Hindawi
Scientific Programming
Volume 2020, Article ID 8847481, 17 pages
https://doi.org/10.1155/2020/8847481

Research Article

Massively Parallel CFD Simulation Software: CCFD Development and Optimization Based on Sunway TaihuLight

Xiazhen Liu,[1,2] Zhonghua Lu,[1,2] Wu Yuan,[1,2] Wenpeng Ma,[3] and Jian Zhang[1]

[1]*Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China*
[2]*University of Chinese Academy of Sciences, Beijing 100049, China*
[3]*College of Computer and Information Technology, Xinyang Normal University, Xinyang, Henan 464000, China*

Correspondence should be addressed to Zhonghua Lu; zhlu@cnic.cn

Received 16 April 2020; Revised 17 June 2020; Accepted 29 June 2020; Published 22 July 2020

Academic Editor: Chenxi Huang

A parallel framework software, CCFD, based on the structure grid, and suitable for parallel computing of super-large-scale structure blocks, is designed and implemented. An overdecomposition method, in which the load balancing strategy is based on the domain decomposition method, is designed for the graph subdivision algorithm. This method takes computation and communication as the limiting condition and realizes the load balance between blocks by dividing the weighted graph. The fast convergence technique of a high-efficiency parallel geometric multigrid greatly improves the parallel efficiency and convergence speed of CCFD software. This paper introduces the software structure, process invocations, and calculation method of CCFD and introduces a hybrid parallel acceleration technology based on the Sunway TaihuLight heterogeneous architecture. The results calculated by Onera-M6 and DLR-F6 standard model show that the software structure and method in this paper are feasible and can meet the requirements of a large-scale parallel solution.

1. Introduction

Computational fluid dynamics (CFD) is a technique for numerical simulation and analysis of fluid mechanics problems. CFD technology is increasingly used in aerospace, meteorological prediction, and other fields [1–4]. Some parallel computing frameworks have been used with parallel CFD [5, 6], e.g., OpenFOAM [7] and SU^2 [8, 9]. Parallel computing methods are increasingly used to solve large-scale computationally intensive problems. Parallel programming environments such as OpenMP [10, 11] and MPI [12, 13] have appeared on parallel machines, networked workstations, and supercomputers. The development of CFD in numerical methods, turbulent models, and mesh generation technology has led to its making strides in the simulation accuracy and capacity of complex geometric shapes. Meanwhile, with the increasing complexity of engineering problems and the rapid advancement of numerical simulation calculation technology, the requirements for simulation accuracy are getting increasingly stringent, and the amount of calculation required has increased geometrically. Parallel CFD technology has become the primary method for solving complex simulation calculations [14, 15].

Current CFD methods use a computational grid to simulate the flows in complex geometries and form different cells through the discreteness of the grid, which is then followed by the numerical calculation [15–17]. In order to simulate the shape structure more realistically, a multiblock approach is generally used to simulate different spatial regions. A commonly used method for solving large-scale CFD problems is applying domain decomposition technology in order to decompose more subblocks and allocating these decomposed calculation regions to different processes or threads for performing parallel calculations [18–20]. The methods of parallel computing are very different under different parallel programming environments. The MPI method allocates one or more computing areas to a processor. As the different computing areas are completely isolated, the data exchange at the boundary needs to be completed through communication. The OpenMP method uses the shared memory in an environment in which the computing area is actually on one processor, and the grid data are completely shared, rendering data communication unnecessary. Therefore, designing different parallel frameworks is important when using different programming environments and grid types [21, 22].

Supercomputer architectures are generally either homogeneous [23] or heterogeneous [24]. Each core architecture of a homogeneous machine is the same, and their status is equal. They can share the same code or execute different codes on each core [25]. Homogeneous processors can be interconnected using shared storage or through cache [26–28]. Heterogeneous architectures represent a new hardware platform that allows different types of processors to work together efficiently in shared memory [24]. The heterogeneous architecture includes not only the traditional CPU but also other accelerator units such as GPUs and MICs [24, 29]. These different computing units have different instruction set architectures and memory spaces [30]. The CPU and the accelerators have unified access to the system memory through the Memory Management Unit (MMU) [31, 32]. With the evolution of supercomputer technology, the traditional homogeneous architecture has been unable to meet the increasing requirements for computing power and storage. For this reason, the heterogeneous architecture has become the most important technology for the development of supercomputers. Heterogeneous architecture machines account for more than 30% of the current TOP500 list of supercomputers [33]. Of the top five parallel computers, four have heterogeneous architectures.

The main purpose of the research content in this paper is to develop an open-source large-scale parallel solver based on a multiblock structure grid. First, multicore parallel computing based on the homogeneous architecture was implemented. Second, based on the Sunway TaihuLight heterogeneous system [34], the calculation of data on the CPU and accelerator communication were achieved using the MPI + OpenACC/Athread hybrid programming model [35] and direct memory access (DMA) technology [36]. Third, through the use of stencil calculation, the boundary information is exchanged through register communication, and the single instruction, multiple data (SIMD), and assembly instructions are used to optimize the computationally intensive area and further improve calculation performance. Finally, the parallel simulation of the Onera M6 wing model [37] realizes a parallel calculation of 500,000 cores of 1 billion grids and achieves ideal parallel acceleration efficiency.

2. CCFD Software Design and Implementation

2.1. CCFD Software Architecture. CCFD is designed by object-oriented programming [38], and its architecture is shown in Figure 1. The entire software platform consists of an input/output control module, a computational geometry module, a solver module, and a parallel algorithm module. The system architecture of CCFD needs to take into account the characteristics of various calculation methods and physical models and design a simulation software with flexibility, robustness, accuracy, and data security. In addition, the parallel framework design needs to meet the requirements for high parallel scalability, high data transmission efficiency, and fast response time. To increase maintainability and scalability, the CCFD architecture is organized with blocks as the basic data structure units. Based on this, the use and loading of data by each module in the platform greatly simplifies the program interface and reduces the difficulty of development.

The CCFD software platform system is built on block units. The structure and basic unit calling of CCFD are the blocks and boundaries. The internal points and boundary points of the blocks are divided into two data structures: BLOCKs and BCs. The main information such as geometric variables and calculation variables are stored in the block, while the boundary type and starting information are stored in the boundary. In accordance with the advantages and disadvantages of an array of structures (AoS) and a structure of arrays (SoA), the variables of CCFD on the block are organized in the form of SoA. The advantages of this method can be stated as follows. On the one hand, the centralized use and loading of grid variables improves the hit rate and data utilization of cache. On the other hand, it facilitates the packaging and collection operation of parallel communication, which is beneficial to the parallel optimization of the large-scale computation of CCFD.

CCFD integrates various computing models and methods effectively. In order to facilitate the software module specifications, code organization development, and extended maintenance, we organize the entire software system with the application layered model, which makes the code modules independent between layers and interconnects the calling relationships. The solver can be divided into five layers. The first layer is grid input, which controls the file input and the load balancing of block calculation and communication on the processor. In the second layer, the calculation is divided into a multiblock method and conventional calculation methods according to the calculation demand. The third layer is divided into Euler, N-S, and turbulent model based on the flow equation. The fourth layer is a method of selecting spatial and temporal terms for different equations. The fifth layer is a basic algorithm layer that includes a sparse matrix solving method, modular parallel communications, and error information output.

2.2. CCFD Software Process. Software process design needs to fully take into account the actual calculation methods used. For example, CCFD uses a multiblock structure grid based on a second-order precision finite volume method to solve partial differential equations and uses multigrid methods to accelerate convergence techniques. In order to allow large-scale structure blocks to be calculated on parallel machines, the domain decomposition technique is used to divide a large-scale block into smaller-scale subblocks. The adjacency relationships between the subblocks are stored in the information of the block, and the information is shared and updated after each iteration of calculation to ensure the compatibility between the results of the partial differential

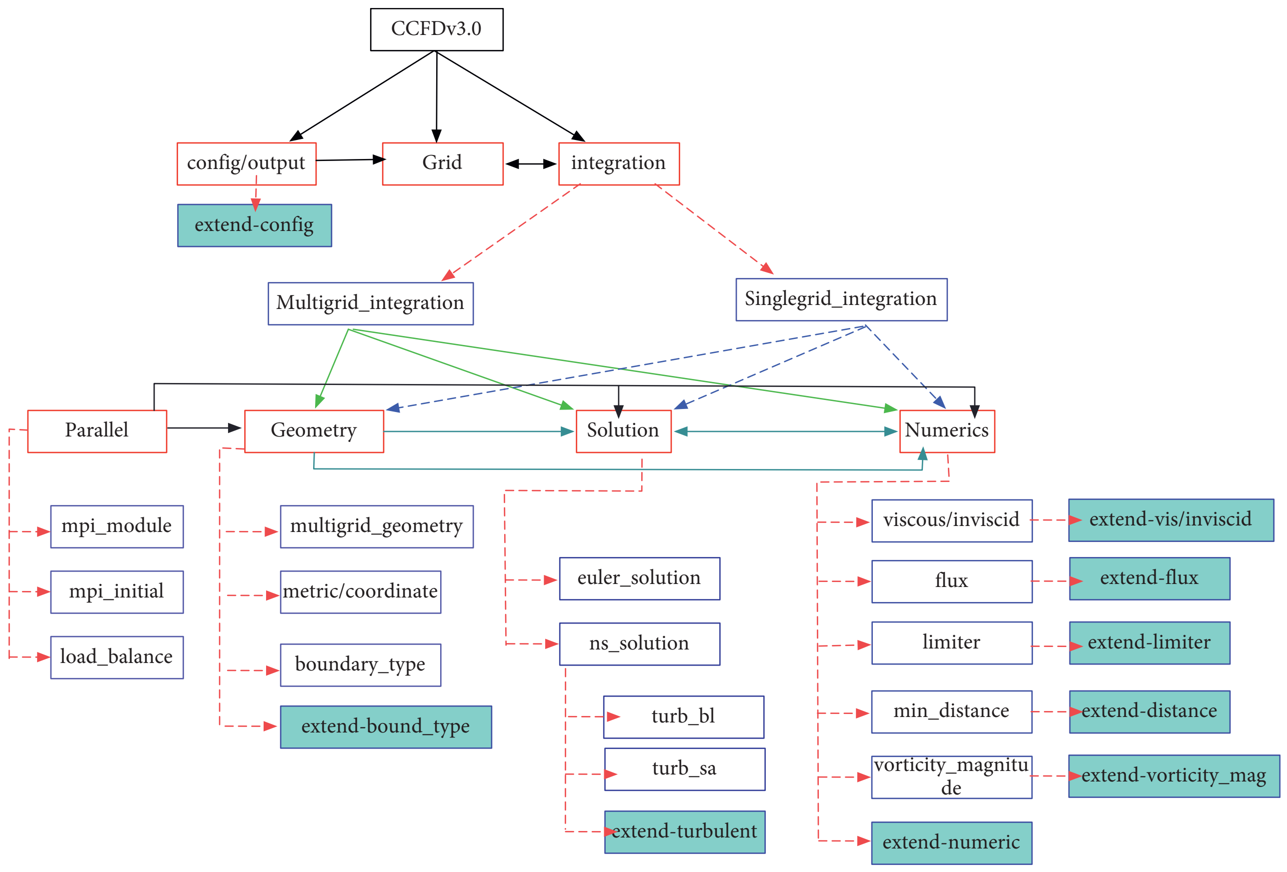


Figure 1: CCFD software framework.

equations and the unpartitioned block. The area of the block is assigned to different processors for calculation. This mapping and allocation process involves the calculation and communication load balancing of the block on the processor. Therefore, the entire software system can be thought of as being roughly divided into five parts: preprocessing, area decomposition, load balancing, flow field calculation, and postprocessing output.

The software process design considers the calculation methods used in practice and meets the communication needs between the multiblocks. Regardless of partition parallel computing or multiblock computing, it is essential to calculate the partial differential equations on each block. The calculation time of subblocks of different sizes differs depending on the process. The larger the scale, the larger the calculation time. If the subblock sizes are not uniform, there will be multiple processes waiting for individual processes. Based on the above characteristics of CCFD software, we designed a software parallel process, as shown in the flowchart in Figure 2.

The unit data based on the block, which is the functional module structure of CCFD, are shown in Figure 3. The structure is divided into eight modules. The calling and connection between each module are based on the block unit. The main body of the calling is the parallel solver. Through the parallel solver, the block geometry module, parallel module, source term solution, right term solution, turbulent viscous, and output control are used to complete the entire solution process. The connection lines in Figure 3 indicate the link relationship between the modules. The entire process largely determines the parallel expansion efficiency of the software system. The allocation strategy of blocks to processors also affects how much the parallel efficiency can be improved. For example, an imbalance in the calculation and the communication load between blocks on the process will cause the process to wait idle for the overall calculation.

3. CCFD Parallel Design

3.1. Load Balancing Strategy. One goal of domain decomposition technology is to subdivide a large size of block into smaller subblocks. The subblocks' computing area is allocated to different processes, or threads, for parallel computing processing [19, 20]. As shown in Figure 4, a three-dimensional structure block is subdivided into multiple smaller subblocks, each having multiple adjacency surfaces, which represent the communication relationship between the subblocks. When the subblocks are calculated on different processors, the adjacent surfaces need to carry out data transmission and communication.

Once the domain is decomposed, a larger number of smaller, more uniform blocks are formed. These subblock computing areas are allocated to different processes, or threads, for concurrent execution. In this particular process, we need to ensure that the calculation and communication

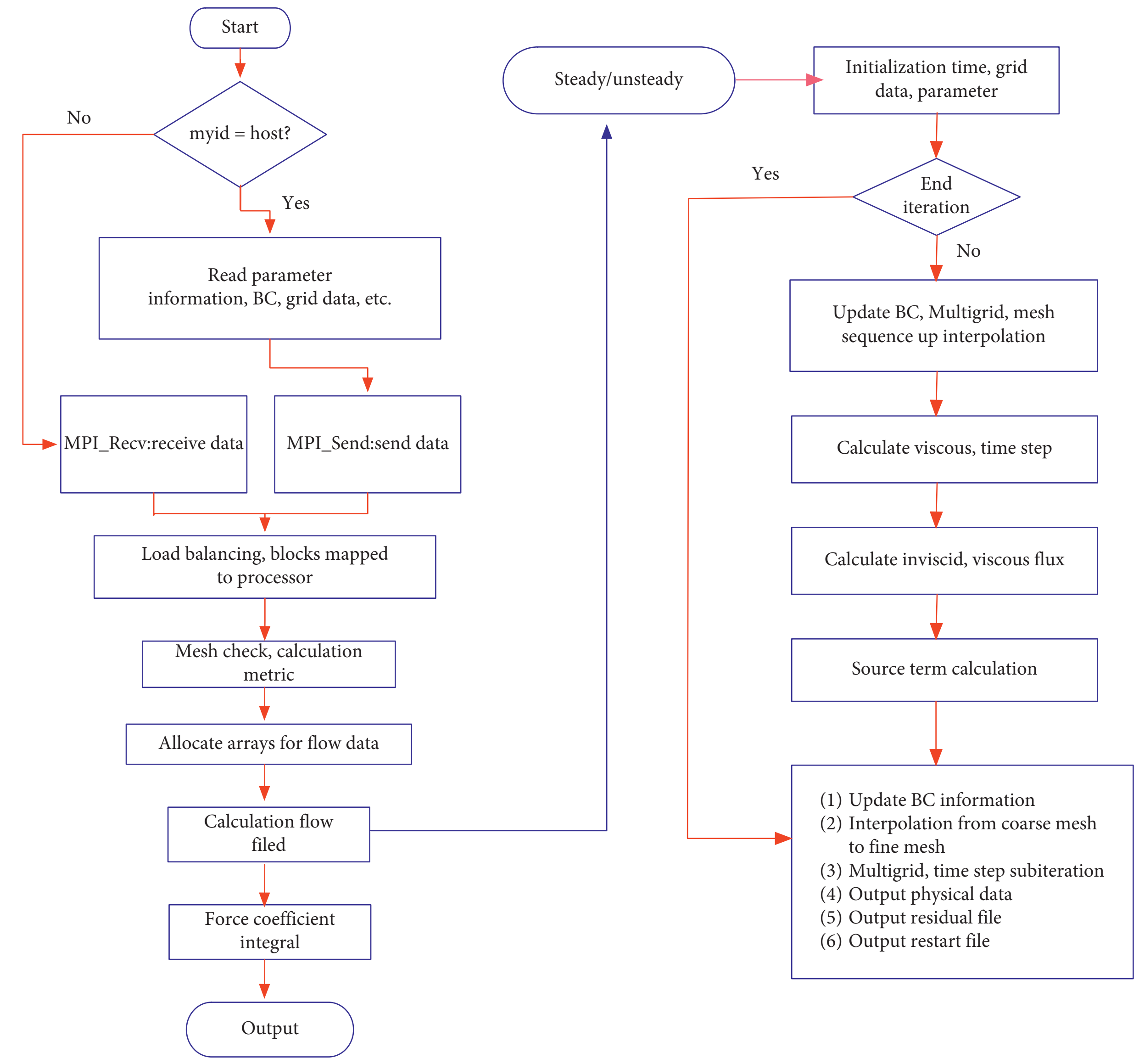

Start
myid = host?
No
Yes
Read parameter information, BC, grid data, etc.
MPI_Recv:receive data
MPI_Send:send data
Load balancing, blocks mapped to processor
Mesh check, calculation metric
Allocate arrays for flow data
Calculation flow filed
Force coefficient integral
Output
Steady/unsteady
Initialization time, grid data, parameter
End iteration
Yes
No
Update BC, Multigrid, mesh sequence up interpolation
Calculate viscous, time step
Calculate inviscid, viscous flux
Source term calculation
(1) Update BC information
(2) Interpolation from coarse mesh to fine mesh
(3) Multigrid, time step subiteration
(4) Output physical data
(5) Output residual file
(6) Output restart file

Figure 2: CCFD software process.

allocated to each processor are relatively balanced, and there is no idle waiting of the processor. This process of block and course mapping allocation is called load balancing. Currently, there is much research taking place on load balancing methods, and several algorithms have been proposed, including greedy algorithm, spectral method, genetic algorithm, and ant colony algorithm [39]. These algorithms are applicable to unstructured grids but cannot be directly applied to structured grids. Traditional structure grid load balancing methods generally use a cyclic allocation method or a uniform distribution method, neither of which consider the load balance of the calculation amount and communication [20].

Aiming at the problem of large-scale multiblock structure grid computing and communication load balancing, CCFD scientists proposed an overdecomposition method. CCFD divides the blocks and processor mapping and distribution process into two layers: coarse and fine. At the coarse layer, domain decomposition methods such as recursive boundary are used to subdivide large and uneven blocks into smaller and more uniform subblocks. The algorithm is a method of approximate linear time complexity, and the decomposition time is related to the number of blocks and the maximum dimension of the block. The purpose of the subdivision operation in the coarse layer is to lay a foundation for further balancing the calculation and communication loads. For example, uneven blocks make it difficult to balance the calculation and communication allocation, which will result in a series of processor-independent block sequences. Before using the fine-layer load balancing algorithm, the subblocks of the coarse layer need to be renumbered. The number of grids in each subblock was used as the statistical standard for the calculation amount, and the adjacent faces of each subblock and other subblocks are the statistical standard for communication volume. This information is redrawn into a multidirectional weighted

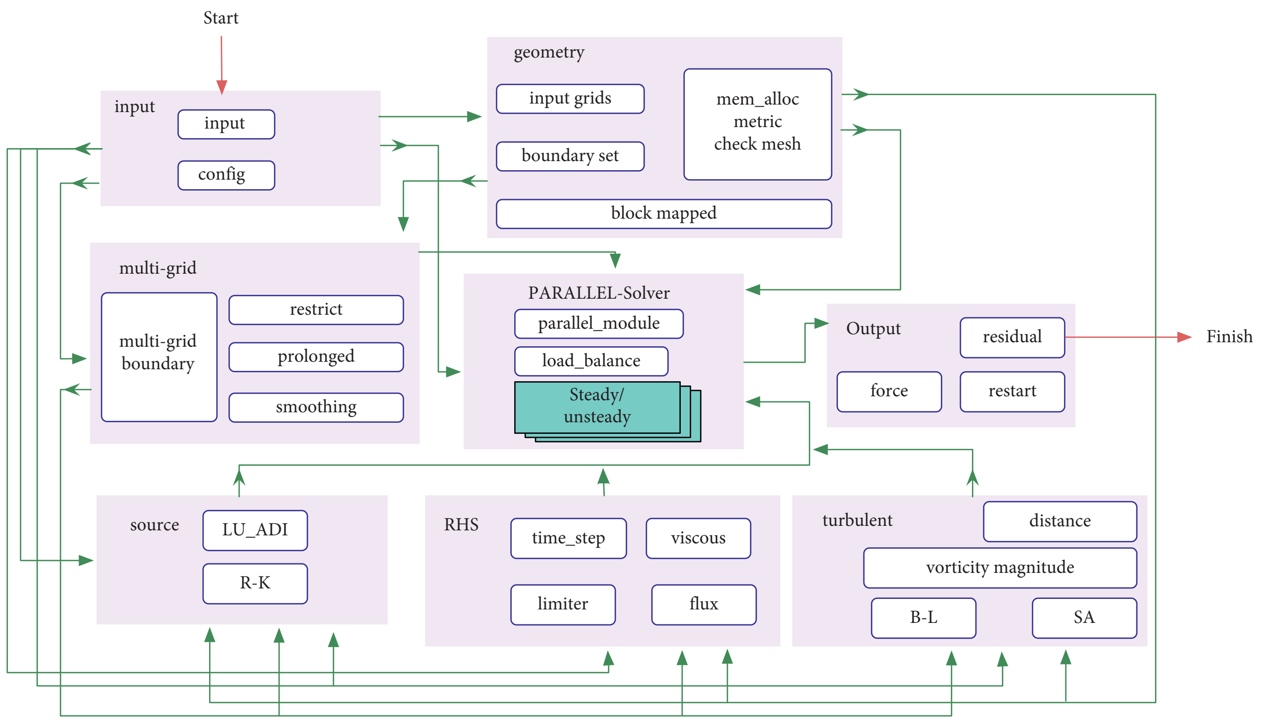


FIGURE 3: CCFD method and module invocation relationship.

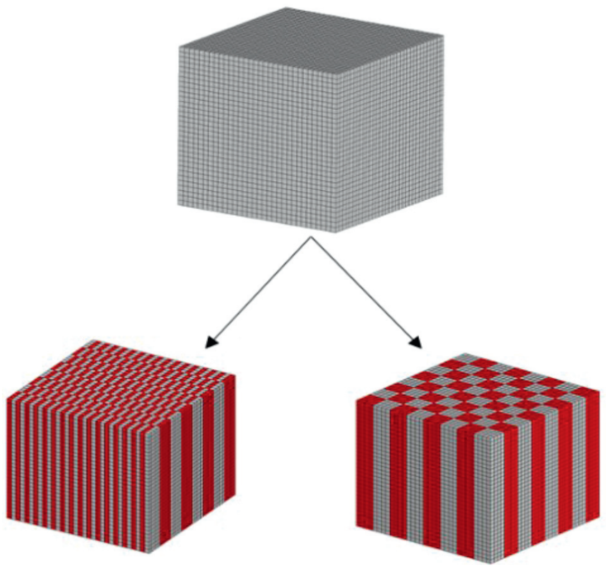


FIGURE 4: Domain decomposition.

graph based on the subblock topology, and the communication volume and calculation volume are the two weights of the weighted graph [40]. The fine-layer graph partitioning algorithm performs graph segmentation based on the weighted graph formed by the upper layer. The total amount of calculations of the subdomains formed as far as possible is balanced, and the total amount of communication in the subdomains is balanced with the smallest amount of communication between subblocks [41]. Figure 5 shows a multidirectional weighted graph $G = (V, E)$ drawn after preprocessing the CCFD blocks, and finding a partitioning of the vertices of G into n sets in such a way that the sums of the vertices weights in each set are as equal as possible, and the sum of the edges weights crossing between sets is minimized. Therefore, this partitioning of the vertices guarantees the computation in each processor is equal, and the communication between processors is minimized. The method based on k-way graph partitioning has moderate computational complexity, its complexity for computing is $O(|V|\log k)$, and $\log k$ is levels of recursive bisection for the algorithm.

In this paper, we discuss how we found that the calculation scale and the number of subblocks of the calculation example have a direct relationship with the amount of calculation and communication load balancing. For example, the number of subblocks and the size of the subblocks directly affect the distribution of the amount of calculation in the process. The greater the number of subblocks and the balance of the size, the more balanced the distribution of the calculations. However, the greater the number of partitions, the larger the adjacent area of the block. This condition directly leads to an increase in communication and affects the realization of communication load balancing among the blocks. As an example, the block size in the M6 calculation ranges from 11,000 to 13,000, and the area of cut face is between 450 and 625. The block size and the adjacent surface after domain decomposition are relatively balanced, which is conducive to load balancing. Figure 6 shows the actual calculation load distribution of each process in the M6 calculation example, using 256 or 512 threads.

Testing the M6 model, we found that, with the increase in parallel scale, the actual distribution of the computing load will become worse, along with an increase in the

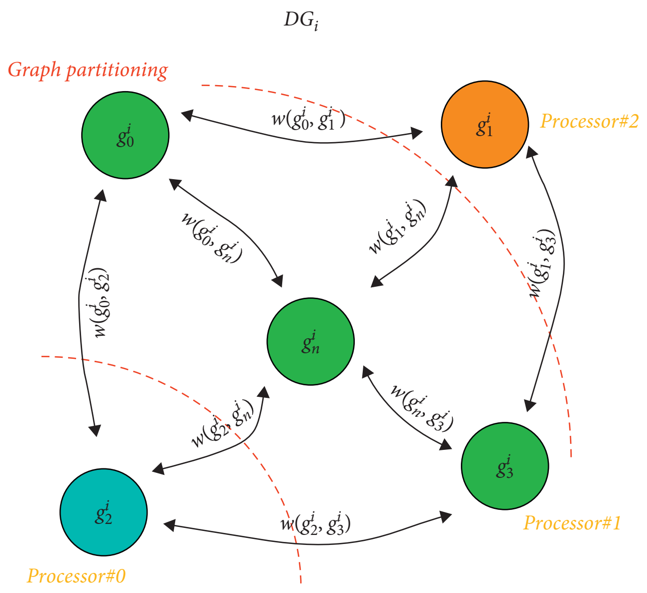


Figure 5: Multiblock structured grids for graph partitioning.

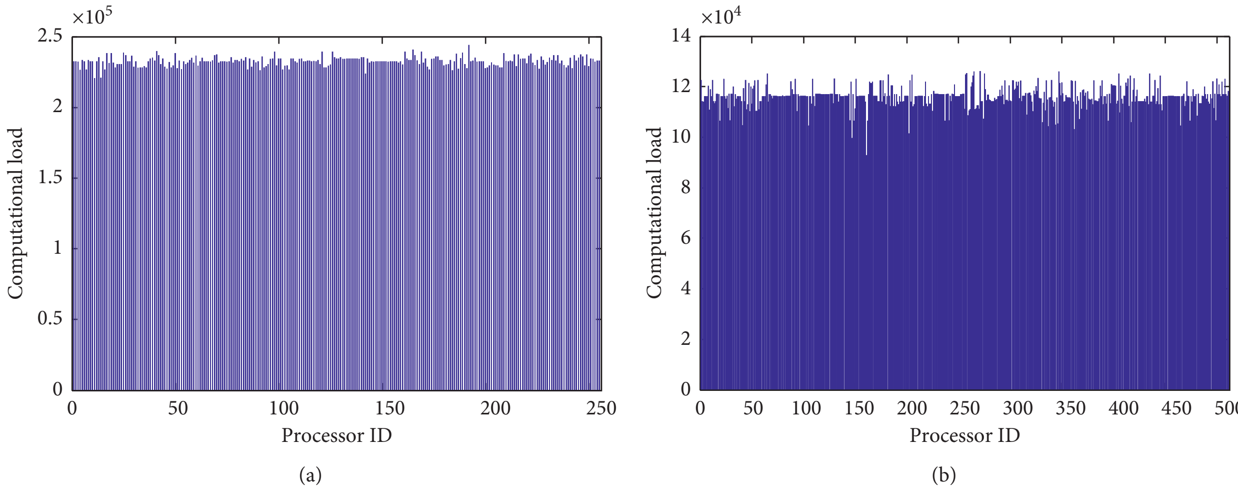


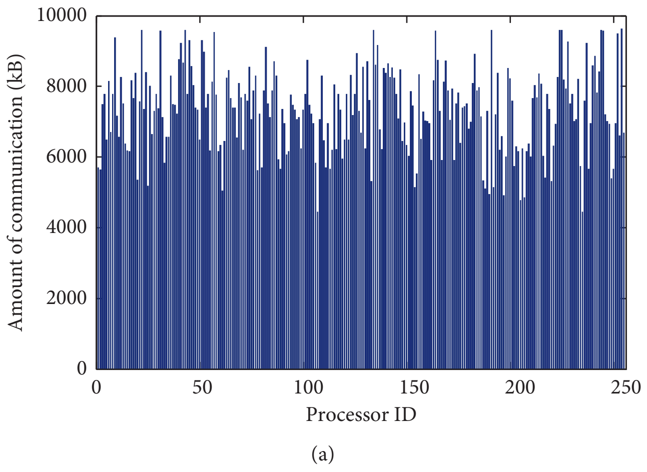
Figure 6: Calculation load distribution in the use of 256 and 512 threads.

number of threads. In this case, 256 threads are the critical point of the calculation load balancing. When the number of threads is less than 256, the overall computational load balance is ideal. However, the actual distribution of computational load will be significantly different when the number of threads is greater than 256. When the subblock size and the total number of blocks in the partition are stable, the number of blocks allocated to each process decreases, with the number of threads increasing. Then, the influence on the distribution of the calculation amount caused by the difference in the number of blocks on each thread and the actual size of the block increases, resulting in a decrease in the degree of interprocess calculation load balancing.

After the domain decomposition, the total calculation amount of the block remains stable, and the absolute amount of communication between the blocks increases with the increase of the number of parallel cores. In terms of communication load balancing between blocks, the degree of communication load balancing decreases with the increase in parallel scale. Figure 7 shows the actual communication load distribution of each process in the M6 calculation example, using 256 and 512 threads.

3.2. Parallel Data Structure. CCFD supports two grid formats: CFD General Symbol Standard (CGNS) grid and PLOT3D. The operation of the software system involves the access to and update of various data, which mainly include control parameters, grid data, flow field data, and temporary variables. The specific block variables are as follows:

```
type, public: blocks_type
type (cell_type), pointer, dimension(:,:,:): cell
type (coordinate_type), pointer, dimension(:,:,:): coordinate
type (metric_type), pointer, dimension(:,:,:): metric
type (variable_types), pointer, dimension(:,:,:): variable
```



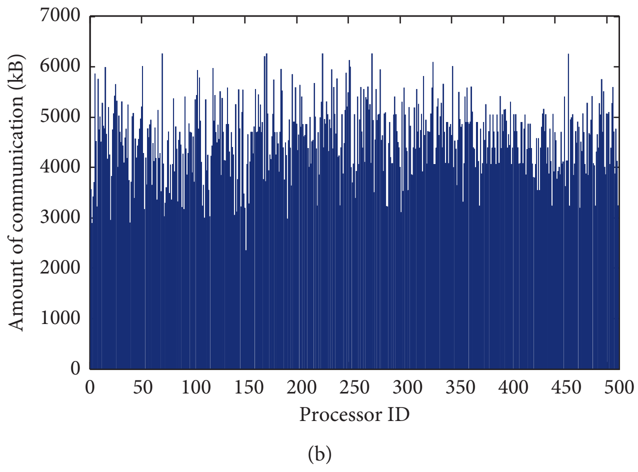


Figure 7: Communication load distribution using 256 and 512 threads.

type (variable_mg_types), pointer, dimension(:,:,): variable_mg

type (turbulent_type), pointer, dimension(:,:,): turbulent

end type blocks_type

All data can be divided into global, local, and exchange data. Global data refer to the data that all processors will save and not modify during calculation, such as control parameters. Local data refer to the data that are only stored locally and are not shared with other processors, such as flow field data in blocks' interior points. Exchange data refer to the data that need to be exchanged with other processes, such as boundary data of adjacent faces.

When large-scale parallel computing is performed in CFD, huge data reads and exchanges offer a great test to the data structure and data flow. The reasonable data structure and data flow design directly affect the reliability and efficiency of the platform operation. In order to ensure the efficiency and security of data, the design of CCFD parallel software is based on the block data structure. The grid data, boundary data, and flow field data are stored in each block unit. As shown in Figure 8, a block stores information such as grid coordinates, boundary information, and primitive variables [40, 41].

3.3. Parallel Communication Optimization. The proportion of blocks computing and communicating is one of the main factors in the parallel expansion performance of software. Ideally, a large amount of local computing is a computational priority problem, and a small amount of communication is good for parallel expansion. If the amount of local computing is small, the large amount of communication is a communication priority problem, which leads to poor parallel scalability. The computing model and method used by parallel software determine the amount of local computing, and the communication mode of parallel software affects the amount of communication. For example, in actual calculations, there are some challenges: many small and frequent data communications, a large amount of data and a fixed number of a block boundary updates, one-to-many broadcast data, and many-to-one master-slave communication. In terms of these challenges, using a different and flexible communication mode facilitates the software's parallel expansion.

CCFD uses the mode of traversing and allocating blocks to communicate. Each process traverses all block numbers. Only the local blocks are calculated, while the nonlocal blocks are used for asynchronous communication according to the actual communication needs. For example, when process 1 communicates with process 2, each process packs the mesh belonging to the adjacency surface relationship with nonblocking transmissions and then directly performs subsequent calculations. It does not detect whether the communication has been completed until the communication data are used locally. If the communication is completed, it can be used directly; otherwise, wait for the communication to complete. In Figure 9, B4 and B6 are the respective senders and receivers of data. Under nonblocking communication, B4 and B6 only need to send data to perform subsequent calculations and do not need to wait for the completion of data reception.

In the actual calculation of CCFD, there is a large amount of small and frequent communication, such as the detection of the local subblock, the updating of the local subblock, the updating of the flow field data, the calculation of the turbulent model, and the multigrid up-down interpolation process, which occurs after the block is read. These small and high-frequency communications are very time-consuming and affect the parallel efficiency of the processor. To solve these problems, we generally use methods such as collective communication, data prefetching, and computational alternative communication. Collective communication packs the number of high-frequency communication times in order to reduce the number of communication requests, which in turn reduces the communication time. The aim of prefetching technology is to obtain data in advance in each calculation iteration and hide the communication time. The alternative method of calculation refers to using calculations to complete the local results, which can

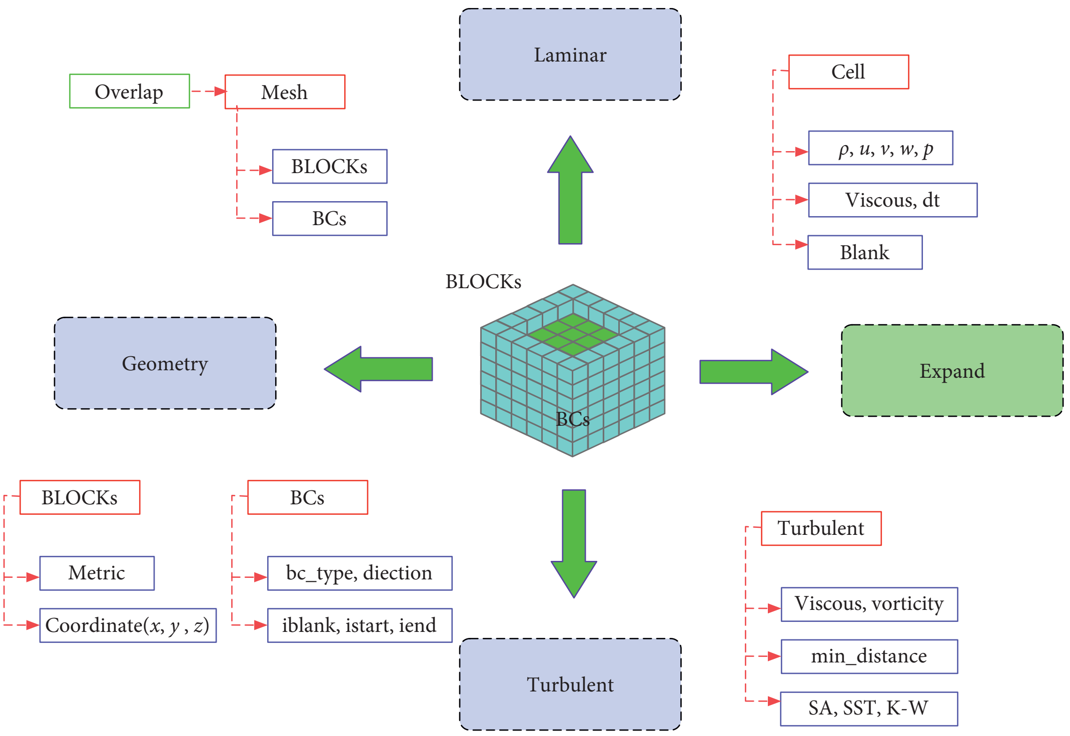


FIGURE 8: Data structure.

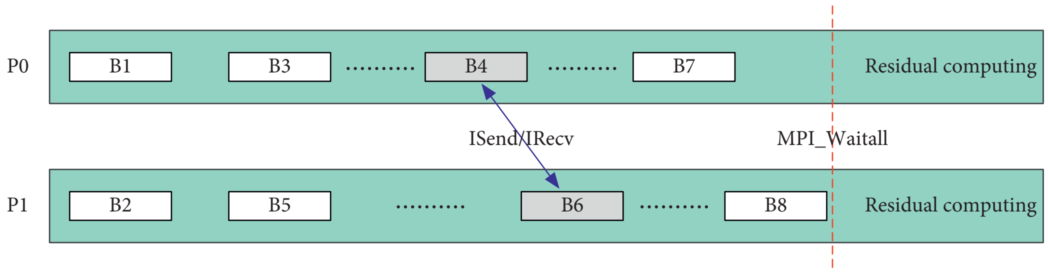


FIGURE 9: Nonblocking communication mode.

improve data utilization and reduce communication times and data volume.

When there are many uneven blocks in the local process, the program is prone to idle waiting in the blocking or sequential traversal execution mode. For example, the small block quickly ends the calculation and enters the communication. Large blocks have not yet entered the communication, and the data are not ready, which will cause a long wait. The overlap of calculation and communication is shown in Figure 10. Each communication request ends after the data are sent, and the detection of the completed communication is not performed until the data to be received are used in the calculation process. For example, processor P1 and processor P2 send data to each other, but these data are not necessarily used immediately once the communication ends. In this case, we can send the data to the background for continued processing, and the processor can perform subsequent calculations until the data are used and whether they have been received is detected. This method allows data communication and calculation to be performed at the same time. Because the communication time is hidden under the calculation time, it is called the overlapping technology of calculation and communication.

The design and selection of communication modes largely determine the computing efficiency and parallel scalability of parallel software. Different computing modes and methods and data structures require the design of a reasonable and flexible communication mode according to the actual situation. CCFD parallel software adopts flexible communication modes and carries out different parallel optimization designs for different calculation models and methods. Moreover, it also presents the modular design for the general data communication methods, which greatly simplifies the process of program programming and maintenance and achieves ideal parallel efficiency and data utilization.

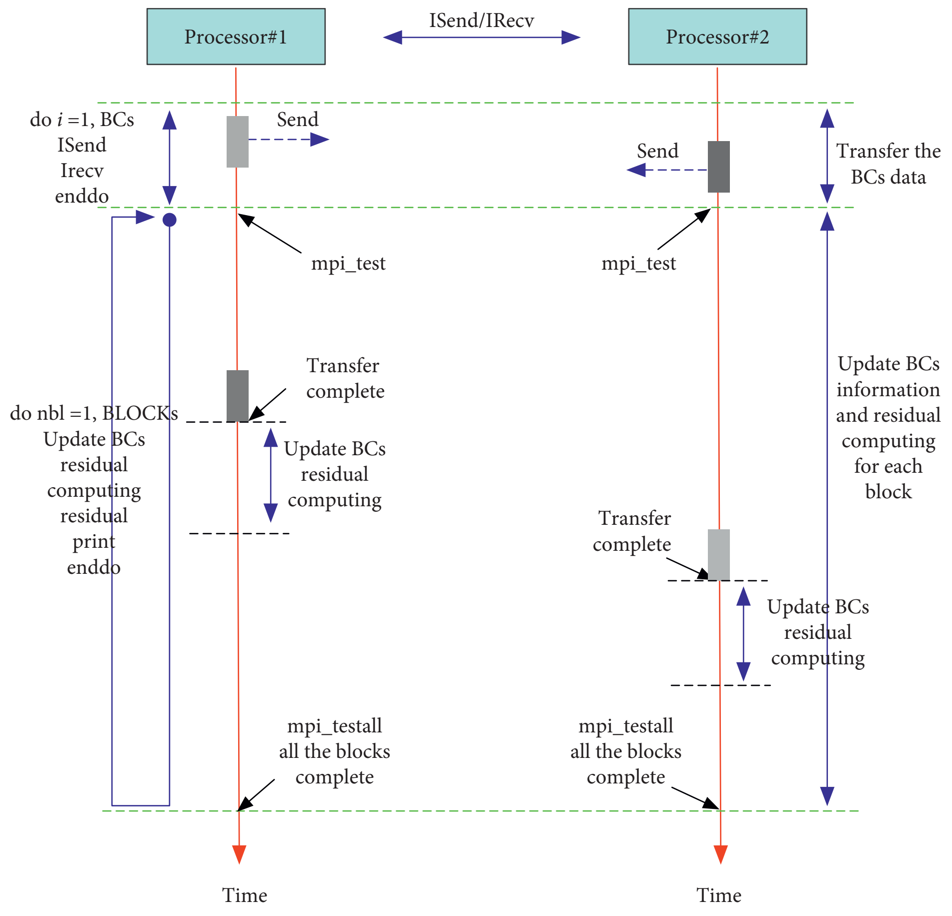


FIGURE 10: Communication and computation overlap.

4. Optimization on Sunway TaihuLight Heterogeneous Platforms

Sunway TaihuLight is the first 100P heterogeneous system supercomputer in China. It was developed by China's National Research Center of Parallel Computer Engineering & Technology (NRCPC) and is installed at the National Supercomputing Center in Wuxi province. It is powered exclusively by Sunway's SW26010 processors, with an HPL mark of 93.0 petaflops, and kept its number three spot in the TOP500 list of supercomputers [33]. The SW26010 multicore processor is composed of four computing core groups (CGs), and the four CGs are interconnected at high speed through the on-chip network, as shown in Figure 11. Each CG consists of a management processing element (MPE) and a multicore array containing 64 computing processing elements (CPEs). MPE can directly access memory, while CPE can directly access main memory and supports DMA batch data transfer [34].

The SW26010 processor has slow memory access and large memory access instruction delay, a memory access problem that has seriously affected the performance of the CFD solver. The CCFD solver has been transplanted and optimized in Sunway TaihuLight, and the program memory access problem has been solved in three ways: using high-speed storage instead of main memory access; eliminating or reducing memory access operations; and hiding memory access time. The optimization work in this paper is divided into the following aspects. We use MPI to achieve parallel communication between the core groups and use Athread to achieve the master-slave core parallelism in the core group. The data of the slave core are stored in the core group, the subarea is partitioned, and DMA batch transfer eliminates main memory fetch operations. The matrix data continuity access is optimized. The fast Carmack algorithm is used. The vectorized calculation and manual assembly instruction are rearranged. The use of a double cache method to cover fetch time significantly improves the parallel efficiency.

4.1. *Data Partitioning and DMA Parameter Tuning.*

CCFD is optimized based on the memory size of SW26010 and the performance of DMA to achieve domain decomposition and DMA parameter tuning. The block division of the many-core group calculation grid used by CCFD is shown in Figure 12. Using a $32 * 32 * 32$ block as an example, in order to efficiently use the storage space of CPE, 64 slave core blocks divide the grid data along the X-Y cross section. Then, the divided block data are allocated to 64 slave cores, and the 64 slave cores execute the same calculation process synchronously to complete a flow field calculation of the entire block.

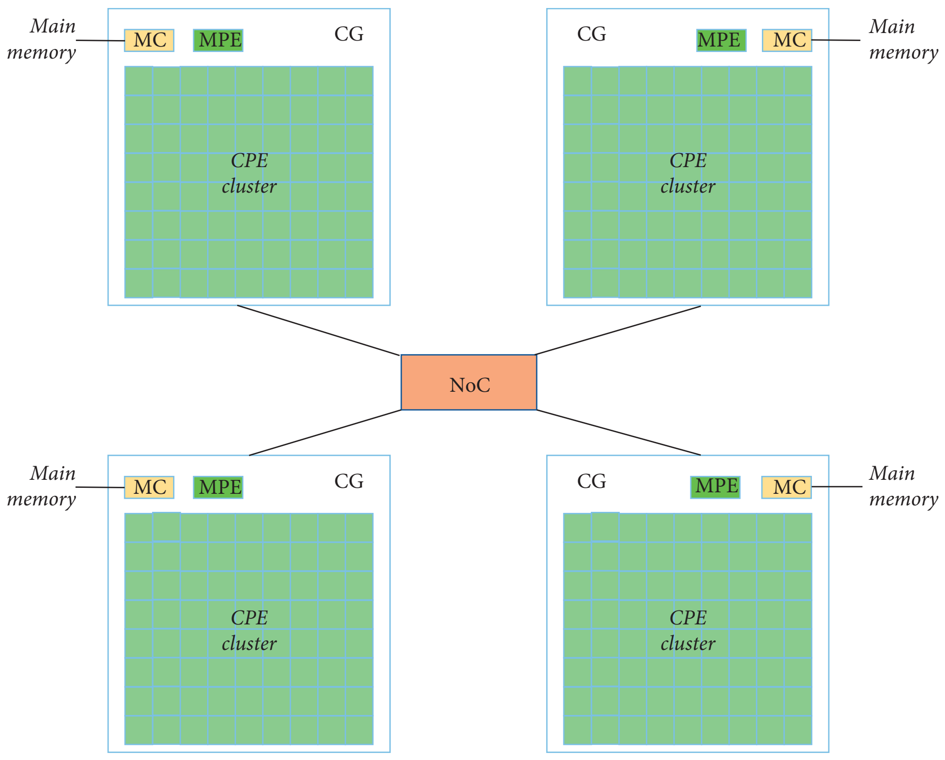


Figure 11: General architecture of the SW26010 processor.

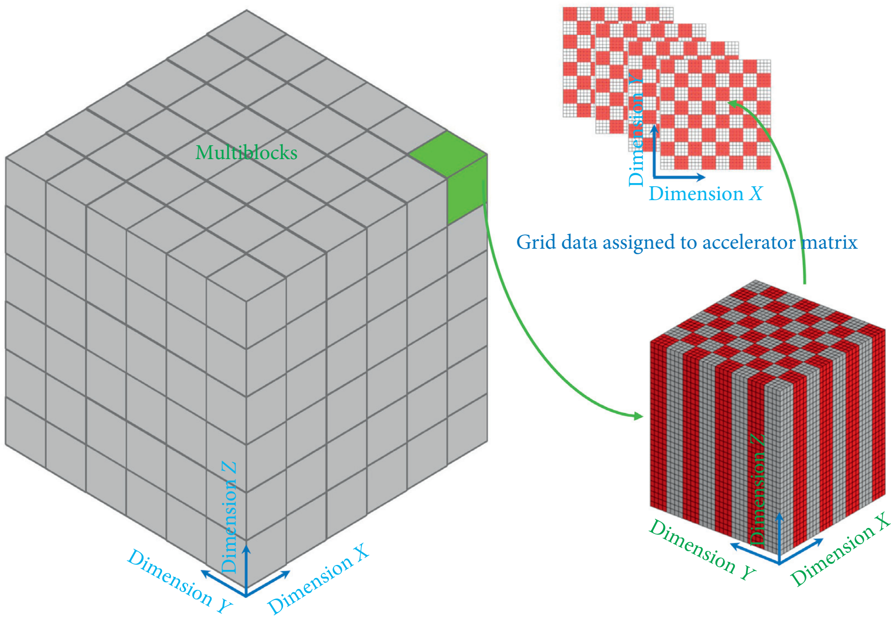


Figure 12: Block's domain decomposition for CPE slave group.

In actual programming, the size partition and direction selection of computational subdomains need to be further tuned, combined with the calculation process and the performance of the DMA. The local data memory (LDM) of the core processing unit is 64 kB, and in the actual flow field calculation, the size of the storage space required for different calculation processes ranges from 12 kB to more than 100 kB. Based on this, the temporarily stored LDM data need to strictly control the calculation scale, flexibly utilize DMA batch transmission, and eliminate the fetch operation to obtain data from the memory, thereby reducing the fetch time. In Figure 11, DMA is called by slave core. X is the lowest dimension of the data block, Y is the secondary dimension, and Z is the highest dimension. The data along the highest dimension are allocated to the same slave processor, and they are calculated along the Z-axis. The data transmitted to local are calculated cyclically by the slave core processor to complete the iterative solution of the flow field of a single block.

4.2. Register Communication Optimization. After the block subarea data are evenly distributed to the slave core arrays, the slave cores that are physically located adjacent to each other on the on-chip bus have the adjacent global grid data. The SW26010 processor supports the same columns of slave core array registers performing point-to-point discrete communication. The aggregate bandwidth of the register communication theory is much larger than the theoretical aggregate bandwidth of the DMA. This paper uses register point-to-point communication to obtain the boundary data stored on the adjacent slave cores on the X-Y plane in the slave core array group. Moreover, the register communication and the asynchronous mechanism of arithmetic operation are used to fill the instructions that have nothing to do with the communication data between the RLC_GET/RLC_PUT instructions, to cover the communication time between the register communications.

As shown in Figure 13, using the XY plane as an example, the green dots indicate the grid data stored on the slave core LDM. The flow field calculation of the local cell is performed on each slave core, and the update result is returned. However, the local cells are calculated using the data of cells in four directions which contain part of halo cells in Figure 13 and assigned to adjacent slave calculation local cell to complete the calculation of the internal point. For the three-dimensional 13-point model discussed in this paper, after reading one layer of calculation subregion data blocks from the core, it is necessary to read two layers of halo data blocks from the surrounding four directions to complete the calculation.

The SW26010 supports register communication between peers and the same column. On the accelerator array, the registers of accelerators in the same row or the same column realize interconnected communication and can transmit 128 bits of data each time. As shown in Tables 1 and 2, the peer processors place the data in the register and specify the target number of the peer processor and then send the data to the destination. The target processor will collect the data over the Internet and save it to the register of the processor, realizing the communication operation of the register.

4.3. SIMD Calculation and Assembly Instruction Optimization. The SW26010 supports the 256-bit SIMD instruction set, which can perform four double-precision or eight single-precision data operations simultaneously. To ensure the use of SIMD, the calculation process and the address alignment of the variable array need to be alignment-adjusted. At the same time, SIMD_LOAD/SIMD_STORE and the calculation instruction are manually adjusted under the premise of ensuring that the calculation is correct, so that the LOAD/STORE and the calculation time are partially hidden. Data access and storage processing and arithmetic of the SW26010 are operated on two pipelines. To achieve the overlap of the data fetch and arithmetic operations, this paper analyzes the order of instructions generated by the compiler, through manually adjusting the order of instructions, and implements the overlap of data fetch load/store operations and data-independent operations. This method can eliminate data dependencies and improve code efficiency.

To further improve the calculation performance, the calculation code is disassembled. As shown in Table 3, by observing the order of the instruction execution sequence after disassembly, we found the memory access and calculation instructions are completely separated, and nearly six access instructions are issued on the pipeline before one calculation instruction is launched. After ensuring that the data are not relevant, we use inline assembly functions and handwritten assembly codes instead of function interface codes. As shown in Table 4, after manually adjusting the assembly code, the overlap of memory access and calculation instruction time is realized.

5. CCFD Application and Results

To verify the architecture and parallel performance of the CCFD software, this paper selects the DLR-F6 wing-body assembly, the CT-1 standard model, and the ultra-large-scale M6 wing model for testing. The test platforms are the Yuan supercomputer at the Computer Network Information Center in Beijing and the Sunway TaihuLight supercomputer at the National Supercomputing Center in Wuxi.

5.1. Optimization Results on Sunway TaihuLight. Using a block of size $32 * 32 * 32$ as the optimization case, the block data were divided into the $8 * 8$ slave cluster on CGs, the main calculation function and hot spots of CCFD were optimized, the flow field calculation was accelerated, and the entire CFD simulation process was completed. Figure 14 shows the SIMD optimization provided by Sunway's compiler in the Roe scheme calculation part, and the comparison of the average cycles number before and after inline assembly instruction optimization. The optimization effect of SIMD and assembly instructions is significant, in that implementing the overlap of fetching and manual instruction adjustment further optimizes the calculation.

Now that CCFD has completed the acceleration optimization of the main functional modules, including flux calculation, turbulent model, and time advancement method, and the slave core acceleration ratio has finally reached more than 30 to 1. Figure 15 shows the comparison of calculation time before and after the optimization of the main calculation function module of CCFD.

5.2. DLR-F6 Wing Body Assembly. The German Aerospace Center's DLR-F6 wing-body assembly is a typical example of a transonic transport aircraft, which verifies the CFD solver's ability to calculate complex shapes [42]. The calculation conditions are Mach number $\mathrm{Mach} = 0.7510$, Reynolds number $\mathrm{Re} = 0.0212465 \times 106$, angle of attack $\alpha = 1.003^\circ$, and side slip angle $\beta = 0^\circ$. The calculation method uses Roe scheme, LU_ADI time advancement method, three-layer mesh sequencing strategy, and the Spalart-Allmaras

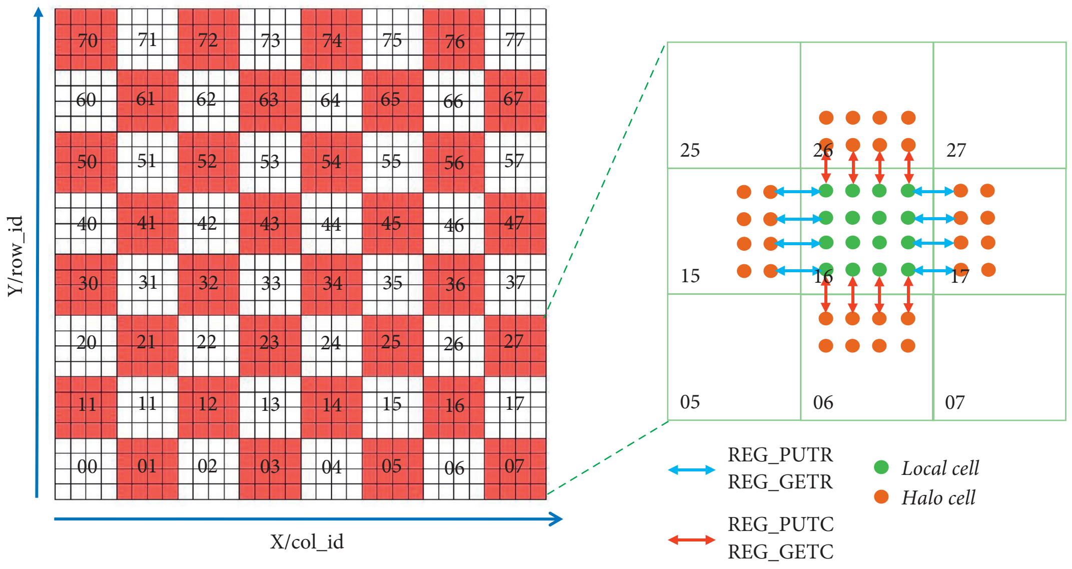


FIGURE 13: Data dependence and register communication from slave cores.

TABLE 1: Send data to the same row register.

asm volatile(\
"*vldd* $0, 0(%0)*n**t*"
"*vldd* $1, 0(%1)*n**t*"\
"*vldd* $2, 0(%2)*n**t*" "*putc* $0, %4*n**t*"\
"*vldd* $3, 0(%3)*n**t*" "*putc* $1, %4*n**t*"
"*putc* $2, %4*n**t*"\
"*putc* $3, %4*n**t*"\
\)

TABLE 2: Receive data from the same row register.

asm volatile(\
"*getc* $1*n**t*"\
"*getc* $2*n**t*"\
"*getc* $3*nt*" "*vstd* $1, 0(%0)*n**t*"\
"*getc* $4*n**t*" "*vstd* $2, 0(%1)*nt*"\
"*vstd* $3, 0(%2)*n**t*"\
"*vstd* $4, 0(%3)*n**t*"
\)

TABLE 3: Order of instructions generated by the compiler.

(1). "vldd $0, 0(%2)\n\t"
(2). "vldd $1, 32(%2)\n\t"
(3). "vldd $2, 64(%2)\n\t"
(4). "vldd $3, 0(%3)\n\t"
(5). "vldd $4, 32(%3)\n\t"
(6). "vldd $5, 64(%3)\n\t"
(7). "vmuld $0, $3, $0\n\t"
(8). "vsubd $1, $4, $1\n\t"
(9). "vdivd $2, $5, $2\n\t"
(10). "vmad $0, $1, $2, $2\n\t"

turbulent model. Figure 16 shows the pressure nephogram of the entire assembly.

Figure 17 shows the comparison of the experimental results and calculated pressure coefficients of the different

TABLE 4: Out-of-order execution of instructions.

(1). "vldd $0, 0(%2)\n\t"
(2). "vldd $3, 0(%3)\n\t"
(3). "vmuld $0, $3,$0\n\t"
(4). "vldd $1, 32(%2)\n\t"
(5). "vldd $4, 32(%3)\n\t"
(6). "vsubd $1, $4,$1\n\t"
(7). "vldd $2, 64(%2)\n\t"
(8). "vldd $5, 64(%3)\n\t"
(9). "vdivd $2, $5, $2\n\t"
(10). "vmad $0, $1, $2, $2\n\t"

positions of the DLR-F6 wing. The CCFD calculation results are consistent with the experimental results, which verifies that CCFD can simulate the flow field calculation of a complex-shaped aircraft.

5.3. CT-1 Standard Model. The CT-1 is a model of high-angle static aerodynamic characteristics released by the China Aerodynamics Research and Development Center in 2005 [43]. We focus on the numerical simulation capabilities of the CFD solution software at high angles of static aerodynamic characteristics. Figure 18 shows the pressure nephogram at different angles of attack.

Figure 19 shows a comparison of the experimental results and the drag coefficients for each angle of attack. By comparing the experimental results with the calculation results of the static aerodynamic characteristics of the CT-1 standard model at high angles of attack, the CCFD calculation results are consistent with the experimental results, which verifies the numerical simulation ability of CCFD to solve the static aerodynamic characteristics at high angle of attack.

5.4. Ultra-Large-Scale Parallel Testing of M6 Wing. The Onera M6 wing standard calculation model [37] uses 650 million grids and 28,160 blocks. It verifies the parallel solver capabilities above

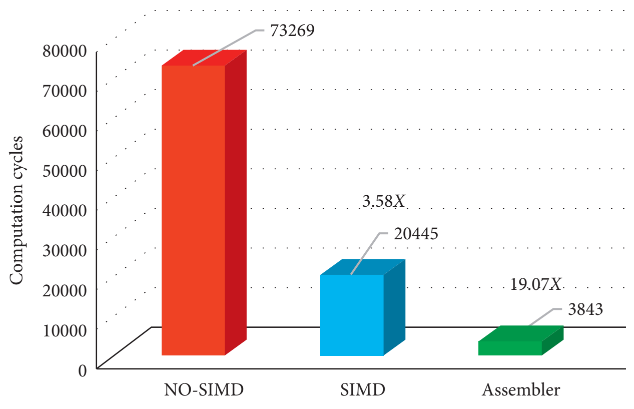


Figure 14: Comparison of cycle before and after SIMD and assembler optimization.

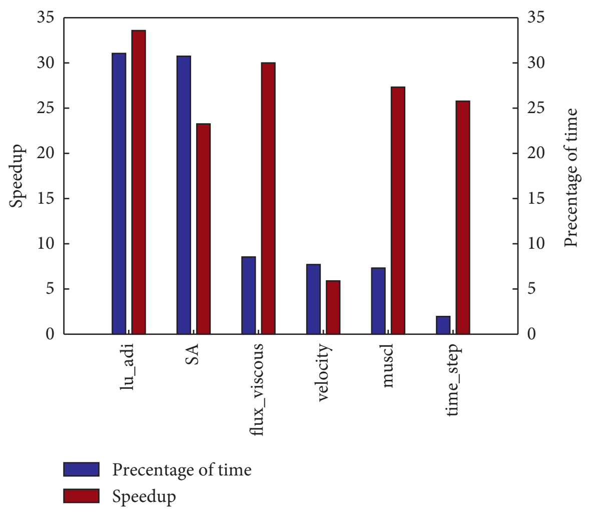


Figure 15: Master-slave core acceleration times and time ratio of the main calculation module after optimization.

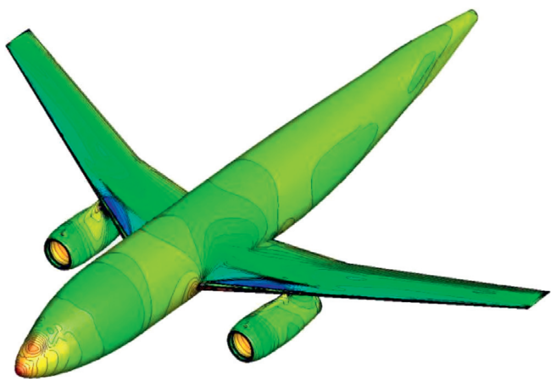


Figure 16: Pressure nephogram for DLR-F6.

the 10,000-core level and the parallel expansion performance of CCFD. The calculation conditions are Mach number Mach = 0.840, Reynolds number Re = 21.70 × 106, angle of attack $\alpha = 3.06^\circ$, and side slip angle $\beta = 0^\circ$. The calculation method uses Roe scheme, LU_ADI time advancement method, and the Spalart-Allmaras turbulent model.

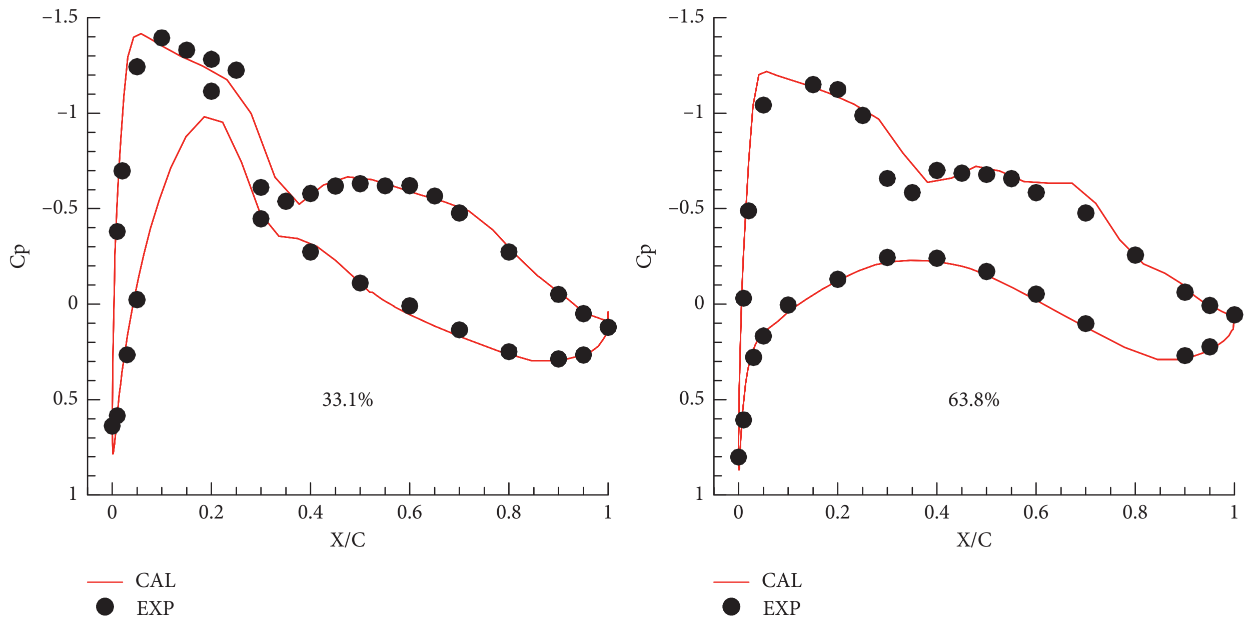


Figure 17: Comparison of the pressure curves of each wing position and the nacelle section with the experimental data.

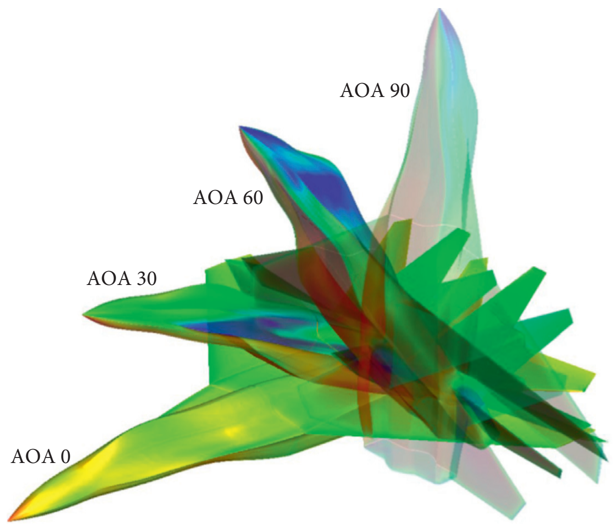


Figure 18: Pressure nephogram at different angles of attack for CT-1.

CCFD is based on the Sunway TaihuLight heterogeneous platform, with its core groups communicating with each other using MPI, and 64 slave cores in the core group accelerated in parallel using Athread. Figure 20 shows that the parallel efficiency of CCFD reached 60% under the Sunway TaihuLight heterogeneous platform with 13,000 cores and 500,500 cores. The parallel result test indicates that the super-large-scale parallel computing scalability of CCFD has met the design requirements.

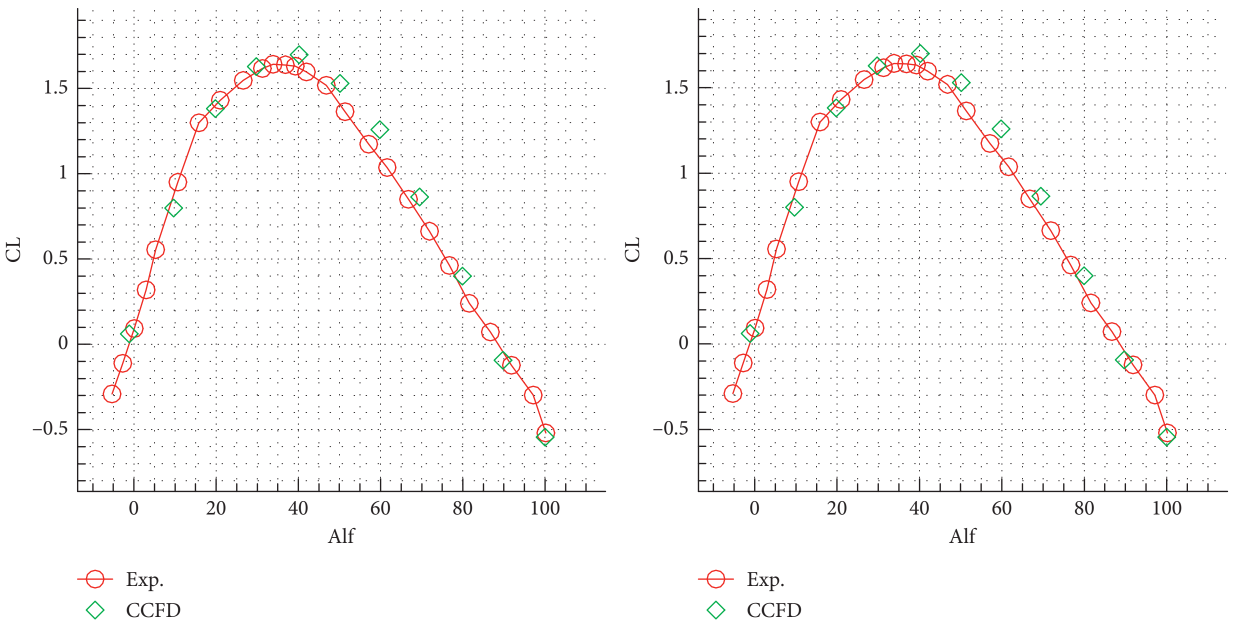


Figure 19: Lift and drag coefficients at different angles of attack for CCFD.

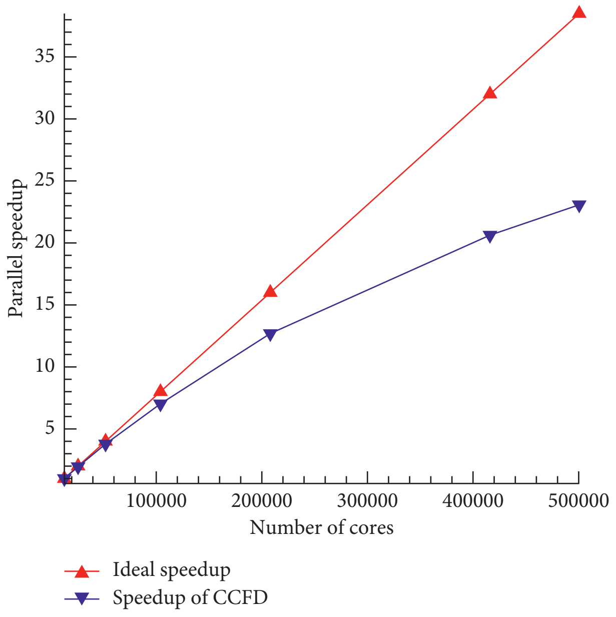


Figure 20: Parallel efficiency of large-scale test for CCFD.

6. Conclusions and Future Work

Using the multiblock structure grid parallel technology of domain decomposition, we design the parallel software framework, software process, parallel data structure, and communication mode of parallel CFD solver software. The designed CCFD can be used for large-scale mega-core parallel computing tasks and has good parallel expansion ability and parallel efficiency. The overdecomposition load balancing strategy for CCFD guarantees the load balancing performance of computing and communication. The use of a mesh sequencing method based on multigrid technology can accelerate the iterative convergence speed. We perform the optimizations on DMA, SIMD, assembly instruction rearrangement, and double buffering in the Sunway TaihuLight heterogeneous architecture. The super-large computational scale test with a maximum of 505,000 cores in parallel across the core group achieved a parallel efficiency of 60% based on 13,000 cores. In the future, we will use direct numerical simulation of turbulence models to achieve more realistic geometric flow simulations, increase the scale of cases and parallelism, and achieve more efficient operation of parallel CFD software in heterogeneous systems.

Data Availability

The grid 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 regarding the publication of this paper.

Acknowledgments

This work was supported by a grant from the National Key R&D Program of China (No. 2019YFB1704204), the National Natural Science Foundation of China (Nos. 61702438 and 11502267), the Chinese Academy of Sciences strategic pilot technology special (Class B, XDB22020102, and Class C, XDC01040100), and the Informatization Special Project of Chinese Academy of Sciences (No. XXH13506-204).

References

[1] J. Blazek, *Computational Fluid Dynamics: Principles and Applications*, Elsevier Science Publication, Amsterdam, Netherlands, 2005.

[2] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics*, Pearson Education Limited, London, UK, 2nd edition, 2007.

[3] P. Wesseling, "Principles of computational fluid dynamics," in *Computational Mathematics*, Vol. 29, Springer-Verlag, Berlin, Germany, 2001.

[4] C.-L. Lin, M. H. Tawhai, G. Mclennan, and E. A. Hoffman, "Computational fluid dynamics," *IEEE Engineering in Medicine and Biology Magazine*, vol. 28, no. 3, pp. 25–33, 2009.

[5] F. He, X. Dong, N. Zou, W. Wu, and X. Zhang, "Structured mesh-oriented framework design and optimization for a coarse-grained parallel CFD solver based on hybrid MPI/OpenMP programming," *The Journal of Supercomputing*, vol. 76, no. 4, pp. 2815–2841, 2020.

[6] A. Gel, E. J. Hu, E. Ould-Ahmed-Vall, and A. A. Kalinkin, "Modernization and optimization of a legacy open-source CFD code for high-performance computing architectures," *International Journal of Computational Fluid Dynamics*, vol. 31, no. 2, pp. 122–133, 2017.

[7] S. Li, R. Paoli, and M. D'Mello, "Scalability of OpenFOAM density-based solver with Runge–Kutta temporal discretization scheme," *Scientific Programming*, vol. 2020, Article ID 9083620, 11 pages, 2020.

[8] F. Palacios, T. D. Economon, and J. J. Alonso, "Large-scale aircraft design using SU^2," in *Proceedings of the AIAA Aerospace Sciences Meeting*, Kissimmee, FL, USA, January 2015.

[9] F. Palacios, A. Alonso, K. Duraisamy et al., "An open-source integrated computational environment for multi-physics simulation and design," in *Proceedings of the AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Grapevine, TX, USA, January 2013.

[10] S. Yakubov, B. Cankurt, M. Abdel-Maksoud, and T. Rung, "Hybrid MPI/OpenMP parallelization of an Euler-Lagrange approach to cavitation modelling," *Computers & Fluids*, vol. 80, pp. 365–371, 2013.

[11] Y.-X. Wang, L.-L. Zhang, W. Liu, X.-H. Cheng, Y. Zhuang, and A. T. Chronopoulos, "Performance optimizations for scalable CFD applications on hybrid CPU + MIC heterogeneous computing system with millions of cores," *Computers & Fluids*, vol. 173, pp. 226–236, 2018.

[12] D. Cheng, C. Xu, B. Cheng, M. Xiong, X. Gao, and X. Deng, "Performance modeling and optimization of parallel LU-SGS on many-core processors for 3D high-order CFD simulations," *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2506–2524, 2017.

[13] O. Bessonov, "OpenMP parallelization of a CFD code for multicore computers: analysis and comparison," in *Proceedings of the 11th International Conference on Parallel Computing Technologies, PaCT 2011*, Kazan, Russia, September 2011.

[14] A. Amritkar, S. Deb, and D. Tafti, "Efficient parallel CFD-DEM simulations using OpenMP," *Journal of Computational Physics*, vol. 256, pp. 501–519, 2014.

[15] V. Moureau, P. Domingo, and L. Vervisch, "Design of a massively parallel CFD code for complex geometries," *Comptes Rendus Mécanique*, vol. 339, no. 2-3, pp. 141–148, 2011.

[16] B. Kong, R. O. Fox, H. Feng et al., "Euler-euler anisotropic Gaussian mesoscale simulation of homogeneous cluster-induced gas-particle turbulence," *AICHE Journal*, vol. 63, no. 7, pp. 2630–2643, 2017.

[17] A. Capecelatro, A. Cauble-Chantrenne, A. Jundt et al., "Running large-scale CFD applications on Intel-KNL-based clusters," *International Journal for Numerical Methods in Fluids*, vol. 86, no. 11, pp. 699–716, 2018.

[18] L. Lapichino, A. Quarteroni, and G. Rozza, "Reduced basis method and domain decomposition for elliptic problems in networks and complex parametrized geometries," *Computers & Mathematics with Applications*, vol. 71, no. 1, pp. 408–430, 2016.

[19] S. Badia and H. Nguyen, "Balancing domain decomposition by constraints and perturbation," *SIAM Journal on Numerical Analysis*, vol. 54, no. 6, pp. 3436–3464, 2016.

[20] S. Badia, A. F. Martín, and J. Principe, "Multilevel balancing domain decomposition at extreme scales," *Siam Journal on Scientific Computing*, vol. 38, no. 1, pp. C22–C52, 2016.

[21] O. S. Lawlor, S. Chakravorty, T. L. Wilmarth, and N. Choudhury, "ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications," *Engineering with Computers*, vol. 22, no. 3-4, pp. 215–235, 2006.

[22] G. T. Abraham, A. James, and N. Yaacob, "Priority-grouping method for parallel multi-scheduling in grid," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 943–957, 2015.

[23] T. Shimokawabe, T. Aoki, and N. Onodera, "High-productivity framework for large-scale GPU/CPU stencil applications," *Procedia Computer Science*, vol. 80, pp. 1646–1657, 2016.

[24] I. Z. Reguly and G. R. Mudalige, "Productivity, performance, and portability for computational fluid dynamics applications," *Computers and Fluids*, vol. 199, Article ID 104425, 2020.

[25] J. Wang, X. Lv, and X. Chen, "Comparative analysis of list scheduling algorithms on homogeneous multi-processors," in *Proceedings of the IEEE International Conference on Communication Software & Networks*, IEEE, Beijing, China, 2016.

[26] L. Chai, A. Hartono, and D. K. Panda, "Designing high performance and scalable MPI intra-node communication support for clusters," in *Proceedings of the IEEE International Conference on Cluster Computing*, IEEE, Tampa, FL, USA, November 2006.

[27] Q. Tang, L.-H. Zhu, L. Zhou, J. Xiong, and J.-B. Wei, "Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 115–127, 2020.

[28] A. Kayi, T. El-Ghazawi, and G. B. Newby, "Performance issues in emerging homogeneous multi-core architectures," *Simulation Modelling Practice and Theory*, vol. 17, no. 9, pp. 1485–1499, 2009.

[29] S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 87, pp. 80–90, 2016.

[30] H. A. Tokel, G. Alirezaei, S. Baig, and R. Mathar, "An optimization framework for planning of WAMS with a heterogeneous communication network," in *Proceedings of the IEEE International Conference on Smart Grid Communications*, IEEE, Sydney, Australia, November 2016.

[31] L. Deng, F. H. Bai, F. H. Bai, and Q. Xu, "CPU/GPU computing for an implicit multi-block compressible Navier-Stokes solver on heterogeneous platform," *International Journal of Modern Physics: Conference Series*, vol. 42, Article ID 1660163, 2016.

[32] W. Cao, C.-F. Xu, Z.-H. Wang, L. Yao, and H.-Y. Liu, "CPU/GPU computing for a multi-block structured grid based high-order flow solver on a large heterogeneous system," *Cluster Computing*, vol. 17, no. 2, pp. 255–270, 2013.

[33] TOP500 Supercomputing Sites (EB/OL), 2019, https://www.top500.org.

[34] Z. Xu, J. Lin, and S. Matsuoka, "Benchmarking SW26010 many-core processor," in *Proceedings of the 2017 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, IEEE, Orlando, FL, USA, pp. 743–752, May 2017.

[35] H. Fu, J. Liao, J. Yang et al., "The Sunway TaihuLight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, pp. 113–128, 2016.

[36] C. Yang, W. Xue, H. Fu et al., "10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, Salt Lake City, UT, USA, 2016.

[37] Z. Johan, K. K. Mathur, S. L. Johnsson, and T. J. R. Hughes, "A case study in parallel computation: viscous flow around an ONERA M6 wing," *International Journal for Numerical Methods in Fluids*, vol. 21, no. 10, pp. 877–884, 1995.

[38] W. S. Brainerd, "Object-oriented programming," in *Guide to Fortran 2008 Programming*, Springer, Berlin, Germany, 2015.

[39] W. Deng, J. Xu, Y. Song, and H. Zhao, "An effective improved co-evolution ant colony optimization algorithm with multi-strategies and its application," *International Journal of Bio-Inspired Computation*, vol. 20, no. 5, pp. 1–10, 2020.

[40] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, "Improving graph partitioning for modern graphs and architectures," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, ACM, Austin TX, USA, November 2015.

[41] Z. Shang, "Large-scale CFD parallel computing dealing with massive mesh," *Journal of Engineering*, vol. 2013, Article ID 850148, 6 pages, 2013.

[42] J. C. Vassberg, A. J. Sclafani, and M. A. DeHaan, "A wing-body fairing design for the DLR-F6 model: a DPW-III case study," Report No. AIAA-2005-4730, AIAA, Reston, VA, USA, 2005.

[43] Y. Wang, G. Wang, and Z. Chen, "Numerical simulation of static aerodynamic characteris of CT-1 model at high angles of attack," *Acta Aeronautica Et Astronautica Sinica*, vol. 29, no. 4, pp. 859–865, 2008.