mathematics

MDPI

Article

Parallel Implicit Solvers for 2D Numerical Models on Structured Meshes

Yaoxin Zhang [1,*], Mohammad Z. Al-Hamdan [1,2,3] and Xiaobo Chao [1]

[1] National Center for Computational Hydroscience and Engineering, University of Mississippi, Oxford, MS 38655, USA; mzalhamd@olemiss.edu (M.Z.A.-H.); chao@ncche.olemiss.edu (X.C.)
[2] Department of Civil Engineering, University of Mississippi, University, MS 38677, USA
[3] Department of Geology and Geological Engineering, University of Mississippi, University, MS 38677, USA
* Correspondence: yzhang@ncche.olemiss.edu; Tel.: +1-6629158972

Abstract: This paper presents the parallelization of two widely used implicit numerical solvers for the solution of partial differential equations on structured meshes, namely, the ADI (Alternating-Direction Implicit) solver for tridiagonal linear systems and the SIP (Strongly Implicit Procedure) solver for the penta-diagonal systems. Both solvers were parallelized using CUDA (Computer Unified Device Architecture) Fortran on GPGPUs (General-Purpose Graphics Processing Units). The parallel ADI solver (P-ADI) is based on the Parallel Cyclic Reduction (PCR) algorithm, while the parallel SIP solver (P-SIP) uses the wave front method (WF) following a diagonal line calculation strategy. To map the solution schemes onto the hierarchical block-threads framework of the CUDA on the GPU, the P-ADI solver adopted two mapping methods, one block thread with iterations (OBM-it) and multi-block threads (MBMs), while the P-SIP solver also used two mappings, one conventional mapping using effective WF lines (WF-e) with matrix coefficients and solution variables defined on original computational mesh, and a newly proposed mapping using all WF mesh (WF-all), on which matrix coefficients and solution variables are defined. Both the P-ADI and the P-SIP have been integrated into a two-dimensional (2D) hydrodynamic model, the CCHE2D (Center of Computational Hydroscience and Engineering) model, developed by the National Center for Computational Hydroscience and Engineering at the University of Mississippi. This study for the first time compared these two parallel solvers and their efficiency using examples and applications in complex geometries, which can provide valuable guidance for future uses of these two parallel implicit solvers in computational fluids dynamics (CFD). Both parallel solvers demonstrated higher efficiency than their serial counterparts on the CPU (Central Processing Unit): 3.73~4.98 speedup ratio for flow simulations, and 2.166~3.648 speedup ratio for sediment transport simulations. In general, the P-ADI solver is faster than but not as stable as the P-SIP solver; and for the P-SIP solver, the newly developed mapping method WF-all significantly improved the conventional mapping method WF-e.

Keywords: parallel; implicit; ADI solver; SIP solver; structured mesh

MSC: 76-10

Citation: Zhang, Y.; Al-Hamdan, M.Z.; Chao, X. Parallel Implicit Solvers for 2D Numerical Models on Structured Meshes. *Mathematics* **2024**, *12*, 2184. https://doi.org/10.3390/math12142184

Academic Editor: Carlo Bianca

Received: 21 June 2024
Revised: 9 July 2024
Accepted: 9 July 2024
Published: 12 July 2024

1. Introduction

In computational fluids dynamics (CFD) analysis, efficient computation still remains a challenge in long-term, large-scale, and high-resolution simulations. Many methodologies/techniques have been developed to alleviate computing efficiency problems, such as the development of efficient numerical schemes [1,2]; model surrogating, i.e., one-dimensional (1D) models surrogating two-dimensional (2D) models [3,4] and data-driven models with machine learning techniques surrogating conventional physics-based

models [5,6]; model couplings, i.e., 1D and 2D model coupling [7,8] and 2D and three-dimensional (3D) model coupling [9]; multi-block algorithm [10]; and parallel computing [11–20].

Since CFD models involve solving a set of highly non-linear partial differential equations (PDEs), their computing efficiency depends largely on numerical solvers, which are designed for different computational mesh systems. Structured meshes and unstructured meshes have been widely used in CFD analysis. For examples, the CCHE2D model [21] and the Delft 3D model [22] are based on the structured mesh system, while the MIKE 21 flow model [23], the HEC-RAS 2D model [8], the SRH-2D model [24], and the CCHE2D-hybrid model [25] use the unstructured system. One big advantage of the structured mesh system is that there are fast numerical solvers available, despite its generation difficulties in complex geometries being more than that of the unstructured mesh system. Among those most widely used solvers, two implicit solvers stand out: the ADI (Alternating Direction Implicit) solver for the tri-diagonal matrix system [26,27] and the SIP (Strong Implicit Procedure) [28] solver for the penta-diagonal matrix system.

The ADI solver is simple and easy to implement, although its implicitness and dependency are not as strong as the SIP solver, and thus its convergence is slower. In fact, the strong dependency of the SIP solver makes it unfriendly to parallelization. In parallel computing, implicit solvers are more difficult to parallelize than explicit solvers because of their strong data dependency. For the ADI solver, based on the Parallel Cyclic Reduction (PCR) algorithm proposed by Hockeny and Jesshope [29], it has been successfully parallelized and applied to flow simulations in CFD analyses [16,17,30] and heat conduction simulations [18,31]. Compared to the ADI solver, the SIP solver is fully implicit and thus computationally more stable. In general, the SIP solver can use a larger time step than the ADI solver. Due to the incomplete LU decomposition, the SIP solver converged much faster than the ADI solver as well.

For the SIP solver, attempts have been made for parallelization. For example, Reeve et al. [32] attempted to parallelize the SIP solver on multi-CPUs (Central Processing Units) using the MPI (Message Passing Interface) framework. Based on a dependency analysis, they proposed two parallel methods, namely, the Wave Front (WF) method and the Red-Black Checkerboard (RBC) method. Deserno et al. [33] also analyzed the effects of nodal numbering on the implementation of the SIP solver on a high performance computing (HPC) platform with shared memory multi-processors in detail and proposed a diagonal process strategy, essentially the same as the WF method. Igounet et al. [34] implemented two variants of the SIP method on the GPU by exploring the possibilities of using shared memory. According to the implementation of the RBC method for the SIP solver for flow simulations, Zhang and Jia [16] recognized that the RBC was just approximations of the SIP algorithm, so that it can be used only for simple problems. Dufrechou et al. [35] designed and implemented a self-scheduling procedure to avoid the overhead of CPU–GPU synchronization implied by the so-called hyper-planes strategy (another version of the WF method).

In this study, the parallel ADI solver (P-ADI) based on the PCR algorithm and the parallel SIP (P-SIP) solver based on the WF method were implemented on the GPU with the CUDA Fortran programming technique [36] and then integrated into a 2D hydrodynamic model, the CCHE2D (Center of Computational Hydroscience and Engineering) model. This study proposed a new mapping strategy for the P-SIP solver, distinguished from the conventional WF method; that is, the matrix coefficients and the solution variables are defined on the WF mesh, and the computation is performed on the WF mesh as well. Two mapping methods of the P-ADI solver and two mapping methods of the P-SIP solvers are validated and compared by examples and applications. It demonstrated that both parallel solvers are capable of significantly improving computing efficiency in complex geometries, such as the flow simulations in the Arkansas River pool 7 with many dikes for navigations, the tidal flow simulations in Lake Pontchartrain, and the sediment transport simulations

with wetting-and-drying process in the Vistula River with dikes and the meandering East Fork River.

2. Numerical Model

2.1. Flow Model

In this study, a two-dimensional (2D) hydrodynamic model, the CCHE2D model [21] is selected for parallelization. The CCHE2D model solves the depth-integrated 2D Navier–Stokes equations as follows:

Continuity equation

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

Momentum equations

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -g\frac{\partial \eta}{\partial x} + \frac{1}{h}[\frac{\partial (h\tau_{xx})}{\partial x} + \frac{\partial (h\tau_{xy})}{\partial y}] + \frac{\tau_{wx} - \tau_{bx}}{\rho h} + f_{Cor}v \tag{2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -g\frac{\partial \eta}{\partial y} + \frac{1}{h}[\frac{\partial (h\tau_{yx})}{\partial x} + \frac{\partial (h\tau_{yy})}{\partial y}] + \frac{\tau_{wy} - \tau_{by}}{\rho h} - f_{Cor}u \tag{3}$$

where u and v are the depth-integrated velocity components in the x and y directions respectively; g is the gravitational acceleration; η is the water surface elevation; ρ is water density; h is the local water depth; f_{Cor} is the Coriolis parameter; τ_{bx} and τ_{by} are shear stresses on the bed surface; τ_{wx} and τ_{wy} are surface wind shear stress.

$$(\tau_{wx}, \tau_{wy}) = \rho_{air} c_{fa} \sqrt{U_w^2 + V_w^2}(U_w, V_w) \tag{4}$$

where c_{fa} is the friction coefficient at the water surface and (U_w and V_w) are wind velocity; and $\tau_{xx}, \tau_{xy}, \tau_{yx}$, and τ_{yy} are the depth-integrated stresses including both viscous and turbulent effects, which are approximated based on Boussinesq's assumption as follows:

$$\tau_{xx} = 2\rho(\nu + \nu_t)\frac{\partial u}{\partial x} \tag{5a}$$

$$\tau_{xy} = \tau_{yx} = \rho(\nu + \nu_t)(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}) \tag{5b}$$

$$\tau_{yy} = 2\rho(\nu_t + \nu)\frac{\partial v}{\partial y} \tag{5c}$$

where ν is the kinematic viscosity of water.

Two zero-equation eddy viscosity models, namely, the parabolic model and the mixing length model, and the more sophisticated two-equation k-ε turbulence model [37] were adopted in the CCHE2D model to calculate the eddy viscosity ν_t, which will not be covered in detail here.

2.2. Sediment Transport Model

The CCHE2D model can simulate the transport of non-uniform sediment mixtures with slope effect correction, secondary flow effect in curved channel, and avalanche effects. The following equations are solved decoupled from flow.

Suspended-load equation [38]:

$$\begin{aligned}\frac{\partial (hC_k)}{\partial t} + \frac{\partial (uhC_k)}{\partial x} + \frac{\partial (vhC_k)}{\partial y} = \frac{\partial}{\partial x}\left(\varepsilon_s h\frac{\partial C_k}{\partial x}\right) + \frac{\partial}{\partial y}\left(\varepsilon_s h\frac{\partial C_k}{\partial y}\right) \\ +\alpha\omega_{sk}(C_{*k} - C_k)\end{aligned} \tag{6}$$

Bed-load equation [38]:

$$\frac{\partial(\delta_b \bar{c}_{bk})}{\partial t} + \frac{\partial(\alpha_{bx} q_{bkx})}{\partial x} + \frac{\partial\left(\alpha_{by} q_{bky}\right)}{\partial y} + \frac{1}{L_t}(q_{bk} - q_{b*k}) = 0 \tag{7}$$

Bed-change equation:

$$(1 - p')\frac{\partial z_{bk}}{\partial t} = \alpha\omega_{sk}(C_k - C_{*k}) + \frac{(q_{bk} - q_{b*k})}{L_t} \tag{8}$$

where ε_s is the eddy diffusivity of sediment; C_k is the concentration of the k-th size class, and C_{*k} is the corresponding suspended-load transport capacity; α is the adaptation coefficient for suspended load; ω_{sk} is the sediment settling velocity; $\alpha_{bx} = u/U$ and $\alpha_{by} = v/U$ q_{b*k} are the direction cosines of bed-load movement; U is the velocity magnitude; q_{bk}, q_{bkx}, and q_{bky} are the bed-load transport capacity, the bed-load transport rate, and transport rate components in x and y directions, respectively; L_t is the adaptation length for bed load; and $p\prime$ is the porosity of bed material, and z_{bk} is the contributed bed elevation.

For multi-sized sediments, the bed-sorting calculation plays a crucial role, which is not listed here but can be found in more detail in [25]. In general, sediment transport is more complex and has more uncertainties than flow simulations due to those empirical/semi-empirical formulas (e.g., transport capacity, incipient motion, settling velocity, adaptation coefficient, porosity, mobile bed roughness, etc.) derived based on physical laws under steady flow conditions [25].

2.3. Discretization

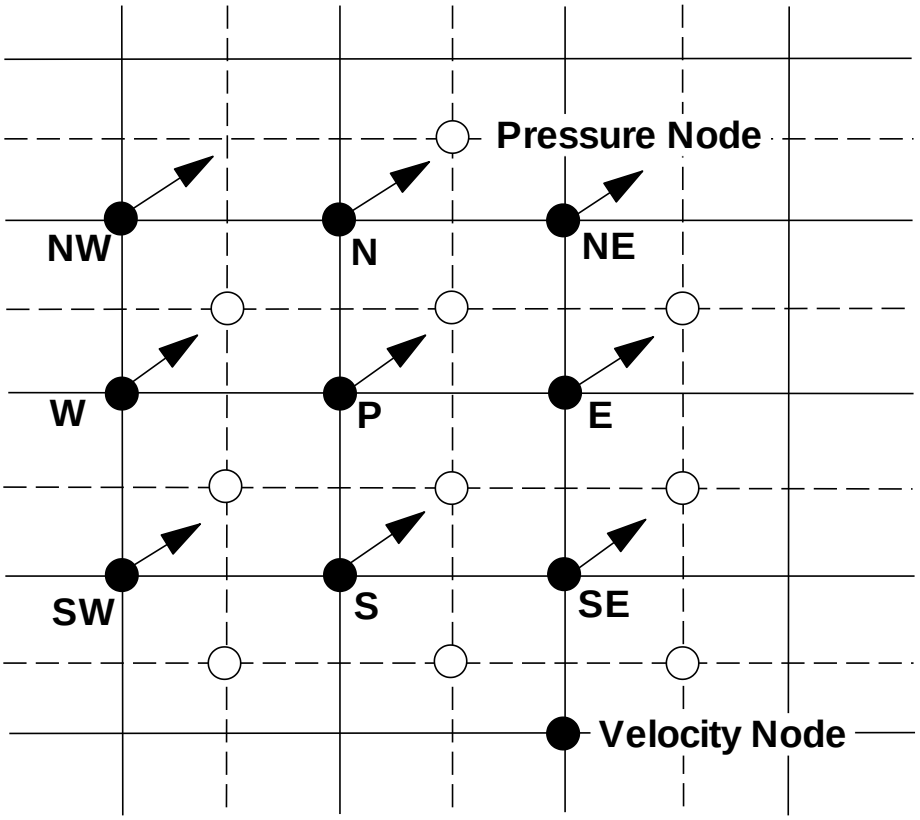
The CCHE2D model [21] uses the efficient element method (EEM) to discretize the governing equations on a staggered structured mesh (Figure 1a). The EEM applies a hybrid scheme of the Finite Element Method (FEM) for the momentum equations and the sediment transport equations, and the Finite Volume Method (FVM) for the continuity equation. When using the FEM for the momentum equations and the sediment transport equations, the nine-node element is used, while for the FVM for the continuity equation, the four-node element is used. Please refer to [21] for more detail.

As shown in Figure 1b, at a typical mesh node $P(i,j)$ with its four neighboring nodes, namely, west $(i-1, j)$, east $(i+1, j)$, south $(i, j-1)$, and north $(i, j+1)$ nodes, an algebraic equation will be formed after discretization:

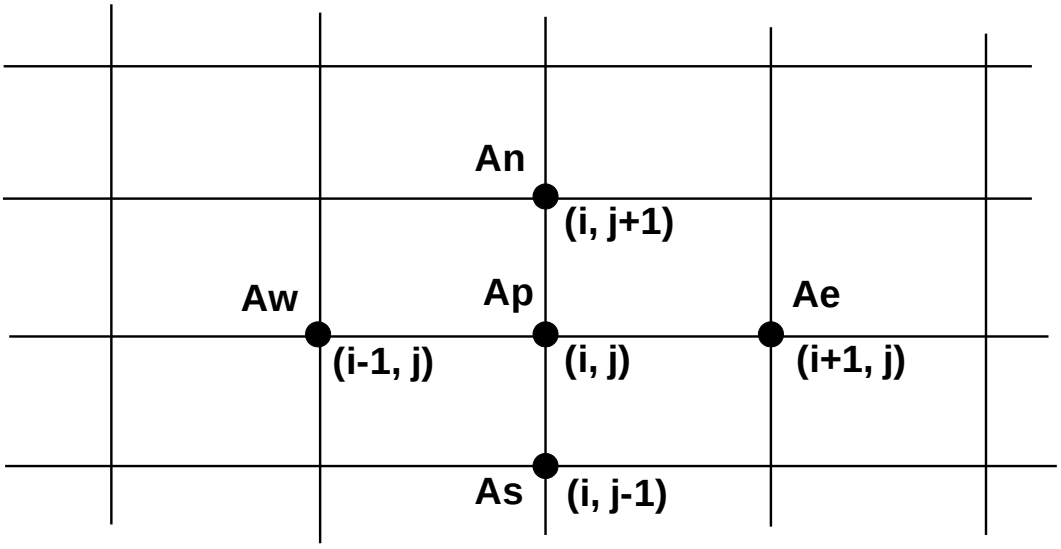
$$A_W\phi_{i-1,j} + A_E\phi_{i+1,j} + A_S\phi_{i,j-1} + A_N\phi_{i,j+1} + A_P\phi_{i,j} = S^{\phi}_{i,j} \tag{9}$$

where ϕ is the dependent variable defined at node (i, j), A is the coefficient matrix, and $S^{\phi}_{i,j}$ denotes the source term.

Equation (9) can be rewritten into the form of a five-diagonal matrix equation system denoted by $[A]\{\phi\} = \{S\}$, where A is the coefficient matrix of $I_{\max} \times J_{\max}$, and it is solved implicitly in the CCHE2D model using the SIP (Strong Implicit Procedure) solver [28].



(**a**)



(**b**)

Figure 1. Four neighboring nodes of node $P_{i,j}$ (**a**) four-nodes element and nine-node element on staggered mesh and (**b**) on non-staggered mesh.

3. CUDA Programming

Parallel computing is an important technique to improve computing efficiency use hardware. Since NVIDIA released the General-Purpose computing Graphic Process Unit (GPGPU) with the CUDA (Computer Unified Device Architecture) framework in 2006, CUDA programming on the GPU has been successfully applied in many disciplines, including CFD analysis [16,17,19,20,30,39].

In NVIDIVA's CUDA architecture (NVIDIA, [40]), the computing units on the GPU are organized in a hierarchical way as block threads (Figure 2). Each thread in blocks that are laid out in a grid would process data in parallel. Basically, the CUDA programming can be divided into three simple steps: (1) copy data from the host memory (CPU) to the device memory (GPU); (2) the host (CPU) calls kernels that perform parallel computing in the device memory (GPU); and (3) copy data from the device memory (GPU) back to the host memory (CPU).

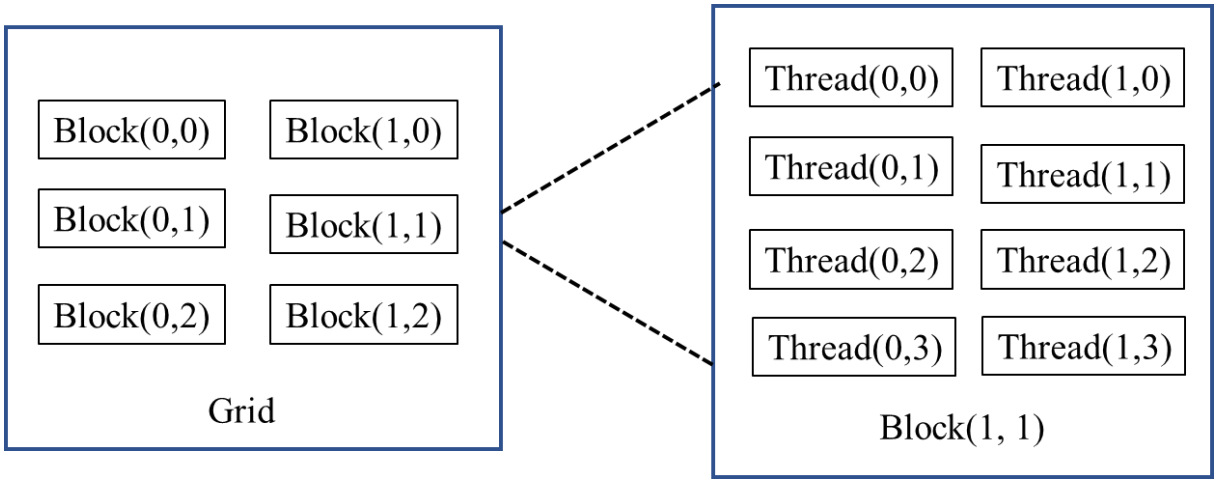


Figure 2. Grid of block threads.

One of the keys for successful CUDA programming is how to map the problem onto the block-threads grid. For a 2D structured mesh ($I_{\max} \times J_{\max}$), an intuitive and straightforward mapping is to use $I_{\max}$ blocks with $J_{\max}$ threads or $J_{\max}$ blocks with $I_{\max}$ threads, which is equivalent to a 2D loop. However, for implicit numerical solvers, such as the ADI solver and the SIP solver, the mapping would become more complicated.

4. Parallel ADI Solver

When using the ADI method to solve Equation (9), a time level is halved so that in each half, one spatial direction is solved implicitly with the other direction explicitly.

$$A_W\phi_{i-1,j}^{n+1/2} + A_E\phi_{i+1,j}^{n+1/2} + A_P\phi_{i,j}^{n+1/2} = S_{i,j}^{\phi} - A_S\phi_{i,j-1}^{n} - A_N\phi_{i,j+1}^{n} \text{ (in } i \text{ direction)} \quad (10a)$$

$$A_S\phi_{i,j-1}^{n+1} + A_N\phi_{i,j+1}^{n+1} + A_P\phi_{i,j}^{n+1} = S_{i,j}^{\phi} - A_W\phi_{i-1,j}^{n+1/2} - A_E\phi_{i+1,j}^{n+1/2} \text{ (in } j \text{ direction)} \quad (10b)$$

where n denotes the time level, and i and j denote two directions in the computational space.

Equation (10) is a tri-diagonal matrix system, which conventionally is solved by the Thomas algorithm for Tri-diagonal Matrix (TDMA) [26]. Rewriting Equation (10) in the form of $[A]\{\phi\} = \{d\}$, one will have

$$a_{i,j} \cdot \phi_{i-1,j} + \phi_{i,j} \cdot \phi_{i,j} + c_{i,j} \cdot \phi_{i+1,j} = d_{i,j} \; [i \in (1, I_{\max}), \; j \in (1, J_{\max})] \quad (11a)$$

where d is the source term matrix, and A is the $I_{\max} \times J_{\max}$ coefficient matrix as follows:

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & a_{I_{\max}-1} & b_{I_{\max}-1} & c_{I_{\max}-1} \\ & & & & a_{I_{\max}} & b_{I_{\max}} \end{pmatrix} \quad (11b)$$

Equation (11) will be used to illustrate the ADI method hereafter in this study.

4.1. TDMA

The TDMA has only two steps. The forward elimination step eliminates the lower diagonal:

$$\begin{aligned} c_1^{new} &= \frac{c_1}{b_1} \\ c_i^{new} &= c_i/(b_i - c_{i-1}^{new} a_i) \end{aligned} \tag{12a}$$

$$\begin{aligned} d_1^{new} &= \frac{d_1}{b_1} \\ d_i^{new} &= \frac{d_i - d_{i-1}^{new} a_i}{b_i - c_{i-1}^{new} a_i} \end{aligned} \tag{12b}$$

where the superscript "new" denotes the updated new values.

The backward substitution step is to obtain the solution:

$$\begin{aligned} \phi_{I_{\max}} &= d_{I_{\max}}^{new} \\ \phi_i &= d_i^{new} - \frac{c_i^{new}}{\phi_{i+1}} \ (i = I_{\max} - 1,\ I_{\max} - 2,\ \ldots,\ 1) \end{aligned} \tag{13}$$

4.2. Cyclic Reduction

Hockeny [27] proposed an alternative algorithm to solve the tri-diagonal system, the Cyclic Reduction (CR) algorithm. Similar to the TDMA, there are also two steps in the CR. In the forward reduction step, odd-numbered rows are successively eliminated by reducing a system to a smaller system with a half number of unknowns until a system of two unknowns is formed. For even-numbered row i, a new equation can be obtained by row operations on $i - 1$, i and $i + 1$; thus, one can obtain

$$\begin{aligned} a_i^{new} &= -a_{i-1} p_1 \\ c_i^{new} &= -c_{i+1} p_2 \\ b_i^{new} &= b_i - c_{i-1} p_1 - a_{i+1} p_2 \\ d_i^{new} &= d_i - d_{i-1} p_1 - d_{i+1} p_2 \end{aligned} \tag{14}$$

where $p_1 = a_i / b_{i-1}$ and $p_2 = c_i / b_{i+1}$.

The backward substitution step successively updates the other half of the unknowns using previously known values:

$$\phi_i = \frac{d_i^{new} - a_i^{new} F_{i-1} - c_i^{new} F_{i+1}}{b_i^{new}} \tag{15}$$

As can be seen, compared to the TDMA, the CR has more computations in the forward reduction steps; thus, it is computationally slower. However, it is much parallel-friendlier in the sense that the TDMA demonstrates stronger dependency than the CR. In the TDMA, the new values of c and d at i depend on their adjacent new values at i – 1, while in the CR, the new values of a, b, c, and d are only related to their previous old values.

4.3. Parallel Cyclic Reduction

Hockeny and Jesshope [29] extended the CR to the Parallel Cyclic Reduction (PCR) due to its parallel-friendly characteristics. The PCR has only the forward reduction step since it halves the system recursively into smaller systems.

Considering that in i direction, the coefficient matrix A is $J_{\max} \times I_{\max}$, while in j direction, the matrix A will be $I_{\max} \times J_{\max}$, Zhang et al. [31] and Heng [41] adopted a simple mapping that each block solves one tri-diagonal system and each thread calculates one node, which is denoted by the one-block mapping (OBM) in this study (Figure 3). Obviously, the OBM is limited by the max number of threads in one block, and the size of the tri-diagonal system to be solved must be less than the square and the cubic of the max thread number in 2D and 3D, respectively.

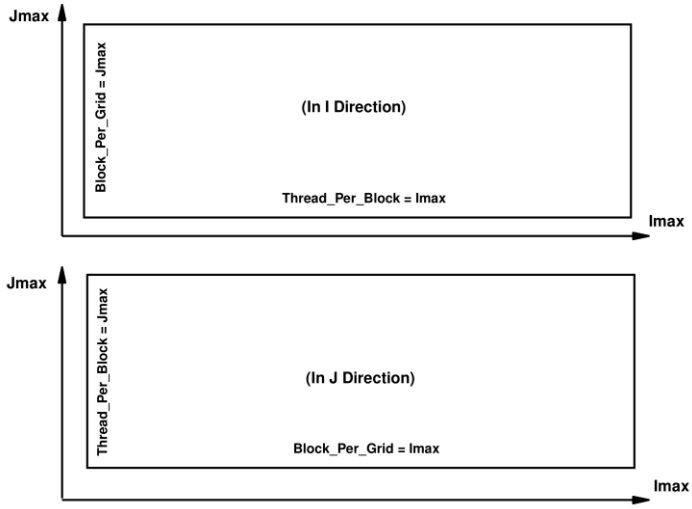


Figure 3. OBM with one block thread.

When using the OBM mapping, in i direction the number of blocks is $J_{\max}$, the number of threads is $I_{\max}$, and the required shared memory will be $4 \times I_{\max} \times 4$ (bytes) for single-precision computation and $4 \times I_{\max} \times 8$ (bytes) for double-precision computation.

To resolve the size limitation problems, Zhang and Jia [16,17] proposed the improved OBM method by allowing the calculations of multiple nodes in one thread, denoted by one-block mapping with iterations (OBM-it). In the OBM-it method, each block is solving one tri-diagonal system, but each thread is calculating multiple nodes. In this method, a partition strategy was proposed as follows:

$$m_i \times I_p \geq I_{\max} \tag{16a}$$

$$m_j \times J_p \geq J_{\max} \tag{16b}$$

where m_i and m_j are the number of partitions in i and j directions and also denote the number of nodes calculated in each thread, and I_p ($\leq$max thread number) and J_p ($\leq$max thread number) denote the number of nodes in each partition in i and j directions, respectively.

Figure 4 shows the mapping of the OBM-it method. In i direction, Thread_Per_Block $\times$ Block_Per_Grid = $J_{\max} \times m_i \times I_p \geq I_{\max}$, while in j direction, Block_Per_Grid $\times$ Thread_Per_Block = $I_{\max} \times m_j \times J_p \geq J_{\max}$. When using OBM-it method, in i direction the number of blocks is $m_j \times J_p$, but the number of threads is I_p, and the required shared memory will be $4 \times I_p \times m_i \times 4$ (bytes) for single-precision computation and $4 \times I_p \times m_i \times 8$ (bytes) for double-precision computation.

Wei at al. [18] proposed the multi-block mapping (MBM) strategy that the whole system is mapped to multiple blocks ($m_i \times I_p \times m_j \times J_p$). When using the MBM method, in i direction, the number of threads is I_p, but the number of blocks is (m_i, $m_j \times J_p$), and the required shared memory will be $4 \times I_p \times 4$ (bytes) for single-precision computation and $4 \times I_p \times 8$ (bytes) for double-precision computation. Compared to the OBM-it method, in the MBM method, the partitions (Equation (16)) are mapped onto the blocks and threads simultaneously [9].

Figure 5 shows the MBM. In i direction, Thread_Per_Block $\times$ Block_Per_Grid = $(I_p, 1) \times (m_i, m_j \times J_p)$, while in j direction, Block_Per_Grid $\times$ Thread_Per_Block = $(m_i \times I_p, m_j) \times (1, J_p)$.

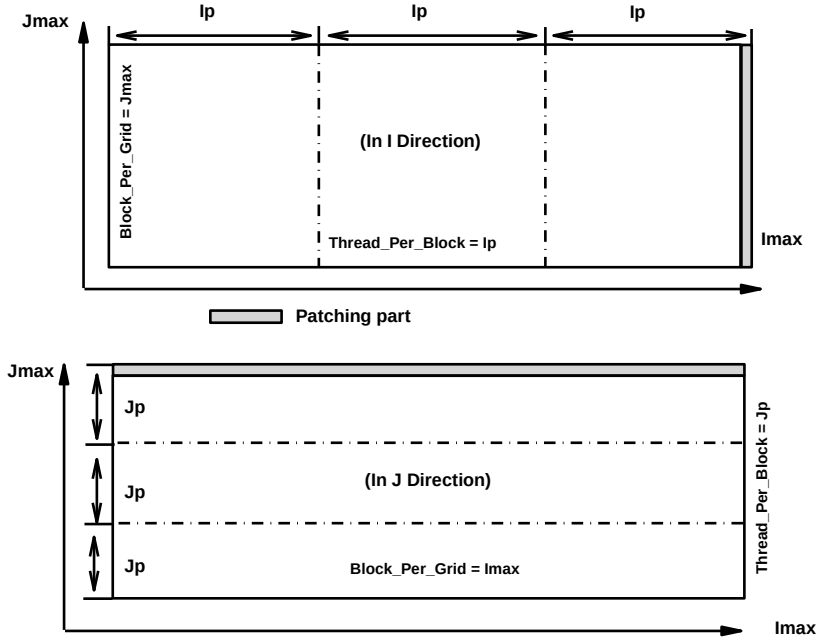


Figure 4. OBM-it with one block thread with iterations.

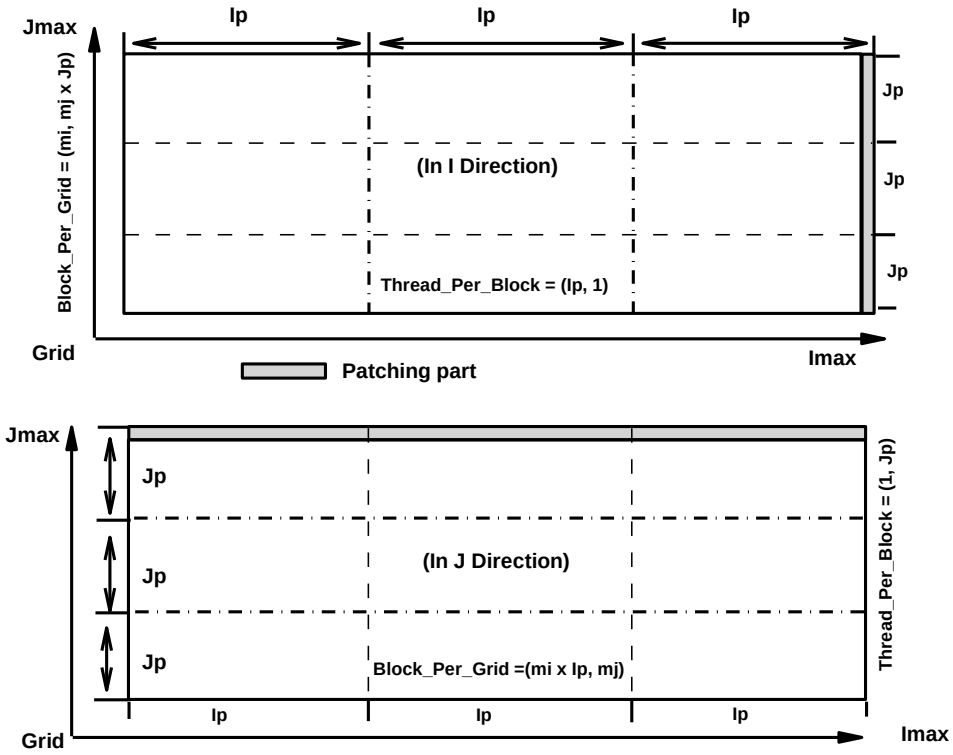


Figure 5. MBM mapping with multiple block threads.

4.4. Relaxation

In practice, in order to improve computation stability and enhance convergences, Zhang and Jia [16] proposed an iteration process with relaxation ($0 < Rx \leq 1$) for the PCR, in which the coefficients a, b, and c are kept unchanged, but the source term d is updated with the new solution.

The OBM-it and MBM of the P-ADI solver are illustrated using pseudo Fortran code in Tables 1 and 2 as follows. Note the different calculations of the block number and thread number in these two mappings.

Table 1. OBM-it of P-ADI solver.

1	Call Build_Matrix
2	
3	Do I = 1, Iteration
4	Block = $I_p * m_i$; Thread = J_p
5	Call PCR_X <<<Block, Thread, $4 * \text{Thread} * 8 * m_j$>>> (a, b, c, d, relaxation, m_j)
6	End Do7
7	Do I = 1, Iter
8	Block = $J_p * m_j$; Thread = I_p
9	Call PCR_Y <<<Block, Thread, $4 * \text{Thread} * 8 * m_i$>>> (a, b, c, d, relaxation, m_i)
10	End Do

Table 2. MBM of P-ADI solver.

1	Call Build_Matrix
2	
3	Do I = 1, Iteration
4	Block = Dim3($I_p * m_i$, m_j, 1); Thread = Dim3(1, J_p, 1)
5	Call PCR_X <<<Block, Thread, $4 * \text{Thread} * 8 * m_j$>>> (a, b, c, d, relaxation, m_j)
6	End Do
7	Do I = 1, Iter
8	Block = Dim3(m_i, $J_p * m_j$, 1); Thread = Dim3(I_p, 1, 1)
9	Call PCR_Y <<<Block, Thread, $4 * \text{Thread} * 8 * m_i$>>> (a, b, c, d, relaxation, m_i)
10	End Do

5. Parallel SIP Solver

5.1. SIP Solver for Penta-Diagonal System

There are four steps in the SIP algorithm, which are simply listed as follows:
The first step LU decomposition factorizes matrix A using $A = LU$ so that

$$\begin{aligned} L^W_{i,j} &= \frac{A_W}{1+r\cdot U^N_{i-1,j}} \\ L^S_{i,j} &= \frac{A_S}{1+r\cdot U^E_{i,j-1}} \\ L^P_{i,j} &= A_P + r\cdot(L^W_{i,j}U^N_{i-1,j} + L^S_{i,j}U^E_{i,j-1}) - L^W_{i,j}U^E_{i-1,j} - L^S_{i,j}U^N_{i,j-1} \\ U^E_{i,j} &= \frac{A_E - r\cdot L^S_{i,j}U^E_{i,j-1}}{L^P_{i,j}} \\ U^N_{i,j} &= \frac{A_N - r\cdot L^W_{i,j}U^N_{i-1,j}}{L^P_{i,j}} \end{aligned} \tag{17}$$

where r is a coefficient in the range of [0, 1] for fast convergence.

For the second step, the residual is calculated by

$$R_{i,j} = S^{\phi}_{i,j} - A\phi_{i,j} \tag{18}$$

In the forward elimination step, the V is computed as

$$V_{i,j} = \frac{R_{i,j} - (L^S_{i,j}V_{i,j-1} + L^W_{i,j}V_{i-1,j})}{L^P_{i,j}}, \tag{19}$$

and the increment of the solution δ^n can then be computed using $U\delta^n = V$.

The backward substitution step calculates the increment of the solution by

$$\delta_{i,j} = V_{i,j} - U^N_{i,j}\delta_{i,j+1} - U^E_{i,j}\delta_{i+1,j} \tag{20}$$

so that the solution can be updated via $\phi^{n+1} = \phi^n + \delta^n$.

Finally, check convergence if δ^n is small enough. If not, go to the second step.

As can be seen, the SIP algorithm demonstrates strong sequential characteristics. In the LU decomposition step, Equation (17) defines the L and U matrix using the coefficient of A in a recursive way; that is, L_W, L_S, and L_P at (i, j) rely on U_N and U_E at $(i-1, j)$ and $(i, j-1)$, respectively, and new values of U_N and U_E at (i, j) require not only L_W, L_S, and L_P at (i, j) but also their adjacent new values at $(i, j-1)$ and $(i-1, j)$. Similarly, in the forward substitution step described by Equation (19), the new value of V at (i, j) depends on its adjacent values at $(i-1, j)$ and $(i, j-1)$. This dependency is indicated in Figure 6a from the lower index to the higher index values. In the backward substitution step, on the contrary, the increment calculation described in Equation (20) should start from the opposite corner as shown in Figure 6b.

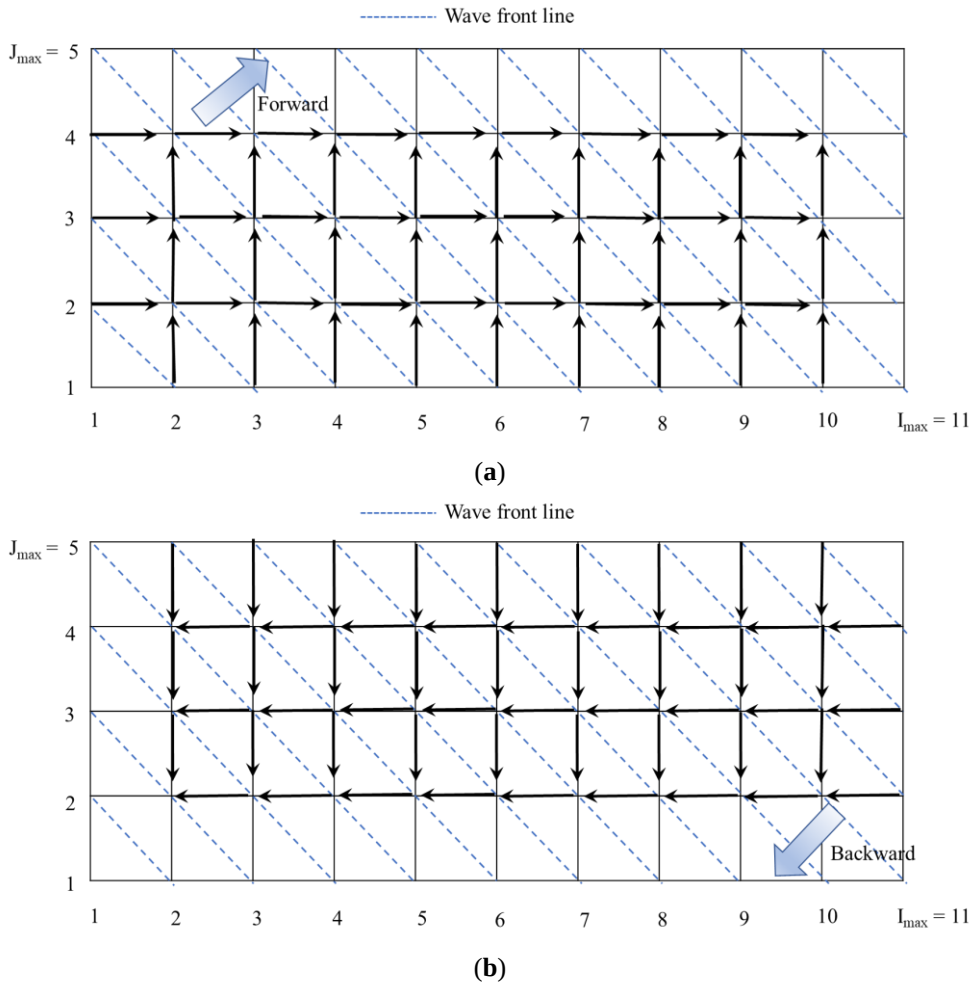


Figure 6. Computation dependencies for SIP solver. (**a**) Forward elimination step; (**b**) Backward substitution step.

5.2. Wave Front Method

As demonstrated by Figure 6, the diagonal dashed lines can be described as the wave front lines, propagating from the left bottom corner to the right top corner for the LU decomposition step and the forward elimination step of the SIP algorithm but in the opposite direction for the backward substitution step. In other words, along the propagating direction, the mesh nodes on one diagonal line have dependency only on the diagonal line behind.

This observation inspired a general parallel strategy for the SIP solver as follows: the parallel computing can be carried out along the wave fronts (WF) line by line, which is

exactly the core concept of the WF method proposed by Reeve et al. [32] and the diagonal process strategy by Deserno et al. [33].

For the WF method, the first step is to identify the WF lines. As shown in Figure 7, for a mesh with the dimension of $I_{max} \times J_{max}$, the number of the WF lines N_w is $I_{max} + J_{max} - 1$. Along one WF line, there are at least one mesh node and at most J_{max} nodes including the boundary nodes. Excluding the boundary nodes, a WF line with at least one internal or non-boundary mesh node is defined as an effective WF line. Therefore, there are $I_{max} + J_{max} - 3$ effective WF lines, which will be computed line by line on the GPU. As identified by Deserno et al. [33], along each WF line, the nodal index (i, j) satisfies the following condition: $(i + j - 1)$ = current WF line number (constant).

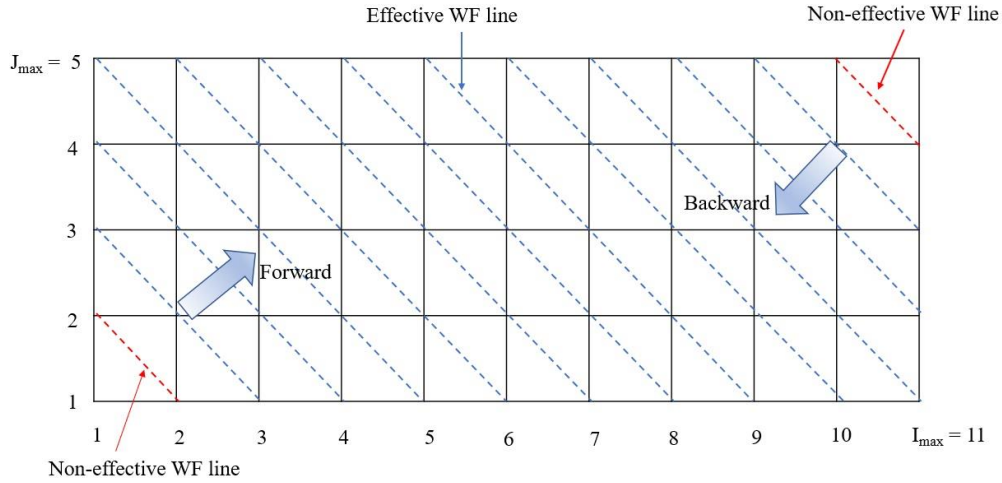


Figure 7. Wave front lines.

Straightforwardly, for one WF line, four kernels can be designed for the LU decomposition step, the residual calculation step, the forward elimination step, and the backward substitution step. For any variable ϕ (=u, v, η, C, q_b) in Equations (1)–(9), the parallel SIP solver can be used in the following way, demonstrated by the pseudo Fortran code in Table 3. As can be seen, three kernels are called in an iterative way by going through all wave front lines, and each kernel is called line by line (number of WF lines) by the host (CPU), which may induce a CPU–GPU overhead problem.

Table 3. P-SIP solver using CPU–GPU overheads.

1	Do I = 1, Num_WF_Line
2	Call LU_Decomposition <<<Block_Num, Thread_Num>>>
3	End Do
4	
5	J = 0
6	J = J + 1
7	Call Cal_Residual <<<Block_Num, Thread_Num>>>
8	Do I = 1, Num_WF_Line
9	Call Forward_Elimination <<<Block,_Num Thread_Num>>>
10	End Do
11	
12	Do I = Num_WF_Line, 1, −1
13	Call Backward_Substitution <<<Block_Num, Thread_Num>>>
14	End Do
15	
16	If (error > Threshold_value and J < Max_Iteration) Go to 6

To alleviate the above CPU–GPU overheads, the kernels should be designed for all WF lines instead of one WF line on the GPU and then be called only once on CPUs

(Table 4). However, due to the strong dependency of the SIP solver, the calculation for all nodes must follow a fixed consecutive order of the WF lines. Moreover, according to the CUDA programming guide, the order of blocks within a grid and warps within a block are undefined. All these concerns raise a question of mapping strategy for the SIP solver: how to carry out the calculation with a fixed order on the block-thread framework of the GPU. Dufrechou et al. [35] identified this problem and proposed a synchronization method to alleviate it. In their method, a global array was defined to record the block number for each WF line. In the main kernel, before computing the next WF line, each block must wait actively for the block number of that global array to become zero. That global array will be decreased by each thread block with a block index lower than the current WF line.

Table 4. P-SIP solver with alleviated CPU–GPU overheads.

1	Call LU_Decomposition <<< Block, Thread >>>
2	
3	J = 0
4	J = J + 1
5	
6	Call Calculate_Residual <<<Block, Thread>>>
7	
8	Call Forward_Elimination <<<Block, Thread>>>
9	
10	Call Backward_Substitution <<<Block, Thread>>>
11	
12	If (Residual > Threshold_value and J < Max_Iteration) Go to 4

5.3. Wave Front Mesh

To fulfill the parallel strategy of the WF method, a mapping needs to be established between the nodes on WF lines and the original computational mesh nodes. Therefore, an auxiliary WF mesh is constructed, through which the computational nodes can be mapped onto the hierarchical block-threads framework of the GPU.

The WF mesh is structured with the size of $N_W \times W_p$. Each WF line has N_p nodes including the computational nodes and the virtual nodes outside of domain. Then this WF mesh is mapped onto the block-thread framework on the GPU as follows: B blocks per grid and W_p threads per block on GPU. All WF lines are nearly equally distributed to each block, so the number of wave lines in each block is $D_w = N_W/B$. To enhance coalescing in memory accessing, the number of nodes along each WF line is adjusted by the warp size (32) so that $W_p = 16 \times (W_p - 1)/16$. In such a case, in the WF mesh there may be more virtual nodes along each WF line. When W_p is greater than the maximum thread number, iterations will be applied for each thread so that more than one node will be calculated in one thread.

Figure 8 shows two WF meshes: one with effective WF lines only (denoted by WF-e) and the other with all WF lines (denoted by WF-all), which correspond to two different mapping strategies.

In addition to the WF-e mesh being smaller in size than the WF-all mesh, the key difference between the WF-e and the WF-all lies in on which mesh the matrix coefficients and solution variable are defined. In the WF-e, the matrix coefficients and the solution variable are defined on the original computational mesh; the mapping between the WF mesh and the original mesh is limited only for the effective WF lines and needed dynamically through an index converter (an array stores the nodal number of the original mesh) in each thread. In the WF-e, the computation is essentially on the original mesh, which is mapped onto block threads using the WF mesh. The index mapping between the original mesh and the WF mesh in each thread hinders fast data retrieving and storing.

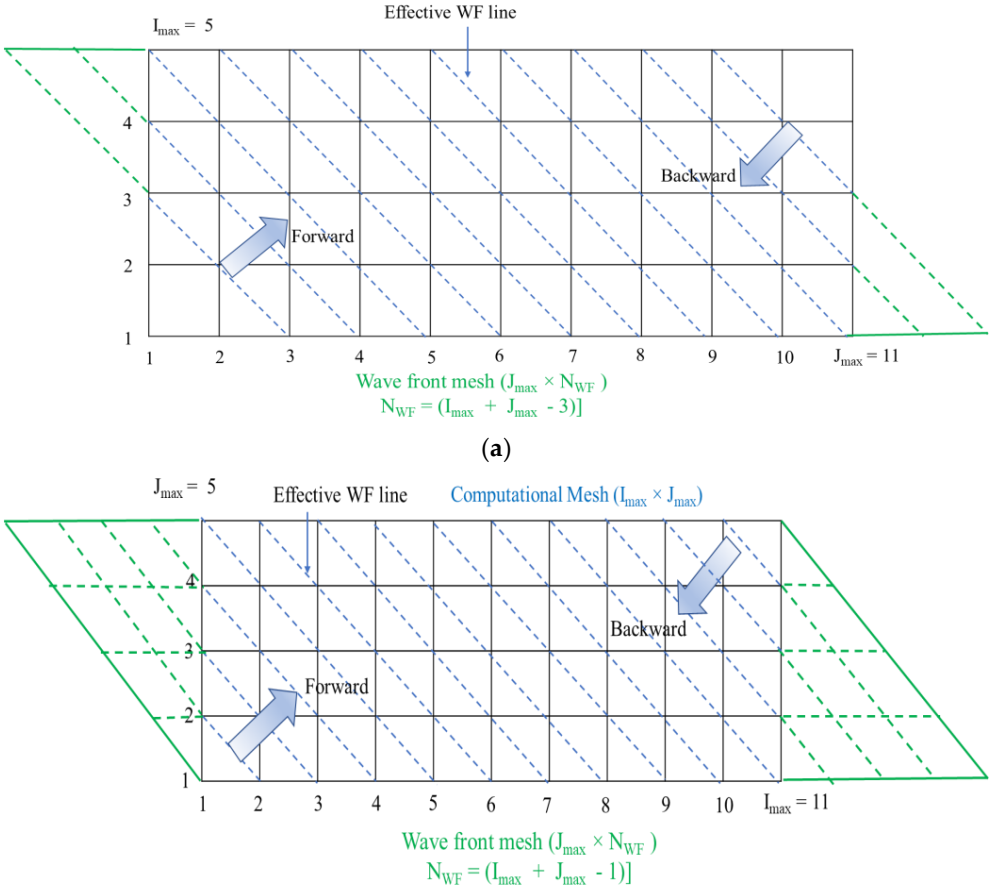


Figure 8. WF mesh: (**a**) WF-e and (**b**) WF-all.

In contrast, in the WF-all, the matrix coefficients and the solution variable are defined on the WF mesh instead, and the mapping between the WF mesh and the original mesh is conducted before the SIP computation. The computation is carried out on the WF at each thread, and after computations, the solution needs to be mapped back onto the original mesh.

Figure 9 shows the mapping between the WF mesh and the original mesh. In a nine-node stencil, as can be seen, the east ($I + 1$, j) and west ($i - 1$, j) nodes on two meshes overlap, but the north node of the original mesh (i, $j + 1$) corresponds to the northeast node of the WF mesh ($i_w + 1$, $j_w + 1$), and the south node of the original mesh (i, $j - 1$) corresponds to the southwest node of the WF mesh ($i_w - 1$, $j_w - 1$).

Since the computation is carried out on the WF mesh, the SIP algorithm for the original mesh (Equations (17)–(20)) needs modification. Following the mapping in Figure 9, the SIP algorithm is modified for the WF mesh as follows:

LU decomposition

$$\begin{aligned}
L^W_{i_w,j_w} &= \frac{A_W}{1+r\cdot U^N_{i_w-1,j_w}} \\
L^S_{i_w,j_w} &= \frac{A_S}{(1+r\cdot U^E_{i_w-1,j_w-1})} \\
L^P_{i_w,j_w} &= A_P + r\cdot (L^W_{i_w,j_w} U^N_{i_w-1,j_w} + L^S_{i_w,j_w} U^E_{i_w-1,j_w-1}) - L^W_{i_w,j_w} U^E_{i_w-1,j_w} - L^S_{i_w,j_w} U^N_{i_w-1,j_w-1} \\
U^E_{i_w,j_w} &= \frac{A_E - r\cdot L^S_{i_w,j_w} U^E_{i_w-1,j_w-1}}{L^P_{i_w,j_w}} \\
U^N_{i_w,j_w} &= \frac{A_N - r\cdot L^W_{i_w,j_w} U^N_{i_w-1,j_w}}{L^P_{i_w,j_w}}
\end{aligned} \tag{21}$$

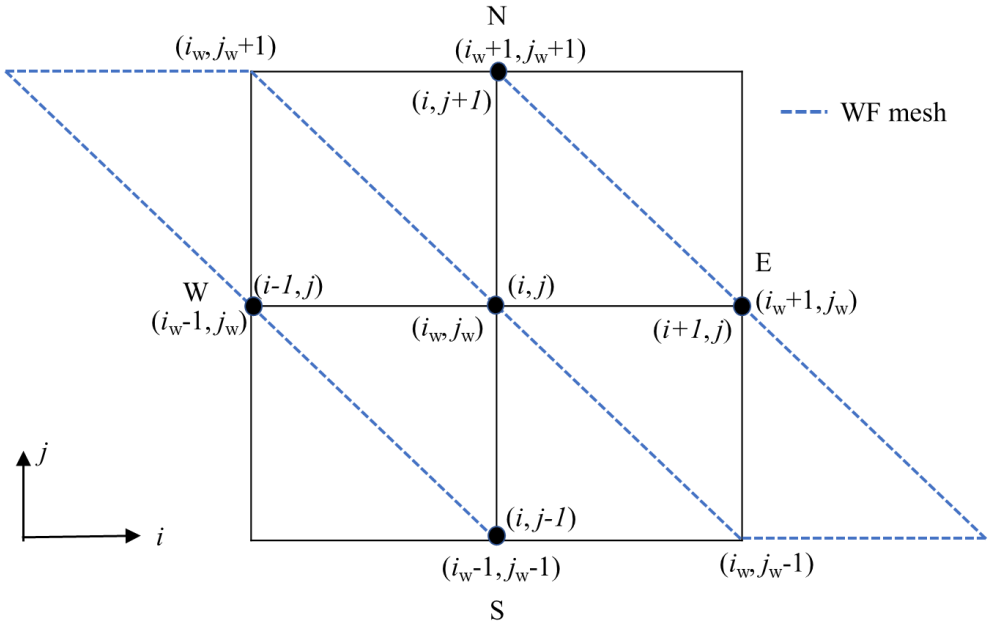


Figure 9. Mapping between WF mesh and original mesh.

Forward elimination

$$V_{i_w,j_w} = \frac{R_{i_w,j_w} - (L^S_{i_w,j_w} V_{i_w-1,j_w-1} + L^W_{i_w,j_w} V_{i_w-1,j_w})}{L^P_{i_w,j_w}} \tag{22}$$

Backward substitution

$$\delta_{i_w,j_w} = V_{i_w,j_w} - U^N_{i_w,j_w} \delta_{i_w+1,j_w+1} - U^E_{i_w,j_w} \delta_{i_w+1,j_w+1} \tag{23}$$

Note the index difference between Equations (17), (19) and (20) and their counterparts, Equations (21)–(23).

For completeness, two mappings of the P-SIP solver, including the kernel for the LU decomposition step (the kernels for the forward elimination step and the backward substitution step are similar), are listed in Tables 5 and 6 using pseudo Fortran code. As can be seen, in the kernel the WF-e needs dynamic mapping from the WF mesh to the OM, and there is one more kernel used to map the solution from WF mesh back to OM in WF-all. This kernel can also be combined into the backward calculation so that the solution mapping can be conducted simultaneously in the backward substitution step.

Table 5. (a) WF-e of P-SIP solver. (b) Kernel of LU decomposition for WF-e.

	(a)
1	Call Build_Matrix_on_OM
2	Block = N_{WF}/D_w; Thread = W_p
3	Call LU_Decomposition_OM<<<Block, Thread>>>
4	Do While (Residual > Threshold And. I < Max_Iteration)
5	Td = 256; Bk = (Total_WF_Node + Td)/Td
6	Call Cal_Residual_on_OM<<<Bk, Td>>>
7	
8	Call Forward_on_OM<<<Block, Thread>>>
9	Call Backward_on_OM <<<Block, Thread >>>
10	I = I+1
11	End Do

Table 5. *Cont.*

	(b)
1	! Index Calculation
2	B = BlockIdx%X; I = ThreadIdx%X ! Block Index and Thread Index
3	! Ln is the total number of WF lines; Nb is number of blocks
4	Do J = (B − 1) ∗ Ln/Nb + 1, B ∗ Ln/Nb ! For WF lines in current block
5	Jw = (J − 1) ∗ Wp + I ! Index on WF mesh
6	k = Map(Jw) ! Index on original mesh
7	Lw(k) = Aw(k)/(1 + alfa ∗ Un(k − J_{max}))
8	Ls(k) = An(k)/(1 + alfa ∗ Ue(k − 1))
9	
10	P1 = alfa ∗ Lw(k) ∗ Un(k − J_{max})
11	P2 = alfa ∗ Ls(k) ∗ Ue(k − 1)
12	Lp(k) = 1./(Ap(k) + P1 +P2 − Lw(k) ∗ Ue(k − J_{max})-Ls(k) ∗ Un(k − 1))
13	
14	Un(k) = (As(k) − P1) ∗ Lp(k)
15	Ue(k) = (Ae(k) − P2) ∗ Lp(k)
16	
17	End Do

Table 6. (a) WF-all of P-SIP solver. (b) Kernel of LU decomposition for WF-all.

	(a)
1	Call Build_Matrix_on _OM_and_Map_to_WF
2	Block = N_{WF}/D_w; Thread = W_p
3	Call LU_Decomposition_on_WF<<<Block, Thread>>>
4	Do While (Residual > Threshold And. I < Max_Iteration)
5	Td = 256; Bk = (Total_WF_Node + Td)/Td
6	Call Cal_Residual_on_WF<<<Bk, Td>>>
7	
8	Call Forward_on_WF<<<Block, Thread>>>
9	Call Backward_on_WF <<<Block, Thread >>>
10	I = I+1
11	End Do
12	Call Map_Solution_from_WF_to_OM<<<Bk, Td>>>
	(b)
1	! Index Calculation
2	B = BlockIdx%X; I = ThreadIdx%X ! Block Index and Thread Index
3	! Ln is the total number of WF lines; Nb is num of blocks
4	Do J = (B − 1) ∗ Ln/Nb +1, B ∗ Ln/Nb ! For WF lines in current block
5	k = (J − 1) ∗ Wp + I ! Index on WF mesh
6	
7	Lw(k) = Aw(k)/(1 + alfa ∗ Un(k − J_{max}))
8	Ls(k) = An(k)/(1 + alfa ∗ Ue(k − 1 − J_{max}))
9	
10	P1 = alfa ∗ Lw(k) ∗ Un(k − J_{max})
11	P2 = alfa ∗ Ls(k) ∗ Ue(k − 1 − J_{max})
12	Lp(k) = 1./(Ap(k) + P1 +P2 − Lw(k) ∗ Ue(k − J_{max}) − Ls(k) ∗ Un(k – 1 − J_{max}))
13	
14	Un(k) = (As(k) − P1) ∗ Lp(k)
15	Ue(k) = (Ae(k) − P2) ∗ Lp(k)
16	
17	End Do

6. Implementation

In the serial version of the CCHE2D model, most of the variables were defined as 2D or 3D arrays, which are conceptually clear and simple. However, in parallelization, to allow faster access of memories on devices (GPUs), one-dimensional arrays in the device memory

were used to store the global variables in order to reduce the amount of memory transfer to a minimum. Specifically, for most of the flow variables defined by 2D arrays ($I_{\max}$, $J_{\max}$), a 1D index $k = (i-1) \cdot J_{\max} + j$ is used to map a 2D index (i, j); for the fractional sediment variables defined in 2D arrays (N_p, N_{Size}) with $N_p = I_{\max} \times J_{\max}$, a 1D index $k = (i-1) \cdot N_{size} + j$ is used; for the layered bed-composition variables defined in (N_p, N_{layer}), a 1D index $k = (i-1) \cdot N_{layer} + j$ is used.

The CCHE2D model consists of many modules which make it capable of simulating flows of all regimes and sediment transports. It uses a time-marching solution procedure. All modules within the time-marching loop were parallelized. Basically, the parallel computing of the CCHE2D model involves three steps, as mentioned previously: (1) in the "initialization" step, all host (CPU) data and device (GPU) data were declared, initialized, and loaded with input data, and data transfers between host and device were carried out as well. For the P-SIP solver, a mapping between the original mesh and the WF mesh is established. For WF-e, a global array is used to store the index of the original mesh along each WF line (WF $\rightarrow$ OM), while for WF-all, a global array is used to store the index of the WF mesh for each original mesh node (OM $\rightarrow$ WF) in order to define matrix coefficients and solution variables on the WF mesh; (2) in the time-marching loop, the host calls kernels to perform computations on the GPU using the device data; and (3) after the time-marching loop is finished, the device data are transferred and copied to the host.

In the implementations, the double-precision floating point data type was used. All the testing examples and application were conducted on a PC with Intel Core i7 (2.93 GHz), 14GB Memory, and a Tesla 2070 GPU card with 48k shared memory. The PGI CUDA-Fortran developed by the Portland Group [36] was used to parallelize the CCHE2D model. Note that since the Portland Group was purchased by NIVIDIA, the PGI CUDA-Fortran used in the current parallelization is fixed on PGI Visual Fortran 12.1 with the CUDA Toolkit 4.2, which is way behind the latest CUDA Toolkit 12.3. Therefore, this parallel implementation cannot use the latest advanced CUDA capabilities, which may affect the whole performance of the model.

7. Example and Application

7.1. 2D Heat Diffusion

The heat-transfer problem was first used to test the P-ADI solver and the P-SIP solver. In a rectangle domain, the heat diffusion equation as follows is solved:

$$\rho c \frac{\partial T}{\partial x} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{24}$$

where T is temperature, ρ is the density of conduction media, c is specific heat, and k is the diffusion coefficient.

The test domain was discretized into 1000 $\times$ 1000 nodes, and there are 1999 WF lines including 1995 effective WF lines and 1008 warp-size-adjusted nodes along each WF line. Seven solvers with different configurations, namely, ADI-TDMA on the CPU, SIP for original mesh (SIP-OM) on the CPU, SIP using WF lines (SIP-WF) on the CPU, P-ADI OBM-it on the GPU, P-ADI MBM on the GPU, P-SIP WF-e on the GPU, and P-SIP WF-all on the GPU were compared.

Table 7 lists the CPU running time for these seven configurations, where "s" denotes second, and "m" denotes minute.

Table 7. CPU time for heat-transfer problem.

Mesh $I_{max} \times J_{max}$	Sim. Time (s)	Time Step (s)	On CPU			On GPU			
			ADI TDMA	SIP-OM	SIP-WF	P-ADI OBM-It	P-ADI MBM	P-SIP WF-e	P-SIP WF-All
1000 × 1000	3600	1	254 s	462 s	48.4 m	32.2 s	36.5 s	312 s	85.2 s
Speedup Ratio			-	-	-	7.89	6.96	1.48	5.42

As can be seen, on the CPU, the ADI with TDMA is faster than the two SIP configurations, and the SIP-WF using the WF lines is the slowest, which demonstrates how much the mapping between the original mesh and the WF mesh affects the SIP computations. The SIP for the original mesh requires $I_{max} \times J_{max}$ iterations for each SIP step, while the WF version needs $J_{max} \times (I_{max} + J_{max} - 5)$ iterations. When $I_{max} = J_{max}$, i.e., the current case, the WF version would need almost double the iterations. In addition, when following the WF lines, the dynamic mapping in each iteration step affects the memory access speed significantly.

On the GPU, correspondingly, two P-ADI configurations are faster than two P-SIP configurations. The WF-all is 3.67 times faster than the WF-e and 5.42 times faster than the SIP-OM; the speedup ratio of WF-e is only 1.48 compared to SIP-OM, while the speedup ratio for the P-ADI OBM-it and the P-ADI BMB is 7.89 and 6.96, respectively. For the P-ADI, Wei et al. [18] reported that the MBM could achieve up to about 70 times speedup when using the single-precision floating point data type.

In this heat-diffusion problem, with the same time step, the ADI solver is faster than the SIP solver due to its lighter computing loads. Computationally, the TDMA has only two steps (two loops in computation) and the SIP has four steps (four loops in computation) with inner iterations to check the residual error. Therefore, the CPU version of ADI is about two times (1.81) faster than that of the SIP.

Similarly, the P-ADI solvers only has two kernels, but the P-SIP solver has four kernels with inner iterations, which partially explains why the P-ADI demonstrated higher efficiency than the P-SIP. With the matrix coefficients and the solution variable defined on the WF mesh, the WF-all performed at a much higher computing speed than the WF-e with the dynamic mapping between the original mesh and the WF mesh in each thread.

7.2. Flow Simulation

The P-ADI and the P-SIP on the GPU were integrated into theCCHE2D model and applied to two flow simulation cases: steady flow in the Arkansas River pool 7 and the unsteady flow in Lake Pontchartrain.

Tables 8 and 9 list the CPU running time for the model using both solvers with different configurations, respectively. Note that the listed CPU running times were for the comprehensive performances of the whole model with many other parallel implementations in the entire time marching loop, not purely for the solvers themselves. According to the CPU time comparisons, the P-ADI solver demonstrated higher computing efficiency than the P-SIP solver in both cases, especially for the WF-e mapping method.

Table 8. Run time for Arkansas River.

Solver		Mesh $I_{max} \times J_{max}$	Sim. Time (h)	Time Step (s)	CPU (s)	GPU (s)	Speed Ratio
P-ADI	OBM-it	65 × 1204	11.1	24	698.6	140.2	4.98
	MBM	65 × 1204	11.1	24	698.6	146.7	4.76
P-SIP	WF-e	65 × 1204	11.1	24	698.6	276.1	2.53
	WF-all	65 × 1204	11.1	24	698.6	169.3	3.73

Table 9. Run time for Lake Pontchartrain.

Solver		Mesh $I_{max} \times J_{max}$	Sim. Time (day)	Time Step (s)	CPU (min)	GPU (min)	Speed Ratio
P-ADI	OBM-it	141 × 224	30.3	100	71	17.24	4.12
	MBM	141 × 224	30.3	100	71	16.98	4.18
P-SIP	WF-e	141 × 224	30.3	100	71	47.92	1.48
	WF-all	141 × 224	30.3	100	71	23.66	3.00

7.2.1. Steady Flow Simulation in Arkansas River Pool 7

In the Arkansas River pool 7 (Figure 10), there are more than 30 dikes serving to improve navigation conditions, which make the flow field complicated. Velocity profiles

at six cross sections from mile 147 to mile 141.4, measured by USGS, were selected for validating the simulation flow fields. Figure 11 compares those simulated velocity profiles by the CPU-serial model with the SIP solver and the GPU-parallel model with the P-ADI and P-SIP solvers to the measurements. Note that two configurations for the P-SIP solver yielded almost the same result, so they are denoted by GPU-SIP in this figure. Similarly, the results from the P-ADI solver are denoted by GPU-ADI. These notations will be applied to other cases hereafter. Table 10 lists the L_2 error norm defined by

$$L_2 = \left(\frac{\sum_i (N_i - M_i)^2}{\sum_i M_i^2}\right)^{1/2} \tag{25}$$

where N denotes the numerical result and M denotes the measured one.

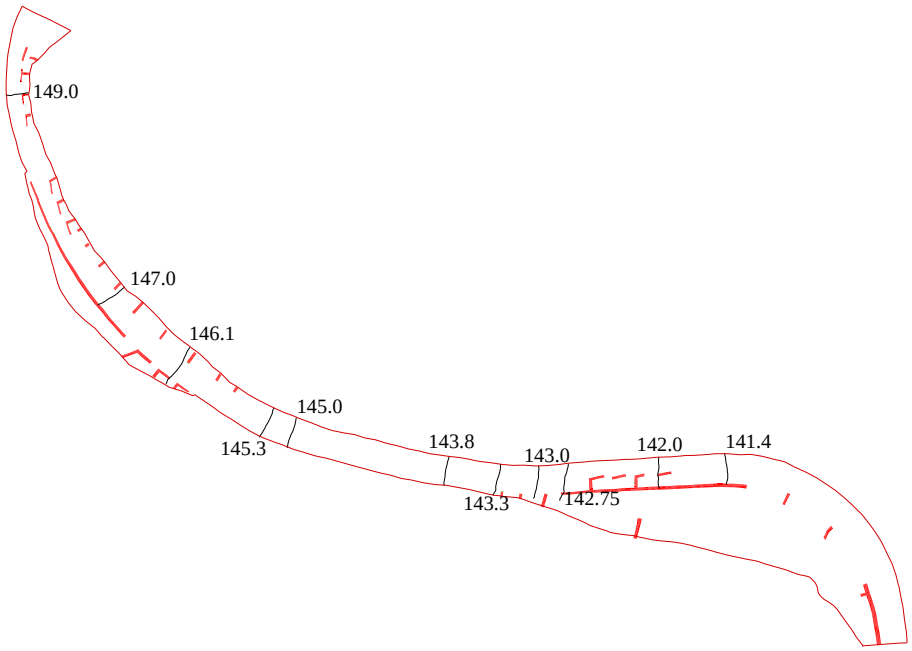


Figure 10. Arkansas River pool 7 with more than 30 dikes.

Table 10. L_2 error norms.

Case		CPU	P-SIP	P-ADI
Arkansas River Pool 7	Mile 141.4-X	0.223	0.213	0.240
	Mile 141.4-Y	0.947	0.955	0.953
	Mile 142.7-X	0.207	0.197	0.169
	Mile 142.7-Y	0.837	0.702	0.923
	Mile 143-X	0.178	0.176	0.195
	Mile 143-Y	0.804	0.826	0.823
	Mile 143.3-X	0.238	0.233	0.263
	Mile 143.3-Y	0.954	0.925	0.960
	Mile 146.1-X	0.258	0.264	0.275
	Mile 146.1-Y	0.245	0.229	0.249
	Mile 147.0-X	0.354	0.329	0.338
	Mile 147.0-Y	0.400	0.375	0.384
Lake Pontchartrain	Mandeville	0.134	0.162	0.189
	Westend	0.262	0.273	0.321
	Southlake-X	0.885	0.877	0.886
	Southlake-Y	0.767	0.750	0.782
East Fork River		0.483	0.391	0.400

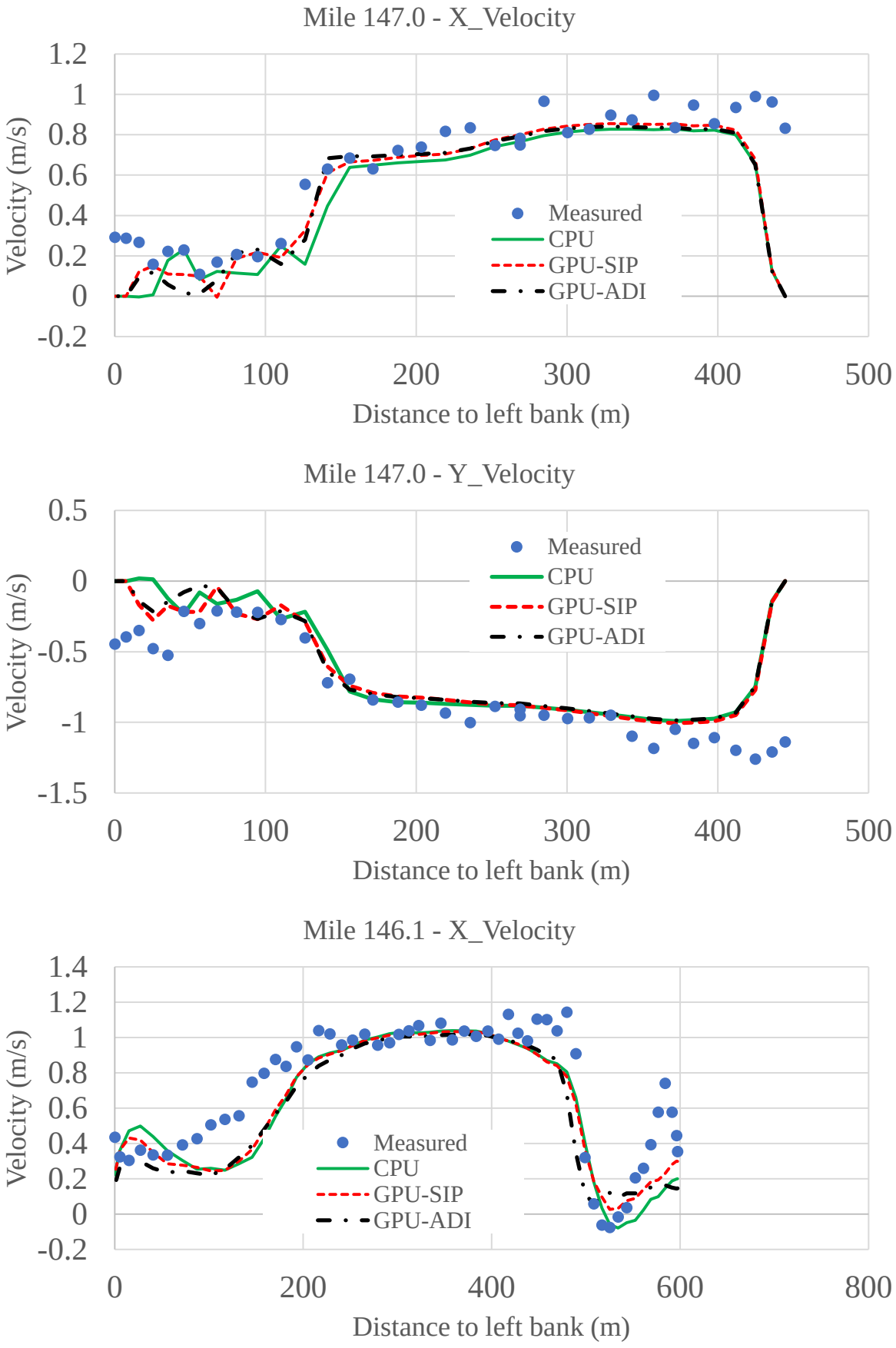


Figure 11. *Cont.*

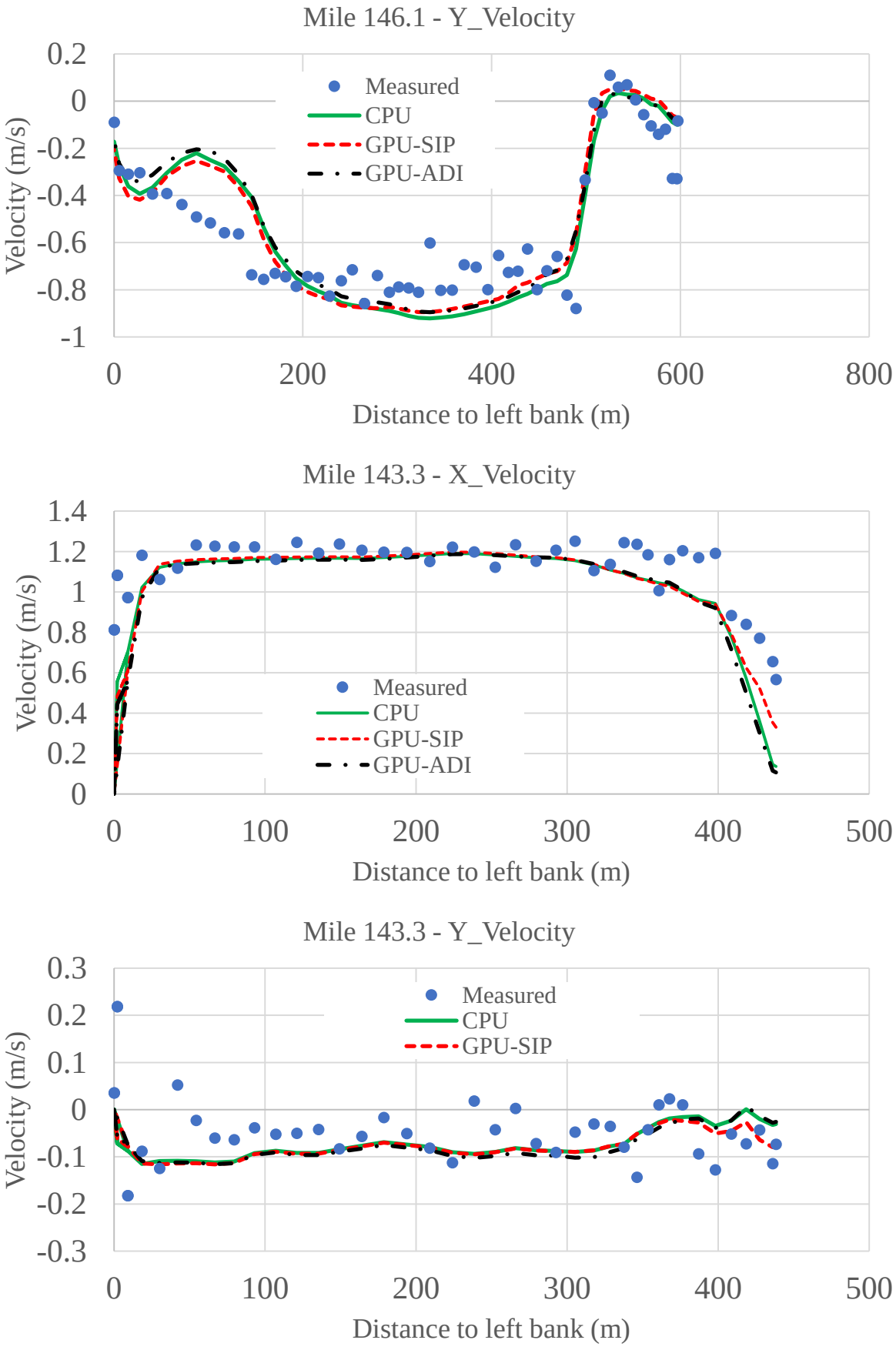


Figure 11. *Cont.*

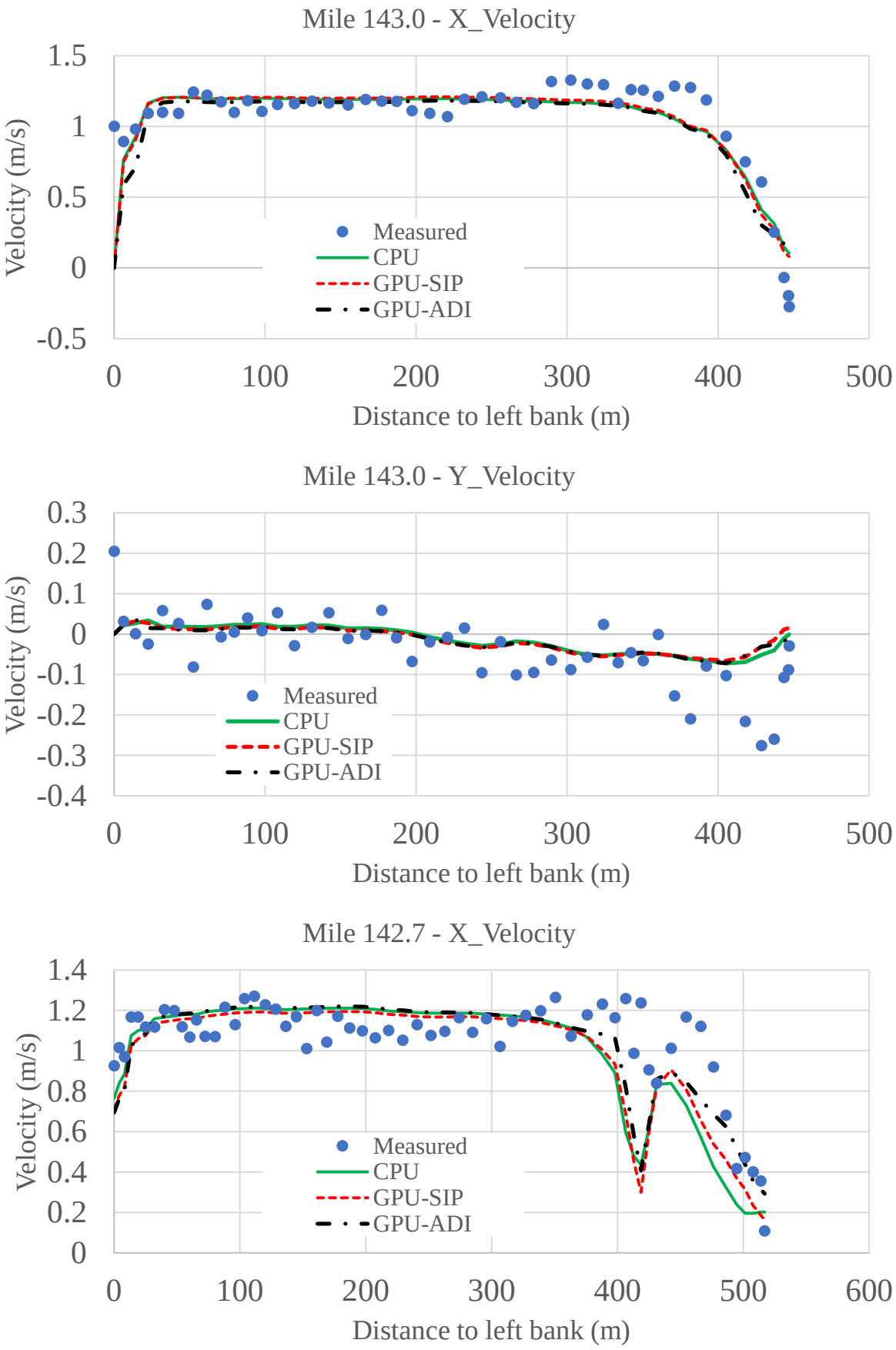


Figure 11. *Cont.*

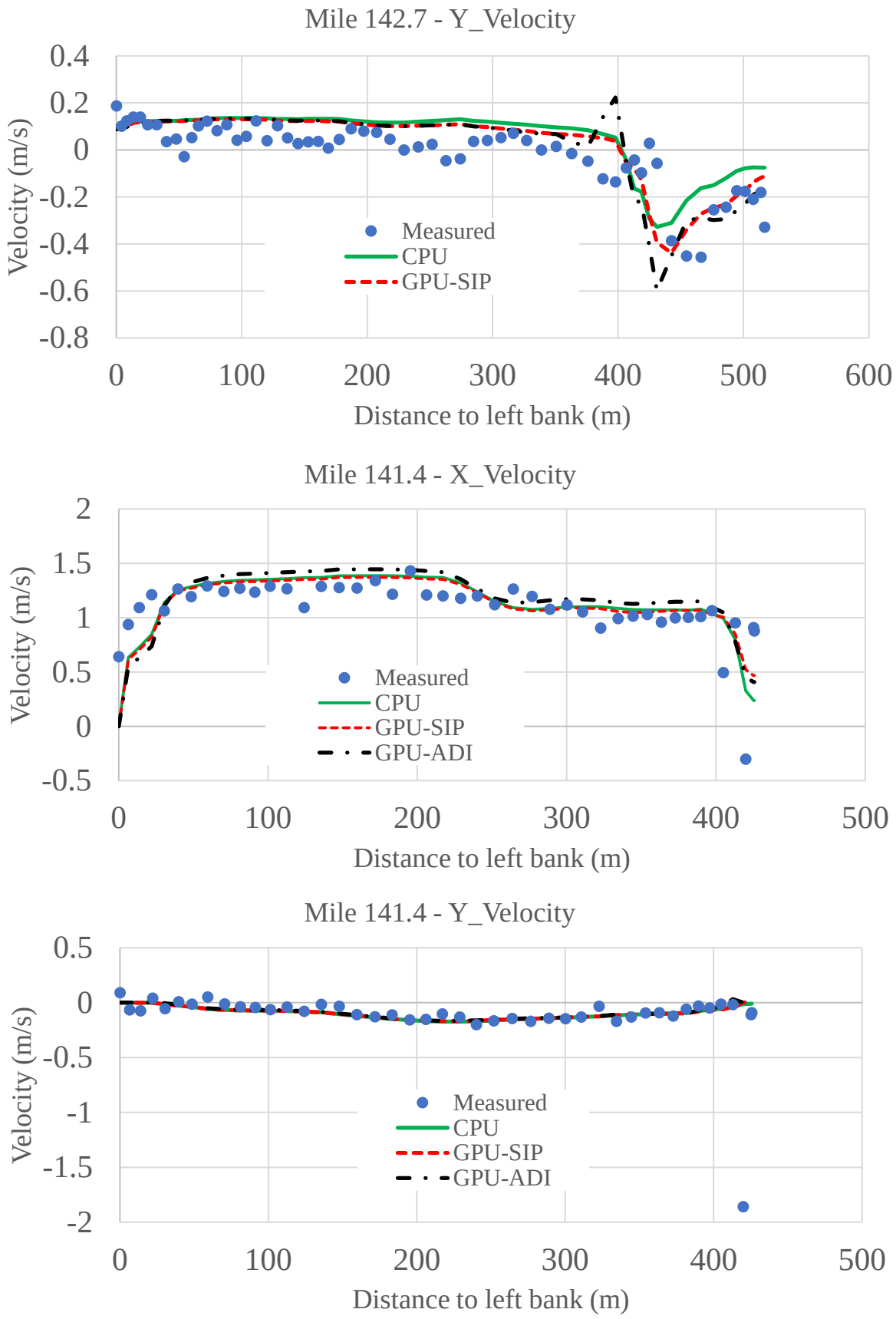


Figure 11. Velocity profiles in Arkansas River pool 7.

As can be seen, all the simulated profiles generally agreed with the measured ones. However, due to the dikes, the flow field is very complicated, which makes it difficult to accurately validate those velocity profiles. Although identical results were obtained, discrepancies between the CPU model and the parallel GPU model can be observed as well, which is expected and mainly due to (1) the different computing capabilities of the CPU and GPU, and (2) the differences between the parallelization coding and the sequential coding when implementing those numerical solvers.

The -ADI solver presented a little oscillation for Y velocity at mile 142.7, which may lie in its alternating implicit characteristics so that it is not as robust as the full implicit P-SIP solver.

7.2.2. Unsteady Flow Simulation in Lake Pontchartrain

Lake Pontchartrain (Figure 12) is located in southeastern Louisiana and serves not only as a brackish estuary connected to the Gulf of Mexico, Lake Borgne, and Lake Maurepas but also a flood release area for protecting the city of New Orleans.

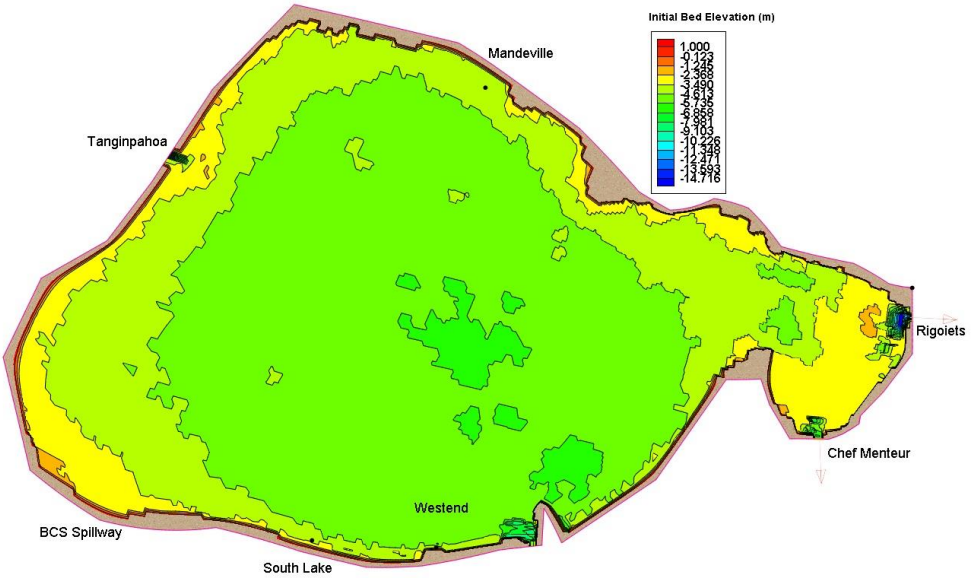


Figure 12. Lake Pontchartrain.

As shown in Figure 12, the tidal flow goes through Rigolets and Chef Menteur into the lake. The simulation results for Lake Pontchartrain on CPU and GPU are validated by using (1) the measured water surface elevation at Mandeville and Westend, and (2) the measured velocity at South Lake.

Figure 13 shows the time series profile of the water surface elevation at two different gauging stations: the Mandeville station at the north and the Westend station at the south. All simulated water level profiles captured well the general trend under tidal flow boundary conditions. The P-SIP solver yielded closer results to the CPU results than the P-ADI solver because of the same solver.

As for the velocity components at South Lake station (Figure 14), generally speaking the basic trend of the measurements was well captured by all simulated velocity profiles, despite their much more frequent fluctuations over time than the water level. Similarly, the P-SIP solver yielded closer results to the CPU results than the P-ADI solver, as observed.

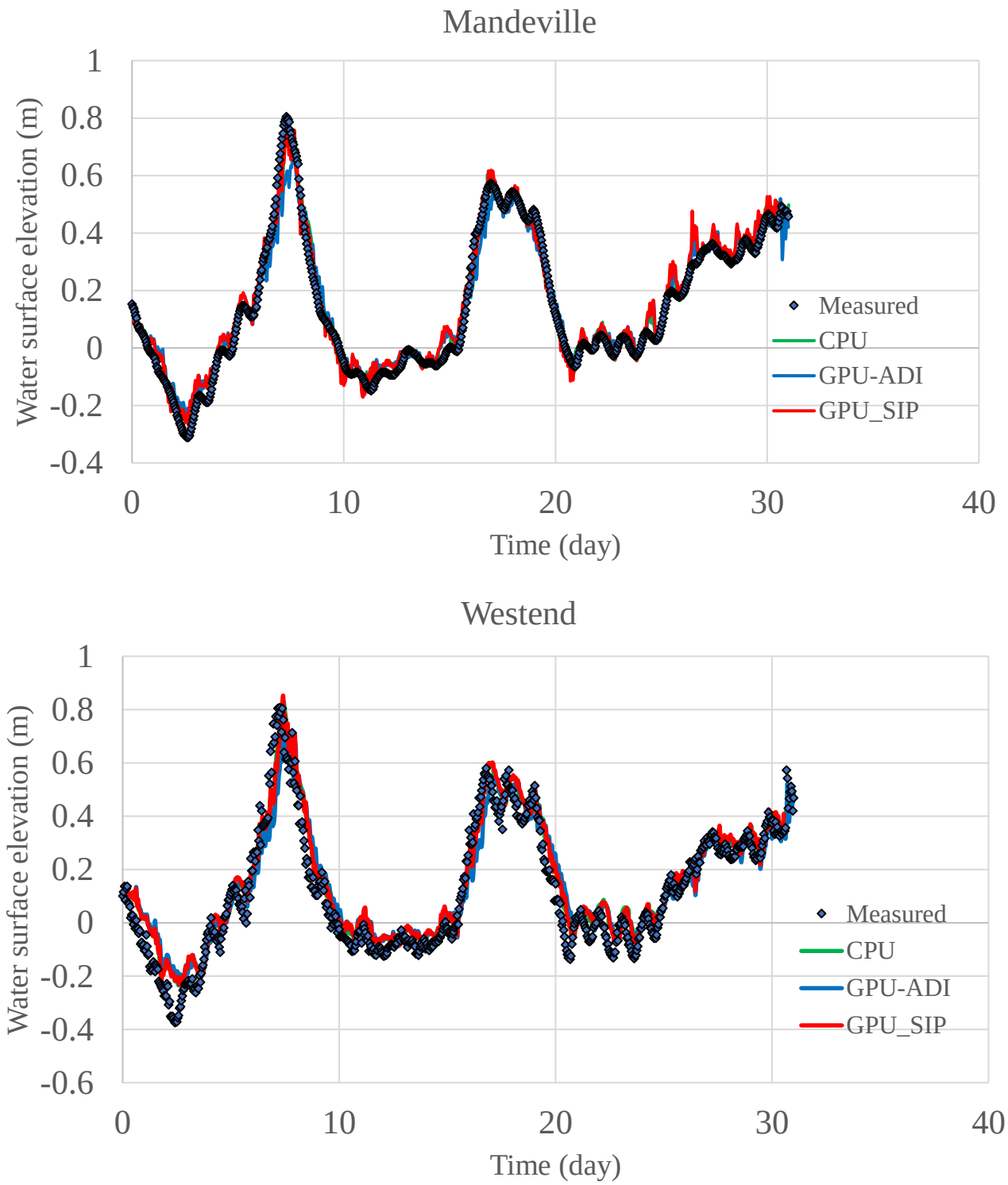


Figure 13. Water surface elevation at Mandeville and Westend of Lake Pontchartrain.

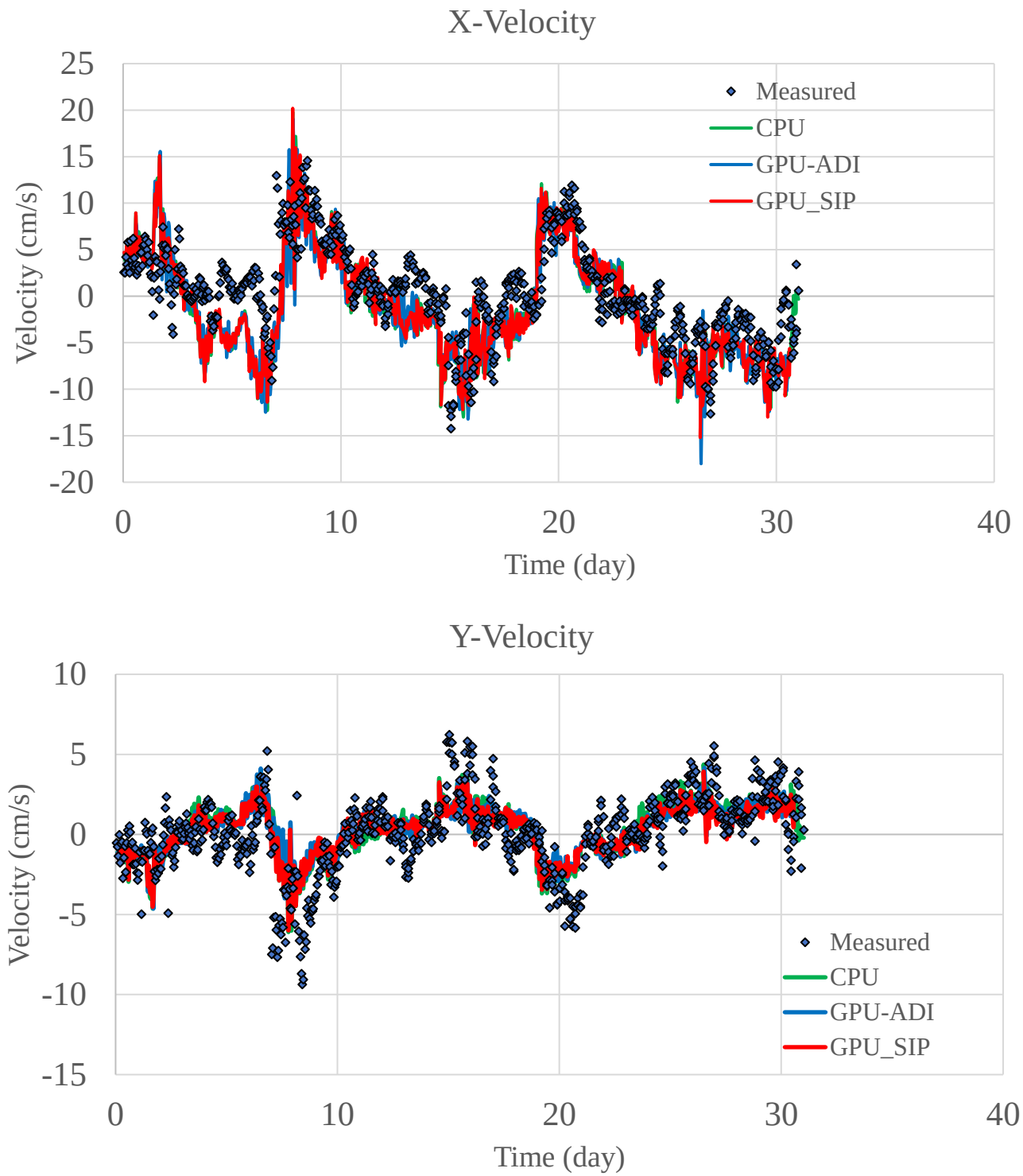


Figure 14. Velocity at South Lake of Lake Pontchartrain.

7.3. Sediment Transport Simulations

7.3.1. Steady Sediment Transport in Vistula River

These two parallel solvers (P-ADI and P-SIP) were further implemented in the sediment transport module of CCHE2D-SED. The steady sediment transport simulation in Vistula River and the unsteady sediment transport simulation in East Fork River were used to validate and test these two parallel solvers.

Vistula River is the largest river of Poland and the drainage basin of the Baltic Sea. As shown in Figure 15, there are 30 dikes in the study reach. In the sediment transport

simulation in Vistula River (Figure 15), there are seven size classes (Table 11). A constant flow discharge of 570 m^3/s, a suspended-load concentration of 11.4 kg/m^3, and a bed-load transport rate of 0.032 kg/m/s are imposed at the inlet, while at the outlet a constant water surface elevation 10.868 m is maintained. The total simulation time is 10,000 s, and the time step is 2 s.

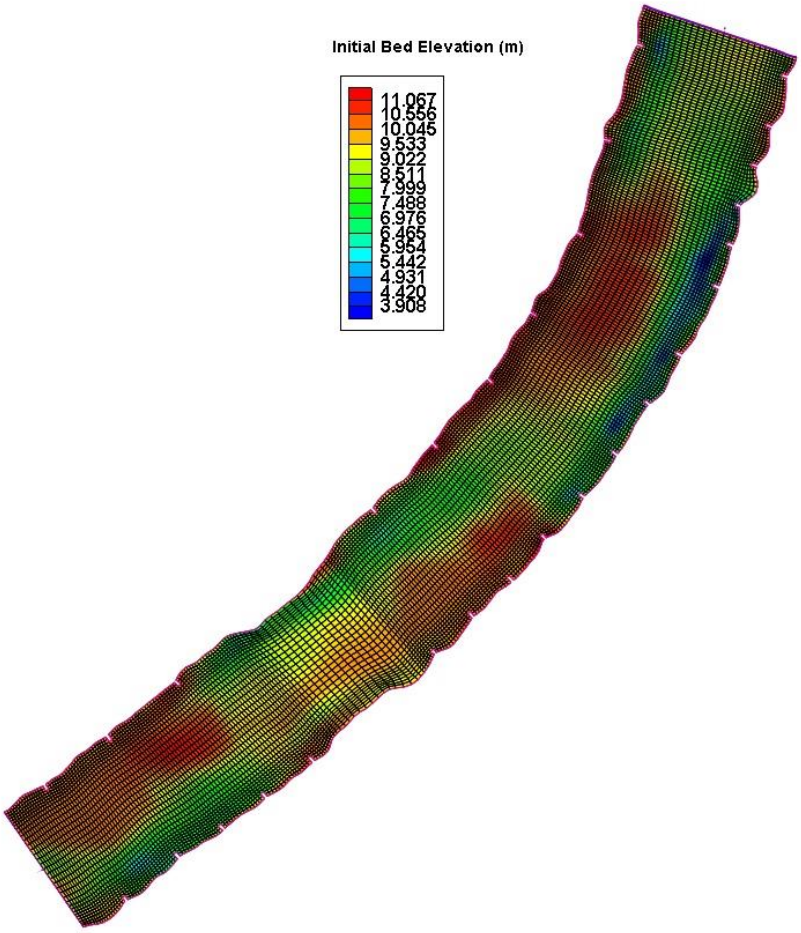


Figure 15. Structured mesh for Vistula River (34 × 339).

Table 11. Size classes in Vistula River.

Size (mm)	**0.075**	**0.1**	**0.15**	**0.25**	**0.5**	**1**	**2**
Fraction	0.054	0.06895	0.26835	0.5465	0.0459	0.01485	0.00145

Figure 16 compares the sediment fields: distributions of bed-load transport rate, suspended-load concentration, and bed change between the serial version on the CPU and the parallel version on the GPU. Identical results from both the CPU and GPU were obtained, but differences due to different computing capabilities of CPU and GPU were also observed.

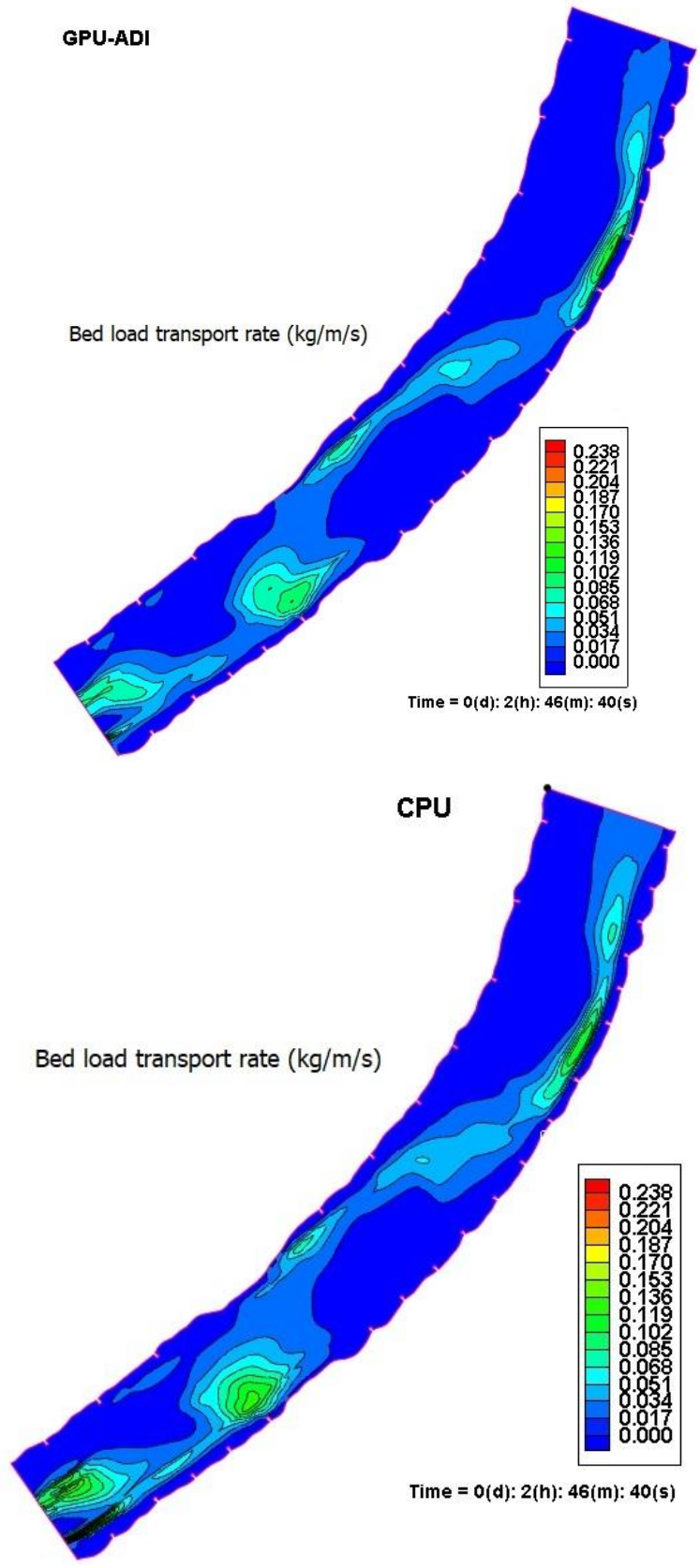
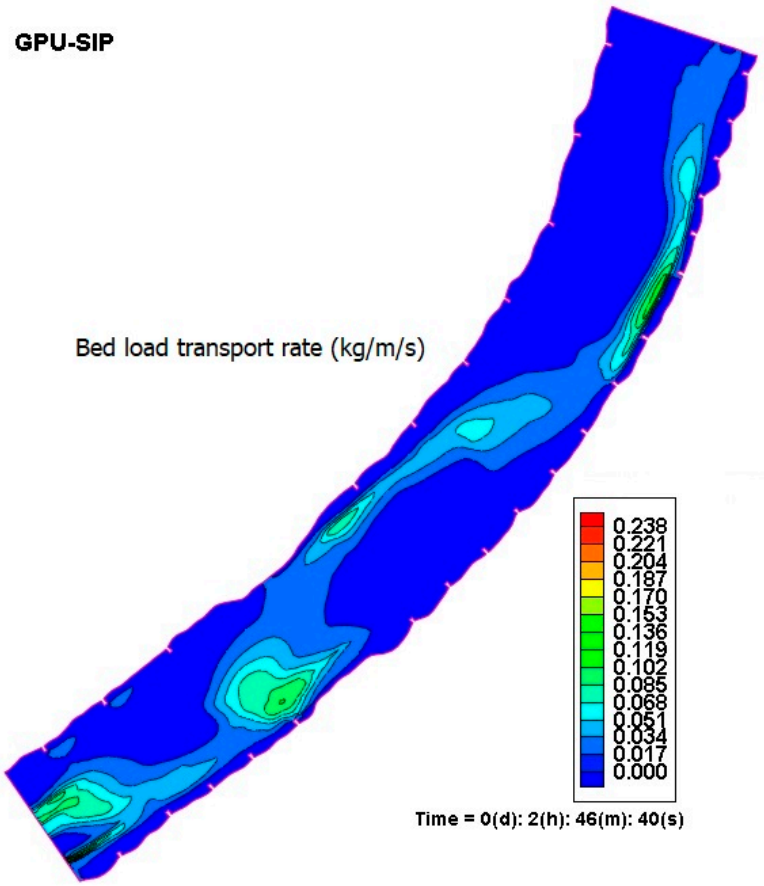


Figure 16. *Cont.*



(**a**)

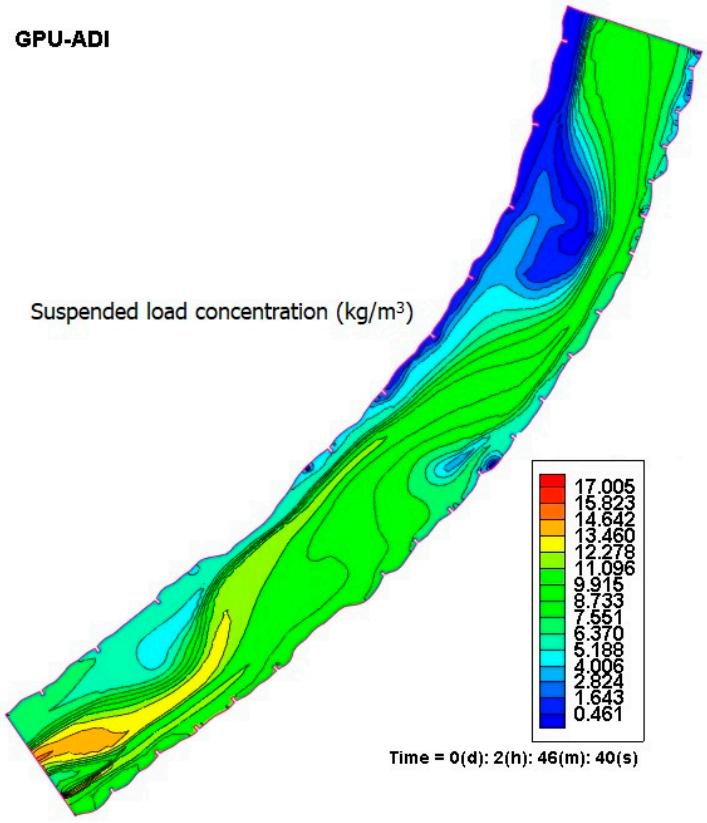
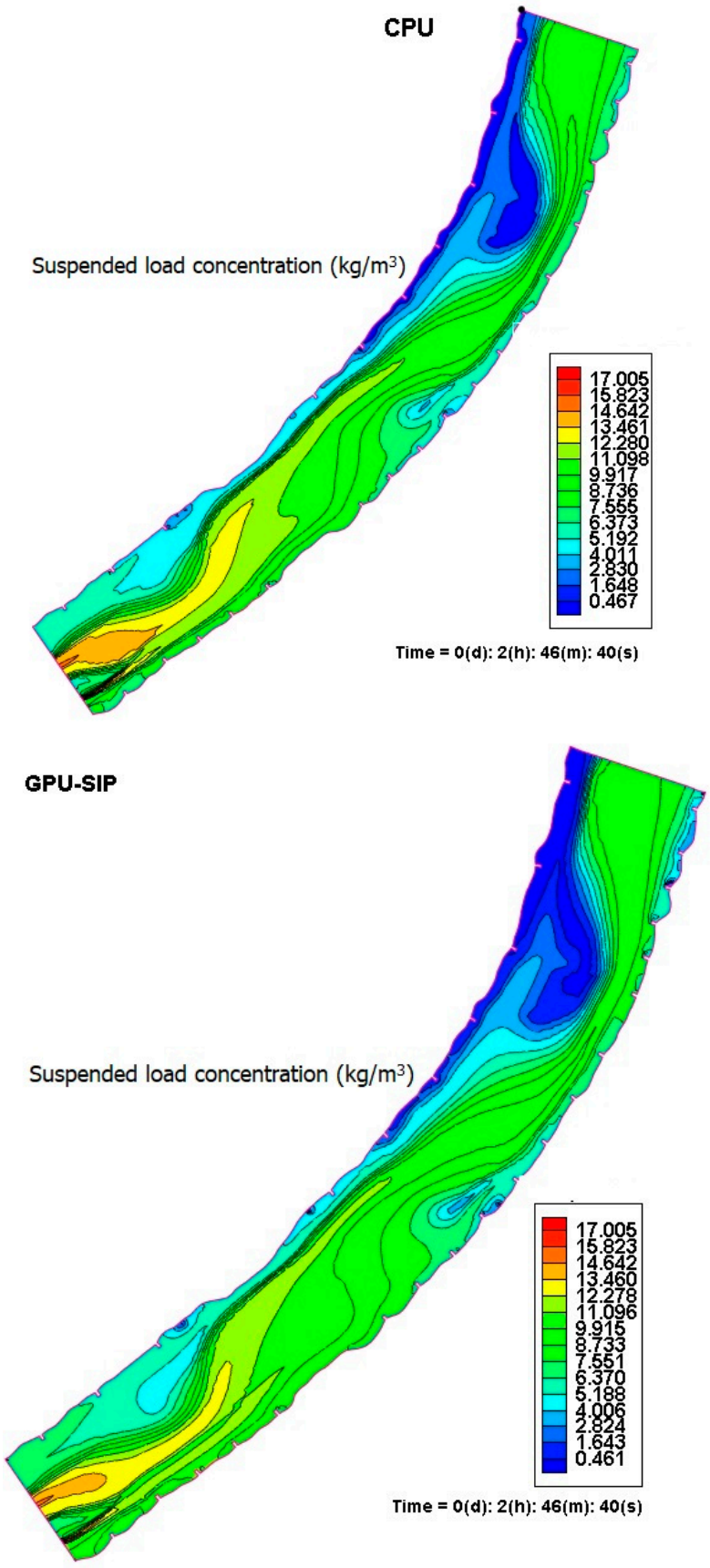


Figure 16. *Cont.*



(**b**)

Figure 16. *Cont.*

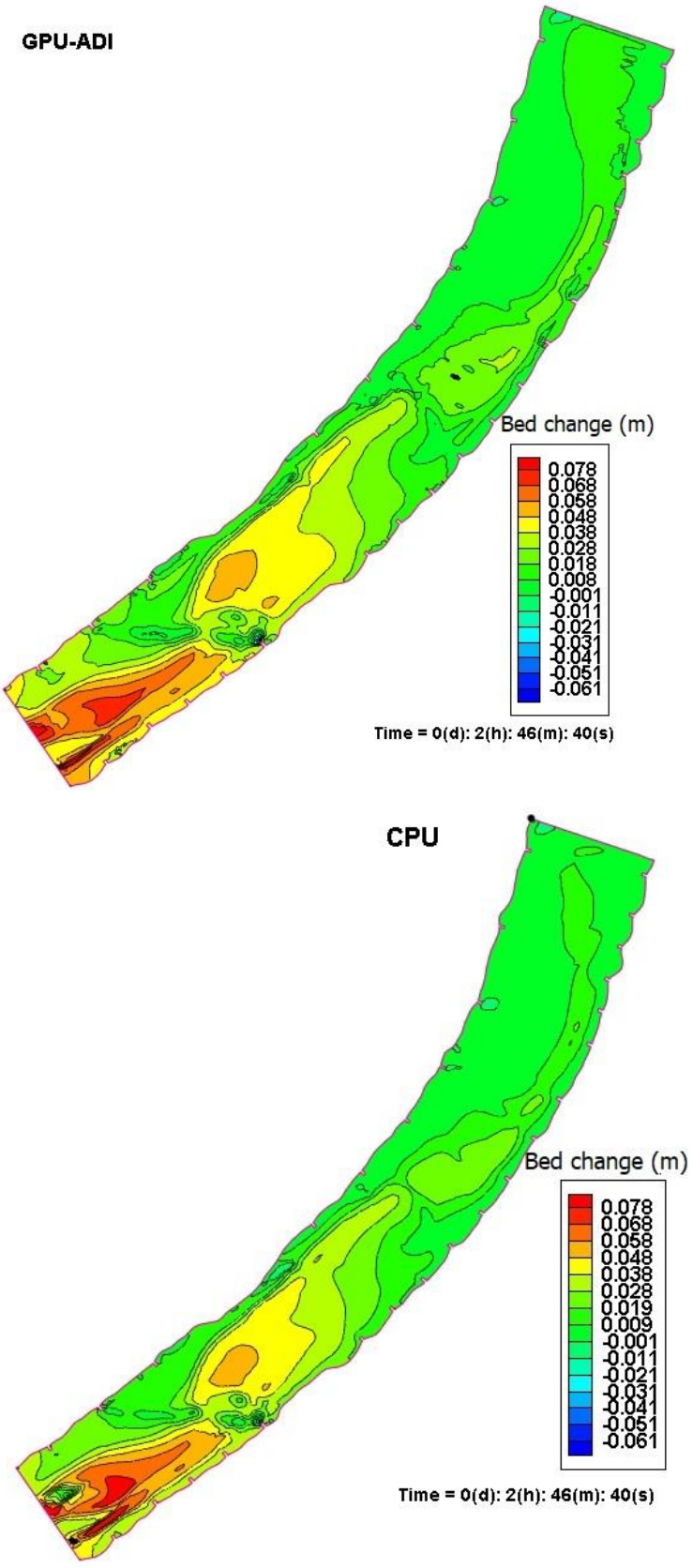
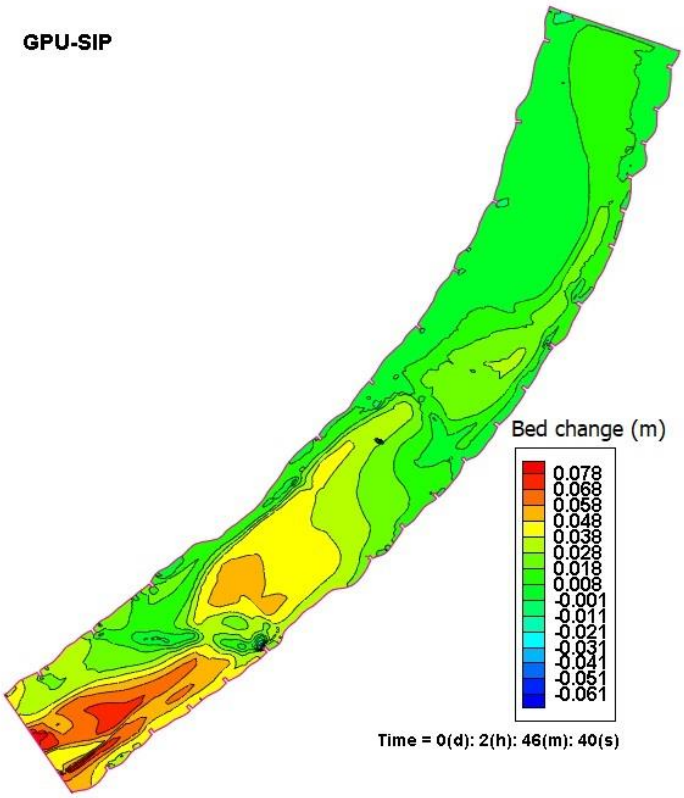


Figure 16. *Cont.*



(**c**)

Figure 16. *Cont.*

Figure 16. Sediment fields at 167 min: (**a**) bed-load transport rate, (**b**) suspended-load concentration, and (**c**) bed change.

Table 12 compares the computing efficiency of the P-ADI solver and the P-SIP solver for the steady sediment transport simulation in Vistula River. Again, note that the CPU running time listed in Table 12 includes all other parallel implementations in the entire time-marching loop. As can be seen, the P-ADI solver is about two times faster than the P-SIP solver with WF-e mapping.

Table 12. Run time for steady sediment transport simulation in Vistula River.

Solver		Mesh $I_{max} \times J_{max}$	Sim. Time (min)	Time Step (s)	CPU (min)	GPU (min)	Speed Ratio
P-ADI	OBM-it	34 × 339	167	2	12	3.315	3.620
	MBM	34 × 339	167	2	12	3.318	3.617
P-SIP	WF-e	34 × 339	167	2	12	6.716	1.787
	WF-all	34 × 339	167	2	12	5.54	2.166

7.3.2. Unsteady Sediment Transport in East Fork River

The unsteady sediment transport simulation in a 3.2 km reach of the East Fork River (Figure 17) located in Wyoming State selected the simulation period of 2–19 June 1979. Figure 18 shows the corresponding hydrograph and the stage graph at the inlet and outlet.

Figure 17. Selected reach of East Fork River. The pink line denotes the boundary of the study reach.

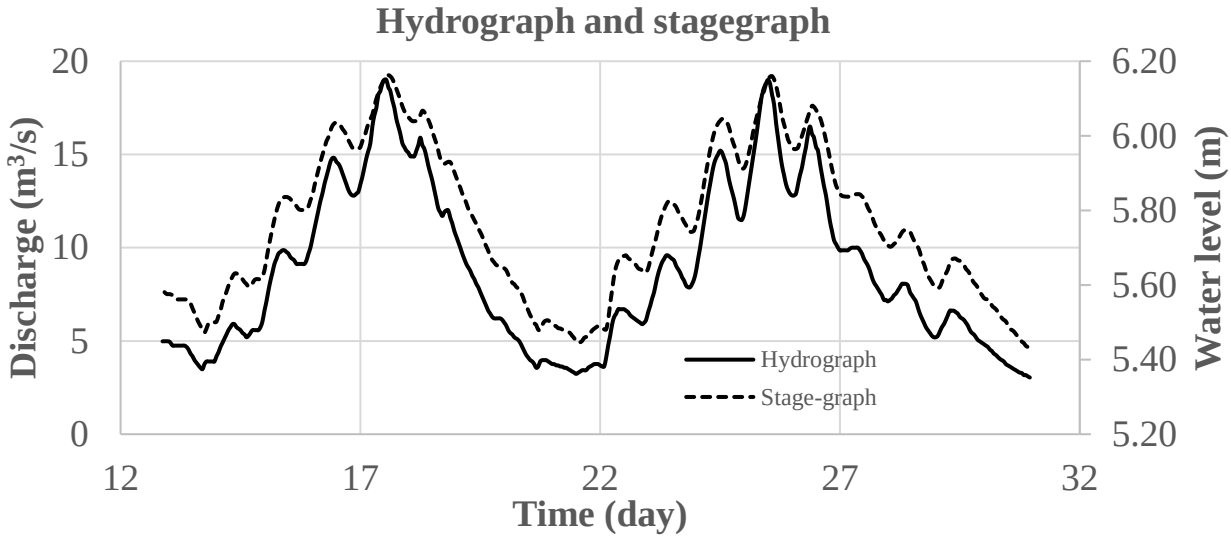


Figure 18. Hydrograph at the inlet and stage graph at the outlet.

In this case, there are nine size classes of sediment mixtures, ranging from 0.088 mm (fine sand) to 32 mm (coarse gravel). Bed sample information was obtained from the USGS technical report [42] for the selected reach. In this simulation, the total simulation time is 18 days, and the time step is 2 s

Figure 19 compares the simulated sediment flux at the bed trap of the outlet with the measurements. The general trend was well captured by all solvers, but the first peak was

over-predicted, the second peak was under-predicted, and again oscillations occurred in the P-ADI solvers. For this unsteady sediment transport simulation, more discrepancies were observed between different solvers. The reason may lie in the complexity of unsteady multi-sized sediment transport that is highly non-linear and depends not only on the solutions of those transport equations but also on empirical/semi-empirical formulas, which tends to create more uncertainties [25].

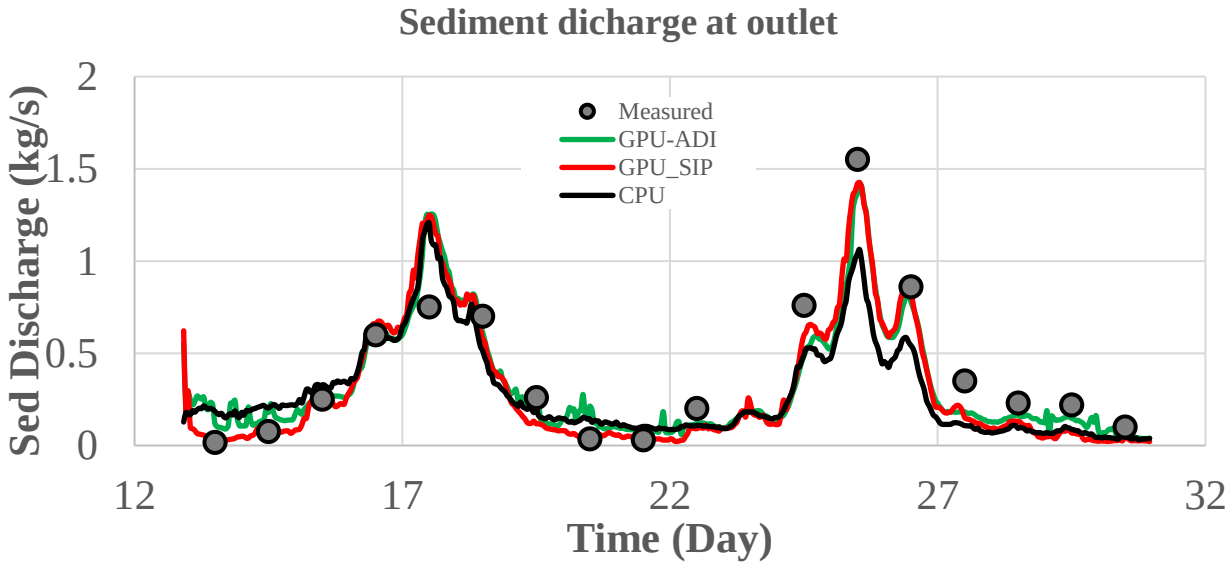


Figure 19. Sediment discharge at the outlet.

Table 13 lists the CPU time used for all solvers. Similar to the Vistula case, the P-ADI has better performances than the P-SIP solver in this unsteady sediment transport simulation as well.

Table 13. Run time for unsteady sediment transport simulation in East Fork River.

Solver		Mesh $I_{max} \times J_{max}$	Sim. Time (day)	Time Step (s)	CPU (h)	GPU (h)	Speed Ratio
P-ADI	OBM-it	33 × 501	18	2	28.5	7.812	3.648
	MBM	33 × 501	18	2	28.5	7.858	3.627
P-SIP	WF-e	33 × 501	18	2	28.5	16.997	1.68
	WF-all	33 × 501	18	2	28.5	13.457	2.118

As can be seen, in general, sediment transport simulations are more complicated with more computations than flow simulations. Correspondingly, the speedup ratio decreases for both parallel solvers.

8. Conclusions

Two widely-used implicit solvers for structured meshes, namely, the ADI solver and the SIP solver, were parallelized and integrated into a 2D numerical model, the CCHE2D, for comparison. For each parallel solver, two mapping methods were implemented to map 2D problems (I_{max}, J_{max}) onto the hierarchical block-thread frame of the CUDA on the GPU: the OBM-it (one-block-thread mapping with iterations) and MBM (multi-block-threads mapping) for the P-ADI solver and WF-e mapping (matrix coefficients and solution variables defined on the original mesh) and WF-all mapping (matrix coefficients and solution variables defined on WF mesh) for the P-SIP solver. According to numerical tests, the following conclusions can be drawn:

(1) Both the P-ADI solver and P-SIP solver were validated by example cases with complex geometries and produced identical results to those of the serial version on the CPU, despite the different computing capabilities of the CPU and GPU;
(2) In general, the P-ADI solver was faster than the P-SIP solver due to its lighter computing loads. The respective speedup ratio of the P-ADI and P-SIP was up to 7.89 and 5.42 for simple 2D heat-diffusion problems, 4.98 and 3.73 for flow simulations, and 3.648 and 2.166 for sediment transport simulations. With increasing complexities of problems, the speedup ratio also decreased;
(3) In the P-ADI solver, both the OBM-it and the MBM had close performances in computing efficiency, and the OBM-it was slightly faster than the MBM. Most of time, the difference was less than 5%. In the 2D heat-diffusion problem, the OBM-it was 13% faster than the MBM;
(4) In the P-SIP solver, the newly proposed WF-all was significantly faster than the conventional WF-e: 3.67 times faster in 2D heat-diffusion problem, 1.47~2.03 times faster in flow simulations, and 1.2 times faster in sediment transport simulations;
(5) The P-ADI solver was not as robust as the P-SIP solver due to its alternating implicit characteristics, and it may suffer from oscillations in complicated simulations, such as unsteady sediment transport in domains with wetting-and-drying processes.

Author Contributions: Conceptualization, Y.Z.; methodology, Y.Z.; validation, Y.Z. and X.C.; writing—original draft preparation, Y.Z.; writing—review and editing, M.Z.A.-H. and X.C.; project administration, M.Z.A.-H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the U.S. Department of Agriculture, Agricultural Research Service, through the Cooperative Agreement No. 6060-13000-030-00D between the USDA-ARS National Sedimentation Laboratory (USDA-ARS-NSL) and the University of Mississippi/National Center for Computational Hydroscience and Engineering (NCCHE).

Data Availability Statement: All data in the examples and application are available upon request.

Acknowledgments: The authors would like to thank Ronald Bingner from the USDA Agricultural Research Service (ARS), National Sedimentation Laboratory (NSL) for his help and support in this study.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Nomenclature

A	matrix coefficient of the assembled algebraic equation
C_k	Suspended-load concentration in volume for k-th sediment class
C_{*k}	transport capacity of suspended load for k-th sediment class
g	gravity acceleration (ms^{-2})
h	water depth (m)
n	Manning's n (m)
q	cross-edge discharge (m^2s^{-1})
q_{bk}	bed-load transport rate for kth sediment class (m^2s^{-1})
q_{b*k}	transport capacity of bed-load for kth sediment class (m^2s^{-1})
u	x velocity (ms^{-1})
v	y velocity (ms^{-1})
η	water surface elevation (m)
ρ	water density (kgm^{-3})
ϕ	dependent variable
ν	kinematic viscosity (m^2s^{-1})
p'	the porosity of bed material
ω_{sk}	the sediment settling velocity for kth sediment class (ms^{-1})
τ_b	bed shear stress (Pa)
τ_w	wind shear stress (Pa)
ε_s	the eddy diffusivity of sediment (m^2s^{-1})

References

1. Cea, L.; Bladé, E. A simple and efficient unstructured finite volume scheme for solving the shallow water equations in overland flow applications. *Water Resour. Res.* **2015**, *51*, 5464–5486. [CrossRef]
2. Xia, X.; Liang, Q.; Ming, X.; Hou, J. An efficient and stable hydrodynamic model with novel source term discretization schemes for overland flow and flood simulations. *Water Resour. Res.* **2017**, *53*, 3730–3759. [CrossRef]
3. Zhang, Y.; Al-Hamdan, M.; Bingner, R.; Chao, X.; Langendoen, E.; O'Reilly, A.; Vieria, D.A. Application of 1D Model for overland flow simulations on 2D complex domains. *Adv. Water Resour.* **2024**, *188*, 104711. [CrossRef]
4. Zhang, Y.; Al-Hamdan, M.; Bingner, R.; Chao, X.; Langendoen, E.; Vieria, D.A. Generation of 1D Channel Networks for overland flow simulations on 2D complex domains. *J. Hydrol.* **2024**, *628*, 130560. [CrossRef]
5. Panchigar, D.; Kar, K.; Shukla, S.; Mathew, R.M.; Chadha, U.; Selvaraj, S.K. Machine learning-based CFD simulations: A review, models, open threats, and future tactics. *Neural Comput. Appl.* **2022**, *34*, 21677–21700. [CrossRef]
6. Lee, C.H.; Cant, S. A grid-induced and physics-informed machine learning CFD framework for turbulent flows. *Flow Turbul. Combust.* **2024**, *112*, 407–442. [CrossRef]
7. Chen, Y.C.; Wnag, Z.Y.; Liu, Z.W.; Zhu, D.J. 1D-2D Coupled Numerical Model for Shallow-Water Flows. *ASCE J. Hydraul. Eng.* **2012**, *138*, 122–132. [CrossRef]
8. Brunner, G.W. *HEC-RAS- River Analysis System 2D Modeling Users' Manual*; US Army Corps of Engineering, Institute for Water Resources, Hydrologic Engineering Center: Davis, CA, USA, 2016.
9. Zhang, Y.; Jia, Y. Towards Efficient Modeling. In Proceedings of the EWRI World Environment & Water Resources Congress 2017, Sacramento, CA, USA, 21–25 May 2017.
10. Zhang, Y.X.; Jia, Y.F.; Wang, S.S.Y. A Conservative Multi-block Algorithm for Two-dimensional Numerical Model. *Int. J. Math. Sci.* **2007**, *1*, 100–112.
11. Navon, I.M.; Cai, Y. Domain decomposition and parallel processing of finite element model of the shallow water equations. *Comput. Methods Appl. Mech. Eng.* **1993**, *106*, 179–212. [CrossRef]
12. Thibault, J.C.; Senocak, I. CUDA Implementation of Navier-Stokes Solver on Multi-GPU Desktop Platforms for Imcompressible Flows. In Proceedings of the 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, FL, USA, 5–8 January 2009.
13. Cohen, J.M.; Molemaker, M.J. A fast double precision CFD Code using CUDA. In Proceedings of the 21st International Conference on Parallel Computational Fluid Dynamics, Parallel CFD 2009, Moffett, CA, USA, 18–22 May 2009.
14. Corrigan, A.; Camelli, F.; Lohner, R.; Wallin, J. Running Unstructured Grid based CFD solvers on Modern Graphics hardware. In Proceedings of the 19th AIAA CFD, San Antomio, TX, USA, 22–25 June 2009.
15. Brodtkorb, A.R.; Sætra, M.L.; Altinakar, M.S. Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation. *Comput. Fuids* **2011**, *55*, 1–12. [CrossRef]
16. Zhang, Y.; Jia, Y. Parallelized CCHE2D Flow Model with CUDA Fortran on Graphics Process Units. *Comput. Fluids* **2013**, *84*, 359–368. [CrossRef]
17. Zhang, Y.; Jia, Y. Parallelization of Implicit CCHE2D Model using CUDA Programming Techniques. In Proceedings of the EWRI World Environment & Water Resources Congress 2013, Cincinnati, OH, USA, 19–23 May 2013.
18. Wei, Z.; Jang, B.; Zhang, Y.; Jia, Y. Parallelizing Alternating Direction Implicit Solver on GPUs. In Proceedings of the International Conference on Computer Science, ICCS 2013, Barcelona, Spain, 5–7 June 2013.
19. Lacasta, A.; Morales-Hernandez, M.; Murillo, J.; Garcı´a-Navarro, P. GPU implementation of the 2D shallow water equations for the simulation of rainfall/runoff events. *Environ. Earth Sci.* **2015**, *74*, 7295–7305. [CrossRef]
20. Hou, J.M.; Kang, Y.D.; Hu, C.H.; Tong, Y.; Pan, B.Z.; Xia, J.Q. A GPU-based numerical model coupling hydrodynamical and morphological processes. *Int. J. Sediment Res.* **2020**, *35*, 386–394. [CrossRef]
21. Jia, Y.F.; Wang, S.S.-Y. Numerical Model for channel flow and morphological changes studies. *ASCE J. Hydraulic Engrg.* **1999**, *125*, 924–933. [CrossRef]
22. Deltares. Delft3D-FLOW. In *Simulation of Multi-Dimensional Hydrodynamic Flow and Transport Phenomena, Including Sediments–User Manual*; Version 3.04, rev. 11114; Deltares: Delft, The Netherlands, 2010.
23. MIKE 21 Flow Model FM Hydrodynamic Module, Users' Guide. 2014. Available online: https://manuals.mikepoweredbydhi.help/2017/Coast_and_Sea/MIKE_FM_HD_2D.pdf (accessed on 20 June 2024).
24. Lai, Y. *SRH-2D Users' Manual: Sediment Transport and Mobile-Bed Modeling*; US Department of the Interior, Bureau of Reclamation: Denver, CO, USA, 2020.
25. Zhang, Y.; Al-Hamdan, M.; Wren, D. Development of a Two-Dimensional Hybrid Sediment-Transport Model. *Appl. Sci.* **2023**, *13*, 4940. [CrossRef]
26. Thomas, L.H. *Elliptic Problems in Linear Differential Equations over a Network: Watson Scientific Computing Laboratory*; Columbia University: New York, NY, USA, 1949.
27. Hockney, R.W. A fast direct solution of Poisson's equation using Fourier analysis. *J. ACM* **1965**, *12*, 95–113. [CrossRef]
28. Stone, H.L. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.* **1968**, *5*, 87–113. [CrossRef]
29. Hockney, R.W.; Jesshope, C.R. *Parallel Computers*; Adam Hilger: Bristol, UK, 1981.

30. Sakharnykh, N. Tridiagonal solvers on the GPU and applications to fluid simulation. In Proceedings of the GPU Technology Conference, San Jose, CA, USA, 30 September–3 October 2009.
31. Zhang, Y.; Cohen, J.; Owens, J.D. Fast Tridiagonal Solver on the GPU. In Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP), India, Bangalore, 9–14 January 2010.
32. Reeve, J.S.; Scurr, A.D.; Merlin, J.H. Parallel Version of Stone's Strong Implicit Algorithm. *Concurr. Comput. Pract. Exp.* **2001**, *13*, 1049–1062. [CrossRef]
33. Deserno, F.; Hager, G.; Brechtefeld, F.; Wellein, G. *Basic Optimization Strategies for CFD-Codes*; Technical Report; Regional Data Center Erlangen (RRZE): Erlangen, Germany, 2002.
34. Igounet, P.; Alfaro, P.; Pedemonte, M.; Ezzatti, P. A GPU implementation of the SIP method. In Proceedings of the 30th International Conference of the Chilean Computer Science Society, Curico, Chile, 9–11 November 2011. [CrossRef]
35. Dufrechou, E.; Ezzatti, P.; Usera, G. Avoiding synchronization to accelerate a CFD solver in GPU. In Proceedings of the 31st International Symposium on Computer Architecture and High-Performance Computing, Campo Grande, Brazil, 15–18 October 2019. [CrossRef]
36. The Portland Group. *CUDA Fortran Programming Guide and References*; NVIDIA Corp.: Santa Clara, CA, USA, 2011.
37. Rodi, W. *Turbulence Models and Their Applications in Hydraulics*, 3rd ed.; IAHR Monograph: Rotterdam, The Netherlands, 1993.
38. Wu, W.M. *Computational River Dynamics*; Taylor & Francis: London, UK, 2007.
39. Igounet, P.; Alfaro, P.; Pedemonte, M.; Ezzatti, P. GPU acceleration of the caffa3d.MB Model. In Proceedings of the ICCSA 2012, Salvador de Bahia, Brazil, 18–21 June 2012; Part IV, LNCS 7336. pp. 530–542.
40. NVIDIA. *CUDA C Programming Guide Version 3.1.1*; NVIDIA Corp.: Santa Clara, CA, USA, 2010.
41. Heng, K.S. *Parallel Alternating Direction Implicit Solver for the Two-Dimensional Heat Diffusion Problem on Graphics Processing Units*; National University of Singapore: Singapore, 2011.
42. Meade, R.H.; Myrick, R.M.; Emmett, W.W. *Field Data Describing the Movement and Storage of Sediment in the East Fork River, Wyoming, Part II Bed Elevations*; 1979, USGS Open-File Rep. 80-1190; United States Geological Survey (USGS): Denver, CO, USA, 1980.