Acceleration of three-dimensional Tokamak magnetohydrodynamical code with graphics processing unit and OpenACC heterogeneous parallel programming

H. W. Zhang[1], J. Zhu[1], Z. W. Ma[1]*, G. Y. Kan[2], X. Wang[1], W. Zhang[1]

[1] Institute for Fusion Theory and Simulation, Zhejiang University, Hangzhou 310027, China

[2] State Key Laboratory of Simulation and Regulation of Water Cycle in River Basin, Research Center on Flood & Drought Disaster Reduction of the Ministry of Water Resources, China Institute of Water Resources and Hydropower Research, Beijing 100038, P. R. China

Abstract: In this research, the OpenACC heterogeneous parallel programming model is successfully applied to modification and acceleration of the three-dimensional Tokamak magnetohydrodynamical code (CLT). The implementation of the OpenACC in CLT, performance test, and benchmarking are introduced in this paper. Combination of OpenACC and MPI technologies conveniently implements the multiple-GPUs parallel programming in CLT. Significant speedup ratios are achieved on NVIDIA TITAN Xp and TITAN V GPUs, respectively, with very few modifications to the source code. Furthermore, the validity of the double precision calculations on the above-mentioned two graphics cards has also been strictly verified with m/n=2/1 resistive tearing mode instability in Tokamak.

Keywords: OpenACC, MPI, GPU, magnetohydrodynamics (MHD)

* Author to whom correspondence should be addressed: zwma@zju.edu.cn

1. Introduction

Computational simulation is more and more popular in magnetic confined fusion researches because the devices, such as Tokamak[1] and Stellarator[2], are too complex to be studied analytically. Considering the restrict of computational capabilities of the single processor core, parallel programming plays a major role in the research and application fields of program acceleration. Multiple parallel acceleration methods, such as MPI[3] and OpenMP[4], have been developed in past decades. In recent years, benefitting from the rapid performance improvement of the graphics processing unit (GPU), new parallel programming methods and tools, such as CUDA[5], OpenCL[6], and OpenACC[7], have been developed vigorously.

OpenACC as a user-driven directive-based performance-portable parallel programming model, is developed to simplify the parallel programming for scientists and engineers. Compared with the CUDA and OpenCL which require great efforts on code redevelopment, OpenACC has many advantages, such as satisfactory acceleration effect with very few modifications on an original serial source code and a good compatibility with other devices, for example central processing unit (CPU). It has been successfully applied in some scientific and engineering codes, like the flow code NeK5000[8], the computational electromagnetics code Nekton[9], the Gyrokinetic Toroidal Code (GTC)[10], the Rational Hybrid Monte Carlo (RHMC) QCD code[11], and the C++ flow solver ZFS[12], etc. The application perspective of the OpenACC technology in other scientific and engineering areas is very good and attractive.

The CLT code (CLT is the abbreviation of Ci-Liu-Ti, which means magnetohydrodynamics in Chinese) is recently developed for the simulation study of magnetohydrodynamics (MHD) instability in toroidal devices like Tokamak[13, 14]. Beset by the poor parallel computational efficiency in accelerating the MHD code with MPI and OpenMP, on the contrary, OpenACC was successfully applied in CLT acceleration in our research. We have obtained satisfactory acceleration effect on NVIDIA TITAN Xp and TITAN V GPUs. Compared with the execution speed of the MPI-parallelized CLT source code executed on 64-core CPUs (Intel® Xeon® Gold 6148F), about four times of acceleration has been achieved on a single TITAN V and

double TITAN Xp GPUs. Besides, the correctness of the simulation result on the GPU is also strictly checked and ensured.

The detailed implementation of OpenACC and its benchmarking will be introduced in this paper. The paper is organized as follows: In Section 2, the MHD model and the main modules of CLT will be introduced. In Section 3, the OpenACC implementation combined with MPI will be given and the acceleration performance of CLT with OpenACC is analyzed in Section 4. Section 5 analyses the benchmarking for the correctness of the OpenACC accelerated code. Finally, conclusions and discussion are given in Section 6.

2. Overview of CLT

CLT is an initial value full MHD code in toroidal geometries focusing on MHD stabilities in toroidal devices like Tokamak. The code is written in FORTRAN 90 with double precision format and it has grown to more than 20,000 code lines with hundreds of subroutines and subfunctions. CLT has been successfully applied in studying the influence of toroidal rotation[13], external driven current[15, 16], and Hall effect[14] on resistive tearing modes in tokamaks. Its hybrid kinetic-magnetohydrodynamic version code, CLT-K, has also been developed to investigate nonlinear dynamics of toroidal Alfvén eigenmodes in Tokamak[17, 18].

2.1 Basic MHD equations

The full set of resistive MHD equations including dissipations can be written as follows:

$$\partial_t \rho = -\nabla \cdot (\rho \mathbf{v}) + \nabla \cdot [D \nabla (\rho - \rho_0)], \tag{1}$$

$$\partial_t p = -\mathbf{v} \cdot \nabla p - \Gamma p \nabla \cdot \mathbf{v} + \nabla \cdot [\kappa (p - p_0)], \tag{2}$$

$$\partial_t \mathbf{v} = -\mathbf{v} \cdot \nabla \mathbf{v} + (\mathbf{J} \times \mathbf{B} - \nabla p) / \rho + \nabla \cdot [\nu (v - v_0)], \tag{3}$$

$$\partial_t \mathbf{B} = -\nabla \times \mathbf{E}, \tag{4}$$

with

$$\mathbf{E} = -\mathbf{v}\times\mathbf{B} + \eta(\mathbf{J} - \mathbf{J_0}) + \frac{d_i}{\rho}(\mathbf{J}\times\mathbf{B} - \nabla p)\,, \tag{5}$$

$$\mathbf{J} = \nabla\times\mathbf{B}\,, \tag{6}$$

where ρ, p, $\mathbf{v}$, $\mathbf{B}$, $\mathbf{E}$, and $\mathbf{J}$ are the plasma density, thermal pressure, plasma velocity, magnetic field, electric field, and current density, respectively. $\Gamma(=5/3)$ is the ratio of specific heat of plasma.

All variables are normalized as follows:

$\mathbf{B}/B_m \to \mathbf{B}$, $\mathbf{x}/a \to \mathbf{x}$, $\rho/\rho_m \to \rho$, $\mathbf{v}/v_A$, $t/\tau_a \to t$, $p/(B_m^2/\mu_0) \to p$, $\mathbf{J}/(B_m/\mu_0 a) \to \mathbf{J}$, $\mathbf{E}/(v_A B_m) \to \mathbf{E}$, and $\eta/(\mu_0 a^2/\tau_a) \to \eta$,

where $\tau_a = a/v_A$ is the Alfvenic time; $v_A = B_m/(\mu_0\rho_m)^{1/2}$ is the Alfvenic speed; B_m and ρ_m are the magnetic field and plasma density at the magnetic axis, respectively; a is the minor radius in the poloidal cross-section.

Constrained by the equilibrium conditions, the following equations should be satisfied:

$$\nabla\cdot(\rho_0\mathbf{v}_0) = 0\,, \tag{7}$$

$$\mathbf{v}_0\cdot\nabla p_0 + \Gamma p_0\nabla\cdot\mathbf{v}_0 = 0\,, \tag{8}$$

$$\rho_0\mathbf{v}_0\cdot\nabla\mathbf{v}_0 = \mathbf{J}_0\times\mathbf{B}_0 - \nabla p_0\,, \tag{9}$$

$$\nabla\times\mathbf{E}_0 = 0\,. \tag{10}$$

Substituting these equilibrium equations into Equations(1)-(4), the MHD equations can be rewritten as

$$\partial_t\rho = -\nabla\cdot(\rho\mathbf{v}_1 + \rho_1\mathbf{v}_0) + \nabla\cdot[D\nabla(\rho - \rho_0)]\,, \tag{11}$$

$$\partial_t p = -\mathbf{v}_1\cdot\nabla p - \mathbf{v}_0\cdot\nabla p_1 - \Gamma(p\nabla\cdot\mathbf{v}_1 + p_1\nabla\cdot\mathbf{v}_0) + \nabla\cdot[\kappa(p - p_0)]\,, \tag{12}$$

$$\begin{aligned}\partial_t\mathbf{v} = &-(\mathbf{v}\cdot\nabla\mathbf{v}_1 + \mathbf{v}_1\cdot\nabla\mathbf{v}_0 + \rho_1\mathbf{v}_0\cdot\nabla\mathbf{v}_0/\rho)\\ &+(\mathbf{J}_1\times\mathbf{B} + \mathbf{J}_0\times\mathbf{B}_1 - \nabla p_1)/\rho + \nabla\cdot[\nu(\nu - \nu_0)]\,,\end{aligned} \tag{13}$$

$$\partial_t \mathbf{B} = -\nabla \times \mathbf{E}_1, \tag{14}$$

where the variables with subscript 0 represent equilibrium components and 1 for perturbation components, e.g., $\mathbf{v}_1 = \mathbf{v} - \mathbf{v}_0$. Thus, numerical errors from equilibrium can be minimized.

2.2 Coordinate systems and numerical schemes

In CLT, the cylindrical coordinate system (R, ϕ, Z), as demonstrated in Figure 1, is applied in this study. Besides the outer magnetic surface, an m/n=3/1 magnetic island inside calculated by CLT is also plotted in the three dimensions. In the toroidal geometry device like Tokamak, R, ϕ and Z indicate the major radius, toroidal, and up-down directions, respectively. One advantage of this coordinate system is that we can avoid the singularity near the $r = 0$ point that occurs in the toroidal coordinate (ψ, θ, ζ). However, the outer boundary handling would be more difficult in the cylindrical coordinate. In the last version of CLT, the fixed boundary condition at the last flux surface of plasma is assumed. Recently, the cut-cell method[19] has been applied in CLT successfully, the details of the new boundary handling method will be introduced in another paper.

The grids are dispersed in the R, ϕ, and Z directions and are rectangular in the $R-Z$ plane. The 4th order finite difference method is employed in the R and Z directions, while in the ϕ direction, either finite difference or pseudo-spectrum method can be used. As for the time-advance, the 4th order Runge-Kutta scheme is applied.

For the parallelization on CPU platform, the simulation domain can be divided into multiple blocks in each direction, and four-layer grids are used for message passing between every two neighboring blocks according to the 4th order finite difference method spatially. Thus, the increase of MPI CPU cores with a fixed problem size will lead to rapid deterioration of the acceleration efficiency.

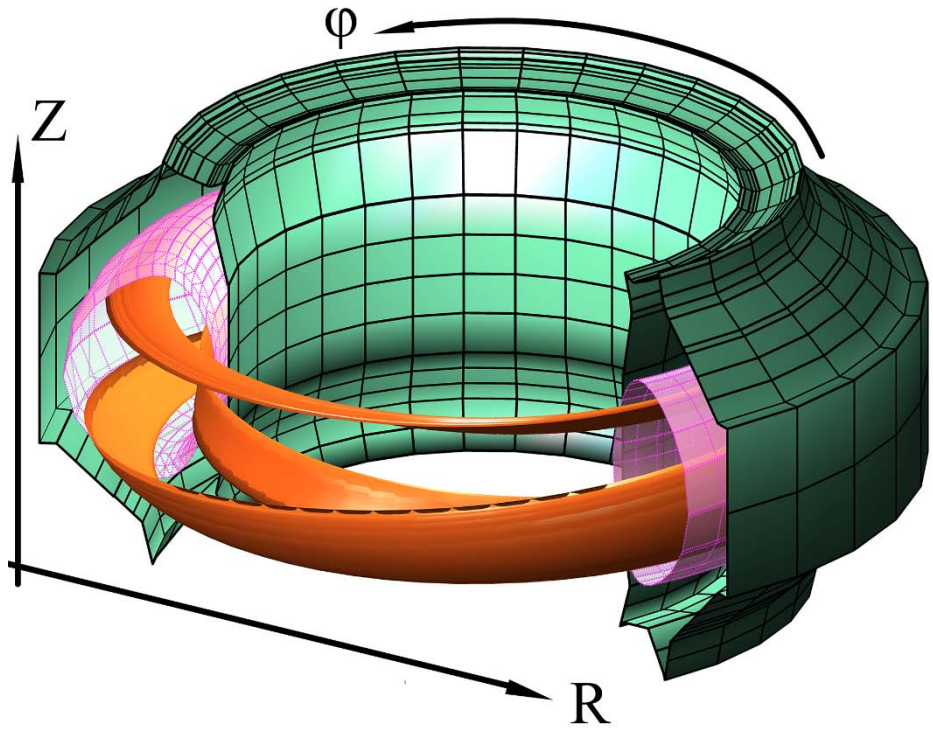

Fig. 1. Sketch of the cylindrical coordinate system (R, φ, Z) in Tokamak. An m/n=3/1 magnetic island inside the outer magnetic surface calculated by CLT is plotted in three dimensions.

2.3 Main modules and basic execution flow of CLT

For each iterative step of CLT, the eight components of $\rho, p, \mathbf{v}, \mathbf{B}$ need to be updated. The main processes of CLT is shown in Figure 2. The modules with red squares are executed on CPU, while the blue ones on GPU, and the green squares mark out the main OpenACC directives in the whole procedure. The **INITIAL** module reads the equilibrium files that are calculated by equilibrium code, like NOVA[20] and EFIT[21]. Then the **DERIVATIVES OF** $\rho, p, \mathbf{v}, \mathbf{B}$ module using the 4th order finite difference method is called to obtain the derivatives. With these derivatives, the values of current and electric field $\mathbf{J}, \mathbf{E}$ are obtained. Then the derivatives, $\mathbf{J}$ and $\mathbf{E}$, are substituted from the **RIGHT HANDS OF EQUATIONS**, and finally $\rho, p, \mathbf{v}, \mathbf{B}$ are updated in **STEP ON** module with the 4th order Runge-Kutta scheme. Note that during the iteration the **BDOUNDARY** module is applied in each step to solve the boundary conditions and deal with data exchange due to MPI

parallelization, while the **TIME STEP** module calculate the time step duration based on the latest $\rho, p, \mathbf{v}, \mathbf{B}$ to satisfy the Courant-Friedrichs-Lewy (CFL) condition on all grids, and some other modules like **DIAGN** (data analysis) and **OUTPUT** (data record) are called every few steps.

The execution time of CLT is mostly spent on the **RIGHT HANDS OF EQUATIONS**, **DERIVATIVES OF** $\rho, p, \mathbf{v}, \mathbf{B}$, and $\mathbf{J}, \mathbf{E}$ modules. Therefore, a large majority of OpenACC directives are added in these modules.

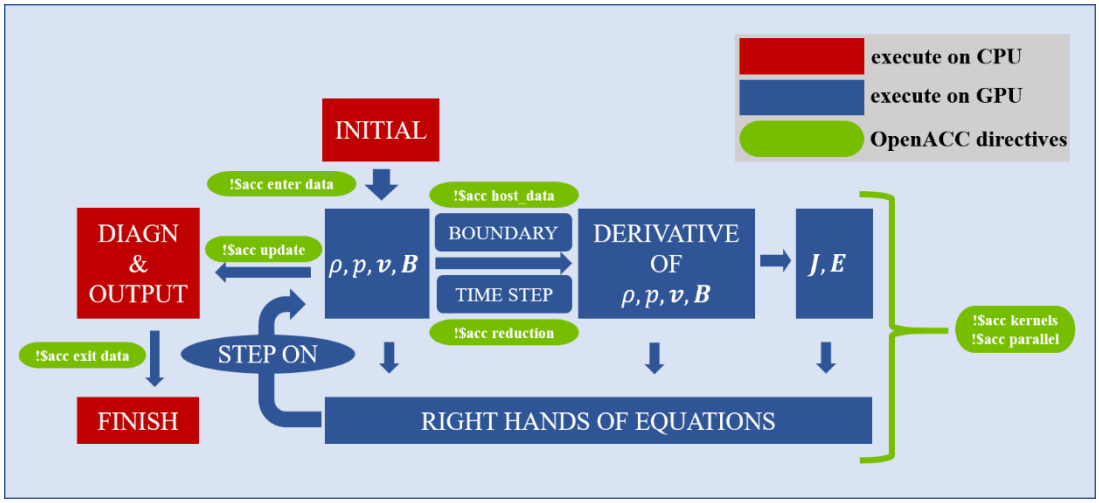

Fig. 2. Main modules and OpenACC directives of CLT. The modules in the blue blocks are executed on GPU, and those in the red blocks are on CPU. The green blocks indicate the OpenACC directives applied in the corresponding position.

3. OpenACC and MPI implementations in CLT

3.1 OpenACC directives added to the serial version of CLT

The main idea of GPU-acceleration using OpenACC is to offload the calculation process from CPU to GPU. The data transfer speed and bandwidth between the memories of CPU and GPU is much slower than that of the memory access. Thus, the first important operation is to copy all variables required in iteration from the CPU memory to the GPU memory. After the **INITIAL** module has read the equilibrium files, all necessary variables are copied into the GPU memory by using *'$acc enter data copyin (variable name list)'* directive and clause as shown in Figure 2. Accordingly, after finishing all iterations, the *'$acc exit data delete (variable name list)'* directive and clause are also used before the program finishing to free

the GPU memory. There are also other methods playing the same role, such as combination of *'!$acc declare create (variable name list)'* and *'!$acc update device (variable name list)'*. The main advantage of *'enter data'* and *'declare create'* construct is that the data lifetime in the GPU memory can be extended crossing subfunctions and subroutines. With this feature, the calculation on the GPU can be continuously proceeded without stopping for data transfer between CPU and GPU.

A demonstrative example for the *'enter data'* constructs in CLT is given in Figure 3. For simplicity, only three variables (xx, yy, zz) are listed in the OpenACC *'copyin'* clause. During the main iterations, CLT will have to call the **DIAGN** and **OUTPUT** modules every few steps to analyze data and record it into hard disk by CPU. Because **DIAGN** is quite complex and depends on some external math libraries, such as MKL[22], this module is still executed on CPU for the difficulty in parallelization with OpenACC. Therefore, just before calling the **DIAGN** and **OUTPUT** modules, the directive *'!$acc update host (variable name list)'* is used to copy the latest data back from GPU to CPU. The time wasted on updating operation is negligible because the number of calls on these two modules is much smaller than the total iteration steps.

```
PROGRAM CLT
! INITIALIZE
CALL INITIA
!$acc enter data copyin(xx, yy, zz)
! MAIN ITERATION PART
DO NSTEP = 1, NSTEP
     CALL STEPON
     IF(MOD(NSTEP,NDIAGN) == 0) THEN
          !$acc update host(xx, yy, zz)
          CALL DIAGN
     ENDIF
     IF(MOD(NSTEP,NOUTPUT) == 0) THEN
          !$acc update host(xx, yy, zz)
          CALL OUTPUT
     ENDIF
ENDDO
!$acc exit data delete(xx, yy, zz)
END PROGRAM CLT
```

Fig. 3. A simple example for *'enter data'* and *'update'* constructs in CLT.

Inside the **STEP ON** module (calculating all intermediate variables and then updating the field quantities), simple transversal differential calculations on each grid for the right hand of the equations occupy the major part. In most cases, three levels of loops are used in the code for the three-dimensional situation. To parallelize the loops, two constructs of *'kernels'* and *'parallel'* are frequently used. A section for parallelization of the **TIME STEP** module is given in Figure 4 as a representative example. The *'present'* clause following *'parallel'* construct directive indicates the compiler that the variables inside the parentheses already exist on the GPU, so as to avoid the needless implicit variable copy or create operations on the GPU. The *'loop'* clause after *'parallel'* directive should be placed just before the loop body, this clause tells the compiler the closely followed loop should be parallelized. As for the **TIME STEP** module, the final time step should satisfy the CFL condition in each grid, therefore, the reduction for the minimum time step is used by adding *'reduction(min:dt1)'*. Other reduction operations are also supported in OpenACC with similar syntaxes[7]. The *'independent'* clause tells the complier that the calculation on each grid has no dependency and therefore can be parallelized directly. The final clause in Figure 4 is *'collapse(3)'*. Collapsing loops means that, for example, three loops of trip counts NX, NZ, and NY will be respectively and automatically turned into a single loop with a trip total count of NX*NZ*NY, which is usually beneficial to the parallelization. Note that the *'parallel'* constructs used here can also be replaced by *'kernels'* construct, the major difference of these two constructs is that *'kernels'* gives the compiler more freedom to find and map parallelism according to the requirements of the target accelerator, while the parallel construct is more explicit, and requires more analysis by the programmer.

```
SUBROUTINE SETDT
USE DECLARE
INCLUDE 'MPIF.H'
DT1=100.d0
!$acc parallel present(x, xx, yy, zz, gdtp_ep)
!$acc loop reduction(min:dt1) independent collapse(3)
DO JY = IY_FIRST + 2, IY_LAST - 2
DO JZ = IZ_FIRST + 2, IZ_LAST - 2
DO JX = IX_FIRST + 2, IX_LAST - 2
     IF(GDTP_EP(JX,JZ,1).NE.4) THEN
          VX=X(JX,JZ,JY,3)
          VY=X(JX,JZ,JY,4)
          VZ=X(JX,JZ,JY,5)
          VA2=(X(JX,JZ,JY,6)**2+X(JX,JZ,JY,7)**2  &
              +X(JX,JZ,JY,8)**2)/X(JX,JZ,JY,1)
          CS2=GAMMA*X(JX,JZ,JY,2)/X(JX,JZ,JY,1)

          VPX=DABS(VX)+SQRT(DABS(CS2+VA2))
          VPY=DABS(VY)+SQRT(DABS(CS2+VA2))
          VPZ=DABS(VZ)+SQRT(DABS(CS2+VA2))

          DTX=DABS(XX(JX)-XX(JX-1))/(VPX/CFL)
          DTZ=DABS(ZZ(JZ)-ZZ(JZ-1))/(VPZ/CFL)
          DTY=DABS(XX(JX)*(YY(JY)-YY(JY-1)))/(VPY/CFL)

          DT2=DMIN1(DTX,DTZ)
          DT3=DMIN1(DTY,DT2)
          DT1=DMIN1(DT1,DT3)
     ENDIF
ENDDO
ENDDO
ENDDO
!$acc end parallel

RETURN
END SUBROUTINE SETDT
```

Fig. 4. A section for the parallelization of the **TIME STEP** module using *'parallel'* constructs.

Another important OpenACC directive used in CLT is the *'!$acc routine'* directive for the procedure call. Figure 5 gives the solution when a subroutine or function is called inside an accelerated region. The subroutine *'interp1d2l'* calculates a simple interpolation with a given data series, and is commonly used in the accelerated region of **BOUNDARY** module.

The *'!$acc routine seq'* directive requires the compiler to generate a device version for this subroutine on the GPU so that it can be called in the accelerated region. An interface for this child subroutine is required inside the parent as given in Figure 5. Then the subroutine can be called inside the OpenACC accelerated region directly.

```
SUBROUTINE INTERP1D2L(X1, X2, X3, Y1, Y2, Y3, Y, ANS)
!$acc routine seq
REAL*8 X1,X2,X3,Y1,Y2,Y3,Y,ANS
REAL*8 D1,D2,D3
D1 = (Y1-Y2)*(Y1-Y3)
D2 = (Y2-Y3)*(Y2-Y1)
D3 = (Y3-Y1)*(Y3-Y2)
ANS = X1*(Y-Y2)*(Y-Y3)/D1 &
    + X2*(Y-Y3)*(Y-Y1)/D2 &
    + X3*(Y-Y1)*(Y-Y2)/D3

RETURN
END SUBROUTINE INTERP1D2L

SUBROUTINE BOUNDARY
USE DECLARE
IMPLICIT NONE
INCLUDE 'MPIF.H'

! INTERFACE FOR ACCELERATED SUBROUTINE
INTERFACE
     SUBROUTINE INTERP1D2L(X1, X2, X3, Y1, Y2, Y3, Y, ANS)
     !$acc routine seq
     REAL*8 X1,X2,X3,Y1,Y2,Y3,Y,ANS
     REAL*8 D1,D2,D3,TMP_ADD
     END SUBROUTINE
END INTERFACE

! OpenACC ACCELERATED REGION
......
CALL INTERP1D2L(X1, X2, X3, Y1, Y2, Y3, Y, ANS)
......
END SUBROUTINE BOUNDARY
```

Fig. 5. An example for the procedure (subroutine interp1d2l) call inside OpenACC accelerated region.

3.2 Combination of OpenACC and MPI technologies for parallelization on multiple

GPUs

CLT was developed with MPI API at first for parallelization in clusters. For ordinary cases we used (256, 256, 32) grids in the (R, Z, ϕ) directions, the GPU memory usage in graphics card is about 2.25GB, thus, the memory in a single graphics card like TITAN Xp or TITAN V (with 12GB of GPU memory) is more than enough for a typical CLT case. However, considering the future extensibility for studying larger size problems like high n modes in Tokamak, a size of (1024, 1024, 128) grids or even larger will be necessary. In such scenarios, more than 64 times memory (over 144GB) will be required on the GPU and the multiple GPU parallelization will be indispensable. Based on the original MPI API in CLT, we also added the directives for the directly commutation between devices without transferring data back to the host.

As shown in Figure 6, firstly, we add the *'!$acc set device_num (rank of GPU)'* directive just after the *'mpi_comm_size'* and *'mpi_comm_rank'* calls. The value of *'rank of GPU'* can be defined as functions of the process rank in MPI parallelization so that each MPI rank can be assigned to a different GPU for the load balance. With this *'rank of GPU'*, the commutation between different GPUs becomes possible. Then, a specific example with OpenACC directives in *'mpi_send'* and *'mpi_recv'* API is given. The *'!$acc host_data use_device (variable name list) if_present'* tells the compiler to use the address of the specified variables on GPU if the variables are present, then the *'mpi_send'* or *'mpi_recv'* API is executed by accessing the data from the GPU address of variable rather the CPU. The using of *'host_data use_device'* avoids the unnecessary data transfer between CPU and GPU before MPI calls. For the purpose of comparison, an equivalent example is given is Figure 7, where the MPI API still accesses the data from the CPU address for specified variables. However, in order to update the data inside the CPU and GPU, another pair of *'!$acc update host (variable name list)'* and *'!$acc update device (variable name list)'* are added before the *'mpi_send'* and after the *'mpi_recv'*, respectively. The functions of the methods used in Figure 6 and Figure 7 are the same, but the later one will spend additional time on data updating between CPU and GPU.

```
PROGRAM CLT
......
! INITIATE MPI
CALL MPI_INIT(IERROR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NSIZE,IERROR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,NRANK,IERROR)
!$acc set device_num(0)
......
!MPI SEND-----------------------------------------------
!$acc host_data use_device(wfx1) if_present
CALL MPI_SEND(WFX1,MYZ8,MPI_DOUBLE_PRECISION, NRANK+1, 0, &
          MPI_COMM_WORLD, IERROR)
!$acc end host_data
......
!MPI RECEIVE--------------------------------------------
!$acc host_data use_device(wfx1) if_present
CALL MPI_RECV(WFX1,MYZ8,MPI_DOUBLE_PRECISION, NRANK-1, 0, &
          MPI_COMM_WORLD, IERROR)
!$acc end host_data
......
END PROGRAM CLT
```

Fig. 6. An example for MPI API called on GPU by using *'host_data use_device'* construct.

```
PROGRAM CLT
......
! INITIATE MPI
CALL MPI_INIT(IERROR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NSIZE,IERROR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,NRANK,IERROR)
!$acc set device_num(0)
......
!MPI SEND-----------------------------------------------
!$acc update host(wfx1)
CALL MPI_SEND(WFX1,MYZ8,MPI_DOUBLE_PRECISION, NRANK+1, 0, &
          MPI_COMM_WORLD, IERROR)
......
!MPI RECEIVE--------------------------------------------
!$acc update device(wfx1)
CALL MPI_RECV(WFX1,MYZ8,MPI_DOUBLE_PRECISION, NRANK-1, 0, &
          MPI_COMM_WORLD, IERROR)
......
END PROGRAM CLT
```

Fig. 7. An example for MPI API called on CPU by using *'update'* construct.

4. Acceleration performance analysis

4.1 Performance on single GPU card

For the case with grids as (256, 256, 16) in R, Z, and ϕ, the comparison for execution time (with 20,000 steps) is carried out on different platforms. Some analysis results are demonstrated in Figure 8. The MPI block division for the CPU version of CLT is carried out uniformly in the R and Z directions, while no block division is applied in the GPU version because we only used one card for acceleration. Note that for the case executing on the CPU, the 'O3' compiler option is turned on so as to obtain the best performance optimization. And the test on CPU is carried out on the cluster KYLIN-2 in our institute, with total 138 nodes and 2 Intel® Xeon® Gold 6148F CPUs for each node (40 cores in each node), while the case for the GPU test is performed on several workstations with single or double NVIDIA GPUs installed.

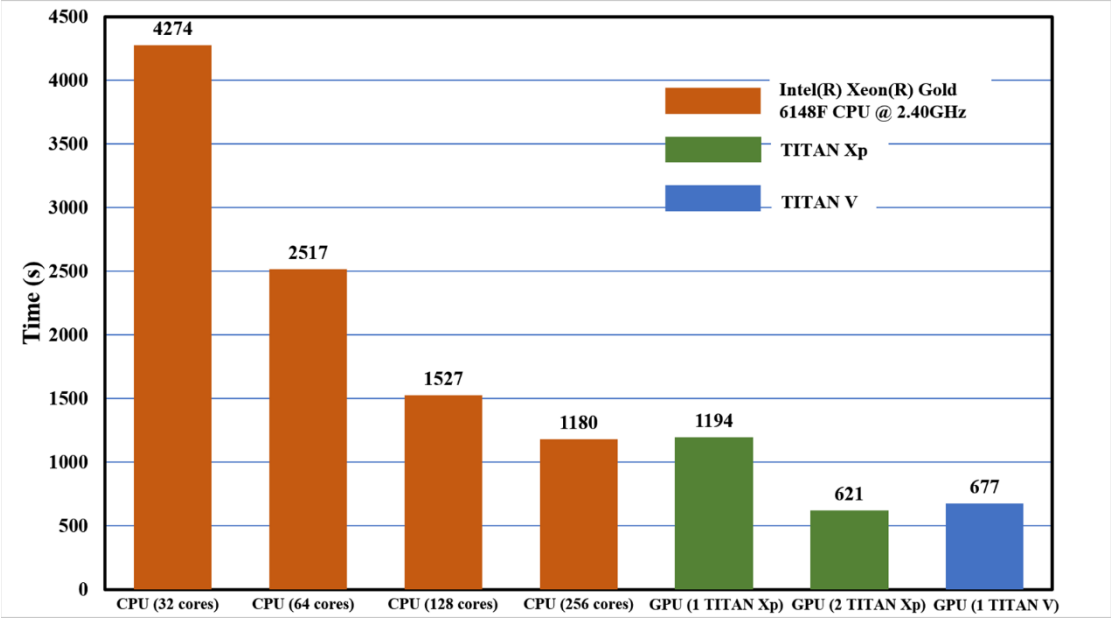

Fig. 8. Comparison for execution time on different platforms (CPU colored by orange, TITAN Xp by green, and TITAN V by blue).

As we can see in Figure 8, the performance of the code based on MPI executed on CPU is still acceptable when total 128 cores (4 nodes) are used, with the parallel acceleration efficiency at 70%. However, the efficiency drops quickly to 45% when the number of CPU

cores is increased to 256 (7 nodes). The reason for the decline of parallel acceleration efficiency with the increase of CPU cores is mainly due to the growth of the overlap area between different MPI blocks with fourth order central difference scheme in space. Therefore, the speedup for MHD code with a fixed problem size is quite difficult on traditional MPI frame. Fortunately, the application of OpenACC based on GPU on our CLT results in a quite good acceleration performance. As the results shown in Figure 8 with the green and blue bars for TITAN Xp and TITAN V, respectively, the elapsed time on TITAN Xp almost equals to that of 256 Intel® Xeon® Gold 6148F CPU cores. What's more, the speed of TITAN V is even 43% faster than that of TITAN Xp. Compared with our frequently-used MPI set of 64 cores, the speedup of TITAN V is about 3.7 times. In addition to the speed performance, another advantage for the GPU acceleration is the much lower price for construct GPU workstations or servers than building a cluster with CPU nodes. The cost of a single TITAN V is comparable with one Intel® Xeon® Gold 6148F CPU (20 cores in each), thus, the cost performance of the former is about ten times better than the latter for our CLT.

Another issue of the acceleration speed for CPU-MPI and GPU-OpenACC must be discussed here is that, the theoretical computational capability for double precision floating points of TITAN Xp is up to 0.38 TFLOPS, which is only comparable with one Intel® Xeon® Gold 6148F CPU with 20 cores (without accurate data). But the execution speed on TITAN Xp is about 4 times of that on CPU with 32 cores. Besides, the double precision computational capability of TITAN V is about 6.90 TFLOPS, which is more than 10 times of TITAN Xp, but the execution speed on TITAN V is only accelerated less than twice compared with that on TITAN Xp. Therefore, the parallelization performance of the code on different platforms is not entirely depends on the theoretical computational capability. Other factors like the memory accessing time, partition, bandwidth, I/O situation, compiler optimization options, and the network conditions will all affect the code's execution speed. Furthermore, the deeper reasons for the difference of OpenACC and MPI acceleration are still under study.

Another important feature used in OpenACC is the combination of three levels parallelization as *'gang'*, *'work'*, and *'vector'*. For the test case shown in Figure 8, we found that the speed of CLT on the specified problem size is most sensitive to the number of *'gang'*: enough gang parallelization (more than 512) leads to the best performance on GPU, while too

few gangs (less than 100) results in slow down of the code compared with the MPI case with 32 cores. The default set of the compiler without adding any artificially specified configurations also leads to the best performance of the acceleration as shown in Figure 8. The numbers of *'gang'*, *'worker'*, and *'vector'* will affect the usage of GPU memory cache and can influence the speed greatly. For codes accelerated with OpenACC, adjusting these configurations are necessary to obtain the best acceleration performance, and the best set is also quite different for different codes, devices, and problem sizes.

4.2 Performance on double GPU cards combined with MPI

For the lack of cluster with multiple GPU nodes, the speed test for the OpenACC combined with MPI has only been performed on up to double GPU cards inside a workstation.

Firstly, a simple test for MPI-GPU combined acceleration is carried out on a single TITAN Xp with 4 MPI ranks for the two implementation methods mentioned in Figure 6 and 7. The first method (as shown in Figure 6, with direct communication between GPU) leads to only 15% slow down compared with time showed by the green bar in Figure 8, while the other method (as shown in Figure 7, use *'update'* clause and communication between CPU) leads to 105% slow down. Therefore, the direct communication of MPI support in GPU makes the multiple GPUs acceleration for large size problems become feasible, while the data exchange between GPU and CPU memories will slow down the execution speed intolerably.

Secondly, a corresponding test for MPI-GPU combined acceleration is performed on double TITAN Xp GPUs. The method of direct communication between GPUs as shown in Figure 6 is chosen. Total two MPI processes are carried out with double TITAN Xp GPUs, that is, each GPU executes one MPI process, and the computational domain is equally divided into two blocks in R direction. As the result shown in Figure 8, the final execution time is 621 seconds, which is even faster than that of single TITAN V. Compared with the performance of single TITAN Xp, the parallel efficiency is 96%. Therefore, the time cost on data interaction through Peripheral Component Interface Express (PCIe) is negligible and the parallelization with multiple GPUs is quite workable, which is necessary and indispensable for simulating larger size problem in future research.

4.3 Results of benchmarking

To demonstrate the validity of the results calculated by the OpenACC version on the GPU, the m/n=2/1 resistive tearing mode for Tokamak is carried out both on the CPU-MPI platform (Intel® Xeon® Gold 6148F CPU @ 2.40GHz, 32 cores) and the GPU-OpenACC platform (TITAN V or TITAN Xp, using one GPU for each test).

The initial equilibrium is calculated by the NOVA code [20] with the initial safety factor q and pressure p profile as shown in Figure 9. The m/n=2/1 mode is most unstable for this equilibrium. The grid set is (256, 256, 16) in R, Z, and ϕ. The resistivity η is chosen to be 1×10^{-5}.

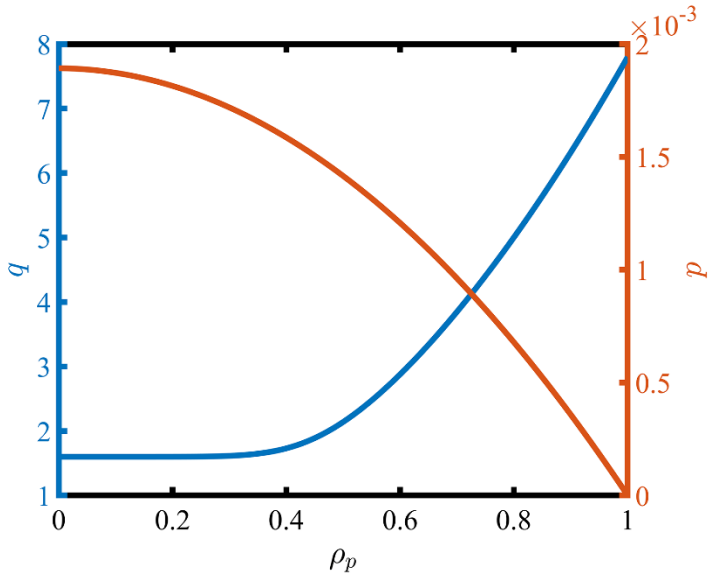

Fig. 9. Initial equilibrium safety factor q profile and pressure p profile for m/n=2/1 resistive tearing mode simulation.

Temporal evolutions for the kinetic energy of the system are given in Figure 10 for the CPU-MPI result and GPU-OpenACC result, respectively. The energy values of the two cases are exactly the same during the whole 400,000 simulation steps, except for some negligible differences on the 14th decimal place of the number, which is almost the machine error limitation of the double precision.

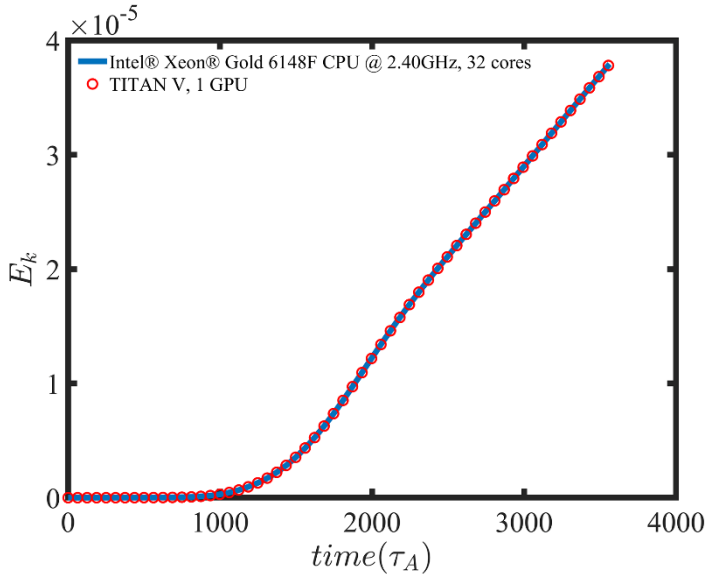

Fig. 10. Temporal evolutions for the kinetic energy of the system for two cases executed on CPU-MPI platform (blue line) and GPU-OpenACC platform (red circles), respectively.

Besides, the mode structures from these two calculations are identical, as the result of the CPU-MPI case demonstrated in Figure 11(a), and the contour plot for the difference of the mode structures by CPU and GPU is also given in Figure 11(b). The mode structure of the toroidal electric field E_ϕ in Figure 11(a) is clearly an m/n=2/1 mode, the result calculated by GPU-OpenACC is omitted due to the indistinguishability. The difference of the results between CPU-MPI and GPU-OpenACC as shown in Figure 11(b) is less than 10^{-14}, which is consistent with the situation of kinetic energy. As for the result in TITAN Xp, it is just the same as that in TITAN V, thus it will not be repeated here.

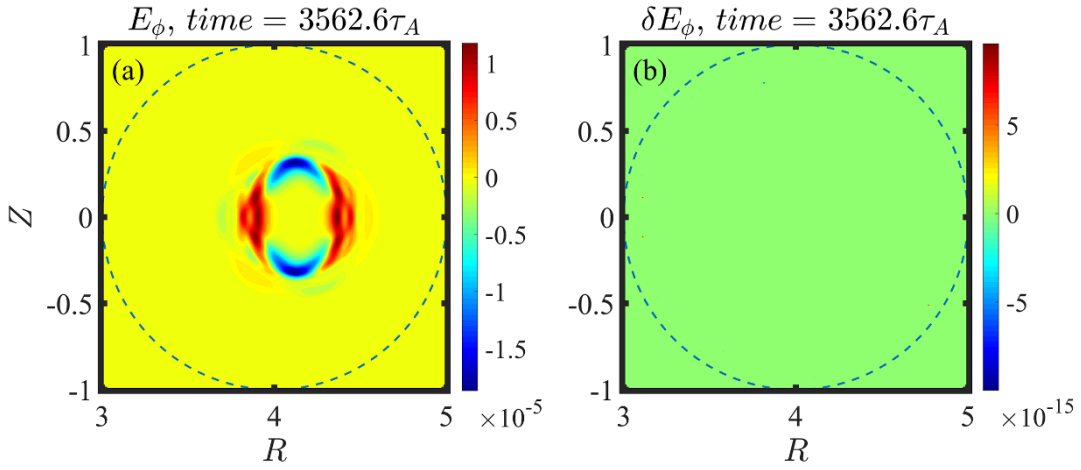

Fig. 11. (a) The m/n=2/1 mode structure of toroidal electric field E_ϕ. (b) The difference of

E_ϕ between the results calculated by CPU-MPI platform and GPU-OpenACC platform.

Therefore, the consistency for the m/n=2/1 tearing mode on both platforms confirms the reliability of GPU-OpenACC for the double precision simulation of CLT. The stability is also verified with dozens of simulations with each over 400,000 steps. The lack of ECC memory in TITAN Xp and TITAN V doesn't influence the correctness of the CLT results so far.

5. Discussion and Conclusion

The successful application of OpenACC[23-28] in CLT leads to great improvement in computational efficiency for studying the MHD instability in Tokamak. The transference of CLT from the CPU-MPI platform to GPU-OpenACC platform is relatively easy compared with completely rewritten the code into CUDA or OpenCL. Only about 500 lines OpenACC directives have been added into the code, which is quite a few compared with total more than 20,000 lines. Some minor changes have been done on the code, such as merging some small subroutines and adjusting the order of cyclic indices to obtain the best acceleration performance.

Compared with the speed of a same case executed on CPU with 64 cores MPI-parallelization, about four times of acceleration has been achieved on a single TITAN V and double TITAN Xp GPUs. The combination of MPI and OpenACC makes the multiple GPU acceleration become feasible by simply calling MPI APIs with the data addressed on GPU memory.

The speedup is found to be depending on more factors than just the theoretical computational capabilities of the CPU and GPU. Only about double speed of TITAN Xp is obtained on TITAN V, while the latter one's theoretical computational capability in double precision is ten times more than that of the former. Therefore, greater potential in TITAN V is observed if more optimizations on CLT, such as increase of data locality, are carried on. However, the speed of the code and the time spent on code modification and optimization is a trade-off. Because the speedup performance on TITAN V is already pretty good considering its low price, and to keep the integrity and readability of code structure, acceleration work on

CLT is paused at this stage.

Most important of all, the benchmarking for the results calculated by GPU are done by comparing that of traditional CPU platforms. The results from both platforms show exactly identical after 400,000 steps for dozens of runs. The double-precision operations on TITAN Xp and TITAN V can be fully trustable for our CLT.

By the way, the migration to OpenACC for kinetic part of hybrid kinetic-magnetohydrodynamic version code, CLT-K, is still under way, which requires more modifications on the code structure and adjustments on the combination of different OpenACC directives. The experience for this work on CLT-K will be introduced in a future paper if the acceleration performance is noteworthy.

Acknowledgments

The authors would like to acknowledge helpful suggestions given by professor C. Yang, and the Sunway TaihuLight Supercomputer Team at National Supercomputing Center, Wuxi, China.

This work is supported by the National Natural Science Foundation of China under Grant No. 11775188 and 41474123, the Special Project on High-performance Computing under the National Key R&D Program of China No. 2016YFB0200603, Fundamental Research Fund for Chinese Central Universities, Beijing Natural Science Foundation (8184094), IWHR Research & Development Support Program (JZ0145B022018).

References

[1] B. Wan, Recent experiments in the EAST and HT-7 superconducting tokamaks, Nucl. Fusion, 49 (2009) 104011.

[2] H. Renner, J. Boscary, V. Erckmann, H. Greuner, H. Grote, J. Sapper, E. Speth, F. Wesner, M. Wanner, The capabilities of steady state operation at the stellarator W7-X with emphasis on divertor design, Nucl. Fusion, 40 (2000) 1083.

[3] D.W. Walker, J.J. Dongarra, MPI: A standard message passing interface, Supercomputer, 12 (1996) 56-68.

[4] L. Dagum, R. Menon, OpenMP: an industry standard API for shared-memory programming, IEEE computational science and engineering, 5 (1998) 46-55.

[5] S. Cook, CUDA programming: a developer's guide to parallel computing with GPUs, Newnes2012.

[6] D.R. Kaeli, P. Mistry, D. Schaa, D.P. Zhang, Heterogeneous computing with OpenCL 2.0, Morgan Kaufmann2015.
[7] R. Farber, Parallel programming with OpenACC, Newnes2016.
[8] S. Markidis, J. Gong, M. Schliephake, E. Laure, A. Hart, D. Henty, K. Heisey, P. Fischer, OpenACC acceleration of the Nek5000 spectral element code, The International Journal of High Performance Computing Applications, 29 (2015) 311-319.
[9] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, M. Min, An MPI/OpenACC implementation of a high-order electromagnetics solver with GPUDirect communication, The International Journal of High Performance Computing Applications, 30 (2016) 320-334.
[10] Y. Wang, J. Lin, L. Cai, W. Tang, S. Ethier, B. Wang, S. See, S. Matsuoka, Porting and optimizing gtc-p on taihulight supercomputer with sunway openacc, HPC China, (2016).
[11] S. Gupta, P. Majumdar, Accelerating lattice QCD simulations with 2 flavors of staggered fermions on multiple GPUs using OpenACC—A first attempt, CoPhC, 228 (2018) 44-53.
[12] J. Kraus, M. Schlottke, A. Adinetz, D. Pleiter, Accelerating a C++ CFD code with OpenACC, Proceedings of the First Workshop on Accelerator Programming using Directives, IEEE Press, New Orleans, Louisiana, 2014, pp. 47-54.
[13] S. Wang, Z. Ma, Influence of toroidal rotation on resistive tearing modes in tokamaks, PhPl, 22 (2015) 122504.
[14] W. Zhang, Z. Ma, S. Wang, Hall effect on tearing mode instabilities in tokamak, PhPl, 24 (2017) 102510.
[15] S. Wang, Z. Ma, W. Zhang, Influence of driven current on resistive tearing mode in Tokamaks, PhPl, 23 (2016) 052503.
[16] W. Zhang, S. Wang, Z. Ma, Influence of helical external driven current on nonlinear resistive tearing mode evolution and saturation in tokamaks, PhPl, 24 (2017) 062510.
[17] J. Zhu, Z. Ma, S. Wang, Hybrid simulations of Alfvén modes driven by energetic particles, PhPl, 23 (2016) 122506.
[18] J. Zhu, Z. Ma, S. Wang, W. Zhang, Nonlinear dynamics of toroidal Alfvén eigenmodes in the presence of tearing modes, Nucl. Fusion, (2018).
[19] L. Duan, X. Wang, X. Zhong, A high-order cut-cell method for numerical simulation of hypersonic boundary-layer instability with surface roughness, J. Comput. Phys., 229 (2010) 7207-7237.
[20] C. Cheng, M. Chance, NOVA: A nonvariational code for solving the MHD stability of axisymmetric toroidal plasmas, J. Comput. Phys., 71 (1987) 124-146.
[21] L. Lao, H.S. John, R. Stambaugh, A. Kellman, W. Pfeiffer, Reconstruction of current profile parameters and plasma shapes in tokamaks, Nucl. Fusion, 25 (1985) 1611.
[22] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, Y. Wang, Intel math kernel library, High-Performance Computing on the Intel® Xeon Phi™, Springer2014, pp. 167-188.
[23] G. Kan, M. Zhang, K. Liang, H. Wang, Y. Jiang, J. Li, L. Ding, X. He, Y. Hong, D. Zuo, Improving water quantity simulation & forecasting to solve the energy-water-food nexus issue by using heterogeneous computing accelerated global optimization method, ApEn, (2016).
[24] G. Kan, X. He, J. Li, L. Ding, Y. Hong, H. Zhang, K. Liang, M. Zhang, Computer aided

numerical methods for hydrological model calibration: An overview and recent development, Archives of Computational Methods in Engineering, (2017) 1-25.
[25] G. Kan, T. Lei, K. Liang, J. Li, L. Ding, X. He, H. Yu, D. Zhang, D. Zuo, Z. Bao, A multi-core CPU and many-core GPU based fast parallel shuffled complex evolution global optimization approach, IEEE Transactions on Parallel and Distributed Systems, 28 (2017) 332-344.
[26] G. Kan, K. Liang, J. Li, L. Ding, X. He, Y. Hu, M. Amo-Boateng, Accelerating the SCE-UA global optimization method based on multi-core CPU and many-core GPU, Advances in Meteorology, 2016 (2016).
[27] G. Kan, X. He, L. Ding, J. Li, Y. Hong, D. Zuo, M. Ren, T. Lei, K. Liang, Fast hydrological model calibration based on the heterogeneous parallel computing accelerated shuffled complex evolution method, EnOp, 50 (2018) 106-119.
[28] G. Kan, X. He, L. Ding, J. Li, K. Liang, Y. Hong, A heterogeneous computing accelerated SCE-UA global optimization method using OpenMP, OpenCL, CUDA, and OpenACC, Water Sci. Technol., (2017) wst2017322.