Reply to Reviewer 1's comments

Manuscript: CaMa-Flood-GPU (egusphere-2025-6500)

Legend

- Reviewers' comments
- Authors' responses
- Unchanged context from the manuscript
- New or changed text in the revised manuscript

This manuscript presents and evaluate a new implementation of the CaMa-Flood global river model. With this new implementation, the original model, written in Fortran and ran on a CPU multi-core architecture, is rewritten for a GPU-based architecture with adapted libraries and kernels with the objective to speedup global scale high resolution simulations without degrading performances. As a first step, the authors carefully analyze the main challenges behind this transposition, including the irregularity of the network topology, the interpolation of runoff inputs, the non linear relationship between water depth and river storage, and the handling of memory and communications between GPUs. Methods adapted to massive parallelism are proposed at each step. The new model, called CaMa-Flood-GPU, is then compared to the original CPU-based CaMa-Flood in terms of computation time and reproducibility. Results show a significant gain in computation time (more than 3 times quicker for a simulation at 1 arcmin resolution) with negligible differences in the outputs (river discharge and depth, flood outlow). The manuscript is well written and organized, figures are of good quality although some could be improved (see comments bellow). I have a few remarks that could further improve the manuscript, remarks that should be easily handled.

To reviewer: We thank the reviewer for the positive assessment of the manuscript and for the detailed comments. In response, we have made five main groups of revisions. First, we expanded the domain-decomposition description to explain how unit-catchments are grouped into basin groups formed by merging primitive river-mouth basins through cross-basin bifurcation links, and then assigned to GPU ranks without splitting such a group across GPUs. Second, we clarified that backwater coupling is local to each GPU rank and that the only runtime inter-GPU communications are the three collectives now named explicitly. Third, we revised the forcing, dataloader, block-size and memory-footprint descriptions so that the performance comparison is reproducible and the OOM entries in Table 3 are explained. Fourth, we redrew Fig. 5, revised Fig. 6, expanded the related captions, and changed Fig. 6 to display river-mouth locations as a primary map layer so the selected drainage units are visually explicit. Finally, we simplified the numerical-stability presentation by retaining the bifurcation-enabled comparison, adding relative differences, adding 0.1 ° ERA5-Land runoff over the unified 1980-2014 period, and broadening the discharge comparison across rivers of different scales. We appreciate that these comments helped us make the implementation choices and evaluation design much clearer. The specific manuscript changes and their locations are identified in the itemized responses below.

Comment 1. Ordering catchments and assigning them to dedicated GPUs is particularly important for efficient parallelism in terms of memory and communications, but it is not clear how this first

step is elaborated. More detailed could be provided, for example in section 2.1.1. This could also include how catchments are assigned to one GPU or another in a multi-GPU configuration (L172).

To reviewer: Thank you for raising this important point. We agree that the submitted manuscript did not explain the first decomposition step in enough detail. The implementation does not simply divide the global array into equal contiguous chunks. Instead, the preprocessing first traces each unit-catchment to its river mouth to obtain primitive river-mouth basins. It then uses the cross-basin bifurcation links to merge primitive basins that are hydrologically connected by bifurcating channels, producing larger basin groups. Only after this grouping step are the unit-catchments topologically ordered and packed into the global state vector.

This distinction is important for the multi-GPU design. Load balancing is applied to whole bifurcation-formed basin groups rather than to individual unit-catchments, using a longest-processing-time-first (LPT) greedy rule. As a result, a GPU subdomain is the union of several complete basin groups, and no such group is split across ranks. This preserves all main-channel and bifurcation connections inside a rank while still giving a practical load-balance strategy. The detailed paragraph below gives the grouping definition, the state-vector ordering and the LPT rank-assignment rule. The manuscript change is shown below for Section 2.1.4, replacing the current communication-overhead paragraph that only described domain decomposition in general terms.

Revised manuscript text:

A key challenge in scaling hydrodynamic models to multiple GPUs is managing communication and data handling overhead. For multi-GPU runs, we decompose the reordered river network by basin group rather than by individual unit-catchment. Let 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 $\mathbf{x} = [\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_K}]$, 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 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.

Comment 2. How are communications between neighbor catchments handled to account for backwater effects (impact of downstream water level on the surface profile and flow dynamics)? In other terms, are there some tricks with the arrangement of catchments into the memory to limit communication time (see also previous comment)?

To reviewer: Thank you. Backwater effects in CaMa-Flood-GPU enter through the local inertial update, where each unit-catchment uses the downstream water state along the river-network connection to compute the water-surface gradient and discharge. This is a network-neighbour dependency, but it is not handled through a peer-to-peer exchange at run time. The reason is the decomposition

described in Main Remark 1: all unit-catchments connected by retained main-channel and cross-basin bifurcation links remain inside the same bifurcation-formed basin group, and each such group is owned by a single GPU rank.

In implementation terms, the downstream index lookup, the bifurcation receiver lookup and the atomic accumulation path therefore operate on local state arrays after the preprocessing and rank assignment have been completed. The only data that cross GPU ranks during time stepping are the global collectives needed for forcing distribution, adaptive time-step synchronization and diagnostic reduction. We added the paragraph below to make this local-memory interpretation explicit before listing the collectives. The manuscript change is shown below for Section 2.1.4, immediately before the paragraph that enumerates the inter-GPU collectives.

Revised manuscript text:

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.

Comment 3. Can floods represented in 2D introduce water exchanges between neighbor catchments that are not directly connected through the river network? What would be the implications for memory exchanges?

To reviewer: Thank you. This is an important distinction between CaMa-Flood's catchment-based macro-scale floodplain (CMF) formulation and a genuine two-dimensional floodplain solver. The main CaMa-Flood approximation is the "uniform water surface" assumption within each unit-catchment: floodplain storage and inundated area are diagnosed locally from total storage and the precomputed sub-grid topographic profile. This local diagnosis changes water depth and flooded fraction inside the unit-catchment, but it does not by itself solve arbitrary lateral 2-D exchange between neighbouring floodplain areas.

At the same time, CaMa-Flood does include a bifurcation scheme to represent channel bifurcations and overbank-flow paths that are outside the main downstream river-network map. These pathways are treated as additional predefined routing links rather than as a 2-D floodplain stencil. In CaMa-Flood-GPU, they are therefore included in the same routing graph and basin-group decomposition as the main river links, so they do not introduce a separate halo-exchange pattern. If the model were extended to a true 2-D floodplain dynamic, with water exchanged across adjacent floodplain cells or catchment boundaries not represented by those links, then the GPU decomposition would need a stencil- or halo-style exchange between neighbouring ranks. The manuscript change is shown below for Section 2.1.3, appended to the paragraph that describes local floodplain-depth diagnosis from storage.

Revised manuscript text:

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. Because the inundation

diagnosis is based on the uniform-water-surface assumption within each unit-catchment, it is embarrassingly parallel on the GPU and 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.

Comment 4. Could you briefly describe what the `scatter_add` and `atomic_add` operations do?

To reviewer: Thank you for the suggestion. We have added a brief functional description that frames the two operations by what they achieve in the model rather than by their argument lists. The manuscript change is shown below for Section 2.1.1, at the end of the paragraph introducing the `scatter_add` and `atomic_add` operations.

Revised manuscript text:

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 explicit locks.

Comment 5. It is not clear how the global scale state array is constructed (see major comment 1).

To reviewer: Thank you. The global state array is built in two preprocessing stages. First, every unit-catchment is traced to its river mouth to form primitive river-mouth basins; cross-basin bifurcation links then merge primitive basins that exchange water through bifurcating channels into larger basin groups G_k. Second, the global state vector is laid out as $\mathbf{x} = [\mathbf{x}_{G_1}, \mathbf{x}_{G_2}, \ldots, \mathbf{x}_{G_K}]$, with basin groups sorted by decreasing size and entries within each G_k ordered upstream-to-downstream along the main river network. For multi-GPU runs, each rank's subdomain is the union of basin groups assigned to it by the longest-processing-time-first rule; because no group is split across ranks, every main-channel and bifurcation link used by the local inertial update connects unit-catchments inside the same rank, so state arrays remain contiguous in GPU memory. The full revised paragraph is given under Comment 1 above (Section 2.1.4, Communication and overhead) and is therefore not repeated here.

Comment 6. By "land grid cells", do you mean "grid cells from the Land Surface Model that produces runoff"?

To reviewer: Yes, thank you for asking. By "land grid cells" we meant the runoff-generating cells delivered by an upstream land-surface model. The manuscript change is shown below for Section 2.1.2, in the runoff-aggregation paragraph that defines the runoff vector.

Revised manuscript text:

Another challenge arising from the irregular network topology is the mismatch between gridded external forcing and our catchment units (Figure 1b). 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 the input grid 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 land-surface model. The resulting runoff vector is then passed to `shard_forcing`, which maps the runoff-generating cells to the local GPU catchments.

Comment 7. I guess the `shard_forcing` interface could easily integrate a specific method to couple CaMa-Flood with a Land Surface Model, right? It might be worth mentioning it.

To reviewer: Yes, thank you. This is exactly the design intent. We have added a sentence clarifying that `shard_forcing` is the hand-off point for online or offline coupling with a land-surface model. The manuscript change is shown below for Section 2.1.2, at the end of the paragraph introducing the `UserDefinedDataset` and `shard_forcing` interface.

Revised manuscript text:

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 a land-surface model (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. The core model remains unaware of these preprocessing details—as long as the dataset object supplies catchment runoff values R_c for each time step, the model will route them.

Comment 8. The figure is not clear and could be improved. For instance, what does the columns (3 in batched runoff, fluxes and errors) represent? Catchments? And the lines? Computation (sub-)time steps? Where is the synchronization between GPUs, does it allow to advance to the next time step? What are dataloader 0 and 1?

To reviewer: Thank you for pointing this out. We redrew Fig. 5 as a wall-clock timeline with separate rows for CPU loader processes and GPU ranks, and we revised the caption to define the tasks, collectives and adaptive sub-steps directly. The manuscript changes are shown below as the revised Fig. 5 caption and the following explanatory sentence in Section 2.1.4.

Revised manuscript text:

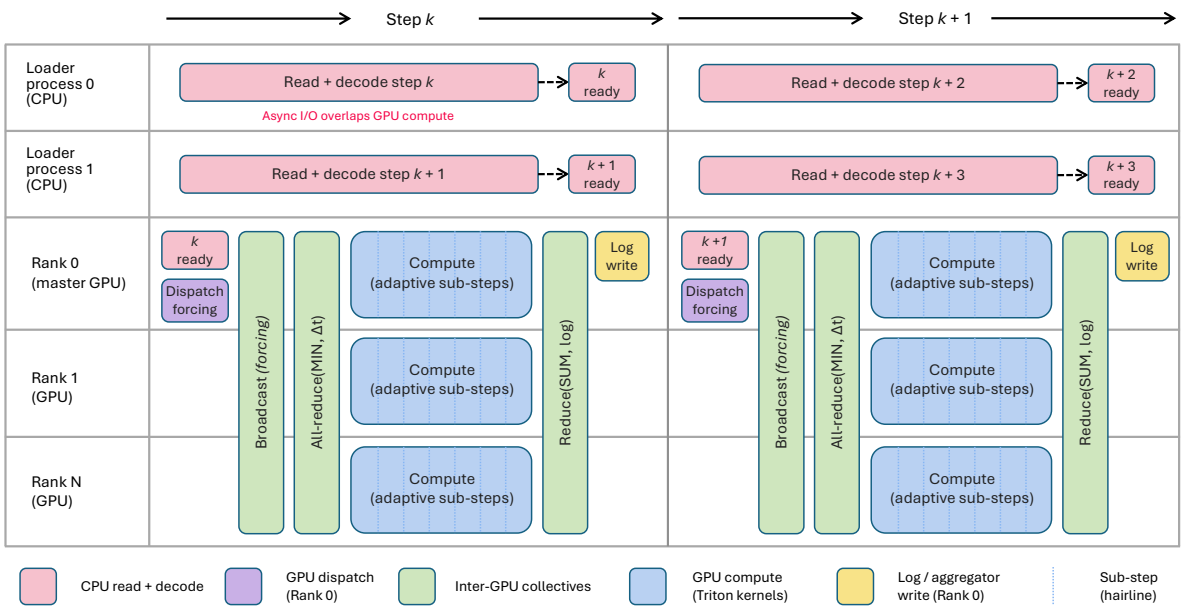

Figure 5. Runtime workflow for asynchronous input preparation, multi-GPU computation, and logging over two consecutive time steps.

To address these overheads, we implemented a suite of asynchronous and optimized data handling strategies (Figure 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.

Comment 9. Isn't the input broadcast also a collective communication? This would give three collective communications at each time step.

To reviewer: You are correct. The input broadcast is also a collective communication. We have updated the count from two to three and now name the three collectives explicitly in the revised text. The manuscript change is shown below for Section 2.1.4, replacing the sentence that previously counted only two collectives per time step.

Revised manuscript text:

As a result, each time step involves exactly two collective communications: one to determine the global minimum sub-step for adaptive time-stepping, and one to gather final log data. 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, a few tens of MB 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.

Comment 10. How is the flexible time step implemented/parallelized? I understand that the same sub-step is chosen for all the catchments of the globe, is that right? Since each GPU works asynchronously, could it be possible to choose a different sub-step for each GPU?

To reviewer: Thank you for the question. We use one globally synchronized sub-step so that the GPU trajectory remains comparable to the CPU baseline regardless of GPU partitioning. The revised manuscript text below adds this point to the adaptive time-step discussion. The manuscript change is shown below for Section 2.1.4, in the paragraph explaining adaptive time-step synchronization.

Revised manuscript text:

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.

Comment 11. The figure could be improved and enlarged: gauge dots are not clearly visible except with a very high zoom, star symbols are not visible at all. Also, in the figure caption, it is written that the catchment outlines are shown; I understand that they are represented by the shaded colors, but in the text, the term catchment corresponds to base unit while in the figure it is more likely the entire basin. Is that right? Finally, why some catchments/basins are so large, encompassing several basins (like orange in South America, pink in Asia, green in Africa or brown in North America)?

To reviewer: Thank you for these comments. We agree that the previous Fig. 6 description could be misread. In the revision, we changed the figure concept to directly show retained-basin river-mouth locations together with cross-basin bifurcation links. This makes the selected drainage units and their correspondence to the point-of-interest filtering step easier to interpret at normal zoom. We also clarify that GPU-rank assignment is described separately in the domain-decomposition section. The revised Fig. 6 is shown below.

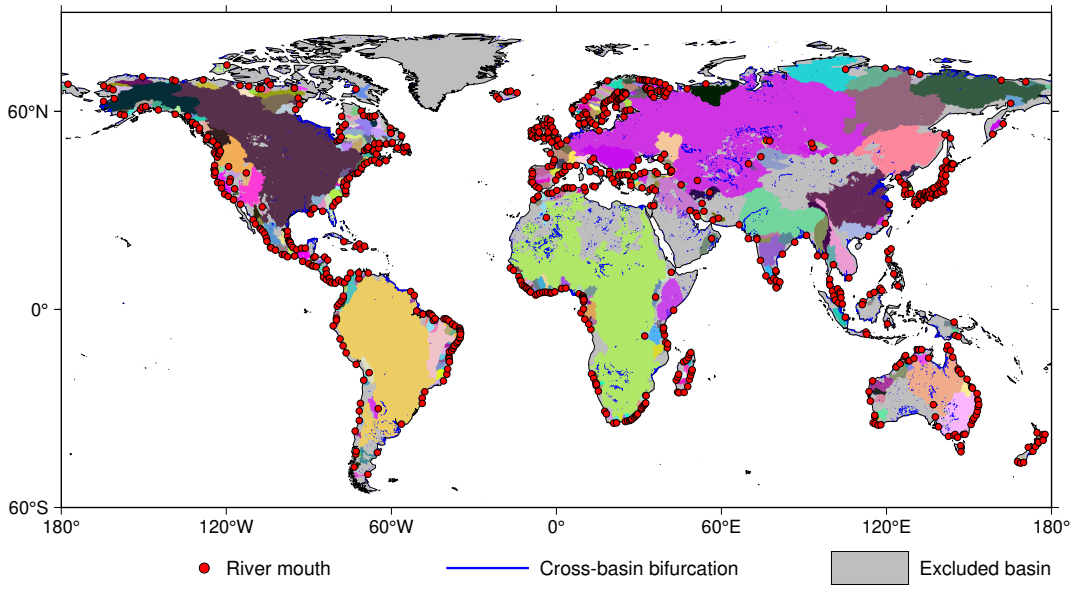

Figure 6. Example of a customized simulation domain constructed from bifurcation-formed basins after the point-of-interest filtering and domain-decomposition step. Each retained basin contains at least one point of interest. Coloured patches denote the retained basins and their multi-GPU LPT groups, while grey areas mark basins excluded from the selected domain; markers indicate river-mouth locations and blue lines denote cross-basin bifurcation links.

Comment 12. It seems from Table 3 that the CPU configuration was not used for the first three machines (4070 Ti, V100 and A100). Why then fill in the CPU and CPU Cores columns for these machines? Also, would it be possible to add the available memory for each machine and node?

To reviewer: Thank you for this useful suggestion. The CPU and CPU-core columns are retained for the GPU machines because these CPUs are the host processors for the GPU runs: they launch the GPU program, handle CPU-side preprocessing and I/O, coordinate the data loader, and provide host memory around GPU execution. The CPU-only benchmark uses the separate CPU row, but the host CPU configuration is still needed to reproduce the GPU cases in Table 3. We therefore kept the CPU information for all machines and added CPU memory per node to Table 1.

Revised manuscript text:

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 (12GB)	16	1
V100	Dual Intel Xeon E5-2640 v4	128 GB	NVIDIA Tesla V100 (16GB)	20	4
A100	Dual AMD EPYC 7543	256 GB	NVIDIA A100 (40GB)	64	4
CPU	Dual Intel Xeon 6248R	256 GB	—	48	—

Comment 13. I understand the idea of using a coarse resolution forcing (1° runoff) to focus more on computation performances. But running a global scale simulation at very high resolution (e.g. 1 arcmin, that is typically not achievable with the current CPU version) would also require high resolution forcing. Maybe an additional experiment would help quantify the added simulation time due to the reading and broadcasting of high resolution forcing.

To reviewer: Thank you for raising this. We did not run a forcing-resolution sweep for the speed benchmark, and we would like to explain why we believe the current setup already isolates the comparison properly. CaMa-Flood-GPU uses a multi-process asynchronous dataloader that reads and decodes the next time step's forcing while the GPU is integrating the current step. In our GPU tests, using NetCDF forcing instead of CaMa-Flood's native binary forcing produced no measurable change in wall-clock time, because the file-format cost was hidden behind the GPU computation. The CPU version, however, reads forcing synchronously, so NetCDF decoding competes with the routing calculation for CPU resources and increases the CPU wall time. The deliberate choice of the 1 ° binary sample forcing distributed with CaMa-Flood therefore keeps the wall-time comparison in Table 3 focused on routing-side performance rather than file-format overhead.

At the same time, we agree that the revised manuscript should demonstrate that CaMa-Flood-GPU can ingest and route higher-resolution gridded forcing. We therefore added a separate numerical-stability experiment based on a daily forcing series prepared from 0.1 ° ERA5-Land runoff over 1980-2014. This experiment is not used to claim a speed impact of high-resolution forcing; instead, it verifies the high-resolution NetCDF input path, forcing-to-unit-catchment mapping and routing kernels under identical CPU and GPU parameter states. We mention this distinction in the revised setup text so the reader can separate the routing-focused speed benchmark from the high-resolution-forcing stability test.

A second consideration is where the resolution mismatch is actually paid for. Interpolating high-resolution runoff onto CaMa-Flood-GPU's irregular unit-catchment network is a preprocessing step rather than a per-time-step cost, and Python's geospatial library ecosystem makes implementing such regridders straightforward. In CaMa-Flood-GPU, the forcing wrapper classes offer optional pre-interpolation and in-memory caching. When this option is enabled, regridding is performed once and reused across runs, whether the source is a structured high-resolution grid or another unstructured mesh. We recommend this opt-in pattern for high-throughput use cases such as parameter calibration and sensitivity analysis, where hundreds to thousands of replicate runs may share the same forcing. As a result, the wall-time impact of high-resolution forcing on the simulation itself is bounded by the asynchronous dataloader pipeline rather than by the regridding cost. The revised Figure 5 further illustrates this loading mechanism, and the expanded numerical-stability experiment exercises this path with long-period high-resolution forcing.

Comment 14. Could you explain what the block size is? Is this related to the catchment assignment (see major remark 1)?

To reviewer: Thank you for raising this. We now define the Triton block size as a kernel granularity and distinguish it from the basin-group assignment described above under Main Remark 1. The manuscript change is shown below for the new experimental-setup paragraph at the start of Section 3.1.

Revised manuscript text:

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: adaptive sub-step integration, bifurcation routing and on-the-fly logging of the selected diagnostic output. The hardware configurations used in the benchmarks are summarized in Table 1.

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) and is included in the CPU release. The CPU reference is CaMa-Flood v4.23 compiled with Intel Fortran using "-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. Table 3 summarizes the benchmark timing results for a 1-year simulation period at the four resolutions on various hardware configurations: a personal workstation with NVIDIA RTX 4070 Ti, the V100 server with 1–4 GPUs, the A100 server with 1–4 GPUs, and the CPU-only server (four CPU nodes) with different numbers of CPU cores.

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.

Comment 15. The amount of memory is also a very important aspect in global scale and high resolution simulations. Could you explain why some configurations encountered lack of memory problems and not others?

To reviewer: Thank you for raising this. We added a short explanation of the 1-arcmin GPU memory footprint and how it relates to the OOM entries in Table 3. The manuscript change is shown below for Section 3.1, appended to the paragraph discussing the 1-arcmin performance results.

Revised manuscript text:

The gap widens at finer resolutions: at 3-arcmin, the CPU assumes 2 h 45 m 19 s, compared to 11 m 47 s on 4×V100 and 7 m 21 s on 4×A100; at 1-arcmin, the CPU needs 140 h 58 m 20 s, versus 6 h 51 m 36 s on 4×V100 and 3 h 51 m 24 s on 4×A100. At the 1-arcmin resolution, the global state requires about 20 GB of GPU memory in total. This footprint is partitioned across ranks by the LPT assignment of bifurcation-formed basin groups, so the per-GPU peak roughly halves with each doubling of the rank count. Configurations whose per-card VRAM in Table 1 is below the resulting per-GPU peak are marked OOM in Table 3.

Comment 16. In the third column, it could be preferable to show the relative difference. In that case, values bellow 10^{-6} could be attributed to numerical errors only (floating-point precision).

To reviewer: Thank you for the suggestion. We accepted the suggestion and redrew the retained numerical-stability figure so the third column reports relative differences. We note, however, that the floating-point noise floor in this comparison is somewhat above 10^{-6}: each CaMa-Flood time step is composed of many adaptive sub-steps, and within each sub-step the routing kernel evaluates several physical processes (flow inertia, friction, floodplain exchange and, when enabled, bifurcation accumulation) in different orders on CPU and GPU. The accumulated round-off across sub-steps and across these operations therefore exceeds the per-operation floating-point precision and is reflected in the relative-difference panel.

Comment 17. What is the period of the simulation?

To reviewer: Thank you for the question. The benchmark period is calendar year 2000 (365 days, 2000-01-01 to 2000-12-31). Year 2000 was selected because it is included in the sample runoff forcing distributed with the CPU CaMa-Flood release, which makes the benchmark straightforward to reproduce. This is already stated in the new experimental-setup paragraph in Section 3.1 shown above.

Comment 18. What is the added value of showing both simulations, with and without the activation of the bifurcation module?

To reviewer: Thank you for the question. We took this comment as an opportunity to substantially expand the numerical-stability section rather than to keep two parallel figures. In the revision, we retain the bifurcation-enabled comparison as the primary case because it includes the base routing operations and additionally activates the bifurcation-specific atomic accumulation path, so CPU-GPU agreement in this configuration also validates the simpler no-bifurcation path. We then expand the retained analysis by adding 0.1 ° ERA5-Land runoff and eartH2Observe Tier-1 ensemble runoff comparisons over the unified 1980-2014 period, and by broadening the station-level discharge comparison to rivers of different scales to test whether CPU-GPU agreement holds beyond the largest channels. The manuscript change is shown below for Section 3.2.

Revised manuscript text:

The numerical-stability comparison follows the setup described in Section 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 8 presents the CPU–GPU comparison under E2O runoff. The CPU and GPU mean river-outflow fields (Figure 8a, b) are visually indistinguishable, and the relative-difference map in Figure 8c is dominated by the gray "below-noise-floor" band of ±0.001%, with non-zero values appearing mainly on the largest river stems. Figure 8d–f repeats the comparison for floodplain outflow and Figure 8g–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 9 reproduces the same comparison under ERA5-Land runoff. The residual magnitude and spatial pattern remain essentially unchanged between Figures 8 and 9, although the two forcing 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 10 and 11 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 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. Figure 11 reproduces the same behaviour under ERA5-Land runoff, and 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. The bifurcation-enabled configuration exercises both the base routing operations and the bifurcation-specific `atomic_add` path. The no-bifurcation configuration is a strict subset of these operations, so agreement with bifurcation enabled also supports agreement for the simpler routing case.

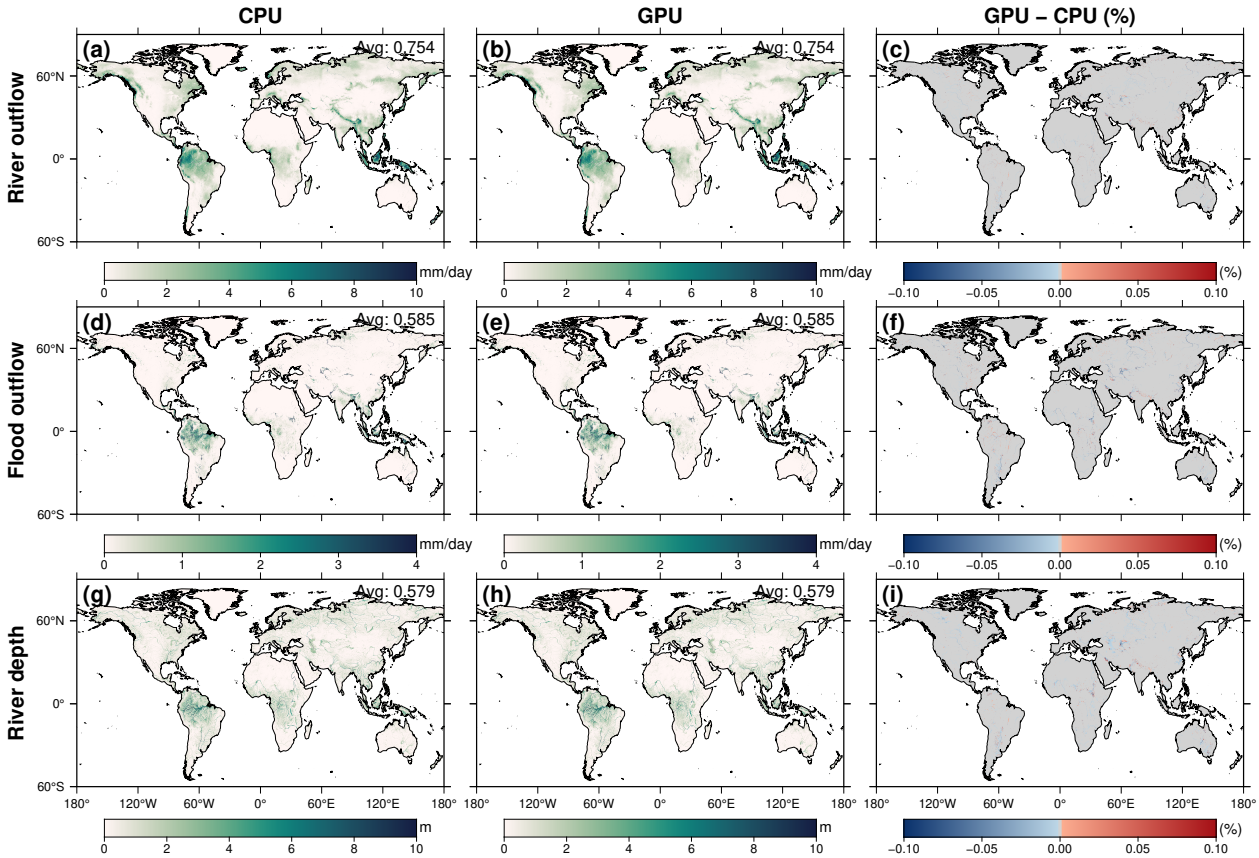

Figure 8. CPU vs. GPU mean fields with the bifurcation module enabled, under E2O Tier-1 ensemble-mean runoff at 0.25 ° for 1980–2014. **(a–c)** Mean river outflow for CPU, GPU and their relative difference. **(d–f)** Mean floodplain outflow for CPU, GPU and their relative difference. **(g–i)** Mean river 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.

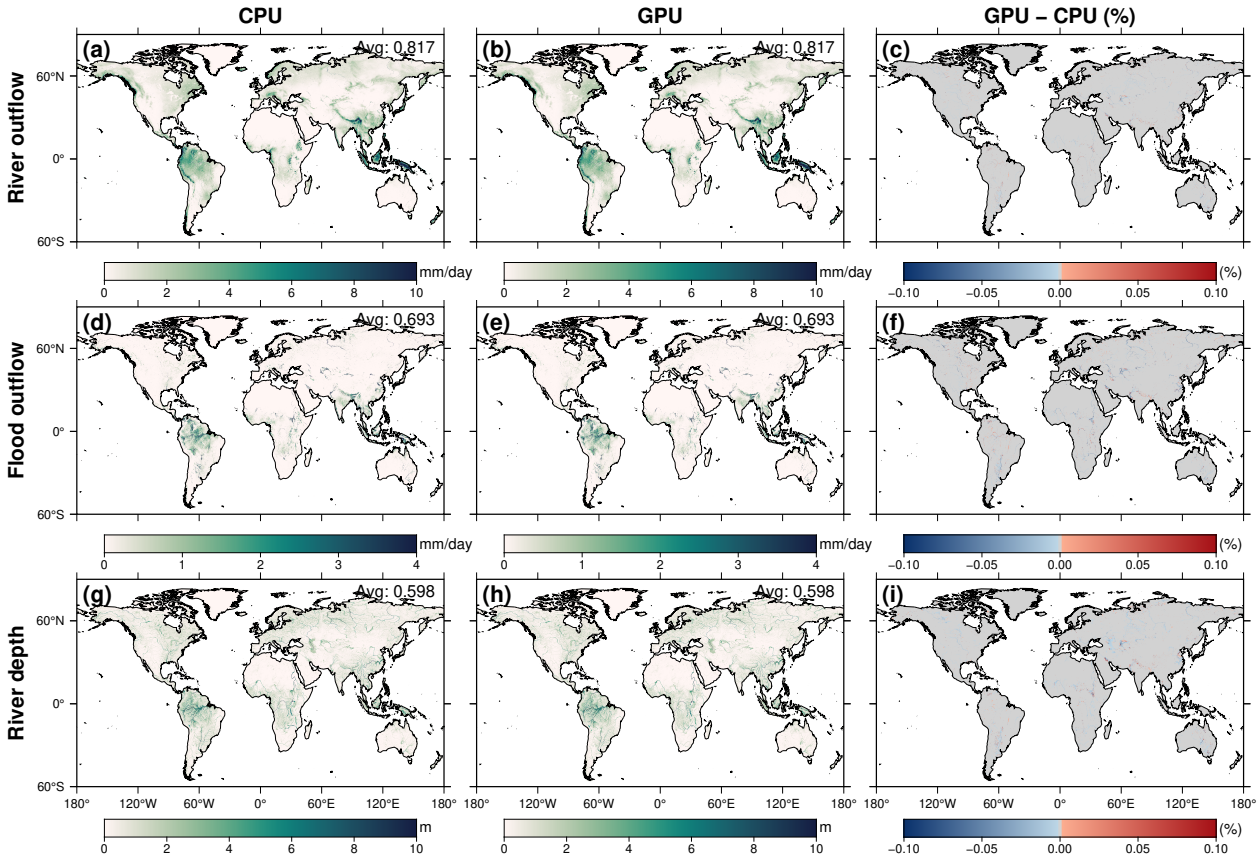

Figure 9. Same as Figure 8, but with ERA5-Land surface runoff at 0.1 °.

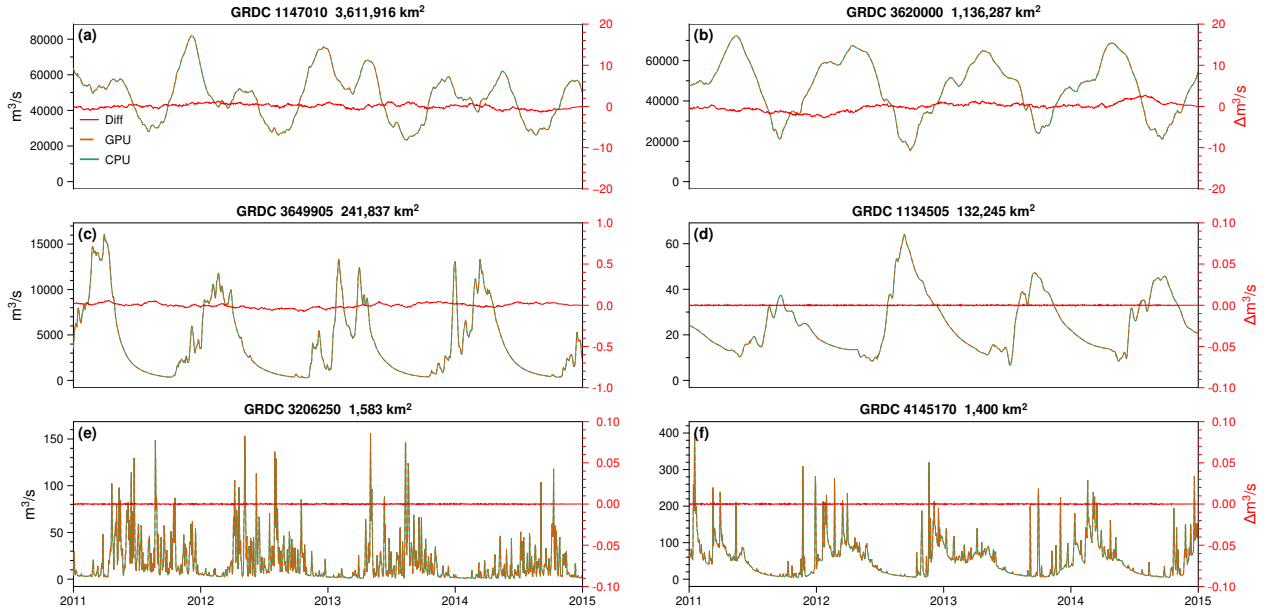

Figure 10. Simulated daily river discharge at the locations of six GRDC gauges spanning four orders of magnitude in drainage area, for the last four years of the 1980–2014 simulation under E2O Tier-1 runoff at 0.25 °. The gauge panels are ordered by drainage area from large to small. CPU (green solid) and GPU (orange dashed) curves overlap; the difference is plotted as a red solid line with values given on the right-hand axis.

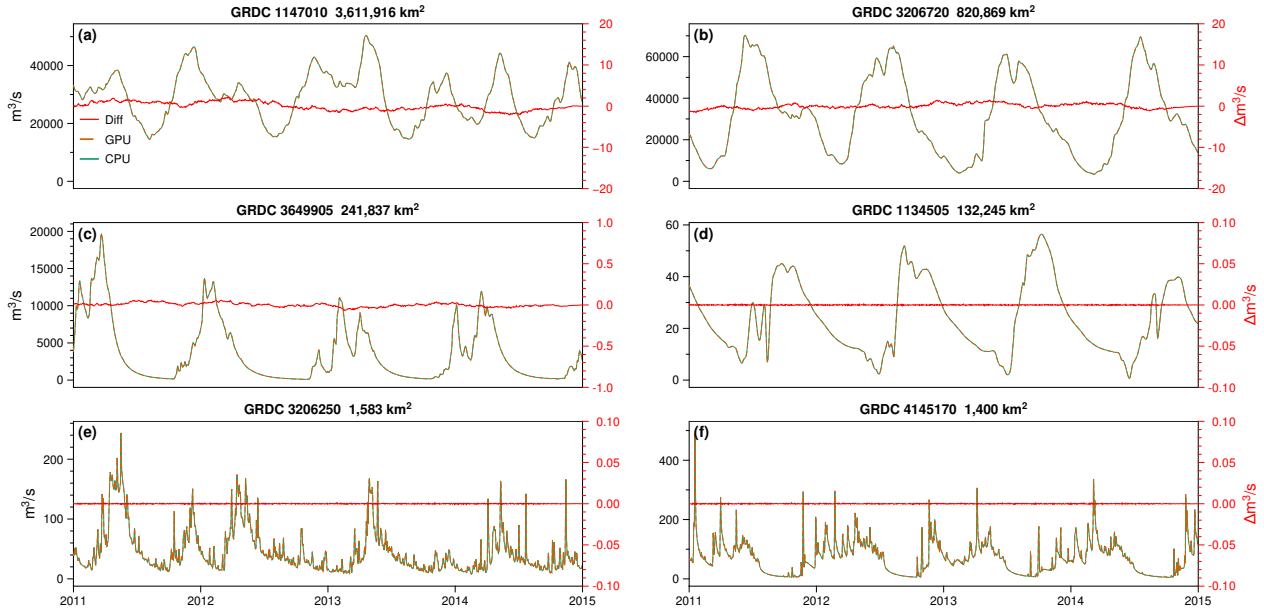

Figure 11. Same as Figure 10, but with ERA5-Land surface runoff at 0.1 °.

References

- Muñoz-Sabater, J., Dutra, E., Agusti-Panareda, A., Albergel, C., Arduini, G., et al. (2021). ERA5-Land: A state-of-the-art global reanalysis dataset for land applications. Earth System

Science Data, 13, 4349-4383. https://doi.org/10.5194/essd-13-4349-2021

- Kim, H., Yeh, P. J.-F., Oki, T., and Kanae, S. (2009). Role of rivers in the seasonal variations of terrestrial water storage over global basins. Geophysical Research Letters, 36, L17402. https://doi.org/10.1029/2009GL039006
- 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. (2017). A global water resources ensemble of hydrological models: the eartH2Observe Tier-1 dataset. Earth System Science Data, 9, 389-413. https://doi.org/10.5194/essd-9-389-2017
- Yamazaki, D., Ikeshima, D., Sosa, J., Bates, P. D., Allen, G. H., and Pavelsky, T. M. (2019). MERIT Hydro: A high-resolution global hydrography map based on latest topography dataset. Water Resources Research, 55, 5053-5073. https://doi.org/10.1029/2019WR024873