our proposed nonlinear multigrid.

2.2. Multigrid methodology

We first introduce some notations before describing the full approximation scheme for solving incompressible Navier-Stokes equations. Q^k represents the flow variables u_1, u_2, u_3, and p at different grid levels. k is the multigrid level number and $k=0$ is the finest grid. The pseudo-time step $\Delta\tau$ at different grid levels is adjusted based on the CFL condition. The restriction and prolongation operators that send the data to grids from the k-th level to the $(k+1)$-th level and vice versa are denoted by

$$\hat{Q}^{k+1}=I_k^{k+1}Q^k, \qquad \tilde{Q}^k=I_{k+1}^k Q^{k+1}, \tag{7}$$

respectively. $L^k(\cdot)$ and $L_0^k(\cdot)$ are the discrete operators corresponding to Eqns. (5) and (6) and Eqns. (3) and (4), respectively. The residuals of the continuity and momentum equation at grid level k are computed as,

$$R_{cont}^k=\left(\frac{\partial u_i^{l+1}}{\partial x_i}\right)^k \tag{8}$$

$$R_{i-mom}^k=\left(\frac{3u_i^{l+1}-4u_i^n+u_i^{n-1}}{2\Delta t}+\frac{\partial}{\partial x_j}(u_i^{l+1}u_j^{l+1})+\frac{1}{\rho}\frac{\partial p^{l+1}}{\partial x_i}-\nu\frac{\partial^2 u_i^{l+1}}{\partial x_j\partial x_j}\right)^k \tag{9}$$

It should be noted that the residuals are not zero, when $p^{l+1}\neq p^l$ and hence $u_i^{l+1}\neq u_i^l$, as shown in Eqns. (5) and (6).

Then the coarse-grid version of governing equations can be formulated by treating the restricted approximate solutions and residuals as a source term as shown in the following form:

$$L^{k+1}\left(Q^{k+1}\right)=\underbrace{L_0^{k+1}\left(\hat{Q}^{k+1}\right)-\hat{R}^{k+1}}_{RHS^{k+1}} \tag{10}$$

Thus, the right-hand side of Eqn. (10) can be further expressed as:

$$RHS_{cont}^{k+1}=\left(\frac{\partial \hat{u}_i^{l+1}}{\partial x_i}\right)^{k+1}-\hat{R}_{cont}^{k+1} \tag{11}$$

$$RHS_{i-mom}^{k+1}=\left(\frac{3\hat{u}_i^{l+1}-4\hat{u}_i^n+\hat{u}_i^{n-1}}{2\Delta t}+\frac{\partial}{\partial x_j}(\hat{u}_i^{l+1}\hat{u}_j^{l+1})+\frac{1}{\rho}\frac{\partial \hat{p}^{l+1}}{\partial x_i}-\nu\frac{\partial^2 \hat{u}_i^{l+1}}{\partial x_j\partial x_j}\right)^{k+1} -\hat{R}_{i-mom}^{k+1} \tag{12}$$

Note that the pseudo-time derivatives should not be included while calculating the RHS term.

The FAS V-cycle scheme can be realized as:

Repeat 1-4 until the fine-grid solution reaches the divergence-free state.

```
1. Presmoothing
     Solve $L^k(Q^k) = 0$ for $Q^k$ by carrying out $\gamma_1$ iterations for the $l$-loop on the finest level $k = 0$.
2. Coarse-grid correction
   2a) Restrict the approximate solution $Q^k$ and the residual $R^k$ at the $k$-level to the next coarse level $k+1$.
         $\hat{Q}^{k+1} = I_k^{k+1} Q^k, \qquad \hat{R}^{k+1} = I_k^{k+1} R^k$
   2b) Compute the right-hand side $RHS^{k+1}$ (Eqns. (11)-(12)).
   2c) Solve the correction equations for $Q^{k+1}$ with $\gamma_{rs}$ iterations.
         $L^{k+1}\left(Q^{k+1}\right) = RHS^{k+1}$
   2d) $k = k+1$
   2e) If grid $k$ is the coarsest grid, go to 2f);
          else go to 2a) end
   2f) $k = k-1$
   2g) Compute the corrected approximate solution at the $k$-th level.
         $Q^k = Q^k + I_{k+1}^k\left(Q^{k+1} - \hat{Q}^{k+1}\right)$    (13)
   2h) Solve the correction equation for $Q^k$ with $\gamma_{pl}$ iterations.
         $L^k\left(Q^k\right) = RHS^k$    (14)
   2i) If the $k-1$-th level is the finest level,
         1. $k = k-1$
         2. $Q^k = Q^k + I_{k+1}^k\left(Q^{k+1} - \hat{Q}^{k+1}\right)$
         3. go to 3.;
       else go to 2f) end
3. Postsmoothing
     Solve $L^k(Q^k) = 0$ by carrying out $\gamma_1$ iterations for the $l$-loop on the finest level $k = 0$.
```

Though the numbers of iterations $\gamma_1, \gamma_{rs}, \gamma_{cg}, \gamma_{pl}$ in the above multigrid scheme play a critical role in determining its performance [18,21], it is quite challenging to find the optimal parameters through theoretical analysis for the numerical methods mentioned above. Therefore, in this work the number of iterations γ was determined by conducting various simulations.

2.3. *Inter-grid transfer operator*

Due to the staggered grid configuration and the need for mesh refinement in the near wall region, the restriction and prolongation operators proposed by [25] are modified for the three-dimensional nonuniform staggered grid. The coarse-grid is constructed by merging two neighboring fine-grid control volumes (CVs) in each direction (x, y, z); therefore, one coarse-grid CV consists of eight fine-grid CVs. In the staggered grid, the scalar, such as pressure p, is positioned at the center of the CV, while the vector, such as velocity, is located on the faces of the CV. Fig. 2 shows the three-dimensional sketch.

The restriction operator of the scalar is a volume-weighted average over all of the eight fine-grid CVs that compose the CV of the coarse-grid point, as shown in Fig. 3. For vector components, the restriction operator is an area-weighted average over the four faces of the fine-grid CV, which subsequently forms the interface of the coarse-grid CV, as shown in Fig. 3.

Similarly to the restriction operator, the prolongation operator varies according to storage locations, and we also consider the nonuniform grid distribution. For a scalar node, a trilinear interpolation formula taking the surrounding eight coarse-grid nodes as stencil proposed in [25] is adopted. Fig. 4(a) shows a two-dimensional (2D) sketch, and the extension to three-dimensional implementation is straightforward. For the vector component, take u velocity correction ϕ_u^k as an example (shown in Fig. 4(b)). First, use the bilinear interpolation formula to obtain ϕ_u^k, shown in green, on surfaces where coarse and fine grids overlap (fine-grid face with the even number index as shown in Fig. 4(b)). Then the value on the fine-grid surface, shown in red, which is located inside the coarse grid (fine-grid face with the odd number index) is calculated using linear interpolation taking two neighboring ϕ_u^k in the x direction as stencils. Similarly, ϕ_v^k and ϕ_w^k can be obtained in the y and z directions respectively.

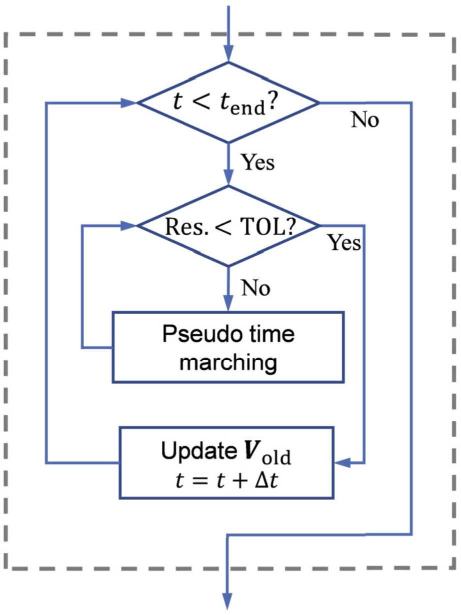


Fig. 1. Flowchart of the numerical implementation.

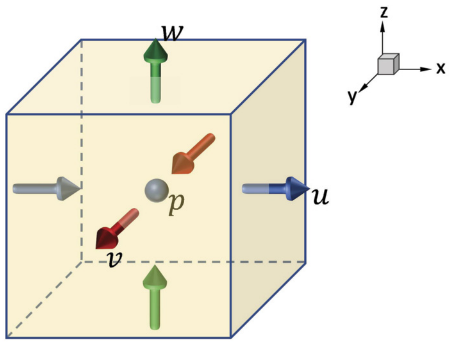


Fig. 2. Location of pressure and velocities of a 3D simulation control volume.

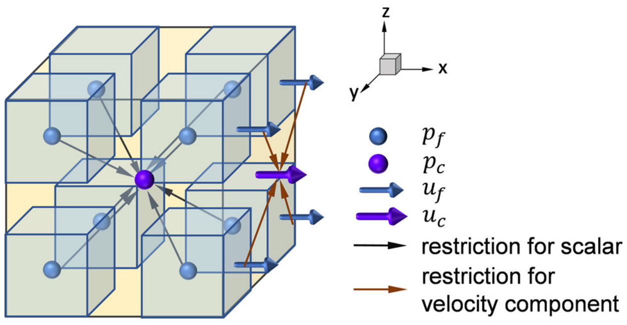


Fig. 3. Schematic of the restriction operation for staggered grid.

3. Multi-GPU implementation

In this section we explain the GPU implementation in detail. First, we present the domain decomposition and the corresponding communication pattern. Then the overlapping strategies for three primary function blocks: solving Eqns. (5) and (6), denoted by `SolveNSEqn`; `restriction` (Restriction); and `interpolation` (Prolongation) in the multigrid process are shown in detail.

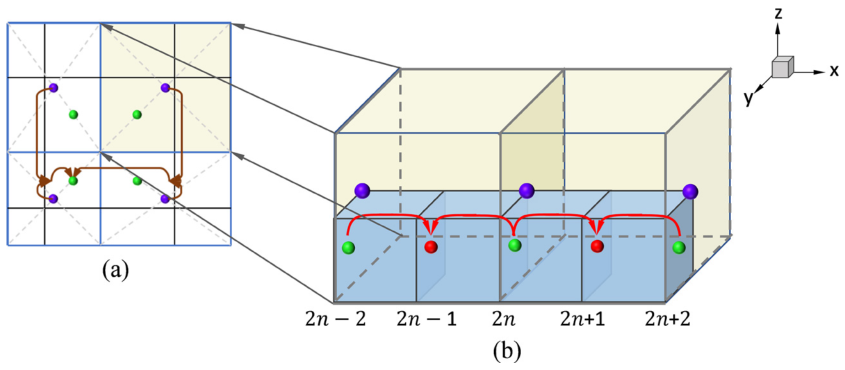


Fig. 4. Schematic of the prolongation operator. (a) The bilinear interpolation in 2D case on nonuniform grid: ●: coarse-grid node; ●: fine-grid node. (b) Prolongation operator with respective to staggered grid (x-direction): ●: calculated by bilinear interpolation from coarse grid as shown in (a); ●: obtained by linear interpolation from the available two neighboring fine-grid values that are calculated first. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

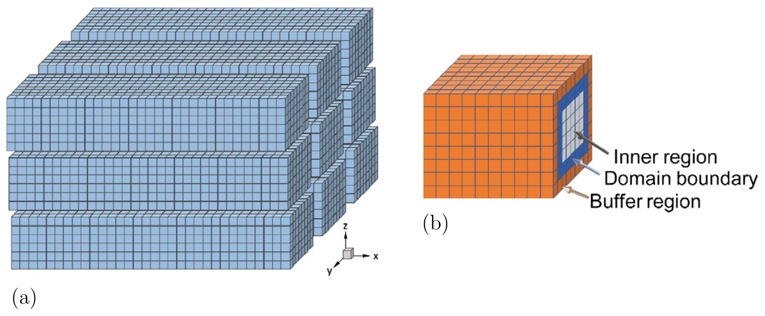


Fig. 5. (a) Partition of the computational domain into 2D sub-domains to feed on GPUs. (b) Diagram of buffer region and domain boundary region within each sub-domain.

Today's CUDA-enabled GPU contains thousands of processor cores; for example, the Nvidia Tesla V100 has 2560 FP64 CUDA cores [29], which allows for tens of thousands of threads running concurrently and yields incredible computing power. In the present work, each MPI rank was mapped to a GPU in a 1:1 manner. GPUs execute the whole computation, and CPUs are responsible only for controlling the program flow of the GPUs, file I/O, and data communication. The present code is written using C++ and CUDA C/C++.

3.1. Data allocation

In the current implementation, each of the flow variables is organized in a separate structure of array (SoA) in the GPU memory and each structure of array consists the pointers of the corresponding flow variable of each multigrid layer.

3.2. Domain decomposition and communication

The current GPU implementation adopts a two-dimensional ("pencil") domain decomposition along the y and z directions, as shown in Fig. 5(a). The flow variables allocated in the GPU memory are slightly larger to accommodate the buffer zone for data communication between neighboring GPUs, as shown in Fig. 5(b). For a sub-domain containing a physical boundary, the boundary condition is applied to the boundary-overlapped buffer region directly. We rely on the CUDA-aware Message Passing Interface (MPI) for communication in the distributed memory system so that explicit data transfer between the host (CPU) and device (GPU) in the communication stage is avoided [30], which yields more compact code and higher efficiency. In two-dimensional domain decomposition, one GPU needs to transfer data to its four neighbors, and Fig. 6 shows the detailed layout.

3.3. Overlapping strategy

To execute the FAS V-cycle scheme on multi-GPU, the present code consists of three major function blocks: `SolveNSEqn`, `Restriction` and `Interpolation`. One can apply any number of grid levels by combining these three functions, and to ensure its efficiency and scalability, an overlapping strategy is implemented for each function block. A typical code structure using two multigrid levels is demonstrated here.

Code structure: Implementation of two multigrid levels

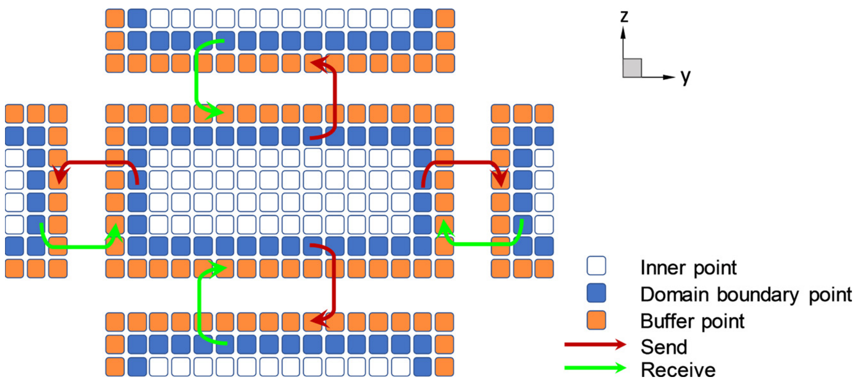


Fig. 6. Data communication pattern corresponding to 2D domain decomposition.

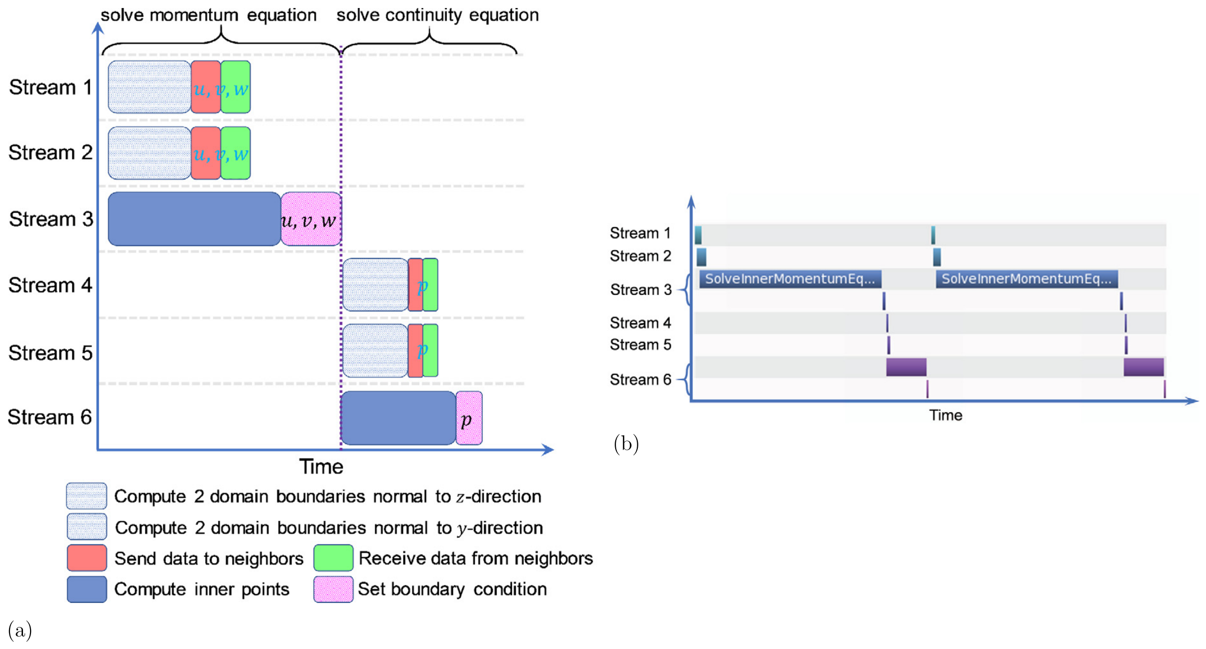


Fig. 7. (a) Streams schedule of `SolveNSEqn` for overlapping the computation and communication during solving Eqns. (2) and (1). The dotted purple line represents that operations after it must wait for the previous operations to complete. (b) Nvidia Visual Profiler (`nvvp`) output of `SolveNSEqn` for a parallel run on 2 GeForce GTX TITAN Z GPUs using a resolution of 128^3, communication process is not profiled.

```
• for <Convergence criteria> do
    – Call SolveNSEqn on grid level 1 for $\gamma_1$ times;
    – Call Restriction to restrict variables from grid level 1 to level 2;
    – Call SolveNSEqn on grid level 2 for $\gamma_{cg}$ times;
    – Call Interpolation to interpolate correction from level 2 back to level 1;
    – Call SolveNSEqn on grid level 1 for $\gamma_1$ times;
    – Calculate continuity error on grid level 1;
• end do
```

For the overlapping strategy [2], it is apparent that operations without dependencies can be executed concurrently. Therefore once they are scheduled in different CUDA streams with non-blocking MPI communication APIs, the communication process may be fully overlapped with the computation of the inner region, which yields higher efficiency and scalability [2,4]. Thus, in this work, we develop multi-stream overlapping strategies to optimize the afore-mentioned three function blocks to maximize code efficiency, and their corresponding overlapping sketches are depicted in Figs. 7 to 9 respectively.

For the `SolveNSEqn` function block, a brief explanation is given below. The computations of the boundary region and inner region are scheduled in different CUDA streams as shown in Figs. 7(a) and 7(b), where the data communication is

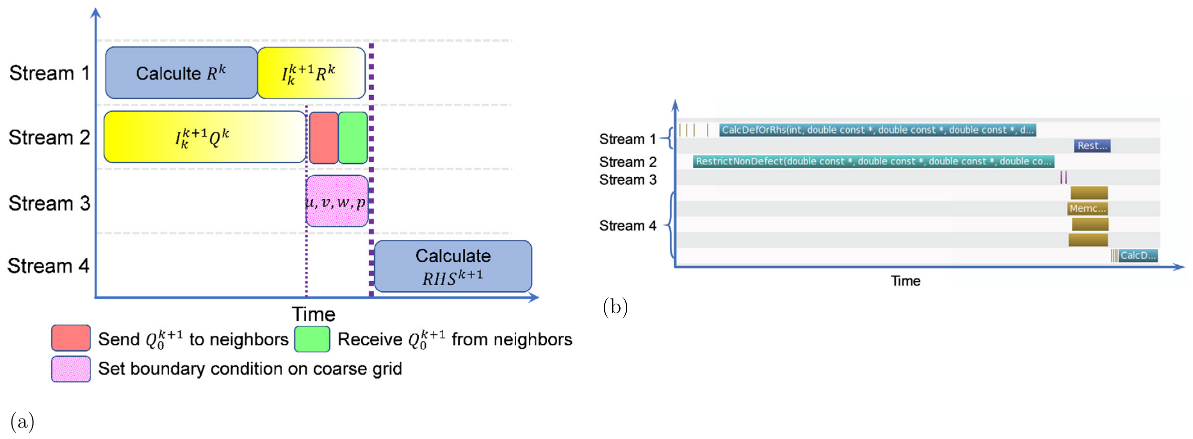

Stream 1
Calculte R^k
$I_k^{k+1}R^k$
Stream 2
$I_k^{k+1}Q^k$
Stream 3
u,v,w,p
Stream 4
Calculate RHS^{k+1}
Time
🟥 Send Q_0^{k+1} to neighbors
🟩 Receive Q_0^{k+1} from neighbors
▨ Set boundary condition on coarse grid
(a)
Stream 1
CalcDefOrRhs(int, double const *, double const *, double const *, d...
Rest...
Stream 2
RestrictNonDefect(double const *, double const *, double const *, double co...
Stream 3
Stream 4
Memc...
CalcD...
Time
(b)

Fig. 8. (a) Streams schedule of `Restriction` during the restriction stage of FAS V-cycle scheme. The dotted purple line has same meaning in Fig. 7. (b) Nvidia Visual Profiler (`nvvp`) output of `Restriction` for a parallel run on 2 GeForce GTX TITAN Z GPUs using a resolution of 128^3, communication process is not profiled.

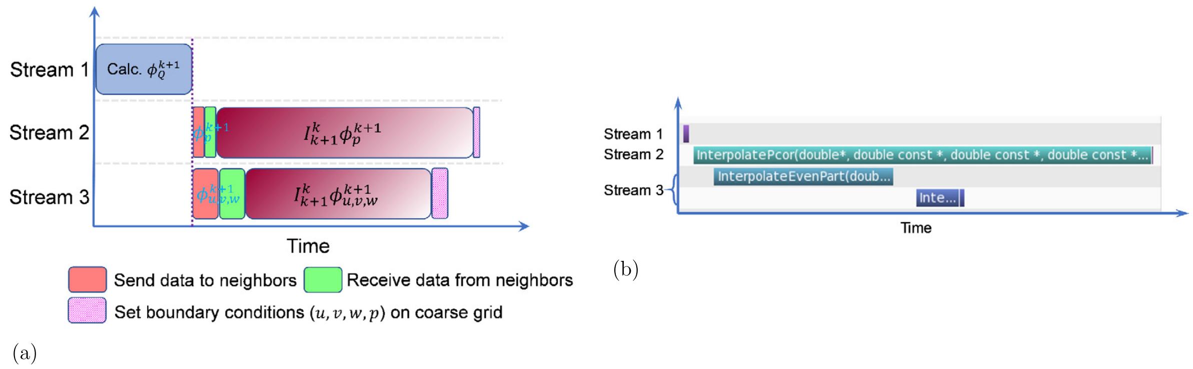

Stream 1
Calc. ϕ_Q^{k+1}
Stream 2
ϕ_p^{k+1}
$I_{k+1}^k\phi_p^{k+1}$
Stream 3
$\phi_{u,v,w}^{k+1}$
$I_{k+1}^k\phi_{u,v,w}^{k+1}$
Time
🟥 Send data to neighbors
🟩 Receive data from neighbors
▨ Set boundary conditions (u,v,w,p) on coarse grid
(a)
Stream 1
Stream 2
InterpolatePcor(double*, double const *, double const *, double const *...
InterpolateEvenPart(doub...
Stream 3
Inte...
Time
(b)

Fig. 9. (a) Streams schedule of `Interpolation` during the prolongation stage of FAS V-cycle scheme. The dotted purple line has the same meaning in Fig. 7. (b) Nvidia Visual Profiler (`nvvp`) output of `Interpolation` for a parallel run on 2 GeForce GTX TITAN Z GPUs using a resolution of 128^3, communication process is not profiled.

executed right after the boundary computation stream. Since the size of the boundary region is much smaller than that of the inner region, the non-blocking communication latency can be partly or fully hidden by the computation.

Usually, the overlapping strategy introduces a need for extra coding work because one may need to duplicate computational kernels for both the domain boundary and the Inner region, which greatly increases the complexity of multi-GPU code. Effective implementation of the overlapping strategy is listed in Fig. 10 based on the function template provided in C++. In this model, only two extra range-limitation kernels need to be introduced. By passing the corresponding range-limitation kernel to `ComputationalKernel`, a single computational kernel can be used for different regions without modification, which improves code reusability and simplifies the optimization procedure.

4. Numerical results

4.1. Unsteady flow between parallel plates

The numerical procedure is first validated by using the unsteady flow within parallel plates separated by a distance h, where the bottom plate moves impulsively at the time beyond zero. Fig. 11 shows the comparisons of the predicted velocity profiles with exact solutions at different times ($t^* = \nu t/h^2$). At the time $t^* = 0.01$, the development of the viscous layer due to the sudden movement of the bottom plate can be observed. As time proceeds, the profiles approach the linear variation of the Couette flow. The present predictions compare favorably with the exact solutions showing the accuracy of the unsteady term discretization implementation.

4.2. Laminar lid-driven cavity flows

The lid-driven cavity flow is a problem widely used for the validation of numerical schemes. Fig. 12 shows a 3D cubic cavity driven by the top lid. Flow within the cavity is characterized by the formation of a central vortex in response to

```
// A range-limitation-kernel corresponding to "Domain boundary"
__device__ static inline
bool isHaloRegion( const int i, const int j, const int k,
  const int ist, const int ien, const int jst, const int jen,
  const int kst, const int ken )
{
  return (i < ist || i > ien || j < jst || j > jen ||
        k < kst || k > ken);
}

// A range-limitation-kernel corresponding to "Inner region"
__device__ static inline
bool isInnerRegion( const int i, const int j, const int k,
  const int ist, const int ien, const int jst, const int jen,
  const int kst, const int ken )
{
  return (i >= ist && i <= ien && j >= jst && j <= jen &&
        k >= kst && k <= ken);
}

__global__
void ComputationalKernel(
  // pointer of arrays
  real *A, real *B, ..., real *Z,
  // variables of domain size
  const int ist, const int ien, const int jst, const int jen,
  const int kst, const int ken,
  // pointer of range-limitation-kernel
  bool (*RegionFunc)( const int, const int, const int, const int,
    const int, const int, const int, const int, const int )
  )
{
  int ix = threadIdx.x + blockIdx.x * blockDim.x;
  int iy = threadIdx.y + blockIdx.y * blockDim.y;
  int iz = threadIdx.z + blockIdx.z * blockDim.z;

  if ( !RegionFunc(ix, iy, iz, ist, ien, jst, jen, kst, ken) )
     return;

  /*
  computation work
  */
}
```

Fig. 10. A code model for overlap strategy.

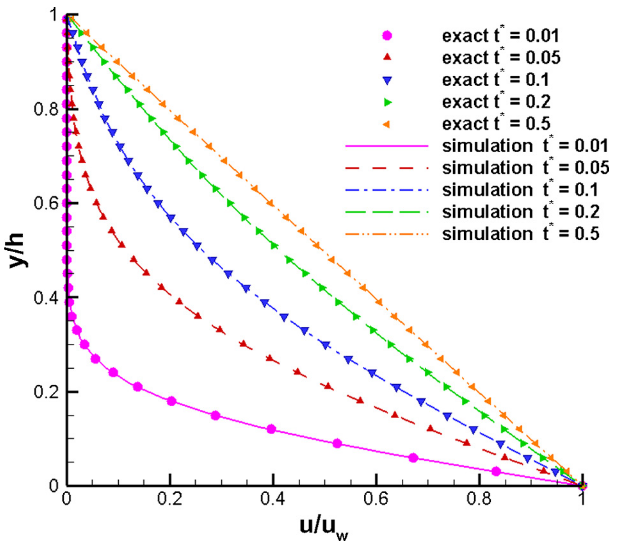


Fig. 11. Predicted velocity compared with exact solutions for unsteady flow between parallel plates ($t^* = \nu t/h^2$).

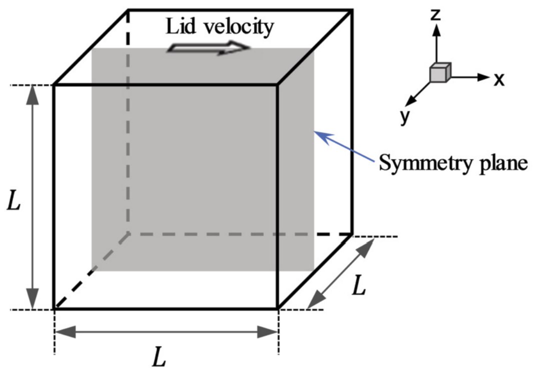


Fig. 12. Geometry setup for 3D lid-driven cavity flow. $x, y, z: -0.5 \sim 0.5$.

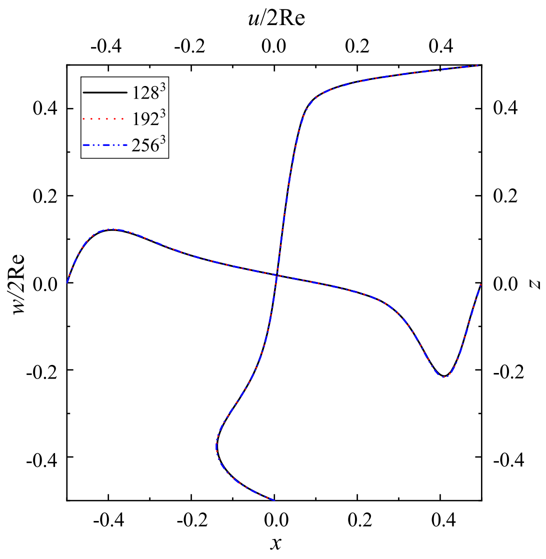


Fig. 13. Grid convergence test at Re=1000.

the moving lid. The flow for Re $<$ 1000 is in the laminar flow regime such that a steady state exists. Fig. 13 shows the grid convergence test at Re=1000, where the three grids, namely 128^3, 192^3 and 256^3, generate the same results. Thus, in subsequent laminar computations, the grid density 128^3 has been adopted.

Ku et al. [31] conducted numerical simulations for the 3D lid-driven cavity flow at Re=100, 400, and 1000; these have been used as basis for comparison. Fig. 14 presents the velocity profiles along the central lines $(x, 0.5, 0.5)$ and $(0.5, 0.5, z)$ in the symmetry plane, where the boundary layer becomes thinner as the Reynolds number increases. The present predictions agree well with the results of Ku et al. [31]. Fig. 15 shows the global vortical structure of Re=1000, where the central vortex and the corner vortices are evident.

4.3. Turbulent lid-driven cavity flows

The capability of the present procedure to predict turbulent flow has been further explored. Prasad and Koseff [32–34] have shown that turbulent flow prevails in the cubic cavity at Re=3200, where experimental measurements are available for comparison. For this Reynolds number (Re=3200), the grid density adopted is 192^3. To resolve the wall turbulence structure, the nonuniform grid is adopted in x, y, and z directions. The grid is clustered toward the wall using the hyperbolic tangent function, i.e.,

$$x_i = \frac{L}{a}\tanh\left[\frac{\xi_i}{2}\ln\frac{1+a}{1-a}\right] \tag{15}$$

where, L is the cavity length; $\xi_i = -1 + 2j/N$, N is the number of grids in the corresponding direction; and $x_i (i = 1, 2, 3)$ represents x, y, and z respectively. Here, a has been taken as 0.8, which results in the maximum aspect ratio being 2.74, as is shown in Fig. 16.

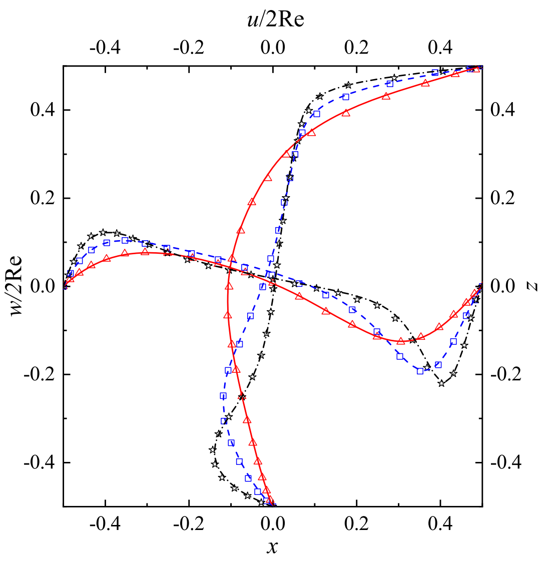


Fig. 14. Normal velocities on the centerlines $(x, 0, 0)$ and $(0, 0, z)$ in the symmetry plane using a resolution of 128^3 uniform grids. The Reynolds number are Re=100 (solid line), Re=400 (dash line), Re=1000 (dash-dot line). The symbols △ (Re=100), □ (Re=400), ☆ (Re=1000) are the results of Ku et al. [31].

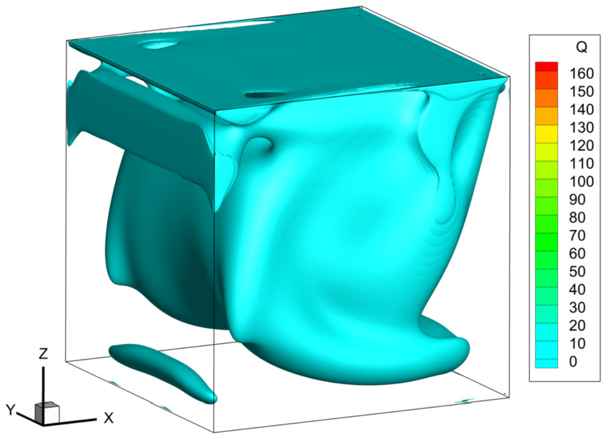


Fig. 15. Vortical structure of 3D lid-driven cavity flow of Re=1000.

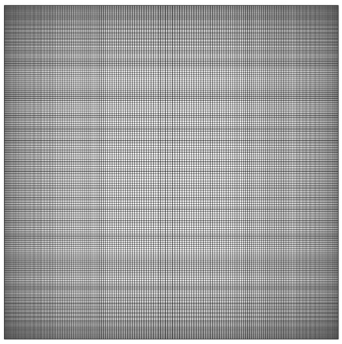


Fig. 16. Nonuniform grid configuration in x-y plane.

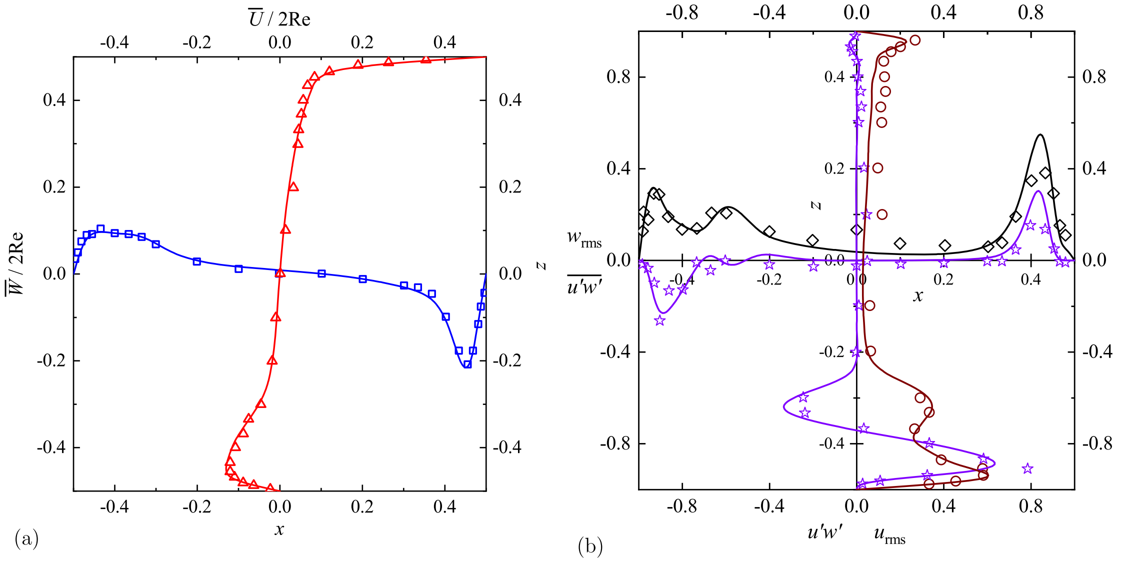


Fig. 17. (a) Mean and (b) turbulence statistics profiles along the horizontal and vertical centerlines in the symmetry plane. Red: U component of mean velocity; Blue: W component of mean velocity; Black: w_{rms}; Brown: u_{rms}; Violet: $\overline{u'w'}$. Lines are present results while symbols are experimental data of Prasad and Koseff [34].

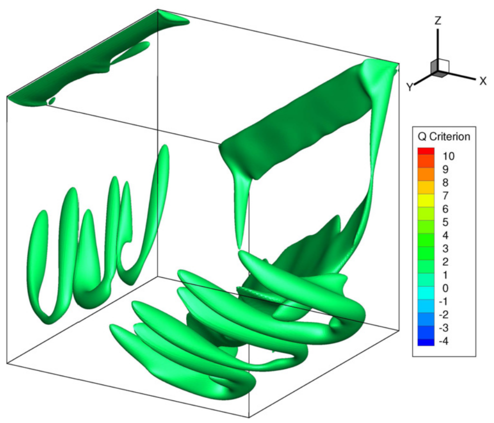


Fig. 18. Instantaneous vortical structure of 3D lid-driven cavity flow of Re=3200.

The simulation has been conducted on four Tesla P100 GPUs (NVLink). Each typical time step takes 25~30 FAS level 4 iterations whose wall time equals 0.68s (CFL=0.36). The simulation is stopped until a statistically steady state is reached after $2000L/u_{lid}$ and is identified by a constant mean shear stress profile. For this grid, we have verified that the cell sizes are of the order of the Kolmogorov length scale (η) given by $\eta = (\nu^3/\epsilon)^{1/4}$, where ϵ is the rate of turbulent kinetic energy dissipation. At the wall, ϵ has been estimated based on the viscous diffusion term in the turbulent kinetic energy equation [35,36]. However, away from the wall, ϵ has been obtained by assuming the turbulent kinetic energy production to be equal to the dissipation.

The comparison of the mean velocity and the Reynolds stress with the experimental results is presented in Fig. 17. The study results agree well with the experimental results of Prasad and Koseff [34]. The uniqueness of the 3D lid-driven cavity flow at Reynolds number 3200 compared with its Re=5000 and Re=10000 counterparts is in the existence of the Taylor-Görtler-Like (TGL) vortices. Additionally, there is a complicated interaction between the DSE and TGL vortices [33]. The presence of the TGL vortices accounts for the peak of $\overline{u'w'}$ and u_{rms} in the lower wall vicinity, as shown in Fig. 17. Furthermore, an instantaneous 3D vorticity has been presented in Fig. 18, where the hairpin-like vortices at the lower and upstream walls can be clearly observed.

5. Performance

The performances of the FAS V-cycle scheme and the current multiple GPU implementation are evaluated on a GPU cluster. The computing nodes contain an eight-core Intel Core^{TM} i7-6900K CPU and different models of Nvidia GPUs. We

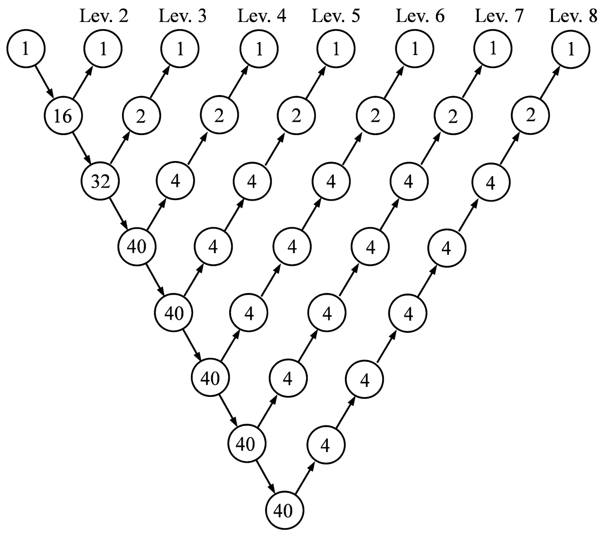


Fig. 19. FAS V-cycle showing iterations at various levels.

Table 1
The multigrid speedups versus Reynolds number (operating conditions as in Fig. 20).

	Re=500	Re=1000	Re=5000
Lev. 2	9.3	9.3	9.3
Lev. 3	38.0	38.0	38.0
Lev. 4	94.0	93.9	93.4

rely on Open MPI v2.1.5 and the CUDA 9.0 toolkit. The Nvidia `nvprof` and `nvvp` profiler provided in CUDA toolkit are used to obtain the performance metrics. All tests are run in double precision.

5.1. Performance of FAS V-cycle scheme

The performance of the explicit FAS V-cycle scheme is investigated based on the 3D lid-driven cavity flows at various Reynolds numbers and mesh sizes. The multigrid performance is presented based on work units (WU) [18], where one WU corresponds to the cost of performing one relaxation sweep on the finest grid in a multigrid system. The convergence history of the scheme is monitored based on the satisfaction of the continuity equation, i.e., the iteration stops when the level of divergence of the flow field reaches 10^{-9}. The configuration of the iteration number at each multigrid level is shown in Fig. 19.

Numerical experiments have been conducted on the unsteady 3D cavity flows. Table 1 shows that the performance of the present FAS V-cycle scheme is insensitive to the simulated Reynolds numbers.

Fig. 20 shows the convergence history of the FAS V-cycle scheme of the 10th time step for levels 2$\sim$ 7 using a resolution of 192^3. The results show that excellent convergence and computational performance are obtained with deep cycles and the speedup of FAS Lev. 7 is up to 250.

For the 3D lid-driven cavity flow simulations with a resolution of 96^3, the grid numbers for a successively coarser grids are $96^3 \to 48^3 \to 24^3 \to 12^3 \to 6^3 \to 3^3$, so the deepest FAS V-cycle is Lev. 6. Fig. 21 illustrates that the WU in the 10th physical time step reduces to $\frac{1}{3.3}$ in tandem with the increase of the multigrid level. Moreover, a larger grid number allows for deeper multigrid cycles, so a wider linear region can be seen in Fig. 21. The horizontal segment at the end means that if the grid number on the coarsest grid is too small, no further speedup can be obtained. Thus, we recommend that the grid number of the coarsest grid be greater than 5^3, which maintains an acceptable multi-GPU performance.

Fig. 22 shows the WU in the 10th physical time step versus grid number, and the WU of each FAS multigrid level is normalized by the WU of 48^3. A constant slope of 2/3 is observed at all multigrid levels for grid numbers greater than about 192^3. Thus, if the grid density is doubled, the increase in WU is $2^{2/3}$ within each physical time step. As already shown in Fig. 21, a larger grid number allows for deeper multigrid cycles, so the WU under the deepest FAS V-cycle of each resolution is examined, and the results are portrayed in Fig. 23, where the WU is normalized by the WU of 48^3. Fig. 23 shows a much smaller slope, 0.3 of the WU versus grid number, compared with that of Fig. 22, which means only $2^{0.3}$ times the WU increase if one doubles the grid number. Besides, Fig. 23 shows that the increasing problem size does improve the speedup of the FAS V-cycle scheme under the deepest V-cycle of each resolution, and the constant slope 0.35 means a $2^{0.35}$ times speedup is obtained when doubling the grid number on the finest grid.

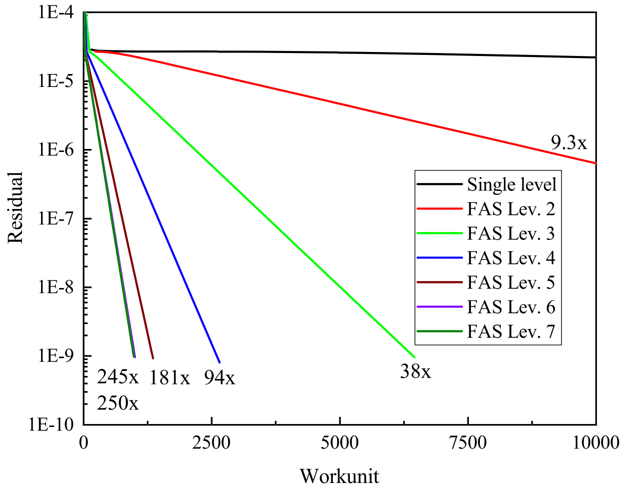


Fig. 20. Convergence history of the 10th time-step of unsteady cubic lid-driven cavity (Re=1000) using a resolution of 192^3, CFL number of physical and pseudo time are 0.6 and 0.4 respectively. Initial field of u, v, w, p were set to be zero.

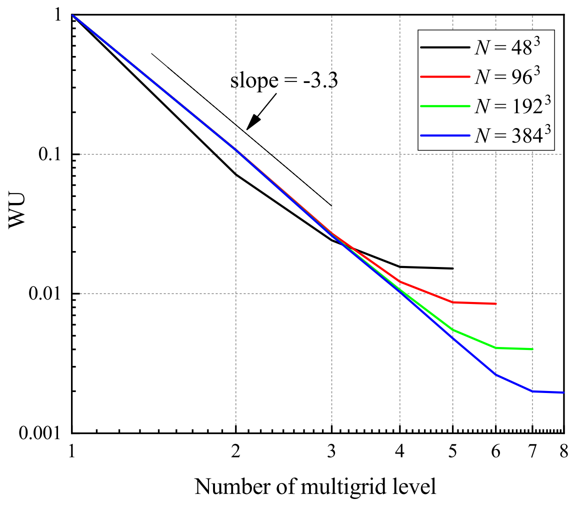


Fig. 21. Required workunits for a single time step versus multigrid level. (operating conditions as in Fig. 20).

5.2. Single node performance

We have reported the performance of the GPU kernels utilizing the Roofline model [37,38], an instructive tool for architectural analysis and program analysis, on a single Nvidia Tesla P100 GPU. The Roofline model combines arithmetic intensity, memory performance and floating-point performance into a 2D graph using bound and bottleneck analysis where the attainable performance in GFLOPs/s is a function of arithmetic intensity in floating point operations (FLOPs) per byte as shown in Eqn. (16).

$$\texttt{Attainable Performance(GFLOPs/s)} \;=\min\begin{cases}\texttt{Peak Performance}\\ \texttt{Peak Bandwidth}\times\texttt{Arithmetic Intensity}\end{cases} \tag{16}$$

Thus, the model defines a region with attainable performance. A traditional way to construct the Roofline model requires one to manually analyze each application kernel to collect metrics such as total FLOPs and bytes transferred, so it is tedious and error-prone. The Empirical Roofline Toolkit (ERT) [39] has been adopted in this work to build the empirical performance region. Furthermore, the error-correcting code memory (ECC) is enabled.

The memory bandwidth (DRAM) of the Tesla P100 GPU, as measured by ERT, is 504 GB/s (69% of the theoretical value) and its double-precision peak performance is 4574 GFOLPs/s (86% of theory). Adopting Eqn. (16), the theoretical and empirical performance regions are plotted in Fig. 24; the black dotted ceilings correspond to the measurements based on the

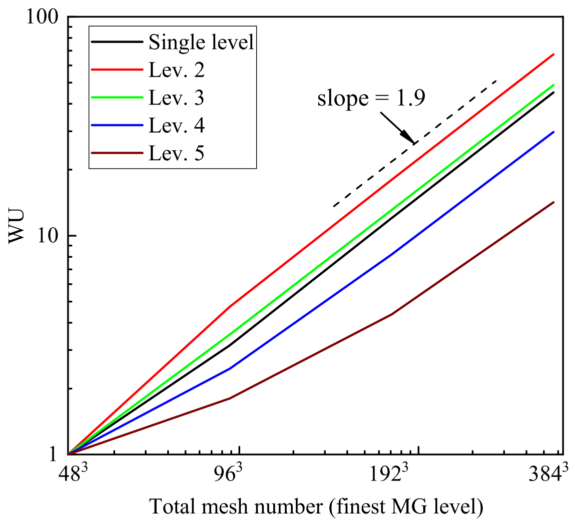


Fig. 22. Required workunits for a single time step versus the total mesh number on the finest MG level. (operating conditions as in Fig. 20).

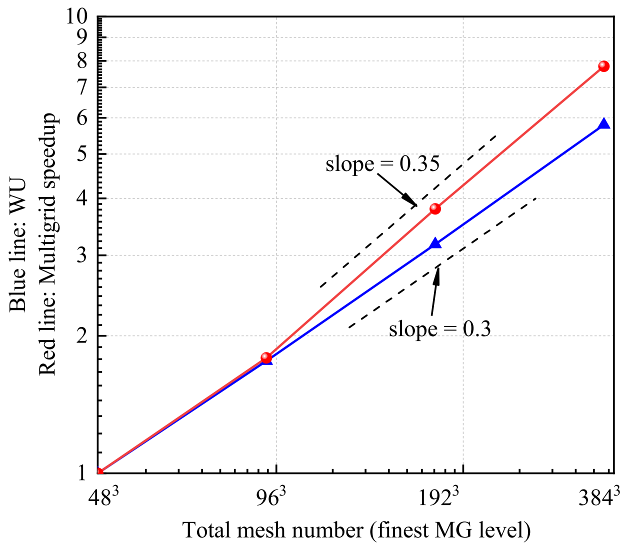


Fig. 23. Required workunit and multigrid performance versus the total mesh number on the finest MG level under the maximum multigrid level. (operating conditions as in Fig. 20).

ERT and the references for 10%, 25%, and 50% of the nominal peak performance and have been indicated by blue lines. We observed that the performance bound increases with the arithmetic intensity, where the arithmetic (operational) intensity is defined as a FLOP per byte. This is a measure of the floating-point operations (FLOPs) performed by a code relative to the amount of memory accesses (Bytes) to support the operations. However, when the arithmetic intensity reaches $4574/504 = 9.08$, denoted as $I_{\max}$, the performance bound saturates. The region where arithmetic intensity is less than $I_{\max}$ is memory-bounded; otherwise, it is compute-bounded.

For the performance of every GPU kernel in the Roofline model, several metrics have been obtained including the floating point operations (FLOPs), device memory read throughput (D_r), device memory write throughput (D_w), and kernel execution duration (T). One can obtain all these for a specific kernel using the following nvprof command:

```
$ nvprof -kernels <kernel specifier>
        -analysis-metrics -o analysis.prof <app> <app args>
```

then export the output file `analysis.prof` into Nvidia Visual Profiler (`nvvp`). The advantage of this method is it is automatic and very little code modification is needed when testing kernel performance. Therefore, the kernel performance for the Roofline model is calculated by [40]:

$$\texttt{Arithmetic Intensity=FLOPs/[(}D_r\texttt{+}D_w\texttt{)]} \tag{17}$$

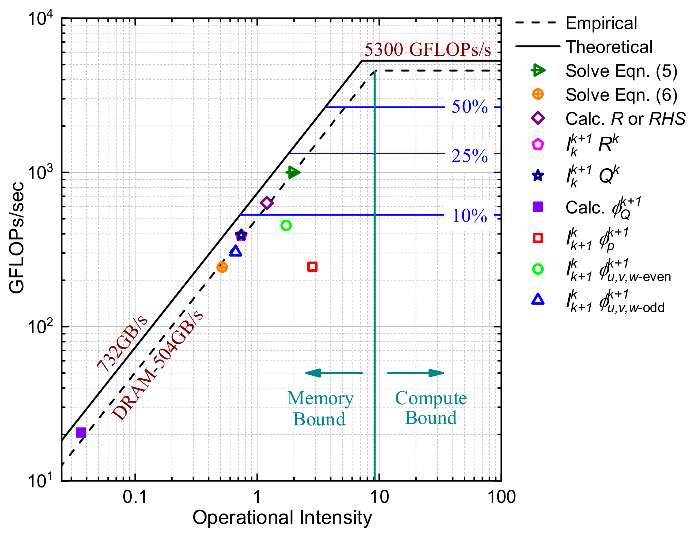


Fig. 24. Roofline for the Tesla P100 GPU with measured performances of each individual kernel.

Table 2
The collected metrics of each individual kernel.

kernel	unified cache hit rate (%)	L2 cache hit rate (%)	warp execution efficiency (%)
Solve Eqn. (5)	65.3	50.0	98.7
Solve Eqn. (6)	62.4	23.2	100.0
Calc. *R* or *RHS*	73.6	58.3	99.2
$I_k^{k+1} R^k$	57.6	11.5	99.6
$I_k^{k+1} Q^k$	66.2	13.3	99.3
Calc. ϕ_Q^{k+1}	28.8	37.3	90.0
$I_{k+1}^k \phi_p^{k+1}$	90.8	66.8	53.7
$I_{k+1}^k \phi_{u,v,w-\text{even}}^{k+1}$	78.2	49.2	58.2
$I_{k+1}^k \phi_{u,v,w-\text{odd}}^{k+1}$	59.9	41.2	84.0

$$\texttt{FLOPs/s=FLOPs}/T \qquad (18)$$

The performances of GPU kernels are depicted in Fig. 24 along with the Roofline, and Table 2 shows the corresponding cache hit rate and warp execution efficiency. It is not surprising to observe that all kernels are memory-bound, for this is typical for the memory-intensive kernels like stencils or SpMV in CFD. There are seven kernels located exactly on the empirical Roofline, which indicate the best performance one could obtain. We can imagine that their performance will directly take advantage of increasing memory bandwidth, so tuning for these kernels should focus on bandwidth improvement such as the use of cache bypass, array padding, shared memory, etc. However, for a real-world application like CFD, it is not straightforward to implement these optimizations by hand without auto-tuning. The increase in memory bandwidth of the Tesla V100, 900 GB/s, does aid this problem.

We noticed that there are two kernels, $I_{k+1}^k \phi_p^{k+1}$ and $I_{k+1}^k \phi_{u,v,w-\text{even}}^{k+1}$, that lie far from the empirical Roofline and show a premature compute-bound behavior, indicating the imperfect algorithm or code implementation. In Table 2, we find strikingly low warp execution efficiency of these two kernels, only about 50%. Fig. 25, which shows the mix of instructions executed by the $I_{k+1}^k \phi_p^{k+1}$ kernel, exhibits a corresponding high inactive instruction, and all indicate the heavy thread divergence. Thus, we deduce that the low attained performance of kernels $I_{k+1}^k \phi_p^{k+1}$ and $I_{k+1}^k \phi_{u,v,w-\text{even}}^{k+1}$ here are owed to the frequent call of the `if ... else ...` statement within these two kernels, which results in thread divergence. This is difficult to avoid since it is necessary when determining the index correspondence between the coarse and fine grid.

Upon further examination of the results shown in Table 2 and also evidenced in Fig. 24, we note that a kernel with low warp execution efficiency usually shows a premature compute-bounded behavior. Also, a short kernel body and simple algorithm, such as in Eqn. (6) and Calc. ϕ_Q^{k+1}, deliver a low L2 cache hit rate as well as low arithmetic intensity so performance is significantly reduced. The restriction operator kernels $I_k^{k+1} R^k$ and $I_k^{k+1} Q^k$ have sufficiently long kernel bodies but give the lowest L2 cache hit rate and low arithmetic intensity, which is caused by the algorithmic operation with low data reuse. For the kernels with arithmetic intensity greater than 1, shown in Fig. 24, both the unified and L2 cache hit rates are greater than 50%. These kernel bodies are sufficiently long, which provides an opportunity for data reuse. However, the register limitation has a significant impact on the number of resident warps on the multiprocessor.

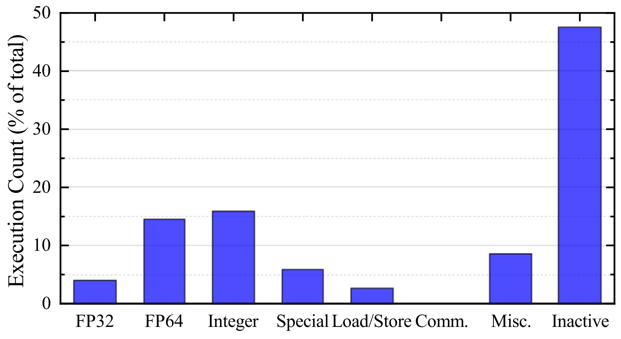


Fig. 25. Instruction execution counts of kernel $I_{k+1}^{k}\phi_{p}^{k+1}$ exported from `nvvp`. The instructions are grouped into classes and for each class the figure shows the percentage of thread execution cycles that are devoted to execution instructions in that class.

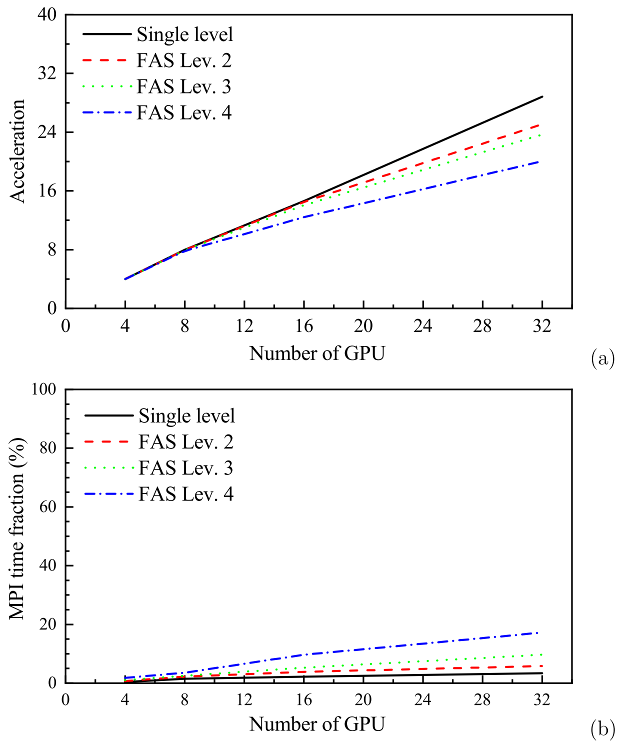


Fig. 26. Performance metrics of multi-GPU implementation on Tesla V100 GPU: (a) Strong scaling and (b) Fraction of time spent on MPI communication, each node consists 8 Nvidia Tesla V100 GPUs.

5.3. Multiple node performance

The strong scaling of the parallel performance along with the fraction of time spent on MPI communication as a function of the number of GPUs is shown in Fig. 26, where a resolution of 768^3 is used during the test. As expected, the single grid case shows satisfactory efficiency, which is attributed to the overlapping strategy. As more multigrid levels are adopted, we see a gradual efficiency drop when the number of GPUs is greater than eight. As well, a noticeably higher fraction of time in MPI communication is found in the FAS Lev. 4 test. The slope variation when GPU number equals eight is due to the lower bandwidth of Infiniband than that of NVLink, since each node contains eight Tesla V100 GPUs. Though most of the communication time can be overlapped with computation for the fine-grid level, this is not the case for the coarse-grid level, since the computation work on a coarse-grid is much less than that of the finest grid ($(1/8)^3$ for FAS level 4). Moreover, the iteration number γ_{cg} on the coarsest grid is one magnitude (40x here) greater than that of the finest grid, which means the latency due to insufficient overlapping is further amplified.

The breakdown of computational time is depicted in Fig. 27, where we find the additional work for multigrid implementation such as the restriction and prolongation operator takes a larger share of total computational time. Note that the relative magnitude of each kernel varies with a change in the multigrid configuration, such as the level of multigrid and the

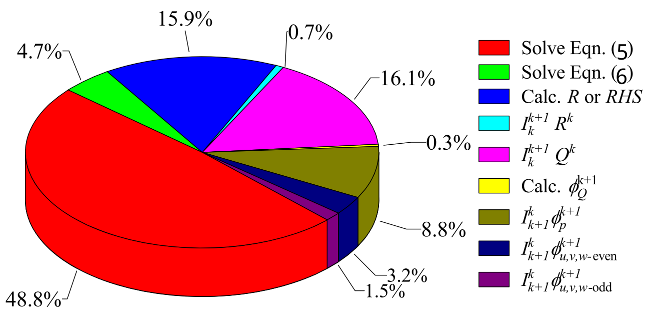


Fig. 27. Breakdown of computation time for a parallel run for FAS Lev. 2 multigrid using 2 Tesla P100 GPUs on a $256 \times 256 \times 256$ grid.

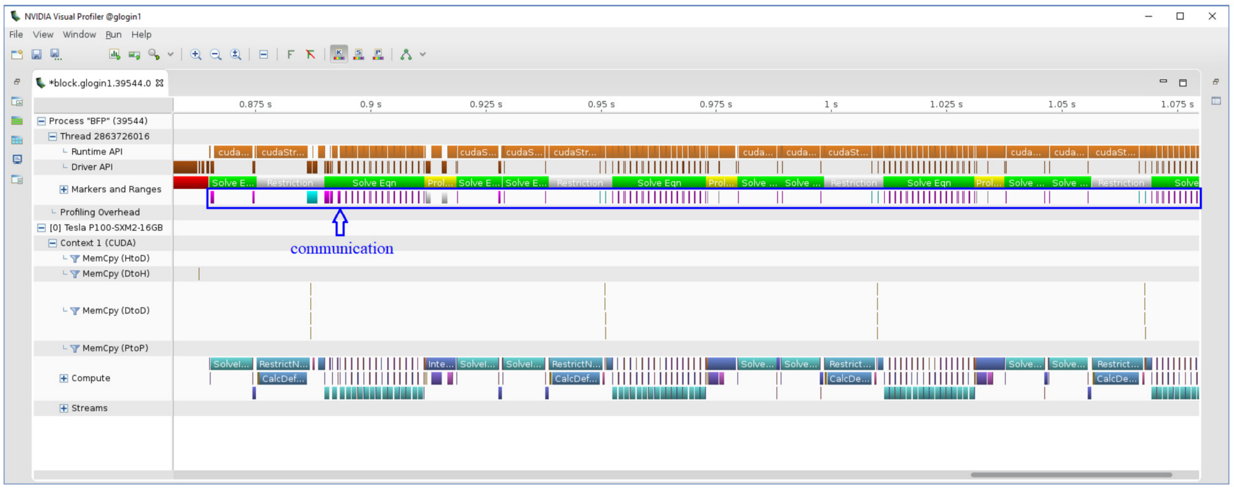


Fig. 28. Profiler output from `nvvp` (operating conditions as in Fig. 27).

iteration number of each layer. Due to the concurrent execution strategy illustrated in Figs. 8 and 9, we find that $I_k^{k+1} Q^k$ overlaps by “Calc. *R* or *RHS*” and both $I_{k+1}^k \phi_{u,v,w-\text{even}}^{k+1}$ and $I_{k+1}^k \phi_{u,v,w-\text{odd}}^{k+1}$ overlaps with $I_{k+1}^k \phi_p^{k+1}$. So, in real application, the combination of “Solve Eqns. (5) and (6)” costs ~72% of the total computational time. As more multigrid levels are adopted, this portion further increases and we observe a cost of ~80% for “Solve Eqns. (5) and (6)” at FAS Lev. 4.

Additionally, we generate the code execution trace through `nvvp` for further investigation of the current multi-GPU implementation. Fig. 28 shows an example of the profile for a parallel run of FAS Lev. 2 multigrid implementation on a 256^3 mesh, where the short but frequent communication workload on the coarse-grid level can be easily observed. In the “Compute” column, we find that the GPU is fully occupied by computational workload through the whole time-line as well as the concurrent kernel execution, which yields the high GPU occupancy of the present implementation.

5.4. Performance comparisons with DP-LUR scheme

Here, the relative performance of the present scheme compared with the DP-LUR scheme by Candler et al. [41] is presented. The performance comparison is based on the reported DP-LUR studies available [42,43]. For example, Tanno et al. [42] used DP-LUR to compute a steady 2-D lid-driven cavity flow at Re=100 with mesh density 512x512. In [42], Roe's approximate Riemann solver is used, where the convergence criterion was not documented. The elapsed time required when reaching a steady-state solution is documented, as shown in Table 3. The present numerical procedure is 3D, so a 3-D lid-driven cavity at the same Reynolds number is conduced, where the grid adopted is 128x128x128. Since this is a steady-state problem, simulation proceeds in pseudo-time with one multi-grid V cycle without physical time. In our simulations, the simulations stop when the mass residual (L_1) falls below 10^{-10}. A separate study by Cox et al. [43] is also included for comparison. In [43], the DP-LUR based high order compact flux reconstruction method on unstructured grids was used to compute NACA0012 airfoil at zero angle of attack at Re=1850. Since the flow is steady, so the physical time was also removed.

In the present study, the GPUs are much advanced than the GPUs in [42], and the numerical implementations are also different. Therefore, the comparisons are not straightforward. Also, when comparing 2-D and 3-D simulations of a steady lid-driven cavity, in general, 3-D flow lid-driven cavity flows require more iterations to reach a steady-state solution. As shown in Table 3, despite the 2D and 3D difference, the performance of the present simulations is encouraging. However,

Table 3
Elapsed time using DP-LUR and the multigrid method to respectively compute steady 2D NACA airfoil (Re=1850) and 2D and 3D lid-driven cavity flows (Re=100).

Scheme (machine)	geometry	Mesh no.	wall-clock time (s)	Mesh no./ Time (s)
DP-LUR (NVIDIA C1060) [42]	2D LDC	512^2	583.79	449
MG-present (NVIDIA K80)	3D LDC	128^3	24.74	84768
MG-present (NVIDIA V100)	3D LDC	128^3	7.036	298060
DP-LUR [43]	NACA0012	32456	1200	2.7

it will be beneficial if the DP-LUR scheme can be incorporated into the present numerical procedure to explore its further improvement.

6. Conclusion

A nonlinear multigrid solver for solutions of unsteady three-dimensional incompressible viscous flow working on multi-GPU cluster is developed. The solver consists of a full approximation scheme (FAS) V-cycle scheme to accelerate computation, where the artificial compressibility method-based Navier-Stokes solver is used as a smoother. The governing equations are discretized in space using the second-order central difference scheme on a non-uniform staggered grid. The pseudo and physical time integrations are performed using first-order forward Euler and second-order backward Euler scheme respectively. Simulations are conducted on the GPU cluster, where multi-stream overlapping strategies are designed to reduce latency across GPU nodes.

The numerical procedure is validated by computing 3D laminar and turbulent flows within a lid-driven cubic cavity. The predicted results compare favorably with previous benchmark solutions and measurements, both in mean and turbulent quantities. Regarding the performance of the FAS V-cycle scheme, up to two orders of magnitude speedups are reported, and the relationship between WU and total grid number N is $O\left(N^{0.3}\right)$ under the deepest FAS V-cycle. A detailed evaluation of the GPU implementation is carried out employing the Roofline model and the scalability analysis. Both the high-performance of a single GPU kernel and the linear scalability of the multi-GPU implementation are obtained, showing that the proposed explicit multigrid solver is suitable for GPU computing.

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.

Acknowledgements

The present study is supported by the Ministry of Science and Technology of Taiwan (Grant No. 105-2221-E-007-061-MY3). The authors are grateful to the National Center for High-performance Computing of Taiwan for computer time and facilities.

References

[1] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips, Gpu computing, Proc. IEEE 96 (2008) 879–899.
[2] Y. Wang, T. Aoki, Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster, Parallel Comput. 37 (2011) 521–535.
[3] D.D. Chandar, J. Sitaraman, D.J. Mavriplis, A GPU-based incompressible Navier—Stokes solver on moving overset grids, Int. J. Comput. Fluid Dyn. 27 (2013) 268–282.
[4] P.Y. Hong, L.M. Huang, L.S. Lin, C.A. Lin, Scalable multi-relaxation-time lattice Boltzmann simulations on multi-GPU cluster, Comput. Fluids 110 (2015) 1–8.
[5] B.M. Davani, F. Marti, B. Pourghassemi, F. Liu, A. Chandramowlishwaran, Unsteady Navier-Stokes computations on GPU architectures, in: 23rd AIAA Computational Fluid Dynamics Conferences, AIAA, 2017.
[6] X. Zhu, E. Phillips, V. Spandan, J. Donners, G. Ruetsch, J. Romero, R. Ostilla-Mónico, Y. Yang, D. Lohse, R. Verzicco, M. Fatica, R.J. Stevens, AFiD-GPU: a versatile Navier—Stokes solver for wall-bounded turbulent flows on GPU clusters, Comput. Phys. Commun. 229 (2018) 199–210.
[7] M.A. Diaz, M.A. Solovchuk, T.W. Sheu, High-performance multi-GPU solver for describing nonlinear acoustic waves in homogeneous thermoviscous media, Comput. Fluids 173 (2018) 195–205.
[8] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier–Stokes equations, J. Comput. Phys. 59 (1985) 308–323.
[9] D. Jacobsen, I. Senocak, A full-depth amalgamated parallel 3D geometric multigrid solver for GPU clusters, in: 49th AIAA Aerospace Sciences Meeting, AIAA, 2011.
[10] R. Deleon, D. Jacobsen, I. Senocak, Large-eddy simulations of turbulent incompressible flows on GPU clusters, Comput. Sci. Eng. 15 (2013) 26–33.
[11] G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, MPi-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner, Comput. Fluids 92 (2014) 244–252.
[12] P. Zaspel, M. Griebel, Solving incompressible two-phase flows on multi-GPU clusters, Comput. Fluids 80 (2013) 356–364.
[13] A. Gorobets, F. Trias, A. Oliva, A parallel MPI+OpenMP+OpenCL algorithm for hybrid supercomputations of incompressible flows, Comput. Fluids 88 (2013) 764–772.

[14] A.J. Chorin, A numerical method for solving incompressible viscous flow problems, J. Comput. Phys. 135 (1997) 118–125.
[15] A. Gilmanov, F. Sotiropoulos, A hybrid Cartesian/immersed boundary method for simulating flows with 3D, geometrically complex, moving bodies, J. Comput. Phys. 207 (2005) 457–492.
[16] S.E. Rogers, D. Kwak, C. Kiris, Steady and unsteady solutions of the incompressible Navier–Stokes equations, AIAA J. 29 (1991) 603–610.
[17] P. Louda, K. Kozel, J. Příhoda, Numerical solution of 2D and 3D viscous incompressible steady and unsteady flows using artificial compressibility method, Int. J. Numer. Methods Fluids 56 (2008) 1399–1407.
[18] A. Brandt, Multi-level adaptive solutions to boundary-value problems, Math. Comput. 31 (1977) 333–390.
[19] A. Brandt, Multilevel adaptive computations in fluid dynamics, AIAA J. 18 (1980) 1165–1172.
[20] A. Brandt, O. Livne, Multigrid techniques. 1984 guide with applications to fluid dynamics, in: revised ed., in: Classics in Applied Mathematics, vol. 67, SIAM, 2011.
[21] W. Briggs, V. Henson, S. McCormick, A Multigrid Tutorial, 2nd edition, SIAM, 2000.
[22] C. Liu, X. Zheng, C. Sung, Preconditioned multigrid methods for unsteady incompressible flows, J. Comput. Phys. 139 (1998) 35–57.
[23] M. Darwish, I. Sraj, F. Moukalled, A coupled finite volume solver for the solution of incompressible flows on unstructured grids, J. Comput. Phys. 228 (2009) 180–201.
[24] H.-W. Hsu, F.-N. Hwang, Z.-H. Wei, S.-H. Lai, C.-A. Lin, A parallel multilevel preconditioned iterative pressure Poisson solver for the large-eddy simulation of turbulent flow inside a duct, Comput. Fluids 45 (2011) 138–146.
[25] D. Drikakis, O. Iliev, D. Vassileva, A nonlinear multigrid method for the three-dimensional incompressible Navier—Stokes equations, J. Comput. Phys. 146 (1998) 301–321.
[26] W. Soh, J.W. Goodrich, Unsteady solution of incompressible Navier–Stokes equations, J. Comput. Phys. 79 (1988) 113–134.
[27] R. Courant, K. Friedrichs, H. Lewy, On the partial difference equations of mathematical physics, IBM J. Res. Dev. 11 (1967) 215–234.
[28] S. Pratap Vanka, A.F. Shinn, K. Sahu, Computational fluid dynamics using graphics processing units: challenges and opportunities, in: ASME 2011 International Mechanical Engineering Congress and Exposition, IMECE, 2011, vol. 6, 2011.
[29] Nvidia Tesla, V100 GPU Architecture, Nvidia Corporation, 2017.
[30] Developing a Linux Kernel Module Using RDMA for GPUdirect: Application Guide, Nvidia Corporation, 2012.
[31] H.C. Ku, R.S. Hirsh, T.D. Taylor, A pseudospectral method for solution of the three-dimensional incompressible navier-stokes equations, J. Comput. Phys. 70 (1987) 439–462.
[32] J. Koseff, R. Street, P. Gresho, C. Upson, J. Humphrey, W. To, Three-Dimensional Lid-Driven Cavity Flow: Experiment and Simulation, Pineridge Press, 1983.
[33] A. Prasad, C.-Y. Perng, J. Koseff, Some Observations on the Influence of Longitudinal Vortices in a Lid-Driven Cavity Flow, 1988, pp. 288–295.
[34] A.K. Prasad, J.R. Koseff, Reynolds number and end wall effects on a lid-driven cavity flow, Phys. Fluids A, Fluid Dyn. 1 (1989) 208–218.
[35] H. Abe, H. Kawamura, Y. Matsuo, Direct numerical simulation of a fully developed turbulent channel flow with respect to the reynolds number dependence, J. Fluids Eng. 123 (2001) 382–393.
[36] B.E. Owolabi, C.-A. Lin, Marginally turbulent couette flow in a spanwise confined passage of square cross section, Phys. Fluids 30 (2018) 075102.
[37] S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (2009) 65–76.
[38] D. Bailey, R. Lucas, S. Williams, Performance Tuning of Scientific Applications, CRC Press, 2010.
[39] Y. Lo Jung, S. Williams, B. Straalen, T. Ligocki, M.J. Cordery, N. Wright, M. Hall, L. Oliker, Roofline Model Toolkit: A Practical Yool for Architectural and Program Analysis, Springer, Cham, 2015, pp. 129–148.
[40] G. Ofenbeck, R. Steinmann, V.C. Cabezas, D.G. Spampinato, M. Püschel, Applying the roofline model, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 76–85.
[41] G.V. Candler, M.J. Wright, J.D. McDonald, Data-parallel lower-upper relaxation method for reacting flows, AIAA J. 32 (1994) 2380–2386.
[42] I. Tanno, K. Morinishi, N. Satofuka, Y. Watanabe, Calculation by artificial compressibility method and virtual flux method on gpu, Comput. Fluids 45 (1) (2011) 162–167.
[43] C. Cox, C. Liang, M.W. Plesniak, A high-order solver for unsteady incompressible navier–stokes equations using the flux reconstruction method on unstructured grids with implicit dual time stepping, J. Comput. Phys. 314 (2016) 414–435.