Hindawi
Mathematical Problems in Engineering
Volume 2017, Article ID 1970628, 12 pages
https://doi.org/10.1155/2017/1970628

Research Article

Comparison of Three Different Parallel Computation Methods for a Two-Dimensional Dam-Break Model

Shanghong Zhang,[1] Wenda Li,[1] Zhu Jing,[1] Yujun Yi,[2] and Yong Zhao[3]

[1]*Renewable Energy School, North China Electric Power University, Beijing 102206, China*
[2]*School of Environment, Beijing Normal University, Beijing 100875, China*
[3]*China Institute of Water Resources and Hydropower Research, Beijing 100038, China*

Correspondence should be addressed to Shanghong Zhang; zhangsh928@126.com

Received 22 May 2017; Revised 31 July 2017; Accepted 22 August 2017; Published 28 September 2017

Academic Editor: Alistair Borthwick

Three parallel methods (OpenMP, MPI, and OpenACC) are evaluated for the computation of a two-dimensional dam-break model using the explicit finite volume method. A dam-break event in the Pangtoupao flood storage area in China is selected as a case study to demonstrate the key technologies for implementing parallel computation. The subsequent acceleration of the methods is also evaluated. The simulation results show that the OpenMP and MPI parallel methods achieve a speedup factor of 9.8× and 5.1×, respectively, on a 32-core computer, whereas the OpenACC parallel method achieves a speedup factor of 20.7× on NVIDIA Tesla K20c graphics card. The results show that if the memory required by the dam-break simulation does not exceed the memory capacity of a single computer, the OpenMP parallel method is a good choice. Moreover, if GPU acceleration is used, the acceleration of the OpenACC parallel method is the best. Finally, the MPI parallel method is suitable for a model that requires little data exchange and large-scale calculation. This study compares the efficiency and methodology of accelerating algorithms for a dam-break model and can also be used as a reference for selecting the best acceleration method for a similar hydrodynamic model.

1. Introduction

Floods are an example of the main types of disaster in the world; they occur frequently and have extensive influence. As the main engineering measure for flood prevention, dams play an important role in flood control. However, along with its protective benefits, a dam itself brings the potential risk of collapse. The damage caused by a dam-break flood is far more than that caused by a general flood; it is enough to take thousands of lives [1]. Therefore, to reduce the risk of a dam project and draw up a disaster emergency treatment plan, it is necessary to establish a precise dam-break model to simulate the flood evolution.

Many methods have been used to simulate dam-break flood behavior, and they are broadly classified into simplified methods and hydrodynamic methods. Simplified methods [2–5] have the advantage of fast simulation. However, the results of some processes are not calculated and the predictions are less accurate because these models either have low-resolution grids or use simplified shallow water equations. To obtain more accurate results, two-dimensional hydrodynamic models are usually applied to simulate dam-break flooding [6].

The simulation accuracy depends mainly on the resolution of the terrain and the methods of solution. In recent years, with the development of telemetry technology, the terrain accuracy has reached the 1 m resolution and is no longer a constraint on the accuracy of model calculations. However, at the same time, the cost of high-resolution 2D dam-break model computation has risen sharply. Taking a 1 m resolution terrain as an example, according to the Courant–Friedrichs–Lewy (CFL) condition, it is necessary that the Courant number should be less than 1.0 so that time step is small enough for stability and convergence of the model. For a time step of 0.1 s, there are about 10^6 meshes per square kilometer. With roughly 10^{12} operations per km^2 per day, it is not acceptable to work with general study areas [7]. To solve the challenges of high-precision models, the use of

parallel methods is very important and extremely necessary [7–11].

The parallel computation algorithms of mathematical models fall into three categories: message-passing interface (MPI) models, shared memory models (OpenMP), and GPU general computing models. In a shared memory model, communication between threads is achieved by reading and writing directly from shared memory, and it is easy to solve the traditional programming problem of load balancing. Models developed under this standard have the advantages of simplicity, good extendibility, and portability [12]. The LISFLOOD-FP explicit storage cell model was parallelized using OpenMP [13], and Neal et al. [14] tested the OpenMP version of the LISFLOOD-FP for a range of test cases with grid sizes that varied from 3 km to 3 m. The latter study showed that such test cases had a significant effect on acceleration. Zhang et al. [15] applied OpenMP to the parallel computation of the dam-break model and gained a speedup factor of 8.64× on a 16-core computer. In the MPI parallel applications, TRENT [16, 17], CalTWiMS [18], RMA [19], and Oger et al. [20] used MPI and regional decomposition in the parallel simulation cases. Pau and Sanders [18] achieved a speedup factor of 1.5–2× on four cores in an implicit total-variation-diminishing finite volume scheme. Villanueva and Wright [17] obtained speedup factors up to 6×, 12×, and 30× on 6, 18, and 32 processors, respectively, in a coupled 1D-2D full shallow water model. Among all the parallel algorithms, GPU models obtain the best acceleration [21–23]. The multistreaming processors and powerful computational capabilities of GPUs are vitally important for efficient parallel computation with large amounts of data and high-precision requirements [24–28]. Bradbrook et al. [29] rewrote the two-dimensional diffusive wave model JFLOW using general-purpose GPUs and achieved substantially faster computation on single-accelerator processors. However, it has the limitations of single precision and having to be completely recoded in a graphics-oriented language. Lamb et al. [30] reported a speedup factor of 112× for a GPU code run on a NVIDIA GeForce 8800GTX over the serial JFLOW code. Zhang et al. [31, 32] applied a GPU-based parallel method, OpenACC, to the parallel calculation of the flow field and the dam-break model. Rueda et al. [33] performed parallel experiments on OpenACC and CUDA based on an algorithm for simulating flood storage area and on the DEM dataset of Africa. The single core CPU simulation took 234 s, whereas OpenACC and CUDA took only 45 s and 11 s, respectively, to complete the model operation. Although there have been many studies on parallel computation, most of them focused on special methods for implementing parallel algorithms and on the associated acceleration effects. There are relatively few examples of comprehensive comparisons among multiple parallel methods. Moreover, there is no clear parallel optimization method for the dam-break model because of the limited cases of different parallel methods being used for different models and applications.

To compare the advantages and disadvantages of various parallel methods for dam-break simulations, the most common way is to use different parallel algorithms for the same dam-break model and simulation cases. Thus, the novel task undertaken here is to build a two-dimensional hydrodynamic dam-break model and compare the three main parallel methods, OpenMP, MPI, and OpenACC. The serial version of the dam-break model is used as a benchmark for the three parallel versions solving the same problem, which means that there are four models to be compared: (1) the serial version of the model; (2) a version of the model that uses the OpenMP application programming interface and shared memory; (3) a version of the model that uses MPI and distributed memory; (4) a version of the model that uses OpenACC and cooperates with the CPU. To compare their performances, the four models are run in different context configurations and the run time for each model version is recorded separately. In addition to the acceleration effect, the environmental requirements, the difficulty of rewriting, and the trend in technological development are taken into consideration. Finally, recommendations are made for selecting a parallel algorithm for a dam-break model.

2. Method

2.1. Dam-Break Flow Model

2.1.1. Control Equation. The three-dimensional Navier–Stokes equation can reasonably describe the mechanism of dam-break flow, but it is too complex and expensive to be used for a large range of flood simulations. Because changes in the dam-break flow characteristics in the depth direction are less pronounced than those in the horizontal direction, they can be regarded as planar and the Navier–Stokes equation can be simplified by averaging in the depth direction. The conservative form of the planar 2D shallow water equations is as follows:

$$\frac{\partial U}{\partial t} + \nabla \cdot G = \frac{\partial U}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = S.$$

$$U = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix},$$

$$G = (E, F),$$

$$E = \begin{bmatrix} hu \\ hu^2 + \dfrac{gh^2}{2} \\ huv \end{bmatrix},$$

$$F = \begin{bmatrix} hv \\ huv \\ hv^2 + \dfrac{gh^2}{2} \end{bmatrix},$$

$$S = \begin{bmatrix} 0 \\ gh\left(S_{ox} + S_{fx}\right) \\ gh\left(S_{oy} + S_{fy}\right) \end{bmatrix},$$

$$S_{ox} = -\frac{\partial Z_b}{\partial x},$$
$$S_{oy} = -\frac{\partial Z_b}{\partial y},$$
$$S_{fx} = -\frac{n^2 u\sqrt{u^2+v^2}}{h^{4/3}},$$
$$S_{fy} = -\frac{n^2 v\sqrt{u^2+v^2}}{h^{4/3}} \tag{1}$$

Here, U is the conservation vector, G is the flux vector, E is the x-direction flux, F is the y-direction flux, and S is the source term vector. Terms u and v are the water velocities in the x and y directions, respectively; g is the gravitational acceleration, S_{ox} and S_{oy} describe the bottom slope and S_{fx} and S_{fy} give the friction gradients in the x and y directions, respectively. Finally, Z_b is the river bottom elevation, and n is the Manning coefficient.

2.1.2. Model Solving Method. A quadrilateral unstructured grid is used in the dam-break flow model. A cell-centred format control volume in which all physical variables are defined in the centroid of the grid cell is used in model solving, for which a single grid cell gives the control volume.

Planar 2D shallow water equation (1) is integrated over the control volume A:

$$\int_{A_j} \frac{\partial U}{\partial t} dA + \int_{A_j} \nabla \cdot G\, dA = \int_{A_j} S\, dA. \tag{2}$$

Gauss's formula is used to convert (2) to

$$\frac{\partial U_i}{\Delta t}\Delta S_i = -\oint_{L_i} G \cdot n\, dl + \int_{A_j} S\, dA, \tag{3}$$

where U_i is the average value of the control unit, L_i is the boundary of control volume k, n is a unit normal vector on the boundary, and ΔS_i is the area of each calculation unit. The line integral of (3) is then discretized as

$$\Delta U = -\frac{\Delta t}{\Delta S_i}\sum_{j=1}^{4}\left(G_{ij}\cdot n_{ij}\right)\Delta l_{ij} + \frac{\Delta t}{\Delta S_i}\int_{A_j} S\, dA, \tag{4}$$

where Δl_{ij} is the length of each side, G_{ij} is the numerical flux via edge I, and n_{ij} is the unit outer normal vector passing through the unit boundary j.

For the Riemann problem at the interface, the Roe scheme is used to solve the normal flux of the computation unit. The detailed process of solving a Roe scheme has been described by Toro [34]. The expression is as follows:

$$G \cdot n = \frac{1}{2}\left[(E,F)_R \cdot n + (E,F)_L \cdot n - \left|\widetilde{J}\right|(U_R - U_L)\right], \tag{5}$$

where $\widetilde{J} = \partial(G\cdot n)/\partial U$ is a Roe-formatted Jacobian matrix and U_R and U_L are conserved variables on both sides of the cell.

The source term solution plays a role in stabilizing the computational format and improving the computational accuracy in the model. Using the bottom slope characteristic decomposition method to obtain a reasonable solution for the bottom slope source term is realistic for complex terrain conditions. The upwind interface fluxes for upwind treatment of the bottom slope source term are

$$\overline{S}_0 = \sum_{i=1}^{4}\sum_{j=1}^{3}\left[\frac{1}{2}\left(1 - \operatorname{sign}\left(\overline{\lambda}^j\right)\right)\beta^j \widetilde{e}^j l_i\right]^i,$$
$$\beta^1 = -\frac{1}{2}\overline{c}\,\overline{z}_b,$$
$$\beta^2 = 0,$$
$$\beta^3 = \frac{1}{2}\overline{c}\,\overline{z}_b, \tag{6}$$

where $\widetilde{e}^j$ $(j = 1, 2, 3)$ is the eigenvector corresponding to the three eigenvalues of the Jacobian matrix, sign() is the sign function, $\overline{z}_b$ is the average that calculates the left and right bottom elevations, and $\overline{c}$ is the average velocity given by the Roe scheme.

With a completely explicit discretization to deal with the friction source term in order to further discretize (4), the following expression for ΔU is obtained:

$$\Delta U = \frac{\Delta t}{\Delta S_j}\left[\left(-\sum_{i=1}^{4} G_i n_i l_i\right) + \sum_{i=1}^{4}\sum_{j=1}^{3}\left(\frac{1}{2}\left(1 - \operatorname{sign}\left(\overline{\lambda}^j\right)\right)\beta^j \widetilde{e}^j l_i\right)^i + \Delta t S_f^n\right]. \tag{7}$$

The flow chart for solving the serial model is shown in Figure 1.

2.2. Model Parallelism Analysis. From the above method for solving the flow, we can see that the variable ΔU at the next time step requires only the flow characteristics of the previous time step. As there is no association with the grid, this provides the basic conditions for parallelism. Hence, at the time T of entering the grid cycle, all mesh cycles can be cut to form a few smaller grid cycle blocks. The tasks including the interface flux calculation, the bottom slope and friction source term calculation, and the simulation data update, which belongs to different grid blocks, will be assigned to all the computing resources. Hence, the grid block can calculate at the same time and the overall simulation process can be accelerated.

The FORTRAN language was used to implement the dam-break model calculation, and the OpenMP, MPI, and OpenACC parallel methods were used to realize parallel computing. Of these methods, OpenMP is an implicit parallel method; it is a shared storage and CPU thread granularity parallel method. The MPI method is an explicit parallel method; it is a distributed storage and CPU process granularity parallel method. The OpenACC method is implicitly parallel; it is a single address storage and GPU parallel method.

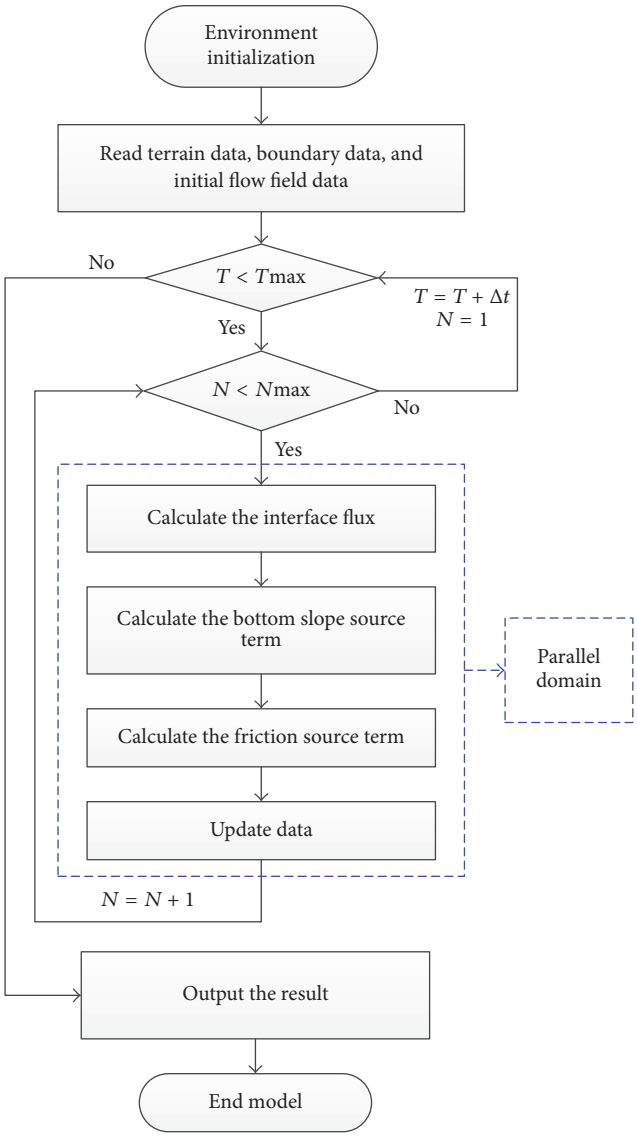

Figure 1: Flow diagram for solving the serial dam-break model.

2.3. Dam-Break Model Based on OpenMP. The general realization method of the OpenMP interface is that the calculation task should be allocated to all virtual threads of the CPU according to the OpenMP directives given to the compiler, wherein each thread completes part of the grids' calculation task. There is no difference with the serial model outside the OpenMP parallel part. In the OpenMP parallel part, OpenMP directives are used to convey information. Shared memory is applied for the shared parameters of the threads and independent storage space is applied for the private variables of the independent thread. The information exchange speed between threads is the memory-level exchange speed.

Rewriting the dam-break model with OpenMP includes the following four steps.

(1) Determine the Scope of the Parallel Domain. All of the parallel operations need to be carried out in their respective parallel domains. The overall range of parallelism is the grid loop part according to the dam-break model flow diagram and the model parallelism analysis. Thus, the whole model can be set up with a whole parallel domain. The specific implementation is as follows: use the $ OMP PARALLEL directive above the grid loop of the 2D hydrodynamic model to turn on the parallel domain, and add the $ OMP END PARALLEL directive at the end of the grid loop to end the parallel field.

(2) Set the Type of Parallel Data Access within the Domain and the Number of Threads. This is the most critical part of the parallel process; identifying the data type of each variable determines the shared boundary within the parallel domain, and it can save compiler stack space and increase the efficiency of acceleration. The realization method is as follows: add the directive word SHARED() and add the shared variables in it; then call the DEFAULT (PRIVATE) directive to set the remaining parallel domain variables to the private-variable properties by default. Using the NUM_THREADS directive to define a reasonable number of threads for the model, the maximum number of threads is the number of host virtual threads.

(3) Model Task Division. The grid cycle of the dam-break model is the main part for parallel computation. The $OMP DO directive is used before the grid loop to dispatch all grid computing tasks to different threads in order to maximize the acceleration effect of the model.

(4) Parallel Protection Measures and Optimization. When the private variables in the parallel domain are turned on, they apply storage memory according to the number of threads; the applied memory will increase along with the increasing amount of threads opened. Because the applied memory for parallel computing will be used when the model runs the PARALLEL directive, if the compiler stack space is insufficient, the compiler can compile the program successfully, but the running program will exit because of a shortage of memory. To solve this problem, one method is to increase the compiler stack storage; another is to redetermine the shared storage variables in the parallel domain so as to decrease the memory needed. The DYNAMIC directive should be added after the $OMP DO directive to ensure that the task allocation is sufficiently reasonable, so that more calculation tasks can be assigned to faster threads.

The solving flow chart of the OpenMP parallel dam-break model is shown in Figure 2. The program directive statements are used to provide parallel purposes to the compiler, and the specific implementation of the parallel computation is completed by the compiler itself. If the directives added in the program are ignored, the program will automatically degenerate to the serial version, so this type of parallel method has good reversibility.

2.4. Dam-Break Model Based on MPI. The thinking behind the MPI interface takes the following steps. Firstly, the calculation tasks are divided manually according to their parallel computation characteristics, and then each divided

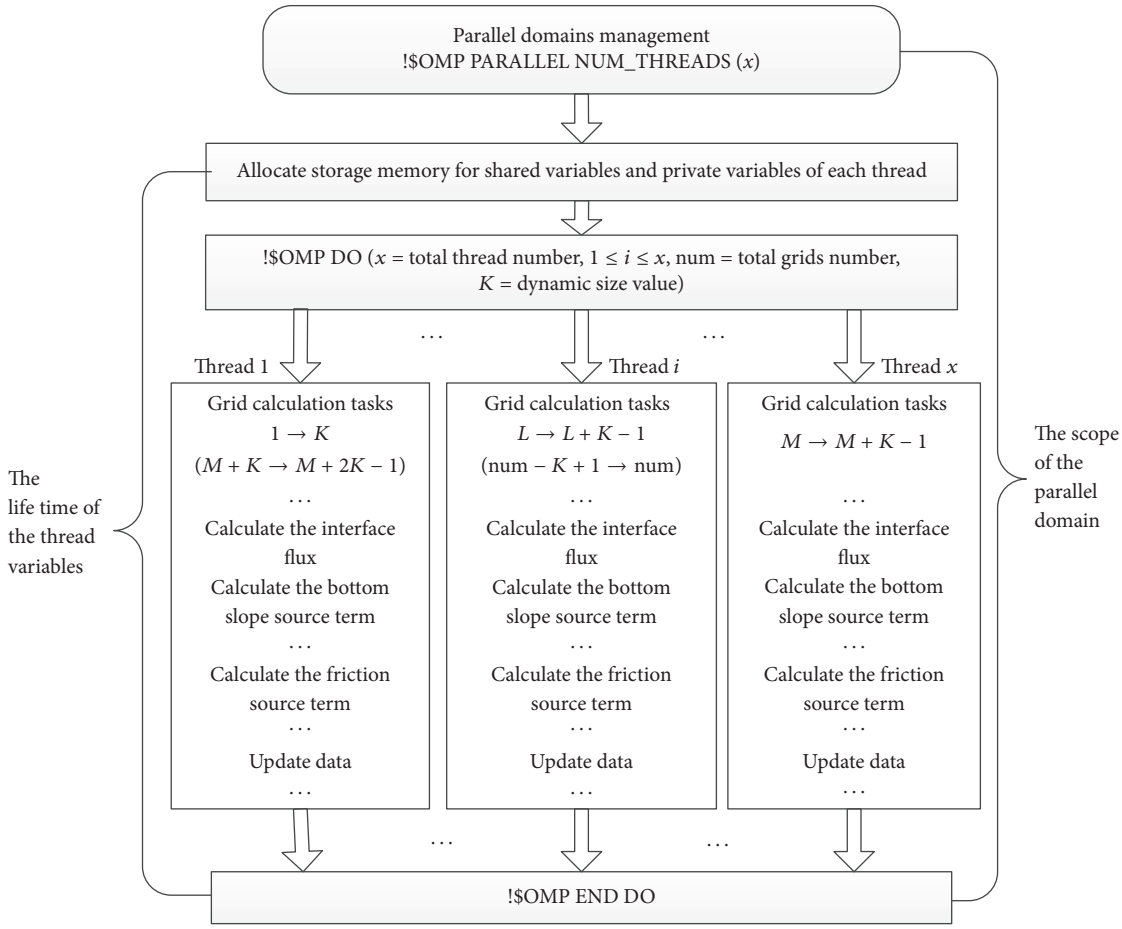

Figure 2: Flow diagram for solving the OpenMP dam-break model.

part is sent to a different process to run. The data exchanges between the processes are achieved by calling the MPI library. All the processes run synchronously and apply their own space, as in the serial program. The data exchange is transmitted internally on the same computer, and the transmission efficiency depends on the means of transmission between different processes under different computers. Unlike the thread-opening method in the OpenMP parallel model, the number of opening processes, including the number of hosts and the opening process number of each host, should be controlled by the mpich2 software in MPI. MPI itself does not provide a parallel guidance statement; its main roles are message communication, data management, and input and output. Thus, it is impossible to add MPI directives directly in the 2D hydrodynamic model to achieve parallelism; the model should be reconstructed based on the serial code. The key points of the reconstruction are the task division and the control of message communication.

Rewriting the dam-break model based on MPI can be divided into the following steps.

(1) MPI Initialization and Environment Identification. Firstly, the MPI environment should be opened by calling the MPI_INIT (ierr) function after the data input part of the model. The current process number and the number of total opening processes should then be obtained by calling the MPI_COMM_RANK (MPI_COMM_WORLD myid ierr) and MPI_COMM_SIZE (MPI_COMM_WORLD proc ierr) functions; the values will be stored in the *myid* and *proc* variables.

(2) Calculate the Offset Array and Data Receiving Array. Data and calculation task division of the model require two key arrays: the offset array and the data receiving array. Setting up these arrays is a key step in the MPI rewriting. The two arrays are used throughout the MPI parallel model, named *displs* and *recvcounts,* respectively, and apply memory space according to the total number of processes. The average number of meshes per process (*num*) can be calculated by dividing the total number of grids (*nelem*) by the total number of processes (*proc*). Hence, the last item of the *recvcounts* array is the sum of *num* and the remainder, and the remaining items are *num*. The first item of *displs* is 0, and the following item value can by calculated by accumulating the *num* value. The main purpose of task division is to consider multiple grid computing units as a whole; the calculation task is distributed manually to each process according to the offset array and data receiving array.

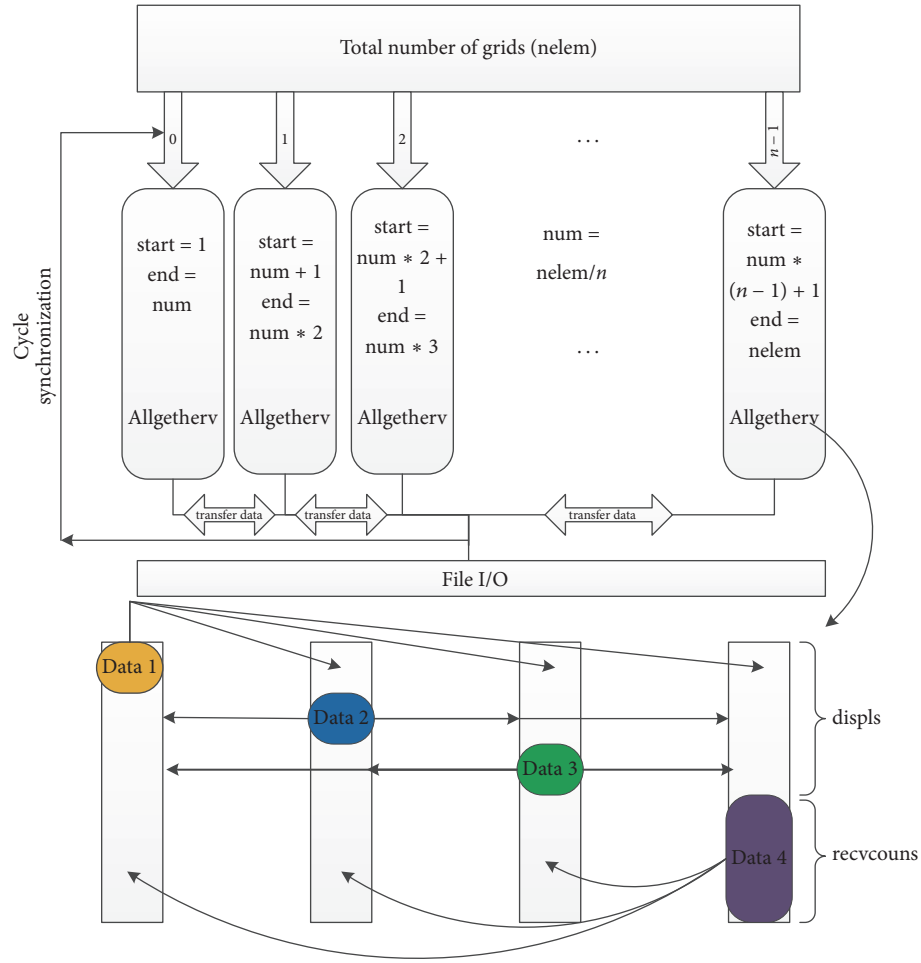

Figure 3: Flow chart for solving the dam-break model based on MPI.

(3) Data Communications and Synchronization. When the grid calculations of all processes are completed, the results from different processes need to be exchanged with each other to enter the next time step of the calculation. The method of broadcast communication is used to exchange data because of the large amount of data exchange. This performs an all-gather collection operation of variable length and uses the in place mechanism of the intergroup communication communicator. To realize data exchange, the start and end points of the calculation process can firstly be used to determine the sizes of the data receiving array and storage location offset array. The MPI_ALLGATHERV function can then be used to realize data transmission. The send port uses MPI_IN_PLACE and the value is zero, and the receiver port should set the array name that needs to be received and then put the offset array and data receiving array into the function to complete the operation. Thus, the model result in the process has been completed in exchange between different processes when the model runs into the next round of cycle time.

At this point, the manual task allocation and data communication of the dam-break model are complete. However, the average distribution of tasks cannot guarantee that the end time was the same. Therefore, the MPI_Barrier function is called after each grid cycle; the communicator identifies the processes that run the program at the same time and that maintain interprocess synchronization. The solving flow chart of the dam-break model based on MPI is shown in Figure 3.

2.5. Dam-Break Model Based on OpenACC. The OpenACC interface is similar to OpenMP in that all operations are done through directive statements. Firstly, the memory is applied in the graphics memory, and then data are transmitted to the GPU and the model calculations are completed on the accelerator. Finally, the results are sent back to the host. The task division on the accelerator has different accelerating effects according to the directive. In this paper, the KERNELS directive is used to realize parallel computing, and this directive can make each accelerator core obtain a basic calculation unit. The data transmission between the GPU and the CPU depends on the data bandwidth between the GPU and the host's external interface. Generally, because of the impact of data packets, the real transmission efficiency is slightly lower than the bandwidth.

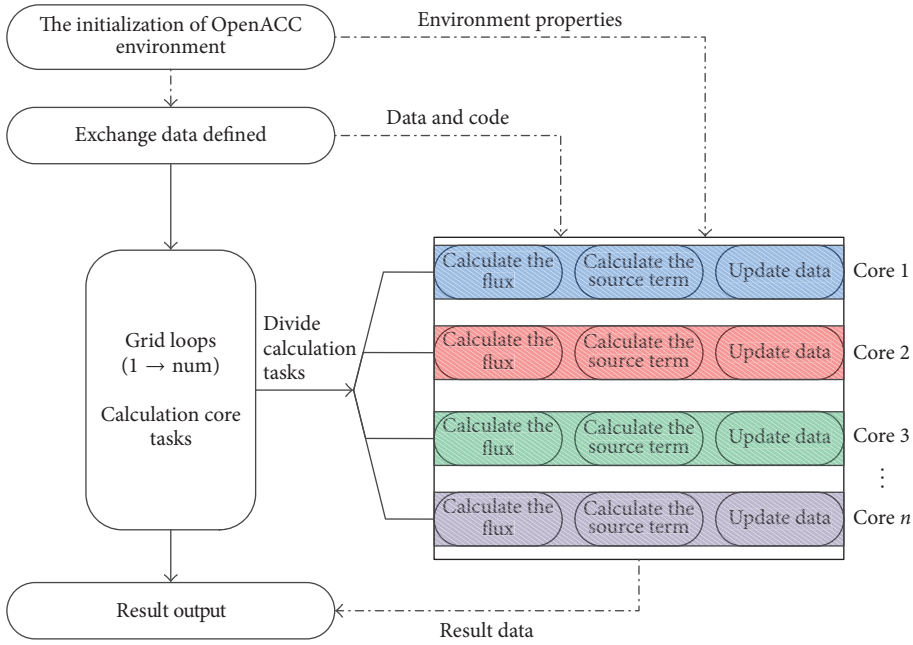

Figure 4: Flow chart for solving the dam-break model based on OpenACC.

There are three steps to implement the dam-break model based on OpenACC.

(1) Subroutine Processing. A number of functions of the dam-break model are realized through the FORTRAN subroutine, such as the functions of interpolation and the bottom slope source calculation. Unlike OpenMP, these subroutines cannot be loaded directly into the GPU. Modularity is used in OpenACC to realize the usage of subroutines. On the one hand, this method maintains the readability of the model, and on the other hand, it reduces the workload of the parallel model modification. The concrete realization is as follows: all of the functions and subroutines are written in front of the main program, the directives MODULE and END MODULE are used to modularize them, and finally the ROUTINE directive is used to transmit information to the GPU. The SEQ directive is used to ensure that the functions and subroutines are calling in single-thread mode.

(2) Environment Setting and Data Copying. Before the dam-break model runs to the time loop and uses the parallel computing, the *acc_init* function should be called firstly to turn on the OpenACC environment, and then the *acc_set_device_num* function should be called to determine which graphics card is to be used in the following calculation. In order to reduce the data exchange between the GPU and CPU, the directives DATA, COPYIN, and COPY should be used to define the exchanged variables and transmission method. The simulated value cannot be used in the CPU if the variable is transmitted to the GPU by the COPYIN directive. In addition, if some temporary variables need not be transmitted between the CPU and GPU, the CREATE directive will be used to define them. The directives DATA and END DATA define all the variables used to exchange between the CPU and GPU.

(3) Task Division. The directive KERNELS LOOP is used to set the parallel instructions at each loop. In order to further optimize the data management of the hydrodynamic model, the directive PRIVATE is used to privatize the variables that are calculated repeatedly, and the INDEPENDENT directive is used to extract the data dependency of KERNELS LOOP.

There are many differences between the OpenACC and OpenMP models; the choice of KERNELS is also important in addition to the above differences in steps. OpenACC also has a PARALLEL directive; multiple GANGs can be created and multiple WORKS can be distributed in the parallel domain to maintain the accelerated state. The solving flow chart of the dam-break model based on OpenACC is shown in Figure 4.

3. Case Study

3.1. Study Area. The Pangtoupao flood storage area is located in the Songhua River basin of China (Figure 5); it undertakes flood control tasks for Harbin City and has a direct impact on the safety of the lives and property of the city's residents. The elevation of the Pangtoupao flood storage area drops gradually from northwest to southeast, and its average east–west width and north–south length are 46 km and 58 km, respectively. It covers an area of 1,994 km^2 and its volume is about 5.5 billion cubic meters. According to flood data measured in 1998, flooding caused multiple levee breaches on the Songhua Rivers, with the Pangtoupao dam being breached on August 15. Outburst floods spread rapidly to the northeast and caused a huge flooded area.

3.2. Dam-Break Event Simulation. A DEM (resolution of 10 m) of Pangtoupao and the flood data for the Songhua River in 1998 were used to build the hydrodynamic model. In total, 337,084 quadrilateral unstructured meshes were partitioned

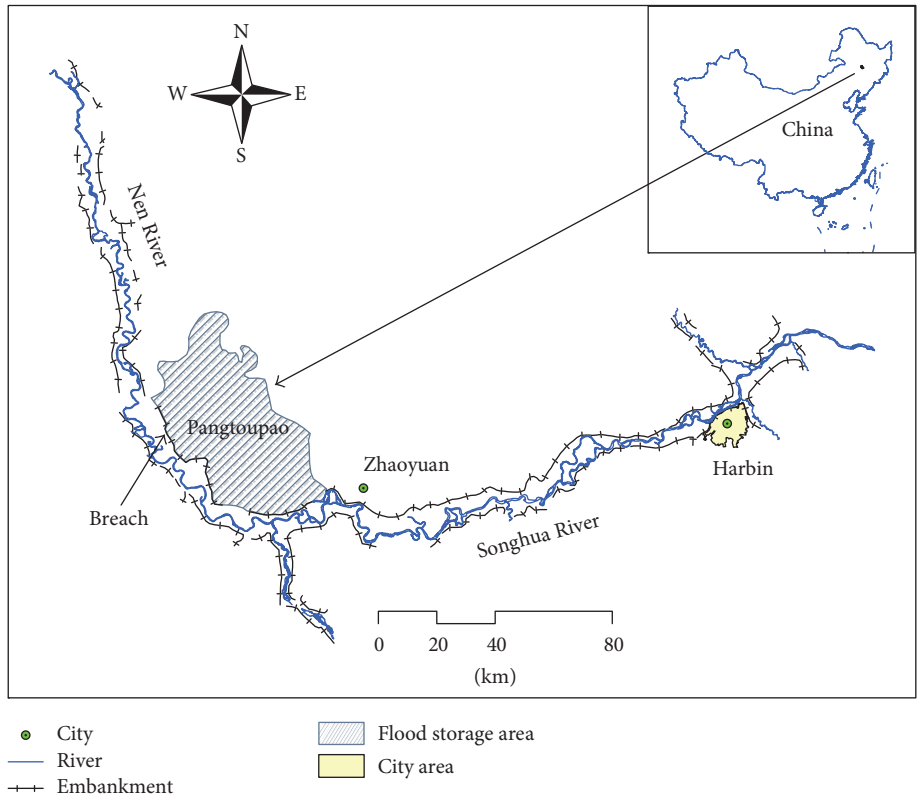

Figure 5: Location of the Pangtoupao flood storage area.

by the gambit software; the average grid length is 80 m. The Manning roughness value of the study area is 0.055. The measured flood occurred on August 15, 1998. The breach expanded rapidly to 530 m, and therefore this breach width is used for the dam-break simulation. The time from the levee breaches to reaching the maximum inundated area of the flood was 320 h, so the simulation time is chosen as 320 h. To examine the accuracy of the model, the maximum inundated area is calculated and compared with the measured value.

3.3. Parallel Efficiency Comparison Scheme. The simulation of the 320 h flood process takes approximately 170 h, which would be too long to compare the efficiencies of different parallel methods if the whole flood process was to be simulated. Instead, here 6,000 time steps (2,988 s) are used to compare the efficiencies of the different parallel computing methods.

The model operating environment of OpenMP and MPI is as follows: a Windows 7 64-bit operating system with Intel Parallel Studio XE 2016 compiler, an Intel Xeon E5-2640v3 CPU with a basic frequency of 2.6 GHz, 16 GB of memory, and a dual processor with 32 virtual cores. The model operating environment of OpenACC is as follows: a Windows 7 64-bit operating system with the PGI PVF version 14.9 compiler, an Intel Xeon E5-2603 CPU with a basic frequency of 1.8 GHz, 4 GB of memory, and an NVIDIA Tesla k20c graphics card. The CPU gap did not affect the consistency of the results because OpenACC mainly uses the GPU to realize acceleration.

Even-numbered threads from 2 to 24 were selected to test the calculation time of the OpenMP parallel model. Even-numbered processes from 2 to 24 were selected to test the calculation time of the MPI parallel model. For the OpenACC parallel model, the calculation time was tested on the computer with the NVIDIA Tesla k20c graphics card.

4. Results and Discussion

4.1. Calculation Results. The results of the model show that the maximum inundation area is 1,180 km^2, which is close to the measured value of 1,160 km^2. Figure 6 shows the flow field and inundation area of the dam-break flood. The parallel results are consistent with the serial results, and the calculation times used to simulate 2,988 s of the dam-break flood are given in Table 1. For this, the speedup ratio S is calculated as follows:

$$S = \frac{T_S}{T_P}, \tag{8}$$

where T_S is the serial computation time and T_P is the parallel computation time. The time-saving ratio δT is calculated as follows:

$$\delta T = \frac{(T_S - T_P)}{T_S}. \tag{9}$$

The speedup ratios of OpenMP and MPI in different threads/processes are shown in Figure 7. From Table 1 and Figure 7, we can see that the OpenACC parallel method

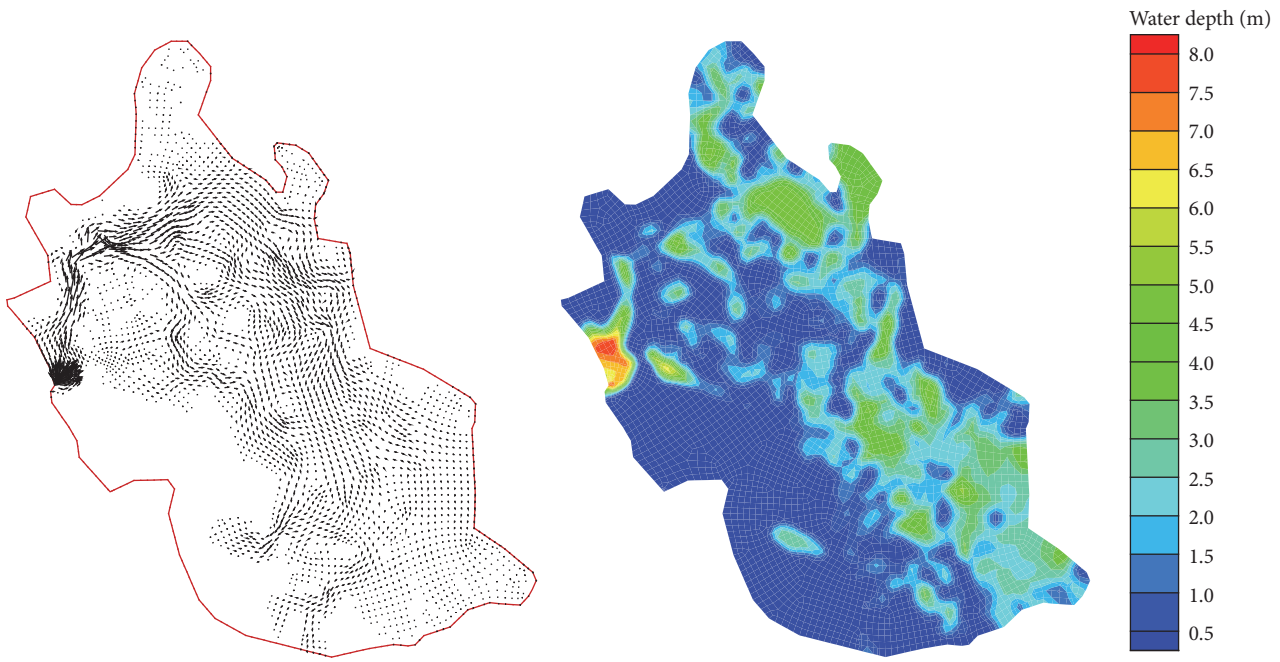

Figure 6: Flow field and inundation area of the dam-break flow in the Pangtoupao flood storage area.

Table 1: Calculation time and acceleration effect of different models (total simulation time: 2,988 s).

	Shortest time (s)	Speedup ratio	Time-saving ratio (%)
Serial model	1,639	1	0
OpenACC model	79	20.7	95
OpenMP model	168	9.8	90
MPI model	324	5.1	80

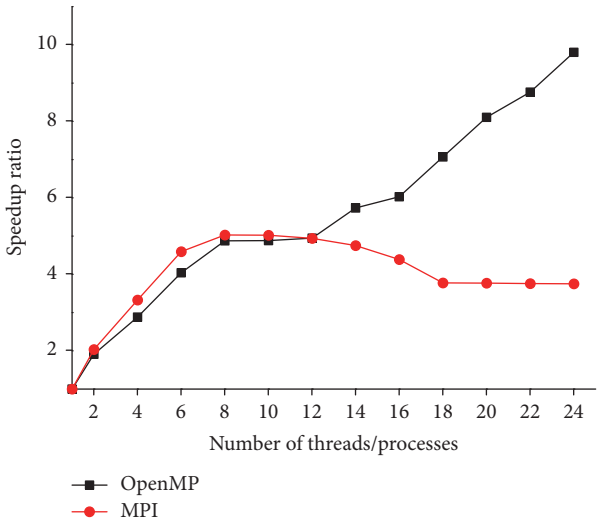

Figure 7: Speedup ratios of OpenMP and MPI parallel methods for different numbers of threads or processes.

achieved the best acceleration effect. The OpenMP parallel model achieved the biggest speedup ratio of 9.8 and was better than the MPI parallel model when the number of threads was greater than 12. The MPI parallel method was slightly better than the OpenMP acceleration effect for fewer than 12 threads (processes) and it achieved the biggest speedup ratio of 5.1.

As two of the more mature parallel approaches, the performance comparison between OpenMP and MPI is well known, although fine-grained OpenMP is not necessarily more efficient than MPI. For the dam-break model, the number of atomic operations is very small, with no extra mutex overhead, so OpenMP gets the upper hand. In the case of low computational resource consumption (fewer than 12 processes and threads), the task allocation overhead of the OpenMP directive is significantly higher than the message delivery cost in MPI, so the MPI acceleration efficiency is higher at the beginning. Along with more allocation of computing resources, MPI will be slightly weaker than OpenMP because of load balancing.

Because only the total amount of the loop (not the amount of calculation) can be divided in MPI, we cannot determine which process is more efficient in order to obtain more computation; the main program has to wait for the slowest process to complete the calculation task before the next round of time cycle. At the same time, the MPI data communication costs are also increasing with the number of processes. This is because each process has to broadcast to all the other processes in order to exchange results when the calculation task has been completed in each time step. In addition, the means of communication are related directly to the communication consumption. In this paper, a single server was used to test the MPI model. The internal transmission speed between different processes is equal to the

memory exchange speed, which is close to twice the speed of universal optical fibre transmission. Even so, the benefits of an increment in computing resources is still counteracted by the data exchange and other expenses. A linear increment in the amount of data exchange locks the speedup ratio of MPI to 5.1, which then declines smoothly.

The OpenACC parallel model realizes its acceleration based on the graphics card, whose computational capability is much higher than that of the CPU. Although it is not as flexible as the CPU parallel mode, which can adjust the amount of calculation resources, the overall acceleration capability of the GPU is considerable. In addition, the development of GPU technology has been very fast in recent years, including the graphics card and the GPU parallel language. Hence, the GPU acceleration mode has great potential in parallel computing.

4.2. Implementation Method Discussion. In order of implementation difficulty from high to low, the three parallel methods are listed as OpenACC, MPI, and OpenMP. Despite the fact that OpenMP requires the least rewrite work, developer has to understand the calculation flow and variables of the program clearly in order to find the correct variables to set as shared variables. The program can still run and achieve correct results with different shared- and private-variable conditions, but the acceleration efficiencies are different. For example, setting the constant variables to private variables, the acceleration effect still exists, but the cost of the error significantly reduces the speed of the original OpenMP model. Overall, OpenMP parallelization has the lowest degree of rewrite difficulty, good reversibility, and high fault tolerance. It is easy to debug and its acceleration efficiency is middling.

The rewrite workload of MPI is large because the serial code should be rewritten totally, and it is difficult to reverse. However, the rewrite difficulty level is low because the data division and transmission of the MPI model are clear. The MPI model can be debugged and tested easily in a single computer, whereas it is more difficult to debug on a multimachine cluster. It is suitable for conditions with less data exchange and sufficiently large amounts of calculation. The OpenACC parallel model has hard rewrite difficulty and is difficult to debug, but its reversibility and fault tolerance are good, and its acceleration effect is the best.

From the point of view of hardware requirements, the OpenACC model needs a high-performance graphics card. With the development of graphics cards, their price is likely to decrease continually; the price of a common graphics card used in parallel computing is now the same as a personal computer. The MPI model generally needs to be run on a computer cluster. The requirement of a single calculation node is low; a common personal computer could be used as a calculation node, although the CPU frequency and memory size will influence the acceleration efficiency. Of course, a huge amount of calculation requires a sufficient number of calculation nodes, which would increase the hardware costs. In addition, because the data transmission method is one of the most important factors influencing the acceleration efficiency, the general recommendation is to use fibre-optic cable or high-efficiency wireless to undertake the transmission. The OpenMP model requires a multicore computer, so the number of cores, the basic frequency, and the memory size are the factors that influence the acceleration efficiency.

5. Conclusion

In this paper, high-performance dam-break flow simulation models based on the finite volume method were established. An explicit scheme was used to discretize the 2D shallow water control equations, and Roe's approximate Riemann solution for the finite volume method was adopted for the interface flux of the grid cells. For the dam-break flow observed in 1998 in the Pangtoupao flood storage area, three parallel computing methods, OpenMP, MPI, and OpenACC, were used to accelerate the dam-break model calculation. The characteristics of these three methods and their acceleration effects were compared and analysed. The conclusions are as follows.

(1) To obtain more accurate results, hydrodynamic models are usually used to simulate dam-break flooding instead of simplified methods. To speed up the calculation, an explicit discrete technique was used to solve the governing equations. Moreover, there is no correlation between the grid calculations in a single time step, so the 2D hydrodynamic dam-break model can be calculated in parallel using the loop structure of the grid calculations. This modification does not substantially change the original program code, so it is easy to convert the original serial model into one that can be computed in parallel.

(2) In the acceleration effect, the OpenACC method was the best. It achieved a 20.7 speedup ratio, with OpenMP and MPI achieving 9.8 and 5.1, respectively. The simulation time for the Pangtoupao dam-break event (320 h) was shortened from 170 h to 8.2 h with parallel computing, which effectively improved the simulation speed of the dam-break model.

(3) In the implementation method, the OpenMP parallel method had the lowest degree of difficulty, good reversibility, and high fault tolerance; it was easy to debug but it was difficult to achieve the best acceleration effect. The code rewrite of the OpenACC model was more difficult, and it was also difficult to debug, but its reversibility and fault tolerance were good. The code rewrite workload of MPI was very large, although the work was not very difficult because data division and transmission could be realized manually. Its reversibility was poor and it was difficult to debug on a multicomputer cluster. It is suitable for calculation cases with less data exchange and sufficiently large calculation tasks.

(4) In the case of small-scale calculations, if the memory of a single computer is enough to calculate the task, the OpenMP parallel method is recommended as a simple and effective method to achieve a better effect. If higher acceleration performance is required, the OpenACC parallel method should be considered. In the case of large-scale calculation, because of the single-machine memory bottleneck, the MPI method is a better choice. In addition, a combination of parallel methods, such as MPI + OpenMP or MPI + OpenACC, would also be effective for large-scale computing acceleration.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was supported by the National Key Research and Development Program of China (2016YFC0401407) and the National Natural Science Foundation of China (51379076).

References

[1] Sichuan Seismological Bureau, *The 1933 Diexi Earthquake*, Sichuan Science and Technology Press, Chengdu, China, 1983.

[2] M. Krupka, S. Wallis, S. Pender, and S. Neélz, "Some practical aspects of flood inundation modelling, transport phenomena in hydraulics, publications of the institute of geophysics," *Polish Academy of Sciences*, vol. 7, no. 401, pp. 129–135, 2007.

[3] P. Samuels, S. Huntington, W. Allsop, and J. Harrop, *Recent development and application of a rapid flood spreading model, River Flow 2008*, CRC Press, 2008, http://eprints.hrwallingford.co.uk/223/1/HRPP361-Recent_development_and_application_of_a_rapid_flood_spreading_method.pdf.

[4] B. Ghimire, A. S. Chen, M. Guidolin, E. C. Keedwell, S. Djordjević, and D. A. Savić, "Formulation of a fast 2D urban pluvial flood model using a cellular automata approach," *Journal of Hydroinformatics*, vol. 15, no. 3, pp. 676–686, 2013.

[5] S. Zhang, T. Wang, and B. Zhao, "Calculation and visualization of flood inundation based on a topographic triangle network," *Journal of Hydrology*, vol. 509, pp. 406–415, 2014.

[6] J. Singh, M. S. Altinakar, and Y. Ding, "Two-dimensional numerical modeling of dam-break flows over natural terrain using a central explicit scheme," *Advances in Water Resources*, vol. 34, no. 10, pp. 1366–1375, 2011.

[7] B. F. Sanders, J. E. Schubert, and R. L. Detwiler, "ParBreZo: A parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale," *Advances in Water Resources*, vol. 33, no. 12, pp. 1456–1467, 2010.

[8] J. C. Neal, T. J. Fewtrell, P. D. Bates, and N. G. Wright, "A comparison of three parallelisation methods for 2D flood inundation models," *Environmental Modelling and Software*, vol. 25, no. 4, pp. 398–411, 2010.

[9] E. R. Vivoni, G. Mascaro, S. Mniszewski et al., "Real-world hydrologic assessment of a fully-distributed hydrological model in a parallel computing environment," *Journal of Hydrology*, vol. 409, no. 1-2, pp. 483–496, 2011.

[10] A. R. Brodtkorb, M. L. Sætra, and M. Altinakar, "Erratum to 'Efficient shallow water simulations on GPUs: implementation, visualization, verification and validation' [Comp Fluids 55 (2012) 1-12] [MR2979690]," *Computers & Fluids. An International Journal*, vol. 59, 125 pages, 2012.

[11] J. M. Kelly, E. A. Divo, and A. J. Kassab, "Numerical solution of the two-phase incompressible Navier-Stokes equations using a GPU-accelerated meshless method," *Engineering Analysis with Boundary Elements*, vol. 40, pp. 36–49, 2014.

[12] A. Amritkar, D. Tafti, R. Liu, R. Kufrin, and B. Chapman, "OpenMP parallelism for fluid and fluid-particulate systems," *Parallel Computing*, vol. 38, no. 9, pp. 501–517, 2012.

[13] P. D. Bates and A. P. J. De Roo, "A simple raster-based model for flood inundation simulation," *Journal of Hydrology*, vol. 236, no. 1-2, pp. 54–77, 2000.

[14] J. Neal, T. Fewtrell, and M. Trigg, "Parallelisation of storage cell flood models using OpenMP," *Environmental Modelling and Software*, vol. 24, no. 7, pp. 872–877, 2009.

[15] S. Zhang, Z. Xia, R. Yuan, and X. Jiang, "Parallel computation of a dam-break flow model using OpenMP on a multi-core computer," *Journal of Hydrology*, vol. 512, pp. 126–133, 2014.

[16] I. Villanueva and N. G. Wright, "Linking Riemann and storage cell models for flood prediction," *Proceedings of the Institution of Civil Engineers: Water Management*, vol. 159, no. 1, pp. 27–33, 2006.

[17] I. Villanueva and NG. Wright, *An Efficient Multi-processor Solver for the 2D Shallow Water Equations. Hydroinformatics*, France, Nice, 2006.

[18] J. C. Pau and B. F. Sanders, "Performance of parallel implementations of an explicit finite-volume shallow-water model," *Journal of Computing in Civil Engineering*, vol. 20, no. 2, pp. 99–110, 2006.

[19] P. Rao, "A parallel RMA2 model for simulating large-scale free surface flows," *Environmental Modelling and Software*, vol. 20, no. 1, pp. 47–53, 2005.

[20] G. Oger, D. Le Touzé, D. Guibert et al., "On distributed memory MPI-based parallelization of SPH codes in massive HPC context," *Computer Physics Communications*, vol. 200, pp. 1–14, 2016.

[21] M. De La Asunción, J. M. Mantas, and M. J. Castro, "Simulation of one-layer shallow water systems on multicore and CUDA architectures," *Journal of Supercomputing*, vol. 58, no. 2, pp. 206–214, 2011.

[22] L. Grillo, F. De Sande, J. J. Fumero, and R. Reyes, "Programming for GPUs: The directive-based approach," in *Proceedings of the 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, pp. 612–617, fra, October 2013.

[23] B. P. Pickering, C. W. Jackson, T. R. Scogland, W.-C. Feng, and C. J. Roy, "Directive-based GPU programming for computational fluid dynamics," *Computers & Fluids. An International Journal*, vol. 114, pp. 242–253, 2015.

[24] A. Hart, R. Ansaloni, and A. Gray, "Porting and scaling OpenACC applications on massively-parallel, GPU-accelerated supercomputers," *The European Physical Journal: Special Topics*, vol. 210, no. 1, pp. 5–16, 2012.

[25] T. Hoshino, N. Maruyama, S. Matsuoka, and R. Takaki, "CUDA vs OpenACC: Performance case studies with Kernel benchmarks and a memory-bound CFD application," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013*, pp. 136–143, nld, May 2013.

[26] R. Reyes, I. López-Rodríguez, J. J. Fumero, and F. de Sande, "accULL: an OpenACC implementation with CUDA and OpenCL support," in *Euro-Par 2012 Parallel Processing*, vol. 7484 of *Lecture Notes in Computer Science*, pp. 871–882, Springer, Berlin, Germany, 2012.

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

[28] X. Xia and Q. Liang, "A GPU-accelerated smoothed particle hydrodynamics (SPH) model for the shallow water equations," *Environmental Modelling and Software*, vol. 75, pp. 28–43, 2016.

[29] K. F. Bradbrook, S. N. Lane, S. G. Waller, and P. D. Bates, "Two dimensional diffusion wave modelling of flood inundation using a simplified channel representation," *International Journal of River Basin Management*, vol. 2, no. 3, pp. 211–223, 2004.

[30] R. Lamb, M. Crossley, and S. Waller, "A fast two-dimensional floodplain inundation model," *Proceedings of the Institution of Civil Engineers: Water Management*, vol. 162, no. 6, pp. 363–370, 2009.

[31] S. Zhang, R. Yuan, Y. Wu, and Y. Yi, "Parallel computation of a dam-break flow model using OpenACC applications," *Journal of Hydraulic Engineering*, vol. 143, no. 1, Article ID 04016070, 2017.

[32] S. Zhang, R. Yuan, Y. Wu, and Y. Yi, "Implementation and efficiency analysis of parallel computation using OpenACC: a case study using flow field simulations," *International Journal of Computational Fluid Dynamics*, vol. 30, no. 1, pp. 79–88, 2016.

[33] A. J. Rueda, J. M. Noguera, and A. Luque, "A comparison of native GPU computing versus OpenACC for implementing flow-routing algorithms in hydrological applications," *Computers and Geosciences*, vol. 87, pp. 91–100, 2016.

[34] E. F. Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics*, Springer, Berlin, Germany, 3rd edition, 2009.