A parallel adaptive finite volume solver for two-dimensional shallow-water equations

Zhou Zhang*, Xianliang Gong

Department of Naval Architecture and Marine Engineering, University of Michigan, 48109, MI, USA

1 Introduction

Floods and tsunamis, triggered by earthquakes or volcanic eruptions, can cause enormous damage to human society, especially for the coastal area. The accurate simulation of them is of critical importance in hazard assessment and precaution. However, numerical approaches solving hydrodynamical models can be computationally expensive. The adaptive mesh refinement (AMR) is a technique adapting the accuracy of a solution by refining or coarsening the mesh locally based on problem-dependent criteria, which effectively reduces the computational costs compared to the static-mesh approaches. When combing with parallel computing, the AMR involves the repartition of the mesh and the data transfer between the processors at each time step, requiring careful consideration to ensure correctness and efficiency.

In this project, we solve the two-dimensional (2D) nonlinear shallow-water equations using a first-order finite volume method. A two-level AMR is developed with a physics-based refinement indicator. The program runs in parallel under the Message Passing Interface (MPI) and the associated load balancing is conducted by a simplified approach based on space-filling curves. We finally benchmark the performance of our algorithm for a broad range of mesh and processor combinations.

2 Methods

2.1 Numerical model and the finite-volume solver

We consider the 2D shallow-water equations with constant fluid density:

$$\frac{\partial}{\partial t}(h) + \frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) = 0 \quad \text{(continuity)}, \tag{1}$$

$$\frac{\partial}{\partial t}(hu) + \frac{\partial}{\partial x}(hu^2 + gh^2/2) + \frac{\partial}{\partial y}(huv) = 0 \quad (x-\text{momentum}), \tag{2}$$

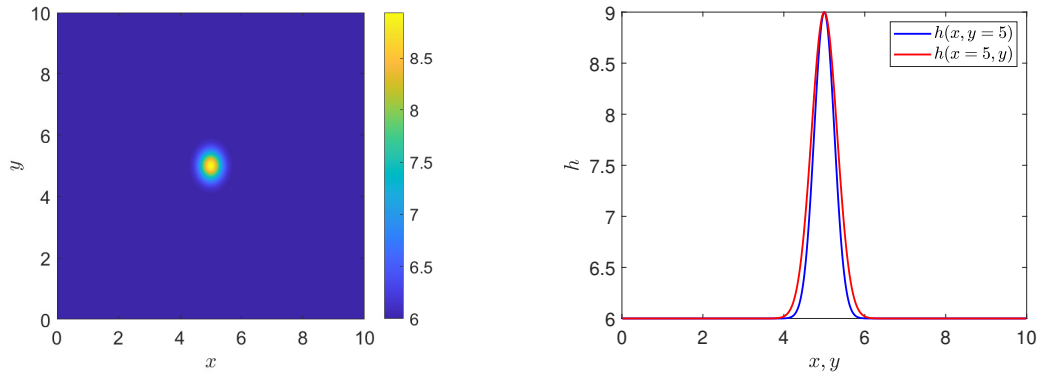
$$\frac{\partial}{\partial t}(hv) + \frac{\partial}{\partial x}(hvu) + \frac{\partial}{\partial y}(hv^2 + gh^2/2) = 0 \quad (y-\text{momentum}), \tag{3}$$

where h is the water height [m], u and v are respectively the water velocities [m/s] at x and y directions, and $g = 9.8\text{m/s}^2$ is the gravitational acceleration. The computational domain is a square basin $\Omega = [0, L] \times [0, L]$ with $L = 10\text{m}$ and no-penetration boundary conditions on all four edges. The initial conditions are set to:

$$\begin{aligned} h^0(x, y) &= 6 + 3\exp\left[-8(x-5)^2 - 5(y-5)^2\right] \text{ [m]}, \\ u^0(x, y) &= 0 \text{ [m/s]}, \\ v^0(x, y) &= 0 \text{ [m/s]}, \end{aligned} \tag{4}$$

*joezhang@umich.edu

with height shown in figure 1 and zero velocity throughout the domain.



(a) (b)

Figure 1: Contour (a) and section profiles along $x = 5$ and $y = 5$ (b) of the initial height $h^0(x, y)$.

The first-order finite-volume method is used to solve (1)-(3), in which the state variable $\mathbf{u} = [h, u, v]^T$ is approximated to be a constant in each cell. At the n-th time step, the value of $\mathbf{u}$ in cell i is updated by

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \frac{\Delta t^n}{A_i}\mathbf{R}_i^n, \tag{5}$$

where Δt^n is the time step size, A_i is the area of cell i and $\mathbf{R}_i^n$ is the residual on cell i. In this project, a constant global time step $\Delta t^n = \Delta t$ is used, which is obtained from the CFL condition at $t = 0$:

$$\Delta t = \min_i \left(\frac{C\ d_i^0}{s_{max,i}^0} \right), \tag{6}$$

where d_i^0 is the length of each edge of cell i, $s_{max,i}^0$ the maximum wave speed. C is the CFL number set to 0.45 to ensure stability of the numerical method. The residual is given by

$$\mathbf{R}_i = \sum_{e=1}^{m_e} \hat{\mathbf{F}}(\mathbf{u}_i, \mathbf{u}_{N(i,e)}, \vec{n}_{i,e}) d_i, \tag{7}$$

where e is the index of the edges of the cell, m_e is the number of edges in cell i, $N(i, e)$ is the neighbor cell of i across edge e, and $\vec{n}_{i,e}$ is the normal pointing out of cell i on edge e. Finally, $\hat{\mathbf{F}}$ is the numerical flux function, for which the Roe flux (Roe, 1981; Ambrosi, 1995) is used in this project. In this program, the calculation of $\hat{\mathbf{F}}$ is the most time-consuming part which needs to run in parallel for an accelerated simulation.

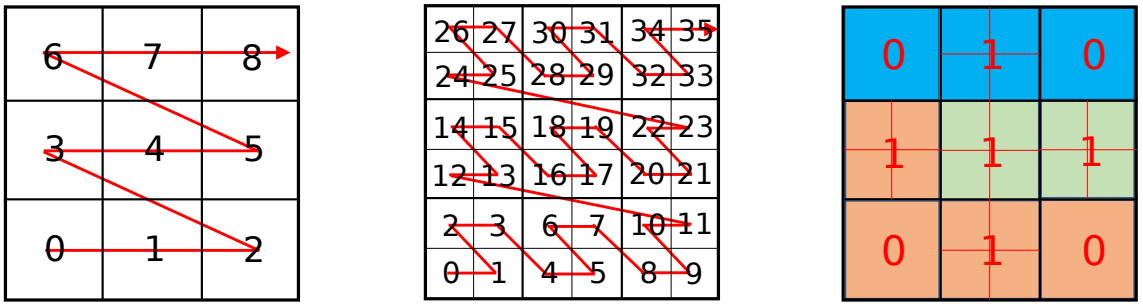
2.2 Adaptive mesh refinement

In our solver, two levels of structured grids, named as coarse and fine meshes, are used to uniformly discretize the domain. The coarse grids are numbered in a Z-order from $i = 0$ to $i = N^2 - 1$ with four fine grids $4i \sim 4i + 3$ sitting in ith coarse grid (figure 2). The refinement of the ith coarse grid is triggered by a physics-based indicator function (similar to Beisiegel et al. (2020)):

$$I_i(t) = \begin{cases} 0, & |\nabla h_i(t)| \leq G \\ 1, & |\nabla h_i(t)| > G \end{cases}, \tag{8}$$

where G is a threshold set as $G = 0.1$ in our simulations. This setting leads to moderate amount of fine grids near the regions with large surface slope. An example of I is shown

in figure 2(c). Specifically, when a coarse grid is refined, its states are copied to 4 corresponding fine grids; when a fine grid is coarsened, the averages of their states are assigned to 1 corresponding coarse grid. During the simulation, the regions with $I = 1$ deform and move with the propagation of water waves in the domain, resulting in the change of the total number of grids on each processor. Therefore, the load balancing and transfer of data between processors at each time step needs to be taken into account, which is a common factor complicating the parallelization of AMR programs (Antepara et al., 2015).



(a) (b) (c)

Figure 2: A typical example of the structure and grid indexes of the two-level adaptive mesh with $N = 3$: (a) coarse mesh, (b) fine mesh, and (c) active mesh determined from values of the indicator function ($I = 0$ for coarse and $I = 1$ for fine). The red arrows in (a) and (b) indicate the order of indexes. Different colors in (c) denote a 3-processor partition.

2.3 Repartition and load balancing

To balance the load, the mesh are partitioned with the Z-order kept in each part (processor) as the Z-order are designed to reduce the communications (Tirthapura et al., 2006). Thus we only focus on balancing the number of grids. Specifically, we first generate a weight list $\{w_i\}_{i=0}^{N^2-1}$ following the coarse index, with each coarse i counting 1 and fine i counting 4. For example, the mesh in figure 2(c) has a weight list $\{1, 4, 1, 4, 4, 4, 1, 4, 1\}$. A P-processor partition is nothing but cut the $\{w_i\}_{i=0}^{N^2-1}$ into P parts at $\{c_1, c_2, \cdots, c_{P-1}\}$ while keeping the sequence within each part. Ideally, the cutting positions are determined by minimizing the maximum summation of the weight in each processor:

$$\{c_1, c_2, \cdots, c_{P-1}\}^* = \min_{\{c_1, c_2, \cdots, c_{P-1}\}} \max \begin{bmatrix} \sum_{i=0}^{c_1-1} w_i \\ \sum_{i=c_1}^{c_2-1} w_i \\ \vdots \\ \sum_{i=c_{P-1}}^{N^2-1} w_i \end{bmatrix}. \tag{9}$$

However, the exact solution of (9) is time-consuming in one processor. A parallel computation, on the other hand, requires back and forth intro-processor communications, making it a bottleneck of the whole algorithm. In implementation, we approximate the solution of (9) by finding the multiples of the average cost:

$$c_k = j \quad \text{if} \quad \sum_{i=0}^{j-1} w_i < k\mu \quad \text{and} \quad \sum_{i=0}^{j} w_i > k\mu, \tag{10}$$

where $\mu = \lfloor \sum_{i=0}^{N^2-1} w_i / P \rfloor$. In this way, a processor, say k, can easily compute its potential cutting points with minimum intro-process information, specifically μ and $\sum_{i=0}^{c_{k-1}} w_i$ performed by MPI_SCAN(SUM) before searching. A demonstration of a three-processor partition are shown in figure 2(c).

After the partition of the mesh is done, the remaining task is to move data among processors based on the cutting points $\{c_k\}_{k=1}^{P-1}$ obtained from (10). In this process, the ghost cells for each part also need to be taken into account. To minimize the communication and improve efficiency, all the data sent from one processor to another is collected into an array, then all the arrays are moved to other processors using MPI_ISEND and MPI_IRECV.

2.4 Summary of solution procedures

In this section, we summarize our implementation of the parallel AMR algorithm. At the initial time $t = 0$, the states on coarse and fine grids (referred to as $\mathbf{u}_c$ and $\mathbf{u}_f$ respectively) are computed by (4), and Δt is calculated from the CFL condition (6). In each time step $t = t_n$ ($n = 0, 1, 2, ...$), the values of h in the ghost cells are first transferred from neighboring processors, then the indicator function I is obtained by (8) with ∇h calculated from a finite difference approximation. Next, the load balancing is performed by the method elaborated in §2.3. The data I, $\{\mathbf{u}_c^n\}_{I=0}$ and $\{\mathbf{u}_f^n\}_{I=1}$ on each processor are moved to corresponding processors according to the repartition of the mesh. Finally, the states on the active grids are updated by (7) and (5) in each processor. The procedures of calculating I, load balancing and updating states are then repeated until the end time t_{end}. The full process is summarized in algorithm 1.

Algorithm 1 Parallel AMR solver of 2D shallow-water equations

Input: spatial resolution N, number of processors P, end time t_{end}
Output: states in active grids $\{\mathbf{u}_c : I = 0\}$, $\{\mathbf{u}_f : I = 1\}$ at $t \le t_{end}$
Initialization initialize states $\{\mathbf{u}_{c,i}\}_{i=0}^{N^2-1}$ and $\{\mathbf{u}_{f,j}\}_{j=0}^{4N^2-1}$ by (4)
calculate Δt from (6)
$t \leftarrow 0$, $n \leftarrow 0$
while $t < t_{end}$ **do**
1. obtain h of the ghost cells from neighboring processors
2. calculate I by (8) on each processor
3. repartition the mesh with the updated I for load balancing
4. transfer I, $\{\mathbf{u}_c^n : I = 0\}$ and $\{\mathbf{u}_f^n : I = 1\}$ among the processors based on step 3
5. update states $\{\mathbf{u}_c^n : I = 0\} \to \{\mathbf{u}_c^{n+1} : I = 0\}$, $\{\mathbf{u}_f^n : I = 1\} \to \{\mathbf{u}_f^{n+1} : I = 1\}$ by (5) and (7)
6. $t \leftarrow t + \Delta t$

end while

3 Results

3.1 Numerical solutions

We run the simulation with a set of spatial resolutions N and processor number P. We have checked that solutions obtained with different N are in qualitative agreement with each other, and P has no influence on the solutions. Figure 3 shows the evolution of the water height field $h(x, y, t)$ and the active grids in the simulation with $N = 100$. We can see that at $t = 0$, the fine grids are concentrated around the center of the domain where the profile of h is steep. At later times, the water waves propagate towards the boundaries and are reflected back several times, leading to the change of the distribution of fine grids. Finally, the maximum $|\nabla h|$ throughout the domain becomes smaller than G due to the spreading of the water and the numerical damping, therefore no fine grid is active and the mesh becomes static and coarse in the entire domain.

The partitions at different t are shown in figure 4. We can see that in regions with more fine grids, the area occupied by one processor becomes smaller, and vice versa. In

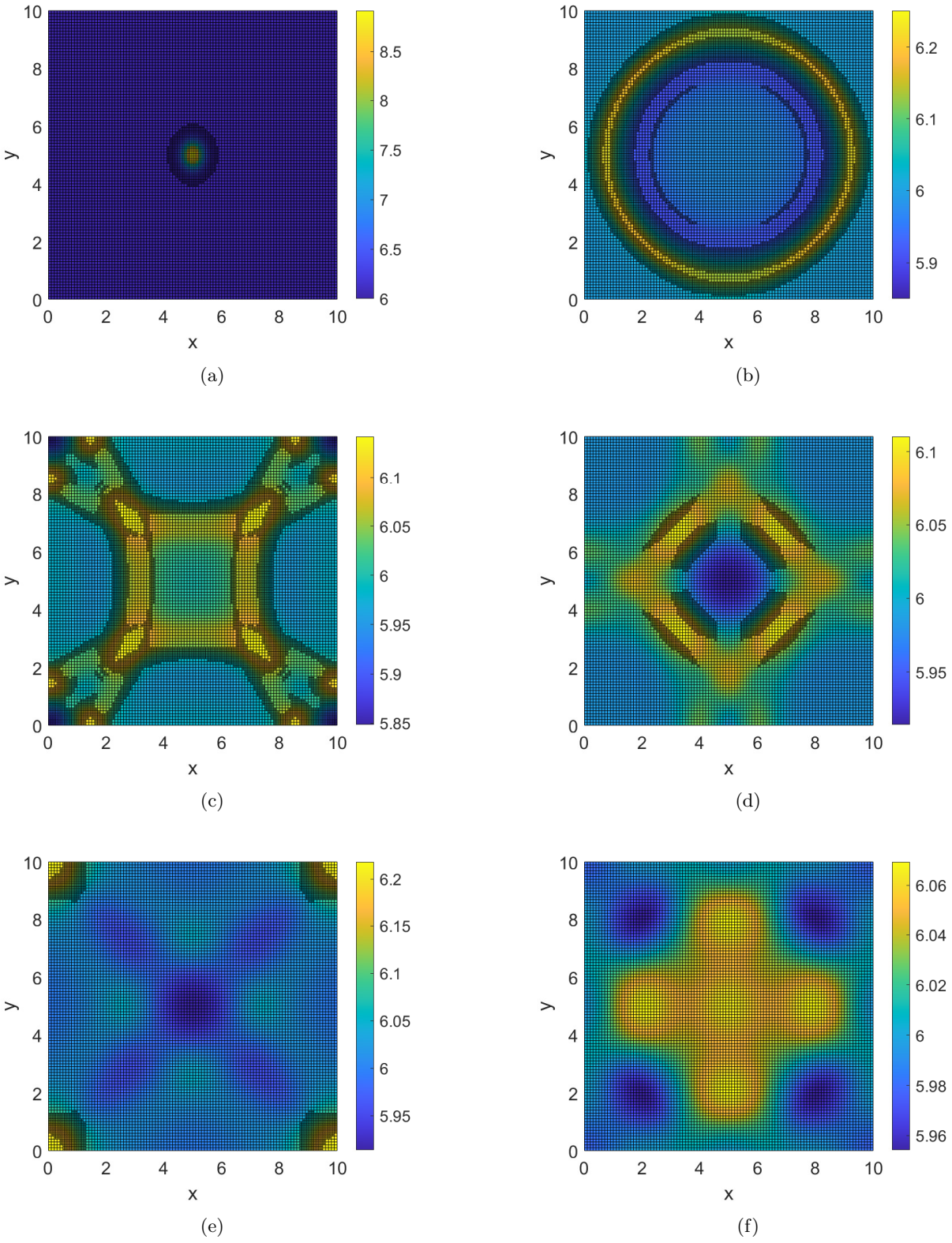


Figure 3: The water height $h(x, y)$ obtained with $N = 100$ at (a) $t = 0$, (b) $t = 0.5$s, (c) $t = 1$s, (d) $t = 1.5$s, (e) $t = 2$s, and (f) $t = 2.5$s.

general, such partition almost evenly allocates the active grids to all processors, which is consistent with our expectation.

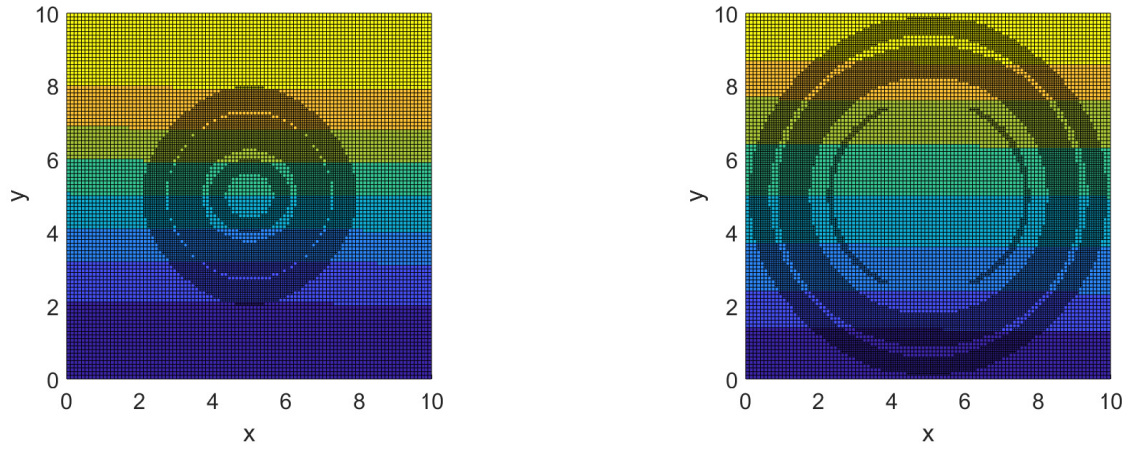


Figure 4: The partition of the mesh in a run with $N = 100$, $P = 8$ at (a) $t = 0.25$s and (b) $t = 0.5$s. Grids in different processors are marked by different colors.

3.2 Performance and scaling

We have performed a parallel scaling study on the Great Lakes Slurm HPC Cluster using one node for each simulation. Table 1 shows the elapsed time of cases with $t_{end} = 0.5$s for different combinations of N and P. We can see that in general the elapsed time increases with N and decreases with P. To obtain a clear view of the scaling performances, we calculate the efficiency of the program with respect to P and N respectively. For P, the efficiency is defined as

$$e_P = \frac{\tau(P = 4, N)}{\tau(P, N) \times (P/4)}, \tag{11}$$

where $\tau(P, N)$ is the elapsed wall time as a function of P and N. For N, we note that the theoretical computational cost scales as $O(N^3)$, estimated from the number of grids ($\sim O(N^2)$) and the number of time steps ($\sim O(N)$ due to the CFL condition (6)). Therefore, the efficiency for N is defined as

$$e_N = \frac{\tau(P, N = 400)}{\tau(P, N)/(N/400)^3}. \tag{12}$$

We summarize the efficiencies calculated from the timing results in tables 2 and 3. From table 2, we can see that for a given N, the efficiency e_P decreases with increasing P; for a given $P > 4$, e_P increases with increasing N. From table 3, we find that e_N shows no obvious dependency on N except for the cases with $P = 4$. For $P > 4$, it is clear that e_N decreases with increasing P. In general, the program shows better efficiency when N increases and P decreases.

Table 1: Elapsed time (seconds) for different runs.

P	4	8	16	32
$N = 400$	10.57	6.02	3.40	2.26
$N = 800$	96.99	49.11	29.47	20.38
$N = 1000$	195.89	97.66	56.71	39.83
$N = 2000$	1728.78	769.00	472.65	321.30

Table 2: Efficiency for different runs with $P = 4$ as baseline.

P	4	8	16	32
$N = 400$	1	0.878	0.777	0.585
$N = 800$	1	0.988	0.823	0.595
$N = 1000$	1	1.003	0.864	0.615
$N = 2000$	1	1.124	0.914	0.673

Table 3: Efficiency for different runs with $N = 400$ as baseline.

P	4	8	16	32
$N = 400$	1	1	1	1
$N = 800$	0.872	0.981	0.923	0.887
$N = 1000$	0.843	0.963	0.937	0.887
$N = 2000$	0.764	0.979	0.899	0.879

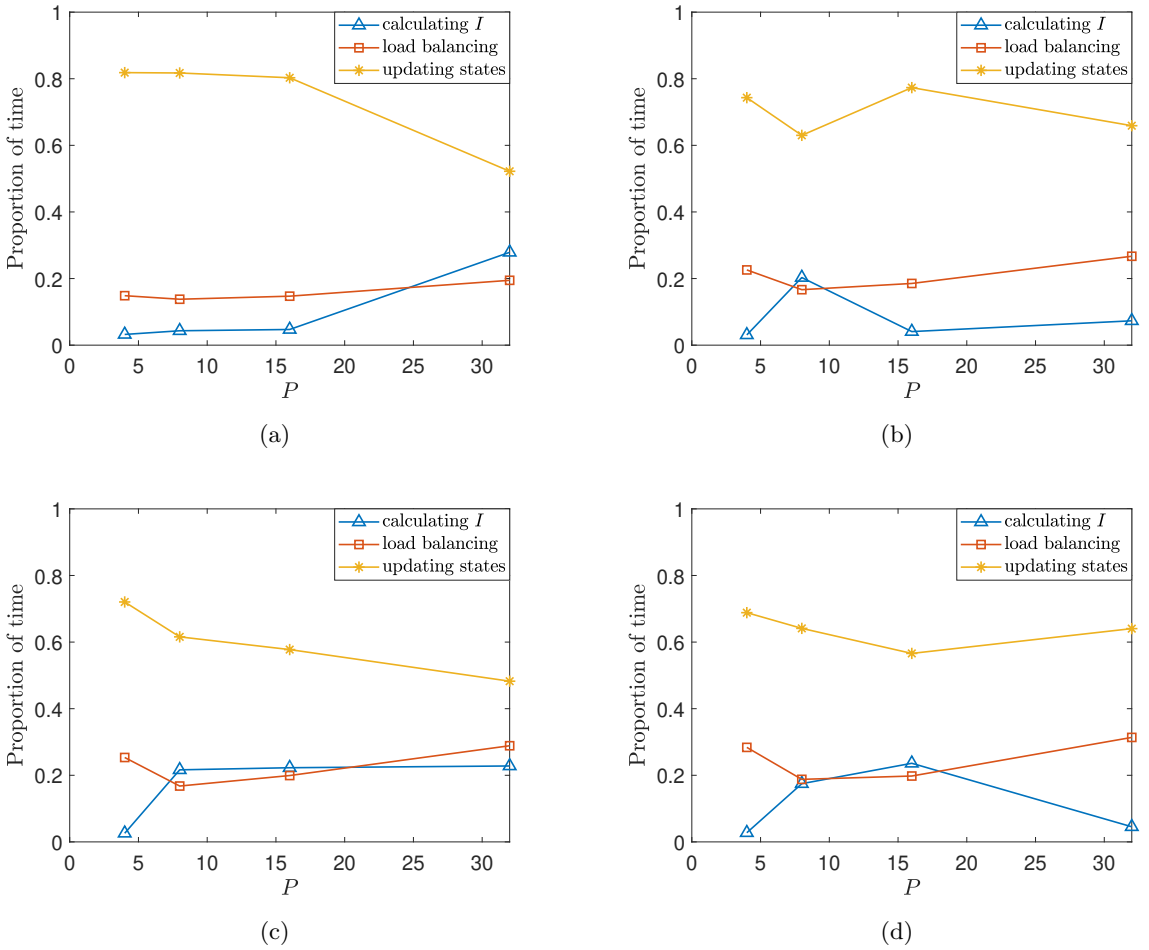


Figure 5: The proportion of elapsed time for each procedure in the solver with (a) $N = 400$, (b) $N = 800$, (c) $N = 1000$ and (d) $N = 2000$.

To investigate the factors leading to the above scaling behaviors, we record the elapsed time for each procedure: (i) calculating I, (ii) load balancing and (iii) updating states in the solver and compute their proportions of time. Figure 5 shows the results as functions of P with different N. We can observe that updating states is the most time-consuming procedure and takes more than 60% elapsed time in most cases. The proportions of calculating I and load balancing are close to each other (fluctuating around 20%). We can

see that the proportion of updating states generally decreases with increasing P, although there are oscillations in figures 5(b) and 5(d). Since the other two procedures both involve data transfer among processors which deteriorates the scaling of parallel programs, the increase of their proportions may account for the decrease of efficiency with increasing P. Further work needs to be performed for better scaling with higher number of CPU cores.

4 Conclusions

In this project, we develop a finite volume model with adaptive mesh that solves the 2D nonlinear shallow-water equations in parallel. The two-level AMR is developed using the gradient of water height as the indicator function to control the refinement or coarsening of the grids. For load balancing, the dynamic mesh is partitioned based on a Z-order space-filling curve to reduce the communication between processors. We conduct a set of numerical tests to validate our parallel AMR algorithm and evaluate the scaling performance of the solver. The numerical results show that the AMR and the partition of the mesh have been effectively implemented. The efficiency of the program is found to increase with the spatial resolution N and decrease with the number of processors P in general. The procedure of updating cell states is dominant (takes more than 60% of elapsed time) in most cases, exhibiting favorable parallel performance.

Future work of this study could involve several aspects: (i) Apply unstructured mesh to discretize the domain which may be suitable for more complex geometries. (ii) Implement more levels of refinement, which may require a new data structure (e.g. quadtree/octree) storing the grid information to reduce the occupied memory. (iii) Develop more efficient load balancing strategies (e.g. use other space-filling curves, compress/decompress the transferred data) to further reduce the communication among processors.

References

Ambrosi, D. (1995). Approximation of shallow water equations by roe's riemann solver. *International journal for numerical methods in fluids*, 20(2):157–168.

Antepara, O., Lehmkuhl, O., Borrell, R., Chiva, J., and Oliva, A. (2015). Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows. *Computers & Fluids*, 110:48–61.

Beisiegel, N., Vater, S., Behrens, J., and Dias, F. (2020). An adaptive discontinuous galerkin method for the simulation of hurricane storm surge. *Ocean Dynamics*, 70:641–666.

Roe, P. L. (1981). Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372.

Tirthapura, S., Seal, S., and Aluru, S. (2006). A formal analysis of space filling curves for parallel domain decomposition. In *2006 International Conference on Parallel Processing (ICPP'06)*, pages 505–512. IEEE.