TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/14 pp56–67
DOI: 10.26599/TST.2018.9010112
Volume 25, Number 1, February 2020

Heterogeneous Parallel Algorithm Design and Performance Optimization for WENO on the Sunway TaihuLight Supercomputer

Jianqiang Huang, Wentao Han, Xiaoying Wang, and Wenguang Chen*

Abstract: A Weighted Essentially Non-Oscillatory scheme (WENO) is a solution to hyperbolic conservation laws, suitable for solving high-density fluid interface instability with strong intermittency. These problems have a large and complex flow structure. To fully utilize the computing power of High Performance Computing (HPC) systems, it is necessary to develop specific methodologies to optimize the performance of applications based on the particular system's architecture. The Sunway TaihuLight supercomputer is currently ranked as the fastest supercomputer in the world. This article presents a heterogeneous parallel algorithm design and performance optimization of a high-order WENO on Sunway TaihuLight. We analyzed characteristics of kernel functions, and proposed an appropriate heterogeneous parallel model. We also figured out the best division strategy for computing tasks, and implemented the parallel algorithm on Sunway TaihuLight. By using access optimization, data dependency elimination, and vectorization optimization, our parallel algorithm can achieve up to 172× speedup on one single node, and additional 58× speedup on 64 nodes, with nearly linear scalability.

Key words: parallel algorithms; Weighted Essentially Non-Oscillatory scheme (WENO); optimization; many-core; Sunway TaihuLight

1 Introduction

In large-scale, high-performance scientific and engineering computing applications, such as Computational Fluid Dynamics (CFD), numerical weather prediction, seismic data processing, genetic engineering, and phase-field simulation, numerical solutions to high dimensional differential equations are major challenges.

Weighted Essentially Non-Oscillatory scheme (WENO) is a solution to hyperbolic conservation laws based on essentially non-oscillatory schemes and suitable for solving high-density fluid interface instability with strong intermittency. These problems have a large complex flow structure. Solving nonlinear hyperbolic conservation laws is one of the most important research topics in computational fluid dynamics. The three conservation laws can be introduced to control the motion of fluid equations, which are mostly non-linear double. The laws of constant law, such as the Euler equation and the magneto-hydrodynamic equation in plasma science, often need to solve the physical models involved in such equations in the fields of aerospace, shipbuilding, and electromagnetic and chemical engineering. Therefore, it is important to study the most accurate and efficient methods for solving the nonlinear hyperbolic

• Jianqiang Huang and Xiaoying Wang are with the State Key Laboratory of Plateau Ecology and Agriculture, Department of Computer Technology and Applications, Qinghai University, Xining 810016, China. E-mail: hjqxaly@163.com; xy.wang@foxmai.com.
• Jianqiang Huang is also with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.
• Wentao Han and Wenguang Chen are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: hanwentao@tsinghua.edu.cn; cwg@tsinghua.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2018-04-17; revised: 2018-07-07; accepted: 2018-07-09

conservation laws. Harten et al.[1] presented a basic no-shock format Essentially Non-Oscillatory (ENO), Liu et al.[2] presented the WENO schemes, while Jiang and Shu[3] proposed a new smooth measure. One of WENO's advantages is that it can achieve high precision on coarse-grained mesh. However, if we want to simulate problems from cosmology at an acceptable resolution, the mesh granularity should be as small as possible, which greatly increases the amount of computation.

Large-scale fluid computing based on homogeneous CPU clusters was a great success in the era of Terascale[4,5]. In the era of Petascale, computing accelerators such as GPUs are employed to provide more computing capability. The usage of GPUs in computing has led to achievements requiring the careful design of mixed code for applications[6,7].

The workload of WENO requires very heavy computation in linear algebra. In this article, we design and implement WENO on the Sunway TaihuLight system, which is currently the top system in the TOP500 list, and has its own special architecture. Compared with GPUs, Sunway is different due to its fast on-chip buffer and limited memory bandwidth, which are the main challenges for us in this work. Our major contributions include:

- We profiled the workload and found the hotspot kernel functions. For each kernel, we analyzed its characteristics in depth, and optimized it with a Sunway-specific mechanism.
- We designed and implemented a multi-level parallel scheme of Message Passing Interface (MPI) + many-core techniques, and figured out the best division strategy for tasks.
- The program was further optimized with an asynchronous Direct Memory Access (DMA) transmission scheme, Single Instruction Multiple Data (SIMD) interface, and division equivalent substitution.

The evaluation shows that the kernel functions can achieve up to 172× speedup on a single node on Sunway TaihuLight. They also achieve 58× speedup on 64 nodes, and show good scalability on Sunway TaihuLight.

2 Background

Many-core accelerators have been used in recent years to solve computationally intensive tasks, as Fig. 1 reveals for a CPU and GPU heterogeneous system[8–10]. The transformation of this architecture brings a huge

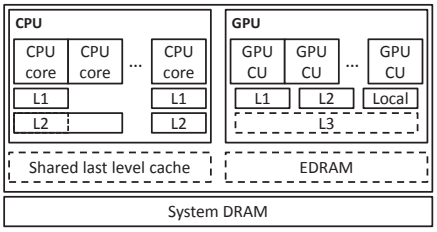

Fig. 1 Integrated CPU/GPU heterogeneous architecture, which integrates CPUs and GPUs on the same chip, sharing the physical memory.

transformation to existing high-performance computing software in many applications, such as earth simulation, climate simulation, material simulation, and phase field simulation.

In general, with climate and weather models requiring more computing resources and more complicated physics[11], there is still a gap between the growing demand and increasing supply of multicore acceleration. In order to fill that gap between the need for, and supply of, computing resources for certain applications[12], we have extensively reconstructed and optimized the WENO model in our work. Sunway is a supercomputer system consisting of Management Processing Elements (MPEs) and clusters of Computing Processing Elements (CPEs)[13]. The WENO application requires more computing resources; traditional hardware, such as a GPU, can be used to speed up numerical calculations, while the Sunway machine has strong computing power and can provide additional computing resources for WENO.

In some cases, the high-order solution is very complex, and the evolution time of these structures is so long that it is impractical to use a low-order method to obtain an acceptable resolution. Such problems often involve special non-oscillating high-end schemes. A very simple example, first used in Ref. [14], shows the evolution of the two-dimensional periodic vorticity of the compressible Euler equation.

The WENO method is quite general for solving high order numerical methods of convection, especially the hyperbolic conservation law. The WENO method is to obtain a higher order approximation by using the final approximation as a low order approximation candidate template for a nonlinear convex combination. The stability and non-oscillation of the WENO process are mainly dependent on the linearity of the linear weight.

The main advantage of this approach is that it can achieve any high-order formal accuracy in smooth areas, non-oscillation, and abrupt intermittent

transition. In several typical linear weights, the WENO program that this approach did numerical simulations, including WENO interpolation, WENO reconstruction, WENO approximation of the first and second derivative, and WENO integration.

The WENO method is widely used in various fields. Some examples include dynamical response of a stellar atmosphere to pressure perturbations[14], shock vortex interactions and other gas dynamics problems[15, 16], incompressible own problems[17], Hamilton-Jacobi equations[18], magneto-hydrodynamics[19], underwater blast-wave focusing[20], composite schemes and shallow water equations[21], real gas computations[22], wave propagation using Fey's method of transport[23], etc.

The mathematical model of the hyperbolic conservation law equation is as follows:

- The linear convection equation is

$$u_t + au_x = 0 \tag{1}$$

This model is the simplest partial differential equation, which is used to test the numerical methods.

- The inviscid Burgers equation is

$$u_t + \left(\frac{u^2}{2}\right)_x = 0 \tag{2}$$

For a scalar equation of one dimension, the model is a simple equation of hyperbolic conservation laws, because the model solution of the Cauchy problem can continuously produce shock; for this reason it is often used for discontinuous solution design, analysis, and simulation of capture method.

- One dimensional Euler system is

$$u_t + f(u)_x = 0 \tag{3}$$

where u and $f(u)$ are the conserved variable and flux.

$$u = \begin{bmatrix} \rho \\ \rho v \\ E \end{bmatrix}, \quad f(u) = \begin{bmatrix} \rho v \\ \rho v^2 + p \\ v(E+p) \end{bmatrix},$$

where v, p, ρ, and E are velocity, pressure, density, and total energy, and four satisfy the equation of state.

$$p = (\gamma - 1)\left(E - \frac{\rho v^2}{2}\right) \tag{4}$$

In Computational Fluid Dynamics, the study of the numerical method of Euler equations has important theoretical significance and application value.

This paper develops a rapid and accurate simulation of hypersonic turbulence with a high order compact format parallel algorithm. Good parallel efficiency can be achieved using a high-order compact format and partitioned parallel computing strategies, and using the *C* and *Fortran* programming languages on the Sunway TaihuLight parallel platform to transplant and optimize WENO.

3 Sunway TaihuLight System

The Sunway TaihuLight System has a theoretical peak performance of 125.4 Pflop/s with 10 649 600 cores and 1.31 PB of primary memory[13].

3.1 Sunway TaihuLight system architecture

The processor chip is composed of 4 Core Groups (CGs), connected via an NoC, see Fig. 2.

3.2 Comparing the Sunway TaihuLight system with other large-scale systems.

Table 1 provides a brief comparison between the Sunway TaihuLight and four other top systems of the

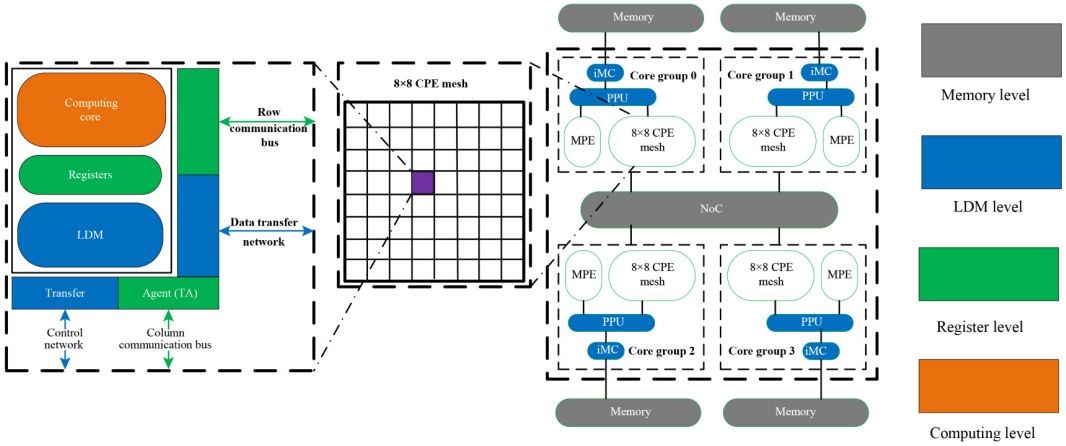

Fig. 2 CPU architecture of Sunway TaihuLight. Each of which includes an MPE and 64 Computing Processing Elements (CPEs) arranged in an 8 × 8 grid. The processor connects to other outside devices through a System Interface (SI)[13]. The CPE is composed of an 8 × 8 mesh of 64-bit RISC cores, supporting only user mode, with a 256-bit vector instructions, 16 KB L1 instruction cache, and 64 KB Scratch Pad Memory (SPM).

Table 1 Comparing the Sunway TaihuLight system with other large-scale systems.

System	Architecture	Peak performance	Linpack
Sunway TaihuLight	One 260-core Sunway CPU (4 MPEs and 256 CPEs)	125 Pflops	93 Pflops
Tianhe-2	Two 12-core Intel CPUs and three 57-core Intel (Xeon Phi Coprocessors)	100 Pflops	61 Pflops
Titan	One 16-core AMD CPU and one K20x NVIDIA GPU (2688 CUDA cores)	27 Pflops	18 Pflops
Sequoia	One 16-core PowerPC CPU	20 Pflops	17 Pflops
K computer	One 8-core SPARC64 CPU	11 Pflops	10 Pflops

TOP500[24] list in June, 2016. Compared with the previous top system, Tianhe-2, the TaihuLight system has double the peak performance, and almost triple the sustainable Linpack performance. The MPE-CPE hybrid architecture of the new Sunway system is also largely different from previous heterogeneous systems. While the MPE is similar to a CPU core, and the CPE cluster is similar to a many-core accelerator, both the CPU and accelerator are now fused onto one chip with a unified memory space, which is different from both the homogeneous clusters with only multi-core CPUs, such as *Sequoia* and *K computer*, and the heterogeneous clusters with both CPUs and PCIe-connected accelerators, such as *Tianhe*-2 and *Titan*.

4 Mapping WENO to the Sunway TaihuLight Supercomputer: Our General Methodology

4.1 General workflow of WENO

In fluid dynamics and engineering calculation, the design of hyperbolic conservation law equation is more than important, it is often in the solution area that some physical quantities have a great gradient transform or a discontinuous issue, such as water jumping issue. WENO, which serves as the Computational Fluid Dynamics, the computation workflow of WENO can be divided into two phases: computing area division and communication domain division. In computing area, as shown in Fig. 3.

The algorithm flow of the whole calculation process is as follows.

- Array initialization for WENO (the array in the program is called u, $u1$);
- WENO computing;
- Strcpy (u, $u1$);
- Array boundary exchange;
- If ($tt < end\_time$), goto WENO computing;
- If ($IStep = KStep\_save$) save data to disk.

4.2 MPE and CPE computer mode

There are four calculation modes for MPE and CPE in the Sunway processor: MPE-CPE synchronization mode, MPE-CPE asynchronous mode, MPE-CPE collaborative model, and MPE-CPE dynamic parallel model. In all four modes, the program is entered by the MPE, then called to accelerate from the CPEs. As shown in Fig. 4, the MPE-CPE asynchronization mode will be used in this paper, for the reason that MPE completes other computing, communication, or I/O operations while accelerating calculations from the CPE, thus improving the parallel efficiency of the

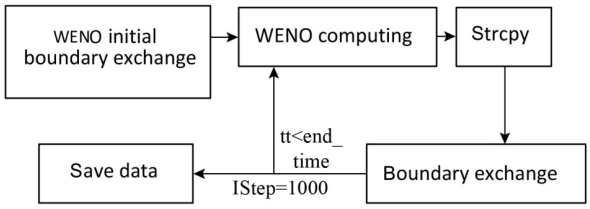

Fig. 3 WENO algorithm flow chart.

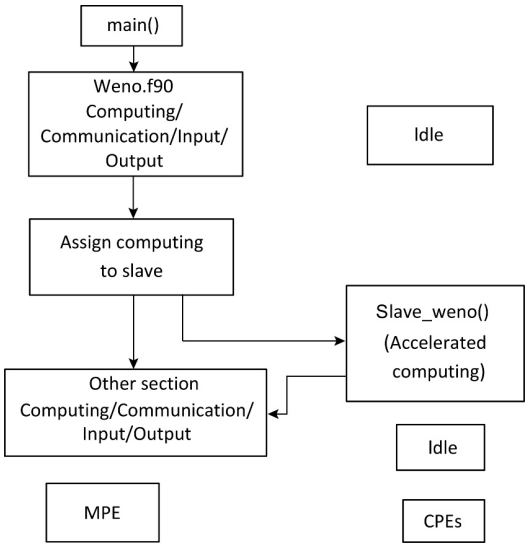

Fig. 4 MPE-CPE asynchronization mode.

master-slave collaboration.

4.3 Porting and optimizing

(1) *Algorithmic overview*

The basic concepts behind the algorithm are as shown in Eq. (1)

- If the high accuracy approaches u_x, it is necessary to use multiple base points;
- If the function in the base point is interrupted, it can cause oscillations;
- The interrupt cannot exist everywhere;
- Divide the base points into multiple groups (templates), and each template calculates the approximations of j dot derivatives independently, get multiple differences;
- Weight is given according to the smoothness of each template;
- Multiple difference results are weighted average. The smoother the template, the greater the weight. If there is a gap in a template, the weight tends to 0. If both are smooth, then combine them into higher order formats.

Figure 5 shows use of the base points $\{j-3, j-2, j-1, j, j+1, j+2, j+3\}$, divided into 4 groups (templates), having four base points per template. Construct a weighted expression of WENO flux. Due to the presence of high nonlinear gravity clustering in the universe, shock waves may occur in CFD. Thus, the discretization of the flux for solving the control equation is based on the seventh order finite difference[25, 26].

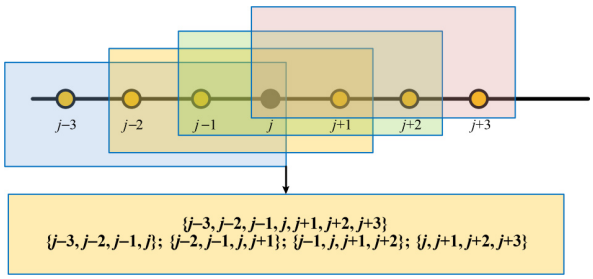

Fig. 5 Seven base points are divided into four groups. In 7th-order WENO scheme, we only used the maximum of 7-point values around, which means we need additional ghost cells to do the splitting for the original "ghost cells". Though the number of ghost-cell layers grows and the amount of end-to-end communication increases, it is still worthy that we successfully remove the global communications for the massive parallelization, and the experiments reveal though local flux splitting is fairly diffusive when applied to first- and second-order discretization, but for the 7th-order even high-order WENO discretization we adopt, the dissipation and dispersion errors of the local one are acceptable and it has very small numerical viscosity.

As an example, keeping the values for Y and Z constants (X, Y, and Z represent three dimensions, respectively. When the derivative operation is carried out in the direction of X dimension, Y, and Z are treated as constants.):

$$\left.\frac{\partial f(u)}{\partial x}\right|_{x=x_j} \approx \frac{1}{\Delta x}(f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}}) \tag{5}$$

The 7th order finite difference WENO scheme has the flux given by

$$f_{j+\frac{1}{2}} = w_1 f^{(1)}_{j+\frac{1}{2}} + w_2 f^{(2)}_{j+\frac{1}{2}} + w_3 f^{(3)}_{j+\frac{1}{2}} + w_4 f^{(4)}_{j+\frac{1}{2}} \tag{6}$$

where $f^{(i)}_{j+\frac{1}{2}}$ are fluxes on three different stencils given by

$$f^{(1)}_{j+\frac{1}{2}} = c_1 f(u_{j-3}) + c_2 f(u_{j-2}) + c_3 f(u_{j-1}) + c_4 f(u_j) \tag{7}$$

$$f^{(2)}_{j+\frac{1}{2}} = c_5 f(u_{j-2}) + c_6 f(u_{j-1}) + c_7 f(u_j) + c_8 f(u_{j+1}) \tag{8}$$

$$f^{(3)}_{j+\frac{1}{2}} = c_9 f(u_{j-1}) + c_{10} f(u_j) + c_{11} f(u_{j+1}) + c_{12} f(u_{j+2}) \tag{9}$$

$$f^{(4)}_{j+\frac{1}{2}} = c_{13} f(u_j) + c_{14} f(u_{j+1}) + c_{15} f(u_{j+2}) + c_{16} f(u_{j+3}) \tag{10}$$

We use more complex cubic decomposition patterns, exploring traffic for a range of patterns (one-decomposition and two-decomposition) of use. Although the process of cubic decomposition brings a more complex communication mode than one-dimensional[27] or two-dimensional decomposition, the number of ghost cells generated by the cube decomposition (Fig. 6) is smallest when the size of the sub-domain is the same. The best parallel granularity of one- or two-dimensional decomposition is limited side long and cubic decomposition is more flexible. In summary, the cubic decomposition method has the best scalability.

- For one-dimensional data decomposition, with spatial direction X, N_x, N_y, and N_z represent the length, width, and height of the object, respectively, and $N_y \times N_z$ represents the area of the cross section, communication size per subdomain is Comm_{one}:

$$\text{Comm}_{\text{one}} = 2N_yN_z.$$

- For two-dimensional data decomposition, with spatial directions X and Y divided by T and P, communication size per subdomain is Comm_{two}:

$$\text{Comm}_{\text{two}} = 2\frac{N_y}{P}N_z + 2\frac{N_x}{T}N_z.$$

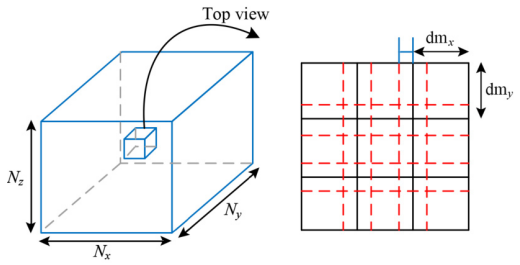

Fig. 6 3D blocking. MPI allows users to submit a data type for the data in discontinuous memory for communication, the communication mode should be six steps: (1) process (x, y, z) sends the custom-type updated local cells to process $(x+1, y, z)$ and at the same time receives the left side ghost cells from process $(x-1, y, z)$; (2) process (x, y, z) sends cells to process $(x-1, y, z)$ and receives the right side ghost cells from process $(x+1, y, z)$; (3) process (x, y, z) sends the custom-type updated local cells to process $(x, y+1, z)$ and at the same time receives the left side ghost cells from process $(x, y-1, z)$; (4) process (x, y, z) sends cells to process $(x, y, z-1)$ and receives the right side ghost cells from process $(x, y, z+1)$; (5) process (x, y, z) sends the custom-type updated local cells to process $(x, y, z+1)$ and at the same time receives the left side ghost cells from process $(x, y, z-1)$; and (6) process (x, y, z) sends cells to process $(x, y, z-1)$ and receives the right side ghost cells from process $(x, y, z+1)$. (For each subdomain, dm_x and dm_y represent the length and width of the subdomain, respectively).

• For three-dimensional data decomposition, with spatial directions X, Y, and Z divided by T, P, and H, communication size per subdomain is $\text{Comm}_{\text{three}}$:

$$\text{Comm}_{\text{three}} = 2\frac{N_y}{P}\frac{N_z}{H} + 2\frac{N_x}{T}\frac{N_y}{P} + 2\frac{N_x}{T}\frac{N_z}{H}.$$

Therefore we choose the cubic decomposition method to divide a 3D grid data and distribute these sub-domains and their ghost cells among the processors using MPI. Figure 6 reveals that in one sub-domain there are six-block ghost cells that require exchanging in every iterative step. To perform the computation, each block is loaded into the on-chip memory, and kernel computation is performed on grid elements that have all of their required stencil elements within the boundaries[28].

In order to ensure stable operation in non-smooth areas, we use flux splitting. Global splitting brings global communication, which results in large-scale parallelization of performance loss. Therefore, we chose to use local Lax-Friedrichs flux splitting to reduce global traffic.

(2) *Major challenges and our solutions*

In the Sunway system, we have an MPE and a group of 64 CPEs. Consequently, we are faced with the first challenge that the direct mapping from the *OpenMP* to *OpenACC* will not provide suitable parallelism for the CPE cluster.

Table 2 provides a brief comparison between the Sunway and Intel i7. The architecture of the Sunway processor is very different from that of Intel processor. There is a large gap between the memory and L3 cache, so we can optimize different programs according to the structure of Sunway processor. Therefore we require making full use of the 64 CPEs to strengthen the performance of the program. The CPE is currently only available in user mode, and interrupts are not supported from the kernel. From the core design it aims to achieve the aggregation of computing power.

The second challenge is that there are two paths of communication between the MPE and CPEs. One path is the *G*-load mode: the MPE and CPE share 8 GB of memory space, which can contain data that can be accessed directly from the CPE. The second path is the DMA approach, by which the transfer of data between MPE and CPE is performed by DMA (get or put). Regarding CPE internal communication for register communication, the CPE cluster is composed of an 8×8 mesh, therefore the register communication is limited to CPEs in the same row or the same column. Otherwise, the data exchange can be accomplished by two methods: CPE can broadcast to the same row or the same column through the put function, and also instruct the CPE to send data, or cyclic delivery can be used if a larger amount of data is transferred. This latter approach will take up a lot of registers, however, so we need to make some tradeoffs in the use of CPE registers, otherwise it will be counterproductive.

Therefore, in most cases, we require a buffering scheme that loads the proper data into the SPM.

For the first challenge to CPE cluster parallelism, we will adjust the calculation sequence and loop structure to aggregate sufficient computation and enable the correct number of parallel threads. For the data buffering and memory bandwidth constraints[29] (see Fig. 7), we improve the code to minimize the put intermediate variables into a limited on-chip buffer.

Table 2 Comparing the Sunway with Intel CPU.

Parameter	Intel i7 4770R	Sunway
Frequency	3.2 GHz	1.45 GHz
Memory	32 GB	8 GB
L1 cache	32 KB	32 KB
L2 cache	256 KB	256 KB
L3 cache	20 MB	No
Vector component	256 bit	256 bit

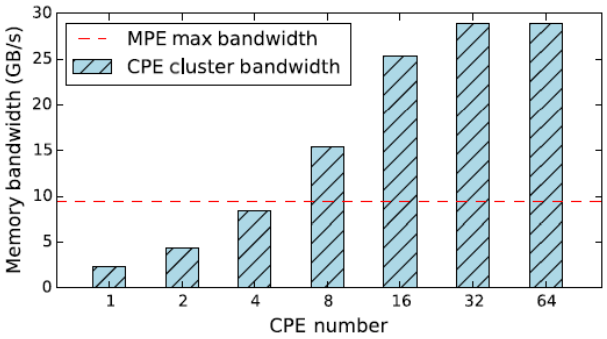

Fig. 7 Memory bandwidth benchmark of using different number of CPEs in a CPE cluster with chunk size of 256 bytes.

5 Optimization of WENO upon the Special Architecture of Sunway

In this section, we will describe the method to put the parallel aspect of the computing tasks onto the CPEs for execution, and simultaneously put the sequential part onto MPEs. Since the sequential portion is relatively small in WENO, the performance of the program can be greatly improved due to the calculation done by the CPEs.

5.1 LDM and DMA optimization

First, we port WENO onto MPE for execution, calculate the running time of each core function, and get the hotspot function (see Table 3), as shown in Algorithm 1.

The 64-bit CPEs of Sunway can directly access the memory, but direct access to memory will be slow, and the memory of the MPE that the 64-bit CPEs access will be congested, greatly increasing access latency. Every CPE has its own private Local Data Memory (LDM) programmable memory, and we attempt to reduce the frequency of CPE accessing MPE memory by using the LDM. The startup of the CPE loads some of the constant data through the establishment of a structure, into the LDM of each CPE. This data can be reused at a later time, thus reducing the frequency of memory space access and improving application performance.

Since WENO computing requires multiple iterations, and each cycle needs to call the CPE, we use DMA to exchange data between MPEs and CPEs, which has no impact on the report data being processed at the same time. The computation and communication can thereby overlap, leading to higher performance of the application.

Table 3 Time of Hotspot Function per iteration.

Hotspot	Time (s)
WENO7	0.300 532 102 5
Copy	$1.558\,184\,623\times10^{-2}$
Exchange boundary	$7.248\,878\,479\times10^{-3}$

Algorithm 1 Maunally timing using MPI_Wtime()

```
1:  t(1) = MPI_Wtime()
2:  call WENO7(u, u1, n_x, n_y, n_z, h_x, LAP, dt);
3:  t(2) = MPI_Wtime();
4:  if my_id.eq.0 then
5:     print*, "WENO7 time for this step is", t(2) − t(1);
6:  end if
7:  t(3) = MPI_Wtime();
8:  for each k ∈ [1, n_z] do
9:     for each j ∈ [1, n_y] do
10:       for each i ∈ [1, n_x] do
11:          u(i, j, k) = u1(i, j, k)
12:       end for
13:    end for
14: end for
15: t(4) = MPI_Wtime();
16: if my_id.eq.0 then
17:    print*, "u copy to u1 time for this step is", t(4) − t(3);
18: end if
19: t(5) = MPI_Wtime();
20: call exchange_boundary_x_standard(u, Iperiodic(1));
21: call exchange_boundary_y_standard(u, Iperiodic(2));
22: call exchange_boundary_z_standard(u, Iperiodic(3));
23: t(6) = MPI_Wtime();
24: if my_id.eq.0 then
25:    print*, "exchange_boundary step time", t(6) − t(5);
26: end if
```

As shown in Algorithm 2, regarding inter-subdomain communication, MPI calls are invoked by MPEs. Packing and unpacking are performed by CPEs via DMA. Edge and corner data are transferred together with face data using enlarged buffers[30]. After three rounds of communication between face-adjacent subdomains in x, y, and z directions, face, edge, and corner data arrive to their destinations shown as follows. (1) Round 1: Face data from the control region are packed into send buffers and sent to adjacent subdomain. During communication, face data in the second direction are packed into send buffer. After receipt, edge/corner data are copied to send buffer, where face data are already filled. (2) Round 2: During communication, face data in the third direction are packed into send buffer. (3) Round 3: Data are transferred in the third direction.

The programming language environment supports *C*, *C++* (for MPE only), *Fortran*, *MPI*, *OpenMP*, *Pthread*, and *OpenACC* for parallel programming. A parallel math kernel library that runs on one core group and an

Algorithm 2 WENO executes code on MPEs and CPEs

```
1: t(1) = MPI_Wtime();
2: s_param%n_x = n_x;
3: s_param%n_y = n_y;
4: s_param%n_z = n_z;
5: s_param%LAP = LAP;
6: s_param%in = loc(u(:, :, :));
7: call athread_spawn(slave_c_WENO7, s_param);
8: call athread_join();
9: t(2) = MPI_Wtime() ;
10: void c_WENO7_(slave_param_t hparam) {
11: DMA_SET_NS(&dma_get, DMA_GET, &reply);
12: DMA_SET_NS(&dma_put, DMA_GET, &reply);
13: DMA_NEW(dma_get, sizeof(slave_param_t));
14: doublev4 h_x, dt;
15: n_x = sparam.n_x;
16: n_y = sparam.n_y;
17: n_z = sparam.n_z; }
```

Athread interface (similar to *Pthread*) are also provided to help with programming the CPEs. Our code is written in *Fortran* for MPE and in *C* for CPE. It adopts the *Athread* interface to spawn and fork threads on the CPEs and to conduct DMA operations, and the $-O3$ compiler option is used.

5.2 Remove redundant data

The array *u1* is a copy of u, and after using CPEs to accelerate the WENO function, the process takes up more than 40% and needs optimization. Here, we delete the memory access and copy process of array *u1*. *u1* is a copy of u, which means that you only need to visit u when you calculate it. There is no need to repeat visiting *u1* because you can write back to u directly. Therefore *u1* is redundant and can be deleted.

5.3 Data refactoring

Due to the limitation of the size of the slave core's LDM space, not all of the data required from the core calculation can be transmitted to the LDM, resulting in a failure to provide enough data to keep the slave core in a task saturated state. Starting a DMA requires 300 cycles, and the transmitted data must be 32 bytes in size, in aligned and contiguous blocks of memory. As much data as possible need to be transmitted at once in order to reduce the number of DMA start ups. In this paper, some data structures in WENO are reconstructed, so that the data needing to be transferred is continuous and aligned and the speed of data exchange between the master and slave cores can be accelerated. This refactoring is as shown in Algorithm 3.

Algorithm 3 Data refactoring

```
1: doubledm[4 × 1024];
2: DMA_SET_SIZE(dma_get, n_x × 8);
3: DMA_SET_SIZE(dma_put, n_x × 8);
4: int i, j, k;
5: for i = tid; i < n_y × n_z; i+ = 64 do
6:    j = i;
7:    k = i ÷ n_y;
8:    DMA_GP(dma_get, in + j × n_x + k × n_y × n_z, lmd, n_x ×
      8, COUNT);
9:    dma_wait(reply, COUNT);
10:   DMA_GP(dma_put, out + j × (n_x + 2 × LAP), lmd, n_x ×
      8, COUNT);
11:   dma_wait(reply, COUNT);
12: end for
```

5.4 SIMD optimization

The vector length of the Sunway processor is 256 bit, which can load up to 4 double types at a time. The most common occurrence in WENO is location, but it is all three dimensions, and in order to leverage the advantages of SIMD, an additional dimension should be introduced. In this way, programming on four dimensions of the structure can accelerate the loading of the SIMD, but the amount of data is also increased accordingly. This has little impact on the MPE, but since the LDM is only 64 KB, this method of changing the data structure is not appropriate. Therefore, we only perform SIMD optimization for most calls in the WENO program. Since too many for loops in the program are dependent, original cycles are combined into one loop for SIMD optimization. Theoretically, this optimization can make a four-fold improvement in the performance of the entire program. After removing the copy of the array, WENO becomes the bottleneck, and the bandwidth is only 3.9 GB/s, which is limited in calculation and meets the requirements and conditions for quantization using the SIMD extension (see Algorithm 4) to the quantization acceleration core computing cycle. As shown in Fig. 8, each CPE reads four rows of data at one time and passes the array structure from Array Of Struct (AOS) to Struct Of Array (SOA) by the simd vshff function. Therefore we can use the SIMD instruction to calculate four rows at a time, and then the simd vshff function can be converted to the *AOS* structure[31].

5.5 Elimination of boundary exchange

After the boundary exchange removed, there was no dependency between the calculation of each row. After

Algorithm 4 Using the SIMD extension

```
// SIMD Before
for k = 1, n_z do
  for j = 1, n_y do
    for i = i_begin, i_end + 1 do
      S10 = [a11 × f(i − 3) + a12 × f(i − 2) + a13 × f(i − 1) + a14] × f(i);
      S11 = a21 × f(i − 2) − a22 × f(i − 1) + a23 × f(i) + a24 × f(i + 1);
      S12 = a31 × f(i − 1) + a32 × f(i) + a33 × f(i + 1) + a34 × f(i + 2);
      S13 = a41 × f(i) + a42 × f(i + 1) + a43 × f(i + 2) + a44 × f(i + 3);
    end for
  end for
end for
// SIMD After
for k = 1, n_z do
  for j = 1, n_y do
    for i = i_begin, i_end + 1 do
      f_v4[4 × i] = simd_vshff(t3, t1, 136);
      f_v4[4 × i + 1] = simd_vshff(t4, t2, 136);
      f_v4[4 × i + 2] = simd_vshff(t3, t1, 221);
      f_v4[4 × i + 3] = simd_vshff(t4, t2, 221);
      S10 = a11 × f_v4[i − 3] + a12 × f_v4[i − 2] + a13 × f_v4[i − 1] + a14 × f_v4[i];
      S11 = a21 × f_v4[i − 2] − f_v4[i − 1] + a23 × f_v4[i] + a24 × f_v4[i + 1];
      S12 = a31 × f_v4[i − 1] + a32 × f_v4[i] + f_v4[i + 1] + a34 × f_v4[i + 2];
      S13 = a41 × f_v4[i] + a42 × f_v4[i + 1] + a43 × f_v4[i + 2] + a44 × f_v4[i + 3];
    end for
  end for
end for
```

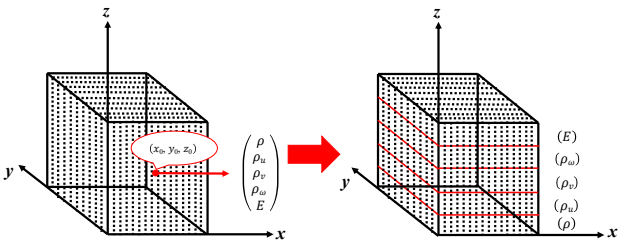

Fig. 8 AOS to SOA. One 3D array of five-component structure is changed to five 3D arrays.

the calculation was completed, the calculation of the cycle in the LDM could be obtained, and the time locality was increased and the visit was optimized as a result.

After the above optimization, the boundary exchange segment calculated at each step becomes the new performance hotspot. Each time step of the program must be repeated to the array u, and the time locality is poor and needs to be optimized. We found that the boundary of the u value is initialized to 0.0001, and will not change with each iteration. Thus the boundary exchange will always be 0.0001, which obviously can be removed, without having any impact on the original algorithm, thus improving the optimization. After the boundary exchange was removed, there was no dependency between the calculation of each row. After the calculation was completed, the calculation of the cycle in the LDM could be obtained, and the time locality was increased and the visit was optimized as a result.

6 Evaluation

6.1 Results of the kernels running on CPE clusters

In this section, we analyzed the run time of the various functions on the Sunway processor and analyzed the acceleration effect from using the CPEs. Figure 9 shows the acceleration ratio and scale of WENO. We compared the computational performance of the mixed version, which uses MPE and 64 CPEs, against that which uses only one MPE of the computer node. For computationally intensive kernels, such as the main part of the WENO function (92.78% of the run-time), we can achieve 12× to 41× acceleration. In particular, for the most time-consuming WENO function, we can achieve acceleration of 41×, making full use of the structural characteristics of CPEs. Of the rest of the functions, the boundary swap (2.35% of total run) and

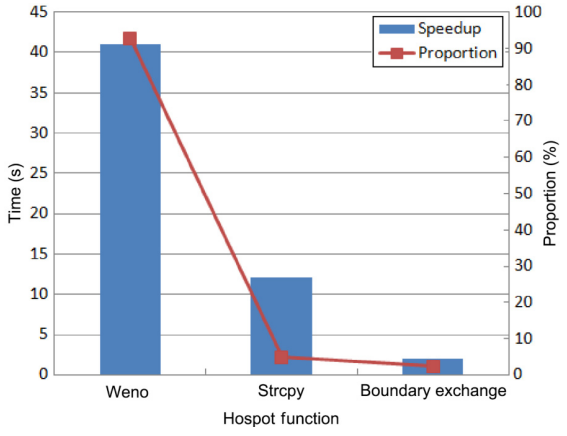

Fig. 9 Speedup of major kernels in WENO that we port onto the CPE clusters, and their proportions in the total runtime of the hotspot function. The speedup is comparing the performance of the kernel running on 4 MPEs and 4 CPEs against the performance of the kernel running on only 4 MPE.

the strcpy function (4.87% of total run) are usually memory-bound, but multithreading of four CPEs can still provide acceleration from 2× to 12×.

6.2 Performance of the entire WENO models

Parameter descriptions are provided in Table 4.

- The total grid number is $n_x \times n_y \times n_z$.
- n_x: The number of grids in the I-direction.
- n_y: The number of grids in the J-direction.
- n_z: The number of grids in the K-direction.
- n_{x0}: The parallel partition block in the i-direction.
- n_{y0}: The parallel partition block in the j-direction.
- n_{z0}: The parallel partition block in the k-direction.
- LAP: The length of the overlapping region of the parallel processing. Usually in the 7th or 8th order, LAP = 4. Setting up the convention results in a decrease in computational efficiency.
- KStep_save: Store the computation file every other KStep_save step.

The regional decomposition method is used to calculate and divide template into n_x, n_y, and n_z in three directions, for a total of $n_{x0} \times n_{y0} \times n_{z0}$.

6.3 Scalability test results

Figure 10 shows the machine scalability and Table 5

Table 4 Properties of coarse-grained data in CFD.

Parameter	Exp1	Exp2	Exp3
n_x	200	250	500
n_y	160	200	400
n_z	200	250	500
n_{x0}	1	1	1
n_{y0}	2	2	2
n_{z0}	2	2	2
LAP	4	4	4
End_time (s)	2.5×10^{0}	2.0×10^{0}	3.0×10^{-1}
KStep_save	1000	1000	300

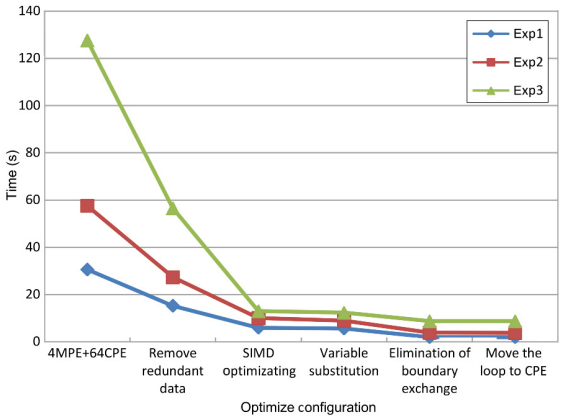

Fig. 10 Entire WENO that we port onto the CPE clusters with different optimize configurations.

Table 5 Properties of fine_grained data in CFD.

Parameter	Exp1	Exp2	Exp3
n_x	200	250	500
n_y	160	200	400
n_z	200	250	500
n_{x0}	4	4	4
n_{y0}	8	8	8
n_{z0}	8	8	8
LAP	4	4	4
End_time (s)	2.5×10^{0}	2.0×10^{0}	3.0×10^{-1}
KStep_save	1000	1000	300

shows the scale of WENO. We compared the computational performance of the mixed version, which uses 64 nodes (Table 6).

6.4 Performance result analysis

The speedup numbers we achieve for various WENO and boundary exchange defined by the above metrics, show that for the performance of the entire model, we can achieve a speedup of 172×.

7 Related Work

WENO is a solution to hyperbolic conservation laws in a group of high-precision generalized Godunov format methods in computational fluid dynamics. In recent years, we have started to see projects that refactor WENO for heterogeneous architectures, such as refactoring WENO on the CPU+GPU

Table 6 Execution time of the entire WENO that we port onto the CPE clusters with different optimize configurations (see Fig. 11). Similarly, we demonstrate both the time (seconds) for running on both MPE and CPE clusters.

Node number	MPE	CPE	Exp1	Exp2	Exp3
1	4	64	1.886	3.603	8.623
16	64	1024	0.149	0.325	0.612
64	256	4096	0.032	0.061	0.156

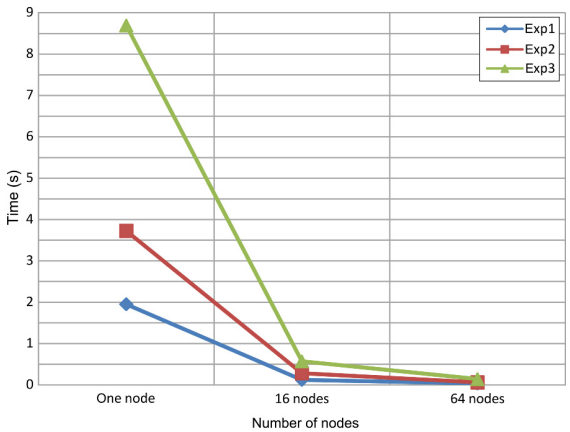

Fig. 11 Scalability test.

architecture[31], refactoring the community atmosphere model[12], phase field simulations of coarsening dynamics[30], 10 m-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics[32], molecular dynamics simulation[33], sea ice model algorithms[34], two compute-bound scientific kernels[35], etc., on the Sunway TaihuLight system.

8 Conclusion

In this paper, we report our contribution to optimizing the WENO model on the Sunway TaihuLight many-core supercomputer. Due to the differences between the Sunway many core processor (4 CGs, each of which consists of 1 MPE and 64 CPEs) and traditional multi-core CPUs, and especially the 64 KB SPM that requires explicit control by the user, we performed an extensive refactor of WENO to expose the right level of parallelism to the 64 CPEs in each CG. We analyzed characteristics of kernel functions, and proposed an appropriate heterogeneous parallel model. We also figured out the best division strategy for computing tasks, and implemented the parallel algorithm on Sunway TaihuLight. By using access optimization, data dependency elimination, and vectorization optimization, our parallel algorithm can achieve up to 172× speedup on one single node and 58× speedup on 64 nodes, with nearly linear scalability.

Acknowledgment

This paper was partially supported by the National High-Tech Research and Development (863) Program of China (No. 2015AA015306), the Science and Technology Plan of Beijing Municipality (No. Z161100000216147), the National Natural Science Foundation of China (No. 61762074), Youth Foundation Program of Qinghai University (No. 2016-QGY-5), the National Natural Science Foundation of Qinghai Province (No. 2019-ZJ-7034), and National Supercomputer Center in Wuxi, China.

References

[1] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes, III, *J. Comput. Phys.*, vol. 71, no. 2, pp. 231–303, 1987.

[2] X. D. Liu, S. Osher, and T. Chan, Weighted essentially non-oscillatory schemes, *J. Comput. Phys.*, vol. 115, no. 1, pp. 200–212, 1994.

[3] G. S. Jiang and C. W. Shu, Efficient implementation of weighted ENO scheme, *J. Comput. Phys.*, vol. 126, no. 1, pp. 202–228, 1996.

[4] J. C. Huang, H. Lin, T. J. Hsieh, and T. Y. Hsieh, Parallel preconditioned WENO scheme for three-dimensional flow simulation of NREL Phase VI Rotor, *Comput. Fluids*, vol. 45, no. 1, pp. 276–282, 2011.

[5] L. Thais, A. E. Tejada-Martínez, T. B. Gatski, and G. Mompeana, A massively parallel hybrid scheme for direct numerical simulation of turbulent viscoelastic channel flow, *Comput. Fluids*, vol. 43, no. 1, pp. 134–142, 2011.

[6] P. Kestener, F. Château, and R. Teyssier, Accelerating Euler equations numerical solver on graphics processing units, in *Int. Conf. Algorithms and Architectures for Parallel Processing ICA3PP*, C. H. Hsu, L. T. Yang, J. H. Park, and S. S. Yeo, eds. Springer, 2010, pp. 281–288.

[7] J. Tölke and M. Krafczyk, TeraFLOP computing on a desktop PC with GPUs for 3D CFD, *Int. J. Comput. Fluid Dynam.*, vol. 22, no. 7, pp. 443–456, 2008.

[8] X. J. Yang, X. K. Liao, K. Lu, Q. F. Hu, J. Q. Song, and J. S. Su, The TianHe-1A supercomputer: Its hardware and software, *J. Comp. Sci. Technol.*, vol. 26, no. 3, pp. 344–351, 2011.

[9] X. K. Liao, L. Q. Xiao, C. Q. Yang, and Y. T. Lu, MilkyWay-2 supercomputer: System and application, *Front. Comput. Sci.*, vol. 8, no. 3, pp. 345–356, 2014.

[10] F. Zhang, J. D. Zhai, B. S. He, S. H. Zhang, and W. G. Chen, Understanding co-running behaviors on integrated CPU/GPU architectures, *IEEE Trans. Parall. Distrib. Syst.*, vol. 28, no. 3, pp. 905–918, 2017.

[11] J. M. Dennis, M. Vertenstein, P. H. Worley, A. A. Mirin, A. P. Craig, and R. Jacob, Computational performance of ultra-high-resolution capability in the community earth system model, *Int. J. High Perform. Comp. Appl.*, vol. 26, no. 1, pp. 5–16, 2012.

[12] H. H. Fu, J. F. Liao, W. Xue, L. N. Wang, D. X. Chen, L. Gu, J. X. Xu, N. Ding, X. L. Wang, C. H. He, et al., Refactoring and optimizing the Community Atmosphere Model (CAM) on the sunway TaihuLight supercomputer, in *Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2016.

[13] H. H. Fu, J. F. Liao, J. Z. Yang, L. N. Wang, Z. Y. Song, X. M. Huang, C. Yang, W. Xue, F. F. Liu, F. L. Qiao, et al., The Sunway TaihuLight supercomputer: System and applications, *Sci. China Inform. Sci.*, vol. 59, no. 7, p. 072001, 2016.

[14] J. Binney, The stellar-dynamical oeuvre, *J. Astrophys. Astron.*, vol. 17, nos. 3&4, pp. 81–93, 1996.

[15] F. Grasso and S. Pirozzoli, Shock wave-thermal inhomogeneity interactions: Analysis and numerical simulations of sound generation, *Phys. Fluids*, vol. 12, no. 1, pp. 205–219, 2000.

[16] D. A. Jacobsen, J. C. Thibault, and I. Senocak, An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters, in *48^{th} AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Orlando, FL, USA, 2010.

[17] J. Y. Yang, S. C. Yang, Y. N. Chen, and C. A. Hsu, Implicit weighted ENO schemes for the three-dimensional incompressible Navier-Stokes equations, *J. Comput. Phys.*, vol. 146, no. 1, pp. 464–487, 1998.

[18] G. S. Jiang and D. P. Peng, Weighted ENO schemes for Hamilton-Jacobi equations, *SIAM J. Sci. Comput.*, vol. 21, no. 6, pp. 2126–2143, 2000.

[19] G. S. Jiang and C. C. Wu, A high-order WENO finite difference scheme for the equations of ideal magnetohydrodynamics, *J. Comput. Phys.*, vol. 150, no. 2, pp. 561–594, 1999.

[20] S. M. Liang and H. Chen, Numerical simulation of underwater blast-wave focusing using a high-order scheme, *AIAA J.*, vol. 37, no. 8, pp. 1010–1013, 1999.

[21] R. Liska and B. Wendroff, Composite schemes for conservation laws, *SIAM J. Numer. Anal.*, vol. 35, no. 6, pp. 2250–2271, 1998.

[22] P. Montarnal and C. W. Shu, Real gas computation using an energy relaxation method and high-order WENO schemes, *J. Comput. Phys.*, vol. 148, no. 1, pp. 59–80, 1999.

[23] S. Noelle, The MoT-ICE: A new high-resolution wave-propagation algorithm for multidimensional systems of conservation laws based on Fey's method of transport, *J. Comput. Phys.*, vol. 164, no. 2, pp. 283–334, 2000.

[24] TOP500 list of the world's top supercomputers, https://www.top500.org/lists/2016/06/, 2016.

[25] G. J. Shan and C. S. Wang, Efficient implementation of weighted ENO schemes, *J. Comput. Phys.*, vol. 126, no. 1, pp. 202–228, 1996.

[26] D. S. Balsara and C. W. Shu, Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy, *J. Comput. Phys.*, vol. 160, no. 2, pp. 405–452, 2000.

[27] D. A. Jacobsen, J. C. Thibault, and I. Senocak, An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters, in *48^{th} AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Orlando, FL, USA, 2010.

[28] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs, in *Proc. 2010 ACM/IEEE Int. Conf. High Performance Computing, Networking, Storage and Analysis*, New Orleans, LA, USA, 2010.

[29] H. Lin, X. C. Tang, B. W. Yu, Y. W. Zhuo, W. G. Chen, J. D. Zhai, W. W. Yin, and W. M. Zheng, Scalable graph traversal on sunway TaihuLight with ten million cores, in *2017 IEEE Int. Parallel and Distributed Proc. Symp.* (*IPDPS*), Orlando, FL, USA, 2017.

[30] J. Zhang, C. B. Zhou, Y. G. Wang, L. L. Ju, Q. Du, X. B. Chi, D. S. Xu, D. X. Chen, Y. Liu, and Z. Liu, Extreme-scale phase field simulations of coarsening dynamics on the sunway TaihuLight supercomputer, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2016.

[31] C. Meng, L. Wang, Z. Y. Cao, L. L. Feng, and W. S. Zhu, Large-scale parallelization based on CPU and GPU cluster for cosmological fluid simulations, in *Proc. 25^{th} Int. Conf. Parallel Computational Fluid Dynamics*, Changsha, China, pp. 207–220, 2014.

[32] C. Yang, W. Xue, H. H. Fu, H. G. You, X. L. Wang, Y. L. Ao, F. F. Liu, L. Gan, P. Xu, L. N. Wang, et al., 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2016, pp. 57–68.

[33] W. Q. Dong, L. T. Kang, Z. Quan, K. L. Li, K. Q. Li, Z. Y. Hao, and X. H. Xie, Implementing molecular dynamics simulation on Sunway TaihuLight system, in *2016 IEEE 18^{th} Int. Conf. High Performance Computing and Communications*, Sydney, Australia, 2016.

[34] B. Y. Li, B. Li, and D. P. Qian, PFSI.sw: A programming framework for sea ice model algorithms based on Sunway many-core processor, in *2017 IEEE 28^{th} Int. Conf. Application-Specific Systems, Architectures and Processors* (*ASAP*), Seattle, WA, USA, 2017.

[35] J. Lin, Z. G. Xu, A. Nukada, N. Maruyama, and S. Matsuoka, Optimizations of two compute-bound scientific kernels on the SW26010 many-core processor, in *2017 46^{th} Int. Conf. Parallel Processing* (*ICPP*), Bristol, UK, 2017.

Jianqiang Huang is an lecturer at Qinghai University, China. He is currently a PhD candidate in the Department of Computer Science and Technology, Tsinghua University. His research interests include high performance computing and graph computing systems.

Wentao Han received the bachelor and PhD degrees from Tsinghua University in 2008 and 2015, respectively. He is currently a postdoctoral researcher in the Department of Computer Science and Technology, Tsinghua University. His research interests include big data processing and neuromorphic systems.

Xiaoying Wang is a professor in the Department of Computer Technology and Applications, Qinghai University, China. She received the PhD degree from Tsinghua University in 2008. Her research interests include cloud computing and parallel computing.

Wenguang Chen received the BS and PhD degrees from Tsinghua University in 1995 and 2000, respectively. He is a professor and associate head in the Department of Computer Science and Technology, Tsinghua University. His research interest is in parallel and distributed computing and programming model.