Geosci. Model Dev., 19, 5623–5640, 2026
https://doi.org/10.5194/gmd-19-5623-2026

Geoscientific Model Development
Open Access
EGU

CaMa-Flood-GPU: a GPU-based hydrodynamic model implementation for scalable global simulations

Shengyu Kang[1], **Jiabo Yin**[1], **and Dai Yamazaki**[2]

[1]State Key Laboratory of Water Resources Engineering and Management, Wuhan University, Wuhan, P.R. China
[2]Institute of Industrial Science, The University of Tokyo, Tokyo, Japan

Correspondence: Jiabo Yin (jboyn@whu.edu.cn) and Dai Yamazaki (yamadai@iis.u-tokyo.ac.jp)

Received: 27 December 2025 – Discussion started: 5 February 2026
Revised: 10 June 2026 – Accepted: 21 June 2026 – Published: 29 June 2026

Abstract. Floods are among the costliest natural hazards, demanding scalable models to simulate river and floodplain dynamics at a global scale. The Catchment-based Macro-scale Floodplain (CaMa-Flood) model is a leading system for this purpose, but its CPU-based implementation is computationally demanding. This paper introduces CaMa-Flood-GPU, a fundamental refactoring of the model optimized for Graphics Processing Unit (GPU) architectures. We systematically reinterpreted its core algorithms – including river routing on irregular networks, runoff interpolation, and water depth diagnosis – into highly parallel, GPU-native operations. Key challenges were addressed by implementing scatter-add for flux updates, sparse matrix multiplication for runoff mapping, and branchless kernels for floodplain dynamics, all while preserving the original model's physical fidelity. Implemented in Python with Triton kernels and PyTorch, CaMa-Flood-GPU achieves multi-GPU scalability through optimized communication patterns that minimize synchronization overhead. The software adopts a modular structure with optional components (e.g., bifurcation routing, adaptive time stepping) and flexible data interfaces. Benchmarks demonstrate an order-of-magnitude speedup over a 192-core CPU baseline and near-linear scaling on multiple GPUs, with negligible numerical differences from the original model. This performance leap reduces simulation times for high-resolution global runs from days to hours, enabling larger ensembles and rapid scenario analysis. By providing a reproducible and efficient tool, CaMa-Flood-GPU lowers the barrier for adopting GPU acceleration in large-scale hydrology. The released implementation provides a reproducible reference for future method development.

1 Introduction

Floods are among the costliest and most widespread natural hazards, motivating the development of scalable hydrodynamic models for river routing and floodplain dynamics at continental to global scales (Kang et al., 2025; Collins et al., 2024; Emerton et al., 2016). While detailed channel processes can be simulated with one-dimensional models (e.g., HEC-RAS), capturing floodplain dynamics requires two-dimensional (2D) models that solve the shallow-water equations on high-resolution grids (Tayefi et al., 2007). However, applying such 2D models to large river basins is computationally prohibitive for many applications. To improve efficiency, simplified models such as LISFLOOD-FP adopt the local inertial formulation, which neglects the convective-acceleration term in the shallow-water momentum equation while retaining gravity and bed-friction terms; this preserves the propagation of flood waves at a fraction of the cost of the full Saint-Venant equations (Bates et al., 2010; De Almeida et al., 2012; Neal et al., 2021). A further-scalable alternative is the sub-grid inundation approach: instead of solving the 2D shallow-water equations explicitly, the inundated area and depth within each computational unit are diagnosed from a pre-computed sub-grid topographic profile, given the unit's total water storage (Yamazaki, 2025). The Catchment-based Macro-scale Floodplain model (CaMa-Flood) is a leading example of this approach, enabling efficient yet physically-based global flood simulations (Yamazaki et al., 2011). CaMa-Flood introduced a novel vectorized unit-catchment discretization of the river network, a departure from traditional uniform grids (Yamazaki et al., 2011). Among available global routing models, CaMa-Flood provides an estab-

lished catchment-based representation of channel storage, floodplain storage and river-network routing. We selected CaMa-Flood as the baseline model for our GPU implementation; it has been adopted as the offline river-routing layer in land-surface and global hydrological models, with benchmark comparisons reporting measurable gains in discharge reproducibility relative to the native routing schemes (Zhao et al., 2017; Heinicke et al., 2024).

In the CaMa-Flood scheme, the world's river basins are divided into numerous irregularly shaped sub-basins (catchments), each treated as a computational unit. The water balance for each catchment c is the core of the model:

$$S_c^{t+\Delta t} = S_c^t + \boldsymbol{R}_c^t\,\Delta t + \sum_{u\in\mathrm{upstream}(c)} Q_{u\to c}^t\,\Delta t - Q_{c\to d}^t\,\Delta t\,, \quad (1)$$

where S_c^t is the water storage, $\boldsymbol{R}_c^t$ is the runoff input, $Q_{u\to c}^t$ is the inflow from upstream neighbor u, and $Q_{c\to d}^t$ is the outflow to the downstream catchment d. The outflow $Q_{c\to d}$ is computed using a local inertial approximation of the shallow-water momentum equations. This formulation neglects convective acceleration but preserves gravity and friction effects, leading to a one-dimensional momentum equation for the flow in each river link:

$$\frac{\partial Q}{\partial t} + gA\frac{\partial h}{\partial x} + g\frac{n^2 Q|Q|}{A^2 R_\mathrm{h}^{4/3}} = 0\,, \quad (2)$$

where Q is the discharge, A is the cross-sectional flow area, h is the water surface elevation, R_h is the hydraulic radius, and n is the Manning roughness coefficient. CaMa-Flood employs an explicit time integration of this equation, where the new outflow is computed based on the water level difference $(h_c - h_d)$ between adjacent catchments. CaMa-Flood is an instance of the catchment-based macro-scale floodplain (CMF) approach (Yamazaki et al., 2011; Yamazaki, 2025), which discretises every river basin into irregular unit-catchments of roughly 5–50 km extracted from MERIT Hydro (Yamazaki et al., 2019) rather than from a regular grid. Within each unit-catchment, water mass is partitioned into a channel storage and a floodplain storage, exchanged according to the channel bank-full geometry. Two simplifying assumptions are imposed: no significant topographic depressions inside a unit-catchment, and a spatially uniform water surface across its floodplain. Under these assumptions, sub-grid topographic profiles, precomputed as the cumulative distribution of high-resolution DEM elevations, diagnose water depth and inundation extent directly from total storage, so no 2D shallow-water solve is needed between neighbouring unit-catchments. Channel routing along the river network follows the local inertial momentum equation (Bates et al., 2010; De Almeida et al., 2012), with the outlet pixel of each unit-catchment supplying an absolute reference elevation so that water-surface gradients, and therefore backwater and flow reversal, can be represented even between coarse-resolution units. Recent extensions remove the single downstream flow path constraint and represent channel bifurcations as additional divergent flows driven by the same local inertial equation (Yamazaki et al., 2014; Mateo et al., 2017).

Over the years, CaMa-Flood has been continually enhanced to improve its realism and applicability. For instance, Yamazaki et al. (2014) extended it to represent river bifurcations and delta channel networks, and high-resolution datasets like MERIT Hydro (Yamazaki et al., 2019) have been incorporated to define channel geometry and catchment parameters globally. Some implementations have introduced adaptive time stepping to ensure numerical stability without significantly compromising efficiency. In an adaptive scheme, the model adjusts the time step Δt based on a Courant-Friedrichs-Lewy condition related to the fastest wave speed in the domain (Hunter et al., 2005). For example, one can require

$$\Delta t < f\,\frac{L_{ij}}{\sqrt{g h_i}} \quad \text{for all river reaches } i \to j\,, \quad (3)$$

where h_i is the local water depth, L_{ij} is the length of the river reach from catchment i to j, and $f < 1$ is a safety factor (e.g. 0.7) ensuring stability. In practice, the smallest allowable Δt from this criterion (across all reaches) is used as the adaptive time step for that update interval. By dynamically limiting the time step based on wave celerity, the model avoids numerical instability even in steep or high-flow segments while maximizing efficiency in calmer regions. Due to its balanced fidelity and efficiency, CaMa-Flood has been widely used in global hydrological studies – from estimating present-day flood hazards to projecting future flood risks under climate change (Hirabayashi et al., 2013; Kimura et al., 2023), and it has been coupled with climate and land-surface models to route runoff in Earth system simulations (Huang and Hattermann, 2018; Marthews et al., 2022; Hamitouche et al., 2025).

Despite these advances, running CaMa-Flood at very high spatial resolutions or with large ensembles remains computationally challenging on conventional CPUs. The model's irregular domain (hundreds of thousands of catchments with diverse sizes) and the need for small time steps in large rivers mean that global simulations at fine scales (on the order of 2–5 km catchments) can require many hours to days of runtime on a multi-core CPU. This limitation motivates the use of modern high-performance computation architectures, in particular Graphics Processing Units (GPUs), to accelerate global flood modeling. GPUs have become a cornerstone of scientific computing because of their massive parallelism, and they have been successfully applied to hydrodynamic models in recent years. GPU acceleration is increasingly being adopted across Earth-system model components, from atmospheric chemistry kernels in coupled climate-chemistry models (Alvanos and Christoudias, 2017) to flood hydrodynamics. In the latter, GPU-based modelling has matured along three complementary directions. First, for the full 2D shallow-water equations, multi-GPU and

single-GPU codes deliver order-of-magnitude speed-ups for high-resolution flood-inundation studies on regular meshes (Morales-Hernández et al., 2021). Second, for simplified local inertial or kinematic formulations, GPU ports of established CPU floodplain models extend GPU acceleration from urban catchments to continental flood mapping while retaining sub-grid topographic detail (Sharifian et al., 2023; Rong et al., 2024; Caviedes-Voullième et al., 2023). Third, for integrated land-surface and routing, GPU acceleration of hydrology models (Hokkanen et al., 2021) and GPU-resident machine-learning surrogates of routing (Zahura et al., 2020) demonstrate that the throughput unlocked by GPUs makes ensemble and global-domain experiments tractable. Most of the above target single-GPU execution on regular or quasi-regular meshes at sub-continental scale. Global river-routing on irregular unit-catchment networks, the regime in which CaMa-Flood operates and which is mandatory for properly representing bifurcations, deltas and floodplain storage on a global mesh, has so far not been ported to GPU. CaMa-Flood-GPU is, to our knowledge, the first multi-GPU implementation of a global, irregular, bifurcation-aware routing model.

The remaining gap therefore lies in global-scale river routing models such as CaMa-Flood, whose irregular unit-catchment networks pose challenges that differ fundamentally from those encountered in regional 2D GPU flood models. This likely stems from several intrinsic characteristics of large-scale hydrodynamic models that make GPU computation more challenging: (1) In 2D models, the connectivity between computational cells is uniform and limited to adjacent neighbors on a regular grid, whereas in global river models, the river network topology is defined by irregular upstream–downstream relationships that must be handled explicitly (Mizukami et al., 2016; Yamazaki et al., 2011). (2) In 2D models, the relationship between water storage and water level within each grid cell is generally linear or prescribed by a simple function, but global river models employ sub-grid floodplain topography, resulting in a nonlinear and spatially variable relationship (Yamazaki et al., 2011). (3) While 2D models use uniformly shaped grid cells as computational units, global river models discretize the land surface into irregular catchment-based units. This introduces interpolation across variable areas and greatly increases the total number of computational elements when performing high-resolution global simulations (Yamazaki et al., 2019), making efficient parallelization far more demanding.

In this study, we aim to clarify and overcome the fundamental challenges that have hindered the application of GPU acceleration to global-scale river models. Using CaMa-Flood as a representative example, our objectives are threefold: (1) to identify which aspects of its model structure and algorithms limit efficient GPU computation, (2) to explore and select appropriate GPU libraries and kernel implementations capable of addressing these limitations, and (3) to achieve global river simulations at kilometer-scale ($\sim$ 1 arcmin) resolution within a practical computational cost. To this end, we developed CaMa-Flood-GPU, a GPU-based reimplementation of the CaMa-Flood hydrodynamic core that reformulates the original CPU algorithms through computational science techniques optimized for modern GPU architectures. This work aims to bridge the gap between hydrological modeling and high-performance computing, providing a foundation for scalable, physically consistent global flood simulations at resolutions and runtimes that were previously impractical.

In the following sections, we detail the design and evaluation of CaMa-Flood-GPU. Section 2 examines the original CaMa-Flood algorithms to identify which aspects of their structure and computation limit efficient GPU acceleration, and then proposes schemes to resolve these bottlenecks through a redesigned hydrodynamic core. Section 3 presents results from a series of tests, including performance benchmarks to evaluate the speed and scalability of the GPU model across different hardware configurations, as well as validation of its numerical correctness against the CPU version. We discuss the speedups observed and analyze the remaining bottlenecks. Section 4 offers conclusions, highlighting the implications of this work and potential future developments. Through this paper, we aim to demonstrate that GPU acceleration can substantially empower global flood modeling, and we provide the tools and references to facilitate broader adoption and further improvements of such approaches.

2 Implementation

Implementing an efficient GPU version of a large-scale hydrodynamic model requires more than simply rewriting the original Fortran code in Python or replacing explicit loops with vectorized array operations. GPUs operate under a fundamentally different execution model from CPUs – one that favors massive, uniform parallelism and coalesced memory access. Consequently, a direct code translation of CaMa-Flood's CPU algorithms would yield suboptimal performance and potentially lose the model's numerical stability. Instead, we reinterpreted the core computational patterns of CaMa-Flood in terms of GPU-native primitives, systematically identifying and employing appropriate libraries and kernels that align with the model's hydrodynamic equations and data structures. This design philosophy enables us to integrate computational science insights – particularly those related to memory hierarchy and parallel reduction – into a physically based global river model, allowing high-performance GPU solvers to be applied to global river routing for the first time.

The implementation of CaMa-Flood-GPU follows a layered, modular architecture to separate concerns and maximize flexibility (Fig. 1). At the highest level, a `CaMaFloodModel` class controls the simulation, coordinating data flow and computation across modules and GPUs.

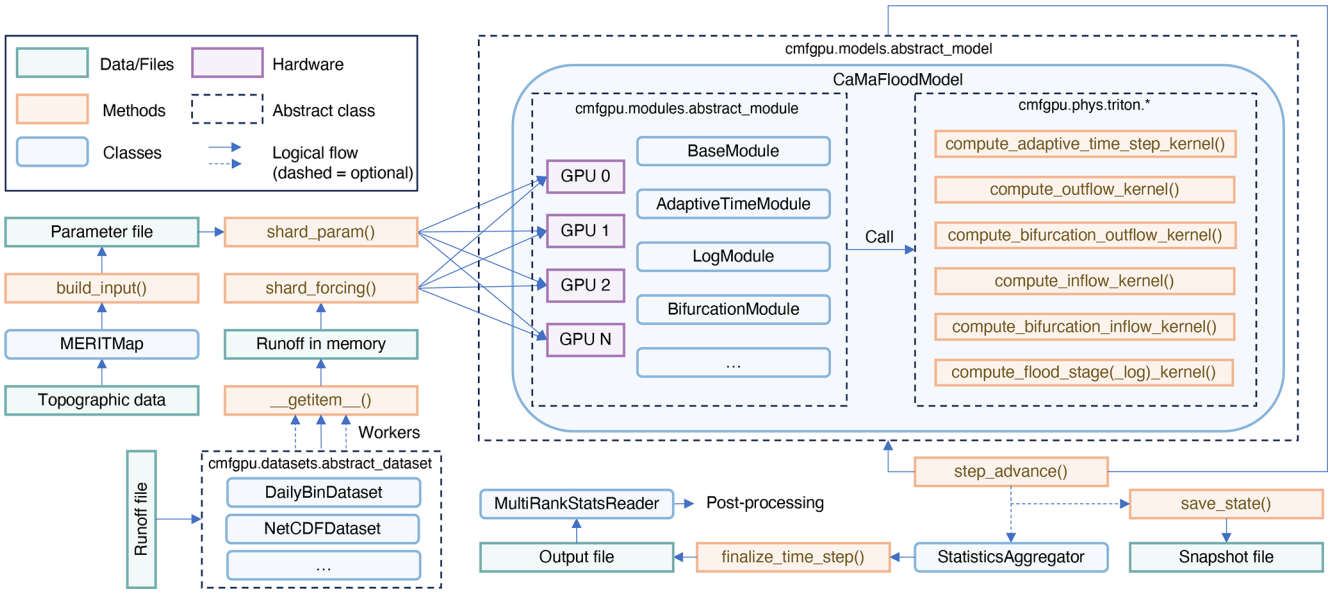

Figure 1. Schematic architecture of the CaMa-Flood-GPU system. Input data (terrain, river network, initial conditions) are partitioned across GPUs via domain sharding. The modular `CaMaFloodModel` coordinates GPU kernels at each time step, with optional sub-modules for adaptive time stepping, bifurcation flows, and logging. Outputs are collected by a `StatisticsAggregator`, combined by a multi-rank reader, and written for final results and post-processing.

The model is constructed with a registry of sub-modules, which implement optional features such as bifurcation flows, adaptive time-stepping, and logging/diagnostics. Underneath the model, we define abstract base classes for data handling and computational modules, allowing multiple implementations or extensions. For example, the input data interface is abstracted so that different forcing datasets (binary files, NetCDF files) can be used without changing the core model code. The model and modules together manage the GPU memory for all relevant state variables (such as water storage, discharge, etc.) and ensure that each GPU holds only the portion of data needed for its assigned catchments. By organizing the code in this modular way, we facilitate customization (users can enable/disable components via configuration) and make the system easier to maintain or extend (each module focuses on one aspect, e.g., bifurcation handling or time-step adaptation).

The implementation strategy is twofold. First, we focus on overcoming the key performance challenges inherent in porting a global, irregular-network model to a massively parallel GPU architecture. This involves addressing issues of data representation and communication overhead. Second, we aim to build a system that offers flexible customization, allowing users to easily adapt the model for different scientific applications, data sources, and regional focuses. The following sections detail our approach to these two goals.

2.1 Challenges

2.1.1 Irregular network topology

The core difficulty in adapting CaMa-Flood for GPUs is representing its irregular, directed catchment network (Fig. 2a) in a way that maps efficiently to parallel hardware. In the CaMa-Flood river network map, each grid cell, or unit-catchment, has exactly one downstream connection, forming a topology where water flows from many upstream sources toward a single outlet. From a computer science viewpoint, this structure can be encoded as a directed graph where each node (catchment) has an out-degree of one, except for terminal nodes (outlets). However, the introduction of bifurcation flows in later versions of CaMa-Flood (Yamazaki et al., 2014b) complicates this topology, allowing some nodes to have an out-degree greater than one. This forms a collection of directed trees rooted at the river mouths. A key computational challenge in this graph is the need to sum fluxes from a variable number of neighboring catchments. The local inertial model used in CaMa-Flood allows for backwater effects, where flow is not always unidirectional; reverse flow can occur if the water surface gradient changes. This means that at any given time, any neighbor can become an upstream contributor, requiring a flexible summation of fluxes. This *gather* or *reduction* operation over an irregular graph is a classic problem in parallel computing (Fig. 3a), as it leads to irregular memory access patterns and potential load imbalance, which are inefficient on GPUs that thrive on structured, uniform workloads.

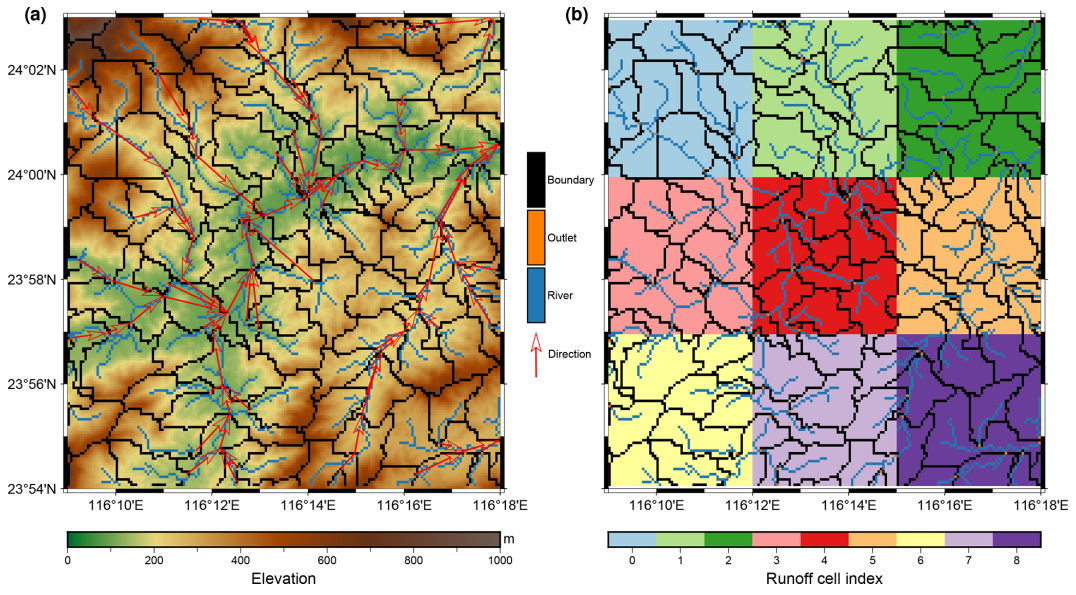

Figure 2. Illustration of catchment–grid misalignment. **(a)** Irregular catchment network with river, outlet, basin boundary, and flow direction. **(b)** Regular input grid of runoff cells, where grid cells do not coincide with catchment boundaries.

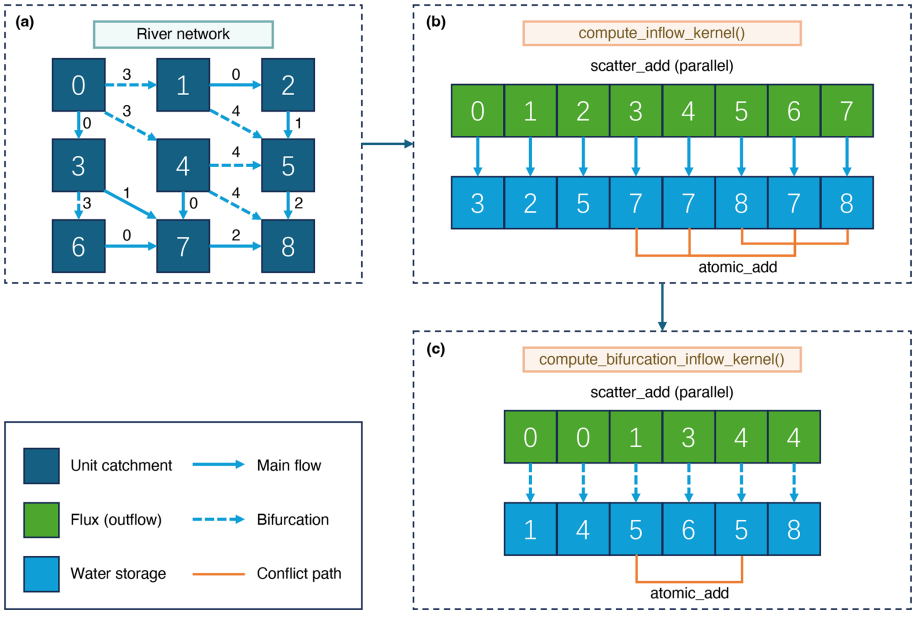

Figure 3. Illustration of parallel flux summation. **(a)** A schematic river network where numbered squares are unit catchments. The numbers on them represent the flux index, which in traditional methods dictates a sequential processing order (e.g., main flows, then bifurcations). **(b)** For main flows, outflow from each source catchment (top green array) is added in parallel to its downstream target (bottom blue array). **(c)** A similar parallel summation is used for bifurcation flows. Atomic operations are required to prevent race conditions when multiple source catchments flow into the same target, ensuring mass conservation.

Fortunately, this challenge is a well-studied problem in computational science, allowing us to reframe the algorithm by adapting established parallel computing solutions. To solve this, we leverage the highly-parallelized and extensively-optimized `scatter_add` operation available in the Triton language. In brief, a `scatter_add` distributes outflows from many source catchments into their downstream destinations in a single parallel pass; whenever several sources write to the same destination, as at confluences and bifurcation receivers, `atomic_add` serializes those individual increments at the hardware level, so the accumulated inflow is order-independent and mass-conserving without ex-

plicit locks. Instead of a *gather* operation where each catchment would need to read from multiple upstream locations, we use a *scatter* approach. As illustrated in Fig. 3b–c, the flux calculated for each river reach is atomically subtracted from the upstream catchment's storage and added to the downstream catchment's storage. This operation enables thousands of threads to safely and simultaneously update shared variables without conflicts, automatically managing the order of memory access. In essence, it lets every catchment update its inflow and outflow at once, while the GPU hardware ensures that no data are lost or overwritten – greatly improving both speed and efficiency compared to sequential processing. A potential race condition occurs when multiple upstream catchments attempt to write to the same downstream catchment's buffer simultaneously. Triton's `scatter_add` implementation automatically handles this by using *atomic add operations*, which guarantee that concurrent updates to the same memory location are correctly serialized, thus ensuring mass conservation without manual synchronization. We encode the river topology using index-based data structures. Catchments are renumbered in a topological order (upstream to downstream), and their state variables (e.g., water storage) are stored in structure-of-arrays format. This layout ensures that memory access is mostly contiguous and coalesced. Upstream connectivity is captured by compact adjacency lists, which allows for efficient, parallel summation using segmented reduction algorithms. This approach regularizes the execution over the irregular graph: GPU kernels iterate over contiguous index ranges, while indirect addressing handles the underlying graph connectivity.

2.1.2 Runoff interpolation

Another challenge arising from the irregular network topology is the mismatch between gridded external forcing and our catchment units (Fig. 2b). Most runoff products are delivered on latitude-longitude grids that do not align with catchment boundaries. At each time step, the forcing reader supplies runoff from an upstream land-surface model (LSM) to the dataset object. If using a global grid, N_r is the number of runoff-generating land grid cells, i.e., the output cells of an upstream LSM. The resulting runoff vector is then passed to `shard_forcing`, which maps the runoff-generating cells to the local GPU catchments. The original CaMa-Flood model addresses this by pre-computing an input matrix that maps runoff grid cells to catchment units. This is accomplished using a dedicated program, which calculates the overlapping area between the input data grid and the river network grid, generating a dense matrix that specifies the contribution of each runoff grid cell to each catchment. While effective, this approach produces a data structure that is not optimized for modern parallel hardware.

For CaMa-Flood-GPU, we reinterpret this mapping as a sparse matrix operation, which is significantly more efficient on GPUs. We define a sparse runoff aggregation matrix that

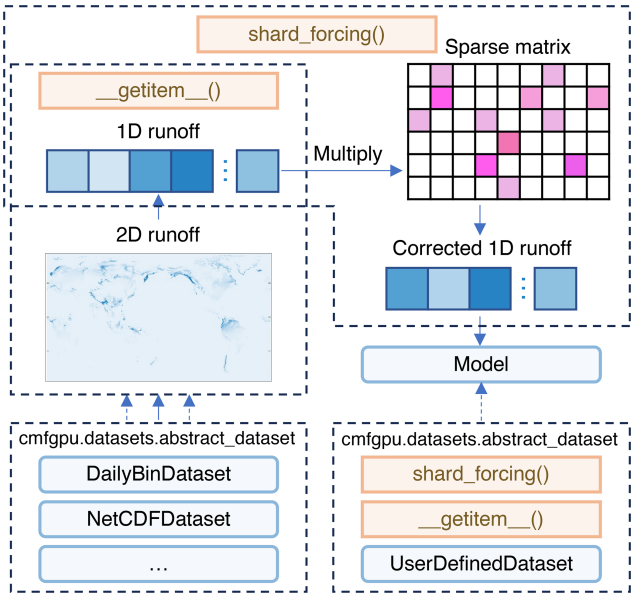

Figure 4. Schematic of the data input and runoff aggregation process. Multiple dataset types (daily binary files, annual NetCDF files, and user-defined datasets) are supported through a common abstract data layer. Runoff inputs are mapped to catchments via a sparse weight matrix, with each GPU holding the portion relevant to its assigned catchments.

maps gridded runoff data to irregular catchments (Fig. 4). Let $\boldsymbol{r} \in \mathbb{R}^{N_r}$ be the vector of runoff values for all grid cells at a given time (for instance, if using a global grid, N_r would be the number of runoff-generating land grid cells), and let $\boldsymbol{R}_c \in \mathbb{R}^{N_c}$ be the vector of total runoff inputs for all N_c catchments in the model. We define a sparse matrix $\mathbf{M}$ of size $N_c \times N_r$ such that:

$$\boldsymbol{R}_c = \mathbf{M}\boldsymbol{r} \tag{4}$$

where each element M_{ij} represents the fraction of runoff in grid cell j that contributes to catchment i. We compute these weights based on area overlap: if A_{ij} is the area of catchment i that lies within grid cell j, and $A_i = \sum_j A_{ij}$ is the total area of catchment i, then we set $M_{ij} = A_{ij}/A_i$ (with $M_{ij} = 0$ if catchment i has no land area in cell j). In other words, M_{ij} is the proportion of catchment i's area that falls in the cell j, so that the catchment's runoff $\boldsymbol{R}_c(i)$ is essentially the area-weighted average runoff over that catchment (assuming uniform runoff within each grid cell). The matrix $\mathbf{M}$ is extremely sparse – each catchment overlaps only a few grid cells, and each grid cell contributes to only a few catchments. This sparsity is ideal for GPU acceleration, as sparse matrix-vector multiplication (SpMV) is also a highly-parallelized and extensively-optimized operation in parallel computing libraries.

In practice, before a simulation, we either precompute or read in this matrix $\mathbf{M}$. For multi-GPU runs, $\mathbf{M}$ is partitioned per GPU. Each GPU stores only the rows of $\mathbf{M}$ correspond-

ing to its assigned catchments and the columns corresponding to the grid cells influencing those catchments. During the simulation, the rank 0 process reads the full runoff vector $\boldsymbol{r}$ and broadcasts it to all other ranks. Each GPU then performs the SpMV operation $\boldsymbol{R}_c = \mathbf{M}\boldsymbol{r}$ using its local sparse matrix to compute its catchment runoff inputs.

Beyond this optimized implementation, we provide a flexible interface for runoff inputs to handle various data sources. By abstracting the data loading and mapping procedure in a general `shard_forcing` interface, we allow for considerable customization. A user can create a new dataset subclass with a custom `shard_forcing` method that defines how to broadcast and map their particular data source onto the catchments. In particular, `shard_forcing` is intentionally designed as the coupling hand-off point for online or offline coupling with an LSM: an LSM can pass its per-time-step runoff tensor, in PyTorch or any compatible array, directly to `shard_forcing`, which performs the grid-to-catchment aggregation on-device without going through the disk. For instance, if a runoff input is provided already as catchment-specific values, `shard_forcing` could simply distribute those values directly to the appropriate GPUs without any grid-to-catchment aggregation. The core model remains unaware of these preprocessing details – as long as the dataset object supplies catchment runoff values $\boldsymbol{R}_c$ for each time step, the model will route them. We have implemented default dataset classes for common formats and provided a template for a `UserDefinedDataset` to guide custom implementations, making the system highly extensible.

2.1.3 Diagnosing water depth from storage

A core challenge in hydrodynamic modeling is to efficiently diagnose water depth and inundation extent from storage volume without resorting to computationally expensive 2D simulations. CaMa-Flood addresses this by discretizing the river network into sub-catchments and using pre-computed topographic profiles. This method relies on two key assumptions: (1) inundation progresses from the lowest elevations upward without being trapped in local depressions, and (2) the water surface elevation is uniform across the floodplain within a single sub-catchment. Under these assumptions, the relationship between water storage, water level, and inundation extent (including the flooded fraction) is represented by a monotonic, piecewise function derived from high-resolution elevation data, establishing a one-to-one correspondence for each catchment. This preserves CaMa-Flood's sub-grid floodplain treatment while avoiding any explicit two-dimensional floodplain solve inside the GPU routing loop. Channel bifurcations and overbank-routing pathways are instead represented as additional network links whose fluxes are accumulated through the same routing reductions as the main channel. This function, which acts as a topographic profile for each unit-catchment, is pre-computed and stored. In the original CPU implementation, this continuous function is discretized into a lookup table for computational efficiency. The detailed topographic profile, derived from the cumulative distribution of relative elevations within the catchment, is simplified into a series of points. These points define a piecewise function that approximates the original complex profile, enabling fast queries during simulation by interpolating between the stored values.

Porting this concept to a GPU requires an implementation that preserves massive parallelism. A naive, direct translation of the CPU logic might involve conditional branching (e.g., "if-else" statements) to find the correct interval in the lookup table for each catchment. However, such branching can severely degrade GPU performance by causing threads within the same warp to diverge and execute different code paths. To avoid this, we adopted a branchless approach. The topographic profiles for all catchments are organized as 2D lookup tables (catchment ID × profile level), which are then flattened into one-dimensional arrays in GPU memory. We then implement a dedicated GPU kernel where each thread, assigned to a single catchment, iterates through all predefined profile levels in a fixed loop. A `where` clause (a conditional assignment) is used to update the water level only when the total storage exceeds the storage value at that level. This ensures every thread performs the exact same sequence of operations, eliminating thread divergence. Once the correct level is identified, the kernel computes the memory address offset for that catchment's profile data and performs an interpolation to find the precise flood depth. The process, detailed in Algorithm 1, is executed in parallel for all catchments. This diagnostic step is an embarrassingly parallel problem, because the depth calculation for each catchment is entirely self-contained and does not depend on any other. The inundation diagnosis is based on the uniform-water-surface assumption within each unit-catchment, so it adds no inter-GPU communication. Water exchange outside the main downstream river path is represented only where the CaMa-Flood bifurcation and overbank-flow scheme defines additional routing links; those links are treated as explicit edges of the routing graph and are included in the basin-group decomposition. This means the GPU can process every catchment simultaneously, with each thread working on its own assigned catchment without needing to communicate or wait for others. In essence, the entire system of catchments can have its water depth updated at the same instant, fully leveraging the GPU's ability to handle millions of independent tasks at once. This allows the model to update the hydraulic state (water level, depth) for millions of catchments simultaneously, fully leveraging the GPU's massively parallel architecture and making it a highly efficient component of the simulation loop.

2.1.4 Communication and overhead

The remaining key challenge in scaling hydrodynamic models to multiple GPUs is managing communication and data

Algorithm 1 Parallel diagnosis of water depth from storage on GPU.

Require: For each catchment i: total storage $S_{\text{total}}[i]$, i.e. the sum of river storage, floodplain storage and incoming runoff.
Require: Lookup tables ($T_{\text{sto}}, T_{\text{dep}}, T_{\text{wdt}}, T_{\text{grad}}$) flattened into 1D arrays in memory; here T_{sto} stores cumulative storage thresholds, T_{dep} stores corresponding depths, T_{wdt} stores total widths, and T_{grad} stores width–depth gradients.
Ensure: For each catchment i: flood depth $D_{\text{fld}}[i]$, the diagnosed floodplain depth returned by the lookup/interpolation.

{1. Find profile level (k) using a branchless scan (parallel over catchments)}
$k \leftarrow -1$
for $j \leftarrow 0$ to N_{levels} **do**
  $o_{\text{tmp}} \leftarrow i \cdot (N_{\text{levels}} + 1) + j$ {temporary memory offset}
  $k \leftarrow$ **where**$(S_{\text{total}}[i] \geq T_{\text{sto}}[o_{\text{tmp}}], j, k)$ {branchless update}
end for
{2. Compute flood depth based on the identified level (k)}
if $k < 0$ **then**
  $D_{\text{fld}}[i] \leftarrow 0$
else
  {compute final memory offset o for level k}
  $o \leftarrow i \cdot (N_{\text{levels}} + 1) + k$
  $S_{\text{prev}} \leftarrow T_{\text{sto}}[o]$; $D_{\text{prev}} \leftarrow T_{\text{dep}}[o]$; $W_{\text{prev}} \leftarrow T_{\text{wdt}}[o]$; $G_{\text{prev}} \leftarrow T_{\text{grad}}[o]$
  if $k = N_{\text{levels}}$ **then**
    $D_{\text{fld}}[i] \leftarrow D_{\text{prev}} + (S_{\text{total}}[i] - S_{\text{prev}})/(W_{\text{prev}} \cdot L_{\text{riv}}[i])$
  else
    $\Delta W \leftarrow \sqrt{W_{\text{prev}}^2 + 2 \cdot (S_{\text{total}}[i] - S_{\text{prev}})/(G_{\text{prev}} \cdot L_{\text{riv}}[i])} - W_{\text{prev}}$
    $D_{\text{fld}}[i] \leftarrow D_{\text{prev}} + \Delta W \cdot G_{\text{prev}}$
  end if
end if

handling overhead. For multi-GPU runs, we decompose the reordered river network by basin group rather than by individual unit-catchment. Let $\mathcal{C}$ be the set of all unit-catchments and let B_m be the primitive basin draining to river mouth m. Cross-basin bifurcation links define an undirected graph among these primitive basins; each connected component of this graph is treated as one basin group G_k. The global state vector is then ordered as $\boldsymbol{x} = [\boldsymbol{x}_{G_1}, \boldsymbol{x}_{G_2}, \ldots, \boldsymbol{x}_{G_K}]$, where basin groups are sorted by decreasing size and the entries within each G_k follow the upstream-to-downstream topological order of the main river network. For P GPU ranks, basin groups are assigned by a longest-processing-time-first (LPT) greedy rule: processing G_k in decreasing $|G_k|$, we assign it to the rank with the smallest current load. The subdomain owned by rank p is therefore the union of all basin groups assigned to that rank. Because no bifurcation-formed basin group is split across ranks, all main-channel and bifurcation links used by the local inertial update connect unit-catchments inside the same rank. The layout keeps state arrays contiguous, balances the number of local unit-catchments among GPUs, and removes the need for peer-to-peer halo exchange during time stepping. With this multi-GPU decomposition in place, the remaining scalability bottlenecks no longer come from neighboring subdomains exchanging halo data, but from the global data movement and synchronization that still couple the ranks. Efficient GPU computation can then be easily undermined by bottlenecks in two main areas: (1) reading and distributing input data (e.g., runoff forcing) across all GPUs at each time step, and (2) synchronizing diagnostic variables and logging results, especially when using features like adaptive time-stepping. For input data, reading large files from disk is an I/O-bound operation that can stall the entire simulation if each GPU performs it independently. For outputs and internal diagnostics, frequent synchronization across GPUs – for example, to calculate a global time step or to verify mass balance – can introduce significant latency from communication, limiting overall performance.

To address these overheads, we implemented a suite of asynchronous and optimized data handling strategies (Fig. 5). For input data, we use a multi-process data loader. While the GPUs are computing the current step, background worker processes are already pre-fetching and preparing the data for the next step, placing it into a memory buffer. This ensures that when the GPUs are ready for new input, the data is already in memory and can be quickly broadcast, effectively hiding the I/O latency. For output, we offload file writing to separate threads, preventing the main simulation loop from stalling. By construction of the basin-group decomposition described above, every pair of unit-catchments coupled by the local inertial step, whether through the main network or through a bifurcation link, lies inside a single bifurcation-formed basin group, and every such group is owned by a single GPU rank. Reading the downstream water state therefore reduces to a contiguous local memory access, and no peer-to-peer halo exchange between GPUs is required at run time. The only inter-GPU traffic per step is the three collective operations enumerated below.

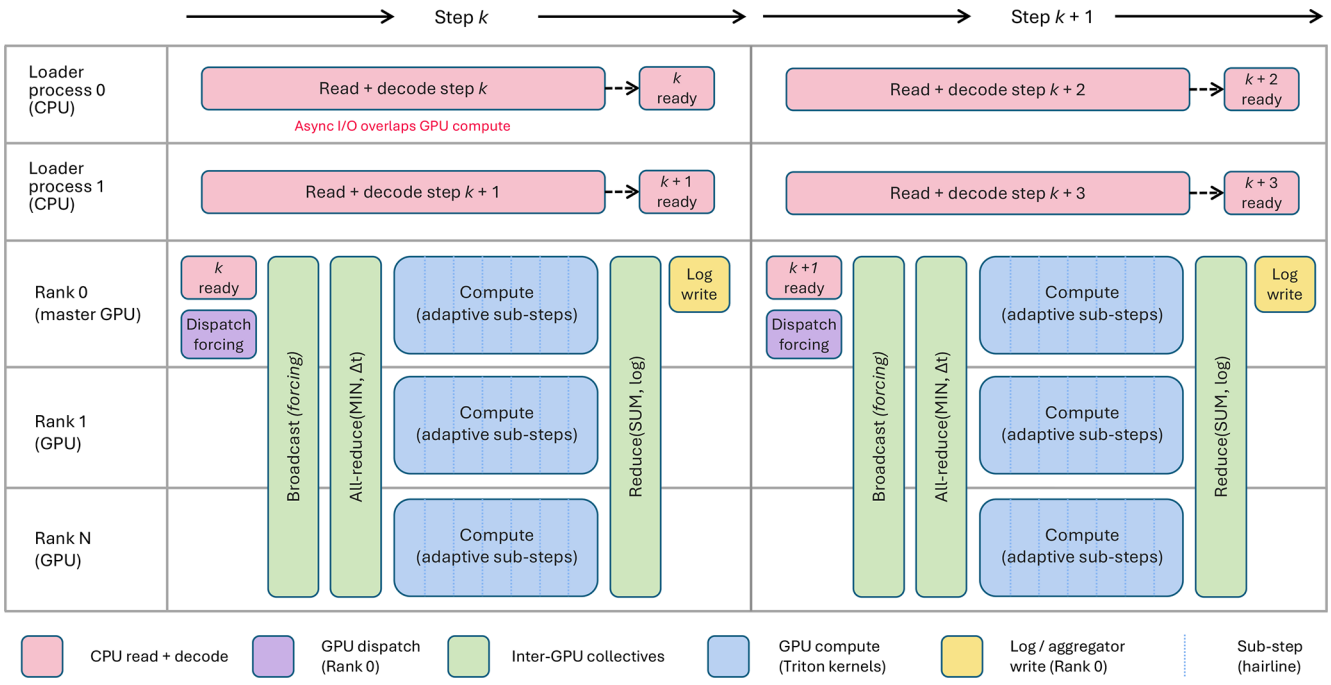

Figure 5. Runtime workflow for asynchronous input preparation, multi-GPU computation, and logging over two consecutive time steps.

For logging and synchronization, we employ a local accumulation and end-of-step reduction strategy. Instead of performing a global reduction at every sub-step, each GPU accumulates diagnostic variables in a local buffer. All GPUs therefore share the same global sub-step, taken as the minimum required Δt across the whole domain, so that the GPU integration trajectory remains identical to the reference CPU run regardless of the number of GPUs. Only at the end of a full time step is a single, collective communication performed to aggregate the results for logging. This minimizes cross-GPU traffic, ensuring that communication occurs only when necessary. As a result, each time step involves three collective operations: the runoff broadcast, the all-rank minimum reduction used to select the global adaptive sub-step, and the end-of-step gather/reduction of diagnostic output. The runoff broadcast carries only N_r floats per step, amounting to on the order of tens of megabytes at the resolutions tested, and is dominated by inter-GPU bandwidth, so its cost stays a small fraction of the per-step compute even at multi-GPU scale. This design significantly reduces communication overhead and allows the model to scale efficiently across multiple GPUs.

2.2 Flexible customization

CaMa-Flood-GPU also allows customizing the simulation domain and outputs to focus on specific geographic locations of interest, such as gauge stations (Fig. 6). Instead of running the model on the entire globe, users can define a domain centered on their locations of interest by providing a list of catchment IDs or coordinates corresponding to gauges. The model then trims the global catchment graph to preserve only the subgraph that hydrologically contributes to those locations, ensuring that upstream water transport relevant to the gauges is preserved while unrelated portions of the network are removed. This strategy minimizes unnecessary computation and enables efficient, targeted simulations for comparison with observations or for calibration at selected sites.

Beyond spatial focus, the output variables and statistics produced by CaMa-Flood-GPU are highly configurable. The user can specify which model variables to record and choose what statistics to save for each variable on each time step. Supported statistical measures include, for instance, the mean over the time step (averaging across all sub-steps), the maximum and minimum during the time step, or simply the final value at the end of the time step. Moreover, instead of saving full 2D fields for all catchments globally, the user can limit the output to particular locations or catchments – such as the set of gauge points mentioned above, or any arbitrary subset of catchments that are of interest. This selective output greatly reduces storage requirements and post-processing effort for large simulations. To implement this efficiently, we developed a `StatisticsAggregator` class that performs on-the-fly computation of the requested statistics at the end of each time step, only for the selected output catchments, and streams the results to disk incrementally. The `StatisticsAggregator` uses a fused-kernel approach on the GPU to minimize overhead: essentially, it dynamically generates a single optimized GPU kernel (using the Triton just-in-time compiler) that will compute all the requested

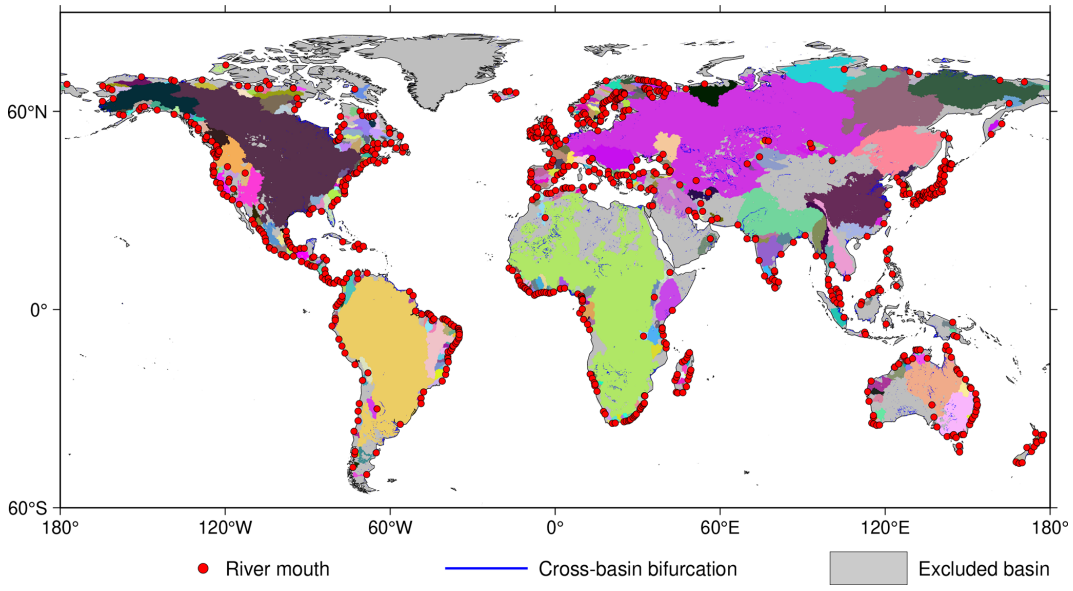

Figure 6. Example of a customized simulation domain constructed after point-of-interest filtering and basin-group domain decomposition. The unit-catchment is the computational element; primitive basins are collections of unit-catchments draining to the same river mouth before cross-basin bifurcations are considered. Coloured patches denote retained basin groups, each containing the upstream unit-catchments needed for at least one point of interest; grey areas mark excluded basin groups. Some retained patches are spatially large because a downstream point of interest can require preserving an entire upstream primitive basin, and cross-basin bifurcation links can merge several primitive basins into one basin group. Markers indicate river-mouth locations and blue lines denote cross-basin bifurcation links.

statistics for a given group of saved catchments in one pass through the data. In other words, instead of launching separate GPU operations for each variable and each statistic, the aggregator combines them into one operation per group of output catchments. After each time step, the computed statistics are immediately written to an output NetCDF file in a streaming fashion. Each GPU writes its own output for the catchments it handles, and the results are indexed by time and location. Because writing is done incrementally and in parallel (using asynchronous write operations, as described in Sect. 2.1.4), the approach keeps memory usage bounded and overlaps I/O with computation. By adjusting the list of saved catchments and the types of statistics collected, the user can obtain exactly the desired outputs from the simulation with minimal performance penalty. This flexible output system enables, for instance, efficient calibration or validation runs (focusing only on certain gauge stations and error metrics) and generally allows the model to integrate seamlessly into workflows where specific results need to be extracted on the fly.

3 Results and evaluation

3.1 Experimental setup

Our evaluation targets the two aspects that matter most for a production-quality global hydrodynamic model: simulation speed and numerical fidelity. To make the comparison fair and reproducible, CPU and GPU runs within each comparison share the same global parameterization (derived from MERIT Hydro), identical forcing and simulation periods, and a controlled I/O setup that minimizes non-compute bottlenecks. We benchmark across four representative compute environments to reflect common usage scenarios: (i) a personal custom workstation with a GeForce 4070 Ti GPU (Ubuntu via Windows Subsystem for Linux 2), (ii) GPU servers with Tesla V100 GPUs, (iii) GPU servers with A100 GPUs, and (iv) CPU-only servers equipped with Intel Xeon processors. All three server configurations run CentOS 7. The CPU version used for comparison is CaMa-Flood v4.23. The software demonstrates good portability: it runs smoothly both on a personal workstation with administrator privileges, where the latest CUDA/toolchains are available, and on shared servers where users typically lack root access and must rely on older system packages. Hardware details are summarized in Table 1.

To make the subsequent performance and numerical-stability analyses reproducible, we first summarize the common experimental setup and then distinguish the choices specific to the wall-clock benchmark and to the stability comparison. All performance benchmarks use CaMa-Flood global parameter sets derived from MERIT Hydro (Yamazaki et al., 2019) at four spatial resolutions, 15, 6, 3 and 1 arcmin; the corresponding numbers of unit-catchments, bifurcation links and approximate adaptive sub-steps are listed in Table 2. The same model options are enabled in both implementations:

Table 1. Summary of computing nodes and their hardware configurations.

Label	CPU	CPU Memory (per node)	GPU	CPU Cores (per node)	GPUs (per node)
4070 Ti	Intel Core i7-13700	64 GB	NVIDIA GeForce RTX 4070 Ti (12 GB)	16	1
V100	Dual Intel Xeon E5-2640 v4	128 GB	NVIDIA Tesla V100 (16 GB)	20	4
A100	Dual AMD EPYC 7543	256 GB	NVIDIA A100 (40 GB)	64	4
CPU	Dual Intel Xeon 6248R	256 GB	–	48	–

Table 2. Catchment and bifurcation statistics at different spatial resolutions.

Spatial resolution	Number of catchments	Number of bifurcations	Number of sub-steps (approximate)
15 arcmin	252 383	17 242	260
6 arcmin	1 562 463	207 858	590
3 arcmin	6 222 566	722 883	1200
1 arcmin	55 812 946	6 864 274	5300

adaptive sub-step integration, bifurcation routing and on-the-fly logging of the selected diagnostic output.

For the speed benchmark, the runoff forcing is the daily 1° binary sample runoff distributed with the CPU CaMa-Flood release; the sample forcing includes year 2000 and was prepared from the output of the Ensemble Land State Estimator (ELSE) (Kim et al., 2009). To limit non-computational bottlenecks, we also control the I/O path so that the wall-clock comparison reflects routing performance rather than file-format overhead. The GPU implementation overlaps forcing input with computation through its asynchronous dataloader, whereas the CPU reference reads forcing synchronously. Each run therefore saves only one key output variable, and the CPU reference writes that output in binary format. The CPU reference is CaMa-Flood v4.23 compiled with Intel Fortran using the flags "-O3 -fp-model precise" and run in hybrid MPI/OpenMP mode; we tested several MPI/OpenMP combinations and used the fastest configuration for each CPU benchmark, with 16 MPI ranks following the official CaMa-Flood recommendation on the multi-core hosts. GPU runs use a Triton block size of 128, which specifies the number of unit-catchments processed by one kernel instance. All reported wall-times are means over five repeated runs.

For the numerical-stability comparison, we use daily forcing series prepared from 0.1° ERA5-Land NetCDF runoff (Muñoz-Sabater et al., 2021) and from the eartH2Observe (E2O) Tier-1 ensemble runoff (Schellekens et al., 2017). The 1980–2014 period is adopted to remain consistent with the temporal coverage of the E2O Tier-1 dataset. For each numerical-stability comparison, CPU and GPU runs use the same 6 arcmin parameter set, identical forcing, initial states and adaptive sub-step settings, with the bifurcation module enabled in both runs. The E2O and ERA5-Land numerical-stability experiments are not used to quantify wall-clock performance overhead from forcing input; they are used to test numerical and input consistency under independent runoff products.

3.2 Performance comparison

Table 3 summarizes the benchmark timing results for a 1-year simulation period at the four resolutions on various hardware configurations: the workstation GPU, the V100 server with 1–4 GPUs, the A100 server with 1–4 GPUs, and the CPU-only server (a combination of four CPU nodes) with different numbers of CPU cores. At the coarsest 15 arcmin resolution the gap between server-class GPUs and CPUs is small: on V100, 1–4 GPUs take 2 min 21 s–2 min 24 s; on A100, 1–4 GPUs take 1 min 14 s–1 min 31 s; and the CPU configuration with 48/96/192 cores takes 2 min 19 s/1 min 32 s/1 min 11 s, respectively. The only clear outlier is the workstation-grade 4070 Ti, finishing 15 arcmin in about 38 s. This behavior is expected for a light problem where wall-time is dominated by non-compute components and frequency/I/O headroom on desktops (e.g., aggressive CPU turbo and SSD-backed reads) can outweigh architectural differences. Focusing on the strongest CPU setup (192 cores), performance still lags significantly behind even single-GPU runs. At the 6 arcmin resolution, the CPU requires 18 min 37 s, whereas a single V100 finishes in 7 min 50 s and a single A100 in 4 min 24 s. The gap widens at finer resolutions: at 3 arcmin, the CPU takes 2 h 45 min 19 s, compared to 11 min 47 s on 4 × V100 and 7 min 21 s on 4 × A100; at 1 arcmin, the CPU needs 140 h 58 min 20 s, versus 6 h 51 min 36 s on 4 × V100 and 3 h 51 min 24 s on 4 × A100. At the 1 arcmin resolution, the aggregate GPU memory footprint is about 30 GB in total, including hydrodynamic state arrays, auxiliary routing and bifurcation buffers, runoff-mapping buffers, and temporary work arrays. This footprint is distributed across GPU ranks by the LPT assignment of bifurcation-formed basin groups. Cases exceeding the available single-card GPU memory are marked OOM in Table 3. These results demonstrate that multi-GPU acceleration yields order-of-magnitude speedups, with 4 × A100 achieving up to 36.6 × faster runtimes than the best CPU configuration. Multi-GPU experiments reveal generally good scaling from 1 to 4 GPUs at finer resolution. At 3 arcmin resolu-

tion, runtimes on the V100 decrease from 40 min 5 s (1 GPU) to 11 min 46 s (4 GPUs), corresponding to a 3.4 × speedup. On the A100, the same case improves from 22 min 49 s to 7 min 21 s, a 3.1 × speedup. At the finest 1 arcmin resolution, where the workload and sub-step count are largest, scaling is even stronger: 816 min with 1 A100 is reduced to 231 min with 4 A100s, achieving a 3.5 × speedup. In contrast, at the coarsest 15 arcmin resolution, adding GPUs can degrade performance (e.g., 4 A100s slower than a single card) because parallelization overheads dominate once the per-GPU workload becomes too small. These results highlight the balance between computation and overhead (I/O, reductions, inter-GPU communication). In summary, CaMa-Flood-GPU delivers minutes-to-hours runtimes for global domains that previously required hours-to-days on large CPU machines, with the biggest wins at fine resolutions and longer runs where computation dwarfs overhead.

3.3 Numerical stability

The numerical-stability comparison follows the setup described in Sect. 3.1 and covers 1980–2014 with two daily runoff forcing series: E2O Tier-1 ensemble runoff at 0.25° and a daily series prepared from ERA5-Land surface runoff at 0.1°. Using these two forcing datasets, we compare CaMa-Flood-GPU with the reference CPU implementation through spatial mean fields and station hydrographs, and assess whether floating-point differences remain bounded over the multi-decadal simulation.

Figure 7 presents the CPU–GPU comparison under E2O runoff. The CPU and GPU mean river outflow fields (Fig. 7a, b) are visually indistinguishable, and the relative-difference map in Fig. 7c is dominated by the gray "below-noise-floor" band of ± 0.001 %, with non-zero values appearing mainly on the largest river stems. Figure 7d–f repeats the comparison for floodplain outflow and Fig. 7g–i for river depth, both with the same noise-floor behaviour and the same concentration of residual signal on the main channels of large basins. Across the three variables, the field-mean relative difference remains below 10^{-3} %. Figure 8 reproduces the same comparison under ERA5-Land runoff. The residual magnitude and spatial pattern remain essentially unchanged between the two forcing products, although the products differ in resolution and temporal variability. This consistency indicates that the residuals arise from the floating-point ordering of the routing calculation rather than from the runoff forcing itself.

Figures 9 and 10 compare simulated discharge for the final four years of the full simulation at six GRDC gauge locations spanning four orders of magnitude in drainage area. At every gauge location, the CPU and GPU curves overlay each other almost exactly, and the right-hand red axis shows the day-by-day CPU–GPU difference at a magnified scale. The residual amplitude follows the depth of the upstream reduction at the station location rather than the absolute discharge magnitude. On the large main stems, GPU `atomic_add` operations are associative and commutative but their execution order is not deterministic, whereas the CPU reduction follows a deterministic order. The two reductions therefore return slightly different floating-point bit patterns, producing a small bounded difference at the $m^3\ s^{-1}$ level, three to four orders of magnitude smaller than the simulated discharge. At the mid-scale tributary and headwater gauge locations, the reduction is shallower and the red difference curve collapses toward zero for most of the displayed period. Neither forcing shows a monotonic component in the residual over the multi-year window. We therefore do not see evidence of numerical drift accumulating through the integration.

Mass conservation on the irregular unit-catchment graph is preserved by construction. The GPU implementation applies the same source-to-target routing fluxes as the CPU algorithm: main-channel fluxes are accumulated through `scatter_add` reductions, and bifurcation fluxes are accumulated through `atomic_add` reductions rather than overwriting destination storage. Each outgoing flux is therefore added to the corresponding downstream storage term, with the CPU–GPU difference limited to the floating-point summation order discussed above. Because bifurcation routing is enabled together with the main-channel routing in the numerical-stability runs, the GPU executes the coupled routing workflow that includes both the base main-channel operations and the bifurcation-specific accumulation pathway. The comparison therefore provides an indirect CPU–GPU consistency check of the complete bifurcation-enabled model configuration, rather than a separate evaluation of individual bifurcation split ratios or point-by-point bifurcation hydraulics. The no-bifurcation configuration is a strict subset of these operations, so agreement with bifurcation enabled also supports agreement for the simpler routing case.

Taken together, the field comparisons, station hydrographs and routing-flux accounting show that CaMa-Flood-GPU reproduces the CPU reference without detectable numerical drift over multi-decadal integrations, across both forcing products and with bifurcations enabled.

4 Conclusions

In this study, we sought to identify and overcome the fundamental challenges hindering the application of GPU acceleration to global river models. We have successfully addressed this by developing CaMa-Flood-GPU, a version of the global flood model refactored for massively parallel hardware. Our work systematically pinpointed the primary bottlenecks for GPU computation – the irregular network topology, the mismatch between gridded inputs and catchments, the conditional logic in water depth diagnosis, and data communication overheads. In response, we explored and implemented a suite of targeted algorithmic solutions: reframing river routing as a parallel scatter-add operation, replacing traditional runoff interpolation with efficient sparse matrix-vector mul-

Table 3. Performance comparison across different hardware configurations for the year 2000 (1-year simulation).

Label	GPUs	CPU cores	15 arcmin	6 arcmin	3 arcmin	1 arcmin
4070 Ti	1	–	38 s	6 min 46 s	48 min 42 s	*
V100	1	–	2 min 21 s	6 min 42 s	40 min 5 s	*
	2	–	2 min 17 s	4 min 26 s	21 min 7 s	*
	3	–	2 min 23 s	4 min 22 s	14 min 58 s	9 h 1 min 58 s
	4	–	2 min 24 s	4 min 16 s	11 min 47 s	6 h 51 min 36 s
A100	1	–	1 min 14 s	4 min 24 s	22 min 49 s	13 h 36 min 2 s
	2	–	1 min 17 s	2 min 58 s	12 min 21 s	6 h 56 min 0 s
	3	–	1 min 23 s	2 min 35 s	8 min 56 s	4 h 50 min 57 s
	4	–	1 min 31 s	2 min 34 s	7 min 21 s	3 h 51 min 24 s
CPU	–	48	2 min 19 s	48 min 31 s	6 h 29 min 1 s	> 14 d
	–	96	1 min 32 s	28 min 50 s	3 h 59 min 16 s	198 h 0 min 12 s
	–	192	1 min 11 s	18 min 37 s	2 h 45 min 19 s	140 h 58 min 20 s

* Indicates that the task exceeded the available GPU memory.

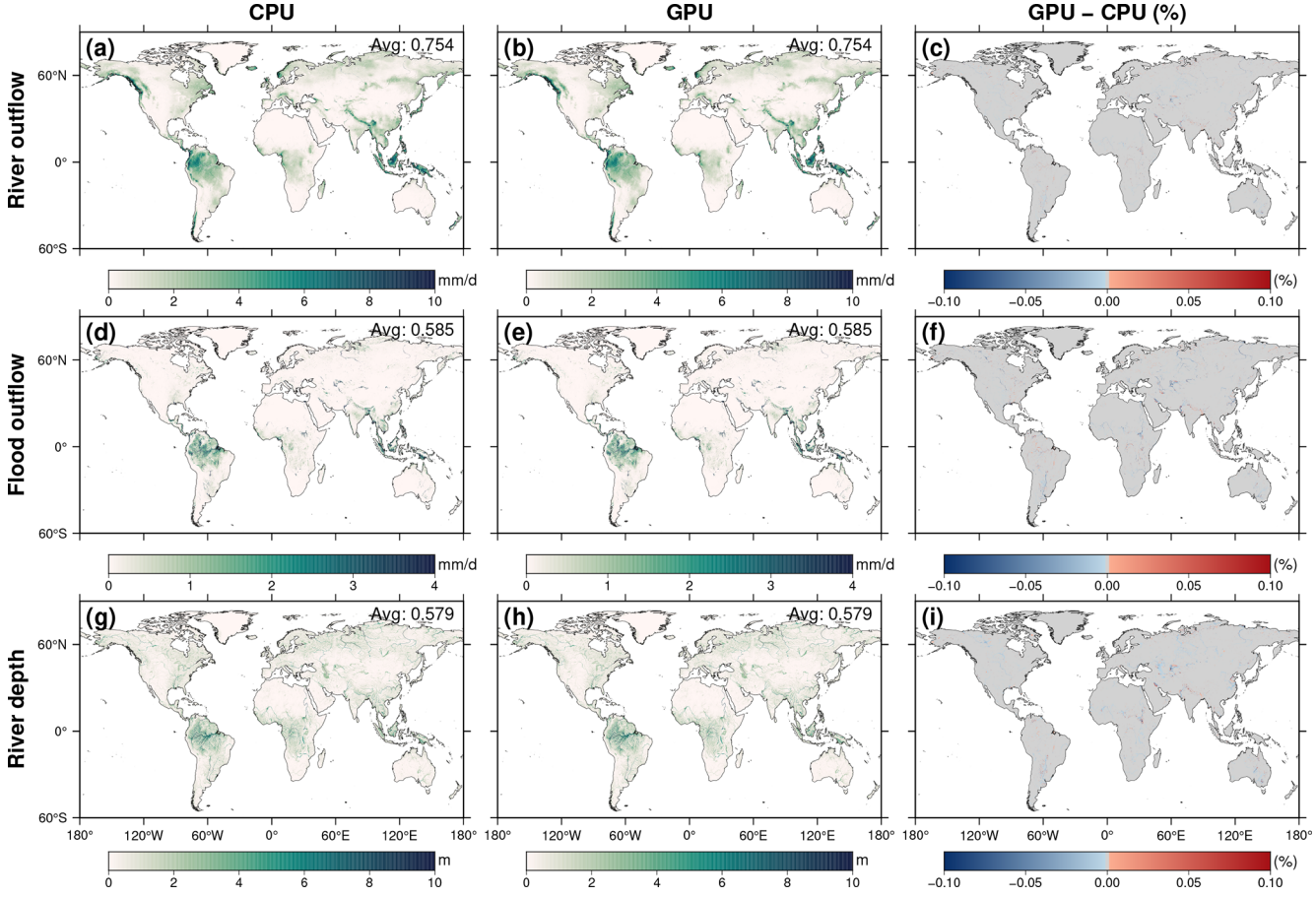

Figure 7. CPU vs. GPU mean fields with the bifurcation module enabled, under E2O Tier-1 ensemble-mean runoff at 0.25° for 1980–2014. **(a–c)** Mean river outflow for CPU, GPU and their relative difference. **(d–f)** Mean floodplain outflow for CPU, GPU and their relative difference. **(g–i)** Mean river depth for CPU, GPU and their relative difference. The third-column colour bar is bipolar with a flat gray band at ±0.001 % to mark the floating-point noise floor.

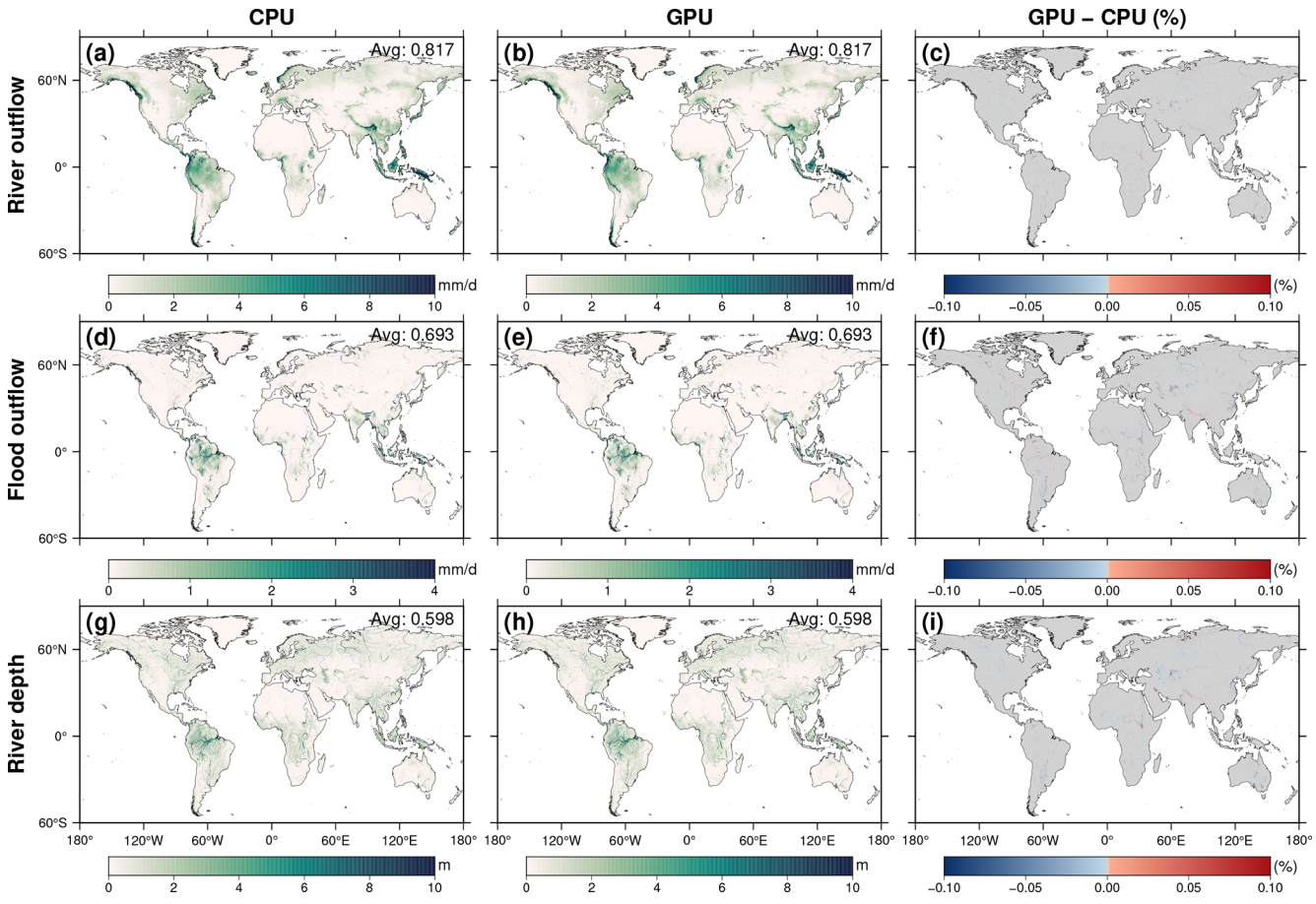

Figure 8. Same as Fig. 7, but using the daily ERA5-Land runoff series prepared at 0.1° for 1980–2014.

tiplication, designing a branchless kernel for water depth diagnosis, and minimizing data handling bottlenecks through asynchronous I/O. These solutions, implemented using Triton and PyTorch, preserve the model's physical integrity, ensuring numerical differences from the CPU version are minimal.

The resulting performance improvements fulfill our objective of making kilometer-scale global simulations computationally practical. Simulations at high resolutions (e.g., 3 arcmin) that previously took hours on a multi-core server can now run in minutes on a multi-GPU system, making simulations at 1 arcmin resolution feasible within hours. This leap in performance broadens the model's applicability, enabling more detailed analyses and the generation of large ensembles for robust uncertainty assessment. The model's flexible, modular design further allows researchers to tailor it to their needs, facilitating its integration into larger Earth system modeling frameworks.

Looking ahead, there are several avenues for further development. First, additional physics and processes could be incorporated. CaMa-Flood-GPU realizes this through a sub-module layer in which each physical process is encapsulated as a self-contained component. A component declares its own per-unit-catchment state fields, registers one update kernel called once per time step by the main integrator, and contributes any fluxes back to the channel network through the same `scatter_add` reduction used by the core routing. New processes are therefore added by writing one such sub-module rather than by modifying the existing flood-routing kernels or the time-step loop. For reservoir operation, the CaMa-Flood community has already developed schemes such as the global flood-control reservoir module of Hanazaki et al. (2022) and the H08–CaMa-Flood coupling of Shin et al. (2020). These schemes both express reservoir storage and release as per-time-step updates at the unit-catchment level, which maps directly onto this interface. Sediment transport schemes such as the global sediment-dynamics model of Hatono and Yoshimura (2020) similarly reduce, on the GPU side, to a small number of additional catchment-level state fields and one update kernel. These extensions can therefore be ported into the GPU framework as optional modules without rewriting the existing code.

Second, a natural research direction for CaMa-Flood-GPU is end-to-end differentiable global routing. The integrator is built in PyTorch and uses custom Triton kernels for the core routing solver: the PyTorch layer already provides automatic

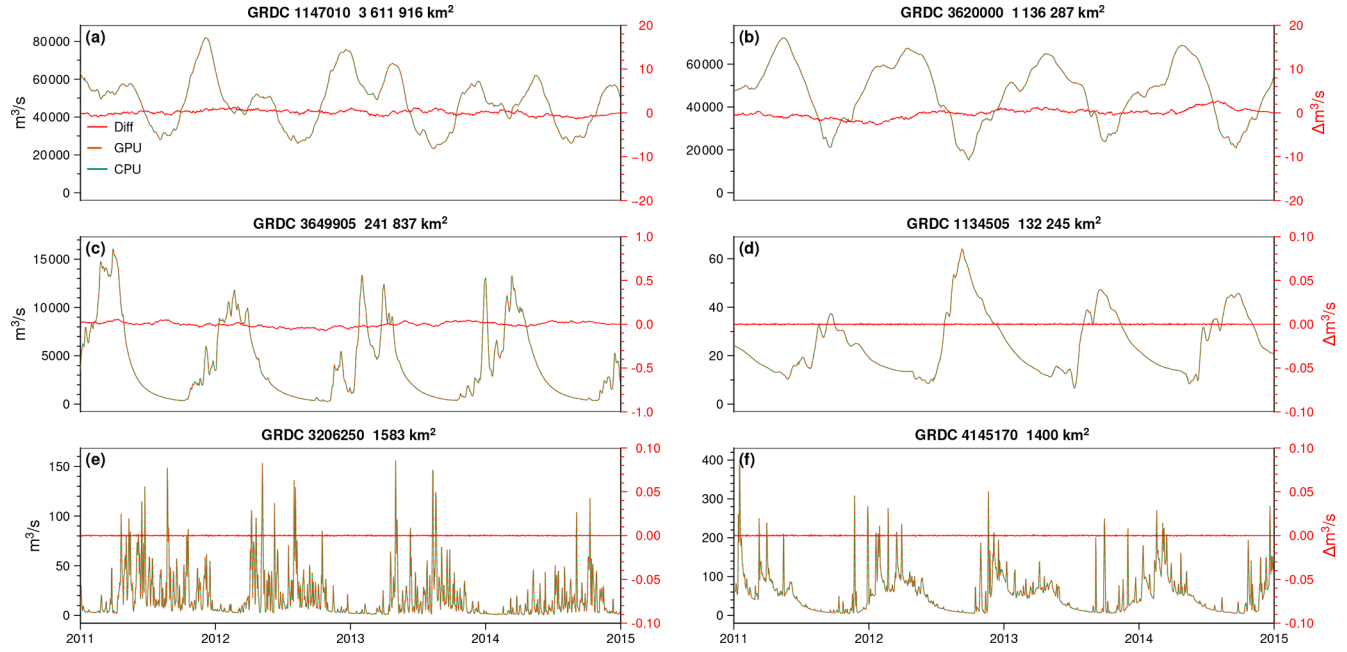

Figure 9. Simulated daily river discharge at the locations of six GRDC gauges spanning four orders of magnitude in drainage area, for the last four years of the 1980–2014 simulation under E2O Tier-1 runoff at 0.25°. The gauge panels are ordered by drainage area from large to small. CPU (green solid) and GPU (orange dashed) curves overlap; the difference is plotted as a red solid line with values given on the right-hand axis.

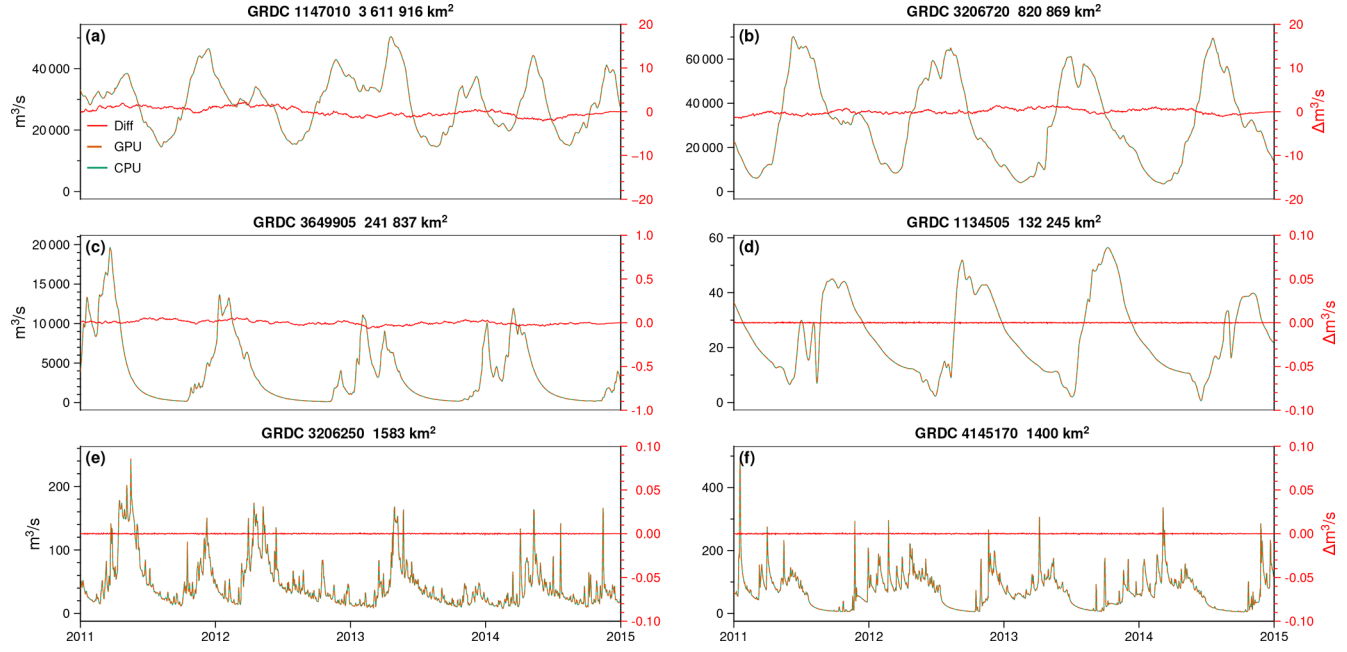

Figure 10. Same as Fig. 9, but with ERA5-Land surface runoff at 0.1°.

differentiation for everything expressed as standard tensor operations, and the Triton kernels can be paired with hand-written backward kernels in the same way as in the wider PyTorch-Triton ecosystem. Once that pairing is in place, parameters such as Manning roughness, river width, floodplain elevation profile and even reservoir operating rules can be calibrated by gradient descent against discharge or altimetry observations, or trained jointly with PyTorch-based land-surface or AI components, in line with the differentiable-geoscience programme advocated by Shen et al. (2023). As

part of our commitment to community engagement, we will provide thorough documentation and example cases to encourage broader adoption.

In summary, the development of CaMa-Flood-GPU successfully bridges the gap between hydrological modeling and high-performance computing. It provides a scalable, physically consistent foundation for global flood simulations at resolutions and runtimes that were previously impractical. We believe this tool, offering a unique combination of speed, scale, and physical fidelity, will help advance global-scale hydrological assessments and risk analysis under climate change.

Code and data availability. The source code and scripts for CaMa-Flood-GPU are available from Zenodo at https://doi.org/10.5281/zenodo.18137445 (Kang, 2026) under the Apache 2.0 license. The CPU version of CaMa-Flood (v4.23) used in this study is available from Zenodo at https://doi.org/10.5281/zenodo.14214989 (Yamazaki et al., 2024). The input datasets used for the simulations, including the CaMa-Flood river topography maps, catchment parameters, and runoff forcing data, are publicly available from the CaMa-Flood project website (https://global-hydrodynamics.github.io/CaMa-Flood/, last access: 25 June 2026).

Author contributions. S.K. coded the GPU model, performed the experiments, and wrote the initial draft. J.Y. and D.Y. contributed to the model design, provided supervision, assisted with code optimization and debugging. All authors contributed to review and editing of the manuscript.

Competing interests. The contact author has declared that none of the authors has any competing interests.

Disclaimer. Publisher's note: Copernicus Publications remains neutral with regard to jurisdictional claims made in the text, published maps, institutional affiliations, or any other geographical representation in this paper. The authors bear the ultimate responsibility for providing appropriate place names. Views expressed in the text are those of the authors and do not necessarily reflect the views of the publisher.

Acknowledgements. The numerical calculations in this paper have been performed on the supercomputing system in the Supercomputing Centre of Wuhan University. We also acknowledge the open-source community behind PyTorch and Triton, which made the GPU implementation efficient and attainable.

Financial support. JY is supported by the National Natural Science Foundation of China (grant nos. T2522026, 52441902, W2521014). DY received support from the NSFC-JSPS Bilateral Joint Research Project (grant no. JPJSBP120257408). This work is also supported by the National Natural Science Foundation of China (grant nos. 52361145864, W2421111), the Natural Science Foundation of Hubei Province (grant no. 2024AFA055), and the Major Science and Technology Project of the Ministry of Water Resources (grant no. SKS-2025043).

Review statement. This paper was edited by Thomas B. Wild and reviewed by two anonymous referees.

References

Alvanos, M. and Christoudias, T.: GPU-accelerated atmospheric chemical kinetics in the ECHAM/MESSy (EMAC) Earth system model (version 2.52), Geosci. Model Dev., 10, 3679–3693, https://doi.org/10.5194/gmd-10-3679-2017, 2017.

Bates, P. D., Horritt, M. S., and Fewtrell, T. J.: A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling, J. Hydrol., 387, 33–45, https://doi.org/10.1016/j.jhydrol.2010.03.027, 2010.

Caviedes-Voullième, D., Morales-Hernández, M., Norman, M. R., and Özgen-Xian, I.: SERGHEI (SERGHEI-SWE) v1.0: a performance-portable high-performance parallel-computing shallow-water solver for hydrology and environmental hydraulics, Geosci. Model Dev., 16, 977–1008, https://doi.org/10.5194/gmd-16-977-2023, 2023.

Collins, E. L., David, C. H., Riggs, R., Allen, G. H., Pavelsky, T. M., Lin, P., Pan, M., Yamazaki, D., Meentemeyer, R. K., and Sanchez, G. M.: Global patterns in river water storage dependent on residence time, Nat. Geosci., 17, 433–439, https://doi.org/10.1038/s41561-024-01421-5, 2024.

De Almeida, G. A. M., Bates, P., Freer, J. E., and Souvignet, M.: Improving the stability of a simple formulation of the shallow water equations for 2-D flood modeling, Water Resour. Res., 48, 2011WR011570, https://doi.org/10.1029/2011WR011570, 2012.

Emerton, R. E., Stephens, E. M., Pappenberger, F., Pagano, T. C., Weerts, A. H., Wood, A. W., Salamon, P., Brown, J. D., Hjerdt, N., Donnelly, C., Baugh, C. A., and Cloke, H. L.: Continental and global scale flood forecasting systems, WIREs Water, 3, 391–418, https://doi.org/10.1002/wat2.1137, 2016.

Hamitouche, M., Fosser, G., Anav, A., He, C., and Lin, T.-S.: Impact of runoff schemes on global flow discharge: a comprehensive analysis using the Noah-MP and CaMa-Flood models, Hydrol. Earth Syst. Sci., 29, 1221–1240, https://doi.org/10.5194/hess-29-1221-2025, 2025.

Hanazaki, R., Yamazaki, D., and Yoshimura, K.: Development of a reservoir flood control scheme for global flood models, J. Adv. Model. Earth Sy., 14, e2021MS002944, https://doi.org/10.1029/2021MS002944, 2022.

Hatono, M. and Yoshimura, K.: Development of a global sediment dynamics model, Prog. Earth Planet. Sc., 7, 59, https://doi.org/10.1186/s40645-020-00368-6, 2020.

Heinicke, S., Volkholz, J., Schewe, J., Gosling, S. N., Müller Schmied, H., Zimmermann, S., Mengel, M., Sauer, I. J., Burek, P., Chang, J., Kou-Giesbrecht, S., Grillakis, M., Guillaumot, L., Hanasaki, N., Koutroulis, A., Otta, K., Qi, W., Satoh, Y., Stacke,

T., Yokohata, T., and Frieler, K.: Global hydrological models continue to overestimate river discharge, Environ. Res. Lett., 19, 074005, https://doi.org/10.1088/1748-9326/ad52b0, 2024.

Hirabayashi, Y., Mahendran, R., Koirala, S., Konoshima, L., Yamazaki, D., Watanabe, S., Kim, H., and Kanae, S.: Global flood risk under climate change, Nat. Clim. Change, 3, 816–821, https://doi.org/10.1038/nclimate1911, 2013.

Hokkanen, J., Kollet, S., Kraus, J., Herten, A., Hrywniak, M., and Pleiter, D.: Leveraging HPC accelerator architectures with modern techniques: hydrologic modeling on GPUs with ParFlow, Comput. Geosci., 25, 1579–1590, https://doi.org/10.1007/s10596-021-10051-4, 2021.

Huang, S. and Hattermann, F. F.: Coupling a global hydrodynamic algorithm and a regional hydrological model for large-scale flood inundation simulations, Hydrol. Res., 49, 438–449, https://doi.org/10.2166/nh.2017.061, 2018.

Hunter, N. M., Horritt, M. S., Bates, P. D., Wilson, M. D., and Werner, M. G.: An adaptive time step solution for raster-based storage cell modelling of floodplain inundation, Adv. Water Resour., 28, 975–991, https://doi.org/10.1016/j.advwatres.2005.03.007, 2005.

Kang, S.: CaMa-Flood-GPU, Zenodo [code], https://doi.org/10.5281/zenodo.18137445, 2026.

Kang, S., Yin, J., Slater, L., Liu, P., Sun, F., Liu, D., and Xia, J.: Global Flood Projection and Socioeconomic Implications Under a Deep Learning Framework, Water Resour. Res., 61, https://doi.org/10.1029/2024wr037139, 2025.

Kim, H., Yeh, P. J.-F., Oki, T., and Kanae, S.: Role of rivers in the seasonal variations of terrestrial water storage over global basins, Geophys. Res. Lett., 36, L17402, https://doi.org/10.1029/2009GL039006, 2009.

Kimura, Y., Hirabayashi, Y., Kita, Y., Zhou, X., and Yamazaki, D.: Methodology for constructing a flood-hazard map for a future climate, Hydrol. Earth Syst. Sci., 27, 1627–1644, https://doi.org/10.5194/hess-27-1627-2023, 2023.

Marthews, T. R., Dadson, S. J., Clark, D. B., Blyth, E. M., Hayman, G. D., Yamazaki, D., Becher, O. R. E., Martínez-de la Torre, A., Prigent, C., and Jiménez, C.: Inundation prediction in tropical wetlands from JULES-CaMa-Flood global land surface simulations, Hydrol. Earth Syst. Sci., 26, 3151–3175, https://doi.org/10.5194/hess-26-3151-2022, 2022.

Mateo, C. M. R., Yamazaki, D., Kim, H., Champathong, A., Vaze, J., and Oki, T.: Impacts of spatial resolution and representation of flow connectivity on large-scale simulation of floods, Hydrol. Earth Syst. Sci., 21, 5143–5163, https://doi.org/10.5194/hess-21-5143-2017, 2017.

Mizukami, N., Clark, M. P., Sampson, K., Nijssen, B., Mao, Y., McMillan, H., Viger, R. J., Markstrom, S. L., Hay, L. E., Woods, R., Arnold, J. R., and Brekke, L. D.: mizuRoute version 1: a river network routing tool for a continental domain water resources applications, Geosci. Model Dev., 9, 2223–2238, https://doi.org/10.5194/gmd-9-2223-2016, 2016.

Morales-Hernández, M., Sharif, M. B., Kalyanapu, A., Ghafoor, S. K., Dullo, T. T., Gangrade, S., Kao, S.-C., Norman, M. R., and Evans, K. J.: TRITON: a multi-GPU open source 2D hydrodynamic flood model, Environ. Model. Softw., 141, 105034, https://doi.org/10.1016/j.envsoft.2021.105034, 2021.

Muñoz-Sabater, J., Dutra, E., Agustí-Panareda, A., Albergel, C., Arduini, G., Balsamo, G., Boussetta, S., Choulga, M., Harrigan, S., Hersbach, H., Martens, B., Miralles, D. G., Piles, M., Rodríguez-Fernández, N. J., Zsoter, E., Buontempo, C., and Thépaut, J.-N.: ERA5-Land: a state-of-the-art global reanalysis dataset for land applications, Earth Syst. Sci. Data, 13, 4349–4383, https://doi.org/10.5194/essd-13-4349-2021, 2021.

Neal, J., Hawker, L., Savage, J., Durand, M., Bates, P., and Sampson, C.: Estimating River Channel Bathymetry in Large Scale Flood Inundation Models, Water Resour. Res., 57, e2020WR028301, https://doi.org/10.1029/2020WR028301, 2021.

Rong, Y., Bates, P., and Neal, J.: GPU-Accelerated Urban Flood Modeling Using a Nonuniform Structured Grid and a Super Grid Scale River Channel, Water Resour. Res., 60, e2023WR036128, https://doi.org/10.1029/2023WR036128, 2024.

Schellekens, J., Dutra, E., Martínez-de la Torre, A., Balsamo, G., van Dijk, A., Sperna Weiland, F., Minvielle, M., Calvet, J.-C., Decharme, B., Eisner, S., Fink, G., Flörke, M., Peßenteiner, S., van Beek, R., Polcher, J., Beck, H., Orth, R., Calton, B., Burke, S., Dorigo, W., and Weedon, G. P.: A global water resources ensemble of hydrological models: the eartH2Observe Tier-1 dataset, Earth Syst. Sci. Data, 9, 389–413, https://doi.org/10.5194/essd-9-389-2017, 2017.

Sharifian, M. K., Kesserwani, G., Chowdhury, A. A., Neal, J., and Bates, P.: LISFLOOD-FP 8.1: new GPU-accelerated solvers for faster fluvial/pluvial flood simulations, Geosci. Model Dev., 16, 2391–2413, https://doi.org/10.5194/gmd-16-2391-2023, 2023.

Shen, C., Appling, A. P., Gentine, P., Bandai, T., Gupta, H., Tartakovsky, A., Baity-Jesi, M., Fenicia, F., Kifer, D., Li, L., Liu, X., Ren, W., Zheng, Y., Harman, C. J., Clark, M., Farthing, M., Feng, D., Kumar, P., Aboelyazeed, D., Rahmani, F., Song, Y., Beck, H. E., Bindas, T., Dwivedi, D., Fang, K., Höge, M., Rackauckas, C., Mohanty, B., Roy, T., Xu, C., and Lawson, K.: Differentiable modelling to unify machine learning and physical models for geosciences, Nat. Rev. Earth Environ., 4, 552–567, https://doi.org/10.1038/s43017-023-00450-9, 2023.

Shin, S., Pokhrel, Y., Yamazaki, D., Huang, X., Torbick, N., Qi, J., Pattanakiat, S., Ngo-Duc, T., and Nguyen, T. D.: High resolution modeling of river-floodplain-reservoir inundation dynamics in the Mekong River Basin, Water Resour. Res., 56, e2019WR026449, https://doi.org/10.1029/2019WR026449, 2020.

Tayefi, V., Lane, S. N., Hardy, R. J., and Yu, D.: A comparison of one- and two-dimensional approaches to modelling flood inundation over complex upland floodplains, Hydrol. Process., 21, 3190–3202, https://doi.org/10.1002/hyp.6523, 2007.

Yamazaki, D.: Advancing global river hydrodynamics simulations by catchment-based macro-scale floodplain modeling approach, Geosci. Lett., 12, 72, https://doi.org/10.1186/s40562-025-00452-z, 2025.

Yamazaki, D., Kanae, S., Kim, H., and Oki, T.: A physically based description of floodplain inundation dynamics in a global river routing model, Water Resour. Res., 47, 2010WR009726, https://doi.org/10.1029/2010WR009726, 2011.

Yamazaki, D., Sato, T., Kanae, S., Hirabayashi, Y., and Bates, P. D.: Regional flood dynamics in a bifurcating mega delta simulated in a global river model, Geophys. Res. Lett., 41, 3127–3135, https://doi.org/10.1002/2014GL059744, 2014.

Yamazaki, D., Ikeshima, D., Sosa, J., Bates, P. D., Allen, G. H., and Pavelsky, T. M.: MERIT Hydro: A High-

Resolution Global Hydrography Map Based on Latest Topography Dataset, Water Resour. Res., 55, 5053–5073, https://doi.org/10.1029/2019WR024873, 2019.

Yamazaki, D., Revel, M., Hatono, M., Hanazaki, R., Nitta, T., Wortmann, M., Zhou, X., DirkEilander, Kang, S., Pilz, T., and Zhao, F.: global-hydrodynamics/CaMa-Flood_v4: Release_v4.23, Zenodo [code], https://doi.org/10.5281/zenodo.14214989, 2024.

Zahura, F. T., Goodall, J. L., Sadler, J. M., Shen, Y., Morsy, M. M., and Behl, M.: Training Machine Learning Surrogate Models From a High-Fidelity Physics-Based Model: Application for Real-Time Street-Scale Flood Prediction in an Urban Coastal Community, Water Resour. Res., 56, e2019WR027038, https://doi.org/10.1029/2019WR027038, 2020.

Zhao, F., Veldkamp, T. I. E., Frieler, K., Schewe, J., Ostberg, S., Willner, S., Schauberger, B., Gosling, S. N., Müller Schmied, H., Portmann, F. T., Leng, G., Huang, M., Liu, X., Tang, Q., Hanasaki, N., Biemans, H., Gerten, D., Satoh, Y., Pokhrel, Y., Stacke, T., Ciais, P., Chang, J., Ducharne, A., Guimberteau, M., Wada, Y., Kim, H., and Yamazaki, D.: The critical role of the routing scheme in simulating peak river discharge in global hydrological models, Environ. Res. Lett., 12, 075003, https://doi.org/10.1088/1748-9326/aa7250, 2017.