CCF Transactions on High Performance Computing
https://doi.org/10.1007/s42514-020-00034-9

REGULAR PAPER

Towards a heterogeneous architecture solver for the incompressible Navier–Stokes equations

Yunting Wang[1] · Xin He[1] · Shaofeng Yang[1] · Guangming Tan[1]

Received: 18 December 2019 / Accepted: 27 April 2020

Abstract
Large-scale supercomputers equipped with GPUs as accelerators are potential to satisfy the future Exascale computing. In this work the solution of large and sparse linear systems of equations by using the Krylov subspace methods, which is crucial for the overall performance of many industrial and scientific applications, is chosen to be accelerated by GPUs' greatly enlarged computing power. To fulfill this objective on the target hardware with a large amount of heterogeneous computing nodes, two main contributions are included in this work. First we propose a communication avoiding variant of the BICGStab solution method which reduces the global synchronization points per iteration from 3 in the classical BICGStab method to 1 in the improved variant. The superiority in terms of a reduction of the expensive global communications via all computing processes can be expected on a large-scale distributed memory cluster. Second, to handle the host-to-accelerator data transfers, the main challenge encountered in the usage of heterogeneous architecture, a communication overlapped implementation of the sparse matrix–vector multiplication is proposed since this kernel features heavily in the Krylov subspace methods. Linear systems of equations arising from the incompressible Navier–Stokes equations are used to evaluate the proposed solution and optimization methods. Evaluations of the GPU and CPU implementations are conducted on up to 256 GPUs and 4096 CPU cores, respectively. It is revealed that to obtain the same computation time a two times reduction of the number of computing nodes is achieved by using the GPU implementation on the heterogeneous node equipped with 4 GPUs and a 32-core CPU. This result can be seen as the advantage of the heterogeneous architecture from the view point of applications, which motivates a wide utilization in other related areas.

Keywords Heterogeneous architecture · Computational fluid dynamics · Krylov subspace methods · Sparse matrix–vector multiplication

1 Introduction

Moving forward to Exascale computing, the high performance computing (HPC) community has recognized the application of heterogeneous architectures as one of the key ingredients to satisfy the future computing requirements. These architectures augment central-processing units (CPUs) by accelerators such as the graphics-processing units (GPUs) which are designed to provide large throughput computing power. The greatly enlarged computational power of the heterogeneous hardware has therefore attracted attention of the scientific computing community looking for ways to accelerate numerical simulations.

Computational fluid dynamics (CFD) is an active area seeking for benefits from the integration of hardware. Consequently, considerable efforts have been devoted to the utilization of GPUs so that an acceleration of CFD applications can be expected. However, it is not easy to achieve this goal with reasonable efforts since usually both the structure and implementations of the CFD codes are not well suited for a GPU implementation. In addition, the existing implementations often contain a large amount of experience with respect to numerical and modeling issues, which is in particular essential for obtaining the reliable solutions of real world problems. It is not feasible to restart such a software project from scratch in order to achieve a better performance by GPU acceleration. Probably this is the main concern for the commercial CFD vendors to slowly adopt this technology.

✉ Xin He
hexin2016@ict.ac.cn

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

Published online: 25 May 2020

But in academia, several groups have already published CFD results on heterogeneous systems. A brief survey is presented in the literature review section of this work. At this moment, it appears that a practical way is to re-use as much as possible the existing implementations while the acceleration is achieved through a re-implementation of the computation intensive parts of the program, e.g. calculating the solutions of linear algebraic systems. The numerical kernel of each CFD software is an algorithm which solves the Navier–Stokes equations. From the discretization and linearization, a system of algebraic equations is obtained. More details are given in the later section. In this work, we focus on accelerating the Krylov subspace solution methods for large and sparse linear systems of equations on heterogeneous compute clusters equipped by GPUs. This is motivated by the fact that efficient implementations of Krylov subspace solution methods are crucial for many problems in industrial and scientific computing.

Although GPUs have been used for general purpose computations for several years already, the library ecosystem is still relatively small. Most libraries are provided by vendors directly. For example, NVIDIA provides several optimized libraries including dense and sparse linear algebra operations, e.g. CUSP (http://cusplibrary.github.io/) and CUSPARSE (http://developer.nvidia.com/cusparse/). Besides, Paralution (http://www.paralution.com/) is a C++ software library with a focus on providing iterative solvers and preconditioners for linear systems of equations with sparse coefficient matrices. It supports CUDA, OpenMP, and OpenCL compute backends and the compute backend must be selected at compile time. Contrary to Paralution, library ViennaCL (http://viennacl.sourceforge.net/) allows a just-in-time selection of different hardware types at run-time, which avoids the error-prone and time-consuming recompilation steps. To the best knowledge of authors, however, the existing GPUs-accelerated libraries only support programming based on shared memory machines. Since large-scale computations in industry can only be performed on distributed memory systems parallelized via the massage passing interface (MPI), library providing the MPI+GPU implementations of the linear solvers becomes a essential requirement to carry out HPC applications on heterogeneous compute clusters.

In this work we consider the BICGStab method, one of the Krylov subspace methods to solve the linear systems of equations with non-symmetric coefficient matrix. Due to the attractive feature, e.g. the short recurrence and improved numerical stability, the BICGStab method is widely adopted in industrial and academic computations. Since the large-scale heterogeneous cluster equipped with GPUs is the target platform, the following two improvements of the BICGStab method are proposed. First is an improved variant with a reduced number of global communications, which outperforms the standard counterpart due to bottleneck created by the expensive inter-nodal communications. Second, the important kernel of the BICGStab method as well as other Krylov subspace methods, i.e. the sparse matrix–vector multiplication is implemented in a communication overlapped manner. In this way, the overhead of data transfers between CPUs and GPUs, one drawback of the usage of heterogeneous system is hidden, at least partially.

Large and sparse linear systems of equations arising from the incompressible flows governed by the Navier–Stokes equations are utilized to evaluate the performance in terms of the speedup and parallel scalability. As far as we know, in most related references the reported results are only available on few compute nodes with multi-GPUs, which can not reveal the comprehensive behavior of the GPU-accelerated solvers on the large-scale heterogeneous clusters. In this sense we believe that the novel results of this work, conducted on up to 256 GPUs supplement the design and optimization techniques towards the large-scale heterogeneous computations. In addition, this work also demonstrates from the application view point the advantage of the heterogeneous architecture, which considerably reduces the amount of the required computing resources to achieve the same computational efficiency as the homogeneous system only equipped with CPUs.

The structure of this paper is as follows. The literature review of the GPU-accelerated CFD simulations is presented in Sect. 2. The Reynolds-Averaged Navier–Stokes equations which governs the incompressible flows are introduced in Sect. 3, followed by the introduction of finite volume discretization and SIMPLE solution method. Section 4 proposes the variant of the BICGStab iterative solution method for the so-arising linear systems of equations and the optimization techniques on the heterogeneous system. Section 5 includes the numerical experiments and conclusions and future work are outlined in Sect. 6.

2 Literature review

In this section, the literature is briefly investigated in order to understand the state-of-the-art with respect to CFD and parallel programming paradigms, especially with accelerators.

Accelerators include GPUs or co-processors (such as Intel's Xeon Phi) which are attached to a host node. GPUs use programming standards such as CUDA or OpenCL. Functions performed on GPUs must be re-written in an appropriate syntax. All forms of accelerators are designed to provide large throughput computing power and favour single-instruction-multiple-data (SIMD) vectorized routines. This is opposed to CPUs which are latency-optimized, focused on performing complex and varied tasks via large caches and diverse processing elements. The usage of accelerators is usually a trade-off between the speedup obtained from using high-throughput processing compared to the cost of moving data to the accelerator.

Gorobets et al. (2013) present a comprehensive CFD-acceleration study and translate the vast majority of CFD operations into OpenCL kernels, including those for the discretization, interpolation, gradient computation and linear-system-equation solving. The author demonstrates that the routines run 7.6–22.5x faster on an NVidia C2050 GPU and 11.6–96x faster on an AMD 7970 GPU, compared to the run-time on a standard Intel CPU (X5670). However, comparisons between CPUs and accelerators are not very revealing since the hardware is fundamentally different. The author goes on to show that the GPU kernels achieve very poor throughput compared to their theoretical peak. This is mostly due to the limited memory bandwidth and host-to-accelerator communication which creates a bottleneck. Furthermore, the code implemented by Gorobets et al. applies a structured Cartesian mesh which could be vectorized for efficient computations on GPUs.

Soukov et al. (2013) performs a similar study, specifically looking at higher-order discretization routines for unstructured CFD meshes. The GPU implementations show good performance up until memory bandwidth is saturated. In order to compare between CPUs and GPUs, the author shows that the speedup is greater than the relative electrical power consumption of the two devices, i.e. the GPU requiring 2.5x more power but producing a larger speedup. Similarly, Rossi et al. (2013) present the GPU acceleration of an unstructured finite-element code. The accelerator becomes more efficient as the test case becomes larger. When one million cells are offloaded, the GPU implementation becomes up to 4.11x faster than the CPU one.

Gorobets et al. (2014) compare the performance of NVidia GPUs, AMD GPUs and first-generation 5110P Xeon Phis for the unstructured discretization routines. The Xeon Phi performs poorly compared to the GPUs, but the author provides reasonable explanations for this. Most notably, the code is optimized for massive multi-threading and this is only possible in the inefficiently parallel parts of the CFD code, e.g. in the assembly routines for linear systems. The Xeon Phi behaves more like a standard CPU, with larger caches and more hierarchical memory access, which allow it to achieve a speedup on more complex routines with irregular memory access or inter-thread communication. In Liu et al. (2013), an NVidia K20X GPU is compared to a Xeon Phi for the sparse matrix–vector multiplication (SpMV) routine, which features heavily in the solve routines. Xeon Phi is considerably faster in this calculation, which could be much more meaningful for the overall CFD performance since the solving routine is expected to be a bottleneck. Gorobets et al. note that despite the poor comparison to GPUs, the Xeon Phi achieves a very good internal scaling, achieving approximately 83% parallel efficiency over its 240 threads, compared to the serial run-time on a single Xeon Phi thread.

Lattice Boltzman method (LBM) is another method of interest for solving fluid problems. It is favorable for massive parallelism due to its parallel nature. LBM is based on cellular automata. The fluid is modeled as particles on a discrete lattice of cells. Various GPU implementations of LBM exist in the literature. An early implementation of LBM on GPUs is done by Fan et al. (2004) while a more recent work is presented in Rinaldi et al. (2012), where a speedup of around 100 over a CPU implementation is achieved for the lid-driven cavity case.

3 Governing equations and solution methods

In this section, we introduce the Reynolds-Averaged Navier–Stokes equations, followed by the finite volume discretization method (FVM) and SIMPLE solution method.

3.1 Reynolds-Averaged Navier–Stokes equations

Incompressible and turbulent flows often occur in the CFD applications. Most commercial and open-source CFD packages rely on the Reynolds-Averaged Navier–Stokes (RANS) equations to model such flows (Ferziger and Peric 2012; Wesseling 2009; Patankar 1980) since more advanced models, such as the Large-Eddy Simulation (LES), are still too expensive for industrial applications. Besides, engineers are firstly interested in the averaged properties of a flow, such as the average forces on a body, which is exactly what RANS models provide.

The RANS equations are obtained from the Navier–Stokes equations by an averaging process referred to as the Reynolds averaging, where an instantaneous quantity such as the velocity, is decomposed into its averaged and fluctuating parts. If the flow is statistically steady, time averaging is used and ensemble averaging is applied for unsteady flows. The averaged part is solved, while the fluctuating part is modelled which requires additional equations, for instance for the turbulent kinetic energy and turbulence dissipation. We refer to Ferziger and Peric (2012), Wesseling (2009) and Patankar (1980) for a broader discussion. The Reynolds-Averaged equations are here presented in the conservative form using FVM for a control volume Ω with the surface S and outward normal vector $\mathbf{n}$:

$$\begin{aligned} &\int_S \rho \mathbf{u}\mathbf{u} \cdot \mathbf{n}\, dS + \int_S P\mathbf{n}\, dS - \int_S \mu_{\text{eff}}(\nabla \mathbf{u} \\ &\quad + \nabla \mathbf{u}^T) \cdot \mathbf{n}\, dS = \int_\Omega \rho \mathbf{b}\, d\Omega, \\ &\int_S \mathbf{u} \cdot \mathbf{n}\, dS = 0 \end{aligned} \tag{1}$$

where $\mathbf{u}$ is the velocity, $P = p + \frac{2}{3}\rho k$ consists of the pressure p and the turbulent kinetic energy k , ρ is the (constant) density, μ_{eff} is the (variable) effective viscosity and $\mathbf{b}$ is a given force field. On the boundaries we either impose the velocity ($\mathbf{u} = \mathbf{u}_{\text{ref}}$ on inflow and $\mathbf{u} = 0$ on walls) or the normal stress ($\mu_{\text{eff}} \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - P\mathbf{n} = 0$ on outflow and far field). The effective viscosity μ_{eff} is the sum of the constant dynamic viscosity μ and the variable turbulent eddy viscosity μ_{t} provided by the turbulence model as a function of k and possibly other turbulence quantities. Notice that for laminar flows, where k and μ_{t} are zero, the RANS equations reduce to the Navier–Stokes equations.

3.2 SIMPLE solution method

The nonlinear system (1) is solved for $\mathbf{u}$ and P as a series of linear systems obtained by the Picard linearization (Ferziger and Peric 2012), i.e. by assuming that the mass flux $\rho\mathbf{u} \cdot \mathbf{n}$, the turbulent kinetic energy k and the effective viscosity μ_{eff} are known from the previous iteration. The turbulence equations are then solved for k and possibly other turbulence quantities, after which the process is repeated until a convergence criterion is met.

Robust finite volume method for unstructured meshes with collocated variable arrangement is the discretisation choice in the industrial practice, and we refer to Ferziger and Peric (2012) for more details. After linearization and discretization of system (1) by the cell-centered and co-located FVM, the linear system reads as

$$\begin{bmatrix} Q & G \\ D & C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ g \end{bmatrix} \quad \text{with} \quad \mathcal{A} := \begin{bmatrix} Q & G \\ D & C \end{bmatrix}, \tag{2}$$

where Q corresponds to the convection–diffusion operator and the matrices G and D denote the gradient and divergence operators, respectively. The matrix C comes from the stabilization method to avoid pressure oscillations when the velocity and pressure are co-located in the cell centers. The pressure-weighted interpolation (PWI) method (Miller and Schmidt 1988) is a widely adopted choice.

System (2) is still coupled, i.e. the velocity components and pressure as physical unknowns appear in the same system of equations. A widely utilized algorithm in commercial CFD codes to solve the resulting system of algebraic equations is SIMPLE (semi-implicit pressure linked equation) by Patankar (1980). It is based on a segregated approach, i.e. only linear systems for a single physical unknown are solved at a time. The derivation of the SIMPLE algorithm can be more clearly seen from the following analysis and the starting point is the block $\mathcal{LU}$ decomposition of the coefficient matrix $\mathcal{A}$ given by

$$\mathcal{A} = \mathcal{LU} = \begin{bmatrix} Q & G \\ D & C \end{bmatrix} = \begin{bmatrix} Q & O \\ D & S \end{bmatrix} \begin{bmatrix} I & Q^{-1}G \\ O & I \end{bmatrix}, \tag{3}$$

where $S = C - DQ^{-1}G$ is the so-called Schur complement matrix. Based on this block $\mathcal{LU}$ decomposition, solving the coupled systems as (2) is divided into the solutions of the velocity sub-system with Q twice and the pressure sub-system with S. Comparing to the velocity sub-system, the more difficult part is to solve the pressure sub-system with S since the inverse action of Q is involved therein so that it is not practical to explicitly form S, especially for the large size computations. The key of the SIMPLE algorithm is to construct a numerically efficient and computationally cheap approximation of the Schur complement matrix S. SIMPLE algorithm provides the Schur complement approximation as

$$\widetilde{S}_{SIMPLE} = C - DF^{-1}G, \tag{4}$$

where $F = \text{diag}(Q)$ denotes the diagonal approximation of Q. With $\widetilde{S}_{SIMPLE}$, SIMPLE algorithm solves the following linear system instead of the original system (2)

$$\begin{bmatrix} Q & O \\ D & \widetilde{S}_{SIMPLE} \end{bmatrix} \begin{bmatrix} I & F^{-1}G \\ O & I \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ g \end{bmatrix} \quad \text{with}$$
$$\widetilde{\mathcal{A}} := \begin{bmatrix} Q & O \\ D & \widetilde{S}_{SIMPLE} \end{bmatrix} \begin{bmatrix} I & F^{-1}G \\ O & I \end{bmatrix}. \tag{5}$$

Solutions of systems (5) with $\widetilde{\mathcal{A}}$ are straightforward, namely,

Algorithm 1 The SIMPLE algorithm

1: Solve $Q\mathbf{u}^\star = \mathbf{f}$
2: Solve $\widetilde{S}_{SIMPLE} \cdot p = g - D\mathbf{u}^\star$
3: Compute $\mathbf{u} = \mathbf{u}^\star - F^{-1}Gp$

An overview of the SIMPLE algorithm is presented in Fig. 3. All non-linearity is tackled in the outer loop. Within each outer loop, the governing equations (velocity components, pressure, etc.) are solved in their uncoupled, linearized, discretized form. The solution of each equation follows the same steps as illustrated on the right of Fig. 1, i.e. assembly, solve, exchange and gradients. Since the MPI parallelization of large-scale simulations is based on a domain decomposition approach, each MPI process has its own portion of the computing grid. Therefore, the updated values for the flow-field variables along partition boundaries must be shared via MPI data exchange. The gradients of the flow-field variables can be computed using Gauss theorem and then exchanged across the domain boundaries once again.

Some remarks regarding the SIMPLE algorithm are as follows. The Schur complement approximation $\widetilde{S}_{SIMPLE}$ is expected to be effective when the matrix Q is diagonally dominant. This feature is satisfied for the steady simulations with a relatively small Reynolds number and the

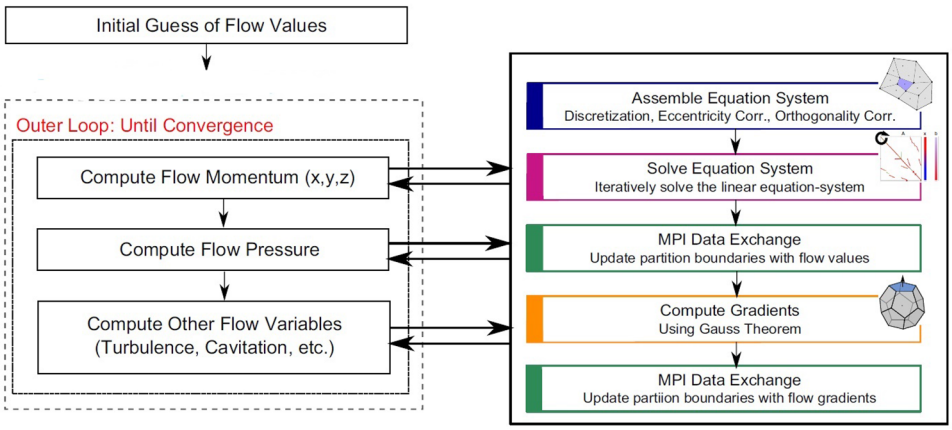


Fig. 1 An overview of the SIMPLE algorithm

time-dependent cases with small time-step sizes. As seen, at each nonlinear step SIMPLE algorithm approximates the original coefficient matrix $\mathcal{A}$ by $\widetilde{\mathcal{A}}$. Because of this, the convergence of the SIMPLE algorithm and its variants is expected to be slower than the coupled approach which solves the original system (2) with $\mathcal{A}$ by Krylov subspace methods. To accelerate the convergence of the coupled solution method, the matrix $\widetilde{\mathcal{A}}$ is used as a preconditioner, which is referred to as the SIMPLE preconditioner. We refer for more details of the preconditioning techniques to the surveys (Benzi et al. 2005; Saad et al. 2000; Pestana and Wathen 2015; Benzi 2002) and the books (Elman et al. 2014; Olshanskii and Tyrtyshnikov 2014).

The most expensive part of the SIMPLE algorithm is the solution of large and sparse linear systems for the velocity, pressure and other flow-field variables with non-symmetric coefficient matrices. Efficient solution methods and optimization techniques are discussed in the next section.

4 Iterative solution methods on heterogeneous systems

As CFD and other scientific and engineering computations governed by partial differential equations, one of the fundamental and time-consuming task is to solve linear systems of equations arising from the discretization of governing equations. Comparing to the direct solution method which obtains the solution through a LU factorization of the coefficient matrix, Krylov subspace methods are considered to be among the most important iterative techniques available for solving large and sparse systems. These techniques are based on projection processes, both orthogonal and oblique, onto Krylov subspaces. For a detailed and comprehensive study, we refer to Saad (2003).

4.1 The improved version of the BICGStab method

Motived by the attractive features introduced in the introduction section, in this work we apply the BICGStab method (der Vorst 1992), one of the Krylov subspace methods to solve the large and sparse linear systems with non-symmetric coefficient matrices. The BICGStab method is illustrated in Algorithm 2. For the Krylov subspace methods the basic computational kernels consist of inner products (dot), vector updates (for instance $ax + y$, called axpy operation) and sparse matrix–vector (SpMV) multiplications. Since the testbed considered in this paper is the distributed memory system, global communications, i.e. accumulation of data from all to one processor are needed for the inner products and neighbour-to-neighbour communications between only nearby processors are required for the sparse matrix–vector multiplications.

Neighbour-to-neighbour communications: Neighbouring partitions of a computing mesh must communicate data on their borders to a handful of processes, but not every process. As the number of partitions increases, namely with more processes, the number of neighbours should not change dramatically. Since various neighbours can communicate simultaneously, this is (mostly) a scalable communication pattern. Eventually, a decreasing cells-per-core ratio will reduce the size of these MPI messages such that they are limited by inter-nodal latency.

Global communications: Not only for the inner products, they are typically used to compute the vector norms or residuals across partitioned vectors in linear solvers. They perform a tree-like hierarchical communication pattern, where every single message is latency-bound. Global communications are not scalable and increase at a rate of $log_2(P)$, where P is the number of participating MPI processes.

In the classical BICGStab method, Algorithm 2 shows that three global communications are required at each iteration to compute the inner products at lines 5, 8 and 11

correspondingly. Frequent communications via all processes potentially deteriorate the parallel scalability, especially for a compute cluster with a low-speed inter-nodal connection. A numerically equivalent variant of the BICGStab method, which significantly reduces the amount of global communications is the main concern of this section. To realize this objective, we proposed an improved variant which does not change the numerical stability while requires only a single global synchronization point per iteration. The improved version is presented in Algorithm 3 and as seen that the global reduction operations of the six independent inner products at lines 14 and 15 can be efficiently done by a single global communication. This demonstrates that the number of the global synchronization points per iteration have been reduced from 3 in the classical BICGStab method to 1 in the improved counterpart. This advantage can be more clearly seen from Fig. 2 where the trace of the two versions are plotted. Therefore, a reduction of the global communication cost on distributed memory systems can be expected. Besides, the number of expensive SpMV operations keeps the same in the two versions, which means that in the proposed version the increase of arithmetic operations due to few additional vector updates is limited and acceptable.

The number of the fundamental kernels, i.e. dot, axpy and SpMV and the associated communications in the classical and improved versions of the BICGStab method is summarized in Table 1 for an easy comparison.

A similar improved version of the BICGStab method is proposed in Yang and Brent (2002) with the same strategy. To avoid repetition, we refer to Yang and Brent (2002) for the derivation of Algorithm 3. Contrary to the version in Yang and Brent (2002), the action of transposing the coefficient matrix A is totally avoided in Algorithm 3. The matrix transpose operation requires additional efforts to reset the structure of the distributed matrix and vector (Fig. 3), which could destroy the data locality especially when applying the structured grids. In this sense, the variant proposed in Yang and Brent (2002) is not implemented and compared with ours. Moreover, pure CPU implementations are considered in Yang and Brent (2002) on a computer without a detailed configuration. A comprehensive evaluation of both CPU and GPU implementations of the improved BICGStab method is illustrated in this paper.

Table 1 The number of the fundamental kernels and the associated communications in the classical (Algorithm 2) and improved (Algorithm 3) versions of the BICGStab method

	Computations			Communications	
	Axpy	SpMV	Dot	Neighbour-to-neighbour	Global reduction
Algorithm 2	4	2	4	2	3
Algorithm 3	9	2	6	2	1

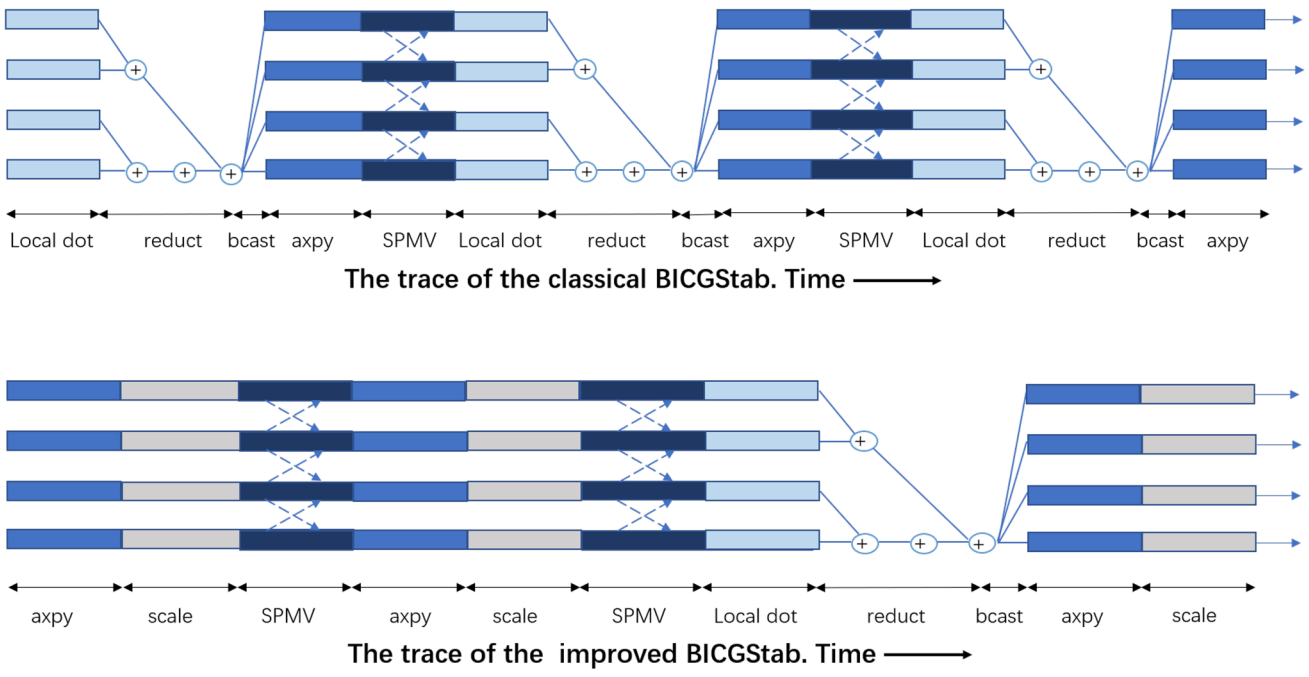
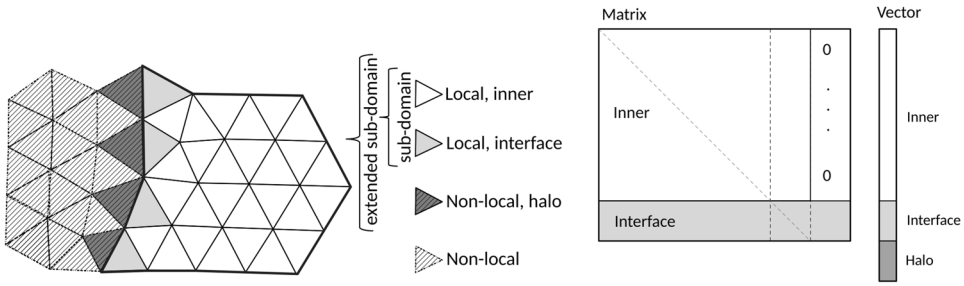


Fig. 2 Illustrative parallel trace of an iteration of the classical and improved versions of BICGStab method, showing the computation and communication patterns when utilizing 4 cores



(a) Domain decomposition (b) Distributed matrix and vector structure

Fig. 3 Domain decomposition and the structure of the distributed matrix and vector

Algorithm 2 The classical BICGStab Method

$r_0 = b - Ax_0$
$\rho_0 = \alpha_0 = w_0 = 1$, $v_0 = p_0 = 0$
for $n = 1, 2, 3, \cdots$, **do**
  $\rho_n = r_0^T r_{n-1}$
  $\beta = \frac{\rho_n}{\rho_{n-1}} \frac{\alpha_{n-1}}{w_{n-1}}$
  $p_n = r_{n-1} + \beta_n (p_{n-1} - w_{n-1} v_{n-1})$
  $v_n = A p_n$
  $\alpha_n = \frac{\rho_n}{r_0^T v_n}$
  $s_n = r_{n-1} - \alpha_n v_n$
  $t_n = A s_n$
  $w_n = \frac{t_n^T s_n}{t_n^T t_n}$
  $x_n = x_{n-1} + \alpha_n p_n + w_n s_n$
  $r_n = s_n - w_n t_n$
  if x_n is accurate enough **then**
    STOP
  end if
end for

Algorithm 3 The improved BICGStab Method

$r_0 = b - Ax_0$, $p_0 = AAx_0$, $t_0 = p_0 + Ar_0$, $\sigma_0 = r_0^T t_0$, $\phi_0 = r_0^T r_0$
$\pi_0 = \gamma_0 = m_0 = 0$, $q_0 = v_0 = z_0 = 0$, $\rho_0 = \alpha_0 = w_0 = 1$
for $n = 1, 2, 3, \cdots$, **do**
  $\rho_n = \phi_{n-1} - w_{n-1} m_{n-1}$
  $\delta_n = \frac{\rho_n}{\rho_{n-1}} \alpha_{n-1}$, $\beta_n = \frac{\delta_n}{w_{n-1}}$
  $\gamma_n = \sigma_{n-1} + \beta_n \gamma_{n-1} - \delta_n \pi_{n-1}$
  $\alpha_n = \frac{\rho_n}{\gamma_n}$
  $v_n = t_{n-1} - w_{n-1} p_{n-1} + \beta_n v_{n-1} - \delta_n q_{n-1}$
  $q_n = A v_n$
  $s_n = r_{n-1} - \alpha_n v_n$
  $t_n = t_{n-1} - w_{n-1} p_{n-1} - \alpha_n q_n$
  $z_n = \alpha_n r_{n-1} + \frac{\beta_n \alpha_n z_{n-1}}{\alpha_{n-1}} - \alpha_n \delta_n v_{n-1}$
  $p_n = A t_n$
  $\phi_n = r_0^T s_n$, $\pi_n = r_0^T q_n$, $\theta_n = s_n^T t_n$
  $k_n = t_n^T t_n$, $m_n = r_0^T t_n$, $\eta_n = r_0^T p_n$
  $w_n = \frac{\theta_n}{k_n}$
  $\sigma_n = m_n - w_n \eta_n$
  $x_n = x_{n-1} + z_n + w_n s_n$
  $r_n = s_n - w_n t_n$
  if x_n is accurate enough **then**
    STOP
  end if
end for

4.2 MPI+GPU implementation of basic operations

The heterogeneity with GPU accelerators is an important aspect to be taken into account when designing an efficient implementation of the target algorithm. Regarding the BICGStab method, the basic linear operations include the vector updates (e.g. axpy), inner products and sparse matrix–vector multiplications. The first one is perfectly suitable for the GPU acceleration and the last two kernels require communications between processors. With respect to the inner product, the local product is calculated by each active GPU processor and then the global sum is collected by means of a reduction communication via the MPI Allreduce functionality.

The most heavy task is the sparse matrix–vector multiplication kernel, which is non-trivial to be efficiently implemented on GPUs. As introduced before, it is necessary to update the ghost components of the vector from the nearby processors. With the standard non-overlapped SpMV approach, referred here as NSpMV, this update is performed by means of a communication procedure previous to the calculations. Obviously, this communication affects negatively the parallel performance.

In order to perform the overlapped SpMV (OSpMV), the matrix M owned by each GPU processor is split into its inner and external parts, i.e. $M = M_d + E_d$. The communications between nearby processors required in order to update the ghost components of the vector, i.e. $x_d^{(e)}$ are performed simultaneously with the calculation of the inner components of the product, i.e. $M_d \cdot x_d$. The calculation of the external components of the product, i.e. $E_d \cdot x_d^{(e)}$ is carried out when the communications are done. Finally, the product is updated by $t_d = M_d \cdot x_d + E_d \cdot x_d^{(e)}$. The overall process of the overlapped SpMV is given in Fig. 4. The number of ghost components of the vector is normally much smaller than that of the inner components since it only represents the couplings of the computational grids at the border of sub-domains. In this sense, the time spend on the ghost components update operation via a MPI layer and the data transfers between the CPUs and GPUs can be hidden at least partially by the calculation of the inner components of the product.

5 Numerical experiments

Numerical experiments are conducted on a large-scale cluster where each computing node is equipped with one AMD CPU and four latest AMD GPUs. Within one node, the two computing components are connected by PCIe bus (Gen 3.0). In our test, there are 1024 nodes (4096 GPUs) connected by a high bandwidth FatTree interconnection network. Each computing node is connected to the FatTree network by a high speed network interface of 200 Gb/s. The detailed configuration of the heterogeneous system is given in Table 2.

The CFD program with the finite volume discretization and SIMPLE solution method introduced in Sect. 3 is implemented by using C programming language and MPI parallelization (openmpi 4.0.1) to perform communications between parallel partitions of the computational domain. To utilize GPUs, the assembled matrices (represented in the compressed sparse row format) and right-hand side vectors available on each process are transfered from the hosts to GPUs. The solution of linear systems of equations is conducted on GPUs with the proposed BICGStab solver and communication overlapped sparse matrix–vector multiplication kernel. After that, the computed solution is transfered back to the hosts in order to continue the procedure of simulations. Instead of OpenCL, in this work the *hip* language is utilized to program on AMD GPUs.

5.1 Lid-driven cavity validation

A common benchmark and validation case for CFD code is the lid-driven cavity flow. It models an idealized recirculation flow with applications to environmental modeling, geophysics and industry. The reason for its popularity as validation case is due to its simple domain description and that it nevertheless exhibits fluid flow behavior of complex nature, such as counter rotating vortices.

In 3D simulations, the flow is modeled by a cubic regime surrounded by 6 walls. One of the walls is set to a constant velocity while the others are set to be stationary. Mathematically, Dirichlet boundary conditions are used for the velocity at all walls, while Neumann is used for the pressure. The interested reader is referred to Erturk (2009), where the lid-driven cavity flow is extensively discussed.

The flow is characterized by its Reynolds number, which is defined by $Re = \frac{\rho UL}{\mu}$, where L is the reference length of the domain and U is the reference velocity. The terms ρ and μ denote the density and dynamic viscosity, respectively. The density and dynamic viscosity of water at atmospheric pressure and 20 degrees Celsius are roughly $\rho = 1000$ (kg/m^3) and $\mu = 0.001$ (kg/m/s). The reference velocity U in (m/s), imposed on the top boundary is adjusted to obtain the given Reynolds number based on the length $L = 1$ (m) of this test case. The computed velocity stream lines in the x–y plane at $Re = 1000$ are plotted in Fig. 5, colored by the velocity magnitude. Observe the main vortex in the middle and the small ones in the lower corners. The validation plots can be seen in Fig. 6. The validation is seen to have a satisfactory agreement for all different validated Reynolds

Table 2 The computing node configuration

CPU		GPU	
Sockets	1	GPUs	4
Cores	32	DP(TFlops)	5.9
DP(GFlops)	384	Memory	16 GB
Memory	128 GB	Bandwidth	1 TB/s

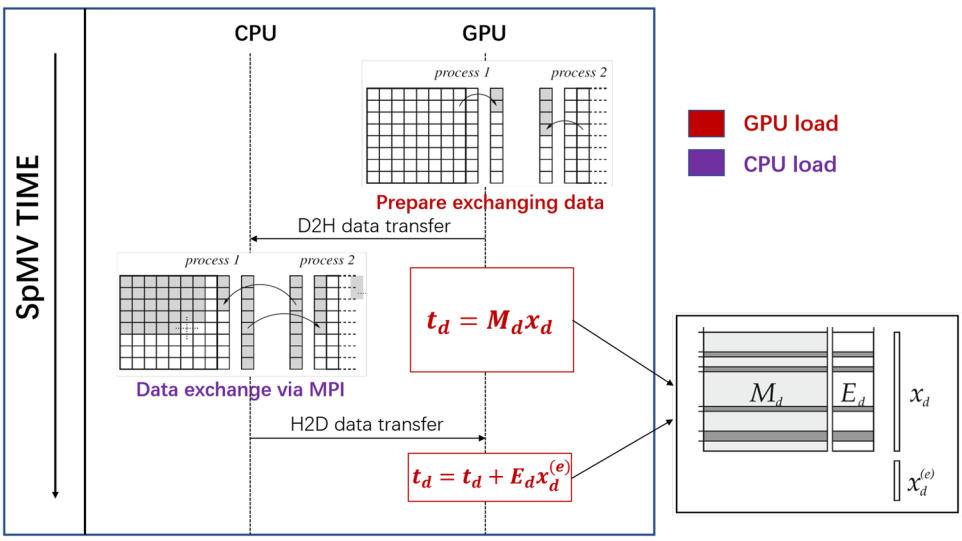


Fig. 4 Schematic representation of the communication overlapped SpMV kernel

numbers with the data provided by Erturk (2009) and Ghia et al. (1982).

In this work we limit ourselves to the simulations of laminar flows since the main focus is on the evaluation of the proposed GPU-accelerated solver. When the flow is turbulent, according to the SIMPLE algorithm additional linear systems of equations are solved in a segregated manner for other turbulence quantities. The application of the proposed GPU-accelerated solver is straightforward and a similar performance can be expected as that reported in the next section.

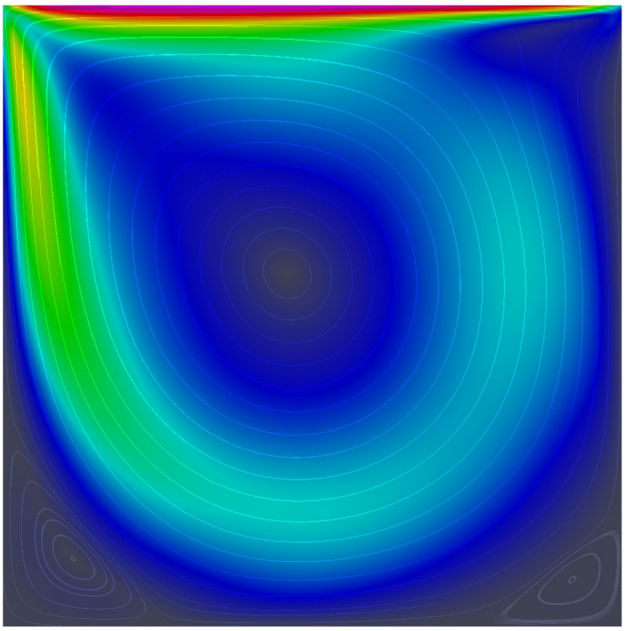
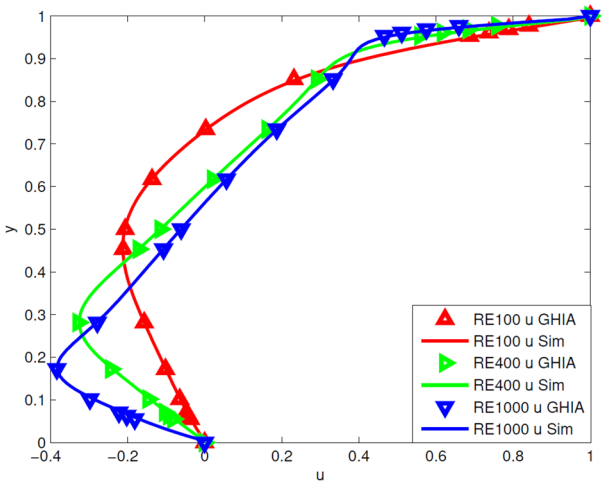


Fig. 5 Stream lines colored by the velocity magnitude for the lid-driven cavity case at $Re = 1000$

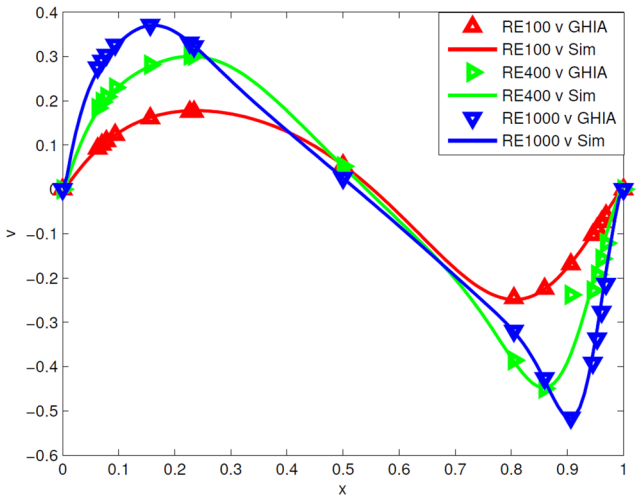
To handle the nonlinearity of the governing NS equations (1), some linearization techniques are applied, for example the Picard method considered in this work. In this sense, the nonlinear solution procedure consists of solving a sequence of the so-arising linear systems as (5) by the SIMPLE algorithm. The improved BICGStab method is utilized to solve the velocity sub-systems with non-symmetric coefficient matrices and the pressure sub-systems with symmetric and positive definite coefficient matrices are solved by the standard conjugate gradient (CG) method. In practice, it is not necessary to solve the sub-systems very accurately since it does not result in a faster convergence of the nonlinear solver. More practically, the solution procedure of the sub-systems is terminated when the desired accuracy or the maximum number of iterations is reached. In this work, we fix the maximum number of the BICGStab and CG iterations as 25 and 50, respectively. This leads to about 5000 nonlinear iterations to achieve the relative stopping tolerance 10^{-8}. Numerical experiments not presented here reveals that the number of nonlinear iterations varies slightly with the CPU and GPU implementations and different number of engaged devices. The above issues fall out of the scope of this work, which focuses on the efficient utilization of GPUs and the communication overlapped optimization to accelerate the BICGStab solution method.

5.2 Performance results

As introduced in Sect. 3, SIMPLE algorithm consists of solving a sequence of linear systems with non-symmetric coefficient matrices. The improved variant of the BICGStab



(a) u velocity along a vertical line via the geometrical center.



(b) v velocity along a horizontal line via the geometrical center.

Fig. 6 Validation of the 2D lid-driven cavity case with Reynolds numbers of 100, 400 and 1000, respectively, to the data of Ghia et al. (1982)

method and the optimization techniques illustrated in Sect. 4 are utilized to solve the so-arising linear systems. In this work, the velocity sub-system is chosen to evaluate the proposed solution methods. Performance of the solver accelerated by GPUs is evaluated against CPU implementation for the lid-driven cavity case with 500^3 uniform cells. Performance comparison in terms of the execution time for 25 iterations is demonstrated in Fig. 7. In x-axis the definition of processor is different for the two implementations, i.e. one processor corresponding to a GPU card and a CPU core, respectively. From Fig. 7 we have the following observations.

1. GPU implementation achieves a significant speedup compared to the CPU counterpart, at least 10 times faster. The speedup is up to 27 with 8 processors while decreases to about 10 with 256 processors. The factor leading to such a deterioration of speedup is that the strong scalability of the GPU implementation is not comparable with the CPU one. The design of CPU and GPU is fundamentally different and each GPU is equipped with much more computing units than one CPU. This implies that for a fixed problem size the ratio of work-per-unit when applying GPUs is much less than that of CPUs, which leads to a reasonably worse scalability of GPU implementation. In the related references the iterative solvers have never been evaluated on so many GPUs, so that the revealed performance in terms of the strong scalability and speedup over CPU implementation is a supplement in this research area.
2. Another observation is that to obtain the same computational time as 256 GPUs, up to 4096 cores are required when performing the CPU implementation. For the utilized cluster syetem, each computing node consists of a 32-cores CPU and 4 GPUs. Regarding the pure CPU implementation totally 128 computing nodes are required. The number of required nodes is 64 when apply the GPU implementation. This demonstrates that a reduction of computing resources by a factor 2 is achieved by using the heterogeneous system equipped with GPUs. This result can be seen as the advantage of the heterogeneous architecture, which motivates a wide utilization in the related areas.

In Sect. 4.2, we propose an improvement of the sparse matrix–vector multiplication kernel, i.e. the communication overlapped variant OSpMV. The advantage over the non-overlapped implementation (NSpMV) is demonstrated in Fig. 8. As seen, for a moderate number of processors (less than 64) the computations overlap most of the communication overhead, which leads to a two times acceleration over NSpMV. However, this superiority turns less obvious by increasing the processors, which is expected. Although

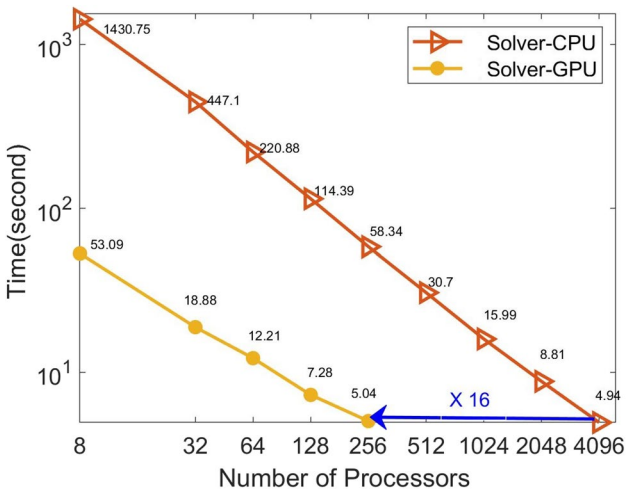


Fig. 7 Performance comparison between the CPU and GPU implementations of the improved BICGStab method

both the computation and communication tasks decrease with more processors, the decreasing ratio of the computation time is larger than that of the communication expense. Therefore, the overhead of the neighbour-to-neighbour communications and data transfers between CPUs and GPUs become dominant with more computing processors.

Figure 9 presents the time distribution of all components involved in the GPU implementation of the improved BICGStab solver on 256 GPUs. Several conclusions can be made based on the reported results therein. The global communication needed for the inner products takes the smallest percentage of the total execution time thanks to the high-speed inter-nodal network and the optimized MPI Allreduce functionality. The motivation to develop the improved variant of the BICGStab solver is to significantly reduce the overhead arising from the global communications, which results in an improved scalability behavior. Compared to the standard BICGStab method, the superiority of the improved variant is more visible when the global communications are slow and dominant. For this reason, the comparison of two versions of the BICGStab method is not included in this work. On the other hand, the proposed BICGStab variant is still attractive for the hardware with a slower and cheaper inter-nodal network.

To perform the sparse matrix–vector multiplication kernel, the ghost elements of the vector from the nearby processors are needed. However, since the matrices and vectors are stored in GPUs data transfers between CPU and GPU are required to update the ghost elements before and after the neighbour-to-neighbour communication via MPI interface. Thus, the overhead of CPU-GPU data transfers is a non-negligible factor which could affect the overall performance. As seen Fig. 9, the time spent on CPU-GPU data transfers is comparable with the neighbour-to-neighbour

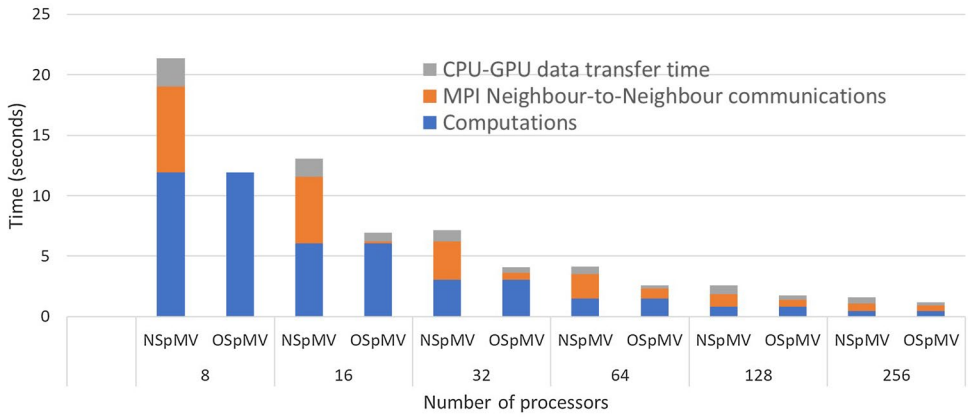


Fig. 8 Comparison of the non-overlapped SpMV (NSpMV) and overlapped SpMV (OSpMV) implementations

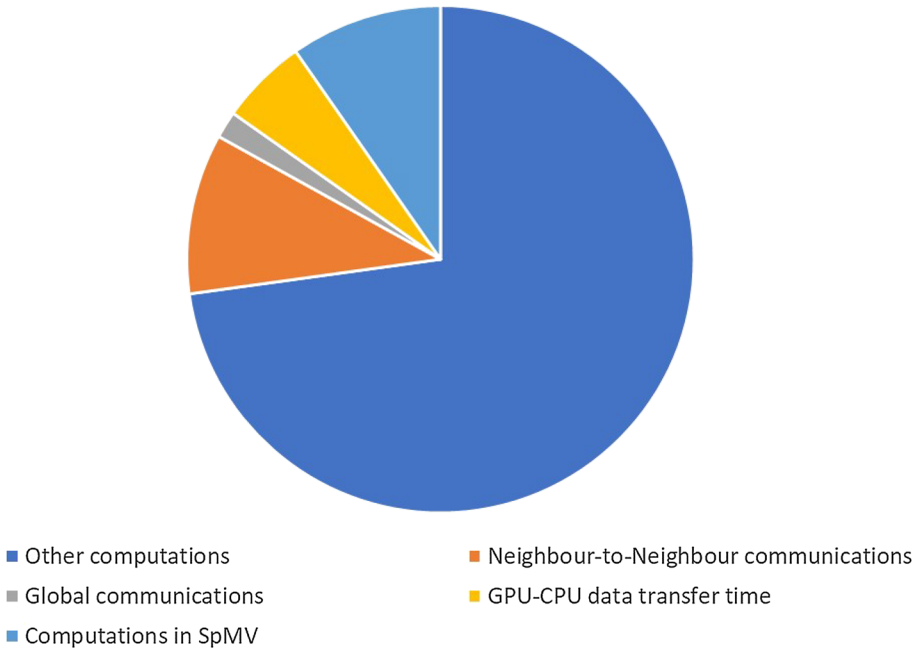


Fig. 9 Time distribution of all components involved in the improved BICGStab solver on 256 GPUs

communications. This drawback could be encountered when using any heterogeneous hardware with accelerators and efforts to handle this drawback should be contributed from both the programmers and vendors. The re-organization of the iterative solution methods with a reduced data transfers is considered as one direction of future research.

In Table 3 we illustrate the performance comparison in terms of the execution time between the PETSc and our implementations of the improved BICGStab method. The improved variant of the BICGStab method implemented in PETSc library is exactly that proposed by Yang and Brent (2002), which requires the matrix transpose action and thus is slightly different from Algorithm 3 given in Sect. 4. In this way, the comparison is reasonably fair. Since PETSc does not support the AMD GPU testbed, the comparison is conducted on the AMD CPUs. As seen from Table 3, our implementation is about two times faster than PETSc. Since PETSc does not publish its codes, we think that the superiority of our implementation partially arises from the circumvention of the matrix transpose action.

6 Conclusion and future work

The main motivation of this work is to develop solution methods which efficiently utilize the enlarged computing power of the large-scale heterogeneous supercomputers. To realize this objective, we propose a communication avoiding variant of the BICGStab solution method and a CPU-GPU communication overlapped implementation of the sparse matrix–vector multiplication, which features heavily in the Krylov subspace methods. The advantage of the heterogeneous architecture is illustrated by the two times reduction of the required computing resources, which is achieved by

Table 3 Performance comparison in terms of the execution time (in s) between the PETSc and our implementations of the improved BICGStab method on CPUs

# CPU cores	8	32	64	128	256
Our-IBICGStab	1430.75	447.1	220.88	114.39	58.34
PETSc-IBICGStab	2530.25	900.20	450.11	220.91	128.35

applying the GPU implementation of the proposed algorithms. Contrary to other related references most performing experiments on multi-GPUs on a shared memory system, up to 256 GPUs and 4096 CPU cores on a distributed memory cluster are utilized in this work to evaluate the proposed solver, which results in a deeper insight of the GPU-accelerated computations.

Numerical explorations demonstrate that the communication overlapped implementation of the sparse matrix–vector multiplication is effective to overlap the data transfers between CPUs and GPUs for a moderate number of processors. The explanations have been given in the numerical experiment section. This implies that other effective efforts should be explored to handle the drawback of the usage of heterogeneous architecture, i.e. the host-to-accelerator communication. The re-organization of the solution procedure, which leads to a reduced amount of data transfers from the algorithm level, is considered as one research direction in future. In addition to the solution phase, other essential parts of the CFD code, e.g. the discretization, interpolation and assembly routines are planned to be accelerated on the heterogeneous cluster. Since these kernels are not well suited for the GPU parallelization, careful optimizations are required.

Acknowledgements We thank to Prof. Shuangling Hu from China Academy of Engineering Physics for his great help on implementing the CFD codes, where the finite volume discreatization kernel is provided by him

References

Benzi, M.: Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. **182**, 418–477 (2002)

Benzi, M., Golub, G., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**(1), 1–137 (2005)

der Vorst, H.V.: Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **13**(2), 631–644 (1992)

Elman, H., Silvester, D., Wathen, A.: Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics. Oxford University Press, Oxford (2014)

Erturk, E.: Discussions on driven cavity flow. Int. J. Numer. Meth. Fluids **60**, 275–294 (2009)

Fan, Z., Qiu, F., Kaufman, A., Stover, S.: GPU cluster for high performance computing. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, p. 47. IEEE (2004)

Ferziger, J., Peric, M.: Computational Methods for Fluid Dynamics. Springer, Berlin (2012)

Ghia, U., Ghia, K., Shin, C.: High resolutions for incompressible flow using the Navier–Stokes equations and a multigrid method. J. Comput. Phys. **48**, 387–411 (1982)

Gorobets, A., Trias, F., Borrell, R., Oliva, A.: Direct numerical simulation of turbulent flows with parallel algorithms for various computing. In: 6th European Conference on Computational Fluid Dynamics (ECFD VI), Barcelona, Spain (2014)

Gorobets, A., Trias, F., Oliva, A.: A parallel MPI+OpenMP+OpenCL algorithm for hybrid supercomputations of incompressible flows. Comput. Fluids **88**, 764–772 (2013)

Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, ICS 2013, New York, USA, pp. 273–282. ACM (2013)

Miller, T., Schmidt, F.: Use of a pressure-weighted interpolation method for the solution of the incompressible Navier–Stokes equations on a nonstaggered grid system. Numer. Heat Transfer Part A Appl. **14**(2), 213–233 (1988)

Olshanskii, M., Tyrtyshnikov, E.: Iterative Methods for Linear Systems: Theory and Applications. SIAM, Philadelphia (2014)

Patankar, P.: Numerical Heat Transfer and Fluid Flow. McGraw-Hill, New York (1980)

Pestana, J., Wathen, A.: Natural preconditioning and iterative methods for saddle point systems. SIAM Rev. **57**(1), 71–91 (2015)

Rinaldi, P., Dari, E., Venere, M., Clausse, A.: A Lattice–Boltzmann solver for 3D fluid simulation on GPU. Simul. Model. Pract. Theory **25**, 163–171 (2012)

Rossi, R., Mossaiby, F., Idelsohn, S.: A portable OpenCL-based unstructured edge-based finite element Navier–Stokes solver on graphics hardware. Comput. Fluids **81**, 134–144 (2013)

Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)

Saad, Y., der Vorst, V., Henk, A.: Iterative solution of linear systems in the 20th century. J. Comput. Appl. Math. **123**(1), 1–33 (2000)

Soukov, S., Gorobets, A., Bogdanov, P.: Opencl implementation of basic operations for a high-order finite-volume polynomial scheme on unstructured hybrid meshes. Proced. Eng. **61**, 76–80 (2013)

Wesseling, P.: Principles of Computational Fluid Dynamics. Springer, Berlin (2009)

Yang, L., Brent, R.: The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing, pp. 324–328. IEEE (2002)