applied sciences

MDPI

Article

swHPFM: Refactoring and Optimizing the Structured Grid Fluid Mechanical Algorithm on the Sunway TaihuLight Supercomputer

Jingbo Li [1], Xingjun Zhang [1,*], Jianfeng Zhou [1], Xiaoshe Dong [1], Chuhua Zhang [2] and Zeyu Ji [1]

[1] School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China; lijingbo17@stu.xjtu.edu.cn (J.L.); jeffzhou@stu.xjtu.edu.cn (J.Z.); xsdong@mail.xjtu.edu.cn (X.D.); zeyu.ji@stu.xjtu.edu.cn (Z.J.)

[2] School of Energy and Power Engineering, Xi'an Jiaotong University, Xi'an 710049, China; chzhang@mail.xjtu.edu.cn

* Correspondence: xjzhang@xjtu.edu.cn

Received: 13 November 2019; Accepted: 17 December 2019; Published: 20 December 2019

Featured Application: The proposed algorithm and optimization methods can be used directly in the high-precision and large-scale fluid mechanical algorithm of the Sunway TaihuLight supercomputer and computing platforms with a heterogeneous many-core architecture.

Abstract: Fluid mechanical simulation is a typical high-performance computing problem. Due to the development of high-precision parallel algorithms, traditional computing platforms are unable to satisfy the computing requirements of large-scale algorithms. The Sunway TaihuLight supercomputer, which uses the SW26010 processor as its computing node, provides a powerful computing performance for this purpose. In this paper, the Sunway hierarchical parallel fluid machinery (swHPFM) framework and algorithm are proposed. Using the proposed framework and algorithm, engineers can exploit the parallelism of the existing fluid mechanical algorithm and achieve a satisfactory performance on the Sunway TaihuLight. In the framework, a suitable mapping of the model and the system architecture is developed, and the computing power of the SW26010 processor is fully utilized via the scratch pad memory (SPM) access strategy and serpentine register communication. In addition, the framework is implemented and tested by the axial compressor rotor simulation algorithm on a real-world dataset with Sunway many-core processors. The results demonstrate that we can achieve a speedup of up to 8.2×, compared to the original ported version, which only uses management processing elements (MPEs), as well as a 1.3× speedup compared to an Intel Xeon E5 processor. The proposed framework is useful for the optimization of fluid mechanical algorithm programs on computing platforms with a heterogeneous many-core architecture.

Keywords: high-performance computing; sunway TaihuLight supercomputer; fluid machine algorithm; heterogeneous many-core architecture

1. Introduction

A fluid machine is a type of powerful machine that uses continuously rotating blades as its body to convert energy between working fluids (fluid, gas, liquid or a gas–liquid mixture) and shaft power. Fluid machines can be divided into axial flow, radial flow, mixed flow, and combined flow machines and mainly include vane compressors, blowers, fans, and pumps. They are widely used in the national defense military, aerospace, and national pillar industries, and they play an important role in many national economies [1]. The fluid mechanical simulation algorithm provides a large amount of instructive data for the fluid mechanical design, which can effectively reduce design costs. However,

due to the development of high-efficiency, high-precision, and high-expansion parallel algorithms for large-scale fluid machinery, the traditional computing platforms are unable to satisfy the computing requirements of the large-scale fluid mechanical algorithm. The Sunway TaihuLight supercomputer provides a powerful computing performance; its theoretical peak performance reaches 125.44 PFlop/s, and its high-performance linpack benchmark (HPL) performance is 93.01 PFlop/s [2]. However, it is difficult to obtain a good performance on SW26010 processors due to the limited local memory space of each computing processing element (CPE). What is more, normal C/C++ and FORTRAN codes can only use the management processing elements (MPEs) and cannot achieve the best possible performance. Fully utilizing Sunway TaihuLight's multilevel computing resources, in combination with the system architecture and algorithm, to exploit the parallelism of applications and to fully utilize the powerful computing power of its SW26010 processor is one of the main challenges in efficiently simulating fluid machinery.

At its inception, the simulation algorithm of fluid mechanics was limited by computer hardware performance, thereby resulting in a small parallel scale and low precision. The simulated model was geometrically regular and simple, and it was difficult to obtain valuable results that were comparable with those of engineering experiments. Thanks to the development of high-performance computing, the exponential growth of computing power provides favorable conditions for the large-scale parallel algorithm of fluid machinery. With the maturity of the parallel programming standards, such as Message Passing Interface (MPI) and Open Multi-Processing (OpenMP), the parallelization of the high-efficiency multiblade row and multichannel unsteady simulation have been realized [3]. The portability and scalability of GPU speedup methods have been demonstrated in radiative heat transfer calculations [4]. The MPI-OpenMP hybrid method has been used to conduct a scalable parallelization of a pseudospectral calculation of fluid turbulence [5]. The three-level parallel method of MPI, OpenMP, and Compute Unified Device Architecture (CUDA) has been used to extend the finite difference method (FDM) to solve the steady-state heat conduction equation in order to fully utilize the computing power of multicore and CUDA cores [6]. The best MPI-OpenMP distribution to accelerate the program on Intel Many Integrated Core (MIC) cluster has been researched [7]. In addition, in other fields of research, a novel method and tool for Sparse matrix-vector (SpMV) multiplication has been proposed [8]. A flexible parallel merge sorting for multicore architectures has been proposed to efficiently use all computing resources and increase the efficiency of sorting [9]. A real-time digital image correlation (DIC) has been implemented in GPUs [10]. The GPU-based 3-D motion tracking has been implemented to improve its computational efficiency [11]. A Spark-based parallel genetic algorithm (GA) has been proposed to obtain the optimal deployment of the underwater sensor network [12]. A compliant parallel mechanism (CPM) has been designed to provide a large load capacity and achieve micrometer-level positioning accuracy [13].

Various studies have been conducted on large-scale parallel program optimization for the Sunway TaihuLight. A nonlinear seismic simulation, based on the Sunway TaihuLight, has been implemented. Using register communication, SIMD and other optimization schemes, a performance of up to 18.9 PFlops has been realized [14]. A highly scalable full-implicit solver, based on the strong time-dependence problem of atmospheric dynamics, in which a memory-related block optimization of the local data memory (LDM) is utilized to reduce the data movement overhead, has been designed [15]. A performance-aware model for general sparse matrix-sparse matrix multiplication (SpGEMM) to select an appropriate compressed storage formats, according to the performance analysis of SpGEMM kernels, has been proposed [16]. The new projected entangled pair states method (PEPS++) for strongly correlated quantum many-body systems, based on a carefully designed tensor computation library for manipulating high-rank tensors and optimizing them by invoking various high-performance matrix and tensor operations, has been implemented [17]. A 50-m resolution earthquake simulation of the Wenchuan Earthquake (Ms 8.0 China) on the Sunway TaihuLight has been performed [18]. A sparse level tile data layout for controlling all data reuse and a producer-consumer pairing method have been proposed [19]. The training process of convolutional neural networks (CNNs), in which customized

deep learning (swDNN) and customized Caffe (swCaffe) by architecture-oriented optimization methods are proposed, have been optimized [20].

However, few optimization studies have been conducted on the fluid mechanical algorithm for the Sunway TaihuLight. The SunwayLB for the Lattice Boltzmann Method (LBM) has been proposed [21]. However, this study only optimized LBM and did not implement a specific fluid dynamics algorithm program.

Therefore, an efficient algorithm and framework for fluid machinery algorithm is proposed in this paper. For the SW26010 many-core processor, according to the characteristics of the application and based on the Athread library, an SPM access strategy-based direct memory access (DMA) and Serpentine register communication are proposed to fully utilize the slave computing resources. In addition, the framework is implemented and tested by the proposed axial compressor rotor algorithm on a real-world dataset. The proposed algorithm and framework have great significance for the efficient optimization of the fluid mechanical algorithm on computing platforms with the Sunway TaihuLight many-core architecture.

The remainder of this paper is organized as follows: The proposed algorithm framework and the optimization method are described in detail in Section 2. Next, Section 3 presents the results of the results and performance comparison analysis of the algorithm and swHPFM framework. In Section 4, comprehensive experiments are conducted to discuss and evaluate the performance of the algorithm swHPFM parallel framework in detail. Finally, the conclusions of this work are presented in Section 5.

2. Materials and Methods

2.1. SW26010 Processor Architecture and Analysis

The Sunway TaihuLight supercomputer was developed by the National Parallel Computer Engineering Technology Research Center. Each computer node contains a SW26010 heterogeneous many-core processor. Each processor is composed of four core groups (CGs), as illustrated in Figure 1. Each CG contains an MPE for latency-sensitive tasks and a CPE cluster of 64 CPEs that are organized as an 8×8 mesh for throughput-sensitive tasks.

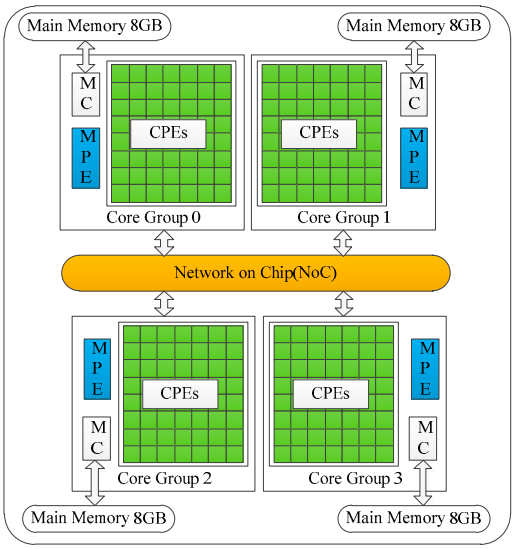


Figure 1. General architecture of the SW26010 processor. Each processor is composed of four core groups (CGs). Each CG contains an management processing element (MPE) for latency-sensitive tasks and a computing processing element (CPE) cluster of 64 CPEs, which are organized as an 8×8 mesh for throughput-sensitive tasks.

Analyzing the computing performance, both the MPE and CPE support vectorization instructions and the fused multiply add (FMA) operation, with a frequency of 1.45-GHz. In addition, the MPE supports dual floating-point double precision (DP) pipelines, and each slave core supports one floating-point DP pipeline. Therefore, the peak performance of the double precision on MPE is 23.2 floating-point operations per second (GFLOPS), and the peak performance of the CPE cluster is 742.4 GFLOPS. In an SW26010 processor, the overall performance of the CPE cluster is 32× that of the MPE [18]. In addition, the MPE supports complete interruption handling, memory management, superscalar processing, and out-of-order execution. Therefore, the MPE performs well in data communication and task management. The CPE is responsible for data parallel computing. Therefore, with the MPE and CPEs working together and distributing kernel functions to CPEs, the computing performance of the SW26010 processor could fully exploited.

In terms of the memory architecture, the SW26010 processor has 32 GB of main memory, and each CG has 8 GB of local main memory. In addition, each MPE has a 32-KB L1 data cache and a 256-KB L2 instruction/data cache, and each CPE has its own 16-KB L1 instruction cache and a 64-KB MPE, whose access speed is equal to that of the L1 cache. The CPE has two types of memory access to the main memory: DMA and global load/store (Gload/Gstore). According to a Stream benchmark test, when being accessed by Gload/Gstore instructions, the Copy, Scale, Add, and Triad maximum bandwidths are only 3.88 GB/s, 1.61 GB/s, 1.45 GB/s, and 1.48 GB/s, respectively. Correspondingly, when using DMA PE mode, the maximum Copy bandwidth reaches 27.9 GB/s, the maximum Scale bandwidth is 24.1 GB/s, the Add bandwidth is 23.4 GB/s, and the Triad bandwidth is 22.6 GB/s [22]. According to the above data, the DMA prefers transferring massive data from the main memory to the SPM of the CPE, and Gload/Gstore prefers transferring small and random data between the main memory and the SPM. Moreover, due to the space limitation of 64 KB of the SPM, redundant data cannot be stored, and a user-controlled buffering strategy is needed to improve the data reuse rate in order to fully utilize the LDM space. In addition, one of the key features of the SW26010 processor is that the CPE cluster offers low-latency register data communication within the 8 × 8 CPE mesh. According to Table 1, the delay of point-to-point communication is approximately 10 cycles, with a delay of approximately 14 cycles for row and column broadcasts [23]. However, due to the limitations of the mesh hardware, the data in the register can only be communicated between the CPEs in the same row or column. This type of communication effectively reduces the delay caused by accessing the main memory. It is important to fully utilize SPM and register communication when optimizing the slave-core in parallel.

Table 1. Delay of the Register Communication.

Register Communication	Cycles
Point-to-Point delay	10
Line broadcast delay	14
Column broadcast delay	14

2.2. Design and Implementation of swHPFM

2.2.1. Algorithm and task mapping scheme

Large fluid machinery has a physical structure that consists of multiplexers, multiblade rows, multiblade channels, and multi-regions [24]. Correspondingly, it is necessary to communicate the boundary data between blade rows when designing simulations. In addition, blade channel boundary and regional boundary communication are required, as illustrated in Figure 2. In order to illustrate the blade row and blade channel, the geometric model of the axial compressor is shown in Figure 3, in which red blades represent rotor blades, green blades are stator blades, and the polygon stands for the blade channel. The blade channel can be divided into blade regions. The full circle is the blade row.

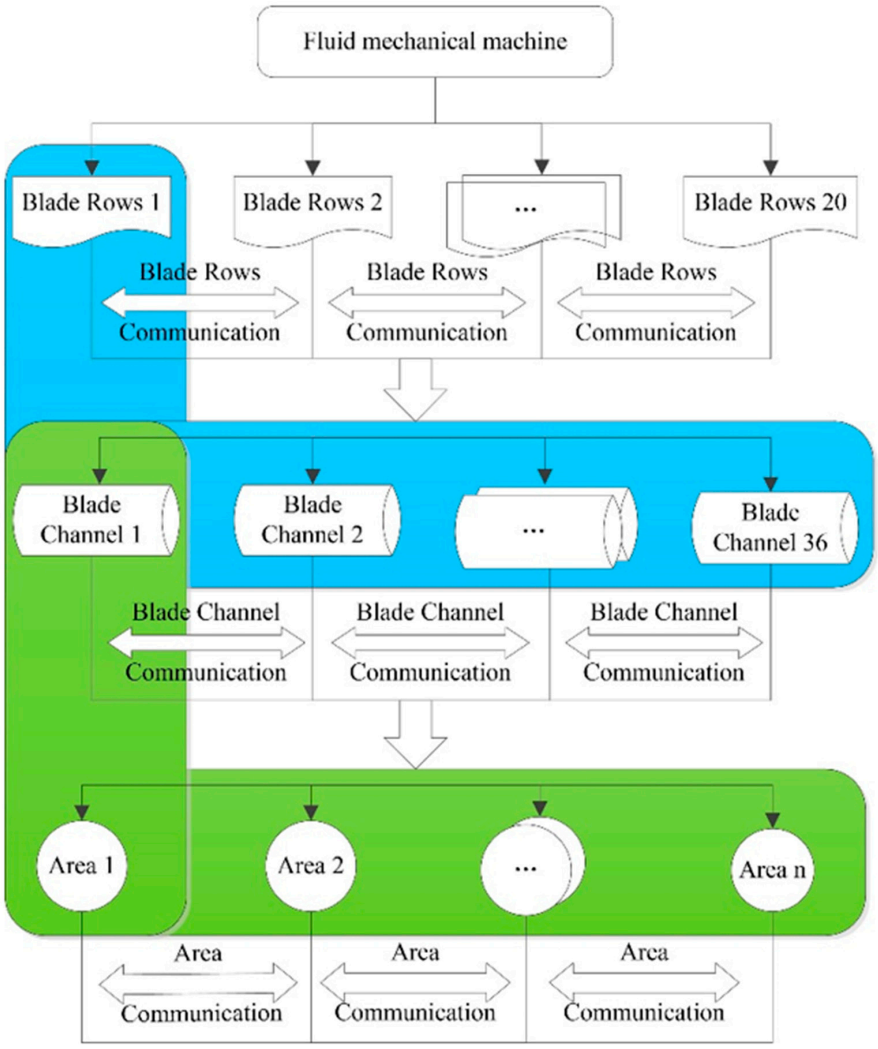


Figure 2. Parallel hierarchy of the axial compressor rotor. Large fluid machinery has a physical structure that consists of multiplexers, multiblade rows, multiblade channels, and multi-regions.

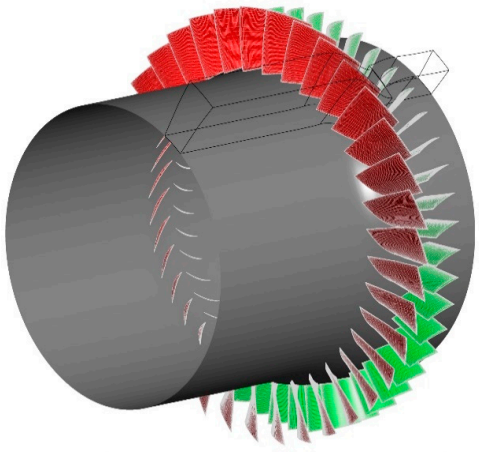


Figure 3. Geometric model of the axial compressor. The red blades represent rotor blades, green blades are stator blades, and the polygon stands for the blade channel. The blade channel can be divided into blade regions.

Similarly, the Sunway TaihuLight has a multilayer hardware architecture that consists of cabinets, supernodes, processors, and CGs, as shown as in Figure 4. From the bottom up, the main components are the CG of MPE+64CPEs, the SW26010 processor, which is composed of four CGs, and the super nodes, which are composed of 256 nodes. The four supernodes constitute the cabinet. The multilevel parallelism of the fluid mechanical model and the multilayer hardware architecture of the Sunway TaihuLight naturally correspond to each other, which provides favorable conditions for the task mapping of them.

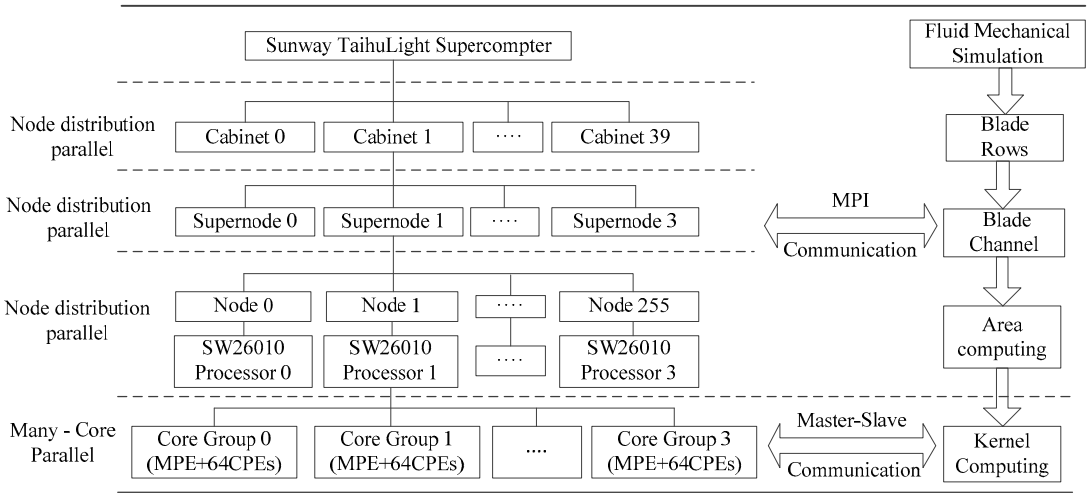


Figure 4. Mapping scheme of swHPFM for the physical model and the hardware architecture. The multilevel parallelism of the fluid mechanical model and the multilayer hardware architecture of the Sunway TaihuLight naturally correspond to each other.

swHPFM adopts two levels of parallelism: coarse-grained task parallelism and fine-grained data parallelism. Before submitting the simulation job to the Sunway TaihuLight supercomputer, the cost of MPI communication among nodes, $Cost_{communication} = Size_{communication} \times frequency$, was calculated. The MPI tasks are allocated under the principle of minimizing the communication cost. The mapping scheme of swHPFM for the simulation model and the hardware architecture is illustrated in Figure 4.

Thanks to its powerful logic control and interruption handling capabilities, the MPE on the node is responsible for handling complex scheduling tasks and decomposing the tasks of the blade rows into computing tasks of the kernel functions, shown as Equation (1), to complete the iterative calculation of the flow field. swHPFM distributes the kernel functions to the CPEs. Simultaneously, during the simulation, each CG returns data that are calculated by the CPE cluster in the main process to determine whether the calculation has converged and whether to exit the flow field simulation.

The algorithm of the axial compressor rotor designed in this paper is suitable for the finite volume method. The differential form of the fluid control equations is expressed in Equation (1)

$$\frac{\partial U}{\partial t} + \frac{\partial F_c}{\partial x} + \frac{\partial G_c}{\partial y} + \frac{\partial H_c}{\partial z} = I + \frac{\partial F_v}{\partial x} + \frac{\partial G_v}{\partial y} + \frac{\partial H_v}{\partial z} \quad (1)$$

where U stands for the conservation-type solving vector, F_c G_c, and H_c are the convection vectors along the coordinate directions x, y, and z, respectively, F_v, G_v, and H_v denote the viscous vectors along the three coordinate directions, and I represents the source term. In addition, each vector has six components, which represent the continuity equation, the three-directional momentum component

equation in Cartesian coordinates, the energy equation, and the S-A (Spalart-Allmaras) turbulence equation [25]. The expressions of the terms are shown in Equations (2) and (3).

$$U = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ \rho e \\ \rho \widetilde{v} \end{bmatrix}, F_c = \begin{bmatrix} \rho w_x \\ \rho w_x v_x + p \\ \rho w_x v_y \\ \rho w_x v_z \\ \rho w_x h \\ \rho \widetilde{v} w_x \end{bmatrix}, G_c = \begin{bmatrix} \rho w_y \\ \rho w_y v_x \\ \rho w_y v_y + p \\ \rho w_y v_z \\ \rho w_y h - \Omega p \\ \rho \widetilde{v} w_y \end{bmatrix}, H_c = \begin{bmatrix} \rho w_z \\ \rho w_z v_x \\ \rho w_z v_y \\ \rho w_z v_z + p \\ \rho w_z h - \Omega p \\ \rho \widetilde{v} w_z \end{bmatrix} \tag{2}$$

$$I = \begin{bmatrix} 0 \\ 0 \\ \rho \Omega v_z \\ -\rho \Omega v_y \\ 0 \\ I_{\widetilde{v}} \end{bmatrix}, F_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \beta_x \\ \alpha_x \end{bmatrix}, G_v = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ \beta_y \\ \alpha_y \end{bmatrix}, H_v = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ \beta_z \\ \alpha_z \end{bmatrix} \tag{3}$$

where ρ stands for the density, v_x, v_y, and v_z mean the absolute velocity component in three coordinate directions, w_x, w_y, and w_z represent the absolute velocity component in three coordinate directions, e is the ratio of the total energy, and $\widetilde{v}$ is the eddy current viscosity variable. The physical meanings and expressions of the other items can be seen in reference [26].

To increase the convergence speed, the solver uses the Runge-Kutta (R-K) explicit time method to simulate the flow field. The convection term and the diffusion term in the solver are discretized via the central differential scheme. In addition, to improve the accuracy and efficiency of the algorithm, redundant terms were constructed. The local time step method and implicit residual smoothing method are used to accelerate the convergence. Among them, since the second-order explicit central scheme of the non-convective diffusion equation is unstable, the R-K method constructs an explicit artificial viscous smoothing numerical oscillation. The implicit residual smoothing replaces the original residual value by weighting the residual at the nodes and the adjacent nodes, accelerating the smoothing of the residual and enlarging the stability range of the time step. In addition, to smooth the residual, while reducing the calculational burden, the residual smoothing is performed only at odd steps in the R-K time marching method [24].

In this paper, a multigrid method, namely, the full multigrid (FMG) method, was used to accelerate the algorithm, which is illustrated in Figure 5. First, the iteration was performed on the coarse grid (Level 3) until the error satisfied convergence criteria, and the iteration process on this layer was completed. Then, the numerical solutions and errors were interpolated to the medium grid (Level 2), where they were iterated via the V-cycle approach, until the convergence criterion was satisfied. Finally, the results were interpolated to the fine grid (Level 1) [27]. This process was iterated, until the convergence of the numerical solution on Level 1 was obtained.

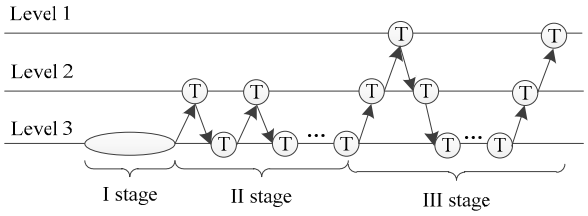


Figure 5. A full multigrid (FMG) cycle structure. Level 1 means the fine grid. Level 2 represents the medium grid. Level 3 stands for the coarse grid.

The algorithm flow chart in each node is illustrated in Figure 6. First, the flow field was initialized by reading the input grid file, which specifies the number of grids, the grid coordinates, the grid values,

and various physical parameters. Then, the core iteration process of the algorithm, namely, the R-K explicit time marching method, was executed, whose main objective was to solve the equation via iteration, until the results converged. At the same time, multiple nodes were executed in parallel, and the data between nodes communicated through MPI.

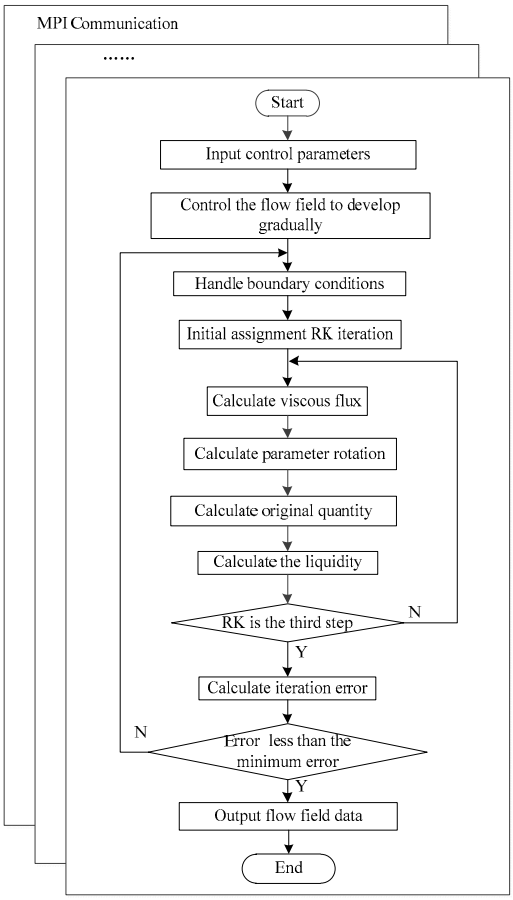


Figure 6. The algorithm flow chart. This algorithm simulates the rotor solver of the axial compressor. The algorithm flow in the rectangular box represents the tasks to be completed inside each node. Meanwhile, many different rectangular boxes exit, which indicates that multiple nodes are executed in parallel, and the data between nodes communicate through message passing interface (MPI).

Correspondingly, the swHPFM simulation computing on each CG of SW26010 processor is illustrated in Figure 7. As previously analyzed in this paper, the MPE is responsible for judging and controlling the whole simulation cycle, including calling the file reading module to read the grid data and communicating the boundary data by using the MPI. The CPE is responsible for data parallel computing. Therefore, kernel functions, such as the original quantity, residual, and source term, are distributed to exploit the computing performance.

For the structured grid fluid mechanical simulation algorithm, the main computing characteristic is regular memory access. In detail, the access mode is continuous access and regular stride access. However, in the innermost part of the array, although it is regular, the stride size is so large that the mode is close to discrete access, and the memory access cost is high [28].

To efficiently distribute data to the CPEs, the SPM access strategy, which is based on the computing characteristics, serpentine register communication, and SIMD was proposed to improve the computing performance.

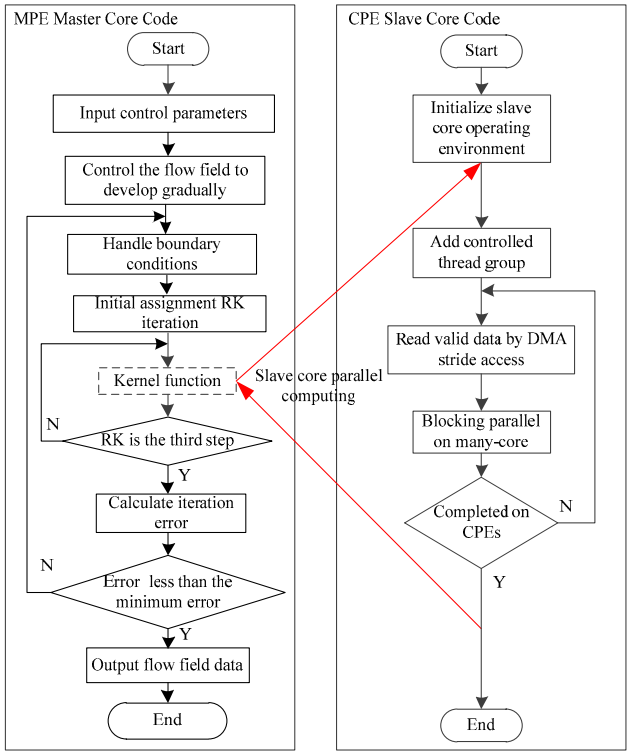


Figure 7. Parallel computing of master-slave cores in CGs. The MPE is responsible for judging and controlling the whole simulation cycle, including calling the file reading module to read the grid data and communicating the boundary data by the MPI. The CPE is responsible for data parallel computing.

2.2.2. SPM Access Strategy Based on DMA

In the structured grid axial compressor rotor algorithm, the array structure can be divided into two regions, namely, an outer layer and an inner layer, as illustrated in Figure 8. For many functions, such as the original quantity, residual, and source item, only the inner layer is calculated, without communicating the outer-layer data. Since the size of the SPM is only 64 KB, it is wasteful to transmit redundant data to the SPM. The proposed SPM access strategy (SAS) can effectively overcome this problem by only reading the valid data.

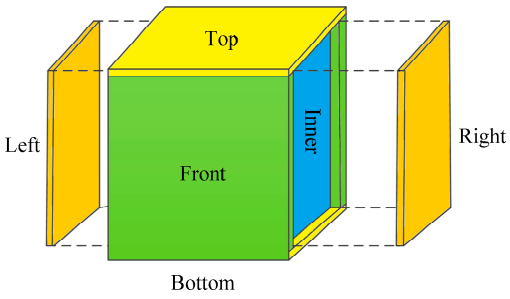


Figure 8. Array structure. In the algorithm, the array structure can be divided into two regions, namely, an outer layer and an inner layer.

When the SW26010 processor transfers data from its main memory to the its CPE' SPM of its CPE, it is limited in four aspects: space, transmission efficiency, mapping, and data dependence. The space limitation refers to the SPM space of the CPE being only 64 KB, and the user-available space in the LDM being limited to about 60 KB, namely, $LDM_{size} \leq 60$ KB. The transmission efficiency limitation refers to the peak performance of the DMA access being reached only if the main memory address of the data is 128-byte-aligned, and its size being a multiple of 128 bytes. To ensure transmission

efficiency, $Block_{size}\%128 = 0$ should be guaranteed on the premise of the alignment of the main memory addresses. The mapping and data dependence limitations means that the amount of transferred data must guarantee the integrity of a kernel function.

Blocking rules that are based on the space, transmission efficiency, mapping, and data dependence limitations are presented, as shown in Equation (4), in which $Total_{size}$ denotes the size of the data to be transferred, and *core_number* represents the total number of computing cores, whose value is 64, if all the slave cores are used. *Block* denotes the total number of blocks that are required for transmission.

$$Block = \left\lceil \frac{Total_{size}}{2^{10} \times LDM_{size} \times \text{core\_}number} \right\rceil \tag{4}$$

To effectively utilize the space of the SPM, only the valid data of the inner layer can be transmitted; hence, the stride access method was used.

First, when executing the data transmission, the stride on the main memory should be calculated, according to the outer and inner data sizes. For double-precision data, $Stride = Boundary_{size} \times 8$ bytes, wherein $Boundary_{size}$ denotes the number of boundary layers in the three-dimensional array of the fluid machinery algorithm. Then, $Carry_{size}$, which denotes the amount of data that are passed by the DMA, is calculated via Equation (5), wherein $Valid\_Data_{size}$ represents the size of the valid data of the inner part.

$$Carry_{size} = \frac{Valid\_Data_{size}}{Block} \tag{5}$$

Finally, the data on the CPE are tiled based on the mapping rules, as expressed in Equation (6). When CPE initiates a load/store data request, it is necessary to map each required datum to the main memory address. Our method enables continuous data blocks to be executed in the same time step. In Equation (6), Bias represents the offset of the main memory, $Block_{index}$ denotes the index of the current data block, and $Thread_{index}$ denotes the current number of threads of the CPE.

$$\text{Bias} = Block_{index} \times 64 + Thread_{index} \tag{6}$$

Meanwhile, to achieve an overlap between the computing and data transmission between the MPE and the CPE cluster, double buffering technology, which doubles the required space from the CPE's SPM, is used to load/store data. The SPM access strategy can fully utilize the SPM and improve the CPE's computing performance.

2.2.3. Serpentine Register Communication

When analyzing the structured grid rotor solver algorithm, in many kernel functions, such as the viscous flux, convective flux, and parameter rotation, stencil computing is used. There is a backward or forward dependence on the *x*-, *y*- and *z*-axes of the neighboring data. For instance, in the function viscous flux, the absolute velocity components, namely, v_x, v_y, and v_z, the original conserved quantity, and the derivatives of the flow parameters at the interface are needed in order to update the value of the function.

In this paper, a scenario in which both forward and backward data are needed was considered as an example, as illustrated in Figure 9. The green cube stands for the current grid that needs to be updated. The red cubes represent the adjacent grids, on which the green one depends. When updating the green grid data block, it is necessary to simultaneously read the forward data and the backward adjacent red blocks of other data. When executing the data transmission, the *x*-axis data are continuous; hence, the *x*-axis data must be completely transmitted. When the data on the discontinuous latitude in the *y*-axis direction are distributed to the CPE, only one datum value can be updated for every three data blocks, and the data utilization rate is so low that the parallel computing performance is severely degraded. In this case, serpentine register communication (SRC) is proposed to improve the data utilization and reduce the number of invalid communications.

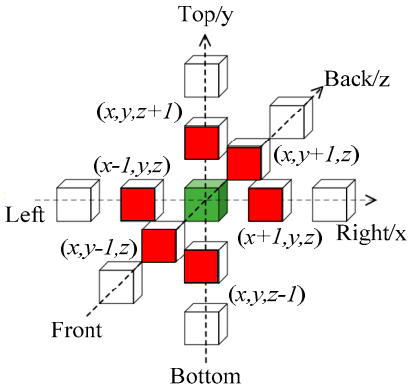
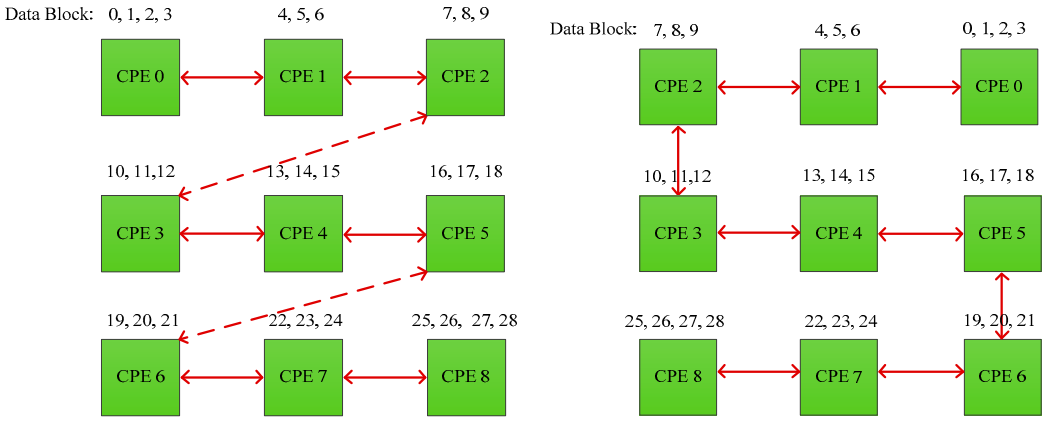


Figure 9. Data dependency in adjacent grids. There is a backward or forward dependence on the *x*-, *y*- and *z*-axes of the neighboring data.

In each SW26010 processor, register communication is designed to directly communicate only between the same row or the same column. To briefly explain the serpentine method used in this paper, it is supposed that there are 9 CPE resources, which form a 3 × 3 mesh. As in the actual scenario, the registers can only communicate within the same row or the same column. Data blocks 1–27 must be updated, and being limited by the SPM space, the SPM for each CPE can store up to 4 data blocks. Simultaneously, each calculation requires one forward data block and one backward data block. The data layout of the original register communication is illustrated in Figure 10a, in which the first CPE and the last CPE receive four data blocks: 1, 2, 3, and 4, as well as 25, 26, 27, and 28, respectively. The middle CPEs store three data blocks. Before communicating, the intermediate data are updated, such as data 5 in CPE1 and data 8 in CPE2. Then, forward communication is performed, and the data of the non-first column are sent forward in the same row. For example, data 4 are transmitted to the CPE0, and the location of data block 0 is covered to update data 3. However, the data in the first and last column, such as CPE3 and CPE2, must be diagonally communicated. Then, backward communication is executed, and the data of the non-last column are sent backward in the same row. For example, data 15 are transmitted to CPE5 to update data 16. The last-column data, such as CPE5 and CPE6, must be diagonally communicated.



(**a**) Original register communication. (**b**) Serpentine register communication.

Figure 10. Register communication method. (**a**) Original register communication in the Sunway TaihuLight supercomputer. (**b**) Serpentine register communication by changing the logical CPE thread number to a snake-like consecutive number.

However, diagonal communication, such as the communication of CPE2 and CPE3, is limited by its design. It must communicate in the same row, initially to transmit data from CPE2 to CPE0, and subsequently to communicate with CPE3 via column communication. This communication

method is complicated and time-consuming. For this scenario, serpentine register communication was proposed. By changing the logical CPE thread number to a snake-like consecutive number, as illustrated in Figure 10b, the number of communications was effectively reduced.

To complete the serpentine register communication, first, the thread numbers of the serpentine CPE are reordered by mapping the number of CPE threads to snake-type number to build the list of neighbors that must be communicated. The neighbor list builder pseudo-code is presented as Algorithm 1, in which *row_front* denotes the thread number of the previous CPE, *row_back* represents the thread number of the next CPE, *column_front* is the previous column of the CPE, and *column_back* denotes the thread number of the next column of the CPE. When the serpentine neighbor list is built, the data distribution will change accordingly, as illustrated in Figure 10.

Algorithm 1. Serpentine Neighbor List Builder.

```
Initialize the original thread ID of CPE
Get the thread number of the current CPE, thread_id ← athread_get_id(−1);
Obtain the row number of the CPE, row ← thread_id / 8;
Obtain the row number of the CPE, column ← thread_id % 8;
if row%2==0:
   serpentine_column ← 7-column;
else serpentine_column ← column;
end if
Update the thread ID of the serpentine CPEs, Serpentine_CPE_thread ← row*8+serpentine_ column;
row_front ← serpentine_column −1;
row_back ← serpentine_column +1;
if row_front==−1:
   column_front ← row+1;
end if
if row_back==8:
   column_back ← row−1;
end if
```

Finally, register communication was executed according to the neighbor list. Since the calculation relies on the forward and backward data, it is necessary to communicate forward and backward. The pseudo-code of the serpentine register communication is presented, as shown in Algorithm 2. In this way, diagonal communication can be executed directly, without intermediate routing registers, and the register communications can be effectively reduced.

2.2.4. Vectorization

In the SW26010 processor, each CPE has a 256-bit-wide SIMD floating-point unit, whose instruction pipeline is fixed to up to 8 stages. To analyze its main cost, an aligned vectorized and nonvectorized test set was designed, in which 512-cycle addition, subtraction, multiplication, and division operations were compared. The results demonstrate that, for the aligned SIMD vectorized program, the computing performance can be accelerated by as much as 3.956x, compared to the scalar operation, as listed in Table 2.

Table 2. Aligned vectorization performance.

Instruction	Non-Vectorized (Cycles)	Vectorized (Cycles)	Speedup
vaddd	3342	846	3.950
vsubd	3417	917	3.726
vmuld	3417	846	4.039
vdivd	18169	4436	4.096
vmad	6159	1551	3.971

Algorithm 2. Serpentine Register Communication.

```
Initialize serpentine neighbor list
while completing false target data communication:
  if serpentine_column != 0:
    send target data to the row_front via row communication;
  else if Serpentine_CPE_thread != 0:
    send target data to column_front via column communication;
  end if
  waiting for the transmission to complete and receiving the data;
  if serpentine_column != 7:
    send target data to the row_back via row communication;
  else if Serpentine_CPE_thread != 63
    send target data to column_back via column communication;
  end if
  waiting for the transmission to complete and receiving the data;
end while
```

However, to use simd_load/simd_store for vectorization, it is necessary to ensure that the standard-type variable is 32-byte-aligned (for floatv4, only 16-byte-aligned). The compiler can only guarantee that the head address of the array is aligned. The remainder of the data address is required for a programming guarantee. If unaligned memory access is executed, the MPE automatically degenerates into unaligned access, executed by simd_loadu/simd_storeu extension functions. However, a higher price will be paid. However, the CPEs will directly output the "Unaligned Exception" error.

In the CPEs, it is necessary to explicitly use the simd_loadu/ simd_storeu extension functions or the vldd_ul/vldd_uh/ vstd_ul/vstd_uh instructions to execute the unaligned operation. Comparing the unaligned vectorized and nonvectorized 512-cycle addition, subtraction, multiplication and division operations, the results demonstrate that the unaligned SIMD vectorized program only realizes an average speedup of 1.822×, compared to the scalar operation, as listed in Table 3.

Table 3. Unaligned vectorization performance.

Instruction	Non-Vectorized (Cycles)	Vectorized (Cycles)	Speedup
vaddd	3405	2315	1.471
vsubd	3407	2316	1.471
vmuld	3408	2317	1.471
vdivd	17895	5819	3.075
vmad	6154	3794	1.622

Comparing Tables 2 and 3, the unaligned SPM access severely degraded the vectorization performance. The absolute latency of the unaligned access was even higher. In addition, the latency of the vectorization operation, which includes the arithmetic and permutation operations [22], is listed in Table 4.

Table 4. Absolute and pipeline latencies of the SIMD instructions.

Category	Instruction	Absolute Latency (Cycles)	Pipeline Latency (Cycles)
Arithmetic	vaddd, vadds, vsubd, vsubs, vmad, vmas	7	1
	vsqrtd/vsqrts	32/18	28/14
	vdivd/vdivs	34/19	30/15
Permutation	vinsw, vinsf, vextw, vextf, vshff, vshfw	1	1

The instruction prefix 'v' stands for the vector operations. The instruction suffixes 'd'/'s' and 'w'/'f' represent double/single-precision and word/ floating-point vectors, respectively. The permutation instructions 'ins'/'ext'/'shf' correspond to insertion/ extraction/shuffle operations. Therefore, to fully utilize the SW26010 vectorization, the kernel array was converted from an array of structures (AOS) into a structure of arrays (SOA) to facilitate the alignment operation in the CPE. In addition, since the shuffling operation has a lower latency than unaligned access, a fine-grained shuffling operation is used to generate data to reduce the number of unaligned loadu/stroeu operations.

3. Results and Analysis

Based on the swHPFM framework, the axial compressor rotor algorithm was implemented to evaluate the performance of the framework. First, the 2340-core-scale program, which simulates one blade channel per process and communicates between each process via MPI, was evaluated. Then, each blade channel was divided into 12 subdomains, with each process simulating a subdomain, and the 28080-core-scale program was implemented. In addition, the accuracy of the numerical simulation in this experiment was determined by using the numerical algorithm. Therefore, the residuals were platform-independent. To ensure the accuracy of the experimental results, each job was submitted 10 times, and the average of these tests was taken as the final result.

3.1. swHPFM Validation and Algorithm Results

Both the 2340-core-scale and the 28080-core-scale programs of the swHPFM converged at −10, after about 9429 iteration steps, as shown in Figure 11. While the residual convergence curve fluctuated slightly, the residual declined. What is more, these trends are the same as those of the Tianhe-2A supercomputer, whose processors are Intel Xeon E5-2692v2. The residual convergence of hydromechanics corresponds to the accuracy of the swHPFM parallel framework.

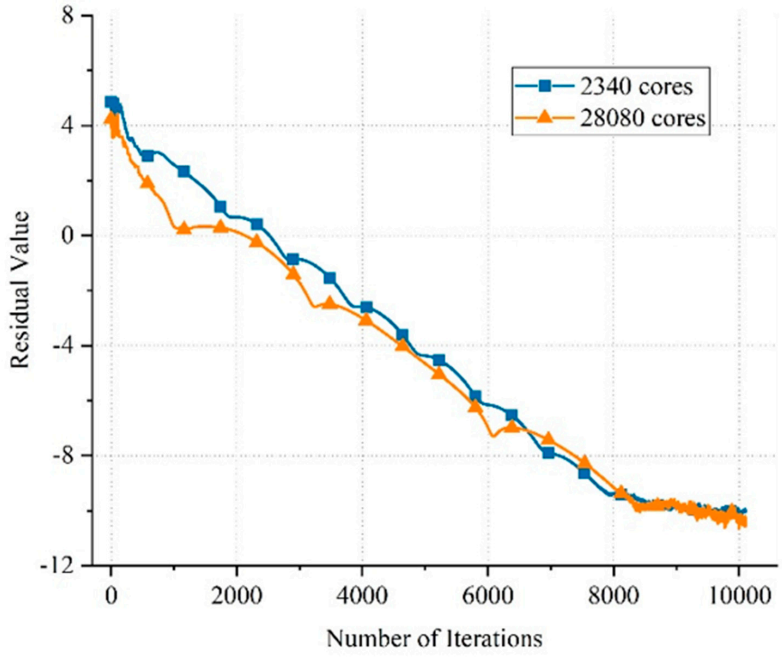
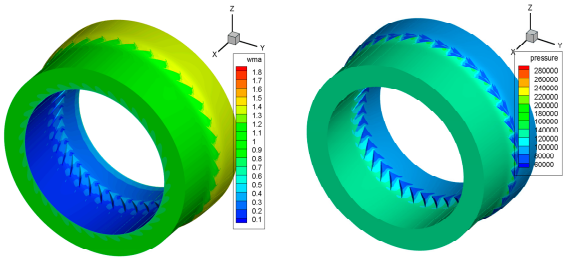


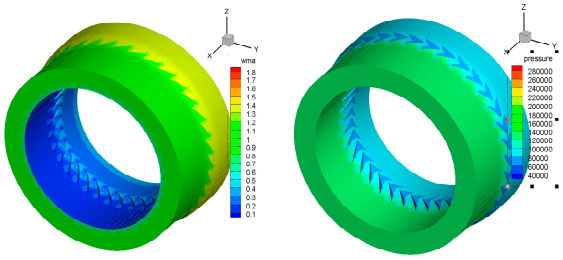
Figure 11. Residuals of the algorithm of a whole blade row of the axial compressor. The trends are the same as those of the Tianhe-2A supercomputer. The residual convergence of hydromechanics corresponds to the accuracy of the swHPFM parallel framework.

In addition, the simulated generated mash and press data were processed by postprocessing the software, Tecplot, to obtain the results of the flow field simulation, which are presented in Figures 12 and 13. All of these results prove that the swHPFM framework is accurate.



(**a**) Relative Mach number contour （**b**） Static pressure contour

Figure 12. Simulation results with 2340 cores. (**a**) Relative mach number contour; (**b**) static pressure contour.



(**a**) Relative Mach number contour (**b**) Static pressure contour

Figure 13. Simulation results with 28080 cores. (**a**) Relative mach number contour; (**b**) static pressure contour.

3.2. swHPFM Performance Results and Analysis

In addition, to evaluate the performance and scalability of the fluid mechanical algorithm and the swHPFM parallel framework, the parallel computing scale is expanded by 12 times, which means that the total number of computing cores is expanded from 2340 to 28,080. After fully utilizing the CPE computing resources, our methods achieved an average speedup of 11.231× for the kernel functions and achieved a parallel efficiency of 93.3%; hence, the algorithm and swHPFM framework has a satisfactory scalability, as shown as Figure 14.

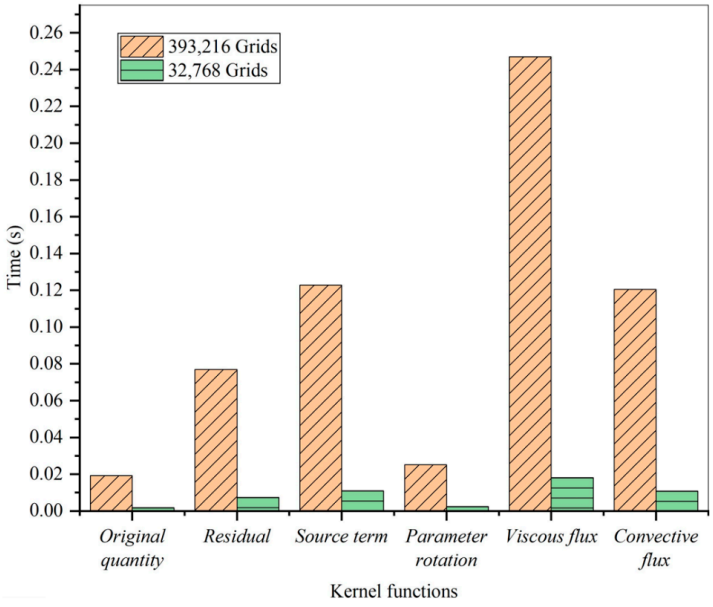


Figure 14. Comparison of swHPFM under two grid workloads: the parallel computing scale is expanded by 12 times, which means that the total number of computing cores is expanded from 2340 to 28,080.

4. Discussion

In this section, to discuss the optimization effect in more detail, comprehensive experiments are conducted to evaluate the performance of the swHPFM parallel framework, and the kernel functions, namely, the Original quantity, Residual, Source term, Parameter rotation, Viscous flux, and Convective flux functions, were optimized and executed on the Tianhe-2A supercomputer, which uses Intel Xeon E5-2692v2 12-core 2.2-GHz processors, and on the Sunway TaihuLight under two different workloads.

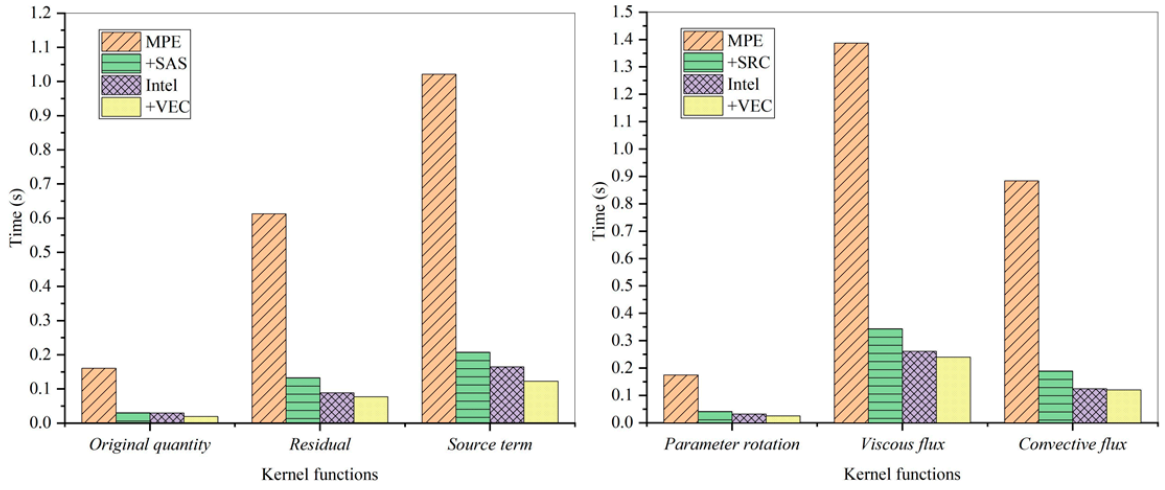
First, without any optimization, the algorithm program was run on the Tianhe-2A supercomputer and on the Sunway TaihuLight supercomputer. If the codes run directly on the SW26010 processor, then only the MPE can be used. As shown in Table 5, under the workload of 393,216 grids of each node, compared with the Intel Xeon E5-2692v2 processor, the kernel functions have an average performance reduction of 6.058×, in which the convective flux decreases to 7.119×. These data demonstrate the urgency of optimizing the code, according to the features of the SW26010 processors.

Table 5. Kernel function runtimes on 393,216 grids.

Kernel Function	SW26010 (s)	Intel Xeon E5 (s)	Performance Degradation
Original quantity	0.160477	0.029675	5.407818
Residual	0.612526	0.088771	6.900069
Source term	1.021608	0.164854	6.197023
Parameter rotation	0.174760	0.032232	5.421941
Viscous flux	1.386930	0.261435	5.305065
Convective flux	0.884039	0.124187	7.118611

Figure 15a plots the runtimes of the kernel functions, Original quantity, Residual, and Source term, on the Intel and SW26010 processors under the 393,216-grid workload. A common feature of these functions is that they only update inner data and have no data dependence in their instructions. For the Original quantity function, the SAS optimization achieves 5.343× speedup, compared to the unoptimized codes, which can only run on the MPE of the SW26010 processors. On this basis, vectorization optimization obtains 8.362× speedup, compared to the unoptimized code. Finally, the swHPFM is 1.546 times faster than the Intel Xeon E5 processor. For the Residual function, SAS achieves 4.615× speedup, compared to the unoptimized codes. On this basis, after vectorization, a speedup of 7.966× is developed, compared to the unoptimized code. Overall, the swHPFM is 1.154 times faster than the Intel Xeon E5 processor. For the Source term function, SAS obtains a 4.923× speedup. On this basis, vectorization obtains a 8.315× speedup. The swHPFM is 1.342 times faster than the Intel Xeon E5 processor. Since the Original quantity function requires fewer parameters, it achieves a higher speedup.

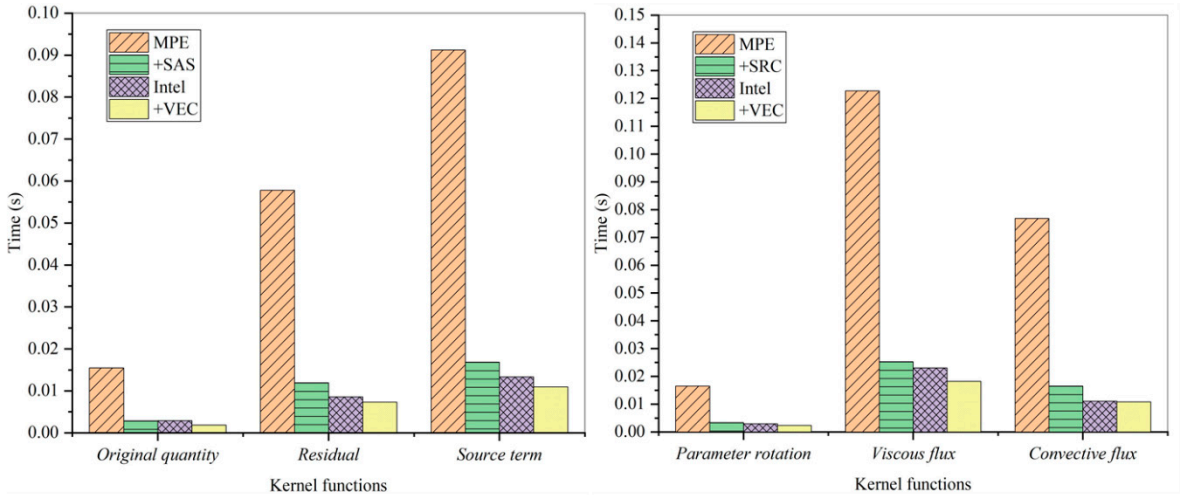
Figure 15b plots the runtimes of the kernel functions Parameter rotation, Viscous flux, and Convective flux, on the Intel and SW26010 processors under a 393,216-grid workload. The common feature of these functions is that they are used for stencil computing. For the Parameter rotation function, the SRC achieves 4.262× speedup, compared to the original codes. On this basis, vectorization yields a 6.955× speedup compared to the unoptimized code. Overall, the swHPFM is 1.283 times faster than the Intel Xeon E5 processor. For the Viscous flux function, SRC obtains a 4.043× speedup compared to the unoptimized codes. On this basis, vectorization obtains a 5.616× speedup, compared to the unoptimized code. Finally, the swHPFM is 1.059 times faster than the Intel Xeon E5 processor. For the Convective flux function, SRC yields a 4.676× speedup. On this basis, vectorization obtains a 7.336× speedup. The swHPFM is 1.031 times faster than the Intel Xeon E5 processor. Since the Original quantity function requires fewer parameters, it realizes a higher speedup.



(**a**) Optimization results of the first type kernel functions (**b**) Optimization results of the second type kernel functions

Figure 15. Optimization results of the kernel functions under a 393,216-grid workload: (**a**) the first type of kernel functions are optimized by SAS+VEC, and (**b**) the second type of kernel functions are optimized by SRC+VEC.

To evaluate the performance and scalability of the algorithm and swHPFM framework in this paper, we extended the parallel scale by 12 times to reduce the computing load on each node. Figure 16 plots the runtimes of the kernel functions under a 49,152-grid workload. According to Figure 16a, in the case of a 12× reduction in the workload, for the Original quantity, Residual, and Source term functions, the swHPFM still realizes an average speedup of 8.212×, compared to the unoptimized codes, and is 1.326 times faster than the Intel Xeon E5 processor.



(**a**) Optimization results of the first type kernel functions (**b**) Optimization results of the second type kernel functions

Figure 16. Optimization results of the kernel functions under a 49,152-grid workload: (**a**) the first type of kernel functions are optimized by SAS+VEC, and (**b**) the second type of kernel functions are optimized by SRC+VEC.

According to Figure 16b, in the case of a 12× reduction in the load, for the Parameter rotation, Viscous flux, and Convective flux functions, the swHPFM realizes an average speedup of 6.921×, compared to the unoptimized codes, which can only run on the MPE of the SW26010 processors, and is 1.170 times faster than the Intel Xeon E5 processor.

The results demonstrate that the algorithm and swHPFM framework proposed in this paper are far beyond the previous state of the art.

In our future work, we will continue modifying our algorithm and framework to develop a more general framework for optimizing algorithms on the Sunway many-core processor. The main objective is to refine the algorithm and the optimization strategies to obtain more suitable mappings of the algorithms and the architecture and increase the fluid mechanical algorithm speed on the Sunway system.

5. Conclusions

In this paper, an algorithm and swHPFM framework are proposed and used to optimize the structured grid fluid mechanical algorithm on the Sunway TaihuLight supercomputer. In this framework, a suitable mapping of the application and the underlying system architecture is developed. In addition, to effectively utilize its 64K SPM, which must be explicitly controlled by the user, an SPM access strategy based on DMA, is proposed in order to optimize the data transmission between the main memory and SPM. In addition, serpentine register communication is designed by changing the logical CPE thread number into a snake-like consecutive number to reduce invalid communications. Moreover, by experimenting with the alignment and unalignment vectorization performances, the vector cost is analyzed, and the kernel array is converted from AOS into SOA to facilitate the alignment operation in the CPEs. Fine-grained shuffling operations are used to generate data in order to reduce the loadu/stroeu operations. Finally, the axial compressor rotor simulation program is implemented to evaluate the framework. The proposed optimization methods and the algorithm can be used directly in the high-precision and large-scale fluid mechanical algorithm on the Sunway TaihuLight supercomputer.

The main contributions of this paper are as follows:

(1) A layered swHPFM framework is proposed. In the framework, task mapping from the algorithm model to the computing node is solved by combining the model characteristics of large fluid machinery with the architecture of the Sunway TaihuLight supercomputer.
(2) For the SW26010 many-core processor, according to the characteristics of the application and based on the Athread library, the SPM access strategy and Serpentine register communication are proposed to fully utilize the slave computing resources.
(3) By building a test set of alignment and misalignment load and store operations, the vectoring cost is analyzed, and the fine-grained method is used to accelerate the vectorization of the kernel functions.
(4) Based on the algorithm and swHPFM parallel framework, the axial compressor rotor algorithm program for the Sunway TaihuLight is optimized. In addition, the performance of the swHPFM is evaluated.

The proposed algorithm and framework have major significance for the efficient optimization of the fluid mechanical algorithm on computing platforms with the Sunway TaihuLight many-core architecture.

Author Contributions: The authors contributions are as follows: Conceptualization, J.L.; methodology, J.L.; software, C.Z.; validation, C.Z.; formal analysis, J.L.; investigation, J.L.; resources, Z.J.; data curation, J.L.; writing—original draft preparation, J.L.; writing—review and editing, J.L.; visualization, J.Z.; supervision, X.Z.; project administration, X.D.; funding acquisition, X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China, grant number: 2016YFB0200902.

Acknowledgments: We greatly appreciate the careful reviews and thoughtful suggestions, provided by the reviewers.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kundu, P.K.; Cohen, I.M.; Dowling, D.R. *Fluid Mechanics*, 6th ed.; Academic Press: Salt Lake City, UT, USA, 2015; pp. 16–25.
2. TOP500 Supercomputer List. Available online: https://www.top500.org/lists/2019/06/ (accessed on 9 November 2019).
3. Afzal, A.; Ansari, Z.; Faizabadi, A.R. Parallelization Strategies for Computational Fluid Dynamics Software: State of the Art Review. *Arch. Comput. Methods Eng.* **2016**, *24*, 337–363. [CrossRef]
4. Peterson, B.; Humphrey, A.; Holmen, J. Demonstrating GPU Code Portability and Scalability for Radiative Heat Transfer Computations. *J. Comput. Sci.* **2018**, *27*, 303–319. [CrossRef]
5. Mininni, P.D.; Rosenberg, D.; Reddy, R. A hybrid MPI–OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence. *Parallel Comput.* **2011**, *37*, 316–326. [CrossRef]
6. Sivanandan, V.; Kumar, V.; Meher, S. Designing a parallel algorithm for Heat conduction using MPI, OpenMP and CUDA. In Proceedings of the Parallel Computing Technologies, Bangalore, India, 19–20 February 2015; IEEE Press: Piscataway, NJ, USA, 2015; pp. 1–7.
7. Utrera, G.; Gil, M.; Martorell, X. In search of the best MPI-OpenMP distribution for optimum Intel-MIC cluster performance. In Proceedings of the International Conference on High Performance Computing & Simulation, Amsterdam, The Netherlands, 20–24 July 2015; IEEE Press: Piscataway, NJ, USA, 2015; pp. 429–435.
8. Muhammed, T.; Mehmood, R.; Albeshri, A.; Katib, I. SURAA: A Novel Method and Tool for Loadbalanced and Coalesced SpMV Computations on GPUs. *Appl. Sci.* **2019**, *9*, 947. [CrossRef]
9. Marszałek, Z.; Woźniak, M.; Połap, D. Fully Flexible Parallel Merge Sort for Multicore Architectures. *Complexity* **2018**. [CrossRef]
10. Blug, A.; Regina, D.J.; Eckmann, S.; Senn, M.; Bertz, A.; Carl, D.; Eberl, C. Real-Time GPU-Based Digital Image Correlation Sensor for Marker-Free Strain-Controlled Fatigue Testing. *Appl. Sci.* **2019**, *9*, 2025. [CrossRef]
11. Peng, B.; Luo, S.; Xu, Z.; Jiang, J. Accelerating 3-D GPU-based Motion Tracking for Ultrasound Strain Elastography Using Sum-Tables: Analysis and Initial Results. *Appl. Sci.* **2019**, *9*, 1991. [CrossRef] [PubMed]
12. Liu, P.; Ye, S.; Wang, C.; Zhu, Z. Spark-Based Parallel Genetic Algorithm for Simulating a Solution of Optimal Deployment of an Underwater Sensor Network. *Sensors* **2019**, *19*, 2717. [CrossRef] [PubMed]
13. Wu, X.; Lu, Y.; Duan, X.; Zhang, D.; Deng, W. Design and DOF Analysis of a Novel Compliant Parallel Mechanism for Large Load. *Sensors* **2019**, *19*, 828. [CrossRef] [PubMed]
14. Fu, H.H.; He, C.H.; Chen, B.W.; Yin, Z.K.; Zhang, Z.G.; Zhang, W.Q.; Zhang, T.J.; Xue, W.; Liu, W.G.; Yin, W.W.; et al. 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: Enabling depiction of 18-Hz and 8-meter scenarios. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, Denver, CO, USA, 12–17 November 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 1–12.
15. Yang, C.; Xue, W.; You, H. 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, Salt Lake City, UT, USA, 13–18 November 2016; IEEE Press: Piscataway, NJ, USA, 2016; pp. 57–68.
16. Chen, Y.; Li, K.; Yang, W.; Xie, X.; Xiao, G.; Li, T. Performance-Aware Model for Sparse Matrix-Matrix Multiplication on the Sunway TaihuLight Supercomputer. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 923–938. [CrossRef]
17. He, L.X.; An, H.; Yang, C.; Wang, F.; Chen, J.S. PEPS++: Towards Extreme-Scale Simulations of Strongly Correlated Quantum Many-Particle Models on Sunway TaihuLight. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 2838–2848. [CrossRef]
18. Chen, B.W.; Fu, H.H.; Wei, Y.W.; He, C.H.; Zhang, W.Q.; Li, Y.X.; Wan, W.B.; Zhang, W.; Gan, L.; Zhang, W.; et al. Simulating the Wenchuan earthquake with accurate surface topography on Sunway TaihuLight. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, Dallas, TX, USA, 11–16 November 2018; IEEE Press: Piscataway, NJ, USA, 2018.
19. Wang, X.L.; Liu, W.F.; Xue, W.; Wu, L. swSpTRSV: A Fast Sparse Triangular Solve with Sparse Leve l Tile Layout on Sunway Architectures. In Proceedings of the Principles and Practice of Parallel Programming, Vienna, Austria, 24–28 February 2018; ACM: New York, NY, USA, 2018.
20. Zhao, W.L.; Fu, H.H.; Fang, J.R.; Zheng, W.J.; Gan, L.; Yang, G.W. Optimizing Convolutional Neural Networks on the Sunway TaihuLight Supercomputer. *ACM Trans. Archit. Code Optim.* **2018**, *15*, 13. [CrossRef]

21. Liu, Z.; Chu, X.S.; Lv, X.J.; Meng, H.S.; Shi, S.P.; Han, W.J.; Xu, J.H.; Fu, H.H.; Yang, G.W. SunwayLB: Enabling Extreme-Scale Lattice Boltzmann Method Based Computing Fluid Dynamics Simulations on Sunway TaihuLight. In Proceedings of the International Parallel and Distributed Processing Symposium, Rio de Janeiro, Brazil, 20–24 May 2019; IEEE Press: Piscataway, NJ, USA; pp. 557–566.
22. Lin, J.; Xu, Z.; Cai, L.; Nukada, A.; Matsuoka, S. Evaluating the SW26010 Many-core Processor with a Micro-benchmark Suite for Performance Optimizations. *Parallel Comput.* **2018**, *77*, 128–143. [CrossRef]
23. Parallel Programming and Optimization of Shenwei TaihuLight. Available online: http://demo.wxmax.cn/wxc/process.php?word=process&i=54 (accessed on 9 November 2019).
24. Zhang, C.H.; Ju, Y.P. *Theory and Calculation of Fluid Mechanical Internal Flow*; Mechanical Industry Publishing: Beijing, China, 2016; pp. 129–145.
25. Deck, D.; Duveau, P.; D'Espiney, P.; Guillen, P. Development and application of Spalart–Allmaras one equation turbulence model to three-dimensional supersonic complex configurations. *Aerosp. Sci. Technol.* **2002**, *6*, 171–183. [CrossRef]
26. Liu, A.; Yu, Y.P.; Zhang, C.Z. Multi-block multi-level grid method and parallel simulation of internal flows of transonic rotor. *J. Aerosp. Power* **2018**, *33*, 1705–1712.
27. Ta'Asan, S. Multigrid method for a vortex breakdown simulation. *Appl. Numer. Math.* **1986**, *2*, 303–311. [CrossRef]
28. Xin, L.; Heng, G.; Sun, R.J.; Chen, Z.N. The Characteristic Analysis and Exascale Scalability Research of Large Scale Parallel Applications on Sunway TaihuLight Supercomputer. *Chin. J. Comput.* **2018**, *41*, 2209–2220.