Future Generation Computer Systems 56 (2016) 759–765

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A hybrid solution method for CFD applications on GPU-accelerated hybrid HPC platforms

Xiaocheng Liu [a,1], Ziming Zhong [b,*,1], Kai Xu [a]

[a] College of Information System and Management, National University of Defense Technology, Changsha, Hunan, China
[b] Complex Aviation Systems Simulation Laboratory, Beijing, China

HIGHLIGHTS

- We propose a hybrid solution method for CFD applications for CPU+GPU platforms.
- We propose a domain decomposition method based on the functional performance model.
- We evaluate the proposed methods with the lid-driven cavity flow application.

ARTICLE INFO

Article history:
Received 9 April 2015
Received in revised form
16 July 2015
Accepted 4 August 2015
Available online 28 August 2015

Keywords:
Computational fluid dynamics
GPU-accelerated multicore system
Performance modeling
Hybrid computing
Data partitioning

ABSTRACT

Heterogeneous multiprocessor systems, where commodity multicore processors are coupled with graphics processing units (GPUs), have been widely used in high performance computing (HPC). In this work, we focus on the design and optimization of Computational Fluid Dynamics (CFD) applications on such HPC platforms. In order to fully utilize the computational power of such heterogeneous platforms, we propose to design the performance-critical part of CFD applications, namely the linear equation solvers, in a hybrid way. A hybrid linear solver includes both one CPU version and one GPU version of code for solving a linear equations system. When a hybrid linear equation solver is invoked during the CFD simulation, the CPU portion and the GPU portion will be run on corresponding processing devices respectively in parallel according to the execution configuration. Furthermore, we propose to build functional performance models (FPMs) of processing devices and use FPM-based heterogeneous decomposition method to distribute workload between heterogeneous processing devices, in order to ensure balanced workload and optimized communication overhead. Efficiency of this approach is demonstrated by experiments with numerical simulation of lid-driven cavity flow on both a hybrid server and a hybrid cluster.

1. Introduction

Computational Fluid Dynamics (CFD) is the analysis of systems involving fluid flow, heat transfer, and associated phenomena by means of computer-based numerical simulation [1]. Over the past few decades, CFD has become a practical cornerstone of most fluid and mechanical engineering applications, such as aerodynamics of aircraft and vehicles.

The governing equations of fluid motion are partial differential equations, which are difficult to solve analytically. Therefore, numerical methods are usually used. CFD simulations commonly require a significant amount of computational resources for accurate solutions, especially for complex simulation scenarios such as transonic or turbulent flows. With high performance computing (HPC) platforms, faster and better solutions can be achieved. Therefore, parallelization of numerical simulations of fluid dynamics is an attractive research topic from the very beginning in this area.

The current trend in gaining high computing power is to incorporate specialized processing resources such as graphics processing units (GPUs) in multicore systems, thus making a computing system heterogeneous. Over recent years, a large number of critical scientific software have been ported to multicore and GPU architectures. Meanwhile, there has been extensive research on utilizing hybrid GPU-accelerated multicore platforms. However, most of the state-of-the-art CFD packages, e.g. OpenFOAM [2], are designed for facilitating the implementation of CFD applications on traditional parallel platforms based on CPU processors. Although

* Corresponding author.
E-mail address: zzm_nudt@163.com (Z. Zhong).
[1] Both authors contributed equally to this work.

http://dx.doi.org/10.1016/j.future.2015.08.002
0167-739X/

many efforts have been invested in improving the performance of CFD simulations on traditional HPC platforms, more is required for hardware-accelerated hybrid HPC platforms.

In this work, we focus on the design and optimization of CFD applications on GPU-accelerated parallel platforms. In a CFD application, the performance-critical part is its linear equation solvers. In order to exploit the computational power of target platforms, we propose to develop hybrid linear equation solvers. A hybrid linear equation solver consists of both one CPU version and one GPU version of code required for solving a linear equations system. The CPU version and GPU version of code are implemented using the programming languages and optimized numerical libraries for CPU and GPU architectures respectively. When a hybrid linear equation solver is invoked during the simulation, the CPU portion and the GPU portion will be run on corresponding processing devices respectively in parallel according to the execution configuration. During the simulation, a portion of the workload will be offloaded to GPU to accelerate the computation, and the result will be uploaded the host when the computation is finished.

Parallel computing in CFD simulations is usually based on domain decomposition, which is essentially data parallelism [1]. To achieve the maximum performance of CFD applications on hybrid HPC platforms, it is important to balance the workload and minimize the volume of communication among processing devices. To this end, we propose a method of heterogeneous domain decomposition. First, by benchmarking the linear equation solvers of a CFD application, we built the functional performance models [3] of processing devices. The FPMs are constructed empirically, and integrate many important features characterizing the performance of both the architecture and the application [4]. Then, the relative speeds of processing devices are calculated using FPM-based data partitioning algorithms, which have been proved accurate and efficient in [5]. Last, the relative speeds are provided to graph partitioning softwares, such as Metis [6], Scotch [7], as input (weights) to subdivide the domain into multiple subdomains, each assigned to one processing device. The graph partitioning softwares guarantee a balanced workload and an optimized communication overhead between heterogeneous processing devices. The problem will then be solved on the entire domain from problem solutions on subdomains.

In this work, without loss of generality, we experiment with a typical CFD application, namely the numerical simulation of the lid-driven cavity flow, on both a GPU-accelerated multicore node and a cluster of hybrid nodes. We develop the hybrid Conjugate Gradient (CG) [8] and Bi-Conjugate Gradient Stabilized (BiCGSTAB) [9] linear equation solvers for GPU-accelerated computing systems, build functional performance models of the processing devices of the experimental platforms, and decompose the domain based on these performance models. Experimental results demonstrate that the proposed heterogeneous domain decomposition methods based on performance models are able to balance the workload and deliver good performance.

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 briefly describes the computational fluid dynamics, the numerical methods and its parallelization. Section 4 presents the proposed hybrid solution method. Section 5 presents the experimental settings and results. Section 6 concludes the paper.

2. Related work

2.1. Computational fluid dynamics

Over the past few decades, a number of CFD packages have been developed, such as OpenFOAM [2], FEniCS [10], PyFR [11]. Most of them are aimed at homogeneous parallel platforms, and use the domain decomposition method to realize process-level parallelism of CFD applications. With the advent of GPU computing [12], many works have been carried out on developing linear equation solvers on GPUs for the speedup of CFD applications [13–15]. The recent progress and challenges in exploiting graphics processors in computational fluid dynamics are reviewed in [16]. Furthermore, there have also been some study on optimizing the CFD applications on GPU-accelerated HPC platforms. For example, [17] presents a high-order flow solver using multi-block structured grids on GPU clusters and propose a workload balancing model for distributing the workload between CPUs and GPUs. [18] proposes methods for designing and optimizing OpenFOAM-based CFD applications on hybrid and heterogeneous HPC platforms.

2.2. Performance models

Performance modeling is a very common technique used for optimization of scientific applications on parallel HPC platforms. A large number of performance models have been proposed for multicore and GPU architectures, such as the Roofline model [19] for multicore processors, and the integrated power and performance prediction model for GPU architecture [20]. The functional performance model (FPM) [5] represents the processor speed by a continuous function of the problem size. The speed is defined as the number of computation units performed per one time unit, and is found experimentally by measuring the execution time. A functional performance model consists of a series of speed measurements taken over a range of problem sizes. In many computationally intensive applications, the performance-critical part that is representative of the whole application in terms of performance can be extracted and used for performance measurement [21].

2.3. Load balancing algorithms

The load balancing algorithms can be classified into two categories, namely *static* and *dynamic* algorithms. Static algorithms, such as data partitioning [22–24], require a priori information about the parallel application and platform. Static algorithms rely on accurate performance models as input to predict the future execution of the application. Static algorithms are particularly useful for applications for which data locality is critical because data redistribution incurs significant overhead. However, these algorithms are unable to balance on non-dedicated platforms, where load changes with time. Dynamic algorithms, such as task scheduling and work stealing [25–30], balance the load by moving fine-grained tasks between processors during the execution. Dynamic algorithms do not require a priori information about execution but may incur large communication overhead due to data migration. A number of FPM-based data partitioning algorithms have been proposed, such as the geometric partitioning algorithm [5] and the dynamic data partitioning algorithm [31]. In this work, we use the FPM-based data partitioning to calculate the accurate relative speeds of processing devices of heterogeneous platforms.

2.4. Graph partitioning

Several software packages for graph partitioning have been developed, which can be used to partition the graph representing a system of sparse linear equations. The goal is to subdivide the graph into smaller subgraphs in order to balance the workload and to minimize the volume of communication among processors. Graph partitioning algorithms implemented in Metis [6], Scotch [7], Jostle [32] reduce the number of edges between the target subdomains, aiming to minimize the total communication cost

of the parallel application. They take into account the platform heterogeneity, which is specified by a weighted graph providing information about the speed of processors and the bandwidth of links. Algorithms implemented in PaGrid [33], Zoltan [34] minimize the execution time of the application using a cost function, which also depends on the weighted graph of the platform. To distribute data between the processors, all these software packages use simplistic computation performance models, where the processor speeds are defined by constants (weights). In this work, we use the more realistic FPMs to calculate the relative speeds of processing devices of heterogeneous platforms. The relative speeds are then provided to graph partitioning softwares as input (weights) for the calculation of a balanced partitioning of the graph.

3. Problem formation

Over the past few decades, a number of CFD packages have been developed. However, most of them are aimed at traditional homogeneous parallel platforms. For example, OpenFOAM [2] is one of the most popular open source framework for developing CFD applications. It provides sufficient functionalities, including mesh handling, finite volume discretization, inter-process communication, and provides a wide range of linear equation solvers. In this work, we choose to use the OpenFOAM to implement the CFD test case for experiments (Section 5.1). While the GPU-accelerated multicore platforms have become the mainstream in HPC, OpenFOAM only supports homogeneous parallel CPU platforms currently. The reason is that hybrid solution methods are still not available in OpenFOAM. In tis work, we present a hybrid solution method which can be used in OpenFOAM-based CFD applications to exploit the computational power of GPU-accelerated parallel platforms.

Parallelization of numerical simulations of fluid dynamics is usually based on domain decomposition, which is essentially data parallelism [1]. In methods based on domain decomposition, the solution domain is divided into a number of subdomains, each assigned to one processor. The problem is solved on the entire domain from problem solutions on subdomains. The same program runs on all processors simultaneously, on its own set of data. The exchange of data between processors is performed when necessary. When solving a CFD problem on a parallel platform, it is very important to decide how to map data to processors. Software packages for graph partitioning, e.g. Metis, are usually used to facilitate the domain decomposition of CFD applications. However, despite that the result of graph partitioning is very sensitive to the weights, these software packages do not provide any methods to find the accurate weights. Therefore, such graph partitioning cannot guarantee a balanced workload and optimized communication overhead of CFD applications on heterogeneous platforms. In this work, we propose a heterogeneous domain decomposition based on functional performance models for balancing the workload and optimizing the communication overhead of OpenFOAM-based CFD applications.

4. Hybrid solution method

In this section, we present a hybrid solution method for CFD applications run on GPU-accelerated HPC platforms. First, the method of implementing hybrid linear solvers is described. Then, the methods for building functional performance models of CFD applications and for using FPM-based domain decomposition to balance the workload and optimize the communication overhead are illustrated.

In order to run CFD applications efficiently on GPU-accelerated computing systems, we propose to design the performance-critical part, i.e. the linear equation solver, in a hybrid way. The CFD application is started on the host CPU system. When the hybrid linear equation solver is invoked during the simulation, a proper portion of the workload is offloaded to the GPU to accelerate the computation, and the result is uploaded to the host when the computation is finished.

Algorithm 1: Parallel Conjugate Gradient Algorithm

k = 0;
$\boldsymbol{\rho}_0 = \boldsymbol{Q} - A\boldsymbol{\phi}_0$;
while *(not convergent)* **do**
  $\boldsymbol{z}_k = M^{-1}\boldsymbol{\rho}_k$;
  $k = k + 1$;
  $s_k = \boldsymbol{\rho}_{k-1}^T \boldsymbol{z}_{k-1}$;
  GC: gather and scatter s_k;
  if $k = 1$ **then**
    $\boldsymbol{p}_1 = \boldsymbol{z}_0$;
  else
    $\beta_k = s_k / s_{k-1}$;
    $\boldsymbol{p}_k = \boldsymbol{z}_{k-1} + \beta_k \boldsymbol{p}_{k-1}$;
  LC: exchange $\boldsymbol{p}_k$ along interfaces;
  $\boldsymbol{v}_k = A\boldsymbol{p}_k$;
  $w_k = \boldsymbol{p}_k^T \boldsymbol{v}_k$;
  GC: gather and scatter w_k;
  $\alpha_k = s_k / w_k$;
  $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} + \alpha_k \boldsymbol{p}_k$;
  $\boldsymbol{\rho}_k = \boldsymbol{\rho}_{k-1} - \alpha_k \boldsymbol{v}_k$;
  GC: gather and scatter $\|\boldsymbol{\rho}_k\|_2$;

The hybrid linear equation solver is designed in the following way. First, both the CPU version and the GPU version of the linear equation solver are implemented, using the programming languages and optimized BLAS libraries for CPU and GPU architectures respectively. For example, the C/C++ and Intel MKL library [35] can be used to implement the CPU-version linear solver, and the CUDA programming language [36] and CUBLAS library [37] can be used to implement the GPU-version linear solver. Then, the CPU version and the GPU version of code are encapsulated in the hybrid linear solver, as shown in the pseudo code below:

```
void Hybrid_Solve(args) {
  Initialization();
  if (args.device_type=="cpu") {
    /* invoking the CPU version of code */
    CPU_Parallel_Solve(args);
  } else {
    /* invoking the GPU version of code */
    GPU_Parallel_Solve(args);
  }
  Finalization();
}
```

In function *Initialization*(), preparatory work such as parsing the execution configuration, offloading the coefficient matrices of the linear equation system to GPU, is implemented for the execution of the hybrid solver. In functions *CPU_Parallel_Solve*() and *GPU_Parallel_Solve*(), linear algebra operations required for solving the linear equations system are implemented by reusing CPU- and GPU-optimized BLAS kernels respectively. According to the execution configuration, more precisely the *device_type* argument here, the CPU version and the GPU version of code are executed on corresponding processing devices in parallel. Parallelism of the hybrid solver is achieved by employing the single program multiple data paradigm. In function *Finalization*(), operations for the termination of the hybrid solver are performed.

For the purpose of illustration, we present the pseudo-code of a basic linear equation solver, namely the Conjugate Gradient

(CG) solver, in Algorithm 1. Matrix A is a diagonal block of the global iteration matrix which represents a particular subdomain, $\boldsymbol{\phi}_0$ is an initial guess of the result, and matrix M represents the preconditioning matrix and the way of preconditioning. All data arrays, namely matrix A and vectors $\boldsymbol{\phi}$, $\boldsymbol{\rho}$, $\boldsymbol{z}$ and $\boldsymbol{p}$, are local parts for a particular processor operating on a particular subdomain. The main linear algebra operations involved in the CG solver include matrix–vector products, vector–vector products, and some vector additions and subtractions. These linear algebra operations can be implemented by calling routines from CPU- and GPU-optimized BLAS libraries. There exist two types of communications in the CG solver, namely the *local* and *global* communication. The local communication (LC) takes place between processors handling neighboring subdomains, while the global communication (GC) involves all processors. In this work, the global communication are performed through MPI collective communication routines. For local communication between processors, non-blocking MPI point-to-point communication routines are used to reduce overall communication overhead.

In the CFD simulation, solving the sparse linear equation systems is the compute-intensive and time-consuming part. Therefore, we build functional performance models of the processing devices executing the linear equation solver, e.g. CG and BiCGSTAB. The speed is calculated by dividing the number of floating-point operations by the computation time of one iteration of the linear equation solver (communication time eliminated). Next, using the FPM-based data partitioning algorithm [5,38], we calculate the numbers of control volumes (CVs) [1] to be assigned to processing devices, which are proportional to their speeds. Using graph partitioning package embedded in OpenFOAM, e.g. Metis, the solution domain is then decomposed into a number of subdomains so that each processing device handles a subdomain with the number of control volumes assigned to it. In this way, the workload is balanced on the target hybrid platforms and the total volume of communication between processing devices are minimized. In this work, the building of FPMs and the FPM-based data partitioning on experimental platforms are realized with the help of the FuPerMod package [21].

5. Evaluation

5.1. Test case: lid-driven cavity flow

The lid-driven cavity flow is a well-known benchmark problem for incompressible, lamina flow of Newtonian fluids. This test case has been studied by many researchers, and accurate solutions are available in the literature (see [39]). The standard case is fluid contained in a square domain with Dirichlet boundary conditions on all sides, with three stationary sides and a lid moving with a tangential unit velocity. The fluid density can be assumed constant for incompressible flows, hence, the Navier–Stokes equations reduce to:

$$\begin{cases} \partial_t \boldsymbol{U} + (\boldsymbol{U} \cdot \nabla)\boldsymbol{U} - \dfrac{1}{Re}\Delta \boldsymbol{U} + \nabla p = 0 \\ \nabla \cdot \boldsymbol{U} = 0 \end{cases}$$

to be solved in $\Omega = [0, 1] \times [0, 1]$. In these equations, $\boldsymbol{U}$ is the velocity vector, t is time, Re is the Reynolds number, and p is the pressure.

For this test case, a system of pressure–velocity coupled equations is produced from discretization of the geometric domain and the governing equations. In this work, we use the PISO algorithm [40], an implicit pressure-correction method, to solve this linear equations system. In PISO, the velocity and pressure are calculated by solving linear systems of velocity equations and pressure-correction equations. In this work, we choose the Conjugate Gradient algorithm (CG) [8] to solve the symmetric linear pressure-correction equations, and the Bi-Conjugate Gradient Stabilized algorithm (BiCGSTAB) [9] to solve the non-symmetric linear velocity equations. For the sake of simplicity, the preconditioning in the linear equation solvers is not considered in this work, therefore, the identity matrix is used as the preconditioning matrix.

Table 1
Specification of the Adonis cluster.

Processor	Intel Xeon E5520	Nvidia Tesla C1060
Cores	2 × 4 cores	240 cores
Clock rate	2.27 GHz	602 MHz
Memory size	24 GB	4 GB
Nodes	9	1 GPU/node
Network	Infiniband 40G	

To experiment with heterogeneous GPU-accelerated platforms, we implement the hybrid CG and BICGSTAB solvers. Most linear algebra operations involved in the hybrid linear solvers are implemented using CUSP 0.3, a C++ template library for sparse matrix computations for both CPU and GPU computing systems. According to experimental results, the performance of a processing device executing the CG and BiCGSTAB solvers is almost the same. The BiCGSTAB solver requires almost exactly twice as many linear algebra operations of each type per iteration as the CG solver. Both the complexity and computation time per iteration of the BiCGSTAB solver are almost twice as much as the CG solver, therefore, their speed remains the same. During the simulation of the lid-driven flow, both the CG and BiCGSTAB solvers will be invoked at each time step. Since the two linear equation solvers have almost the same performance, it is reasonable to partition the workload based on performance models built by executing either solver. In this work, we choose to decompose the domain based on the functional performance models built by executing the CG solver.

In this work, the test case is implemented using the OpenFOAM package [2], a flexible and programmable environment for CFD simulation. For this simple CFD test case, the geometric domain is discretized into a regular mesh, using the OpenFOAM utility *blockMesh*. It is worth noting that the proposed partitioning method can also be applied to CFD applications with complex geometries which necessitate the use of unstructured meshes. For simplicity, one-dimensional heterogeneous domain decomposition is used, which is able to handle simple rectangular domains discretized into regular meshes, but the communication is not optimized. In OpenFOAM, the pre-processing and post-processing tasks, such as mesh generation and domain decomposition, involve a significant number of file input and output operations which consume a large amount of time. However, these tasks are handled by stand-alone utilities, such as *blockMesh* and *decomposePar* which we use in this study, so the parallelization and load balancing of such tasks are considered separate problems and not studied in this work.

5.2. Experimental testbed

The experimental testbed is the Adonis cluster, which is specified in Table 1. Each Adonis node consists of two four-core NUMA nodes. Each Adonis node is equipped with a NVIDIA GPU, which exacerbates the processor heterogeneity of the platform. Experiments are conducted on both a single Adonis node and a cluster of Adonis nodes.

5.3. Experimental results

In this work, the GPU and its dedicated CPU core are regarded as a combined processing unit. The data transfer time between the

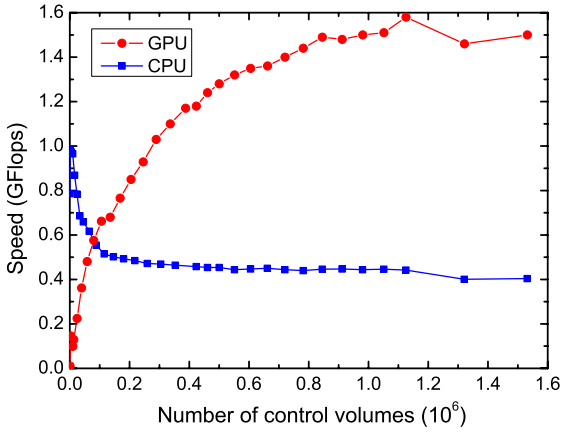


Fig. 1. Speed functions of the CPU cores and GPU processing unit in experiments on a single Anodis node.

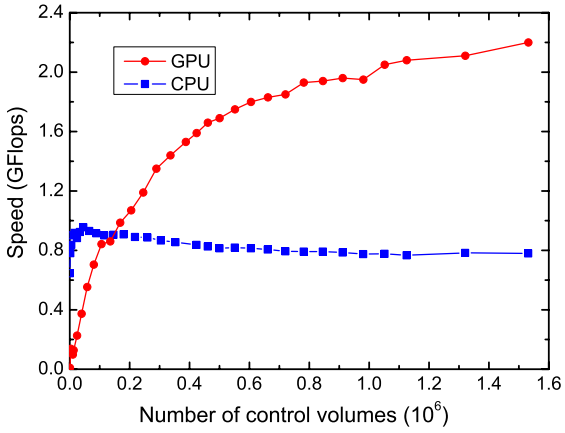


Fig. 2. Speed functions of the CPU cores and GPU processing unit in experiments on the Adonis cluster.

main memory and GPU memory is included in the performance measurement of the GPU processing unit. In the experiment on a single Adonis node, the speeds of the GPU processing unit and other seven CPU cores are measured by executing the hybrid CG solver for a wide range of problem sizes to build their functional performance models. The speed of the processing devices of the same NUMA node are measured simultaneously, taking into account the impact of resource contention on their performance. According to measurements, the speeds of all CPU cores are approximately the same. For the sake of brevity, we only present one FPM for the CPU cores in the figure. Fig. 1 presents the pre-built functional performance models of the GPU processing unit and the CPU cores on a single Adonis node. As we can see, the performance of the GPU processing unit increases quickly firstly and then grows slowly, while the performance of the CPU core drops dramatically firstly and then decreases very slowly.

In order to evaluate the FPM-based decomposition method, we compare the actual speedup in execution time when using the FPM-based heterogeneous decomposition method over the homogeneous decomposition method with the estimated upper bound of speedup in execution time. The execution time consists of the computation time and communication time. Therefore, the speedup in execution time, $S_{exec} = (t^h_{comp} + t^h_{comm})/(t^f_{comp} + t^f_{comm})$, where t_{comp} stands for the computation time, t_{comm} stands for the communication time, h indicates that the time is measured in experiments when the homogeneous decomposition is used, and f indicates that FPM-based heterogeneous decomposition is used. Because the geometric domain is decomposed in one-dimension, the total volume of communication does not depend on whether the domain is decomposed homogeneously or heterogeneously. Therefore, t^f_{comm} can be used as an estimate of t^h_{comm}. By dividing the numerator and denominator by t^f_{comp}, we get that $S_{exec} = (S_{comp} + r_f)/(1 + r_f)$, where $S_{comp} = t^h_{comp}/t^f_{comp}$, which is the speedup in computation time, $r_f = t^f_{comm}/t^f_{comp}$, which is the ratio of the communication time to the computation time. In this work, S_{comp} is calculated based on the pre-built functional performance models. To calculate r_f, we measure the execution time, and the communication time with all computation in the linear solvers removed, which can be regarded as a lower bound of the communication overhead. For a given problem size, after the values of S_{comp} and r_f are calculated, the upper bound of the speedup in execution time on the given heterogeneous platform can then be calculated.

Table 2 presents the actual speedup in execution time when using FPM-based decomposition method over the homogeneous decomposition method. It is worth noting that the speedup that can be achieved depends on the level of the processor heterogeneity of the experiment platform. As shown in the table, the actual speedup in execution time is around 1.22 in experiments on a single Adonis node. Based on the pre-built functional performance models, we can calculate that the speedup in computation time is up to 1.4. Based on experimental results, we can calculate that r_f is around 0.6. Therefore, the upper bound of S_{exec} is 1.25. As we can see, the actual speedup and the estimated upper bound are quite close, which demonstrates that the FPM-based heterogeneous domain decomposition is able to balance the workload on a GPU-accelerated multicore node.

The mesh generation utility used in the experiments, *blockMesh*, can only run on a single node, so the range of problem size with which we can experiment is limited by the memory capacity of a single node. If the largest number of control volumes that can be generated on a single node are distributed evenly to all processing elements of the Adonis cluster, each will receive only a relatively small number of control volumes. In that case, the computing capacity of GPU is barely fulfilled, and the level of performance heterogeneity of CPU and GPU is low, which is of no interest in this study.

In order to experiment with a wide range of problem sizes on the Adonis cluster, only one GPU (with a dedicated CPU core) and seven CPU cores are used in the experiments, each from one Adonis node. Fig. 2 presents the pre-built functional performance models of the GPU processing unit and the CPU core. On the Adonis cluster, the speedup in execution time is around 1.1. Using the same method described above, we can calculate that the speedup in computation time S_{comp} is up to 1.25. Based on experimental results, the ratio of the communication time to the computation time r_f is around 0.67. Therefore, we can estimate that the speedup in execution time S_{exec} is up to 1.15. As we can see, the actual speedup and the estimated upper bound of the actual speedup are reasonably close, which demonstrates the effectiveness of FPM-based heterogeneous domain decomposition.

6. Conclusion

In this work, we aim at the maximum performance of OpenFOAM-based CFD applications on GPU-accelerated HPC platforms. To this end, we propose a hybrid solution method for CFD applications so that they can be run on GPU-accelerated parallel platforms, exploiting the computational power of heterogeneous processing devices. Furthermore, we propose a FPM-based heterogeneous domain decomposition method for balancing the workload and optimizing the communication overhead of CFD

Table 2
The execution time of the CFD test case when using different decomposition methods, and the speedup in execution time when using the FPM-based heterogeneous decomposition method over the homogeneous decomposition method.

Number of CVs ($\times 10^6$)	Adonis node			Adonis cluster		
	t^h_{exec} (s)	t^f_{exec} (s)	S_{exec}	t^h_{exec} (s)	t^f_{exec} (s)	S_{exec}
2.82	282	232	1.22	120	110	1.09
4.84	485	399	1.21	215	196	1.10
6.76	675	559	1.21	338	303	1.12
9.00	901	732	1.23	452	417	1.08
10.6	1061	877	1.21	532	490	1.09

applications on target platforms. The proposed methods are evaluated with a typical CFD application, namely the numerical simulation of lid-driven cavity flow, on both a hybrid server and a hybrid cluster. Experimental results demonstrate that CFD applications designed in the proposed method can scale well on GPU-accelerated parallel platforms, and the FPM-based heterogeneous domain decomposition can balance the workload and deliver good performance.

Acknowledgments

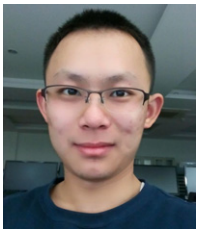
This research was conducted with the financial support of NSFC award nos. 61403402, 61374185 and 71303252. Experiments were carried out on platforms provided by the Grid5000 developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). A lot of thanks should be given to referees and editors, their valuable comments greatly improved the quality of the manuscript.

References

[1] J.H. Ferziger, M. Peric, Computational Methods for Fluid Dynamics, third ed., Springer, 2002.
[2] H.G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, Comput. Phys. 12 (6) (1998) 620–631.
[3] A. Lastovetsky, J. Twamley, Towards a realistic performance model for networks of heterogeneous computers, in: Proceedings of IFIP TC5 Workshop, World Computer Congress, August 22–27 2004, Toulouse, France, High Performance Computational Science and Engineering, Springer, 2005, pp. 39–58. Springer.
[4] A. Lastovetsky, R. Reddy, R. Higgins, Building the functional performance model of a processor, in: Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC 2006), ACM, Dijon, France, 2006, ACM.
[5] A. Lastovetsky, R. Reddy, Data partitioning with a functional performance model of heterogeneous processors, Int. J. High Perform. Comput. 21 (2007) 76–90.
[6] G. Karypis, K. Schloegel, ParMETIS: Parallel graph partitioning and sparse matrix ordering library, 2013. http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf.
[7] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering, Parallel Comput. 34 (6–8) (2008) 318–331.
[8] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, MD, USA, 1996.
[9] H.A. van der Vorst, 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) (1992) 631–644.
[10] A. Logg, K.-A. Mardal, G.N. Wells, et al., Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012.
[11] F. Witherden, A. Farrington, P. Vincent, PyFR: An open source framework for solving advectiondiffusion type problems on streaming architectures using the flux reconstruction approach, Comput. Phys. Comm. 185 (11) (2014) 3028–3040.
[12] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips, Gpu computing, Proc. IEEE 96 (5) (2008) 879–899.
[13] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform, in: 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2010, pp. 583–592.
[14] P. Niksiar, A. Ashrafizadeh, M. Shams, A. Madani, Implementation of a GPU-based CFD code, in: 2014 International Conference on Computational Science and Computational Intelligence, CSCI, Vol. 1, 2014, pp. 84–89.
[15] R. Strzodka, J. Cohen, S. Posey, GPU-accelerated algebraic multigrid for applied CFD, Procedia Eng. 61 (2013) 381–387.
[16] K.E. Niemeyer, C.-J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J. Supercomput. 67 (2) (2014) 528–564.
[17] W. Cao, C.-f. Xu, Z.-h. Wang, L. Yao, H.-y. Liu, CPU/GPU computing for a multi-block structured grid based high-order flow solver on a large heterogeneous system, Clust. Comput. 17 (2) (2014) 255–270.
[18] A. AlOnazi, Design and optimization of openFOAM-based CFD applications for modern hybrid and heterogeneous HPC platforms, 2013.
[19] S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (2009) 65–76.
[20] S. Hong, H. Kim, An integrated GPU power and performance model, in: Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA'10, ACM, New York, NY, USA, 2010, pp. 280–289.
[21] D. Clarke, Z. Zhong, V. Rychkov, A. Lastovetsky, FuPerMod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous HPC platforms, in: PaCT-2013, in: LNCS, vol. 7979, Springer, 2013, pp. 182–196.
[22] M. Fatica, Accelerating Linpack with CUDA on heterogenous clusters, in: GPGPU-2, ACM, 2009, pp. 46–51.
[23] C. Yang, et al. Adaptive optimization for petascale heterogeneous CPU/GPU computing, in: Cluster'10, 2010, pp. 19–28.
[24] Y. Ogata, et al. An efficient, model-based CPU–GPU heterogeneous FFT library, in: IPDPS 2008, 2008, pp. 1–10.
[25] J. Quintin, F. Wagner, Hierarchical work-stealing, in: Euro-Par 2010 Parallel Processing, 2010, pp. 217–229.
[26] M.D. Linderman, J.D. Collins, H. Wang, et al., Merge: a programming model for heterogeneous multi-core systems, SIGPLAN Not. 43 (2008) 287–296.
[27] G. Quintana-Ortí, et al., Solving dense linear systems on platforms with multiple hardware accelerators, SIGPLAN Not. 44 (2009) 121–130.
[28] X. Zhu, X. Qin, M. Qiu, QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, IEEE Trans. Comput. 60 (6) (2011) 800–812.
[29] X. Zhu, C. He, K. Li, X. Qin, Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters, J. Parallel Distrib. Comput. 72 (6) (2012) 751–763.
[30] X. Zhu, C. Chen, L. Yang, Y. Xiang, ANGEL: Agent-based scheduling for real-time tasks in virtualized clouds, IEEE Trans. Comput. PP (99) (2015) 1–1.
[31] A. Lastovetsky, R. Reddy, Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models, in: HeteroPa'09, in: Lecture Notes in Computer Science, vol. 6043, Springer, 2010, pp. 91–101.
[32] C. Walshaw, M. Cross, Multilevel mesh partitioning for heterogeneous communication networks, Future Gener. Comput. Syst. 17 (5) (2001) 601–623.
[33] E. Aubanel, X. Wu, Incorporating latency in heterogeneous graph partitioning, in: IPDPS 2007, 2007, pp. 1–8.
[34] U. Catalyurek, E. Boman, K. Devine, et al. Hypergraph-based dynamic load balancing for adaptive scientific computations, in: IPDPS 2007, 2007, pp. 1–11.
[35] Intel MKL Library. https://software.intel.com/en-us/intel-mkl.
[36] CUDA programming language, 2015. https://developer.nvidia.com/cuda-zone.
[37] CUBLAS Library, 2015. https://developer.nvidia.com/cublas.
[38] Z. Zhong, V. Rychkov, A. Lastovetsky, Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications, in: Cluster, 2012, pp. 191–199.
[39] U. Ghia, K.N. Ghia, C.T. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, J. Comput. Phys. 48 (1982) 387–411.
[40] R.I. Issa, Solution of the implicitly discretised fluid flow equations by operator-splitting, J. Comput. Phys. 62 (1) (1986) 40–65.

Xiaocheng Liu is a lecture on parallel and distributed systems. He is from College of Information Systems and Management, National University of Defense Technology (PR China).

Ziming Zhong is a researcher from Complex Aviation Systems Simulation Lab, Beijing. His research interests include performance modeling and parallel computing on GPU-accelerated multicore platforms.



Kai Xu is a Ph.D. student on high performance computing. He is from College of Information Systems and Management, National University of Defense Technology (PR China).