VNU Journal of Science, Earth Sciences 25 (2009) 104-115

Application of a Godunov type numerical scheme and a domain decomposition technique to the parallel computation of tidal propagation

Nguyen Tat Thang*, Nguyen The Hung

Institute of Mechanics
Vietnam Academy of Science and Technology, VAST

Received 15 June 2009; received in revised form 23 June 2009

Abstract. A parallel computing model for the numerical solution of the general 2D shallow water equations in conservative form has been developed, tested and implemented in the MPI parallel environment set up on a parallel computer with four 2.8 GHz CPUs in Institute of Mechanics, VAST. The model is based on a Godunov-type numerical scheme, which is devised for 2D unstructured computational meshes, and on a domain decomposition technique. That newly developed parallel computing model has been applied to a tidal wave propagation problem in the Gulf of Tonkin in the South China Sea. The computed results show good agreement with that of the TIDE2D model. It is of high importance that computation time is significantly reduced.

Keywords: Parallel Computing; Godunov Type Numerical Scheme; 2D Shallow Water Model; Domain Decomposition Technique.

1. Introduction

Numerical methods have long been developed and applied to many scientific problems. The finite volume method (FVM) which is conservative has recently been applied to fluid dynamics simulations and good results have been obtained [1]. The application of the FVM method in combination with a Godunov type numerical scheme and the approximate Riemann solver of Roe is a prospective approach for the numerical solution of the shallow water equations [2, 3]. This method is especially appropriate in problems where hydraulic jumps, shockwaves or wave propagation may present [4]. A numerical model based on this approach for the solution of the general 2D shallow water equations in conservative form has been developed, validated and applied to test cases. That model shows high potential for application to practical problems. Moreover, the unstructured computational mesh used in the model is highly flexible in dealing with the complex geometry of 2D domains [5].

When applying this model to large spatial and/or temporal scale problems, such as the simulation of tidal propagation in the Gulf of Tonkin in the South China Sea or of the process

* Corresponding author: Tel.: (+44) 01224 273519.
Email: thang.tat.nguyen@abdn.ac.uk

of floods in the Red River system in Vietnam etc., for a long period of time (days to months), this method requires massive computation time even with the most powerful PCs. It is therefore of essential importance to reduce computation time by applying parallel computing.

Recently parallel computing based on low cost parallel cluster computers (distributed systems) or ordinary parallel computers with multiple CPUs has been applied in some research institutions in Vietnam. Applications of such systems to fluid dynamics problems have been being carried out in the Institute of Mechanics, VAST. Firstly we applied a distributed system based on PVM (Parallel Virtual Machine) environment built on a local network of heterogeneous ordinary PCs. The efficiency of the system largely depends on the degree of the heterogeneity of the PCs and the speed of the local network. Recently parallel computing based on MPI (Message Passing Interface) and a single parallel computer with four CPUs have been studied and applied.

Based on our study of the parallelization techniques and on our numerical model, a domain decomposition strategy is applied to parallelize the solution algorithm of the numerical model. Then a parallel computing model has been developed in the MPI environment and applied to the simulation of tidal propagation in the Gulf of Tonkin. This problem is of very large spatial and temporal scales.

This paper first presents the numerical model for the solution of the 2D shallow water equations in conservative form in Part 2. Parallelization techniques are briefly mentioned in Part 3. Part 4 shows the parallelization of the numerical model. Parallel computations for the problem of tidal propagation in the Gulf of Tonkin are presented in Part 5. And computed results, comparisons of computation times, analysis of the computation time efficiency and remarks are also given in that part. Finally conclusions and remarks are stated in Part 6. A list of references also provided at the end of this paper.

2. Numerical model for the solution of the 2D shallow water equations

2.1. The system of equations

The model is based on the 2D system of the unsteady Saint-Venant equations written in conservative form as shown below [5]:

$$\frac{\partial U}{\partial t}+\frac{\partial F(U)}{\partial x}+\frac{\partial G(U)}{\partial y}=S(x,y,U) \quad (1)$$

where $U=\begin{pmatrix} h \\ q_x \\ q_y \end{pmatrix}$ (conservative variable),

$$F=\begin{pmatrix} q_x \\ \frac{q_x^2}{h}+\frac{gh^2}{2} \\ \frac{q_x q_y}{h} \end{pmatrix}, \quad G=\begin{pmatrix} q_y \\ \frac{q_x q_y}{h} \\ \frac{q_y^2}{h}+\frac{gh^2}{2} \end{pmatrix},$$

$q_x=uh$, $q_y=vh$; h is the water depth; g is the gravity acceleration; (u,v) are the x and y components of the depth averaged velocity respectively; S is the source term. Equation (1) can be rewritten in the following form:

$$\frac{\partial U}{\partial t}+\nabla.E(U)=S(x,y,U) \quad (2)$$

where $E=\begin{pmatrix} F \\ G \end{pmatrix}$.

The unknowns that need to be computed are h, q_x and q_y or h, uh and vh.

2.2. Numerical technique

For a fixed control volume Ω as shown in Fig.1, the integral form of (2) is written as:

$$\int_{\Omega}\frac{\partial U}{\partial t}d\Omega + \int_{\Omega}\nabla.E(U)d\Omega = \int_{\Omega}S(x,y,U)d\Omega \quad (3)$$

Applying the Gauss's theorem, (3) can be rewritten in the following form

$$\frac{\partial}{\partial t}\int_{\Omega}Ud\Omega + \oint_{\partial\Omega}(E.n)ds = \int_{\Omega}Sd\Omega \quad (4)$$

where $\partial\Omega$ denotes the boundary surface of the 2D volume Ω, and n is the unit outward normal vector (Fig.1).

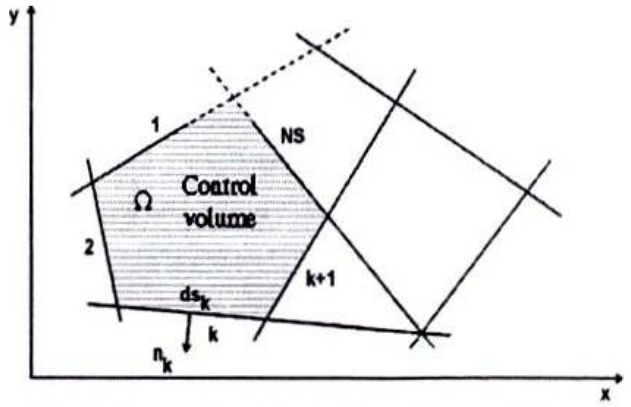

Fig. 1. A control volume (element or cell) in two dimensions (NS: number of sides; ds_k: the length of the side k).

Since equation (4) is written for each individual control volume (an element or cell of the computational meshes), the discretization technique is applied to each element. Denoting by U_i the average (or discrete) value of conservative variable over the volume Ω_i, using equation (4), the following conservation equation can be written for each cell i:

$$\frac{\partial U_i}{\partial t}A_i + \oint_{\partial\Omega_i}(E.n)ds = \int_{\Omega_i}Sd\Omega \quad (5)$$

where A_i is the area of the 2D volume Ω_i [5].

Applying the mid-point rule to approximate the contour integral in (5) and a simple approximation for the time derivative, a finite difference like form of (5) is written as:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{A_i}\left(\sum_{k=1}^{NE}E_k^*.n_k.ds_k\right)_i^n + \Delta t S_i^{*n} \quad (6)$$

The ideas of the Godunov method and the Roe's approximate Riemann solver [2], which are originated in aerodynamics, are applied to the approximation of the E_k^* flux [4].

All details of the system of equations and discretization scheme should be referred to [5].

As for boundary conditions, the model uses three types of boundary conditions. Each of those is used where relevant. The first one is the condition of the river water discharges from river outlets flowing into the simulation domain. The second one is the reflective and no-slip boundary condition applied to rigid boundaries. And the last one is the free flow condition at open sea boundaries [5].

The numerical scheme shown here, for unstructured meshes in general, is highly efficient for the solution of the propagation of waves in spatial domains of complicated geometry [4]. Therefore it will be applied in this study.

3. Parallelization techniques

3.1. Parallelization

Parallel computing is based on partitioning a main task, which is usually complicated and time consuming, into small tasks that will be executed simultaneously on separate processors. There are a number of parallelization techniques that can be applied to enhance the computation process. The most popular one is using an appropriately written routine that automatically analyzes the up coming tasks. This analysis provides necessary information to the processors and these processors can exchange information among each other to solve the tasks [6, 7].

It should be aware that not all problems can be parallelized. If there are no tasks that can be

processed simultaneously or if the tasks are strongly dependent (fine-grained), all attempt to parallelize the problem may lead to increasing computation time.

In practice, there are many complicated scientific problems that can be decomposed into small, separate tasks to run simultaneously on multiple processors. The most common approach is applying the spatial or temporal decomposition to partitioning the domain of the problem. So the original whole domain will be partitioned into space and/or time sub-domains. These sub-domains are used in parallel computation. A typical example is the simulation of flood process where computation is performed on separate control volumes or sub-regions. That problem is of the coarse-grained type and can be parallelized with little difficulty. In that case, high performance parallel computing can be attained [6,7].

However most of scientific problems needed to be parallelized are between coarse-grained and fine-grained ones. Parallel computing applied in such cases requires interactions among sub-domains. Therefore the processors have to exchange data and information among each other. The technical term of this exchange process is message passing. An example where message passing is required is when we decompose the original domain of a fluid flow into two parts and assign information of each part to two individual processors. In this case, the two processors must exchange information of the shared boundary between the two sub-domains in order to carry out computation in each sub-domain since the flow condition at the shared boundary is now the natural boundary condition of each sub-domain. Hereinafter, a shared boundary is the boundary that appears due to the sub-domain decomposition. Natural boundaries imply the boundaries of the whole original domain.

The partitioning of the original computation domain into small sub-domains should be performed carefully since there is always dependency among these sub-domains. In many cases, the exchanged information may become extremely huge that the required time for the massage passing process exceeds the computation time. In such cases, the parallelization has negative result. It is therefore of critical importance to consider the balance between the computation time and the time needed for message passing [6,7].

3.2. Message passing

There are a number of message passing techniques used in parallel computing systems. Some examples are Threads and Inter-Process Communication (IPC) used in a single system with multiple processors, or other techniques such as TCP sockets, Remote Procedure Calls (RPCs), or even message passing techniques based on the storing and exchanging of information by means of using shared files in the system [8]. However the most efficient way is using particular libraries pre-developed for the purpose of message passing in a local network or in a single system. There are two most popular libraries: PVM and MPI. Because of the popularity of those libraries, parallel codes with message passing based on PVM or MPI can be executed in many computing systems ranging from laptops to large mainframe CRAY systems.

The PVM library can be used in different parallel computing platforms. This library is written for applications based on the programming languages such as C, C++, and FORTRAN. The recent versions can run on Java or Python systems. This library has advanced features appropriate for complicated distributed-computing applications. However its execution is quite slow due to the

mechanism of the calls to execute message passing sub-routines. Moreover the calls, which are related to the organizing of exchanged messages, do not have a sense of a friendly programming environment on its own [9, 10].

On the contrary, MPI is another message passing library released in 1994 with some different features. This library has special advantageous if applications is running on a single Massively Parallel Processor (MPP). MPI has more communication functions than PVM does. Therefore MPI is of much interest when applications are intended to use special communication modes which are not available in PVM. A usually cited example is the non-blocking send function in MPI. In our parallel computing system, since we use a single parallel computer with 4 CPUs, the MPI library is most appropriate [10, 11].

3.3. Three steps of the parallelization of a problem

Typically there are three steps in parallelizing a problem. These steps include [12]:

a) Decomposing the problem into sub-domains and sub-data. Subsequently, in this step, there are three possibilities:

- Complete or full parallelization: the problem can be fully parallelized if it can be decomposed into individual tasks that rarely or even may not need to exchange information among each other. The parallel efficiency may reach 100% in this case. Such problems are really few.

- Domain decomposition parallelization (data decomposition): to apply this method, the data of the problem usually has main features: large, separate and static.

- Control decomposition: this method is applied to problems with non-static data and an unfixed number of tasks. In this case, data decomposition can not be applied. However each processor will have different tasks and we can control the task flow of the application to parallelize the problem.

b) Assignment: this step is mapping data and distributing tasks to nodes. This step is closely related to the concept of load balancing that determine the breaking of a program into tasks and assigning those tasks to nodes based on processor (node) availability.

c) Orchestration: this step aims to orchestrate threads to reduce communication and synchronization costs, and increase parallel efficiency.

Considering the parallelization techniques and our numerical method shown above, the spatial domain decomposition technique is the most suitable one. Therefore the technique is applied in this study.

4. Parallelization of the numerical model

4.1. Domain decomposition and data transmission

Since each sub-domain among decomposed sub-domains may shares at least one boundary with the remained sub-domains. We consider and show below the parallel algorithm in the case that 2 CPUs are used. In that case each CPU always controls only one sub-domain. The algorithm, when we have more than 2 sub-domains i.e. more than 2 CPUs, is simply a multiple of the algorithm in the case of 2 CPUs.

- Assuming that the original whole domain is divided into triangular computational cells (or unstructured meshes in general) and is decomposed into two sub-domains (named sub-domain I and sub-domain II) as shown in Fig.2. The two sub-domains share the boundary denoted by the bold red line (shared boundary). CPU 1 is responsible for the left sub-domain and CPU 2 for the right sub-domain.

- Considering 2 cells numbered I and IV as shown in Fig.2. Cell 1 is surrounded by cells II, III and IV. Cell IV is surrounded by cells I, V and VI.

- At time step n, the required input to compute h, hu and hv for cell IV is the values of h, hu and hv, which were computed during the previous time step, of the cells I, V and VI. CPU 1 already stores the values of cell V and VI. The values of cell I were computed in CPU 2 and now need to be transmitted to CPU 1. Similarly we see the same calculation and transmission for cell I. Generally speaking, if we have N shared boundary cells in each sub-domains, at each time step we need to exchange $3N$ values of h, hu and hv from CPU 1 to CPU 2 and vice versa. In the case of 2 CPUs, the number of variables that needs to be exchanged between them is $6N$.

Further assuming that the original domain has M computational cells and each sub-domain has $M1 = M/2$ cells. Then the steps of the computation each CPU does, in the case of 2 CPUs i.e. 2 sub-domains, include:

Step 1: Each CPU reads spatial and geometric data, initial and natural boundary conditions of the whole domain, and information of the shared boundary then does computation for the whole domain (for the first time step only).

Step 2: For next time steps, each CPU does the following tasks:

CPU 1:

- Do computation for the sub-domain I.

- Send $3N$ values of the shared boundary to CPU 2.

- Receive $3N$ values of the shared boundary from CPU 2.

CPU 2:

- Do computation for the sub-domain II.

- Send $3N$ values of the shared boundary to CPU 1.

- Receive $3N$ values of the shared boundary from CPU 1.

Step 2 is repeated until the last time step, i.e. when the computation completes.

Step 3: CPU 1 receives all the computed results of the sub-domain II, of all time steps, from CPU 2. In this step:

- CPU 1: Receive $3M1$ values of the sub-domain II from CPU 2.

- CPU 2: Send $3M1$ values of the sub-domain II to CPU 1.

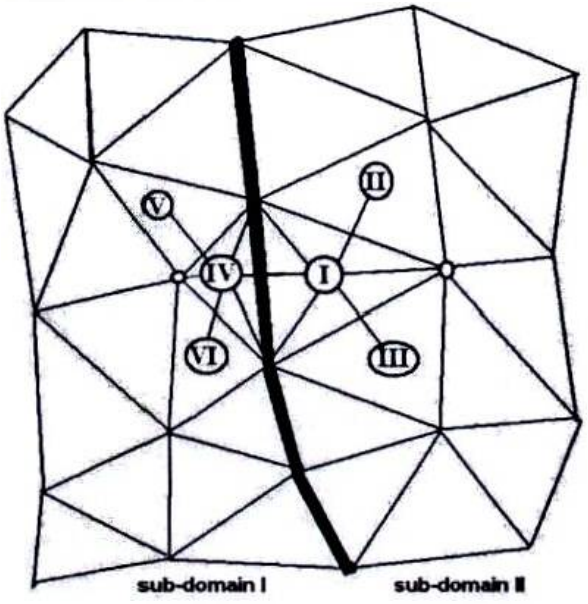

Fig. 2. A sketch of 2 sub-domains and computational cells.

Remarks:

- The reduction of the computation time comes mainly from Step 2 which is repeated.

- The computation time of the sequential computation is equal to the computation time of one CPU for the whole domain consisting of M cells. The computation time of the parallel computation is equal to the computation time one CPU needs to complete computation for one sub-domain consisting of $M/2$ cells, plus the data transmission time.

Generalizing for the case that we have NP CPUs (in that case each sub-domain has M/NP cells):

- Depending on the mesh generation technique used, each sub-domain may have one or more shared boundaries with other sub-domains. The optimizing mesh generation strategy is the one that reduces the most the numbers of shared boundary cells and ensures those numbers, of all shared boundaries, are about equal to each other. That feature is important to reduce the time of sending and receiving data and to synchronize the data transmission processes among CPUs.

- The computation time of the parallel computation is equal to the computation time one CPU needs to complete computation for one sub-domain consisting of M / NP cells, plus the data transmission time.

4.2. The flowchart of the parallel computation

The flowchart of the parallel computation executed in each CPU is shown below:

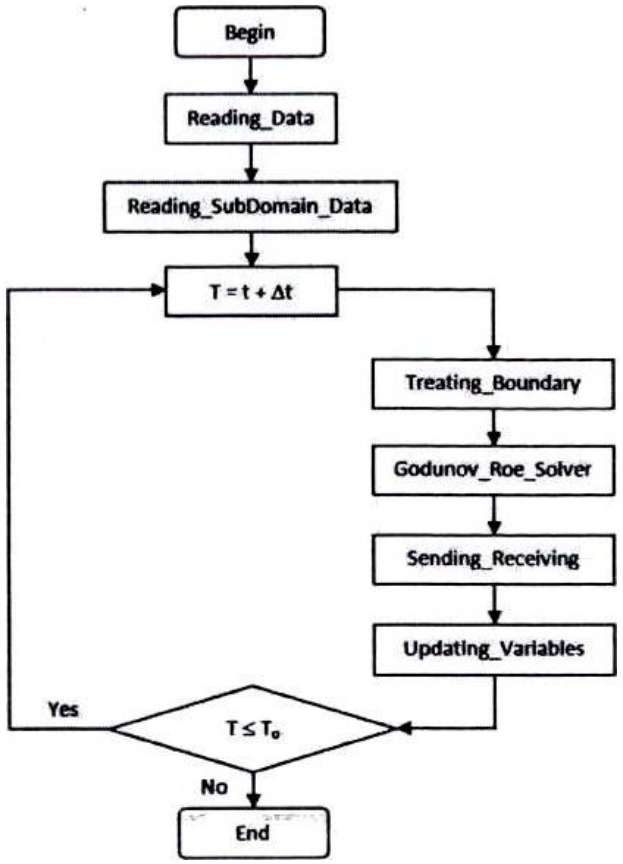

Fig. 3. The flowchart of the parallel computation executed in each CPU.

The routines and their features are:

- Reading_Data: this is the first routine called when the program is executed. This routine reads the data of the computational meshes, domain geometry, natural boundary and initial conditions of the original whole domain. It also does initializing tasks required for the whole parallel computing process.

- Read_SubDomain_Data: this routine reads control data of each sub-domain, and information of the shared boundaries. Each CPU stores the information of its sub-domain and of its shared boundaries, and then does computation for only its pre-assigned sub-domain.

- Treating_Boundary: this routine calculates (interpolates) values of the variables at different types of natural boundaries using boundary data provided.

- Godunov_Roe_Solver: this routine solves the 2D shallow water equations based on the numerical method shown above.

- Sending_Receiving: this routine sends/receives new computed values of the variables of the cells along the shared boundaries to/from other sub-domains. This routine uses MPI_SEND() function to send data and MPI_RECV() function to receive data.

- Updating_Variables: this routine updates new computed values. Those values are used in the next time steps.

5. Parallel computation for the problem of tidal propagation in the Gulf of Tonkin

5.1. Geometric data, mesh generation, boundary conditions and domain decompositions

The computational domain is shown in Fig.4. The computational meshes, which are

unstructured triangular meshes, are generated using a commercial mesh generation package (Fig.5) [13].

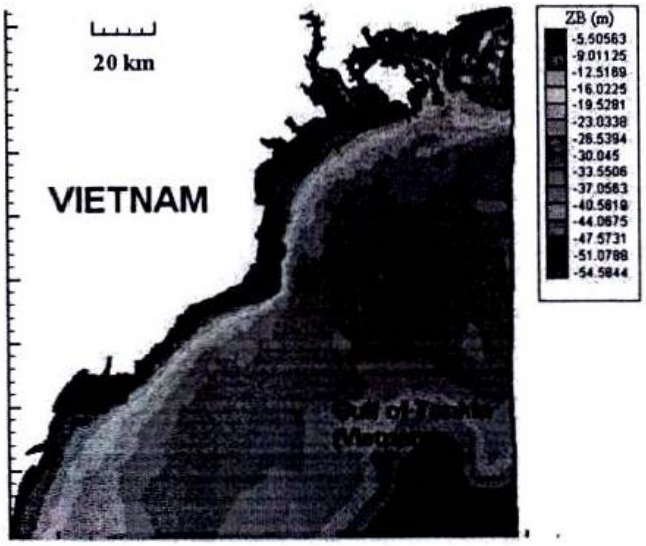

Fig. 4. The geometry of the Gulf of Tonkin.

In this study, there is no wind data so that wind effect is not considered. Further improvements of the model will include wind effect in the source term of the governing equation.

Natural boundaries are shown in Fig.6 below. The following conditions are used at those boundaries.

- River discharges into the computational domain are obtained from the computed results by a one-dimensional hydraulic model developed at the Institute of Mechanics.
- Reflective and no-slip boundary conditions are used at the rigid boundary.
- Free-flow and water level (tidal level) boundary conditions are applied at the open sea boundaries.

As for initial condition, constant water level and zero velocity are set for the whole domain.

The computational meshes for the whole original domain and the natural boundaries are shown in Fig.5 and Fig.6 respectively below.

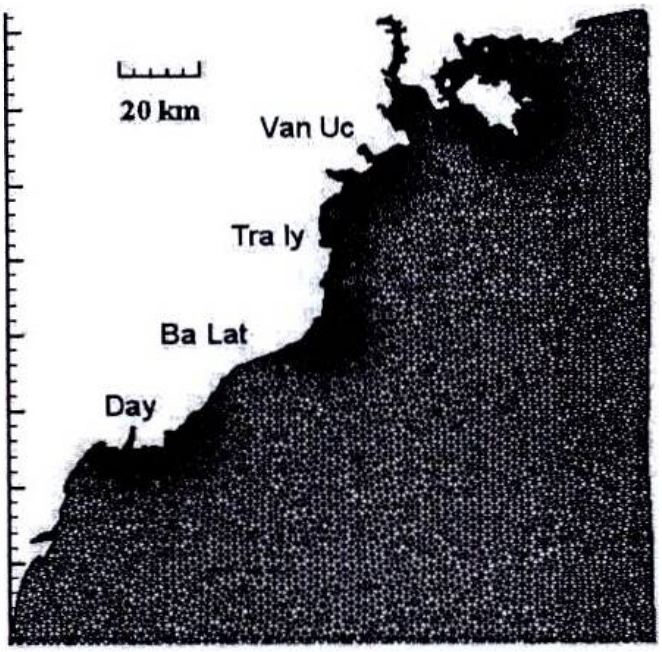

Fig. 5. The computational meshes of the whole computational domain and the river outlets.

The whole domain is divided into 24249 triangular meshes and 12928 nodes (the total number of the vertexes of triangles).

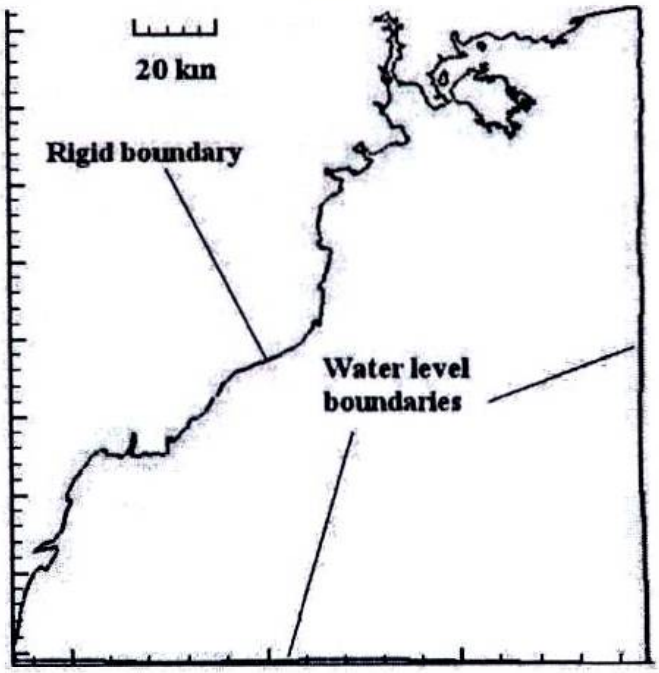

Fig. 6. Natural boundaries.

Three cases of sub-domain decompositions, which correspond to 2, 3 and 4 sub-domains, are shown below in Fig.7 to Fig.9 respectively.

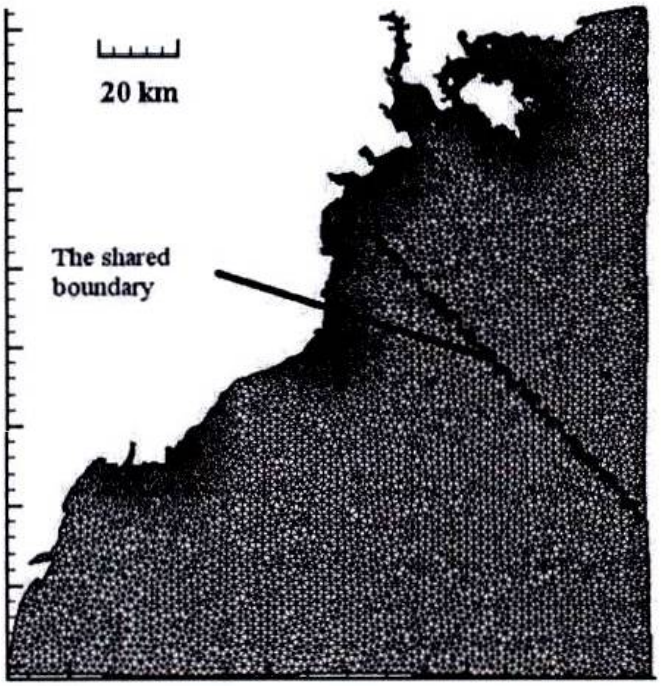

Fig. 7. Sub-domain decomposition: 2 sub-domains.

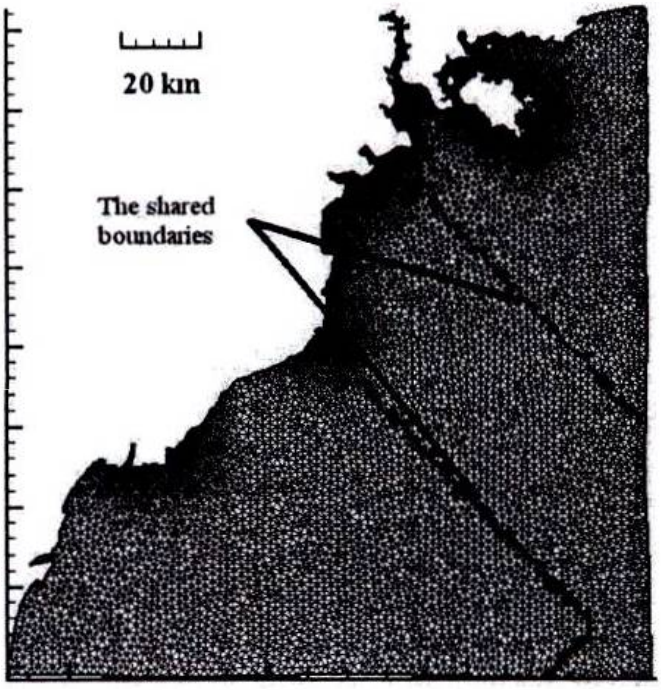

Fig. 8. Sub-domain decomposition: 3 sub-domains.

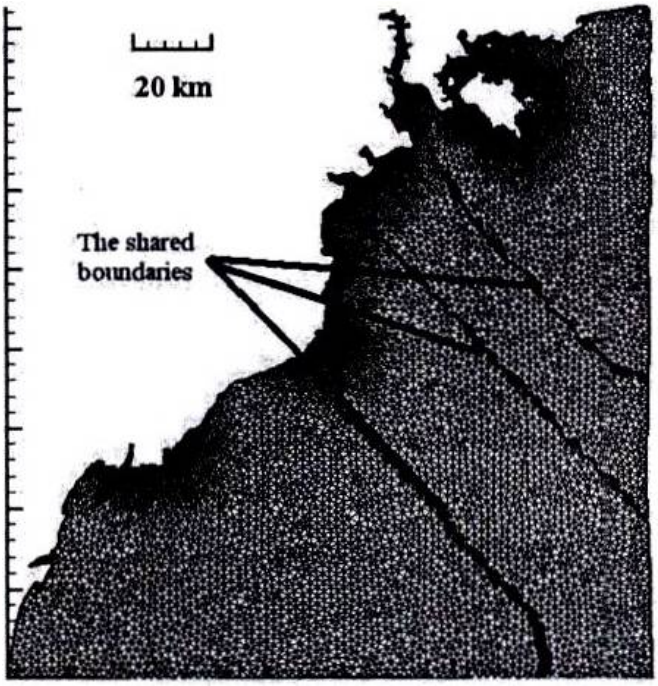

Fig. 9. Sub-domain decomposition: 4 sub-domains.

5.2. Computed results and comparisons

The computed results by parallel computations are first compared with that by a sequential computing model TIDE2D which is also developed, validated and applied in Institute of Mechanics, VAST. Both models are applied to simulate tidal propagation in the Gulf of Tonkin during the flood seasons in 2004 and 2005. The water level computed by the two models at the river outlets Day, Ba Lat, Van Uc and Tra Ly (Fig.5) are used in the comparisons. Good agreements are obtained as shown in the figures below. For ease of showing the computed results, the parallel model is now called PARALLEL2D. It should be noted here that, the computed results by the PARALLEL2D model shown in from Fig.10 to Fig.13 were obtained when computations were performed using only one CPU for the whole domain (no parallel computation, i.e. no sub-domain decomposition).

- Computed results from 1:00pm on July 14, 2004 to 23:00pm on August 12, 2004

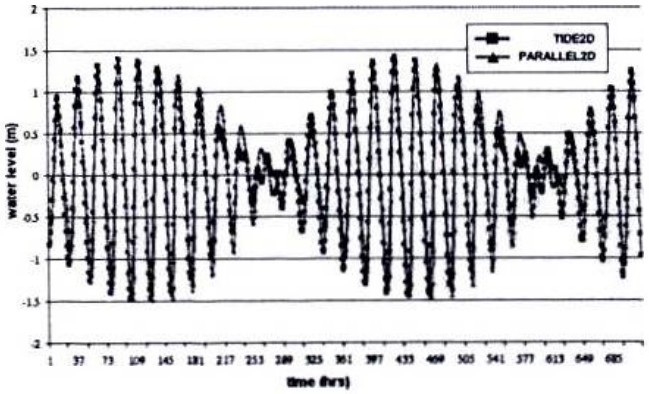

Fig. 10. Computed water levels at Day outlet.

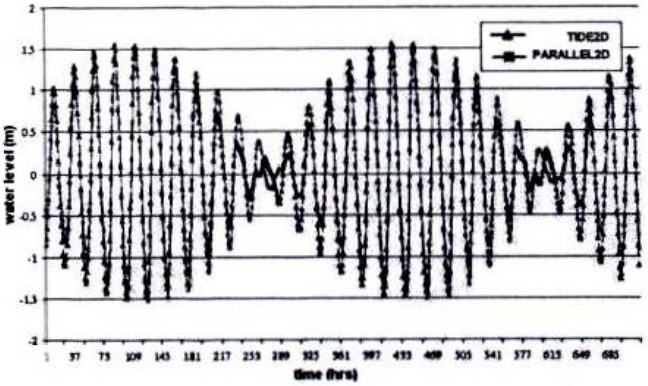

Fig. 11. Computed water levels at Ba Lat outlet.

- Computed results from 1:00pm on June 01, 2005 to 23:00pm on June 30, 2005

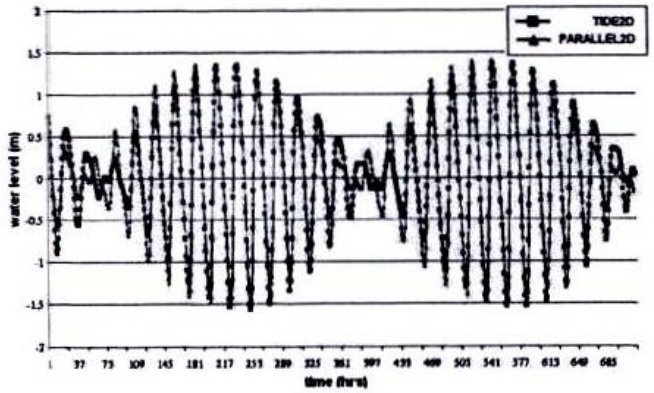

Fig. 12. Computed water levels at Van Uc outlet.

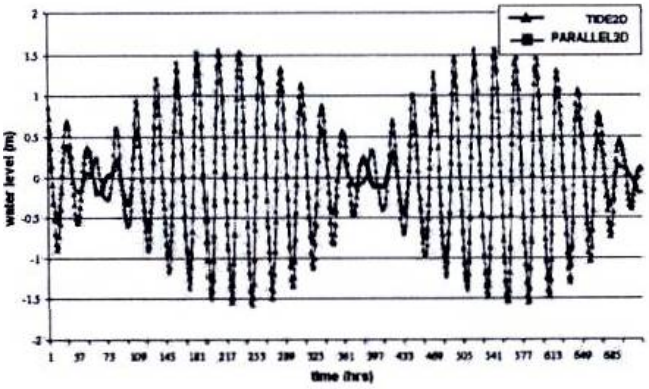

Fig. 13. Computed water levels at Ba Lat outlet.

Next are the comparisons among the computed results by the PARALLEL2D model when parallel computations are performed using different numbers of CPUs. The numbers of CPUs, i.e. the numbers of sub-domains respectively, used in parallel computations are 1, 2, 3 and 4 CPUs. Those comparisons are also of high importance before considering the computation time efficiency since they ensure the consistency among the computed results when the original domain is decomposed into different numbers of sub-domains. The comparisons were performed for the whole domain. It is important to emphasize that the computed results, when 1, 2, 3 and 4 CPUs were used, are all the same. For brevity, computed results at Day and Tra Ly outlets from 1:00pm on June 01, 2005 to 23:00pm on June 30, 2005 are shown in Fig.14 and 15 below.

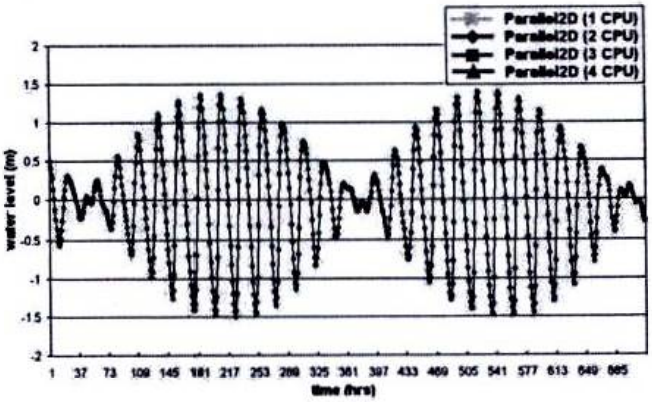

Fig. 14. Computed water levels at Day outlet.

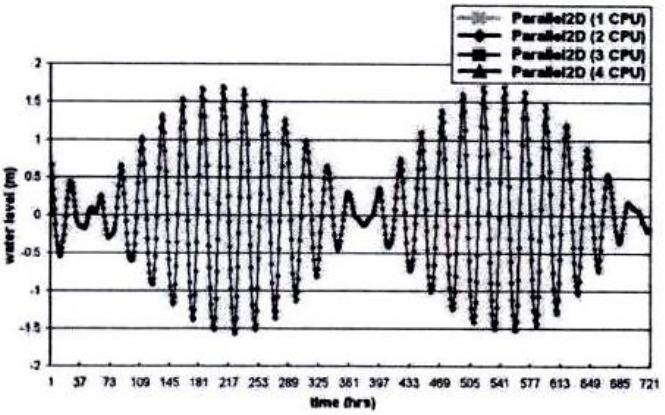

Fig. 15. Computed water levels at Tra Ly outlet.

Results of computation time efficiency calculated corresponding to each number of CPUs used, which are 1, 2, 3 and 4 CPUs, are shown in Fig.16.

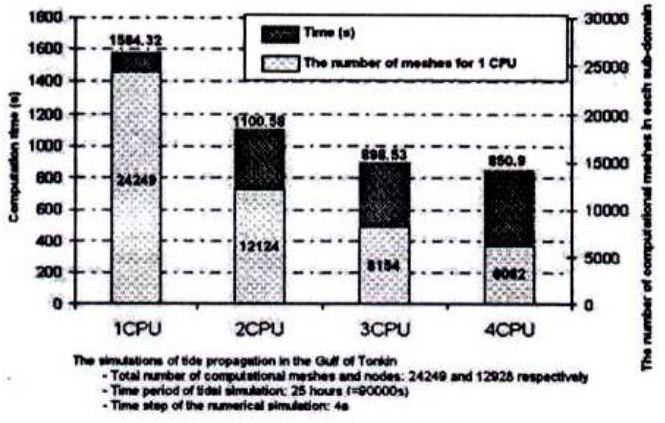

Fig. 16. Computation time corresponding to each number of CPUs used.

Fig.16 shows that computation time, when the numbers of CPUs used are 2, 3 and 4 CPUs, is 69.5%, 56.7% and 53.7% respectively of the computation time when only one CPU is used (no parallel computation). The reduction of computation time is not linear since when the number of CPUs increases, i.e. the number of sub-domains increases, the required time for data transmission increases disproportionately. On the overall, computation time is reduced significantly when parallel computations are performed using multiple CPUs.

6. Conclusions

The domain decomposition parallelization, which is applied to the Godunov type numerical scheme for parallel computation of the 2D shallow water model, has been carried out. The research results are highly prospective. When the parallel model is applied to simulate tidal propagation in the Gulf of Tonkin, on a parallel platform with four 2.8GHz CPUs, the computation time is only about half of that of the sequential computation using only one CPU. This parallel model should be further improved to run on other parallel platforms with more CPUs.

References

[1] C.A.J. Fletcher, *Computational techniques for fluid dynamics I & II*, Springer, 1991.

[2] P.L. Roe, Approximate Riemann solvers: parameter vectors and difference schemes, *Journal of Computational Physics* 43 (1981) 357.

[3] F. Alcrudo and P. Garcia-Navarro, A high-resolution Godunov-type scheme in finite volumes for the 2D shallow-water equations, *International Journal for Numerical Methods in Fluids* 16 (1993) 489.

[4] E.F. Toro, *Riemann Solvers and Numerical Methods for Fluids Dynamics: A Practical Introduction* (2nd Edition), Springer Verlag, Berlin, 1999.

[5] Nguyen Tat Thang, Nguyen Van Hanh and Nguyen The Duc, Some initial results of the application of a Godunov type scheme to the numerical solution of the 2D shallow water equations, *Proceedings of the Annual National Conference on Fluid Mechanics*, 2004, 565 (in Vietnamese).

[6] *An Introduction to parallel programming*, Hanoi University of Technology (in Vietnamese).

[7] Geoffrey C. Fox, Roy D. Williams, and C. Paul, *Messina Parallel Computing Works*, Morgan Kaufmann, 1994.

[8] Kay Robbins and Steve Robbins, *UNIX Systems Programming: Communication, Concurrency and Threads* (2nd edition), San Antonio, Texas, 2004.

[9] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S. Sunderam, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*, The MIT Press (1994).

[10] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos, PVM and MPI: a Comparison of Features, *Calculateurs Paralleles* 8 (1996) 137.

[11] Yukiya Aoyama and Jun Nakano, RS/6000 SP: *Practical MPI Programming, International Technical Support Organization*, SG24-5380-00, 1999.

[12] Joseph D. Sloan, *High performance Linux clusters with OSCAR*, Rocks, openMosix, and MPI, O'Reilly, 2004.

[13] Hoang Van Lai, Nguyen Thanh Don, Nguyen Hong Phong, An optimizing mesh-generation for the parallelization of a 2D hydraulic model, *Proceedings of the Annual National Conference on Fluid Mechanics*, 2007, 307 (in Vietnamese).