Acta Oceanol. Sin., 2022, Vol. 41, No. 9, P. 158–168
https://doi.org/10.1007/s13131-022-2004-6
http://www.aosocean.com
E-mail: ocean2@hyxb.org.cn

A GPU accelerated Boussinesq-type model for coastal waves

Kezhao Fang[1, 2], Jiawen Sun[1, 2*], Guangchun Song[1], Gang Wang[3], Hao Wu[1, 2], Zhongbo Liu[4]

[1] State Key Laboratory of Coastal and Offshore Engineering, Dalian University of Technology, Dalian 116024, China

[2] National Marine Environmental Monitoring Center, Dalian 116023, China

[3] Marine Geological Resources Survey Center of Hebei Province, Qinhuangdao 066001, China

[4] College of Transportation Engineering, Dalian Maritime University, Dalian 116026, China

Received 2 September 2021; accepted 14 February 2022

Abstract

This study presents an efficient Boussinesq-type wave model accelerated by a single Graphics Processing Unit (GPU). The model uses the hybrid finite volume and finite difference method to solve weakly dispersive and nonlinear Boussinesq equations in the horizontal plane, enabling the model to have the shock-capturing ability to deal with breaking waves and moving shoreline properly. The code is written in CUDA C. To achieve better performance, the model uses cyclic reduction technique to solve massive tridiagonal linear systems and overlapped tiling/shared memory to reduce global memory access and enhance data reuse. Four numerical tests are conducted to validate the GPU implementation. The performance of the GPU model is evaluated by running a series of numerical simulations on two GPU platforms with different hardware configurations. Compared with the CPU version, the maximum speedup ratios for single-precision and double-precision calculations are 55.56 and 32.57, respectively.

Key words: Boussinesq model, GPU, speedup ratio, coastal waves

Citation: Fang Kezhao, Sun Jiawen, Song Guangchun, Wang Gang, Wu Hao, Liu Zhongbo. 2022. A GPU accelerated Boussinesq-type model for coastal waves. Acta Oceanologica Sinica, 41(9): 158–168, doi: 10.1007/s13131-022-2004-6

1 Introduction

Accurate prediction of wave propagation and transformation in the coastal region is crucial for scientists and engineers. Phase-resolving simulation is desirable since it can provide a comprehensive picture of intra-wave characteristics under consideration compared with phase-averaged type. The growth in development and application of Boussinesq-type (BT) models has been explosive since the 1990s with the aid of more powerful computers (Kirby, 2016). In addition to modeling coastal waves and nearshore currents, BT models have been used to simulate coastal sediment transport (Klonaris et al., 2018), ship-generated waves (Shi et al., 2018), landslide tsunamis (Fang et al., 2020). Comprehensive reviews on BT theories and model applications are made recently in literature (e.g., Brocchini, 2013; Kirby, 2016, 2017; Liu et al., 2018 and many others).

BT simulations require a large number of discrete grids to obtain detailed intra-wave hydrodynamics and high resolution of coastal structures, depending on the size of the computational domain and wave conditions. The time-marching step is also restricted (in the order of $O(T/100)$, where T is the typical wave period) to meet the stability requirement from the Courant-Friedrichs-Lewy condition. BT simulation is thus computationally demanding at fine resolutions in practical applications. On top of that, the model tends to collapse due to the difficulties in numerically treating high-order derivatives (Liu et al., 2019), rapid changes in bathymetry, wave breaking, and moving shoreline (Shi et al., 2012), and even the intrinsic instability of equations (Madsen and Fuhrman, 2020), which further increases the computational burden. Two primary efforts have been devoted to overcoming this constraint: efficient numerical techniques and adaptation to parallelization computation. Realizing the conservative and substantial stability merit of the finite volume (FV) method, many studies proposed hybrid FV and finite difference (FD) solvers to resolve BT equations in the past decade (Erduran et al., 2005; Shi et al., 2012; Fang et al., 2013; Kim et al., 2009b). Remarkable improvement of model stability is achieved at the cost of increased computational time in constructing variables and flux evaluation between cell interfaces.

On the other hand, parallel computing has become an essential requirement for studying the increasingly complex problems of coastal hydrodynamics with large spatial scales and high temporal resolution. The Message Passing Interface has been commonly utilized to gain better performance of BT models, e.g., paralleled version of FUNWAVE-TVD (Shi et al., 2012) and COULWAVE (Lynett, 2002). These accelerated models can handle a large-scale computation of the order 10 km×10 km, but a large number of CPU cores in High-Performance-Computing (HPC) clusters may be required to fulfill the job (Yuan et al., 2020). The cost of using multiple CPUs is high due to hardware investment and related use. Alternatively, the Graphics Processing Unit (GPU) technology is specially designed for massive computation characterized by high throughput and low latency and offers the performance of smaller clusters with less disbursement. Thanks to the development of GPU hardware and the release of general-purpose programming languages (e.g., CUDA C, CUDA Fortran, and OpenACC), the successful implementation of GPU emerges

Foundation item: The National Key Research and Development Program under contract No. 2019YFC1407700; the National Natural Science Foundation of China under contract Nos 51779022, 52071057 and 51809053.
*Corresponding author, E-mail: jwsun@nmemc.org.cn

across a wide range of areas in the recent decade, among which GPU-accelerating coastal hydrodynamics simulation is gaining much attention.

However, the GPU implementation of BT models is rarely reported in the literature until very recently. Tavakkol and Lynett (2017) presented a single-GPU-accelerated BT model named "Celeris" using C++ and Direct3D libraries. The software has the ability for real-time, interactive simulation and visualization. This model is further developed into "Celeris Base" utilizing the Unity3D game engine and C# language, with the new features of 360-degree video capturing, geographic map overlays, built-in real-time gauge plotters (Tavakkol and Lynett, 2020). Kim et al. (2018) successfully ported their BT model to a single GPU via CUDA Fortran. They found that the GPU model showed about 20 times faster computational time compared with the CPU-based code. Yuan et al. (2020) lately described the multiple-GPU acceleration of FUNWAVE-TVD also via CUDA Fortran. Efficiency evaluation shows that, compared with the CPU version running at a 36-core HPC node, speedup ratios of 4–7 and above 10 can be observed for single-GPU and double-GPU runs, respectively.

The specific objective of the present work is to document a GPU accelerated BT model and evaluate its simulation accuracy and computation efficiency. The paper is organized as follows: Section 2 briefly describes the Boussinesq model; Section 3 presents the GPU implementation; four test cases, including a challenging application of the model in a specific field site (Jinmeng Bay, China), are included in Section 4 to demonstrate the model's capabilities, showing the accuracy and performance of the BT model in different configurations and architectures.

2 Boussinesq model

The GPU implementation is based on our previous work (Fang et al., 2013), where Boussinesq equations are solved using the hybrid FV/FD method on a rectangular grid and programmed in serial mode with Fortran. This section briefly describes the governing equations, numerical scheme and boundary conditions.

2.1 *Governing equations*

The two-dimensional Boussinesq equations with weakly dispersion and nonlinearity for rapidly varying bathymetry (Kim et al., 2009a) are written in the conservative form as

$$\boldsymbol{U}_t + \boldsymbol{F}(\boldsymbol{U})_x + \boldsymbol{G}(\boldsymbol{U})_y = \boldsymbol{S}(\boldsymbol{U}), \tag{1}$$

in which $\boldsymbol{U}$ is vector variable, $\boldsymbol{F}$ and $\boldsymbol{G}$ are vector fluxes along x and y direction, respectively, which can be expressed as

$$\boldsymbol{U} = \begin{pmatrix} \eta \\ U \\ V \end{pmatrix}, \boldsymbol{F} = \begin{pmatrix} P \\ P^2/d + g(\zeta^2 - z_b^2)/2 \\ PQ/d \end{pmatrix},$$
$$\boldsymbol{G} = \begin{pmatrix} Q \\ PQ/d \\ Q^2/d + g(\zeta^2 - z_b^2)/2 \end{pmatrix}, \tag{2}$$

with

$$U = P - \left(B + \frac{1}{3}\right) h^2 P_{xx} - hh_x \frac{1}{3} P_x - hh_{xx} \frac{1}{6} P + \frac{1}{3} h_x^2 P -$$
$$\left(B + \frac{1}{3}\right) h^2 Q_{xy} - hh_x \frac{1}{6} Q_y - hh_y \frac{1}{6} Q_x - hh_{xy} \frac{1}{6} Q + \frac{1}{3} h_x h_y Q, \tag{3}$$

$$V = Q - \left(B + \frac{1}{3}\right) h^2 Q_{yy} - hh_y \frac{1}{3} Q_y - hh_{yy} \frac{1}{6} Q + \frac{1}{3} h_y^2 Q -$$
$$\left(B + \frac{1}{3}\right) h^2 P_{xy} - hh_y \frac{1}{6} P_x - hh_x \left(\frac{1}{6} P_y\right) -$$
$$hh_{xy} \left(\frac{1}{6} P\right) + \frac{1}{3} h_x h_y P, \tag{4}$$

where the subscripts x, y, t denote the partial derivatives with respect to x, y and time; η is the surface elevation, $d(=h+\eta)$ is the total water depth, h is the still water depth; ζ is water level and z_b is bed elevation; $P=du$ and $Q=dv$ are the velocity flux components in x and y directions, respectively; $B=1/15$ is the dispersion parameter; $\boldsymbol{S}$ in Eq. (1) is the source vector and grouped into three parts, namely, bottom slope ($\boldsymbol{S}_b$), bottom friction ($\boldsymbol{S}_f$) and dispersive terms ($\boldsymbol{S}_d$)

$$\boldsymbol{S}(U) = \boldsymbol{S}_b + \boldsymbol{S}_f + \boldsymbol{S}_d = \begin{pmatrix} 0 \\ -gdz_{bx} \\ -gdz_{by} \end{pmatrix} + \begin{pmatrix} 0 \\ \tau_x \\ \tau_y \end{pmatrix} + \begin{pmatrix} 0 \\ \psi_x \\ \psi_y \end{pmatrix}, \tag{5}$$

with ψ_x and ψ_y rearranged into

$$\psi_x = Bgh^3(\eta_{xxx} + \eta_{xyy}) + Bgh^2(\eta_{yy} + 2\eta_{xx} + h_y\eta_{xy} + h_{xx}\eta_x + h_{xy}\eta_y), \tag{6}$$

$$\psi_y = Bgh^3(\eta_{yyy} + \eta_{xxy}) + Bgh^2(\eta_{xx} + 2\eta_{yy} + h_{xy}\eta_{xy} + h_{yy}\eta_y + h_{xy}\eta_x), \tag{7}$$

$$\tau_x = -fu\sqrt{(u^2 + v^2)}, \tau_y = -fv\sqrt{(u^2 + v^2)}, \tag{8}$$

where f in Eq. (8) is the bottom friction coefficient.

It is worth noting that (1) retaining higher-order bottom slope terms in equations helps to improve model performance for rapidly varying bathymetry (Kim et al., 2009a); otherwise, the equations revert to Madsen and Sørensen's equations (Madsen and Sørensen, 1992); (2) the flux term ($\boldsymbol{F}$ and $\boldsymbol{G}$) and the bed slope term ($\boldsymbol{S}_b$) are expressed by ζ and z_b to facilitate the implementation of a well-balanced scheme (Fang et al., 2016) to handle changing seabed and moving shoreline correctly.

2.2 *Numerical scheme*

The implementation of the hybrid FV/FD shock-capturing scheme primarily follows our previous work (Fang et al., 2013, 2016). The main procedures are summarized herein for clarity and completeness. A Godunov-type finite volume method is used on a rectangular cell system to deal with the convective parts while the rest terms are discretized using the finite difference method. The fourth-order Monotone Upstream-Centred Schemes for Conservation Laws construction and central upwind scheme are used for flux calculation. The third-order Runge-Kutta scheme with the Strong Stability Preserving property and the adaptive time step is used for time marching. For numerical stability requirement, the time increment must be restricted by the Courant number v. The velocity can then be obtained by solving the tridiagonal system, resulting from discretizing Eqs (3) and (4) using the second-order finite difference formula.

2.3 *Boundary conditions*

A minim value of water depth (d_{min}) is specified to distinguish the dry ($d \leqslant d_{min}$) and wet cell ($d > d_{min}$). The hydrostatic construction technique (Wang et al., 2011) is then used during variable construction to capture the moving shoreline in an accurate and efficient manner. The model captures breaking wave

as a shock by deactivating dispersive terms once local wave height exceeds 0.8 h or wavefront slope is steep than 30°. For the reflective wall, tangential velocity and normal velocity on ghost cells are determined from the inner domain by imposing symmetric and anti-symmetric conditions with respect to the solid wall. The incident waves are generated internally in the computation domain by adding the source function to the mass equations. A sponger layer is also placed at the front of solid walls to absorb wave energy once necessary (Kirby et al., 1998).

3 Implementation on GPU

CUDA includes an extension of the C language and facilitates the programming on GPUs for general purpose applications by preventing the programmer from dealing with low-level language programming on GPU. CUDA C is thus chosen to implement the aforementioned numerical scheme.

3.1 *Algorithm overview*

It is essential to know the working mechanism of GPU by means of CUDA from two aspects. The first is regarding the software architecture. The fundamental element to be processed is the thread, a great number of threads is organized into a block, and any group of blocks is called a grid. The second aspect is the hardware architecture. The minimum unit is the Streaming Processor (SP, also known as CUDA core), where a single thread is executed. Block is assigned to an streaming multiprocessor (SM). GPU executes groups of threads within a block known as warps in single instruction multiple thread (SIMT) fashions. These threads access consecutive memory locations and execute the same computation instructions.

The core idea of GPU simulation is to put a large amount of computation load on the GPU (device) and a small amount of computation on the CPU (host). The code flowchart is depicted in Fig. 1. After reading the input data and initialization, the simulation starts and runs until the required simulation time is reached. The calculation information is written out at specified intervals during the time loop. The functions of reading input data, initialization, and writing output are implemented as host code (CPU subroutines), running on CPU in sequential mode. The intense computation such as fluxes evaluation, source term calculation, and time integration is written as device code (GPU kernels), running on GPU in parallel. Once called, a kernel creates a given number of threads and blocks in the GPU, and the threads execute the same part in the kernel function concurrently. The information exchange between host and device is undertaken by calling the intrinsic memory copy functions in CUDA C, Memcpy host to device, and Memcpy device to host.

In the present study, each thread corresponds to one cell used to discretize the computational domain, and each block consists of $K\times K$ cells. Since the computational domain is discretized by means of a uniform rectangle grid, the index of threads and blocks, as well as their neighboring information, is easily retrieved thanks to the correspondence between the physical position of a cell/block and its position on the two-dimensional array where data concerning that cell/block are stored in the memory.

3.2 *CUDA optimization*

Kernels usually need variable values from neighboring cells (or a stencil) to accomplish the computation. For example, evaluating the third derivative η_{xxx} at the ith cell in Eq. (6) needs a stencil of size five (from the cell $(i-2, j)$ to $(i+2, j)$, where i and j are the cell index along x and y direction). In this way, each cell value in the discretized domain will be accessed five times by the kernel (if η is declared as a global variable and stored in global memory). This strategy will bring heavy memory access and a low degree of data reuse. Overlapped tiling is a good technique to reduce the data-sharing requirements for stencil computations by introducing redundant computing CUDA blocks and using shared memory. Figure 2a shows a simplified modeling domain mapped into two-dimensional blocks with a dimension of 8×8. The traditional implementation calculates the variables of each cell in a block. Assuming that the value of a cell is updated with four adjacent cells, the edge cells in a block (not the gray cells) have to use the variables of the cell in the other block. Compared with the use of shared memory, the use of global memory has more delays. Only gray cells can be computed if shared memory is introduced

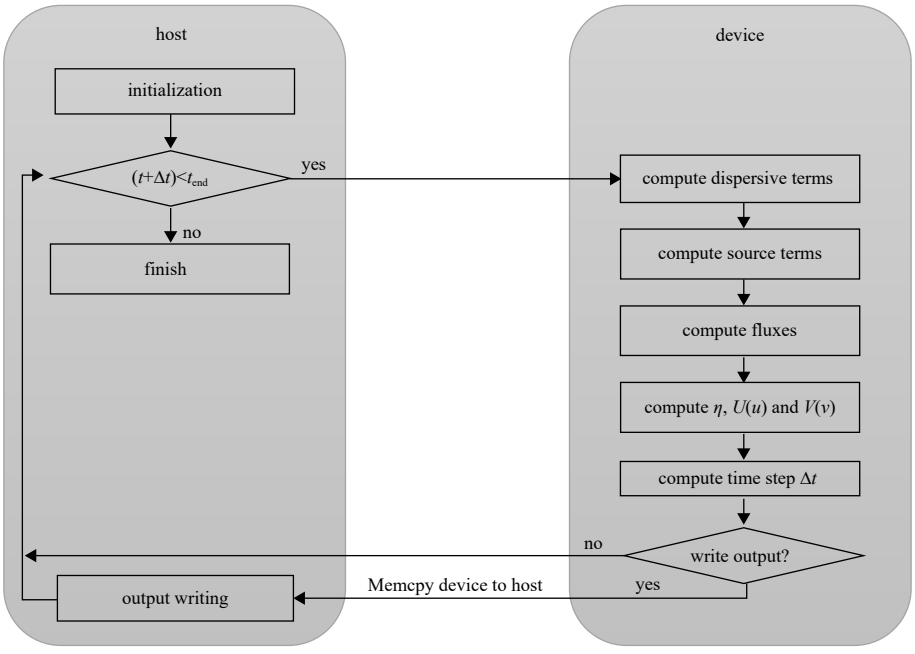

Fig. 1. Flowchart for GPU implementation of the Boussinesq-type model.

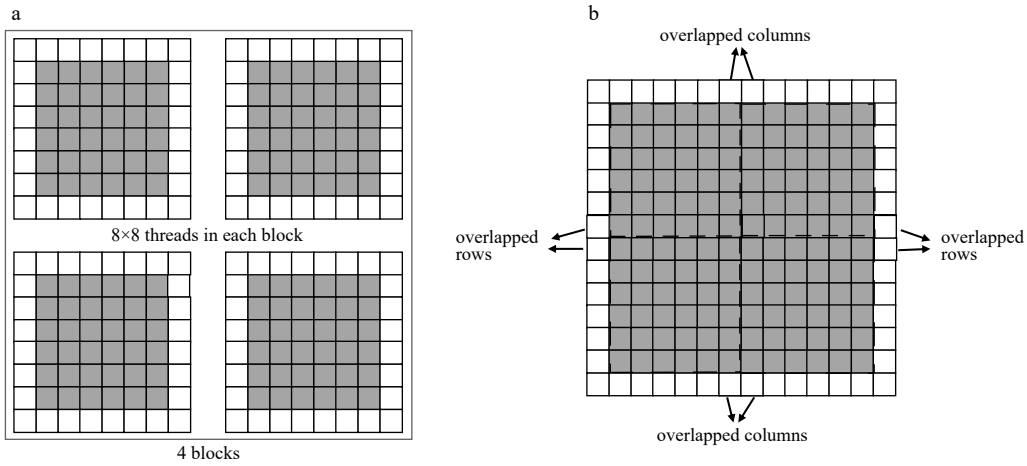

Fig. 2. A simplified modeling domain combined with 4 blocks that is not overlapped (a) and the overlapped tiling of blocks (b).

in the stencil problem because all the resident threads within a block share the fast-retrieving shared memory. In order to complete the calculation of all the cells in the modeling domain, Fig. 2b shows the overlapped tiling of blocks. Only the rows and columns of the edges (no-gray cells) overlap and 6×6 inner cells (gray cells) can be calculated, mapping the entire modeling domain within one block. Because of the overlapped tiling, the number of computing blocks in the grid would increase in the CUDA kernel. While the index of cells in every tile along the x-direction (x_{local}) ranges from 1–8, its global index increases by x_{local}+ (i–1)×(BlockDim-2). The number of blocks needed for whole modeling domain should be [m/(BlockDim-2)]+1 and [n/(BlockDim-2)]+1.

In the CPU version of the model (Fang et al., 2013), the tridiagonal system is solved by using Thomas Algorithm, which involves a forward elimination phase and a backward substitution phase. The algorithm is simple but inherently serial. The Cyclic Reduction (CR) algorithm (Zhang et al., 2010) is used herein to solve massive tridiagonal systems. CR also consists of two phases, forward reduction, and backward substitution. The forward reduction phase successively reduces a system to a smaller system with half the number of unknowns until a system of 2 unknowns is reached. The backward substitution phase successively determines the other half of the unknowns using the previously solved values. CR algorithm has more computation steps than Thomas algorithm, but is suitable for parallel calculation on GPU.

4 Numerical results and computation efficiency

Numerical simulations for four test cases, Cases 1–4 hereafter, are carried out on two GPU platforms with different configurations. The numerical results are compared with analytical solutions and experimental data to verify correct coding and prediction accuracy. This section discusses the computation efficiency of GPU implementation in detail.

4.1 *Case 1: exact solitary wave propagation over constant water depth*

Case 1 is designed to simulate a solitary wave that travels over a long distance. It provides a good test of the stability and conservative properties of the basic numerical scheme as any improper numerical treatment tends to upset the strict balance between nonlinearity and dispersion. The analytical solution of the solitary wave to Eqs (1)–(3) (Orszaghova et al., 2012) will be used as a reference in the simulation.

A highly nonlinear solitary wave of amplitude H=0.6 m propagating over a constant water depth of h=1.0 m has been calculated. The wave flume is 500 m long and discretized into finite cells with equal increment Δx=0.10 m. Figure 3 shows the initial wave profile (centered at x=60.0 m) and the subsequent evolu-

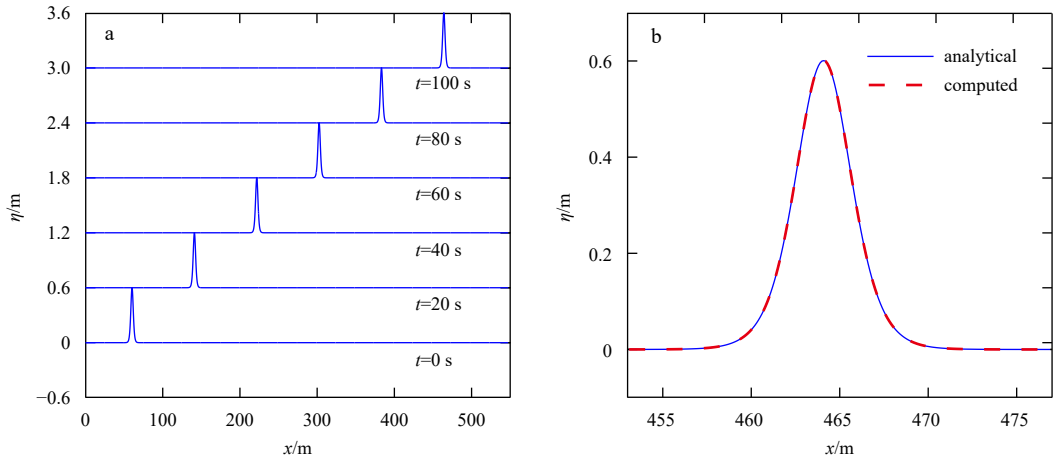

Fig. 3. Snapshots of modelled water surface for exact solitary wave propagation from 0 s to 100 s with an interval of 20 s (a); comparison between modelled (thick dashed line) and exact (thin solid line) surface profiles at t=100 s (b).

tion stage. The exact and computed surface profiles after long propagation (t=100 s) are further compared in Fig. 3. As seen from these figures, the solitary wave propagates with constant speed without any form change, confirming that the governing equations have been correctly discretized and accurately time marched.

4.2 *Case 2: regular wave propagation over an elliptical shoal*

The laboratory experiment of regular wave propagation over an elliptical shoal resting on a plane slope (Berkhoff et al., 1982) has been widely chosen to verify BT models. The bottom topography and eight measurement transects are plotted in Fig. 4. The working water depth for the considered case is 0.45 m. A monochromatic wave with a period of 1.0 s and an amplitude of 0.023 2 m is generated by a wavemaker located at x=−10 m and propagates in the entire domain. Two vertical sidewalls are located at y=−10 m and y=10 m.

The computational domain is the same as in Fig. 4 except for two sponge layers with a width of 2 m and 3 m are placed respectively behind the wavemaker and on the end of the beach to absorb wave energy. The extended domain is discretized into cells of Δx=0.05 m and Δy=0.10 m. The Courant number is set to be υ=0.35. The computed wave height at eight measurement transects is compared against the experimental data in Fig. 5. The agreements are generally good, demonstrating that the present numerical model can accurately simulate the combined effects of wave shoaling, refraction, diffraction, and reflection, as involved in this test.

The numerical results from CPU and GPU simulation are virtually identical. Furthermore, the computed wave height obtained with the GPU and CPU implementations were statistically compared in terms of the normalized Root Mean Square Deviation (RMSD) (in %):

$$\mathrm{RMSD} = \frac{1}{Y_{\max} - Y_{\min}} \sqrt{\frac{\sum_{i=1}^{N} (X_i - Y_i)}{N}} \times 100, \tag{9}$$

where X_i and Y_i are the values computed with CPU and GPU code respectively, N is the cell numbers. The average value of RMSD= 1.2% was found taking into account the wave height along 8 transects in Fig. 5, supporting the good resemblance between series suggested by the visual comparison. Only GPU simulation results will be presented hereafter.

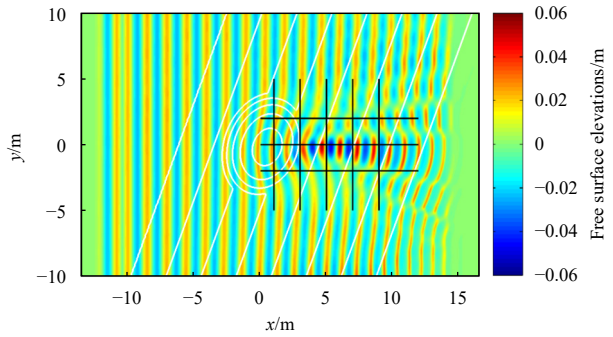

Fig. 4. The experimental setup (Berkhoff et al., 1982) and simulated free surface elevations at the final stage.

4.3 *Case 3: solitary wave propagation over a three-dimensional reef with a conical island*

Lynett et al. (2010) conducted a series of laboratory experiments for solitary wave transformation over a three-dimensional reef with a conical island in a large wave basin at Oregon State University's O.H. Hinsdale Wave Research Laboratory. This test is much more stringent than previous tests as wave breaking and time-dependent wet-drying interface are involved. The sketch of the wave basin and measurement location for the experiment is shown in Fig. 6. The basin is 48.8 m long and 26.5 m wide. The solitary incident wave has a dimensionless wave height of 0.39 m/ 0.78 m=0.5, a strongly nonlinear wave condition.

The computation domain is discretized into finite cells using cell size Δx=Δy=0.10 m, and the bottom friction coefficient f is set to be 0.005. A small Courant number (υ=0.15) is used in the simulations to properly capture wave breaking and moving shoreline. A series of the snapshot of computed free surface elevations are shown in Fig. 7. At t=5.0 s, the incident wave begins to feel the triangular reef flat, and slight wave spilling appears. The apparent wave runup and overtopping on the conical island are seen at the subsequent three snapshots (t=6.6 s, 8.6 s and 10.0 s). Breaking waves, captured by the model as the bore, spread across the entire flat and propagate towards the shoreline. At the next three moments (t=12.4 s, 14.4 s and 15.4 s), the flow transitions into a surge moving up the initially dry beach and inundates the top of the reef model.

Figure 8 presented the time series of calculated and measured free surface elevations at nine wave gauge locations. For gauges located in the offshore region (gauges 1 and 4), model results and data are in good agreement, indicating that the shoreward propagation and its reflection from the shore are well captured. The model also captures the steepening of the solitary wave over the reef apex at gauge 2. The model accurately predicts the superposition of the refracted and diffracted waves behind the cone island (gauge 3) but overestimates the peak of runup. At gauges 5–9, the model/data comparison is reasonably good, showing the formation and movement of the hydraulic jump correctly. The flow velocity is also available from the experiments at some acoustic Doppler velocimetry (ADV) locations. The comparison between the computed and measured velocities in the basin is shown in Fig. 9, again in good agreement.

4.4 *Case 4: application of the model to field site Jinmeng Bay*

The last test case is intended to evaluate the performance and applicability of the GPU model in a realistic configuration. Jinmeng Bay is located on the western coast of Bohai Sea. It is about 9 km long and aligned in an approximately north-south orientation (Fig. 10). There are two artificial islands, i.e., Lianhua Island and Hailuo Island, with complicated shapes, in the offshore region. The existence of Tang he River, Qinhuangdao Port, and marina further increase the complexity of the coast. The numerical simulation is thus a challenging task for the study site.

The computational domain is 2.7 km in cross-shore direction and 6.1 km in longshore direction, covering the northern part of Jinmeng Bay. Uniform rectangular cells with the size of Δx=1.325 m and Δy=3.0 m are used to discretize the computational domain, resulting in 4 194 304 cells in all. The simulation neglects the bottom friction, involves moving shoreline ($d_{\min}$=0.02 m) and wave breaking. Random wave with directional JONSWAP spectrum (significant wave height H_{si}=2.14 m, peak period T_p=7.06 s, peak factor γ=3.3) normally incident the computational domain. The total simulation time is 1 201 s to ensure the wavefield can reach a steady state. The Courant number is set to be 0.15.

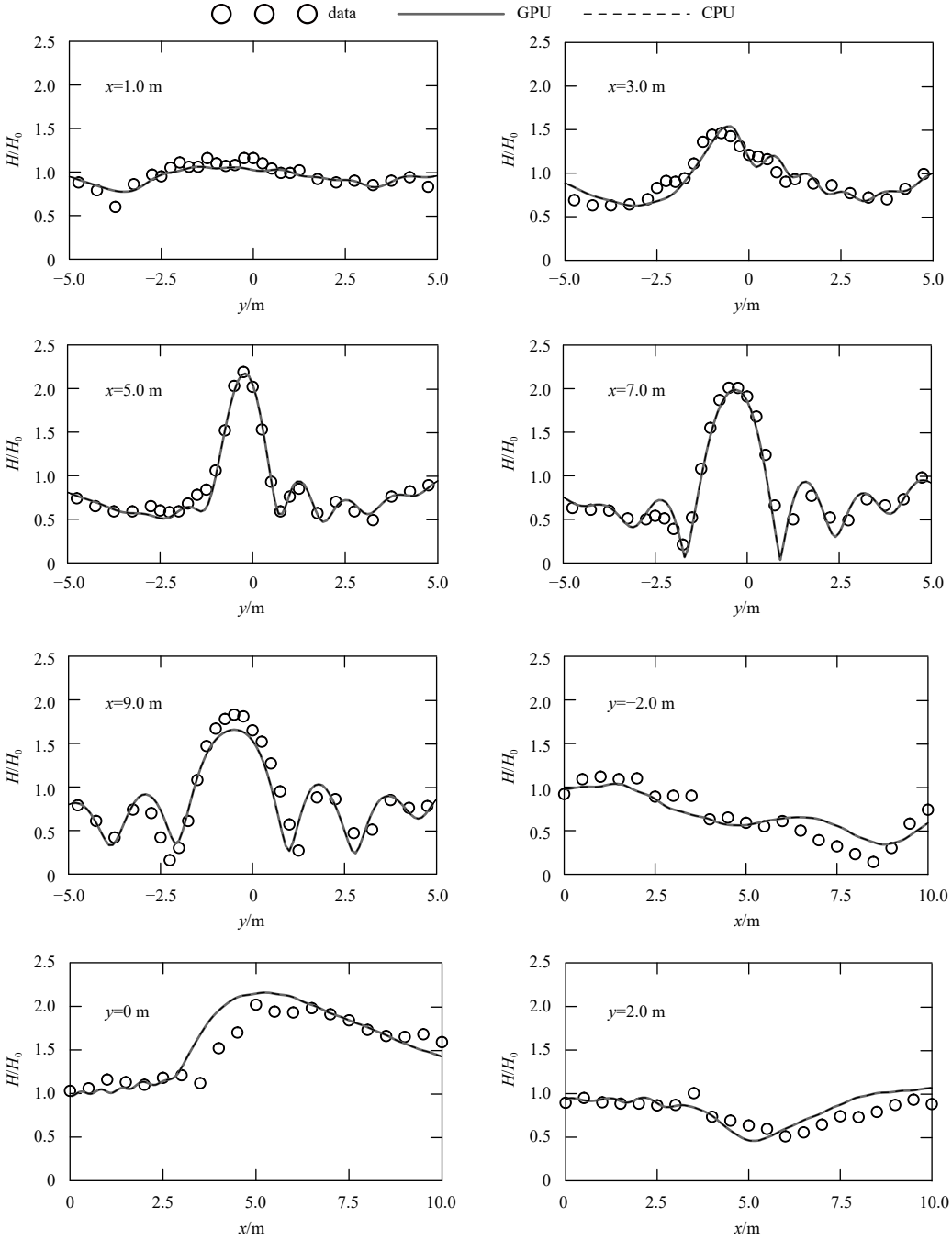

Fig. 5. Comparison of the normalized wave height between experimental data (circles) and computed results from GPU (solid lines) and CPU (dashed lines) simulation.

Figure 11 shows the snapshot of free surface elevation at t= 1 200 s and the wave height field. A uniform distribution of the numerical results is observed in the upper region (y>3 500 m). However, the wave pattern in the rest becomes complex due to the existence of offshore and coastal structures. The standing wave pattern due to the reflection from vertical structures is found adjacent to the offshore boundary. The incident waves can only penetrate the nearshore region via the gaps among islands and ports. Hailuo Island has a better shielding effect than Lianhua Island since the former is emergent while the latter is submerged under the water. Breaking events occur during waves enter into the shallow water on the outer boundary of Lianhua Island and the nearby shoreline. The moving shoreline is observed after carefully checking the simulation results (not shown in the figure). All these phenomena are reasonably captured by the model. The speedup ratio for this particular case is 55.56 (=482 736.19/8 688.34). Therefore, the present simulations have demonstrated the accuracy and efficiency of the GPU model.

4.5 *Computation efficiency*

Those above three two-dimensional horizontal simulations, Cases 2–4, are used to demonstrate the computation efficiency of GPU implementation in comparison to its CPU version. Numerical simulations were run on two GPU platforms with different hardware configurations, as listed in Table 1. Platform I is equipped with dual Intel Xeon Gold 6226R CPUs (16 cores,

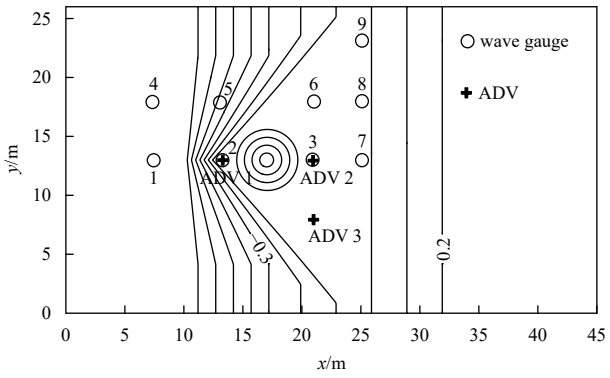

Fig. 6. Bathymetry contours (in meters with 0.1 m interval) and measurement locations used in model simulations for OSU tank bathymetry (Lynett et al., 2010).

2.90 GHz) and a Geforce RTX 3090 GPU that provides 10 496 streaming processor cores, 24 GB of global device memory (GDDR6X), and 936.2 GB/s bandwidth. Platform II is equipped with dual Intel Xeon Gold 6130 CPUs (16 cores, 2.10 GHz) and a Tesla V100S GPU that provides 5 120 streaming processor cores, 32 GB of global device memory (HBM2), and 1 133 GB/s bandwidth. Regarding the software environment, the operation systems are CentOS 8.0 and Ubuntu 16.04 for Platform I and Platform II respectively. GUN GCC Compile 8.4.1 and CUDA toolbox Toolkit 11.4 are installed on Platform I while GUN GCC Compile 5.4.0 and CUDA toolbox Toolkit 10.2 are installed on Platform II.

Both single and double precision simulations were conducted. Note that the cell number along the x and y direction is set to the power of two to achieve the CR algorithm's best performance and avoid the inactive warps. The computation efficiency is generally evaluated by the speedup ratio, defined as the ratio of the computational time for the CPU model and GPU model. To avoid the possible interference from the soft and hard configurations, each case were simulated 6 times and the average value was used as the final execution time.

Figure 12 illustrates the performance of GPU code with different mesh grid sizes for Case 2 and Case 3. The execution time from CPU and GPU simulations is also summarized in Table 2, where the results from double-precision computation are listed in the brackets. Overall, the speedup ratio increases with the increase of the grid size, which is in consistent with previous study of Yuan et al. (2020). For single-precision simulations, the speedup ratio increases from 22.88 (512×256) to 48.38 (2 048×512) on Platform I and from 23.97 (512×256) to 48.34 (2 048×512) on Platform II for Case 2. For Case 3, the speedup ratio increases from 16.15 (512×256) to 25 (1 024×512) on Platform I and from 16.40 (512×256) to 23.85 (1 024×256) on Platform II. The GPU computation lies a certain overhead, mainly due to memory transfers and kernel launch latency. The increase of the grid size actives more threads/blocks in SM, the corresponding higher workload covers the overhead and thus better performance is achieved. This demonstrate the merit of GPU's massive parallelism for high throughout modelling task. We further explore this issue by profiling the GPU kernel of solving the tridiagonal equations (the most time-consuming part in our model) for Case 2 using CUDA *nvprof* tool. The profiling indeed shows a clear increase of occupancy (the number of active warps on that scheduler every clock cycle) with the increase of grid size, i.e., 5.76%, 9.39%, 11.32%, 18.52%, 22.44% for grid size 512×256, 1 024×256, 1 024×512, 2 048×512 and 2 048×1 024.

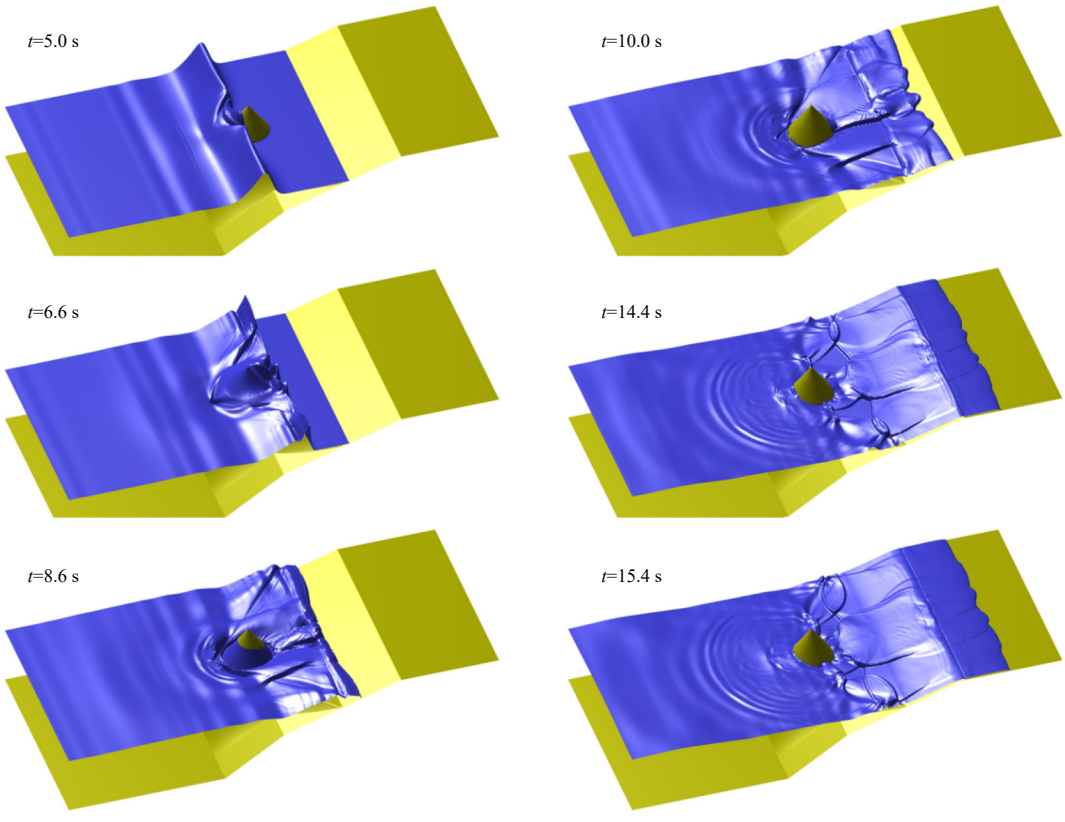

Fig. 7. Snapshots of modelled water surface for solitary wave transformation over a three-dimensional reef.

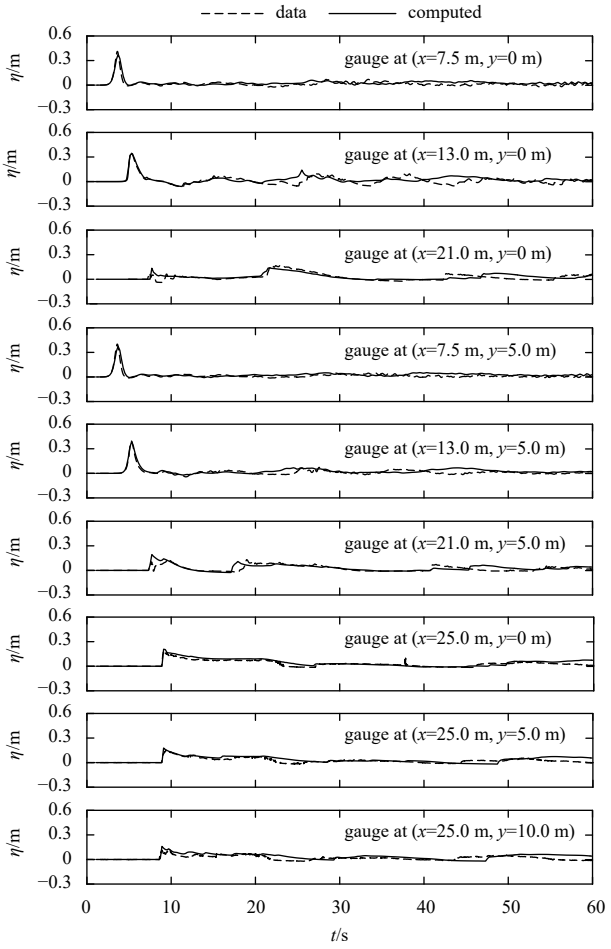

Fig. 8. Time series of modelled and measured surface elevation for solitary wave transformation over a three-dimensional reef.

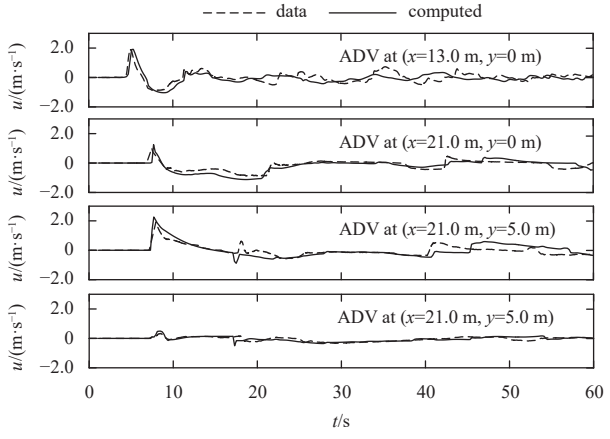

Fig. 9. Time series of modelled and measured velocity for solitary wave transformation over three dimensional reef.

The speedup ratio is case depending. Violent wave breaking and rapid moving of shoreline are involved in Case 3, the functions of dealing with breaking waves (e.g., frequent switching off/on dispersive terms) and moving shoreline (e.g., employing hydrostatic construction technique along wet/dry interfaces) are implemented in GPU kernels. While Case 2 is for non-breaking regular waves, these GPU kernels are not activated at all. As a result, the speedup ratio for Case 2 in 1 024×512 is 41, while it is only 17 for Case 3 of the same grid size. Meanwhile, a decrease of speedup ratio for Case 3 with increasing grid size from 1 024×256 to 1 024×512 is recorded. The following reason might be responsible for this, only around 68.95% of the domain is dry and violent time-varying shoreline motion involves. A loss of efficiency is therefore caused by thread divergence due to an "if statement" implemented in the code to avoid the computations on dry cells. Dealing with more breaking events brought by refined grids is also an extra computation burden.

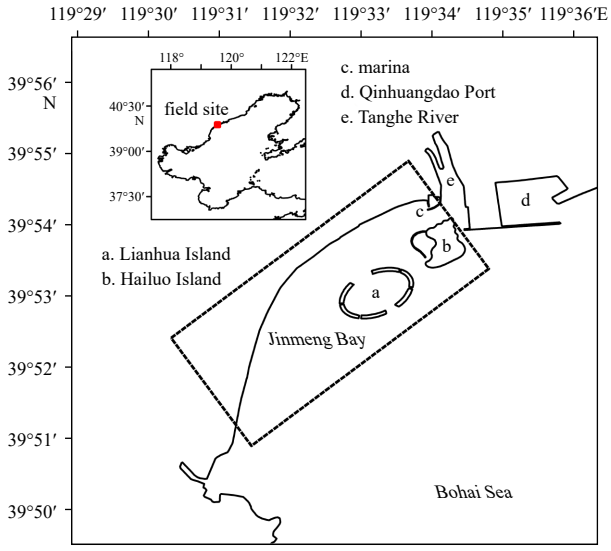

Fig. 10. Location of the field site Jinmeng Bay (the computational domain is bounded by the rectangle box).

As expected and listed in Table 2, double-precision simulation requires more computational time than single-precision simulation. The corresponding speedup ratio for Case 2 is reduced by 20.7% (18.15) for mesh size 512×256 and by 40% (29.05) for mesh size 2 048×512 on Platform I, reduced by 18.7% (19.48) for mesh size 512×256 and by 32.6% (32.57) for mesh size 2 048×512 on Platform II. However, the speedup ratio increases once a double-precision simulation is conducted for Case 3. The underlying reason for this performance is not confirmed yet.

The running time of the single-precision simulation of Case 2 and Case 3 is given in Fig. 13 to illustrate the efficiency of the CR algorithm in solving the tridiagonal equations. Compared against the CR algorithm, the execution time of TMD is approximately an average of 1.72 times and 1.5 times that of the CR algorithm on Platform I and Platform II, respectively.

It is worth noting that the execution time for Case 2 under the mesh size 512×256 is 34.99 s on Platform I and 40.03 s on platform II, illustrating the GPU model runs faster than or almost equal to the real-time 40.0 s.

5 Conclusions

This paper describes the GPU implementation of an existing Boussinesq-type wave model with shock-capturing ability. The acceleration is achieved by utilizing CUDA C as a programing tool on a single GPU device. A brief introduction on governing equations and numerical schemes, the details of GPU implementation, and its optimization are presented. Three numerical tests are performed to validate the correct implementation. By conducting simulations with increasing mesh grid sizes and using speedup ratio as a measure, the computation performance of

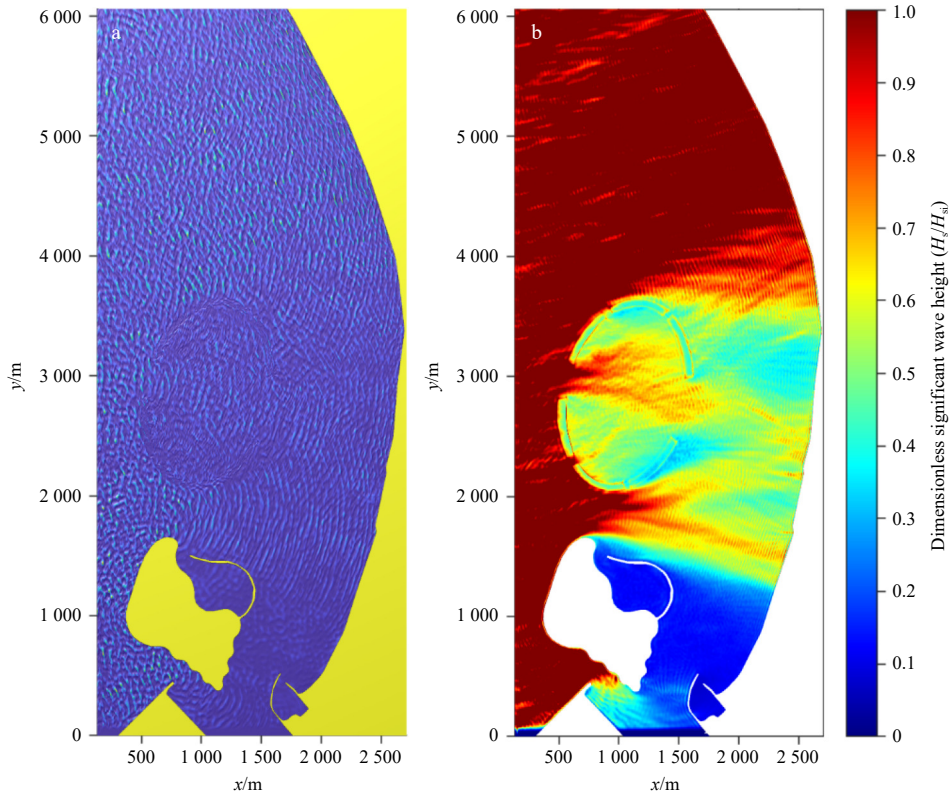

Fig. 11. Snapshots of modelled free surface elevation (a) and wave height field (b) for the field site Jinmeng Bay.

Table 1. Hardware configurations for the CPU and GPU boards

Platform number	CPU/GPU	Cores per CPU / CUDA cores per GPU	Clock frequency /GHz	Single precision performance/ teraflops	Double precision performance/ teraflops
I	Xeon Gold 6226R	16	2.900	1.480	0.740
	RTX 3090	10 496	1.700	35.580	0.556
II	Xeon Gold 6130	16	2.100	1.070	0.535
	Tesla V100S	5 120	1.245	16.350	8.177

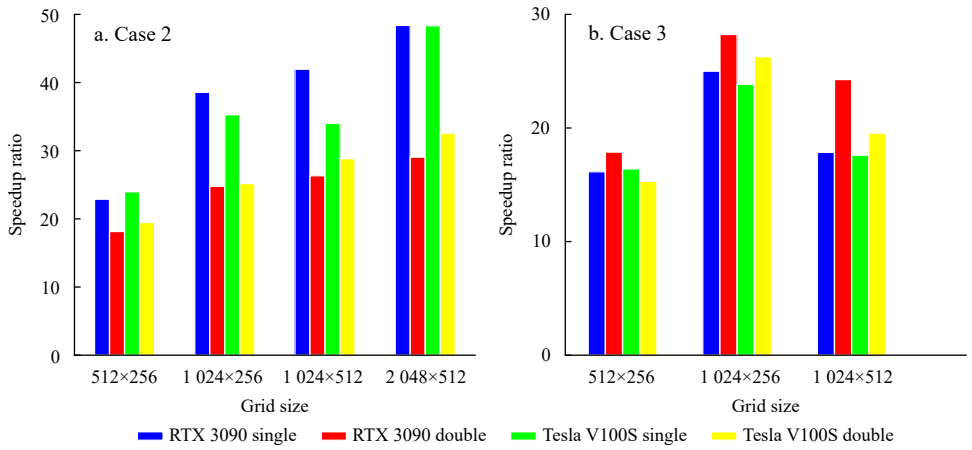

Fig. 12. Speedup ratio for nonbreaking regular wave propagation over an elliptical shoal (a) and solitary wave transformation and breaking on three-dimensional reef bathymetry (b).

the GPU model on two platforms with different hardware configurations is evaluated. The comparison shows that the GPU code is superior to the CPU code with the increasing computation grids. For the considered cases, a speedup of 16.15–48.38 is achieved for single-precision simulations. The speedup is reduced to 15.31–32.57 for double-precision simulations. The execution time of the TMD algorithm is generally 1.5 and 1.72 times that of the CR algorithm on two platforms, respectively. The Jinmeng Bay's single-precision simulation demonstrated the capabilities, accuracy and efficiency of the GPU model, which has sig-

Table 2. A summary of execution time from GPU and CPU simulations

Test case	Grid size	Platform I			Platform II		
		CPU/s	GPU/s	Speedup ratio	CPU/s	GPU/s	Speedup ratio
2	512×256	800.65 (859.48)	34.99 (47.36)	22.88 (18.15)	959.53 (859.48)	40.03 (44.13)	23.97 (19.48)
	1 024×256	3 195.61 (3 447.42)	82.85 (139.15)	38.57 (24.77)	3 767.29 (3 447.42)	106.78 (136.86)	35.28 (25.19)
	1 024×512	6 372.63 (7 046.26)	151.89 (267.65)	41.96 (26.33)	7 455.92 (7 046.26)	219.12 (244.34)	34.03 (28.84)
	2 048×512	25 837.8 (28 017.51)	534.01 (964.36)	48.38 (29.05)	30 320.47 (28 017.51)	627.28 (860.11)	48.34 (32.57)
3	512×256	1 830.65 (3 695.28)	113.38 (206.71)	16.15 (17.88)	2 119.50 (3 076.82)	129.24 (200.96)	16.40 (15.31)
	1 024×256	7 440.89 (15 607.14)	297.62 (552.78)	25.00 (28.23)	8 690.43 (13 354.04)	364.33 (508.3)	23.85 (26.27)
	1 024×512	14 356.59 (36 242.61)	804.11 (1 493.7)	17.85 (24.26)	16 750.92 (25 385.49)	951.42 (1 298.3)	17.61 (19.55)
4	2 048×2 048	482 736.19	8 688.34	55.56	–	–	–

Note: The results from double-precision computation are listed in the brackets. – represents no data.

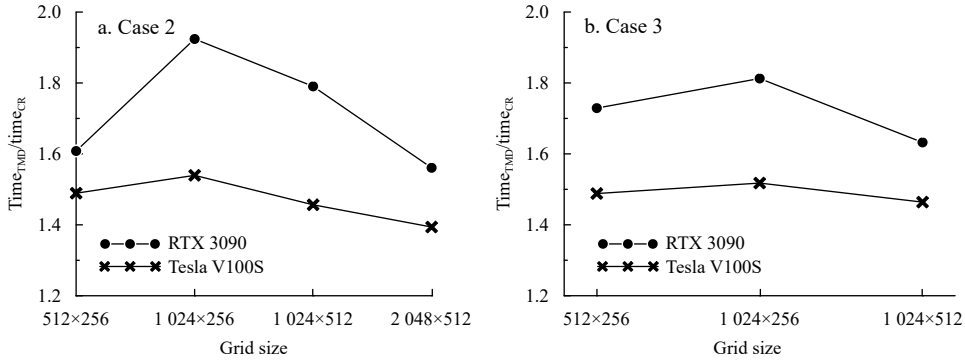

Fig. 13. Comparison of execution time of TMD and CR algorithm (single precision simulation on Platform I).

nificantly accelerated large-scale numerical modeling.

In addition, it is worth noting that the speedup of the numerical model depends to a large extent on computer hardware. Some simulation scenarios show that the present GPU model has the potential of running faster than in real-time. The ongoing multi-GPU implementation will be able to further greatly improve the acceleration performance. The executable code is available upon request.

References

Berkhoff J C W, Booy N, Radder A C. 1982. Verification of numerical wave propagation models for simple harmonic linear water waves. Coastal Engineering, 6(3): 255–379, doi: 10.1016/0378-3839(82)90022-9

Brocchini M. 2013. A reasoned overview on Boussinesq-type models: the interplay between physics, mathematics and numerics. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 469(2160): 20130496

Erduran K S, Ilic S, Kutija V. 2005. Hybrid finite-volume finite-difference scheme for the solution of Boussinesq equations. International Journal for Numerical Methods in Fluids, 49(11): 1213–1232, doi: 10.1002/fld.1021

Fang Kezhao, Liu Zhongbo, Sun Jiawen, et al. 2020. Development and validation of a two-layer Boussinesq model for simulating free surface waves generated by bottom motion. Applied Ocean Research, 94: 101977, doi: 10.1016/j.apor.2019.101977

Fang Kezhao, Liu Zhongbo, Zou Zhili. 2016. Fully nonlinear modeling wave transformation over fringing reefs using shock-capturing boussinesq model. Journal of Coastal Research, 32(1): 164–171

Fang Kezhao, Zou Zhili, Dong Ping, et al. 2013. An efficient shock capturing algorithm to the extended Boussinesq wave Equations. Applied Ocean Research, 43: 11–20, doi: 10.1016/j.apor.2013.07.001

Kim G, Lee C, Suh K D. 2009a. Extended Boussinesq equations for rapidly varying topography. Ocean Engineering, 36(11): 842–851, doi: 10.1016/j.oceaneng.2009.05.002

Kim D H, Lynett P J, Socolofsky S A. 2009b. A depth-integrated model for weakly dispersive, turbulent, and rotational fluid flows. Ocean Modelling, 27(3–4): 198–214, doi: 10.1016/j.ocemod.2009.01.005

Kim B, Oh C, Yi Youngmin, et al. 2018. GPU-accelerated boussinesq model using compute unified device architecture FORTRAN. Journal of Coastal Research, 85(sp1): 1176–1180

Kirby J T. 2016. Boussinesq models and their application to coastal processes across a wide range of scales. Journal of Waterway, Port, Coastal, and Ocean Engineering, 142(6): 03116005

Kirby J T. 2017. Recent advances in nearshore wave, circulation, and sediment transport modeling. Journal of Marine Research, 75(3): 263–300, doi: 10.1357/002224017821836824

Kirby J T, Wei Ge, Chen Qin, et al. 1998. FUNWAVE 1.0 fully nonlinear Boussinesq wave model-documentation and user's manual. Newark: University of Delaware

Klonaris G T, Memos C D, Drønen N K, et al. 2018. Simulating 2DH coastal morphodynamics with a Boussinesq-type model. Coastal Engineering Journal, 60(2): 159–179, doi: 10.1080/21664250.2018.1462300

Liu Zhongbo, Fang Kezhao, Cheng Yongzhou. 2018. A new multi-layer irrotational Boussinesq-type model for highly nonlinear and dispersive surface waves over a mildly sloping seabed. Journal of Fluid Mechanics, 842: 323–353, doi: 10.1017/jfm.2018.99

Liu Zhongbo, Fang Kezhao, Sun Jiawen. 2019. A multi-layer Boussinesq-type model with second-order spatial derivatives: theoretical analysis and numerical implementation. Ocean Engineering, 191: 106545, doi: 10.1016/j.oceaneng.2019.106545

Lynett P J. 2002. A multi-layer approach to modeling generation, propagation, and interaction of water waves [dissertation]. New York: Cornell University

Lynett P J, Swigle D, Son S, et al. 2010. Experimental study of solitary wave evolution over a 3D shallow shelf. In: Proceedings of the 32nd Conference on Coastal Engineering. New York: Curran Associates Inc., 813–823

Madsen P A, Fuhrman D R. 2020. Trough instabilities in Boussinesq formulations for water waves. Journal of Fluid Mechanics, 889: A38, doi: 10.1017/jfm.2020.76

Madsen P A, Sørensen S R. 1992. A new form of the Boussinesq equations with improved linear dispersion characteristics. Part 2. A slowly-varying bathymetry. Coastal Engineering, 18(3–4): 183–204, doi: 10.1016/0378-3839(92)90019-Q

Orszaghova J, Borthwick A G L, Taylor P H. 2012. From the paddle to the beach—A Boussinesq shallow water numerical wave tank based on Madsen and Sørensen's equations. Journal of Computational Physics, 231(2): 328–344, doi: 10.1016/j.jcp.2011.08.028

Shi Fengyan, Kirby J T, Harris J C, et al. 2012. A high-order adaptive time-stepping TVD solver for Boussinesq modeling of breaking waves and coastal inundation. Ocean Modelling, 43–44: 36–51

Shi Fengyan, Malej M, Smith J M, et al. 2018. Breaking of ship bores in a Boussinesq-type ship-wake model. Coastal Engineering, 132: 1–12, doi: 10.1016/j.coastaleng.2017.11.002

Tavakkol S, Lynett P. 2017. Celeris: a GPU-accelerated open source software with a Boussinesq-type wave solver for real-time interactive simulation and visualization. Computer Physics Communications, 217: 117–127, doi: 10.1016/j.cpc.2017.03.002

Tavakkol S, Lynett P. 2020. Celeris Base: an interactive and immersive Boussinesq-type nearshore wave simulation software. Computer Physics Communications, 248: 106966, doi: 10.1016/j.cpc.2019.106966

Wang Yueling, Liang Qiuhua, Kesserwani G, et al. 2011. A 2D shallow flow model for practical dam-break simulations. Journal of Hydraulic Research, 49(3): 307–316, doi: 10.1080/00221686.2011.566248

Yuan Ye, Shi Fengyan, Kirby J T, et al. 2020. FUNWAVE-GPU: multiple-GPU acceleration of a Boussinesq-type wave model. Journal of Advances in Modeling Earth Systems, 12(5): e2019MS001957

Zhang Yao, Cohen J, Owens J D. 2010. Fast tridiagonal solvers on the GPU. ACM SIGPLAN Notices, 45(5): 127–136, doi: 10.1145/1837853.1693472