Fully GPU resident wavelet-based adaptive gridding: application to two-dimensional finite volume hydrodynamic modelling

Alovya Ahmed Chowdhury¹, Georges Kesserwani¹ *, Charles Rougé¹, Paul Richmond²

¹Department of Civil and Structural Engineering, University of Sheffield, Mappin St, Sheffield City Centre, Sheffield, UK
²Department of Computer Science, University of Sheffield, Mappin St, Sheffield City Centre, Sheffield, UK

Abstract

First order finite volume (FV1) models that use uniform grids are often used in computational engineering, but may become prohibitively costly to run on a fine resolution and/or large areas. To reduce these costs, FV1 models have adopted adaptive gridding or parallelisation on graphics processing units (GPU). FV1 models that combine adaptive gridding and parallelisation usually generate the adaptive grid on the central processing unit (CPU), yielding extra costs for data transfer between the CPU and the GPU. This paper presents a computational innovation that avoids these costs by enabling GPU resident adaptive gridding, based on the multiresolution analysis (MRA) of Haar wavelets (HWs). It combines the indexing of Z order curves, to ensure coalesced access of GPU memory, and a newly adopted Parallel Tree Traversal (PTT) that minimises warp divergence of GPU threads. The resulting GPU resident adaptive gridding method is presented as part of a parallelised, HWFV1 hydrodynamic model (GPU-HWFV1). The model’s runtime performance is benchmarked against its CPU predecessor (CPU-HWFV1) and a GPU-FV1 uniform grid model for a range of test cases ran on the finest resolution grid accessible to the HWFV1 models. Tests demonstrate the robustness of the results. As for runtime performance, GPU-HWFV1 is up to 400x faster than CPU-HWFV1, while remaining 30x faster than GPU-FV1 especially in applications that require increased depth in the grid resolution and high sensitivity to resolution refinement. The findings are significant, making a strong case for applying the proposed GPU resident adaptive gridding method to further speed-up FV1 models.

Keywords: Adaptive mesh refinement; Multiresolution analysis; GPU computing; computational efficiency assessments; hydraulic modelling.
1. Introduction

Computational hydrodynamic models are widely used to support hydraulic engineering applications [1–3]. Most operational hydrodynamic models adopt a finite volume formulation for numerically solving the two-dimensional (2D) shallow water equations [4,5]. In many applications, the scale difference between the smallest spatial features and the domain size leads to extremely large and dense uniform grids, thus potentially prohibitive runtimes [6–12]. To accelerate runtimes, efficiency enhancement measures have become common with hydrodynamic and other models, such as adaptive gridding [13–15] or parallel computing [12,16,17].

By adopting an adaptive gridding method, a hydrodynamic model can gain performance by using coarser cells at grid locations where the modelled features of interest are insignificant, instead of retaining fine resolution all over the grid [18]. However, adaptive gridding methods entail the need for context-specific modifications to ensure robustness across heterogeneously sized cells, such as ensuring mass conservation or employing a well-balanced discretisation between the topography and flux gradients with different wetting and drying conditions [19–22]. Adaptive wavelet-based hydrodynamic models can preserve similar results to their uniform grid model counterparts without needing context-specific modifications. This is because these models inherently achieve adaptive gridding when constructing an (updatable) flow solution using the multiresolution analysis (MRA) of wavelets [14,23–27]. This MRA process allows to encode (coarsen), decode (refine) and rigorously reconstruct both the flow solution (time-variant) and the topography representation (time-invariant) across a hierarchy of 2D grids at various resolutions. Wavelet-based adaptive hydrodynamic models are found to be up to 20× faster than their reference models run on uniform grids for simulations on central processing units (CPUs) [27]. However, wavelet-based adaptive gridding can still lead to significant runtime costs for hydraulic modelling applications over large 2D spatial areas with the overhead costs of the MRA process being a large factor in the overall runtime [26].

An alternative for reducing runtime costs is to use the parallel computing architecture of graphical processing units (GPU). Parallelised finite volume models running on 2D fixed grids have
become mature and include models parallelised on CPUs (e.g. [17,28]), on GPUs accessing thousands of computing cores (e.g. [12,16,29–31]) and on a combination of CPUs and GPUs (e.g. [32,33]). At present, GPU-accelerated finite volume models on fixed grids are the de-facto of operational hydraulic modelling packages [34–38] because cells in fixed grids can be conveniently indexed using unique pairs of integers, e.g. \([i, j]\), and it is easy to ensure coalesced memory access, i.e. access of contiguous memory locations by adjacent threads [39]. Such convenient indexing is not possible when using adaptive grids, making it more difficult to parallelise adaptive gridding methods. Existing finite volume models that combine adaptive gridding and parallelisation are only partially GPU resident, in the sense that the adaptive gridding is performed on the CPU [40–42]. This increases the models’ runtime costs because data must be transferred between the CPU and the GPU every time step [40,43].

This paper is motivated by the rationale that such costs could be avoided in wavelet-based hydrodynamic models because MRA operates on a hierarchy of uniform 2D grids. As each of these grids is uniform, the adaptive gridding afforded by MRA could be parallelised in a way ensuring coalesced memory access. The challenge in achieving this is devising an innovation that is able to rework the recursive nature of the MRA process, namely the encoding/decoding operations and the process of identifying the cells making up the adaptive grid, into a form suitable for parallelisation.

To address this challenge, this paper presents such a computational innovation not yet applied to finite volume models, namely the simultaneous adoption of a specialised way of indexing cells across resolutions and a tree traversal approach on the GPU. This allows to present a GPU resident adaptive gridding method based on Haar wavelets (HWs) as part of a state-of-the-art first-order finite volume (FV1) hydrodynamic model (GPU-HWFV1). The theories underpinning the HWFV1 algorithm are reported in previously with extensive benchmarking of its serial implementation on the CPU (CPU-HWFV1) for one-dimensional (1D) and 2D hydraulic modelling test cases [14,26,27]. The innovation presented in this paper is needed to make the runtimes of adaptive wavelet-based models competitive with GPU parallelised finite volume models used for real-world hydraulic
modelling applications, such as FV1 solver of the LISFLOOD-FP 8.0 software package, referred hereafter to as GPU-FV1 [12].

The rest of this paper is as follows. In Section 2, the HWFV1 algorithm is briefly presented focusing on the recursive nature of encoding and decoding in its MRA process over the hierarchy of 2D grids. Z order curves are used to index the cells residing in the hierarchy, and a new Parallel Tree Traversal (PTT) approach is adopted to identify the indices of the cells making up the generated 2D adaptive grid. In Section 3, the GPU resident HWFV1 hydrodynamic model (GPU-HWFV1) is verified, and its runtime performance assessed versus CPU-HWFV1 and GPU-FV1 on a uniform grid using the finest resolution accessed by the HWFV1 models. In Section 4, conclusions are drawn on the potential benefits of GPU-HWFV1 for modelling applications and on when it can be a faster alternative to GPU-FV1 modelling for hydraulic simulations.

2. GPU resident adaptive gridding

This section presents a computational innovation that involves the simultaneous adoption of specialised cell indexing across resolutions and a tree traversal approach on the GPU. This innovation enables parallelisation of the adaptive gridding driven by the MRA process of the HWs within the HWFV1 algorithm. To understand why this is so, Section 2.1 first gives an overview of the algorithm. Section 2.2 then describes how parallelisation of the algorithm is facilitated by indexing cells across resolutions using Z order curves (Section 2.2.1), and by adopting a Parallel Tree Traversal approach to achieve tree traversal on the GPU (Section 2.2.2).

2.1. Overview of the HWFV1 algorithm

The HWFV1 algorithm runs simulations on a 2D adaptive grid achieved by coupling an FV1 scheme with the MRA of HWs [14,27]. In hydrodynamic simulations, the FV1 scheme is obtained from a discrete translation of the conservative form of the 2D depth-averaged shallow water equations in a Godunov-type framework with treatments of source terms with wetting and drying to support real-world applications [5]. In this context, the flow variables are $h(x,y,t)$ standing for the water depth (m)
at a location \((x,y)\) and time \(t\), and \(u(x,y,t)\) and \(v(x,y,t)\) the components of the velocity field \((\text{m/s})\).

The bed height \(z(x,y)\) is involved in terms of bed slope source terms and the friction source terms involve the Manning’s roughness parameter, \(n_M\), assumed constant in this presentation for simplicity.

In the FV1 scheme, flow variables \((h, hu \text{ and } hv)\) and bed height \((z)\) are represented as piecewise-constant data over each cell on any grid. These quantities will be referred to as the modelled data, each of which will be denoted by a scalar \(s \in \{h, hu, hv, z\}\) for convenience of presentation.

The principle behind the application of the MRA process is to generate, at every timestep, a 2D adaptive grid made up of non-uniformly sized cells. This 2D adaptive grid is generated by analysing the modelled data \(s\) over a hierarchy of uniform grids of increasingly coarser resolution to select cells at resolution levels appropriate to capture the spatiotemporal dynamics of \(s\). The coarsest grid on the hierarchy is made of a single cell and the higher resolution grids comprise \(2^n \times 2^n\) cells with \(n = 0, 1, \ldots, L\), where \(L\) is a user-specified maximum refinement level leading to the finest grid.

The cornerstone of the MRA process of the HWs is to produce the data and identify the indices of the relevant cells within the hierarchy that makes up the 2D adaptive grid. These cells will be referred to as “leaf cells” on which a cell-wise FV1 scheme update is applied. A full description of the HWs and FV1 scheme operations involved in the HWFV1 algorithm for a serial CPU implementation (CPU-HWFV1) is available in [26]. Here, the HWFV1 algorithm is described with a
focus on how the operations in the MRA process can be parallelised to make the algorithm GPU resident (GPU-HWFV1). Without loss of generality, the operations of the MRA process will be presented for a hierarchy of 2D grids up to \( L = 2 \), as shown in Fig. 1.

The MRA process starts by accessing the modelled data \( s \), initially available at the cells on the finest resolution grid (Fig. 1a). From this finest resolution data, data at the coarser resolution grids are produced by encoding in descending order, level by level, over the hierarchy (Fig. 1a). As encoding is performed recursively for all the cells across two consecutive grids, one “fine” and one “coarse”, it suffices to explain it for an operation for extracting the parent (coarse) cell data from the child (fine) cell data (Fig. 1b). An operation of encoding is described in Eq. 1a where the parent cell data, denoted by \( s_{\text{coarse}} \), is produced from its four child cells’ data, denoted by \( s_{\text{fine}}^{[0]} \), \( s_{\text{fine}}^{[1]} \), \( s_{\text{fine}}^{[2]} \), and \( s_{\text{fine}}^{[3]} \). Encoding also allows the extraction of the difference between the child cells’ data and the parent cell data (Fig. 1b), in terms of coefficient of “details”, \( d_H^{\text{coarse}} \), \( d_V^{\text{coarse}} \), and \( d_D^{\text{coarse}} \), through the operations in Eqs. (1b - 1d). The scalar coefficients \( H^0 \), \( H^1 \), \( G^0 \) and \( G^1 \) in Eq. 1 are filter banks derived from the scale-dependent decomposition of the HWs basis from its scaling basis [14,44]. The details \( d_H^{\text{coarse}} \), \( d_V^{\text{coarse}} \), and \( d_D^{\text{coarse}} \) represent the encoded difference along the vertical, horizontal, and diagonal directions, respectively.

\[
\begin{align*}
  s_{\text{coarse}} &= H^0 \left( H^0 s_{\text{fine}}^{[0]} + H^1 s_{\text{fine}}^{[2]} \right) + H^1 \left( H^0 s_{\text{fine}}^{[1]} + H^1 s_{\text{fine}}^{[3]} \right) \tag{1a} \\
  d_H^{\text{coarse}} &= H^0 \left( G^0 s_{\text{fine}}^{[0]} + G^1 s_{\text{fine}}^{[2]} \right) + H^1 \left( G^0 s_{\text{fine}}^{[1]} + G^1 s_{\text{fine}}^{[3]} \right) \tag{1b} \\
  d_V^{\text{coarse}} &= G^0 \left( H^0 s_{\text{fine}}^{[0]} + H^1 s_{\text{fine}}^{[2]} \right) + G^1 \left( H^0 s_{\text{fine}}^{[1]} + H^1 s_{\text{fine}}^{[3]} \right) \tag{1c} \\
  d_D^{\text{coarse}} &= G^0 \left( G^0 s_{\text{fine}}^{[0]} + G^1 s_{\text{fine}}^{[2]} \right) + G^1 \left( G^0 s_{\text{fine}}^{[1]} + G^1 s_{\text{fine}}^{[3]} \right) \tag{1d}
\end{align*}
\]

These details become increasingly significant around the portions of the grids where the modelled data exhibit kinks or discontinuities. The details that have an overall magnitude bigger than a dimensionless error threshold \( 0 < \varepsilon < 1 \) are deemed significant. For hydrodynamic simulations, a value for \( \varepsilon \) around \( 10^{-3} \) is recommended [26,27]. The details are produced level-wise via applying Eq. 

\[
\begin{align*}
  s_{\text{coarse}} &= H^0 \left( H^0 s_{\text{fine}}^{[0]} + H^1 s_{\text{fine}}^{[2]} \right) + H^1 \left( H^0 s_{\text{fine}}^{[1]} + H^1 s_{\text{fine}}^{[3]} \right) \tag{1a} \\
  d_H^{\text{coarse}} &= H^0 \left( G^0 s_{\text{fine}}^{[0]} + G^1 s_{\text{fine}}^{[2]} \right) + H^1 \left( G^0 s_{\text{fine}}^{[1]} + G^1 s_{\text{fine}}^{[3]} \right) \tag{1b} \\
  d_V^{\text{coarse}} &= G^0 \left( H^0 s_{\text{fine}}^{[0]} + H^1 s_{\text{fine}}^{[2]} \right) + G^1 \left( H^0 s_{\text{fine}}^{[1]} + H^1 s_{\text{fine}}^{[3]} \right) \tag{1c} \\
  d_D^{\text{coarse}} &= G^0 \left( G^0 s_{\text{fine}}^{[0]} + G^1 s_{\text{fine}}^{[2]} \right) + G^1 \left( G^0 s_{\text{fine}}^{[1]} + G^1 s_{\text{fine}}^{[3]} \right) \tag{1d}
\end{align*}
\]
(1a-1d) and after encoding over the hierarchy, a tree-like structure referred to as the “tree of details” is obtained.

After getting the tree of details, leaf cells are those at which the details either stop being significant (e.g. blue coloured cells in Fig. 1c) or otherwise belong to the finest grid (e.g. green coloured cells in Fig. 1c). To produce leaf cells data, decoding is applied recursively across the cells of two consecutive grids by ascending the tree of details and adding them upon the coarsest resolution data. An operation of decoding is described in Eqs. (2a - 2d) whereby child cells’ data are produced from the parent cell data. After decoding, the leaf cells’ data are available on the hierarchy but their indices must be identified so that the data can be fetched to assemble them into the 2D adaptive grid (Fig. 1d).

\[
\begin{align*}
    s_{[0]}^{fine} &= H^0(H^0s^{coarse} + G^0d^{coarse}_H) + G^0(H^0d^{coarse}_V + G^0d^{coarse}_D) \\
    s_{[2]}^{fine} &= H^0(H^1s^{coarse} + G^1d^{coarse}_H) + G^0(H^1d^{coarse}_V + G^1d^{coarse}_D) \\
    s_{[1]}^{fine} &= H^1(H^0s^{coarse} + G^0d^{coarse}_H) + G^1(H^1d^{coarse}_V + G^0d^{coarse}_D) \\
    s_{[3]}^{fine} &= H^1(H^1s^{coarse} + G^1d^{coarse}_H) + G^1(H^1d^{coarse}_V + G^1d^{coarse}_D)
\end{align*}
\] (2a) (2b) (2c) (2d)

On the assembled 2D adaptive grid, the FV1 scheme update is applied to each leaf cell’s data using a forward Euler time integration in which the spatial operator requires access to the data of its four neighbouring cells [12]. Note that coping with the update at a coarse cell that is adjacent to finer cells is not problematic with HWFV1 [26] because a matching coarse resolution data was already produced during encoding. This would make the GPU parallelisation of the FV1 scheme within the HWFV1 algorithm similar to that reported for uniform grid FV1 schemes in alternative studies [12,45]. The timestep is selected based on a maximum Courant number of 0.5 and the friction source terms are integrated cell-wise at the start of each timestep using a split implicit friction scheme [12].

From the updated data on the 2D adaptive grid, encoding is reapplied with zero details on the tree of details for those cells on the hierarchy that were not assembled into the 2D adaptive grid. With
this, the HWFV1 algorithm can be repeated for each timestep until the end of a simulation, as shown in Fig. 2.

Fig. 2. Operations performed by the HWFV1 algorithm every timestep.

2.2. GPU parallelisation of the HWFV1 algorithm

To effectively parallelise the HWFV1 algorithm using the CUDA programming model, ensuring coalesced memory access and avoiding warp divergence are essential [46]. In CUDA programming, GPU data resides in memory that is accessed by workers, called threads. Coalescing occurs when adjacent threads access contiguous memory locations, and is necessary to efficiently update the adaptive grid. For the same reason, batches of 32 threads, or warps, must avoid the execution of differing instructions to avoid divergence.

Coalescing requirements are straightforward to meet when parallelising FV1 schemes on a uniform grid, including the FV1 component of the HWFV1 algorithm [12]. In contrast, the HWs component of HWFV1 is not easy to coalesce because the MRA process is recursive and involves a hierarchy of grids. The operations therein must be reworked to ensure coalesced memory access during encoding and decoding, and minimise warp divergence during the identification of leaf cell indices. The operations of the MRA process are reworked by applying the indexing of Z order curves (Section 2.2.1) and the adoption of a Parallel Tree Traversal approach (Section 2.2.2).

2.2.1. Indexing of Z order curves

A Z order curve is a way to index the cells in a $2^n \times 2^n$ grid to a 1D array such that the indices of the cells close together in space are also close together in the array. It is created by following the Morton
codes of each cell, and a Morton code is obtained by bit interleaving a cell’s X and Y positional indices [47]. The process of bit interleaving is indicated by the alternating red and black digits in Fig. 3a in a $2^2 \times 2^2$ grid as an example. The resulting Morton codes, in decimal form, are followed in ascending order to create the Z order curve for this grid, as shown in Fig. 3b.

![Fig. 3. Indexing the cells of a grid using a Z order curve: (a) bit interleaving of the X and Y indices in bit representation for each cell, leading to Morton codes in binary form; and, (b) following the decimal form of the Morton codes for each cell spans a Z order curve.](image)

In GPU-HWFV1, Z order curves are created for each grid in the hierarchy and continuity in the indexing of the Z order curves is enforced by starting the indexing from the coarsest grid and ending at the finest grid, as shown in the left panel of Fig. 4. With this unambiguous indexing of each cell in the hierarchy, the data and details needed for encoding and decoding are made to reside in adjacent memory locations, as shown in the zoomed-in portion in Fig. 4. Hence, when Eqs. 1a - 1d are applied for encoding (resp. Eqs. 2a - 2d for decoding), the memory access is coalesced.
**Fig. 4.** Enforced continuity in the indices of the cells in the hierarchy using Z order curves.

### 2.2.2. Parallel Tree Traversal (PTT)

The Z order curve allows the construction of the tree of details and the production of the leaf cells’ data in a fully coalesced manner. However, although the leaf cells’ data are available on the hierarchy, their indices must be identified in parallel to assemble the 2D adaptive grid on the GPU. This work adapts a PTT algorithm on the GPU developed by NVIDIA for computer graphics applications [48], and implements it within GPU-HWFV1. For simplicity, the proposed implementation of the PTT algorithm is explained for the simultaneous climbing of the tree of details shown before in Fig. 1c, but after the indexing of Z order curves is applied, as shown in Fig. 5a.

The starting point is to launch as many threads as the number of cells on the finest uniform grid, for instance $2^2 \times 2^2 = 16$ threads in Fig. 5a., denoted by $t_0, t_1, \ldots t_{15}$. The threads are tasked to start simultaneous traversal from the single cell on the coarsest grid. Each thread $t_m$ attempts to climb to the cell on the finest grid indexed with Morton code $m$ referring back to Fig. 3. During traversal, by checking against the tree of details, a thread stops as soon as it reaches a leaf cell and records its index in memory. An example can be seen in the hierarchy in Fig. 5a-5b: $t_4$ will attempt to reach the cell with index 9 (which has Morton code 4, see Fig 3), climbing along the cells with indices $\{0, 2, 9\}$ to stop at the leaf cell with index 9 and recording it in memory. In doing so, warp divergence is
avoided because adjacent threads perform similar traversals due to the continuous indexing of the Z curves spanning the hierarchy of 2D grids. Many of the threads climb to the same leaf cells without reaching a cell on the finest grid. For example, as indicated in Fig. 5a-5b, threads \{t_4, t_5, t_6, t_7\} climb up the cell with index 0 to then reach the cell with index 2 and finally settles for the cells with indices \{9, 10, 11, 12\} located on the the finest grid. In contrast, threads \{t_0, t_1, t_2, t_3\} settle for the leaf cell with index 1, which does not belong to the finest grid, each of them recording the same index 1 in memory and this results in duplicates.

**Fig. 5:** Description of the parallel tree traversal (PTT) algorithm. (a) Climbing the hierarchy after continuous indexing of Z order curves up to leaf cells (green and blue coloured); (b) Leaf cells’ indices recorded by the threads after climbing. There are duplicate indices for cells not belonging to the finest grid, which have to be removed after reusing them to detect the indices of the leaf cells’ four neighbours; and, (c) Assembly of the unique indices of the leaf cells making up the adaptive grid.

After PTT, an array of indices is recorded in memory that contains duplicate indices, shown in Fig. 5b. These duplicate indices are scrutinised by the threads to identify and record the indices of the four cells neighbouring the leaf cells from the east, west, north and south. For example, \(t_3\), which has already recorded the index of a leaf cell after PPT, will also record the indices of the leaf cell’s direct neighbours, namely the indices 1, 11, 1 and 14 shown in Fig. 5b (upper part). The indices of the four neighbour cells are also recorded in memory, resulting in more duplicates. Duplicate indices reside in adjacent locations in memory due to Z ordering, as seen in Fig. 5b (middle part), and can therefore be removed via stream compaction using the CUB library (NVIDIA, 2022). After
compaction, the remaining indices that are unique to the leaf cells making up the 2D adaptive grid, and their four neighbour cells, are used to fetch these cells’ data on the adaptive grid, as shown in Fig. 5c, and apply for the FV1 scheme update cell-wise.

3. Benchmarking the GPU-HWFV1 hydrodynamic model

The proposed GPU-HWFV1 model is benchmarked against two existing hydrodynamic models (Table 1). Benchmarking GPU-HWFV1 against its CPU predecessor CPU-HWFV1, allows to quantify the benefit of parallelising the adaptive gridding of HWFV1. Whereas, benchmarking it against the GPU-FV1 flow solver in LISFLOOD-FP 8.0 leads to identifying the simulation scenarios where adaptive gridding outperforms simulations on a uniform grid. The benchmarking is performed for five test cases (see Table 2). It entails checking the closeness of GPU-HWFV1’s simulation outputs to the predictions made by GPU-FV1 with reference to exact solutions or experimental data (to verify robustness) as well as also assessing its runtime performance (to quantify efficiency).

| Model name        | Developed in | Grid type                                      | GPU-parallelised |
|-------------------|--------------|-----------------------------------------------|------------------|
| CPU-HWFV1         | [26]         | Same adaptive grid as GPU-HWFV1              | No               |
| GPU-FV1           | [12]         | Uniform grid at the finest resolution          | Yes              |

| Test name                                                                 | Test type | Reason for use       | Previously used in |
|--------------------------------------------------------------------------|-----------|----------------------|--------------------|
| Quiescent flow over irregular topographies with different steepness. Dam-break flow over realistic terrain with friction effects (Section 3.1) | Synthetic | Verifying robustness | [26,49–52]          |
| Circular 2D dam-break flow (Section 3.2)                                  | Synthetic | Assessing runtime performance | [26,53]          |
| Pseudo-2D dam-break flow (Section 3.2)                                    | Synthetic | Assessing runtime performance | [26,27]          |
| Dam-break wave interaction with an                                        | Experimental | Assessing runtime   | [25,26,54]          |

The first test case (Section 3.1) will be used to verify robustness, by confirming the well-balanced property of GPU-HWFV1 (i.e. ability to preserve quiescent flow in the presence of steep terrain including wet/dry zones and fronts), and by checking its ability to replicate the accuracy of reference GPU-FV1 predictions when simulating dynamic flows over the same steep terrain.
After confirming the validity of using the proposed GPU-HWFV1 model, the second and third test cases (Section 3.2) will further focus on assessing the runtime performance of GPU-HWFV1 against CPU-HWFV1 and GPU-FV1 for synthetic dam break flows over flat terrain. As there is no terrain data to consider in these scenarios, they will be used to systematically analyse the runtime performance in relation to: the sensitivity to triggering grid refinement ($\varepsilon$), the depth in grid resolution ($L$), and the flow type (vigorous or smooth). Therefore, simulations are run by pairing different values for $\varepsilon$ and $L$, with $\varepsilon = \{10^{-4}, 10^{-3}, 10^{-2}\}$ to cover the recommended ranges for maintaining a fair balance between the predictive accuracy and runtime performance [26, 27] and, $L = \{8, 9, 10, 11\}$ as no gains in runtime performance was identified for $L \leq 7$ and using $L \geq 12$ was not affordable within the memory capacity of the GPU card used (RTX 2070). The effects of these parameters and the flow type on the adaptive gridding of the HWFV1 models is summarised in Table 3.

**Table 3.** Aspects against which the runtime performance of GPU-HWFV1 over CPU-HWFV1 and over GPU-FV1 are assessed.

| Aspects | Description | Finest grid resolution |
|---------|-------------|------------------------|
| $L$     | Controls the finest accessible grid resolution | Deeper with higher $L$ |
| $\varepsilon$ | Controls how far the finest grid resolution is accessed | More accessible with smaller $\varepsilon$ |
| Flow    | Vigorous (with discontinuities) to smooth (up to flat) | Triggered often for vigorous flows |

The expectations on runtime performance established from the synthetic test cases will be finally explored in the fourth and fifth test cases (Section 3.3), which involve realistic topographies represented by Digital Elevation Models (DEM) and using experimental data for model verification. Running the HWFV1 model with the presence of a DEM means that the maximum refinement $L$ must be set to accommodate the DEM resolution without allowing any coarsening in its grid beyond what the MRA of the DEM suggests.

**3.1. Verification of robustness**
The first synthetic test case in Table 2 is considered to verify the robustness of GPU-HWFV1, with a dual objective: (i) to verify its well-balanced property in the presence of wet-dry fronts with different levels of steepness in topography and different wetting conditions and, (ii) to reproduce a realistic dam-break flow with friction effects and moving wet-dry fronts. The domain area is 70 m × 30 m with closed wall boundaries and includes humps to represent an irregular topography profile. To verify the well-balanced property for realistic topographies, three hump shapes are considered with increasingly steeper bed slopes as shown in the top panels of Fig. 6 (smooth on the left, steeper in the middle and rectangular on the right). For each hump shape, appropriate initial conditions are applied (Table 4) with zero velocities to generate an unmoving free-surface elevation that leads to different wetting conditions around and/or at the humps.

![Fig. 6: Verification of robustness. Well-balanced property verification in the presence of wet-dry fronts with different levels of steepness in topography and different wetting conditions: (a) smooth humps, (b) steeper humps (c) rectangular humps. The top panel shows the geometrical profiles of the humps in the domain area, and the lower panel includes the time history of the maximum discharge errors where \( q_x = hu \) and \( q_y = hv \).](image)

| Hump profile | \( h + z \) (m) | Wetting conditions | Reference |
|--------------|-----------------|--------------------|-----------|
| Smooth       | 0.875           | Dry around the highest hump, critical \( (h = 0 \text{ m}) \) over the two small humps | \([49,51,52]\) |
| Steeper      | 1.78            | Dry around the highest hump, critical \( (h = 0 \text{ m}) \) at the peak of the medium hump, wet above the shortest hump \( (h > 0 \text{ m}) \) | \([50]\) |
| Rectangular  | 1.95            | Dry around the highest hump, critical \( (h = 0 \text{ m}) \) at the peak of the medium hump, wet above the shortest hump \( (h > 0 \text{ m}) \) |          |
GPU-HWFV1 simulations are run up to 100 s with a maximum refinement level $L = 8$ and an error threshold $\varepsilon = 10^{-3}$ (requiring around 3,000 timesteps to complete). The time histories of the maximum discharge errors are shown in the bottom panels of Fig. 6 for the three hump profiles. These errors are seen to become increasingly higher with increased irregularity in the hump profile but remain bounded to close to machine precision as also observed for the CPU model counterparts [26]. This demonstrates that GPU-HWFV1 is numerically well-balanced irrespective of the steepness of the bed slope and the presence of wet-dry zones and fronts in the domain area.

Next, GPU-HWFV1 is applied to reproduce a frictional dam-break flow ($n_M = 0.018 \text{ m}^{1/3}/\text{s}$) for the smooth hump profile (top left panel, Fig. 6). The initial dam-break flow conditions assume a water body of $h = 1.875 \text{ m}$ held by an imaginary dam located at $x = 16 \text{ m}$ with zero discharges. Using the same choice of $\varepsilon$ and $L$, an GPU-HWFV1 simulation is run up to 12 s. A GPU-FV1 simulation on the fine uniform grid is also performed to allow for like-for-like comparisons of flood depth profiles at outputs times reported in previous studies [51,52]. Fig. 7 includes the 2D contour maps of the flood depths predicted by GPU-HWFV1 (left panel) compared to those predicted by GPU-FV1 (right panel) at 0, 6 and 12 s. At 0 s (top panel), both models are seen to start from the same flood depth profile. At 6 s (middle panel), both models predict that the small humps are completely submerged and that the dam-break wave has reached the large hump, and the $L^1$ error difference between depths predicted by GPU-HWFV1 and GPU-FV1 is $4.6 \times 10^{-4}$. There are similar wave patterns surrounding the large hump by 12 s (bottom panel) and the $L^1$ error is $9.2 \times 10^{-4}$. In all the predictions, GPU-HWFV1 shows symmetrical flood extent profiles that are similar to those reproduced GPU-FV1 and other hydrodynamic model profiles reported in previous works (e.g. [51,52]). This indicates that the GPU-HWFV1 implementation is as robust as well-established models used for real-world applications.
Fig. 7: Verification of robustness. realistic dam-break flow with friction effects and moving wet-dry fronts. Flood depth profiles predicted by GPU-HWFV1 and GPU-FV1 on the left and right panels, respectively. At 0 s, the dam break wave emerges (top panels). At 6 s, the wave has submerged the small humps (middle panels). At 12 s, the wave starts to surround the large hump (bottom panels).

3.2 Assessing runtime performance for synthetic test cases

Circular 2D dam-break flow. This test case has often been used to verify new model implementations by capturing the symmetric propagation of shocks and rarefaction waves in the closed \([-20 \text{ m}, 20 \text{ m}]^2\) domain area [4]. Initially, the water depth inside the cylindrical dam is 2.5 m, separating it from a water depth of 0.5 m elsewhere. The dam-break flow occurs over a frictionless and flat terrain, resulting in a shock moving radially outwards and a rarefaction wave moving radially inwards, which eventually collapses to form a secondary shock. It is first used to further verify GPU-HWFV1 using the same choice of \(\epsilon\) and \(L\) as in Section 3.1 and by comparing its simulation outputs to those of CPU-HWFV1 and FV1-GPU. As in [26], simulations are run up to \(t = 3.5 \text{ s}\) for GPU-HWFV1, CPU-HWFV1 and FV1-GPU. Fig. 8 shows the water depth centrelines predicted by the three models. GPU-HWFV1 predicts water depths that are identical to those predicted by CPU-HWFV1, GPU-FV1 and the reference solution. The reference solution was produced using the FV1
numerical solution to 1D radial form of the 2D shallow water equations using $256 \times 256$ cells, following [4].

![Circular 2D dam-break flow. Verification of GPU-HWFV1 using the same choice of $\varepsilon$ and $L$ as in Section 3.1 ($L = 8$ and $\varepsilon = 10^{-3}$): water depth centrelines at $3.5$ s predicted by GPU-HWFV1, CPU-HWFV1, and GPU-FV1 compared to the reference solution.](image)

**Fig. 8**: Circular 2D dam-break flow. Verification of GPU-HWFV1 using the same choice of $\varepsilon$ and $L$ as in Section 3.1 ($L = 8$ and $\varepsilon = 10^{-3}$): water depth centrelines at $3.5$ s predicted by GPU-HWFV1, CPU-HWFV1, and GPU-FV1 compared to the reference solution.

To perform speed-up analysis, the models are rerun for the combinations of $\{\varepsilon, L\}$, and their runtimes were recorded for producing the speed-up ratios of GPU-HWFV1 relative to CPU-HWFV1 and GPU-FV1, respectively. Fig. 9 contains the plots of the speed-up ratios with increasing maximum refinement level $L$, relative to CPU-HWFV1 in the left panel and to GPU-FV1 in the right panel. The black lines indicate the average speed-up ratios obtained for the three error thresholds and the dash-dotted lines indicate the breakeven point above which GPU-HWFV1 demonstrates speed-up (used also in the subsequent figures).
**Fig. 9:** Circular 2D dam-break flow. Speed-up ratios to accomplish a 3.5 s simulation: GPU-HWFV1 over CPU-HWFV1 (left panel) and over GPU-FV1 (right panel).

GPU-HWFV1 is identified to be 5 to 46× faster than CPU-HWFV1. This speed-up is proportional to the increase in $L$ and decrease in $\varepsilon$. This suggests that adaptive gridding is much more efficient when made GPU resident, in particular as the maximum resolution refinement level is deepened and the sensitivity to refine resolution is increased. Compared to the runtime performance of FV1-GPU, GPU-HWFV1 is not faster in this test (right panel of Fig. 9) until $L \geq 9$ for all the $\varepsilon$ values, reaching a maximum of 3× for the largest $\varepsilon = 10^{-2}$, and around 2× for the smaller $\varepsilon = 10^{-3}$ and $10^{-4}$. This means that GPU-HWFV1, despite the overhead costs from the MRA process, can still compete with the speed of a fine uniform grid GPU-FV1 simulation even for a vigorous flow that would cause overrefinement on the adaptive grid. Namely, GPU-HWFV1 is likely to be faster than GPU-FV1 the deeper the grid resolution (which would lead to an excessively fine uniform grid for GPU-FV1) and the lower the sensitivity for triggering grid refinement. Next, a transient analysis of the speed-ups is performed in a longer simulation that sees a gradual change in the flow from vigorous to very smooth.

**Pseudo-2D dam-break flow.** This 1D dam-break flow test case has conventionally been used to verify hydrodynamic models for a short simulation run (2.5 s) involving transient shock and rarefaction waves propagation in two opposite directions. It was used recently for a much longer simulation time (40 s) to assess speed-up for CPU-based adaptive grid models to their uniform grid counterparts by considering a flow with gradual transition from vigorous to smooth [26,27].
Fig. 10: Pseudo-2D dam-break flow. Verification of GPU-HWFV1 using the same choice of $\varepsilon$ and $L$ as in Section 3.1 ($L = 8$ and $\varepsilon = 10^{-3}$): Water depths centerlines predicted by GPU-HWFV1, CPU-HWFV1 and GPU-FV1 at 2.5 s compared with the exact solution.

The domain area is $50 \text{ m} \times 25 \text{ m}$ and assumed to be flat and frictionless with open boundary conditions. The dam, located at $x = 10 \text{ m}$, initially separates an upstream water depth of 6 m from a downstream water depth of 2 m. After the dam removal, at $t = 0 \text{ s}$, the shock and rarefaction waves remain present in the domain area up to 2.5 s. After 3 s, the shock wave has left the domain area from the downstream and the flow dynamics are only driven by the presence of the rarefaction wave until 10 s, after which it exits from the upstream. Therefore, after 10 s, the flow dissipates gradually with increased smoothness until 40 s. The models are first verified by running simulations up to 2.5 s using the same choice of $\varepsilon$ and $L$ as in Section 3.1 ($L = 8$ and $\varepsilon = 10^{-3}$) for the HWFV1-based models and the finest uniform grid for the GPU-FV1 model. Fig. 10 shows the plots of the water depth centerlines predicted by the models all showing a good agreement with the exact solution [55].

To assess speed-up, GPU-HWFV1 and CPU-HWFV1 simulations are rerun for up to 40 s for the combinations of $\{\varepsilon, L\}$ alongside GPU-FV1 simulations on the finest uniform grid. Time histories of the runtimes are recorded throughout the 40 s simulations during which the flow transitions from vigorous to very smooth. Time series of the speed-up ratios of GPU-HWFV1 over CPU-HWFV1 and GPU-FV1 for the different values of $L$ and $\varepsilon$ are plotted in Fig. 11.

\[
\varepsilon = 10^{-2} \quad \varepsilon = 10^{-3} \quad \varepsilon = 10^{-4}
\]
Fig. 11: Pseudo-2D dam-break flow. Speed-up ratios of GPU-HWFV over CPU-HWFV1 (top panels), and over GPU-FV1 (bottom panels), for the three values of the threshold error $\varepsilon$ and considering different maximum refinement $L$.

Looking at the speed-up over CPU-HWFV1 (Fig. 11, top panels), GPU-HWFV1 exceeds the breakeven for all but the largest $\varepsilon = 10^{-2}$ and the lowest maximum refinement level $L = 8$ up to 2.5 s (Fig. 11, top left panel). This means that CPU-HWFV1 only remained as fast as GPU-HWFV1 when the flow included the shock and the rarefaction waves and for the setting with the least depth in resolution refinement and the least sensitivity to trigger grid refinement. However, even at $\varepsilon = 10^{-2}$ up to 8× speed-up is noted after 3 s when the shock wave is not present anymore. With any other combinations of $\{\varepsilon, L\}$, there is a significant demonstration of speed-up: with reduced $\varepsilon$ and increased $L$, GPU-HWFV1 becomes increasingly faster than CPU-HWFV1 up to reaching, for the highest $L$ and smallest $\varepsilon$, an average speed-up of 68× throughout the simulation and a maximum speed-up of 88× at 2.5 s when flow discontinuities were still present. This confirms the benefit of deploying GPU resident adaptive gridding as an alternative to the CPU version for general purpose modelling involving all types of flow.

In terms of speed-ups over GPU-FV1 (Fig. 11, bottom panels), at $\varepsilon = 10^{-2}$, GPU-HWFV1 demonstrates a maximum speed-up of 25× when $L = 11$, though it could only outrun GPU-FV1 for $L$
\[ L \geq 9 , \] beyond which GPU-HWFV1 increasingly shows speed-up within increased smoothening in the flow. At \( \varepsilon = 10^{-3} \), GPU-HWFV1’s maximum speed-up reduces to 12\( \times \) and outruns GPU-FV1 for \( L \geq 10 \), whereas for \( L \leq 9 \), it begins to demonstrate speed-up only after 10 s when the flow starts smoothening. This suggests to expect less speed-up over GPU-FV1 with increased sensitivity for triggering grid refinement with GPU-HWFV1 and reduced depth of the finest resolution. The same can be noted with \( \varepsilon = 10^{-4} \), but here GPU-HWFV1 starts to be faster than FV1-GPU for \( L \geq 9 \) and the overall maximum speed-up reduces to 8\( \times \) (reached again after 10 s when the flow is smoothening). These analyses indicate that an adaptive-grid GPU-HWFV1 simulation is likely to be more efficient than a uniform-grid GPU-FV1 simulation for very fine resolution modelling of gradual to smooth flows, with \( L \geq 9 \), and when the sensitivity to grid refinement is not maximal, with \( \varepsilon > 10^{-4} \).

3.3. Further investigations into runtime performance: realistic flow simulations

**Dam-break wave interaction with an urban district.** This test case has widely been used for model verification (e.g. [25]) as it has a set of spatial experimental data for the water depth and the velocities [56]. It involves a dam-break wave propagation in a 36 m × 3.6 m smooth channel \((n_M = 0.01)\) that includes a wall barrier with a gate initially separating an upstream water body of 0.4 m from a water depth of 0.011 m (Fig. 12). Downstream of the gate, there are twenty-five 0.3 m × 0.3 m square blocks, with 0.1 m gaps. The ground height for the wall barrier and the square blocks is 2 m. Based on this height and the dimension reported in [56], a DEM file was built at a resolution of 0.02 m × 0.02 m, made of 324,000 cells. The DEM includes the two rectangular blocks forming the wall barrier linked to the gate and the twenty-five square blocks. These discontinuous blocks are included in the grid and are accounted for as part of the well-balanced topography integration.

As the gate opens abruptly, a dam-break wave forms and flows swiftly to collide with the blocks. The blocks almost entirely impede the shock, creating a backwater zone upstream, while the unimpeded flow cascades through the gaps to form a hydraulic jump downstream as the simulation progresses (e.g. Fig. 24 i[56]).
A 10 s simulation is run using GPU-HWFV1 with $L = 11$ for two values of $\varepsilon = \{10^{-4}, 10^{-3}\}$, and using GPU-FV1 on a uniform grid using the finest resolution accessible to GPU-HWFV1. Fig. 13 shows the water depth (left panel) and velocity (right panel) profiles along $y = 0.2$ m at 6 s predicted by the GPU-HWFV1 and GPU-FV1 as well as the experimental profiles. All the models predicted profiles are within the expected range of agreement with the experimental profiles [25, 56]. Compared to the prediction made by GPU-FV1, those made by GPU-HWFV1 with $\varepsilon = 10^{-4}$ are closer than with $\varepsilon = 10^{-3}$ though the difference is not significant.

To analyse speed-ups, a CPU-HWFV1 simulation is also run. The recorded runtimes for the three models were used to calculate the time series of speed-up ratios of GPU-HWFV1 over CPU-HWFV1 and over GPU-FV1, which are plotted in the left and right panels of Fig. 14. On average, GPU-HWFV1 is found $19 \times$ and $25 \times$ faster to run than CPU-HWFV1, with $\varepsilon = 10^{-3}$ and $10^{-4}$ respectively, throughout the 10 s simulation. Higher levels of speed-up are demonstrated with larger $\varepsilon$, which is in line with the findings in Section 3.2. GPU-HWFV1 is also faster than GPU-FV1 in this
test, on average ~2.4× faster with both \( \varepsilon = 10^{-3} \) and \( 10^{-4} \). This can be expected for a run with \( L = 11 \) accommodating the very fine resolution of the DEM. Up to 2 s, the run with GPU-HWFV1 at \( \varepsilon = 10^{-3} \) demonstrates higher levels of speed-up than at \( \varepsilon = 10^{-4} \), which is in line with the observations made in Section 3.2. In contrast, after 2 s, GPU-HWFV1 at \( \varepsilon = 10^{-3} \) reduces the level of speed-up to become lower than with GPU-HWFV1 at \( \varepsilon = 10^{-4} \). This could be due to GPU-HWFV1’s higher sensitivity to grid refinement around the rectangular and square topographic blocks. Overall, GPU-HWFV1, besides being more performant than CPU-HWFV1, also remains faster than GPU-FV1 for this test. Supported by the analysis in Section 3.2, this can be expected given the maximised depth in the resolution level \( (L = 11) \) needed to accommodate the domain size to the fine resolution of the DEM.

---

**Fig. 14:** Dam-break wave interaction with an urban district. Speed-up ratios of GPU-HWFV1 over CPU-HWFV1 (left panel) and GPU-FV1 (right panel).

**Tsunami wave propagation over a complex beach.** The test case considers a 1:400 scaled replica of the 1993 Okushiri tsunami [57]. It has been used in other works for model verification and for runtime performance assessments of wavelet-based adaptive models versus their uniform counterparts for simulations on the CPU (i.e. [25,26]). It is here used to assess the runtime performance of the adaptive GPU-HWFV1 model versus CPU-HWFV1 and GPU-FV1 models.
Fig. 15: Tsunami wave propagation over a complex beach; (a) Topography contours over the domain area including the gauge point indicated in red. The tsunami-generated wave enters throughout the western boundary causing tsunami-generated flooding in the coastal area located in the eastern end (coloured in yellow); (b) Free-surface water elevation predicted by GPU-HWFV1 and GPU-FV1 compared to the experimental data.

The physical replica consists of a 5.488 m × 3.402 m smooth area ($n_M = 0.01 \text{ m}^{1/3}/\text{s}$) that has a uniform resolution of 0.014 m × 0.014 m on a DEM made of 163,840 cells (i.e. around twice fewer cells than the previous test case). The domain area has closed boundaries except for the western boundary through which a tsunami-generated inflow [26] enters and eventually reaches the coastal area to the east, before which there is a gauge point ($x = 4.521 \text{ m}, y = 1.696 \text{ m}$) hit by the tsunami-generated flood wave. Experimental time histories of the free-surface water elevation are available at this point and will be used to verify the GPU-HWFV1 and GPU-FV1 models’ ability to achieve a 22.5 s simulation. Fig. 15a displays a view of the domain area including the gauge point location, marked by a red dot, and the coastal area at the easten end (yellow colour). Given the smaller size of the domain area, the depth in the resolution level for the DEM to the domain size requires using $L = 9$ in this test to run the GPU-HWFV1 simulations with $\varepsilon = \{10^{-3}, 10^{-4}\}$. The GPU-FV1 simulation was run on a uniform grid at the DEM resolution. Fig. 15b contains time histories of the free-surface water elevations predicted by the models, which are in a good agreement with the experimental time histories. It can be seen that all the models predict the expected gradual retraction in the free-surface elevation between 12 s and 15 s, followed by a sharp increase that peaks at around 17 s. GPU-HWFV1 at $\varepsilon = 10^{-4}$ leads to predictions that are visually indistinguishable from those predicted by GPU-FV1.
With $\varepsilon = 10^{-3}$, the predictions remain comparable subject to small, localised discrepancies at times where there is a sharp flow transition such as at around 18 and 21 s.

Fig 16: Tsunami wave propagation over a complex beach. Speed-up ratios of GPU-HWFV1 over CPU-HWFV1 (left panel) and over GPU-FV1 (right panel).

Fig 16 contains the plots of the time histories of the speed-up ratios for GPU-HWFV1 over CPU-HWFV1 run with similar setting (left panel) and over GPU-FV1 (right panel) during the 22.5 s simulation. GPU-HWFV1 is seen to be significantly faster than CPU-HWFV1, leading to average speed-ups of 200× and 400× with $\varepsilon = 10^{-3}$ and $10^{-4}$, respectively. Compared to the previous test case, the terrain is complex all over the domain and the grid cannot be coarsened much, leading to an overrefined grid that is further refined by flow disturbances during HWFV1 simulations. In such a case, GPU-HWFV1 demonstrates remarkable speed-up over CPU-HWFV1, which increases as $\varepsilon$ is decreased from $10^{-3}$ to $\varepsilon = 10^{-4}$ in line with the observations in Section 3.2, but the speed-up doubles in this test.

 Compared to GPU-FV1, GPU-HWFV1 only demonstrates speed-up with $\varepsilon = 10^{-3}$ (around 1.25×) and is slightly slower to run with $\varepsilon = 10^{-4}$ where its speed-up falls below the breakeven line. This implies that it is worthwhile to perform adaptive gridding in this test for $\varepsilon = 10^{-3}$ but not for $\varepsilon = 10^{-4}$, at which the grid is overrefined (due to the terrain and flow disturbances). Overall, GPU-HWFV1 is shown to be generally faster than CPU-HWFV1 but could not outrun GPU-FV1 for $\varepsilon =$
10^{-4} (which leads to an overrefined grid) and for $L = 9$ (to accommodate a relatively small domain). Nonetheless, for $\varepsilon = 10^{-3}$, GPU-HWFV1 remains a viable choice over GPU-FV1.

4. Summary, conclusions and implications

This paper presented a computational innovation that was essential in developing an adaptive gridding method based on the multiresolution analysis (MRA) of the Haar wavelets (HWs) that runs entirely on the graphics processing unit (GPU). The MRA process features recursive operations that encode (coarsen) and decode (refine) data and details over a hierarchy of two-dimensional (2D) grids at various resolutions. The coarsest grid in the hierarchy is made up of a single cell while the finest grid is made up of $2^L \times 2^L$ cells, where $L$ is maximum refinement level. In the MRA process, the encoding operations need an error threshold $\varepsilon$ to identify significant details that have an overall magnitude beyond $\varepsilon$. After encoding, the decoding operations follow by adding up these significant details to produce the data of the cells that make up the 2D adaptive grid.

The presented innovation is the first to lead to an effective parallelisation of the MRA process. This innovation involves simultaneously applying the indexing of Z order curves and adopting a Parallel Tree Traversal (PTT) approach. The indexing of Z order curves ensures coalesced memory access in the encoding and decoding operations, while the PTT approach minimises warp divergence during the identification of the indices of the cells making up the adaptive grid.

Implementing the innovation allowed the development of a GPU resident, first-order finite volume (FV1) hydrodynamic model incorporating the adaptive gridding of HWs (GPU-HWFV1). GPU-HWFV1 was first verified and then its runtime performance assessed against a predecessor running on the central processing unit (CPU-HWFV1) as well as an operational GPU-FV1 uniform grid hydrodynamic model ran on the finest resolution accessible to the HWFV1 models. The verification was performed using $\varepsilon = 10^{-3}$ (recommended for hydrodynamic modelling) and $L = 8$ for four synthetic test cases involving motionless, vigorous, gradual and smooth flows. A systematic runtime performance assessment was performed for two synthetic test cases of a dam-break flows
over flatbed (without a DEM), where a lower and higher order-of-magnitude for $\epsilon = \{10^{-2}, 10^{-3}, 10^{-4}\}$ was also considered in combination with an increase in the maximum refinement level $L = \{8, 9, 10, 11\}$. Verification and runtime performance assessments were finally performed for realistic test cases with DEMs for which the value of $L$ was selected based on the resolution of the DEM to the overall extent of the domain area, and by running GPU-HWFV1 for $\epsilon = \{10^{-3}, 10^{-4}\}$.

The overall performance of GPU-HWFV1 for all the test cases provides strong evidence that it is as robust as GPU-FV1 in replicating the realistic flows including the presence of uneven topographies, wet-dry fronts and friction effects. In terms of runtime performance over CPU-HWFV1, GPU-HWFV1 yields significant speed-ups for all the test cases, ranging between $20\times$ to $400\times$. Hence, this work offers compelling evidence to apply the GPU resident adaptive gridding method to other fields in computational engineering. From the systematic runtime performance assessment for the synthetic test cases, GPU-HWFV1 tends to demonstrate speed-up of around $1.1\times$ to $30\times$ over GPU-FV1 for $L \geq 9$ and/or by avoiding the smallest $\epsilon = 10^{-4}$. For the test cases involving realistic flows over real DEMs, GPU-HWFV1 could comfortably show speed-up over GPU-FV1 for the test with $L = 11$ and for $\epsilon = 10^{-3}$ for the test with $L = 9$. Hence, GPU-HWFV1 can be favoured to gain runtime performance over GPU-FV1 for hydrodynamic modelling over real DEMs, namely with an increased fineness in the DEM resolution alongside an increased size of the domain area.
Declaration of interests

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.

Acknowledgements

We would like to thank Mohammad Kazem Sharifian for sharing the CPU code for HWFV1 and for helping out in the setting up of the input files for the GPU simulations. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

Funding

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R007349/1. It is part of the SEAMLESS-WAVE (SoftwarE infrAstructure for Multipurpose fLood modElling at variouS scaleS based on WAVElets) project, https://www.seamlesswave.com, where information on software availability can be found.

CRediT authorship contribution statement

Alovya Ahmed Chowdhury: Conceptualisation, Methodology, Software, Writing – original draft, Writing – review and editing. Georges Kesserwani: Writing – original draft, Writing – review and editing, Supervision, Funding acquisition. Charles Rougé: Writing – review and editing, Supervision. Paul Richmond: Writing – review and editing, Supervision.
[1] K. Guo, M. Guan, D. Yu, Urban surface water flood modelling – a comprehensive review of current models and future challenges, Hydrology and Earth System Sciences. 25 (2021) 2843–2860. https://doi.org/10.5194/hess-25-2843-2021.

[2] J. Macías, M.J. Castro, C. Escalante, Performance assessment of the Tsunami-HySEA model for NTHMP tsunami currents benchmarking. Laboratory data, Coastal Engineering. 158 (2020) 103667. https://doi.org/10.1016/J.COASTALEN.2020.103667.

[3] J. Teng, A.J. Jakeman, J. Vaze, B.F.W. Croke, D. Dutta, S. Kim, Flood inundation modelling: A review of methods, recent advances and uncertainty analysis, Environmental Modelling & Software. 90 (2017) 201–216. https://doi.org/10.1016/J.ENVSOFT.2017.01.006.

[4] E.F. Toro, Shock-capturing methods for free-surface shallow flows, Wiley-Blackwell, 2001.

[5] E.F. Toro, P. Garcia-Navarro, Godunov-type methods for free-surface shallow flows: A review, Journal of Hydraulic Research. 45 (2007) 736–751. https://doi.org/10.1080/00221686.2007.9521812.

[6] R. Akoh, T. Ishikawa, T. Kojima, M. Tomaru, S. Maeno, High-resolution modeling of tsunami run-up flooding: A case study of flooding in Kamaishi city, Japan, induced by the 2011 Tohoku tsunami, Natural Hazards and Earth System Sciences. 17 (2017) 1871–1883. https://doi.org/10.5194/nhess-17-1871-2017.

[7] G. de Almeida, P. Bates, H. Ozdemir, Modelling urban floods at sub-metre resolution: challenges or opportunities for flood risk management?, Journal of Flood Risk Management. 11 (2016). https://doi.org/10.1111/jfr3.12276.

[8] H. Gallegos, J. Schubert, B. Sanders, Two-dimensional, high-resolution modeling of urban dam-break flooding: A case study of Baldwin Hills, California, Advances in Water Resources. 32 (2009) 1323–1335. https://doi.org/10.1016/j.advwatres.2009.05.008.

[9] Y. Xing, Q. Liang, G. Wang, X. Ming, X. Xia, City-scale hydrodynamic modelling of urban flash floods: the issues of scale and resolution, Natural Hazards. 96 (2019). https://doi.org/10.1007/s11069-018-3553-z.

[10] I. Echeverribar, M. Morales-Hernández, P. Brufau, P. García-Navarro, 2D numerical simulation of unsteady flows for large scale floods prediction in real time, Advances in Water Resources. 134 (2019) 103444. https://doi.org/10.1016/J.ADVWATRES.2019.103444.

[11] X. Xia, Q. Liang, X. Ming, A full-scale fluvial flood modelling framework based on a high-performance integrated hydrodynamic modelling system (HiPIMS), Advances in Water Resources. 132 (2019) 103392. https://doi.org/10.1016/J.ADVWATRES.2019.103392.

[12] J. Shaw, G. Kesserwani, J. Neal, P. Bates, M.K. Sharifian, LISFLOOD-FP 8.0: the new discontinuous Galerkin shallow-water solver for multi-core CPUs and GPUs, Geosci. Model Dev. 14 (2021) 3577–3602. https://doi.org/10.5194/gmd-14-3577-2021.

[13] L. Arpaia, M. Ricchiuto, r–adaptation for Shallow Water flows: conservation, well balancedness, efficiency, Computers & Fluids. 160 (2018) 175–203. https://doi.org/10.1016/J.COMPFLUID.2017.10.026.

[14] D.A. Haleem, G. Kesserwani, D. Caviedes-Voullième, Haar wavelet-based adaptive finite volume shallow water solver, Journal of Hydroinformatics. 17 (2015) 857–873. https://doi.org/10.2166/hydro.2015.039.

[15] Y. Lakhliﬁ, S. Daoudi, F. Boushaba, Dam-Break Computations by a Dynamical Adaptive Finite Volume Method, Journal of Applied Fluid Mechanics. 11 (2018) 1543–1556. https://doi.org/10.29252/jafm.11.06.28564.
A.R. Brodtkorb, M.L. Sætra, M. Altinakar, Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation, Computers & Fluids. 55 (2012) 1–12. https://doi.org/10.1016/J.COMPFLUID.2011.10.012.

B.F. Sanders, J.E. Schubert, PRIMo: Parallel raster inundation model, Advances in Water Resources. 126 (2019) 79–95. https://doi.org/10.1016/J.ADVWATRES.2019.02.007.

J. Hou, R. Wang, Q. Liang, Z. Li, M.S. Huang, R. Hinkelmann, Efficient surface water flow simulation on static Cartesian grid with local refinement according to key topographic features, Computers & Fluids. 176 (2018) 117–134. https://doi.org/10.1016/J.COMPFLUID.2018.03.024.

R. Donat, M.C. Marti, A. Martinez-Gavara, P. Mulet, Well-Balanced Adaptive Mesh Refinement for shallow water flows, Journal of Computational Physics. 257 (2014) 937–953. https://doi.org/10.1016/J.JCP.2013.09.032.

G. Kesserwani, Q. Liang, Locally Limited and Fully Conserved RKDG2 Shallow Water Solutions with Wetting and Drying, J. Sci. Comput. 50 (2012) 120–144. https://doi.org/10.1007/s10915-011-9476-4.

Q. Liang, A.G.L. Borthwick, Adaptive quadtree simulation of shallow flows with wet–dry fronts over complex topography, Computers & Fluids. 38 (2009) 221–234. https://doi.org/10.1016/J.COMPFLUID.2008.02.008.

F. Zhou, G. Chen, Y. Huang, J.Z. Yang, H. Feng, An adaptive moving finite volume scheme for modeling flood inundation over dry and complex topography, Water Resources Research. 49 (2013) 1914–1928. https://doi.org/https://doi.org/10.1002/wrcr.20179.

D. Caviedes-Voullième, G. Kesserwani, Benchmarking a multiresolution discontinuous Galerkin shallow water model: Implications for computational hydraulics, Advances in Water Resources. 86 (2015) 14–31. https://doi.org/10.1016/J.ADVWATRES.2015.09.016.

N. Gerhard, D. Caviedes-Voullième, S. Müller, G. Kesserwani, Multiwavelet-based grid adaptation with discontinuous Galerkin schemes for shallow water equations, Journal of Computational Physics. 301 (2015) 265–288. https://doi.org/10.1016/J.JCP.2015.08.030.

D. Caviedes-Voullième, N. Gerhard, A. Sikstel, S. Müller, Multiwavelet-based mesh adaptivity with Discontinuous Galerkin schemes: Exploring 2D shallow water problems, Advances in Water Resources. 138 (2020) 103559. https://doi.org/10.1016/J.ADVWATRES.2020.103559.

G. Kesserwani, M.K. Sharifian, (Multi)wavelets increase both accuracy and efficiency of standard Godunov-type hydrodynamic models: Robust 2D approaches, Advances in Water Resources. 144 (2020) 103693. https://doi.org/10.1016/J.ADVWATRES.2020.103693.

G. Kesserwani, J. Shaw, M.K. Sharifian, D. Bau, C.J. Keylock, P.D. Bates, J.K. Ryan, (Multi)wavelets increase both accuracy and efficiency of standard Godunov-type hydrodynamic models, Advances in Water Resources. 129 (2019) 31–55. https://doi.org/10.1016/J.ADVWATRES.2019.04.019.

J. Pau, B. Sanders, Performance of Parallel Implementations of an Explicit Finite-Volume Shallow-Water Model, Journal of Computing in Civil Engineering - J COMPUT CIVIL ENG. 20 (2006). https://doi.org/10.1061/(ASCE)0887-3801(2006)20:2(99).

M.J. Castro, S. Ortega, M. de la Asunción, J.M. Mantas, J.M. Gallardo, GPU computing for shallow water flow simulation based on finite volume schemes, Comptes Rendus Mécanique. 339 (2011) 165–184. https://doi.org/10.1016/J.CRME.2010.12.004.
[30] A. Lacasta, M. Morales-Hernández, J. Murillo, P. García-Navarro, GPU implementation of the 2D shallow water equations for the simulation of rainfall/runoff events, Environ Earth Sci. (2015). https://doi.org/10.1007/s12665-015-4215-z.

[31] R. Vacondio, A. Dal Palù, A. Ferrari, P. Mignosa, F. Aureli, S. Dazzi, A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models, Environmental Modelling & Software. 88 (2017) 119–137. https://doi.org/10.1016/J.ENVSOFT.2016.11.012.

[32] M. Morales-Hernández, M.B. Sharif, A. Kalyanapu, S.K. Ghafoor, T.T. Dullo, S. Gangrade, S.C. Kao, M.R. Norman, K.J. Evans, TRITON: A Multi-GPU open source 2D hydrodynamic flood model, Environmental Modelling & Software. 141 (2021) 105034. https://doi.org/10.1016/J.ENVSOFT.2021.105034.

[33] M. Turchetto, R. Vacondio, A.D. Pal\'u, Multi-GPU Implementation of 2D Shallow Water Equation Code with Block Uniform Quad-Tree Grids, in: G. la Loggia, G. Freni, V. Puleo, M. de Marchis (Eds.), HIC 2018. 13th International Conference on Hydroinformatics, EasyChair, 2018: pp. 2105–2111. https://doi.org/10.29007/xwce.

[34] Flood Modeller 2D, Flood Modeller by Jacobs, (2022).

[35] MIKE 21 GPU, MIKE Powered by DHI: GPU - Guidelines, (2019).

[36] TUFLOW HPC, TUFLOW Classic/HPC User Manual, (2018).

[37] InfoWorks ICM, GPU Runtime Results for 2D InfoWorks ICM Models, (2018).

[38] HiPIMS-CUDA, HiPIMS-CUDA, (2021).

[39] A. Lacasta, M. Morales-Hernández, J. Murillo, P. Garcia-Navarro, An optimized GPU implementation of a 2D free surface simulation model on unstructured meshes, Advances in Engineering Software. 78 (2014) 1–15. https://doi.org/10.1016/j.advengsoft.2014.08.007.

[40] X. Qin, R.J. LeVeque, M.R. Motley, Accelerating an Adaptive Mesh Refinement Code for Depth-Averaged Flows Using GPUs, Journal of Advances in Modeling Earth Systems. 11 (2019) 2606–2628. https://doi.org/https://doi.org/10.1029/2019MS001635.

[41] M. de la Asunción, M.J. Castro, Simulation of tsunamis generated by landslides using adaptive mesh refinement on GPU, Journal of Computational Physics. 345 (2017) 91–110. https://doi.org/10.1016/j.jcp.2017.05.016.

[42] M. Sætra, A. Brodtkorb, K.-A. Lie, Efficient GPU-Implementation of Adaptive Mesh Refinement for the Shallow-Water Equations, Journal of Scientific Computing. 63 (2014). https://doi.org/10.1007/s10915-014-9883-4.

[43] H.-Y. Schive, J.A. ZuHone, N.J. Goldbaum, M.J. Turk, M. Gaspari, C.-Y. Cheng, gamer-2: a GPU-accelerated adaptive mesh refinement code – accuracy, performance, and scalability, Monthly Notices of the Royal Astronomical Society. 481 (2018) 4815–4840. https://doi.org/10.1093/mnras/sty2586.

[44] F. Keinert, Wavelets and Multiwavelets, Chapman & Hall/CRC Press, Boca Raton, FL, 2003.

[45] M. Morales-Hernández, M.B. Sharif, S. Gangrade, T.T. Dullo, S.-C. Kao, A. Kalyanapu, S.K. Ghafoor, K.J. Evans, E. Madadi-Kandjani, B.R. Hodges, High-performance computing in water resources hydrodynamics, Journal of Hydroinformatics. 22 (2020) 1217–1235. https://doi.org/10.2166/hydro.2020.163.

[46] NVIDIA, CUDA C++ Programming Guide, (2021).

[47] J. Baert, Morton encoding/decoding through bit interleaving: Implementations, (2013).
[48] T. Karras, Thinking Parallel, Part II: Tree Traversal on the GPU, (2012) undefined.

[49] Y. Huang, N. Zhang, Y. Pei, Well-Balanced Finite Volume Scheme for Shallow Water Flooding and Drying Over Arbitrary Topography, Engineering Applications of Computational Fluid Mechanics. 7 (2013) 40–54. https://doi.org/10.1080/19942060.2013.11015452.

[50] G. Kesserwani, J.L. Ayog, D. Bau, Discontinuous Galerkin formulation for 2D hydrodynamic modelling: Trade-offs between theoretical complexity and practical convenience, Computer Methods in Applied Mechanics and Engineering. 342 (2018) 710–741. https://doi.org/10.1016/J.CMA.2018.08.003.

[51] M. Shirvani, G. Kesserwani, P. Richmond, Agent-based simulator of dynamic flood-people interactions, Journal of Flood Risk Management. 14 (2021) e12695. https://doi.org/10.1111/jfr3.12695.

[52] L. Song, J. Zhou, Q. Li, X. Yang, Y. Zhang, An unstructured finite volume model for dam-break floods with wet/dry fronts over complex topography, International Journal for Numerical Methods in Fluids. 67 (2011) 960–980. https://doi.org/10.1002/fld.2397.

[53] Y. Wang, Q. Liang, G. Kesserwani, J.W. Hall, A 2D shallow flow model for practical dam-break simulations, Journal of Hydraulic Research. 49 (2011) 307–316. https://doi.org/10.1080/00221686.2011.566248.

[54] W. Jeong, J.S. Yoon, Y.S. Cho, Numerical study on effects of building groups on dam-break flow in urban areas, Journal of Hydro-Environment Research. 6 (2012) 91–99. https://doi.org/10.1016/J.JHER.2012.01.001.

[55] O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T. Vo, F. James, S. Cordier, SWASHES: a library of Shallow Water Analytic Solutions for Hydraulic and Environmental Studies, (2011).

[56] S. Soares-Frazão, Y. Zech, Dam-break flow through an idealised city, Journal of Hydraulic Research. 46 (2008) 648–658. https://doi.org/10.3826/jhr.2008.3164.

[57] M. Matsuyama, H. Tanaka, An experimental study of the highest run-up height in the 1993 Hokkaido Nansei-Oki earthquake tsunami, in: 2001.