Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 2053156, 15 pages
https://doi.org/10.1155/2019/2053156

Research Article

A Multi-GPU Parallel Algorithm in Hypersonic Flow Computations

Jianqi Lai, Hua Li, Zhengyu Tian, and Ye Zhang

College of Aerospace Science and Engineering, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Jianqi Lai; laijianqi_kd@nudt.edu.cn

Received 2 December 2018; Revised 28 January 2019; Accepted 25 February 2019; Published 17 March 2019

Academic Editor: Xesús Nogueira

Computational fluid dynamics (CFD) plays an important role in the optimal design of aircraft and the analysis of complex flow mechanisms in the aerospace domain. The graphics processing unit (GPU) has a strong floating-point operation capability and a high memory bandwidth in data parallelism, which brings great opportunities for CFD. A cell-centred finite volume method is applied to solve three-dimensional compressible Navier–Stokes equations on structured meshes with an upwind AUSM+UP numerical scheme for space discretization, and four-stage Runge–Kutta method is used for time discretization. Compute unified device architecture (CUDA) is used as a parallel computing platform and programming model for GPUs, which reduces the complexity of programming. The main purpose of this paper is to design an extremely efficient multi-GPU parallel algorithm based on MPI+CUDA to study the hypersonic flow characteristics. Solutions of hypersonic flow over an aerospace plane model are provided at different Mach numbers. The agreement between numerical computations and experimental measurements is favourable. Acceleration performance of the parallel platform is studied with single GPU, two GPUs, and four GPUs. For single GPU implementation, the speedup reaches 63 for the coarser mesh and 78 for the finest mesh. GPUs are better suited for compute-intensive tasks than traditional CPUs. For multi-GPU parallelization, the speedup of four GPUs reaches 77 for the coarser mesh and 147 for the finest mesh; this is far greater than the acceleration achieved by single GPU and two GPUs. It is prospective to apply the multi-GPU parallel algorithm to hypersonic flow computations.

1. Introduction

Generally, real flows have different levels of compressibility. In comparison with incompressible flow, the density of compressible flow varies, which increases the complexity of the solution of governing equations. According to the Mach number, compressible flow can be divided into subsonic flow ($Ma<0.8$), transonic flow ($Ma=0.8$–1.2), supersonic flow ($Ma=1.2$–5.0), and hypersonic flow (Ma>5.0) [1]. Hypersonic flight has been with us since 1963; North American X-15 achieved a successful flight at Mach number 6.7. Over the last six decades, with the development of scramjet engines, high-speed aircrafts, hypersonic missiles, and hypersonic reentry vehicles, research on hypersonic flow theory has made considerable progress and attracts much attention of scientists [2, 3]. In hypersonic flow, some of the preceding physical phenomena become important. For flow over a hypersonic body, a strong shock wave with high temperature is formed. The flow field between the shock wave and the body is defined as shock layer, and the distance of the shock layer can be small. At a high temperature, the vibrational energy may be excited, and chemical reactions can occur. Accurate simulations of hypersonic flow need a more sophisticated meshing, which enlarges the computational work remarkably [4]. What is more, it is attractive for engineers and researchers to obtain accurate numerical results as soon as possible. The graphics processing unit (GPU) has exhibited significant achievements in the accelerated solution of hypersonic flow problems [5, 6].

With the rapid development of computational fluid dynamics (CFD) and computer technology, CFD plays a vital role in the optimal design of aircraft and the analysis of complex flow mechanisms [7]. At present, CFD is used to simulate problems ranging from molecular level to macroscopic magnitude, from simple two-dimensional configurations to three-dimensional complex real aircrafts, and from the simulation of subsonic flow problems to hypersonic flow problems [8, 9]. A large number of applications of CFD can reduce the development costs and provide technical support for the research on aircrafts. High Reynolds number flows, which are ubiquitous in real flows, such as turbulence, vortices, separation, aeroacoustics, and other multiscale complex flow problems, require intensive computational meshes [10]. When modeling real gas flow simulations, chemical reaction models that contain multiple components significantly increase the computational demands [11]. High-fidelity simulations are inaccessible to engineering activities and scientific research because of large-scale computing requirements. To reduce the computational time and obtain accurate numerical results quickly, the extensive calculations can only be completed by parallel computing. In recent years, the progress of the central processing unit (CPU) has shifted into a bottleneck because of the limitations in power consumption and heat dissipation prevention, and adding CPU cores cannot improve the overall performance of CPU-based parallel clusters.

In comparison with the Intel CPU, the GPU has strong floating-point operations and a high memory bandwidth in data parallelism, as illustrated in Figures 1 and 2 [12]. Traditionally, a GPU is limited to graphics rendering. In 2007, NVIDIA introduced the compute unified device architecture (CUDA), which is a parallel computing platform and programming model for GPU that allows developers to use C/C++ as a high-level language and reduces the complexity of programming. The emergence of CUDA as a programming model marked the beginning of the widespread use of GPUs in general-purpose computing. Currently, GPUs are widely used in the area of scientific computing, such as molecular dynamics (MD), direct simulation Monte Carlo (DSMC), CFD, weather forecast (WF), and artificial intelligence (AI) [13–17]. The development of GPU-based parallelization brings increased opportunities and challenges for high-performance computing. In the field of CFD, GPU parallelization has achieved numerous achievements. Brandvik et al. [18] studied the solution of three-dimensional Euler equations on an NVIDIA 8800GTX GPU with explicit time-stepping schemes, and the speedup reached 16 times. Khajeh–Saeed et al. [19] accomplished direct numerical simulation (DNS) of turbulence using GPU accelerated supercomputers, which demonstrated that scientific problems could benefit significantly from advanced hardware. Wang et al. [20] discussed DNS and Large Eddy Simulation (LES) on a turbulent wall-bounded flow using Lattice Boltzmann method and multiple GPUs, and the acceleration performance has been discussed. Emelyanov et al. [21] discussed the popular CFD benchmark solution of the flow over a smooth flat plate on a GPU with various meshes, and the speedup reached more than 46 times. Zhang et al. [22] performed an implicit meshless method for compressible flow on an NVIDIA GTX TITAN GPU, and the solution agrees well with experimental results.

The main purpose of this paper is to design an extremely efficient multi-GPU parallel algorithm to study hypersonic flow characteristics of an aerospace model, and the acceleration performance of the algorithm has been analysed to apply it to large-scale scientific calculation.

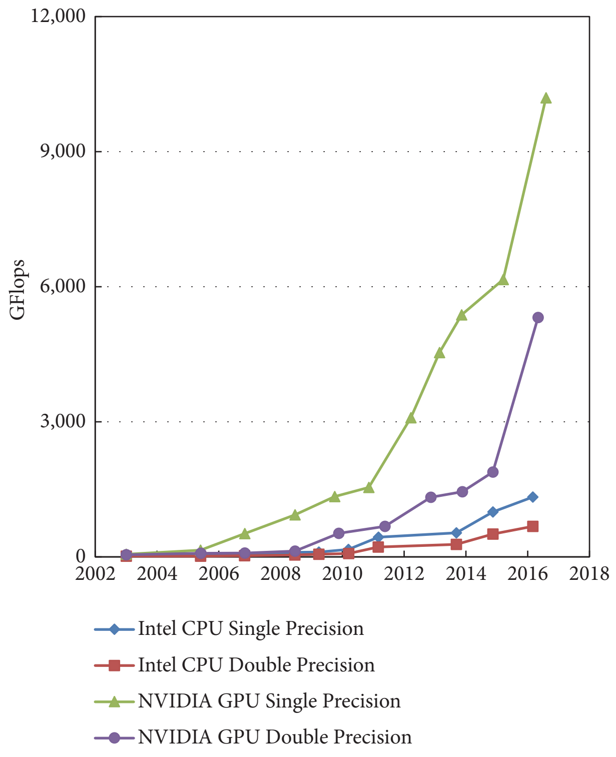


Figure 1: Floating-point operations per second for the CPU and GPU [12].

The remainder of this paper is organized as follows. Section 2 presents the governing equations and numerical schemes. Section 3 introduces CUDA and the GPU parallel algorithm for CFD, and then multi-GPU parallel algorithm based on MPI+CUDA is established. Section 4 describes the physical model and mesh generation of an aerospace plane in sufficient detail. Section 5 shows the computed results compared with experimental measurements and analyses the performance of multi-GPU parallel platform. Section 6 provides the conclusions of this work.

2. Governing Equations and Numerical Schemes

2.1. Governing Equations. In the field of CFD, without regard to volume sources due to body forces and volumetric, three-dimensional compressible Navier–Stokes (NS) equations can be written in integral form as follows [23]:

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{W}\,d\Omega+\oint_{\partial\Omega}\left(\vec{F}_c-\vec{F}_v\right)dS=0 \tag{1}$$

where $\vec{W}$ is the vector of conservative variables, which contains three dimensions of five components; $\vec{F}_c$ is the vector of convective fluxes; $\vec{F}_v$ presents the vector of viscous fluxes; Ω is the control volume; and $\partial\Omega$ is the cell surface.

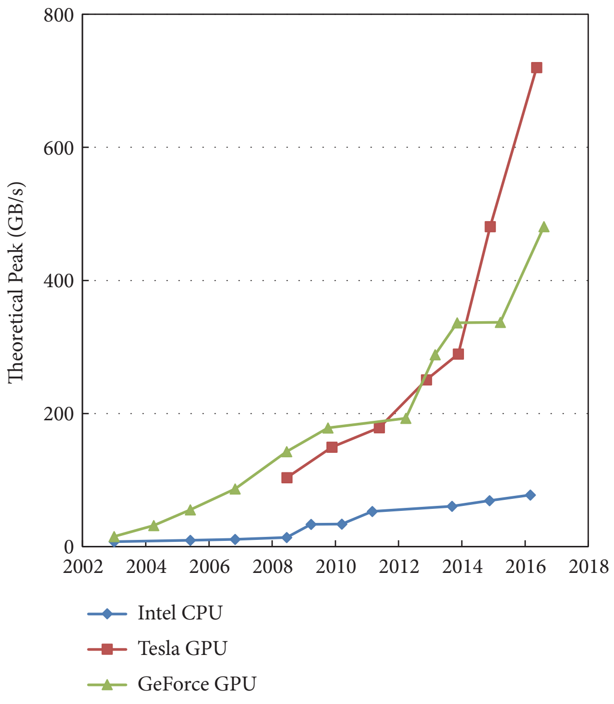


FIGURE 2: Memory bandwidth for the CPU and GPU [12].

$$\vec{W} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix}, \qquad \vec{F}_c = \begin{bmatrix} \rho V \\ \rho u V + n_x p \\ \rho v V + n_y p \\ \rho w V + n_z p \\ \rho H V \end{bmatrix} \tag{2}$$

$$\vec{F}_v = \begin{bmatrix} 0 \\ n_x \tau_{xx} + n_y \tau_{xy} + n_z \tau_{xz} \\ n_x \tau_{yx} + n_y \tau_{yy} + n_z \tau_{yz} \\ n_x \tau_{zx} + n_y \tau_{zy} + n_z \tau_{zz} \\ n_x \Theta_x + n_y \Theta_y + n_z \Theta_z \end{bmatrix} \tag{3}$$

$$\begin{aligned} \Theta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} - q_x \\ \Theta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} - q_y \\ \Theta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} - q_z \end{aligned} \tag{4}$$

In the above equations, ρ is the density, $\vec{V} = (u, v, w)$ are the local Cartesian velocity components, p is the static pressure, T is the static temperature, $\vec{n} = (n_x, n_y, n_z)$ are the unit normal vector of the cell surface, V is the scalar product of the velocity vector and the unit normal vector, H is the total enthalpy, and E is the total energy, which can be related to the static pressure, p, through the equation of state:

$$p = \rho R T \tag{5}$$

$$E = \frac{p}{(\gamma - 1)\rho} + \frac{u^2 + v^2 + w^2}{2} \tag{6}$$

where γ is the ratio of specific heat capacities which is taken as 1.4 for calorically perfect gas and R is the specific gas constant.

For the vector of viscous fluxes, the components of viscous stress τ_{ij} and heat flux vector q_i are expressed as follows:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) \tag{7}$$

$$q_i = -k \frac{\partial T}{\partial x_i} \tag{8}$$

where μ and k are the viscosity coefficient and thermal conductivity coefficient, respectively, which can be obtained by Sutherland's law and the definition of Prandtl number, respectively, shown as follows:

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0} \right)^{1.5} \left(\frac{T_0 + T_s}{T + T_0} \right) \tag{9}$$

$$Pr = \frac{\mu c_p}{k} \tag{10}$$

where T_0 is a constant equal to 273.11 K, $\mu_0 = 1.716 \times 10^{-5}$ kg/(m s), $S_0 = 110.56$ K for air, c_p is the specific heat at constant pressure, and Pr is the Prandtl number, which equivalents to 0.72.

2.2. Nondimensionlisation of Governing Equations. In order to solve the NS equation more conveniently due to its complexity, nondimensionlisation of these equations is based on the freestream values: density (ρ_∞), speed of sound (c_∞), static temperature (T_∞), and a reference length (L_0).

$$\begin{aligned} u^* &= \frac{u}{c_\infty}, \\ v* &= \frac{v}{c_\infty}, \\ w* &= \frac{w}{c_\infty}, \\ \rho^* &= \frac{\rho}{\rho_\infty}, \\ T* &= \frac{T}{T_\infty}, \\ p^* &= \frac{p}{\rho_\infty c_\infty^2} \end{aligned} \tag{11}$$

$$x_i^* = \frac{x_i}{L_0} \tag{12}$$

Nondimensionlisation of governing equations is as follows [24]:

$$\gamma p^* = \rho^* T^* \tag{13}$$

$$E^* = \frac{p^*}{(\gamma - 1)\rho^*} + \frac{(u^*)^2 + (v^*)^2 + (w^*)^2}{2} \tag{14}$$

$$\tau_{ij}{}^* = \frac{Ma_\infty}{Re_\infty}\left[\mu^*\left(\frac{\partial u_i{}^*}{\partial x_j{}^*} + \frac{\partial u_j{}^*}{\partial x_i{}^*} - \frac{2}{3}\frac{\partial u_k{}^*}{\partial x_k{}^*}\delta_{ij}\right)\right] \tag{15}$$

$$Re_\infty = \frac{\rho_\infty V_\infty L_0}{\mu_\infty} \tag{16}$$

$$q_i{}^* = -\frac{Ma_\infty}{Re_\infty}\frac{\mu^*}{Pr(\gamma - 1)}\frac{\partial T^*}{\partial x_i{}^*} \tag{17}$$

$$\mu^* = (T^*)^{3/2}\frac{1 + S}{(T^* + S)}, \quad S = \frac{110.4\text{K}}{T_\infty} \tag{18}$$

2.3. Numerical Schemes. Structured or unstructured meshes are used to discretize the governing equations. An outstanding advantage of structured meshes is that the indices I, J, K represent the computational space, since it directly corresponds to how the flow variables are stored in the computer memory. This property allows it to implement computational algorithms more efficiently in numerical simulation. Meanwhile, CFD codes are easily performed on GPUs due to their regular data structures which are convenient for parallel computing. However, the generation of structured meshes is more complicated than that of unstructured meshes for complex geometries.

The cell-centred finite volume method based on structured meshes is used in the spatial discretization of governing equations. For the control volume $\Omega_{I,J,K}$, the discrete form of formula (1) can be written in the following form:

$$\Omega_{I,J,K}\frac{d\overrightarrow{W}_{I,J,K}}{dt} = -\left[\sum_{m=1}^{N_F}\left(\overrightarrow{F_c} - \overrightarrow{F_v}\right)\Delta S_m\right] \tag{19}$$

The term in square brackets on the right-hand side of formula (19) is called the residual, which is denoted by $\overrightarrow{R}_{I,J,K}$

$$\Omega_{I,J,K}\frac{d\overrightarrow{W}_{I,J,K}}{dt} = -\overrightarrow{R}_{I,J,K} \tag{20}$$

The convective fluxes are computed by an upwind AUSM+UP numerical scheme [25], which has a high resolution and computational efficiency for all speeds. The underlying idea of the approach is that the convective fluxes are decomposed into two parts (the convective and pressure parts) based on the mass flow function

$$\begin{aligned}\left(\overrightarrow{F_c}\right)_{I+1/2} &= \left(\overrightarrow{F}^{(c)}\right)_{I+1/2} + \left(\overrightarrow{p}\right)_{I+1/2}\\ &= (\dot{m})_{I+1/2}\begin{cases}\left(\overrightarrow{\psi}\right)_L, & (\dot{m})_{I+1/2} \geq 0\\ \left(\overrightarrow{\psi}\right)_R, & (\dot{m})_{I+1/2} < 0\end{cases}\\ &\quad + \left(\overrightarrow{p}\right)_{I+1/2}\end{aligned} \tag{21}$$

where subscript $I + 1/2$ refers to the cell face; subscripts L and R refer to the control volume on the left and right edges of the cell face; $\overrightarrow{F}^{(c)}$ denotes the vector of the convective part; $\overrightarrow{p}$ denotes the vector of pressure; $\dot{m}$ denotes the mass flux; and $\overrightarrow{\psi}$ is a vector quantity that is convected by $\dot{m}$, in which

$$(\dot{m})_{I+1/2} = (c)_{I+1/2}(Ma)_{I+1/2}\begin{cases}\rho_L, & (Ma)_{I+1/2} \geq 0\\ \rho_R, & (Ma)_{I+1/2} < 0\end{cases} \tag{22}$$

$$\left(\overrightarrow{\psi}\right)_{L/R} = \begin{cases}[1, u, v, w, H]_L^{\mathrm{T}}, & (\dot{m})_{I+1/2} \geq 0\\ [1, u, v, w, H]_R^{\mathrm{T}}, & (\dot{m})_{I+1/2} < 0\end{cases} \tag{23}$$

In this paper, second-order accuracy is achieved through the monotone upstream-centred schemes for conservation laws (MUSCL) [26] with the Van Leer's limiter, which can prevent the generation of oscillations and spurious solutions in regions with a large gradient.

The central scheme is selected to solve the viscous fluxes due to its elliptic mathematical properties. The values U at the cell face result from

$$U_{I+1/2} = \frac{1}{2}\left(U_I + U_{I+1}\right) \tag{24}$$

For the temporal discretization of formula (20), the method of lines is used, such that space and time integration can be handled separately. Space integration, which consists of the solution of convective fluxes and viscous fluxes, is introduced hereinbefore. The four-stage Runge–Kutta time-stepping method [23, 27] is implemented for the temporal discretization, in which

$$\begin{aligned}\overrightarrow{W}_{I,J,K}^{(0)} &= \overrightarrow{W}_{I,J,K}^{(n)}\\ \overrightarrow{W}_{I,J,K}^{(1)} &= \overrightarrow{W}_{I,J,K}^{(0)} - \alpha_1\frac{\Delta t_{I,J,K}}{\Omega_{I,J,K}}\overrightarrow{R}_{I,J,K}^{(0)}\\ \overrightarrow{W}_{I,J,K}^{(2)} &= \overrightarrow{W}_{I,J,K}^{(0)} - \alpha_2\frac{\Delta t_{I,J,K}}{\Omega_{I,J,K}}\overrightarrow{R}_{I,J,K}^{(1)}\\ \overrightarrow{W}_{I,J,K}^{(3)} &= \overrightarrow{W}_{I,J,K}^{(0)} - \alpha_3\frac{\Delta t_{I,J,K}}{\Omega_{I,J,K}}\overrightarrow{R}_{I,J,K}^{(2)}\\ \overrightarrow{W}_{I,J,K}^{(4)} &= \overrightarrow{W}_{I,J,K}^{(0)} - \alpha_4\frac{\Delta t_{I,J,K}}{\Omega_{I,J,K}}\overrightarrow{R}_{I,J,K}^{(3)}\\ \overrightarrow{W}_{I,J,K}^{(\mathrm{n+1})} &= \overrightarrow{W}_{I,J,K}^{(4)}\end{aligned} \tag{25}$$

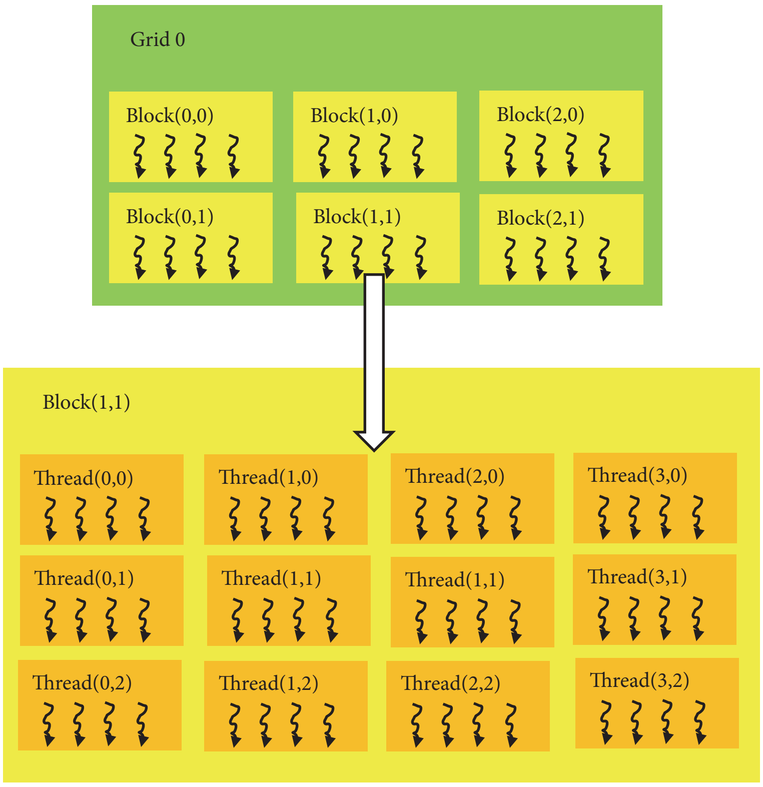


Figure 3: Grid–block–thread.

This scheme is second-order accurate in time, where $\Delta t_{I,J,K}$ is the time step of control volume $\Omega_{I,J,K}$ and α_k ($\alpha_1 = 0.1084$, $\alpha_2 = 0.2602$, $\alpha_3 = 0.5052$, $\alpha_4 = 1.0000$) represents the stage coefficients. The multistage scheme is good at data parallelism, which is of considerable significance to parallel computing. Furthermore, this method reduces the storage expense because only the initial conservative variables and the latest calculated residual value are stored in memory. The implicit residual smoothing method is used to accelerate the solution of the governing equations.

3. CUDA and GPU Parallel Algorithm for CFD

3.1. CUDA Overview and GPU Architecture. CUDA is a general-purpose parallel computing platform and programming model for researchers to perform codes on GPU flexibly to solve many complex scientific problems [9, 12, 28]. In CUDA, a C program is extended to the execution of kernels, which are executed by a large number of threads. Individual threads are grouped into thread blocks, and the block size is no more than 1,024 for current devices. Thread blocks are organized into a grid, and the number of thread blocks in a grid is usually decided by the size of the data being processed, which ensures that a thread loads the computation of a mesh. Figure 3 shows the relationship among thread, block, and grid.

The GPU architecture contains different types of memory to access data, such as global, constant, texture, shared, and local memories and registers [12, 21]. Figure 4 shows the memory hierarchy of a GPU. At a low level, local memory and registers are private to the threads. The input or intermediate output variables in the threads will be stored in registers or local memory. At a high level, each thread block has shared memory, and all threads in the same thread block can cooperate with one another. At an even higher level, the texture and constant memories are used to store read-only parameters that need to be frequently accessed. The global memory is available to CPU and GPU and enables data transfer between them. The memory hierarchy of a GPU guarantees that high-performance computing can be achieved.

The GPU architecture is built based on an array of streaming multiprocessors (SMs). A multiprocessor is designed to execute hundreds of threads concurrently to help a GPU hide the latency of the memory access, which is called Single-Instruction, Multiple-Thread (SIMT). When the host invokes a kernel function, the thread blocks of the grid are distributed to SMs. For a SM, thread blocks are executed in turn: once old thread blocks terminate, new thread blocks are launched immediately. Threads within the same block are guaranteed to run on the same SM concurrently, and interthread communication is accomplished through shared memory.

3.2. GPU Parallel Algorithm for CFD. Figure 5 shows the GPU parallel algorithm corresponding to the solution of CFD. The solid boxes represent the general cooperation of CPU and GPU, and the dashed box represents the solution of governing

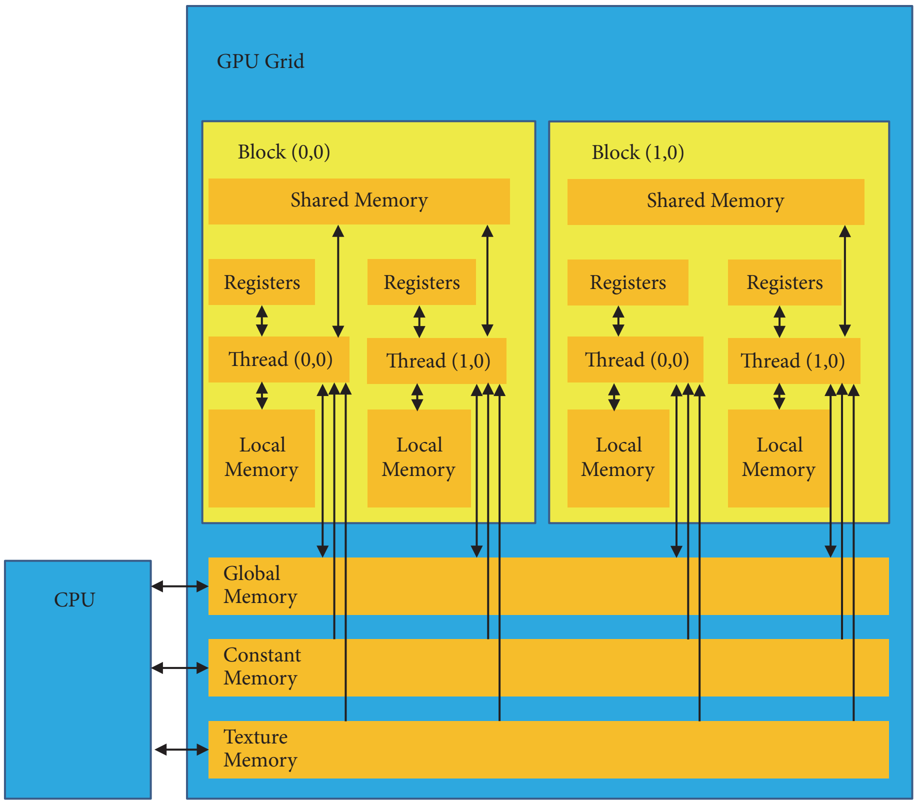


Figure 4: The memory hierarchy of a GPU.

equations implemented on GPU. The GPU parallel algorithm follows seven steps: device setting, memory allocation, data transfer from host to device, kernel execution, data transfer from device to host, freeing memory, and device resetting. For the solution of governing equations, each time step of the computation contains a series of kernel functions, including the processing of boundary conditions, the calculation of fluxes of cell face, and the update of primitive variables. The exchanges of primitive variables and their gradients between GPUs are essential to calculate the fluxes of cell face accurately when running the GPU-based CFD codes on multi-GPU parallel cluster.

3.3. Multi-GPU Parallelization Based on MPI+CUDA. The Message Passing Interface (MPI) is widely used on shared and distributed memory machines to implement large-scale calculations on multi-GPU parallel cluster [5, 29, 30]. Figure 6 shows the multi-GPU programming model based on MPI+CUDA. At the beginning of the calculation, the *MPI_Init* function is called to enter the MPI environment. The computing tasks are then evenly distributed to each GPU to achieve load balance by one-dimensional domain decomposition method [31]. At the end of the calculation, the *MPI_Finalize* function is called to exit the MPI environment. Here, each node consists of one Intel Xeon E5-2670 CPU with eight cores and four NVIDIA GTX 1070 GPUs.

In a multi-GPU parallel cluster, the ghost and singularity data will be exchanged between GPUs. For the current devices, data transfer cannot be completed directly between GPUs, and CPU plays a role as medium. Figure 7 shows data transfer process between GPUs. The function *cudaMcemcpy-HostToDevice* is called to transfer data from the GPU to the relevant CPU through the PCI-e bus. After fulfilling data transfer between CPUs, the function *cudaMcemcpyDevice-ToHost* is called to transfer the data from CPU to the target GPU. The latest device introduced by NVIDIA Corporation is Tesla V100, which provides an NVLink bus technique [32] to achieve communication between GPUs directly.

4. Physical Model and Mesh Generation

4.1. Physical Model. Figure 8 shows the physical model of an aerospace plane. Reference [33] provides a wide variety of experimental data sets for this model, and these experiments are carried out in a hypersonic wind tunnel. The aerospace plane is significantly important for future space warfare and transportation, which are drawing increasing interest. In this paper, the hypersonic flow over the aerospace plane model has been studied on the multi-GPU parallel platform. Orthographic views and the main dimensions (given in mm) of the model are presented in Figure 9, which exactly corresponds to the experimental model.

Two Mach numbers, namely, 8.02 and 10.02, are chosen to analyse the hypersonic flow characteristics of aerospace plane. The slip angle for all cases is 0^0, and the angles of attack are 0^0 and 5^0 for Mach number 8.02, and 0^0 and 10^0 for Mach

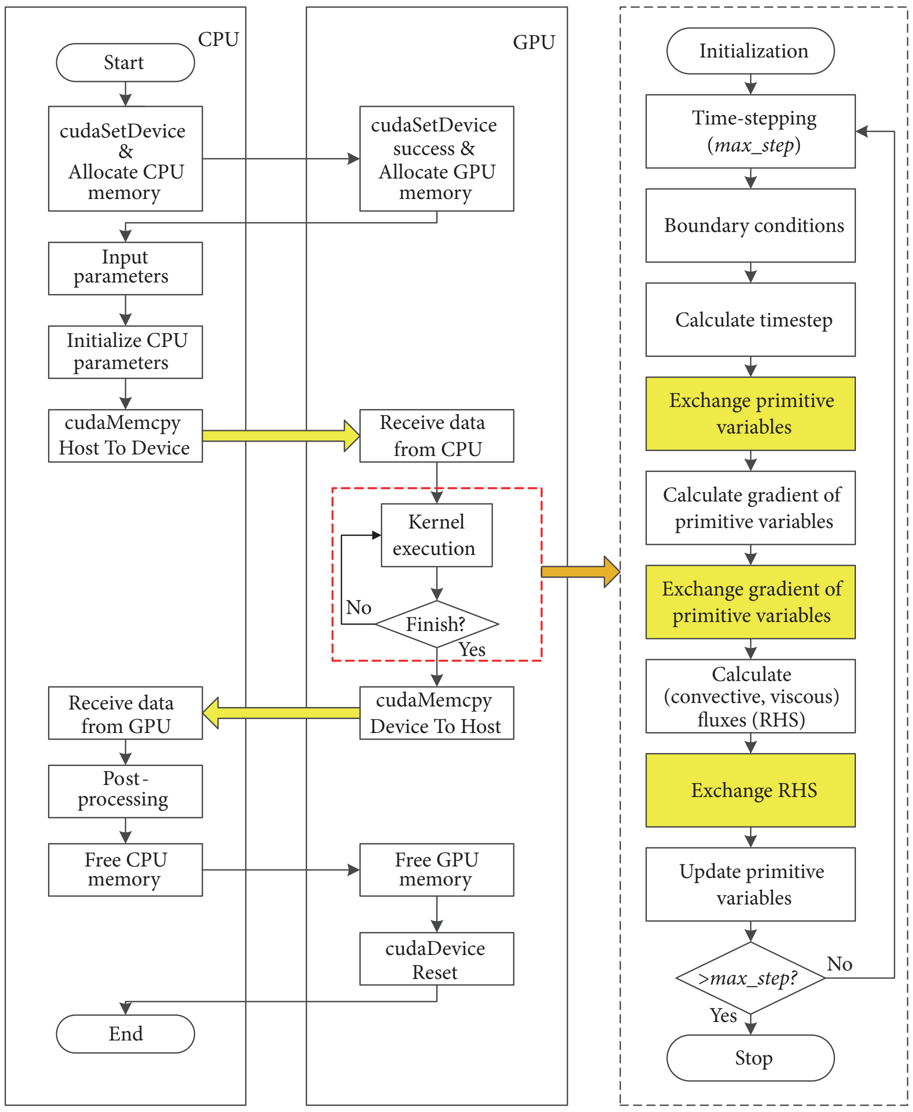

CPU
Start
cudaSetDevice & Allocate CPU memory
Input parameters
Initialize CPU parameters
cudaMemcpy Host To Device
Receive data from GPU
Post-processing
Free CPU memory
End
GPU
cudaSetDevice success & Allocate GPU memory
Receive data from CPU
Kernel execution
No
Finish?
Yes
cudaMemcpy Device To Host
Free GPU memory
cudaDevice Reset
Initialization
Time-stepping (max_step)
Boundary conditions
Calculate timestep
Exchange primitive variables
Calculate gradient of primitive variables
Exchange gradient of primitive variables
Calculate (convective, viscous) fluxes (RHS)
Exchange RHS
Update primitive variables
>max_step?
No
Yes
Stop

FIGURE 5: GPU parallel algorithm corresponding to the solution of CFD.

number 10.02. The temperature of isothermal wall T_w is set to 300 K. Table 1 shows the flow conditions for this study. The freestream values can be obtained as follows:

$$T_\infty = \frac{T_{t,0}}{\left(1 + \left((\gamma - 1)/2\right) Ma_\infty{}^2\right)} \quad (26)$$

$$\mu_\infty = \mu_0 \left(\frac{T_\infty}{T_0}\right)^{1.5} \left(\frac{T_0 + T_s}{T_\infty + T_0}\right) \quad (27)$$

$$\rho_\infty = \frac{Re_\infty \mu_\infty}{Ma_\infty \sqrt{\gamma R T_\infty}} \quad (28)$$

$$p_\infty = \rho_\infty R T_\infty \quad (29)$$

4.2. Mesh Generation. The commercial software Pointwise is used to generate the multiblock structured mesh. Only half of the flow field is simulated because of its symmetry with no sideslip. The mesh is refined near the walls and at the corners, and the magnitude of the grid height of the first layer is 1e-6 m to capture the characteristics of the boundary layer accurately. The mesh in the boundary layer (see the orange part in Figures 10 and 11) is generated separately from the main flow to ensure the quality of the grid in this region. Meanwhile, this method of mesh generation has strong flexibility. The mesh has 1.046×10^7 nodes and 1.004×10^7 hexahedra. Figures 10 and 11 show the computational mesh of symmetry plane and base plane, respectively. Figure 12 shows the boundary conditions, which are marked in different colours. Here, four types of boundary conditions are employed: wall, symmetry, inflow, and outflow.

5. Results and Discussion

5.1. Multi-GPU Parallel Platform. In this paper, we use CUDA version 8.0, Visual Studio 2013 for C code and MPICH2 1.4.1 for MPI communication. All simulations are performed

Table 1: The flow conditions for the aerospace plane.

Case	Ma_∞	Re_∞ [1/m]	$P_{t,0}$ [MPa]	$T_{t,0}$ [K]	α [^{0}]
Case 1	8.02	1.34×10^7	6.0	740	0
Case 2	8.02	1.34×10^7	6.0	740	5
Case 3	10.02	2.20×10^7	6.9	1,457	0
Case 4	10.02	2.20×10^7	6.9	1,457	10

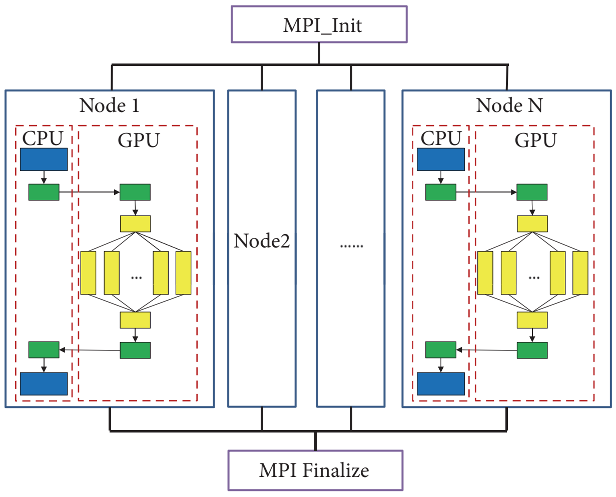


Figure 6: Multi-GPU programming model based on MPI+CUDA.

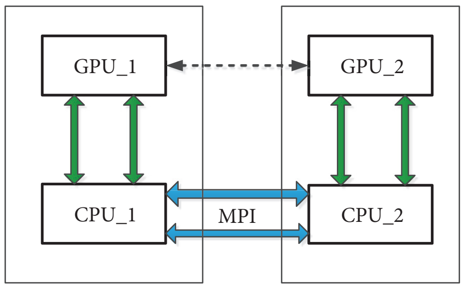


Figure 7: Data transfer process between GPUs.

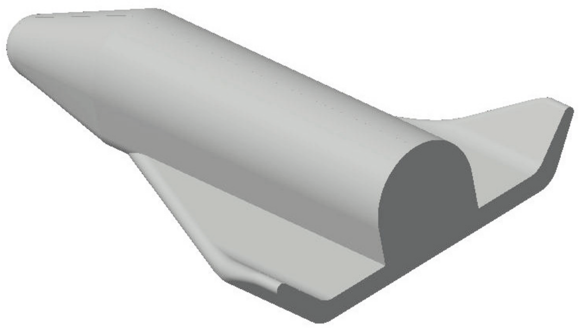


Figure 8: The configuration of the aerospace plane model.

Table 2: The main parameters of multi-GPU parallel platform.

NVIDIA GTX 1070 GPU	Device memory (GB)	8
	Compute capability	6.1
	Streaming multiprocessors	15
	Stream processors	1,920
	Single-precision (GFLOPS)	6,080
	Double-precision (GFLOPS)	194
	Memory bandwidth(GB/s)	256.3
Intel Xeon CPU E5-2670	Operating system	Windows 7
	Number of cores	8
	RAM (GB)	64
AIPs	CUDA	Version 8.0
	MPI	Version 1.4.1
	C	Visual studio 2013

on one node, which contains one Intel Xeon E5-2670 CPU at 2.60 GHz and four NVIDIA GTX 1070 GPUs. The main parameters of the multi-GPU parallel platform are shown in Table 2. The theoretical peak single-precision floating-point operations are more than those of double-precision. Hence, the single-precision data for GTX 1070 GPU parallelization is used.

5.2. Flow over the Physical Model at Mach Number 8.02. The Mach number and pressure contours on the symmetry plane at Mach number 8.02 are shown in Figures 13 and 14. The GPU parallel algorithm can clearly capture the wave structures in the flow field. The approximate boundary layer thickness is present in Figure 13. The bow shock at the head and the expansion wave at the upper surface (shown in Figure 14) have an important influence on the distributions of the flow field parameters.

The surface pressure distributions on the symmetry plane at Mach number 8.02, shown in Figures 15 and 16, depict pressure distributions on the symmetry plane, and the experimental results are presented here to facilitate a direct comparison. The experimental results are all from [33]. The pressure reaches a maximum at the head due to the effect of bow shock and then rapidly decreases. At the upper surface, the pressure decreases at the corner of cone and cylinder because of the expansion and then maintains about the same. As the angle of attack increases, the upper surface pressure decreases, while the opposite holds for the lower surface. It can be seen that the numerical results agree well with experimental data, and the agreement between numerical and experimental results is favourable.

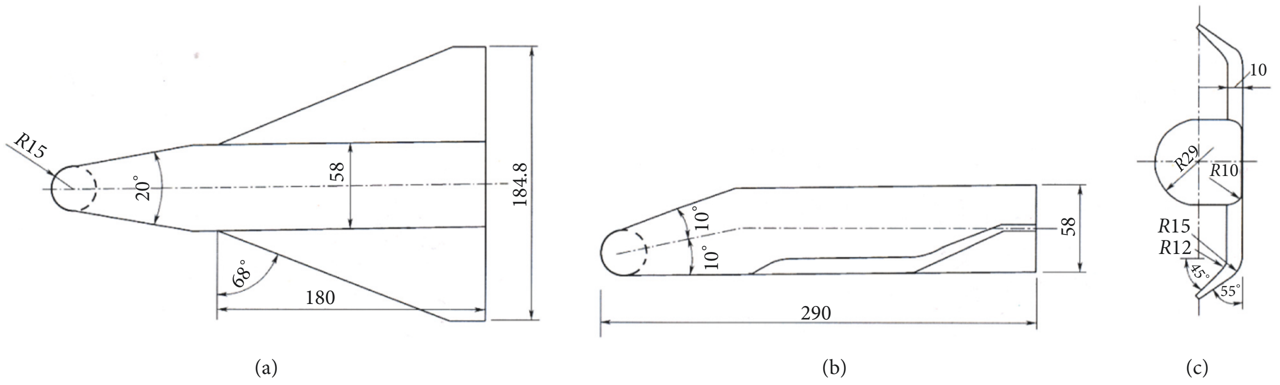


Figure 9: Orthographic views and the main dimensions. (a) Front view. (b) Top view. (c) Side view.

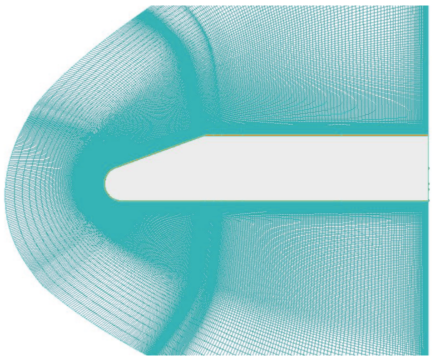


Figure 10: Computational mesh of the symmetry plane.

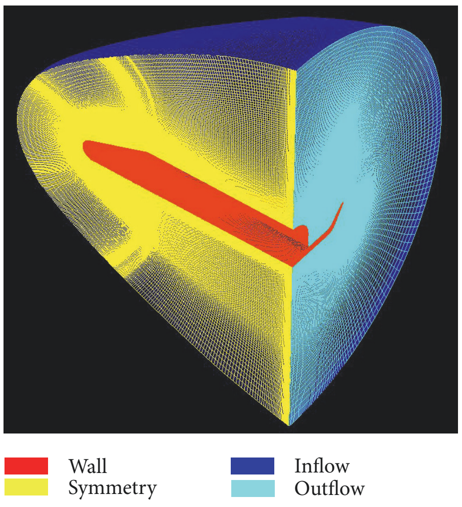


Figure 12: Boundary conditions of computational domain.

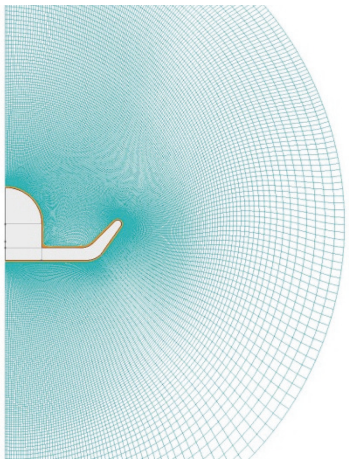
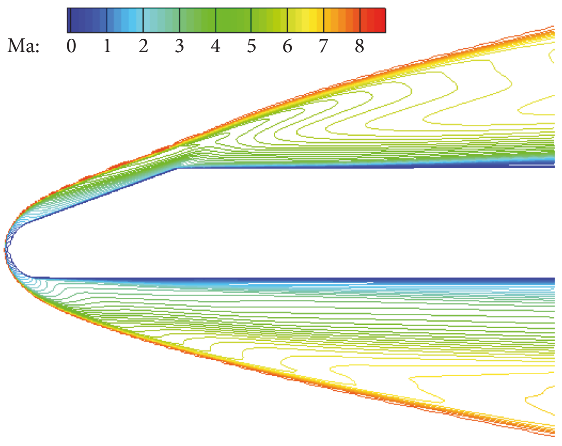


Figure 11: Computational mesh of the base plane.

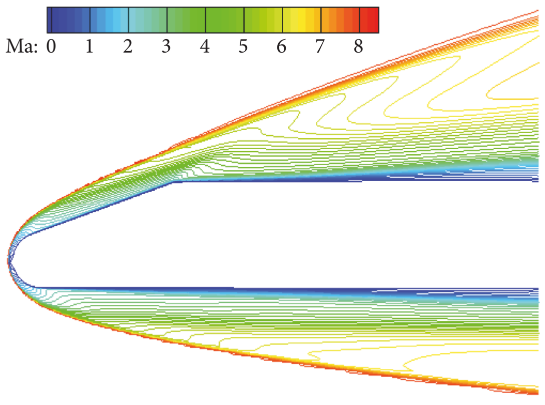
5.3. Flow over the Physical Model at Mach Number 10.02. In this section, the computed results of cases 3 and 4 are discussed. Figure 17 shows the heat flow contours of the aerospace plane. As clearly shown in Figure 17, extreme values occur at the head and wing twist due to the drastic changes of the temperature gradients. Figure 18 shows the temperature contours on the symmetry plane. Obviously, a high temperature zone is formed at the head of the physical model due to the influence of bow shock.

Figures 19 and 20 show the heat flow distributions on the symmetry plane of cases 3 and 4, which are close to the experimental data reported in [33]. For calculation of Q_W, see (8), and the reference value Q_{Ref} equals 441.1KW/m^2. As can be seen from the figures, the distributions of heat flow are consistent with the trend of pressure. Nevertheless, discrepancy exists between simulation and measurements, especially at the corner of cone and cylinder, due to the grid resolution. Overall, the agreement between numerical simulation and experimental measurements is also favourable.

5.4. GPU Parallel Performance Analysis. For case 1, numerical computations are performed on six different grid-refinement levels: coarser (1.78 million), coarse (3.56 million), medium (5.02 million), fine (7.08 million), finer (10.04 million), and finest (20.08 million), to analyse the performance of multi-GPU parallel algorithm.

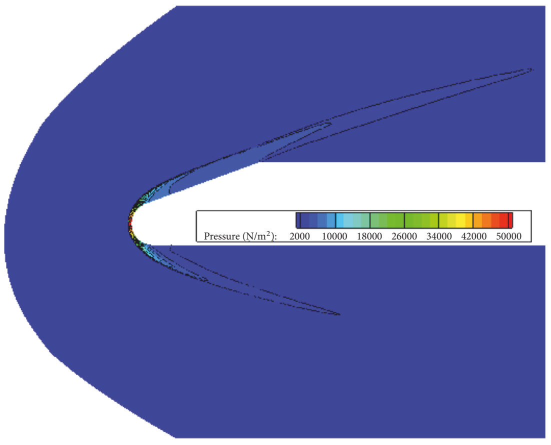


(a) $\alpha = 0^0$

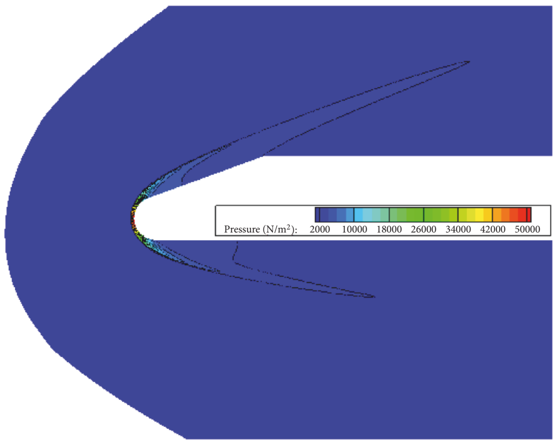


(b) $\alpha = 5^0$

Figure 13: Mach number contours on the symmetry plane.



(a) $\alpha = 0^0$



(b) $\alpha = 5^0$

Figure 14: Pressure contours on the symmetry plane.

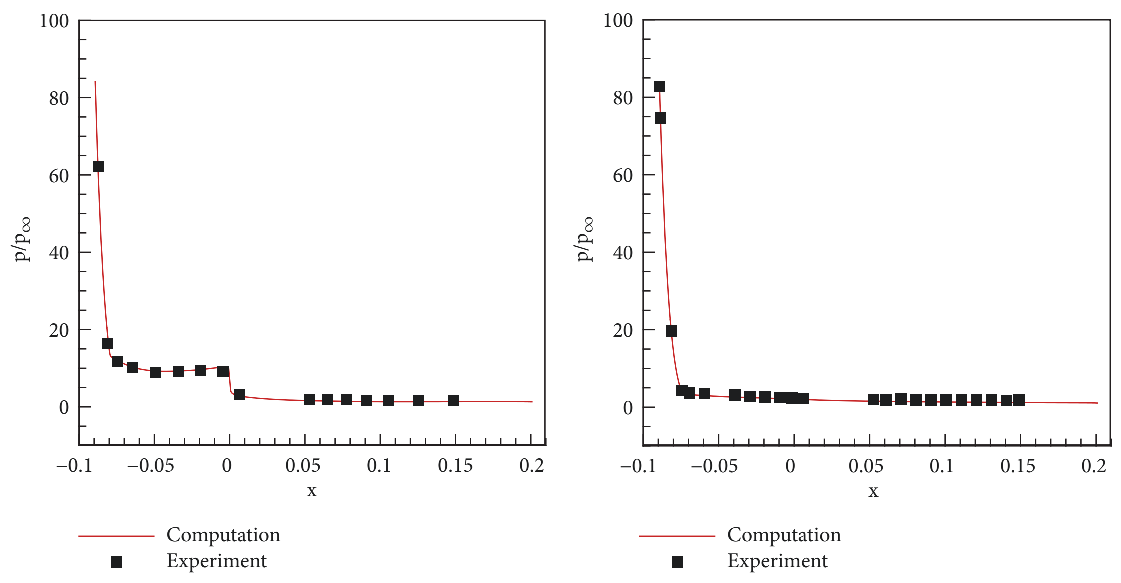
A good parallel system requires better acceleration, and speedup is an important parameter to measure the performance of parallel algorithms. The speedup is defined as the runtime of one CPU with eight cores divided by that of the GPU. In this paper, we simulate ten thousand time steps, and the runtime of one iteration step is achieved by averaging the execution time.

$$Sp = \frac{t_{CPU}}{t_{GPU}} \tag{30}$$

The runtime of one iteration step for one CPU and single GPU is presented in Table 3 (time is given in ms). Figure 21 shows that the speedup of the single GPU increases with the grid size. In comparison with CPU-based parallel computing, GPU parallelization can considerably improve the computational efficiency. The speedup reaches 63.22 for the coarser mesh, and 77.59 for the finest mesh. The speedup increases rapidly and then gradually becomes flat with the increase of grid size; this is because the proportion of the single instruction task increases (iteration, etc.) as the grid size increases compared with the branch instruction task (data transfer, etc.). GPUs are better suited for compute-intensive tasks than traditional CPUs. After the grid size attains the "limit value," the speedup remains basically unchanged and the "limit value" is closely related to the properties of GPU architecture.

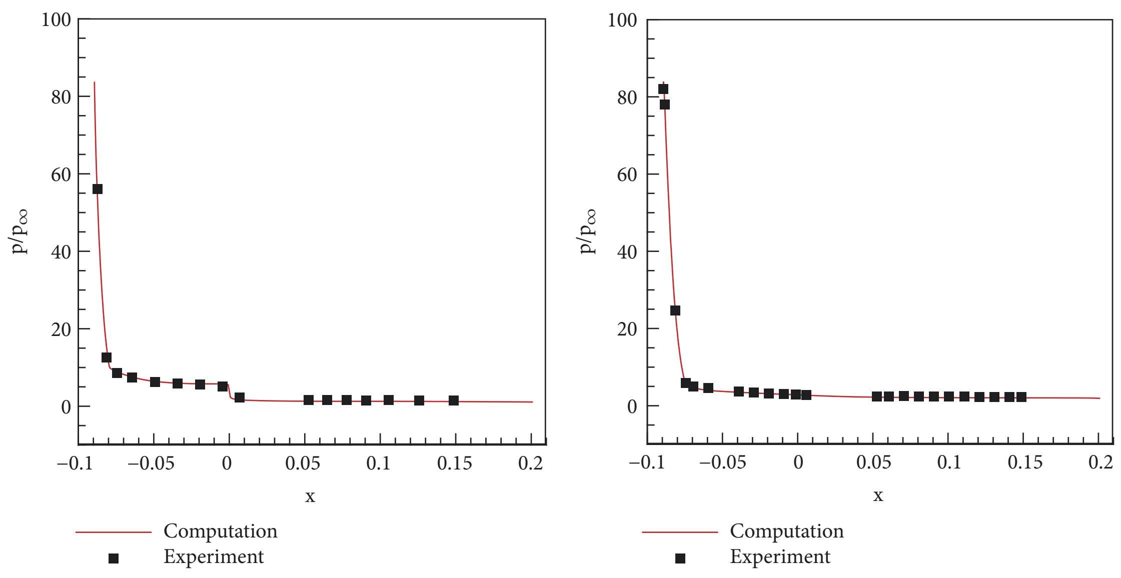
Table 3: Runtime of one CPU and single GPU.

No.	Grid size [million]	CPU [ms]	Single GPU [ms]
1	1.78	1,720.12	27.21
2	3.56	3,658.74	52.99
3	5.02	5,227.01	72.01
4	7.08	7,662.74	101.68
5	10.04	10,978.25	142.86
6	20.08	22,412.56	288.84



(a) Upper surface (b) Lower surface

Figure 15: Pressure distributions on the symmetry plane at $\alpha = 0^0$.



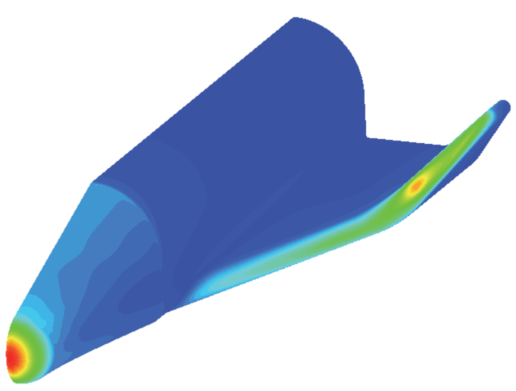
(a) Upper surface (b) Lower surface

Figure 16: Pressure distributions on the symmetry plane at $\alpha = 5^0$.

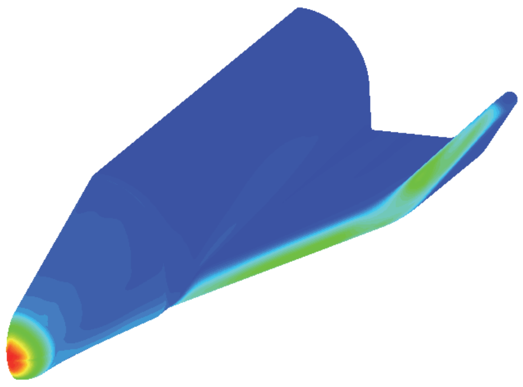
Table 4: Runtime of multi-GPUs.

No.	Grid size [million]	Two GPUs [ms]	Four GPUs [ms]
1	1.78	23.23	22.39
2	3.56	40.41	37.16
3	5.02	49.53	40.48
4	7.08	67.12	54.59
5	10.04	94.19	75.62
6	20.08	191.16	152.93

Table 4 shows the runtime of one iteration step for multi-GPUs (time is given in ms). Figure 22 shows that the speedup of different numbers of GPUs increases with the grid size, starting with a single GPU and going up to four GPUs. Multi-GPU parallelization prominently improves the computational efficiency with the increased grid size. The speedup of four GPUs reaches 76.83 for the coarser mesh and 146.55 for the finest mesh; this is far greater than the acceleration achieved by single GPU and two GPUs, which indicates that the multi-GPU parallel algorithm established in this paper can be applied to large-scale scientific computations.

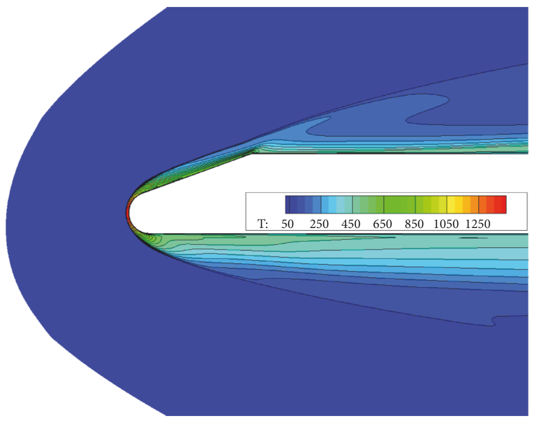


(a) $\alpha = 0^0$

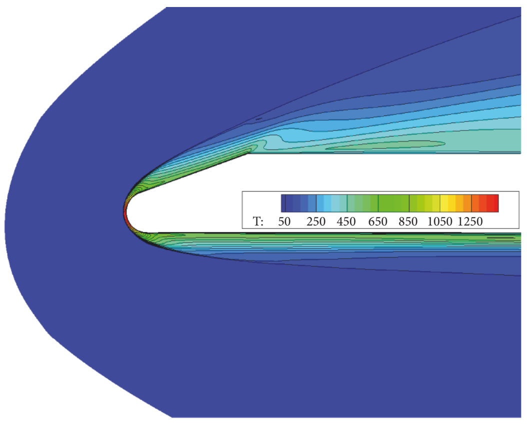


(b) $\alpha = 10^0$

FIGURE 17: Heat flow contours of the aerospace plane.

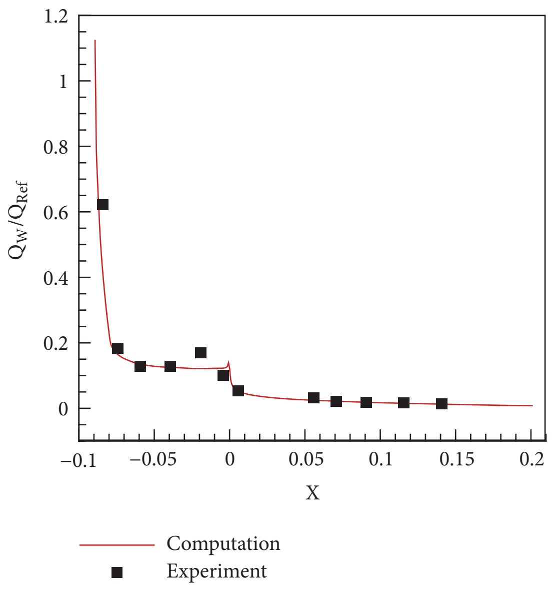


(a) $\alpha = 0^0$

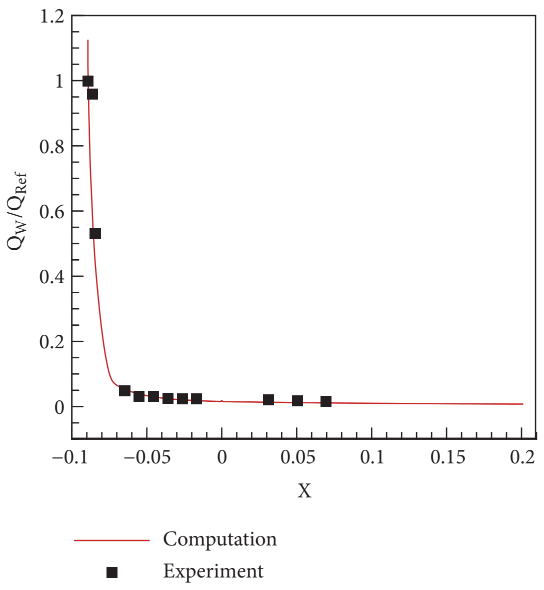


(b) $\alpha = 10^0$

FIGURE 18: Temperature contours on the symmetry plane.

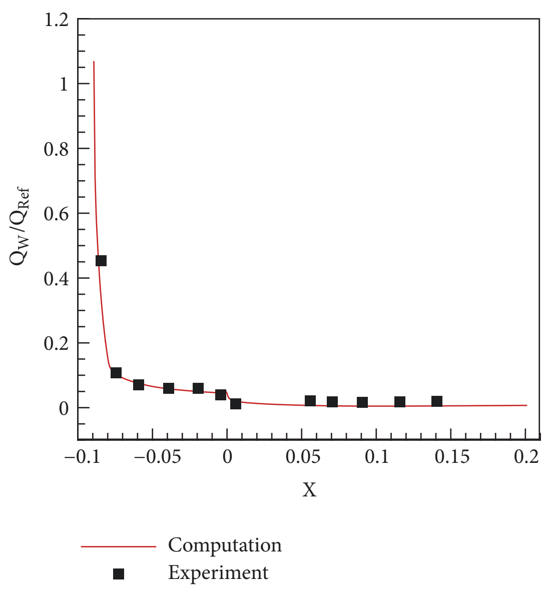


(a) Upper surface

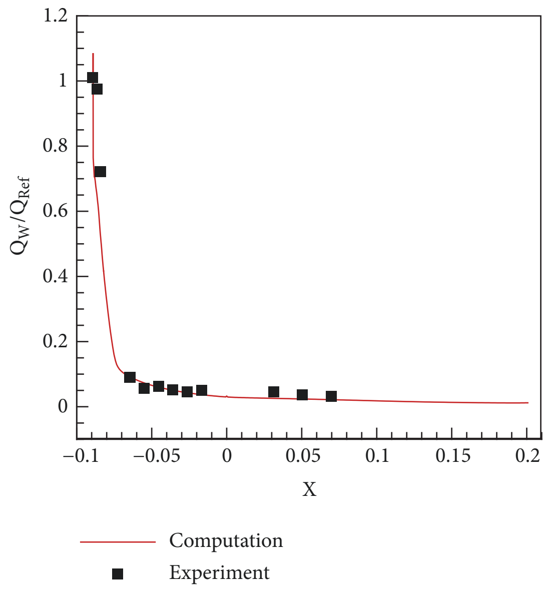


(b) Lower surface

FIGURE 19: Heat flow distributions on the symmetry plane at $\alpha = 0^0$.



(a) Upper surface



(b) Lower surface

Figure 20: Heat flow distributions on the symmetry plane at $\alpha = 10^0$.

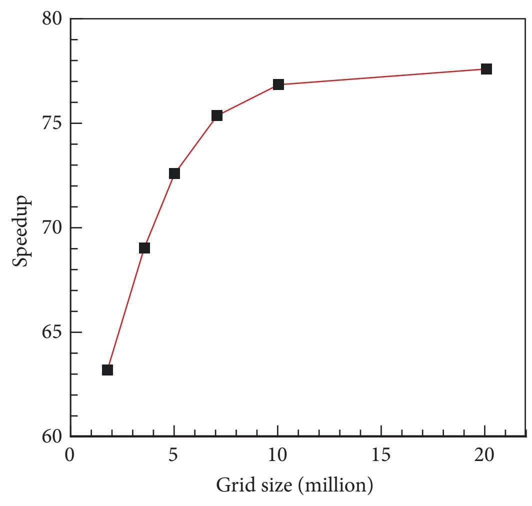


Figure 21: Speedup of single GPU increases with the number of meshes.

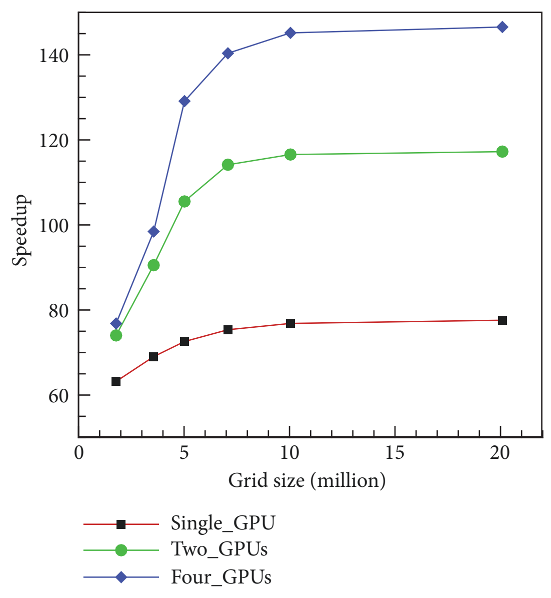


Figure 22: The speedup of multi-GPUs increases with the grid size.

6. Conclusions

In the present work, a multi-GPU parallel algorithm based on MPI+CUDA is established to accelerate the computations of hypersonic flow problems. Flow over an aerospace plane model at Mach numbers 8.02 and 10.02 is studied on NVIDIA GTX 1070 GPUs to verify the algorithm. The numerical results agree well with experimental data by comparing pressure and heat flow distributions. The speedup of single GPU reaches 63.22 for the coarser mesh and 77.59 for the finest mesh. In comparison with CPU-based parallel computing, GPU parallelization can considerably improve the computational efficiency. The speedup of four GPUs reaches 76.83 for the coarser mesh and 146.55 for the finest mesh; this is far greater than the acceleration achieved by single GPU and two GPUs, which indicates that the multi-GPU parallel algorithm established in this paper can be applied to large-scale scientific computations.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The first author would like to thank Dr. Feng Liu. All authors would like to express their thanks for the support from the National Natural Science Foundation of China (No. 11472004).

References

[1] J. D. Anderson Jr., *Fundamentals of Aerodynamics*, Tata McGraw-Hill Education, 5th edition, 2010.

[2] J. J. Bertin and R. M. Cummings, "Fifty years of hypersonics: where we've been, where we're going," *Progress in Aerospace Sciences*, vol. 39, no. 6-7, pp. 511–536, 2003.

[3] E. T. Curran, "Scramjet engines: the first forty years," *Journal of Propulsion and Power*, vol. 17, no. 6, pp. 1138–1148, 2001.

[4] J. D. Anderson Jr., *Hypersonic and High-Temperature Gas Dynamics*, AIAA, 2nd edition, 2006.

[5] F. Bonelli, M. Tuttafesta, G. Colonna, L. Cutrone, and G. Pascazio, "An MPI-CUDA approach for hypersonic flows with detailed state-to-state air kinetics using a GPU cluster," *Computer Physics Communications*, vol. 219, pp. 178–195, 2017.

[6] E. Elsen, P. LeGresley, and E. Darve, "Large calculation of the flow over a hypersonic vehicle using a GPU," *Journal of Computational Physics*, vol. 227, no. 24, pp. 10148–10161, 2008.

[7] W. L. Oberkampf and T. G. Trucano, "Verification and validation in computational fluid dynamics," *Progress in Aerospace Sciences*, vol. 38, no. 3, pp. 209–272, 2002.

[8] A. Afzal, Z. Ansari, A. Rimaz Faizabadi, and M. K. Ramis, "Parallelization strategies for computational fluid dynamics software: state of the art review," *Archives of Computational Methods in Engineerin: State-of-the-Art Reviews*, vol. 24, no. 2, pp. 337–363, 2017.

[9] K. E. Niemeyer and C.-J. Sung, "Recent progress and challenges in exploiting graphics processors in computational fluid dynamics," *The Journal of Supercomputing*, vol. 67, no. 2, pp. 528–564, 2014.

[10] T. Chen, Y. Ning, A. Amritkar et al., "Multi-GPU solution to the lattice Boltzmann method: an application in multiscale digital rock simulation for shale formation," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 19, Article ID e4530, 2018.

[11] M. J. Goldsworthy, "A GPU-CUDA based direct simulation Monte Carlo algorithm for real gas flows," *Computers & Fluids*, vol. 94, no. 94, pp. 58–68, 2014.

[12] NVIDIA, "CUDA C programming guide v8.0," 2017, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html.

[13] J. Huang, J. A. Lemkul, P. K. Eastman et al., "Molecular dynamics simulations using the drude polarizable force field on GPUs with OpenMM: implementation, validation, and benchmarks," *Journal of Computational Chemistry*, vol. 39, no. 21, pp. 1682–1689, 2018.

[14] S. S. Sawant, O. Tumuklu, R. Jambunathan, and D. A. Levin, "Application of adaptively refined unstructured grids in DSMC to shock wave simulations," *Computers & Fluids*, vol. 170, pp. 197–212, 2018.

[15] A. Khajeh-Saeed and J. B. Perot, "Computational fluid dynamics simulations using many graphics processors," *Computing in Science & Engineering*, vol. 14, no. 3, pp. 10–19, 2011.

[16] X. Guo, B. Tang, J. Tao et al., "Large scale GPU accelerated PPMLR-MHD simulations for space weather forecast," in *Proceedings of 16th IEEE/ACM International Symposium on Cluster*, pp. 576–581, 2016.

[17] P. S. Tamizharasan and N. Ramasubramanian, "Analysis of large deviations behavior of multi-GPU memory access in deep learning," *The Journal of Supercomputing*, vol. 74, no. 5, pp. 2199–2212, 2018.

[18] T. Brandvik and G. Pullan, "Acceleration of a 3D Euler solver using commodity graphics hardware," in *Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008.

[19] A. Khajeh-Saeed and J. B. Perot, "Direct numerical simulation of turbulence using GPU accelerated supercomputers," *Journal of Computational Physics*, vol. 235, no. 4, pp. 241–257, 2013.

[20] X. Wang, Y. Shangguan, N. Onodera, H. Kobayashi, and T. Aoki, "Direct numerical simulation and large eddy simulation on a turbulent wall-bounded flow using lattice Boltzmann method and multiple GPUs," *Mathematical Problems in Engineering*, vol. 2014, Article ID 742432, 10 pages, 2014.

[21] V. N. Emelyanov, A. G. Karpenko, A. S. Kozelkov, I. V. Teterina, K. N. Volkov, and A. V. Yalozo, "Analysis of impact of general-purpose graphics processor units in supersonic flow modeling," *Acta Astronautica*, vol. 135, pp. 198–207, 2017.

[22] J.-L. Zhang, Z.-H. Ma, H.-Q. Chen, and C. Cao, "A GPU-accelerated implicit meshless method for compressible flows," *Journal of Computational Physics*, vol. 360, pp. 39–56, 2018.

[23] J. Blazek, *Computational Fluid Dynamics: Principles and Applications*, Elsevier, 3rd edition, 2015.

[24] R. Chamberlain, "Calculation of three-dimensional jet interaction flowfields," in *Proceedings of the 26th Joint Propulsion Conference*, 1990.

[25] M.-S. Liou, "A sequel to AUSM, part II: AUSM$^+$-up for all speeds," *Journal of Computational Physics*, vol. 214, no. 1, pp. 137–170, 2006.

[26] B. van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method," *Journal of Computational Physics*, vol. 32, no. 1, pp. 101–136, 1979.

[27] A. Jameson, W. Schmidt, and E. Turkel, "Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes," in *Proceedings of the 14th Fluid and Plasma Dynamics Conference*, Palo Alto, Calif, USA, 1981.

[28] R. Yam-Uicab, J. López-Martínez, E. Llanes-Castro, L. Narvaez-Díaz, and J. Trejo-Sánchez, "A parallel algorithm for the counting of ellipses present in conglomerates using GPU," *Mathematical Problems in Engineering*, vol. 2018, Article ID 571463, 17 pages, 2018.

[29] P. D. Mininni, D. Rosenberg, R. Reddy, and A. Pouquet, "A hybrid MPI-OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence," *Parallel Computing*, vol. 37, no. 6-7, pp. 316–326, 2011.

[30] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters," in *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010.

[31] B. Baghapour, A. McCall, and C. J. Roy, "Multilevel parallelism for CFD codes on heterogeneous platforms," in *Proceedings of the 46th AIAA Fluid Dynamics Conference*, 2016.

[32] S. Potluri, A. Goswami, D. Rossetti, C. J. Newburn, M. G. Venkata, and N. Imam, "GPU-centric communication on NVIDIA GPU clusters with InfiniBand: a case study with OpenSHMEM," in *Proceedings of the 24th IEEE International Conference on High Performance Computing, HiPC*, pp. 253–262, 2017.

[33] S. X. Li, *Hypersonic Flow Characteristics of Typical Shapes*, National Defense Industry Press, 2007.