https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

WarpFlow v1.0: A Single-Source CPU/GPU Shallow-Water Solver for Operational City-Scale Urban Flood Modeling

Lars Backhaus[1] and Jürgen Stamm[1]

[1]Institute of Hydraulic Engineering and Technical Hydromechanics, Faculty of Civil Engineering, TUD Dresden University of Technology, Dresden, Germany

Correspondence: Lars Backhaus (lars.backhaus@tu-dresden.de)

Abstract. Operational urban flood management increasingly needs two-dimensional hydrodynamic models that re-run on demand, are driven by data pipelines without hand editing, and run on hardware a municipality already owns. We present WarpFlow, an open-source solver for the two-dimensional shallow-water equations built against these three constraints. WarpFlow solves the full shallow-water equations on a staggered Arakawa C-grid, retaining advective momentum. The scheme combines an Audusse well-balanced hydrostatic reconstruction, a first-order upwind mass flux, and a semi-implicit momentum update with implicit bed friction and advective damping. The velocity update is evaluated in Jacobi form, so it is independent of traversal order. Together with a single-source, three-backend implementation, this yields CPU and GPU results that agree to single-precision round-off. A positivity floor guarantees non-negative depth, and an optional flux-corrected-transport limiter is available where required at wetting fronts. Rainfall, infiltration, drainage sinks, stage and discharge boundaries, and weir/orifice culverts with lumped transport delay are treated as first-class source terms. WarpFlow is a Python package whose kernels compile to multi-core CPU code and to NVIDIA CUDA through Numba and CuPy. The CPU, dense CUDA and memory-sparse tiled CUDA backends share one kernel formulation. Static fields are compressed by lookup-table encoding, and the tiled backend allocates memory only for terrain-bearing tiles, so city-scale domains fit in consumer-GPU memory. Two automatic runtime controls reduce work: active-tile scheduling skips dry terrain tiles, and an adaptive CFL-stride controller amortizes global time-step reductions when the characteristic speed is stable. A hook-based plugin layer and scripting API expose every stage of a run, allowing WarpFlow to operate inside automated modeling pipelines. We verify WarpFlow against analytical dam-break and lake-at-rest solutions, and demonstrate water-budget closure, grid convergence and CPU/GPU reproducibility. We validate it against the Environment Agency 2D benchmark subset and the Malpasset dam break. As an operational showcase, we simulate a 100-year, one-hour Euler-II design storm over the administrative area of Dresden, Germany, across a two-hour window. At $5\,\mathrm{m}$ resolution ($\approx 365\,\mathrm{km}^2$ terrain area), the run completes in $51.68\,\mathrm{s}$. At $2\,\mathrm{m}$ resolution ($\approx 1.66 \times 10^8$ total cells, $\approx 9.1 \times 10^7$ terrain cells), it completes in $\approx 14.4\,\mathrm{min}$ on one consumer GPU, compared with $\approx 15.4\,\mathrm{h}$ for the same solver's 24-thread CPU backend ($\approx 64.3\times$ speedup). WarpFlow is the engine behind the operational Urban Digital Flood Twin Dresden; here we document the solver. The code is released under the AGPLv3 license.

Keywords. Urban hydrology; Shallow water equations; Numerical modeling; Graphic processing unit (GPU); Finite volume methods; High-performance computing

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

1 Introduction

1.1 Motivation

Pluvial and fluvial flooding of urban areas is intensifying under climate change and continued urbanization (Kundzewicz et al., 2014; Guo et al., 2021). Three distinct pressures shape the tooling this demands. The first is a climate pressure. Precipitation extremes are becoming more intense and more frequent, so cities must assess hazard and risk both quickly and credibly. The second is a usability pressure. Assessment tools should run in minutes, be configurable without specialist training, and expose modern programmatic interfaces. A planner can then answer a "what-if" question within a working session rather than through a multi-month consultancy tender, which returns authority over a piece of critical infrastructure to the municipality that operates it. The third is a regulatory and maintenance pressure. The EU Floods Directive mandates a six-year update cycle for flood hazard maps (European Parliament and Council, 2007). River flooding is routinely mapped inside formal risk plans, but pluvial flooding remains widely under-assessed. Germany's nationwide $1\,\mathrm{m}$ pluvial hazard map is scheduled for completion by the end of 2026 (Bundesamt für Kartographie und Geodäsie, 2026), yet it carries no established procedure for keeping it current, so it will already be at least two years out of date at release. Cities, however, change continually: surfaces are resealed, buildings are added or removed, drainage assets are modified, and temporary works alter local flow paths. Recent reviews of urban-water and flood-risk digital-twin applications identify the same operational bottleneck from another direction: many systems remain conceptual, prototype-based, or closer to static digital models and one-way digital shadows than to routinely rerunnable operational tools, with practical deployment limited by data interoperability, computational cost and specialist workflows (Altındaş et al., 2026; Hlal et al., 2025; Lo et al., 2026). System-level flood applications such as FlowsDT illustrate the resulting expectation that hydraulic simulations feed geospatial warning, routing and decision-support workflows rather than remain stand-alone map products (Mandal et al., 2026).

These pressures translate into three concrete requirements. (i) City-scale runs must complete in minutes, so that assessments stay current and interactive queries remain answerable; (ii) the solver must be driven programmatically, as one stage of an automated pipeline, rather than configured through hand-edited text files, so that models can be regenerated ad hoc from continuously changing municipal data; and (iii) it must run on the commodity hardware a city IT department can procure and maintain, not on an HPC cluster.

WarpFlow serves three use cases of the Smart City Dresden Urban Digital Twin for floods: (a) city-wide, high-grade hazard and risk assessment; (b) a hydrodynamic flash-flood nowcast with sub-minute turnaround, including operational disaster support, for example evaluating where to place temporary flood barriers and to what effect; and (c) high-fidelity district planning at $0.25\,\mathrm{m}$–$1\,\mathrm{m}$ in human-friendly compute times. Together these demand a solver that is efficient, memory-frugal and, above all, predictable.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

1.2 Numerical modeling of urban surface flow

Most operational urban inundation models solve the shallow-water equations (SWE), or a reduced form of them. The SWE are the depth-averaged Navier–Stokes equations under the hydrostatic-pressure assumption (Lukáčová-Medvid'ová and Teschke, 2006), and they model pluvial and fluvial flood routing well where vertical accelerations are not the dominant process.

Two solver families dominate raster-based flood modeling. The first solves the full SWE with finite-volume, discontinuous-Galerkin or related conservative schemes. Godunov-type finite-volume methods use approximate Riemann solvers at cell faces to resolve shocks, hydraulic jumps and transcritical flow (Toro, 2001; LeVeque, 2002), and recent GPU models such as SERGHEI-SWE and TRITON show that full-SWE raster solvers can run on very large domains (Caviedes-Voullième et al., 2023; Morales-Hernández et al., 2021).

The second family simplifies the momentum equation. The local-inertial formulation of Bates et al. (2010) neglects convective acceleration and underpins LISFLOOD-FP ACCeleration (ACC) (De Almeida and Bates, 2013). RIM2D follows the same local-inertial approximation for rapid urban inundation modeling, but couples it to an explicitly urban raster workflow: buildings are treated as flow obstacles, and infiltration, sewer drainage and roof drainage are represented through capacity-based source/sink terms rather than through an explicit coupled surface–sewer network (Apel et al., 2024). This class is fast and well suited to many operational flood-mapping applications, especially where short runtimes are required for probabilistic risk assessment or forecasting, but its range of validity depends on the neglected momentum terms and on the flow regime. The LISFLOOD-FP 8.0 study states this boundary clearly for ACC: ACC is recommended for gradually varying, subcritical fluvial, pluvial and coastal flooding over sufficiently rough surfaces, whereas full-SWE solvers are needed for dam breaks, steep-catchment flash floods, rapidly varying supercritical flow, shock waves and very smooth surfaces (Shaw et al., 2021). LISFLOOD-FP has therefore added full-SWE finite-volume and discontinuous-Galerkin solvers, including GPU and adaptive-grid variants (Shaw et al., 2021; Sharifian et al., 2023; Chowdhury and Kesserwani, 2025; Chowdhury et al., 2023).

Table 1 positions WarpFlow against the solver families and operational models that define its design space, rather than cataloging the field. Broader overview tables already exist for SWE and GPU flood solvers (Caviedes-Voullième et al., 2023; Fuenzalida Alarcón et al., 2026). Here the table makes one engineering point: WarpFlow targets a specific combination of full-SWE physics, raster data flow, Python control and single-workstation GPU execution.

Grid choice is itself part of the numerical design. Unstructured triangular meshes fit breaklines, channels and infrastructure edges accurately (Schubert et al., 2008), but they require a meshing step that is hard to automate when the input data change often. WarpFlow uses a structured Cartesian raster, because municipal data usually start as rasters or GIS layers. A raster grid has a fixed topology and is generated directly from a DEM, so buildings, roughness classes, infiltration fields, drainage features and administrative boundaries all project onto the same cell model and stay inspectable in common GIS tools. This choice has a cost: it resolves curved and narrow geometry only to the grid scale, and it can make urban micro-topography sensitive to resolution. We accept that cost because the target workflow is automated model regeneration from city data, not hand-built meshing.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 1. Design-space comparison of selected 2D inundation solvers relevant to WarpFlow. The table is not a solver catalog. It highlights the model families and operational tools that define WarpFlow's engineering target. Row citations are given in the table note.

Solver / family	Numerical formulation	Grid and acceleration model	Relation to WarpFlow
LISFLOOD-FP ACC / 8.x	ACC is local-inertial and neglects convective acceleration. LISFLOOD-FP 8.x also includes full-SWE FV and DG solvers.	Structured raster. (NU_)ACC is optimized for GPU execution. The FV/DG solvers include GPU variants.	Closest raster flood-model lineage. WarpFlow keeps the raster workflow, but retains advective momentum and exposes the solver through a Python API.
RIM2D	Local-inertial approximation of the SWE. Convective acceleration is neglected. Urban drainage is represented by capacity-based infiltration, sewer-drainage and roof-drainage terms, with buildings treated as flow obstacles.	Structured raster. CUDA Fortran implementation for NVIDIA GPUs. Input data are standard urban rasters, including DEM, building footprints and land-use or sealed-surface maps.	Closest operational urban comparator: rapid pluvial inundation simulation for risk assessment and forecasting. WarpFlow shares the raster and GPU target, but retains full-SWE advective momentum and emphasizes Python-first pipeline control, backend parity and single-consumer-GPU memory limits.
SERGHEI-SWE	Full SWE with a first-order upwind finite-volume formulation. The scheme is well balanced and positivity preserving.	Structured Cartesian grid. C++/Kokkos implementation with MPI and CPU/GPU performance portability.	Shows that first-order Cartesian full-SWE solvers are practical at high spatial resolution. WarpFlow targets single-workstation municipal runs.
TRITON	Full two-dimensional hydrodynamic flood model with GPU-oriented finite-volume numerics.	Structured Cartesian/raster grid. CUDA and multi-GPU execution.	Represents the high-performance raster full-SWE class. WarpFlow emphasizes API integration, backend parity and memory use on one consumer GPU.
SWEpy	Full SWE finite-volume solver with a well-balanced, positivity-preserving, higher-order central-upwind/WENO-type formulation.	Unstructured triangular mesh. Python/CuPy implementation for CUDA GPUs.	Shows the value of Python/GPU SWE software. WarpFlow makes the opposite grid choice: structured rasters for automated municipal data pipelines.
WarpFlow	Full SWE in advective velocity form with hydrostatic reconstruction, upwind mass flux, semi-implicit face momentum and an optional FCT limiter.	Staggered Cartesian raster. Single-source CPU, dense CUDA and terrain-sparse tiled CUDA backends.	Targets operational city-scale flood twins: API-first execution, plugins, backend reproducibility and memory use sized for one consumer GPU.

Row references: LISFLOOD-FP ACC / 8.x: Bates et al. (2010), De Almeida and Bates (2013), Shaw et al. (2021); RIM2D: Apel et al. (2024); SERGHEI-SWE: Caviedes-Voullième et al. (2023); TRITON: Morales-Hernández et al. (2021); SWEpy: Fuenzalida Alarcón et al. (2026).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

The design space that motivated WarpFlow is narrow but underserved: a full-SWE, structured-grid solver that is API-first, runs city-scale domains on a single consumer GPU, and ships as a single Python package rather than a monolithic executable. Existing solvers cover parts of this space. ACC-type and other local-inertial models are fast and operationally mature, and RIM2D shows how this class can be specialized for rapid urban pluvial risk assessment and forecasting with parsimonious drainage representations (Apel et al., 2024). HPC full-SWE raster solvers target large-scale performance and multi-device scaling. Python/GPU solvers improve accessibility, but often use unstructured meshes or higher-order schemes aimed at different wave regimes. WarpFlow closes the specific gap needed by an automated municipal flood twin: full-SWE raster physics, API-first control and single-workstation GPU execution.

This choice is consistent with recent high-performance SWE design. SERGHEI-SWE treats first-order finite-volume schemes as a practical option at very high resolution, where resolution partly offsets the diffusion of low-order schemes, and it also adopts Cartesian grids to simplify GPU memory access, cache use and domain decomposition (Caviedes-Voullième et al., 2023). The same tradeoff has been quantified for urban flash-flood modeling by Buttinger-Kreuzhuber et al. (2022), whose GPU-accelerated full-SWE framework on a structured Cartesian grid found that a first-order scheme on a finer grid is more efficient than a second-order scheme for the same computational budget, and which runs city-scale rainfall–runoff domains of order $220\,\mathrm{km}^2$ at metre resolution on a single GPU. WarpFlow follows the same engineering logic, but constrains the hardware target more tightly.

WarpFlow began as an experiment with the local-inertial GPU formulation of LISFLOOD-FP ACC (Bates et al., 2010; De Almeida and Bates, 2013; Sharifian et al., 2023), but the operational requirements pushed the design away from that starting point. In its present form WarpFlow solves the full SWE in velocity form, retains upwinded advective momentum transport, and uses a well-balanced hydrostatic reconstruction. It enforces non-negative depth through a positivity floor on the mass update, with an optional flux-corrected-transport limiter where exact conservation across draining fronts is required. It also replaces the file-driven interface with a scripting API and a plugin system built for GIS pipelines and consumer-GPU memory limits.

1.3 Contributions

1. A **full 2D SWE scheme on a staggered C-grid** (Sect. 2): Audusse well-balanced reconstruction, upwind donor-cell mass flux, a semi-implicit friction/advective-damping momentum update in Jacobi form, positivity by construction, and a CFL criterion formulated for the unsplit update. The design keeps well-balancing and checkerboard suppression as two independently justified choices. The individual components are established; the contribution is their combination into a single robust scheme.

2. A **single-source, three-backend implementation** (Sect. 3): CPU, dense CUDA and memory-sparse tiled CUDA from one kernel formulation. The order-independent velocity update yields CPU/GPU agreement at single-precision round-off, and memory optimizations, including LUT compression, flat bit-addressed tiled storage and terrain-sparse allocation,

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

keep city-scale domains within a consumer GPU's memory. Every kernel and operation runs in parallel on all three backends.

3. An **API-driven, plugin-extensible** solver (Sect. 4), released under AGPLv3. A hook-based layer with a runtime kernel override mechanism lets new diagnostics, source terms and physics be added on all three backends without touching the flux kernels. We demonstrate it with four bundled plugins: hydraulic diagnostics, dynamic infiltration capacity, an online water-balance monitor, and a lightweight performance profiler used to instrument the benchmarks of Sect. 6.

4. A **verification, reproducibility and performance study** (Sect. 5–6), including mass balance, grid convergence, an explicit accounting of the cost of the solver's adaptivity mechanisms, and an operational city-scale demonstration for Dresden.

The remainder of the paper is organized as follows. Section 2 presents the governing equations and their discretization; Sect. 3 the software architecture and optimizations; Sect. 4 the plugin architecture; Sect. 5 verification and validation; and Sect. 6 the performance and scalability study. Sections 7 and 8 discuss and conclude.

2 Governing equations and numerical formulation

We state the scheme once. It applies to all three backends, and the tiled CUDA form is the canonical reference. Throughout, h is water depth, u, v are the depth-averaged face velocities, z is bed elevation, and $\eta = h + z$ is the water-surface elevation. In the continuous equations $q_x = uh$ and $q_y = vh$ are the unit discharges (m^2/s). The discrete face-integrated volumetric discharges used in the update carry the transverse cell width and are denoted by $Q_x = u\,h_f\,\Delta y$ and $Q_y = v\,h_f\,\Delta x$ (m^3/s, Sect. 2.3). Constants are single-precision (Sect. 3.2), and $g = 9.806\,65\,\mathrm{m/s}^2$. The two cells adjacent to an x-face are L (west) and R (east), and those adjacent to a y-face are B (south) and T (north). A cell's four faces are labeled by compass point: W, E on the x-axis and S, N on the y-axis (Fig. 1).

2.1 Shallow-water equations and conserved variables

Mass is conserved in depth form, written as a residual balance

$$\frac{\partial h}{\partial t} + \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} - S_h = 0, \tag{1}$$

where the volumetric source term S_h collects the physically distinct additions and removals of water: rainfall R, infiltration I, drainage sinks, and any interior discharge or prescribed-water-level forcing (Sect. 2.9). When no such forcing is active, $S_h = 0$ and Eq. (1) reduces to the homogeneous transport of depth. The initial water depth is an initial condition, not a source. Likewise, boundary discharge and water-level data enter through the boundary and source handling (Sect. 2.8, Sect. 2.9) rather than through the base equation. In the solver core the active contributions are rainfall minus infiltration, $S_h = R - I$ (Sect. 2.9); a dynamic infiltration closure is available as a plugin (Sect. 4.2).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Momentum is advanced in advective (non-conservative) velocity form. For the x-component,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -g\frac{\partial \eta}{\partial x} - g\,n^2\,\frac{u\,|u|}{h^{4/3}}, \tag{2}$$

with Manning roughness n; the y-component is analogous. Retaining the advective terms $u\,\partial_x u + v\,\partial_y u$ distinguishes the formulation used here from the local-inertial approximation (Bates et al., 2010; De Almeida and Bates, 2013; Shaw et al., 2021).

This places WarpFlow between ACC-type local-inertial solvers and conservative Godunov FV/DG solvers. It retains advective momentum and therefore solves a full-SWE velocity-form balance, but it is not a Riemann shock-capturing method: mass is updated conservatively through Eq. (1), whereas momentum is advanced in the advective, non-conservative velocity form of Eq. (2). The severe-bore and shock regimes identified by LISFLOOD-FP 8.0 as requiring full conservative SWE solvers are therefore treated here as validation stress tests rather than as WarpFlow's primary design target (Shaw et al., 2021).

2.2 Grid, datum and precision

The domain is discretized on a uniform Cartesian grid with constant spacings Δx and Δy, which need not be equal, arranged as a staggered Arakawa C-grid (Arakawa and Lamb, 1977). Scalars h, z and roughness class sit at cell centers; u, Q_x on east–west faces and v, Q_y on north–south faces (Fig. 1). This staggering lets the control-volume depth update use face-centered fluxes directly and avoids pressure–velocity decoupling. The domain carries a one-cell ghost ring (padded shape $(n_y{+}2) \times (n_x{+}2)$) for boundary conditions.

Bed elevations are referenced to a centered vertical datum, $z_{\text{centered}} = z_{\text{raw}} - z_{\text{offset}}$ with $z_{\text{offset}} = \frac{1}{2}(z_{\min} + z_{\max})$. This preserves significant bits for the hydrostatic reconstruction (Sect. 2.3), which differences nearby values of the water-surface elevation η; prescribed water-level boundary or source data are datum-shifted accordingly (Sect. 2.9). The single-precision state that this precaution protects is a deliberate memory and bandwidth choice, motivated in Sect. 3.2 and validated in Sect. 5.

2.3 Well-balanced reconstruction and upwind mass flux

The face quantities used by the momentum and mass updates are constructed here. Two choices are kept deliberately separate: the well-balanced pressure-gradient reconstruction and the donor-cell depth used to convert the updated face velocity into a mass discharge.

C-property. The scheme preserves the still-water steady state of zero velocity and a flat free surface, $\eta = h + z = \text{const}$, over arbitrary wet bathymetry, i.e. the discrete C-property of Bermúdez and Vázquez-Céndon (Bermúdez and Vázquez, 1994). We use an Audusse-type hydrostatic reconstruction (Audusse et al., 2004). At a face between cells L and R, the interface bed is taken as the higher neighboring bed,

$$z_f = \max(z_L, z_R), \tag{3}$$

and the reconstructed one-sided depths are

$$h_L^\star = \max(0, \eta_L - z_f), \qquad h_R^\star = \max(0, \eta_R - z_f). \tag{4}$$

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

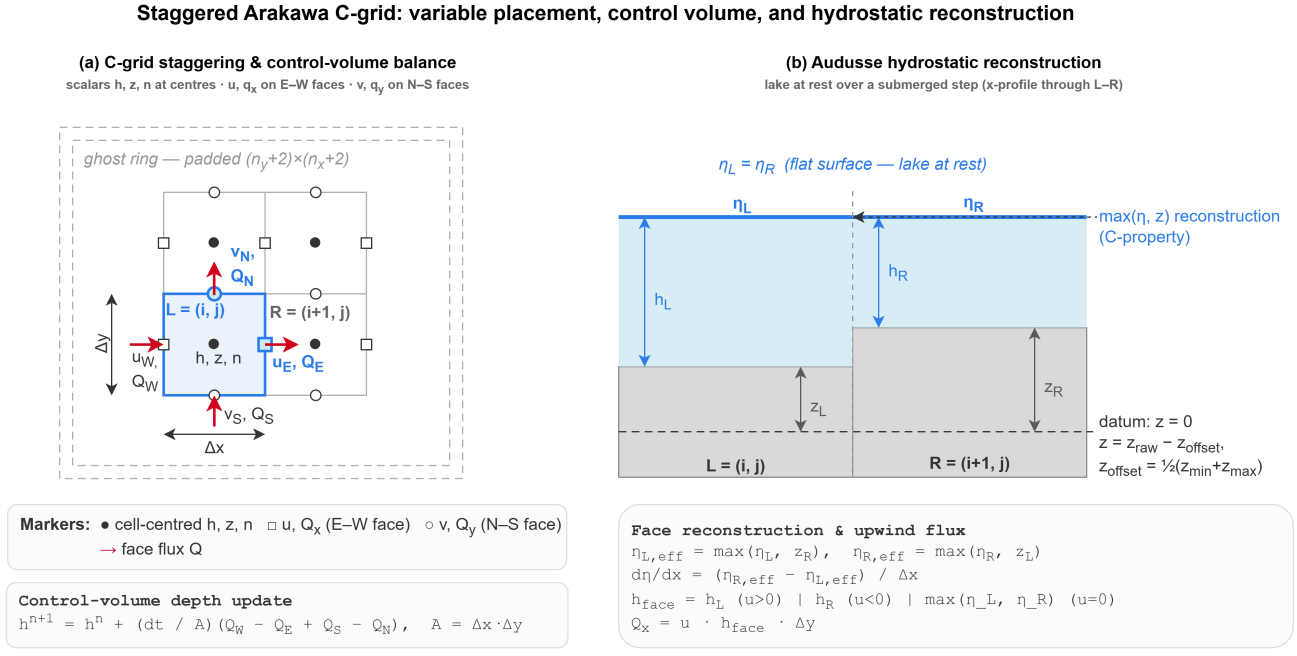


Figure 1. Staggered Arakawa C-grid spatial discretization and hydrostatic interface reconstruction: (a) Variable placement on the structured staggered grid showing cell-centered scalars (h, z, η), face-centered velocities (u, v) and face-integrated volumetric discharges (Q_x, Q_y), and ghost-ring padding for control-volume depth update; (b) Audusse hydrostatic reconstruction over a submerged bed step preserving the lake-at-rest C-property via higher-bed hydrostatic reconstruction of the effective surface-elevation gradient ($d\eta/dx$) and donor-cell face depth (h_{face}) selection.

Equivalently, the reconstructed free surfaces are

$$\eta_L^\star = z_f + h_L^\star = \max(\eta_L, z_f), \qquad \eta_R^\star = z_f + h_R^\star = \max(\eta_R, z_f). \tag{5}$$

For physically admissible cell states, $\eta_L \geq z_L$ and $\eta_R \geq z_R$, this is implemented in the compact form

$$\eta_L^{\text{eff}} = \max(\eta_L, z_R), \qquad \eta_R^{\text{eff}} = \max(\eta_R, z_L), \qquad \frac{d\eta^{\text{eff}}}{dx} = \frac{\eta_R^{\text{eff}} - \eta_L^{\text{eff}}}{\Delta x}. \tag{6}$$

For a lake at rest, the reconstructed left and right face states coincide, so the discrete pressure-gradient forcing vanishes. In the `lake_at_rest` cases this leaves $\max|v|$ and $\max|\Delta\eta|$ at the single-precision round-off floor (Sect. 5.2).

The reconstruction is separate from the friction closure. The Manning term uses the same semi-implicit face-friction update as Sect. 2.4; only the pressure-gradient state and the final flux depth are reconstructed. Therefore, velocity or arrival-time differences relative to a centred local-inertial implementation are attributed to the reconstruction and flux-depth convention, not to a changed Manning friction law. Where such differences are observed, they are reported in Sect. 5.7.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Upwind mass flux and checkerboard suppression. After the face velocity has been updated, the discharge uses the upwind hydrostatically reconstructed donor depth. For an x-face,

$$z_f = \max(z_L, z_R), \qquad h_L^\star = \max(0, \eta_L - z_f), \qquad h_R^\star = \max(0, \eta_R - z_f), \tag{7}$$

and

$$h_{q,f}^\star = \begin{cases} h_L^\star, & u_f^{n+1} > 0, \\ h_R^\star, & u_f^{n+1} \le 0, \end{cases} \qquad Q_x = u_f^{n+1} h_{q,f}^\star \Delta y. \tag{8}$$

The y-face expression is analogous, with $Q_y = v_f^{n+1} h_{q,f}^\star \Delta x$. Thus the final discharge is donated by exactly one upstream reconstructed state. On a flat bed this reduces to the usual donor-cell depth rule, while over variable topography it prevents flux through the portion of a face blocked by the higher bed.

This flux depth is distinct from the momentum/friction depth $h_{m,f}$ used in Sect. 2.4. The latter is selected from the raw cell depths using the old face velocity u^n, and is used for the wet-face test, the Manning denominator and the velocity cap. The reconstructed depth $h_{q,f}^\star$ is used only after the velocity update, when forming the mass discharge.

The donor-cell flux is compatible with the optional FCT limiter of Sect. 2.5, which scales only the outgoing faces of a donor cell when the requested outflow volume would exceed the locally stored volume. With this limiter disabled, the positivity floor of Sect. 2.6 remains the default guardrail. The upwind flux, together with the optional upwind advection terms in Sect. 2.4, provides numerical dissipation of odd–even modes. In contrast, the original LISFLOOD-FP inertial formulation of Bates et al. (Bates et al., 2010) uses a centred local-inertial discretisation; Bates et al. note that low-friction instabilities can occur in that centred approach and that friction contributes to stabilisation. Any comparison to a later de Almeida θ-diffusion variant (De Almeida and Bates, 2013) is therefore made explicitly in Sect. 5.7, rather than being assumed here.

Wall and drain interface faces. The reconstruction and upwind selection above apply only to faces between two terrain cells. Faces adjoining the special DEM cell types, the -9999 wall and -9998 drain sentinels of Sect. 3.3, are classified before either rule is evaluated. A terrain–wall face is impermeable: the normal velocity and flux are set to zero, so a wall cell never accumulates depth. A terrain–drain face is treated as an open outflow: if the terrain-side depth exceeds the wet-depth threshold, water is ejected at critical velocity, $|u| = \sqrt{gh}$, with the sign set by the terrain side; otherwise the face velocity and flux are set to zero. Faces between two drain cells are also assigned zero velocity and zero flux.

2.4 Semi-implicit momentum update

The staggered face velocities are advanced with a semi-implicit update of the two-dimensional shallow-water momentum equations in advective velocity form. For an east–west terrain–terrain face between cells L and R, the update is

$$u_f^{n+1} = \frac{u_f^n + \Delta t\left(-g\,\frac{d\eta^{\mathrm{eff}}}{dx} - C_{\mathrm{adv}}\right)}{1 + D_{\mathrm{fric}} + A_{\mathrm{damp}}}, \tag{9}$$

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

where $d\eta^{\mathrm{eff}}/dx$ is the well-balanced hydrostatic surface-gradient reconstruction of Eq. (6). The update is evaluated in Jacobi form: all velocities on the right-hand side are read from the frozen start-of-step state, and u_f^{n+1} is written to a separate velocity buffer before the buffer swap (Sect. 3.9).

Two face depths are used, and they are distinct. The momentum update uses a raw upwind momentum depth $h_{m,f}$,

$$h_{m,f} = \begin{cases} h_L, & u_f^n > 0, \\ h_R, & u_f^n < 0, \\ h_L, & u_f^n = 0 \text{ and } \eta_L > \eta_R, \\ h_R, & u_f^n = 0 \text{ and } \eta_L \le \eta_R, \end{cases} \tag{10}$$

which is used for the wet-face test, Manning friction and velocity cap. The hydrostatically reconstructed donor depth $h_{q,f}^{\star}$ of Eq. (8) is used only later, after the velocity update, to form the mass discharge. If $h_{m,f} \le h_{\mathrm{wet}}$, the face is treated as dry and both the normal velocity and discharge are set to zero.

The advective acceleration is evaluated explicitly from the frozen velocity field,

$$C_{\mathrm{adv}} = u_f^n \frac{\partial u}{\partial x} + v_f^n \frac{\partial u}{\partial y}. \tag{11}$$

The longitudinal derivative is first-order upwinded by the sign of u_f^n. For an x-face,

$$\frac{\partial u}{\partial x} = \begin{cases} \dfrac{u_f^n - u_{f-W}^n}{\Delta x}, & u_f^n > 0, \\ \dfrac{u_{f+E}^n - u_f^n}{\Delta x}, & u_f^n \le 0, \end{cases} \tag{12}$$

where u_{f-W}^n and u_{f+E}^n denote the neighboring staggered east–west face velocities. If the requested neighbor lies outside the valid stencil, the local value u_f^n is reused, giving a zero one-sided derivative at that edge.

When cross-advection is enabled, the transverse velocity at the x-face is interpolated from the four surrounding staggered v-faces,

$$v_f^n = \frac{1}{4}\left(v_{j-1,i}^n + v_{j,i}^n + v_{j-1,i+1}^n + v_{j,i+1}^n\right), \tag{13}$$

and the transverse derivative is upwinded by the sign of this reconstructed velocity,

$$\frac{\partial u}{\partial y} = \begin{cases} \dfrac{u_f^n - u_{f-S}^n}{\Delta y}, & v_f^n > 0, \\ \dfrac{u_{f+N}^n - u_f^n}{\Delta y}, & v_f^n \le 0. \end{cases} \tag{14}$$

If cross-advection is disabled, the second term in Eq. (11) is omitted. If advection is disabled entirely, $C_{\mathrm{adv}} = 0$ and the advective regularizer below is also zero.

Bed-stress resistance uses a component-wise Manning formulation on the staggered faces. Roughness values are stored as pre-squared class-LUT values (Sect. 3.3) and averaged across adjacent cells,

$$n_f^2 = \frac{1}{2}\left(n_L^2 + n_R^2\right). \tag{15}$$

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

The semi-implicit friction contribution in the denominator of Eq. (9) is

$$D_{\mathrm{fric}} = \frac{g\, n_f^2\, \Delta t\, |u_f^n|}{\max(h_{m,f}, h_{\min})^{4/3}}, \tag{16}$$

where $h_{\min}$ is the friction depth floor (`min_flow_depth_friction`, default $1 \times 10^{-3}\,\mathrm{m}$). This depth floor is a stability device for thin rain-on-grid sheets, not a change to the mass-conserving flux depth. Very shallow overland flow is a known stress case for SWE solvers: in the LISFLOOD-FP 8.0 rain-on-grid study, widespread thin water layers produced large local velocities and restrictive friction-related time steps in the DG2 formulation, motivating improved implicit friction treatments for overland flow (Shaw et al., 2021; Xia and Liang, 2018). WarpFlow therefore evaluates Manning resistance on the raw upwind momentum depth $h_{m,f}$ with a small floor, while the final mass discharge uses the hydrostatically reconstructed donor depth $h^{\star}_{q,f}$ of Eq. (8). The bed-stress closure is strictly face-normal: on an east–west face it depends on $|u_f^n|$, and on a north–south face it depends on $|v_f^n|$. The transverse velocity is used only for cross-advection and for the optional advective regularization.

The optional advective regularizer contributes another denominator term,

$$A_{\mathrm{damp}} = a_d\, \frac{S_{\mathrm{spd}}\, \Delta t}{\Delta x}, \tag{17}$$

with

$$S_{\mathrm{spd}} = \begin{cases} |u_f^n|, & \text{longitudinal advection only,} \\ \sqrt{(u_f^n)^2 + (v_f^n)^2}, & \text{with cross-advection.} \end{cases} \tag{18}$$

Here a_d is `advection_damp_coef`, which defaults to zero. Thus, in the default configuration, the denominator damping is only the semi-implicit Manning term. A nonzero a_d adds a linear regularization rate of order $a_d S_{\mathrm{spd}}/\Delta x$, without introducing a time step-singular damping term.

After Eq. (9) is evaluated, the face velocity is capped by a Froude-type bound,

$$|u_f^{n+1}| \le \max\left(0.1, c_F \sqrt{g\, h_{m,f}}\right), \tag{19}$$

with $c_F = 3$ by default. The cap uses the same raw momentum depth $h_{m,f}$ as the wet-face test and the friction denominator.

The updated velocity is then converted to a mass discharge using the hydrostatically reconstructed donor depth from Sect. 2.3, not the raw momentum depth. For an east–west terrain–terrain face,

$$z_f = \max(z_L, z_R), \qquad h_L^{\star} = \max(0, \eta_L - z_f), \qquad h_R^{\star} = \max(0, \eta_R - z_f), \tag{20}$$

and

$$h^{\star}_{q,f} = \begin{cases} h_L^{\star}, & u_f^{n+1} > 0, \\ h_R^{\star}, & u_f^{n+1} \le 0. \end{cases} \tag{21}$$

The face-integrated discharge is

$$Q_x = u_f^{n+1}\, h^{\star}_{q,f}\, \Delta y. \tag{22}$$

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

On a flat bed, $z_L = z_R$, so $h_L^\star = h_L$ and $h_R^\star = h_R$; the flux then reduces exactly to the ordinary donor-cell depth rule. Over variable topography, the reconstructed donor depth prevents flux through the part of the face blocked by the higher bed elevation.

The north–south update is identical under the exchange $u \leftrightarrow v$, $x \leftrightarrow y$, $L, R \leftrightarrow B, T$, and $\Delta y \leftrightarrow \Delta x$.

Alternative solver modes. For diagnostic and benchmark testing, the momentum balance can be simplified. Setting $C_{\mathrm{adv}} = 0$ disables advective acceleration and gives a local-inertial-like baseline. Disabling only cross-advection removes the transverse derivative and uses $S_{\mathrm{spd}} = |u_f^n|$ on x-faces ($|v_f^n|$ on y-faces). These modes are experimental baselines; the operational default retains the full two-dimensional advective update.

On damping and dissipation. The semi-implicit Manning term D_{fric} and the optional regularizer A_{damp} enter only through the denominator of Eq. (9). Their per-step contributions are proportional to Δt, so the continuous-time limit is well behaved. This denominator damping is separate from the numerical dissipation introduced by the first-order upwind advection and by the donor-cell mass flux of Sect. 2.3.

2.5 Optional flux-corrected transport limiter

Between the momentum update and the depth update the pipeline admits an optional flux-limiting stage (the `dry_check` kernel), disabled by default. It addresses the single situation in which the scheme could otherwise produce a negative depth. Near a wetting front, a cell's total outgoing face volume within one step, $\Delta t\, Q_{\mathrm{out}}$, can exceed the volume it currently stores, $h\,A$, which the conservative depth update of Sect. 2.6 would then drive below zero. A flux-corrected-transport (FCT) donor-cell limiter (Zalesak, 1979; Kuzmin, 2012) removes this at source by scaling only a cell's outgoing faces so that total outflow cannot exceed the stored volume:

$$Q_{\mathrm{out}} = \max(0, -Q_W) + \max(0, Q_E) + \max(0, -Q_S) + \max(0, Q_N), \quad \alpha = \frac{h\,A}{\max(\Delta t\, Q_{\mathrm{out}}, \varepsilon)} \in [0, 1), \tag{23}$$

each donor scaling its outgoing Q and the corresponding face velocity by α on the freshly written velocity buffers, before the Jacobi swap of Sect. 2.4; ε is a small positive guard against division by zero. Because $\alpha < 1$ only where a cell would over-drain, the limiter guarantees $h \geq 0$ unconditionally and conserves mass exactly across draining fronts.

Parallel-safety. A face is outflow for exactly one of its two adjacent cells: positive flux from the left and negative flux into the right are mutually exclusive, so only the donor writes it. The non-owner reads the flux solely to classify its sign, and scaling by $[0, 1)$ is sign-stable. The limiter therefore needs no atomics. This is a deliberate departure from the flux limiter of Bates et al. (2010), which anticipates inflowing fluxes and is consequently serial. By conservatively assuming no inflow, WarpFlow's limiter is race-free through single-donor face ownership and runs fully parallel on all three backends. It can be enabled only where exact conservation across draining fronts is required. By default, positivity is instead handled by the depth-update floor of Sect. 2.6, at the cost of small, bounded mass creation.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

2.6 Mass update and positivity

The cell depth is advanced by the discrete conservation law with a non-negativity floor,

$$h_{i,j}^{n+1} = \max\Big(h_{i,j}^{n} + \frac{\Delta t}{A}\big(Q_x^W - Q_x^E + Q_y^S - Q_y^N\big),\, 0\Big), \tag{24}$$

after storing an in-place snapshot h_{prev} for active-tile change detection (Sect. 3.4) and rate-of-rise statistics.

With the optional flux-corrected transport (FCT) limiter (Sect. 2.5) disabled (default), the $\max(\cdot, 0)$ floor acts as a lightweight, passive positivity guardrail. It catches localized, transient sub-grid CFL overshoots at wetting and drying fronts, where rapid thin-layer sheet flow can momentarily drive the flux divergence past the cell volume within one time step Δt. This clamp ensures $h \geq 0$ unconditionally, without a front-tracking algorithm.

The floor is not strictly conservative: clamping an over-drained cell creates a small amount of localized mass, the scheme's primary conservation defect. The defect is fully transparent and directly quantified. The residual of the global water budget equals the cumulative mass added by the floor, mass_error $= -V_{\text{clamp bias}}$, which the water-balance plugin tracks (Sect. 4.4).

Cells with $h \leq h_{\text{wet}}$ (`wet_depth_threshold`, default 1×10^{-3} m) are classified as dry: flux kernels return zero, and the active-tile scheduler skips the cell (Sect. 3.4).

2.7 CFL time step control

The per-cell characteristic speed S, fused with the max-depth reduction into the output buffers, is

$$S = \max(0, u_E) + \max(0, -u_W) + \max(0, v_N) + \max(0, -v_S) + \sqrt{2}\sqrt{g h}, \tag{25}$$

and the adaptive time step is $\Delta t = \mathrm{CFL} \cdot \min(\Delta x, \Delta y)/S_{\max}$, capped by $\Delta t_{\max}$; the default is $\mathrm{CFL} = 0.95$.

Two choices in Eq. (25) are deliberate. The advective contribution sums the net outgoing face velocities rather than a centered average, enforcing that the directional Courant numbers satisfy $C_x + C_y \leq 1$ for the 2D unsplit update. The $\sqrt{2}$ factor accounts for isotropic diagonal wave propagation. This form constrains full non-linear momentum advection alongside gravity-wave propagation, allowing safe operation up to the theoretical limit $\mathrm{CFL} \approx 1.0$.

A calm-water threshold drops the celerity term $\sqrt{2}\sqrt{g h}$ when the depth h exceeds `calm_h_floor`, the net outgoing advective speed is below `calm_v_epsilon`, and the four-neighbor surface gradient satisfies $\max|\Delta\eta| <$ `calm_eta_grad_epsilon`. On a lake at rest, all adjacent cells satisfy $\max|\Delta\eta| \leq \varepsilon$, so the celerity term is unnecessary and would otherwise over-constrain the time step in large ponded areas. Computing the CFL criterion requires a GPU reduction of the maximum characteristic speed. The async side-stream that hides its latency, and the adaptive stride that controls its execution frequency, are described in Sect. 3.5.

2.8 Domain-boundary conditions

This section concerns the outer rectangular domain boundary only. Interior forcing (rain, discharge, prescribed water level, drainage, culverts) is a source term (Sect. 2.9). The flux behavior at interior wall and drain interfaces is part of the face-flux computation (Sect. 2.3), and the cell-type sentinel encoding belongs to the data model (Sect. 3.3).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

The rectangular boundary is imposed through the one-cell ghost ring, each edge carrying an independent type so that mixed conditions may coexist: (i) `free_flow`: a zero-gradient (Neumann) condition; the interior row or column is copied outward into the ghost cells, with corner cells filled diagonally, so water leaves the domain without reflection; (ii) `wall`: a zero normal-flux (reflective) condition, $u = 0$, $Q_x = 0$ across the edge; and (iii) `skip`: no ghost fill, safe only where every adjacent interior cell is itself a drain or wall, so no flux would cross the edge in any case. In the tiled backend the rectangular boundary is handled implicitly: an out-of-bounds tile lookup returns `CELL_WALL`, so the boundary kernel is a no-op.

A domain need not be rectangular. Irregular outlines, such as catchment divides or administrative bounds, are encoded as `CELL_DRAIN` in the DEM (Sect. 3.3). Cells outside the true domain are then excluded from the terrain index and, in the tiled backend, carry no storage at all (Sect. 3.6), while the terrain–drain faces along the outline receive the critical-ejection treatment of Sect. 2.3.

2.9 Source terms, interior forcing and sub-grid features

All interior additions and removals of water, represented by the S_h term in Eq. (1), are treated here as first-class source terms, whether they act on the full grid (rain, infiltration) or at points and lines anywhere in the interior (discharge, prescribed water level, drainage inlets, culverts). Point and line sources are rasterized to ghost-padded cell indices: points directly, lines by Bresenham's algorithm. Where several sources fall on one cell, they are collapsed and folded into a single per-cell operation, so co-located sources never demand hardware atomics and give identical CPU/GPU results.

Rain-on-grid and infiltration. Rain-on-grid forcing, in which net rainfall enters the mass balance directly as a distributed source term, follows established direct-rainfall practice for high-resolution shallow-water models (Sampson et al., 2013); the accumulated runoff source RR in Eq. (1) corresponds to the net-rainfall term of Guo et al. (2021). Rainfall is supplied as one of three types: a single constant rate, a spatially uniform time series, or a fully spatiotemporal field. In every case it is applied per terrain cell from a coarse rain grid projected on the fly onto the DEM raster, for example from a 1 km rain grid to a 2 m model grid, through two precomputed one-dimensional index maps, one per axis. Storing these two 1-D maps rather than a full 2-D correspondence reduces the mapping memory from $O(n_x n_y)$ to $O(n_x+n_y)$. The sign asymmetry between a top-left-origin DEM and a bottom-left-origin NetCDF rain grid is absorbed directly into the floor division of the index maps. Infiltration is drawn from a class lookup table (LUT; Sect. 3.3), and the unpulsed update is $h^{n+1} = \max\big(h^n + (R - I)\Delta t, 0\big)$.

In single-precision IEEE 754 arithmetic, adding a small source increment $\Delta h = (R - I)\,\Delta t$ to a large depth h truncates the mantissa whenever $\Delta h < \frac{1}{2}\,\mathrm{ULP}(h)$, where $\mathrm{ULP}(h)$ is the magnitude of the least significant mantissa bit at depth h. To preserve exact discrete mass conservation against round-off, an optional source-pulsing scheme updates a cell only every $2^{s(h)}$ steps, using the scaled instantaneous increment $2^{s(h)}\Delta h$.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

The stride exponent $s(h)$ scales with local depth to match power-of-two mantissa thresholds:

$$s(h) = \begin{cases} 0, & h \le 1\,\mathrm{m}, \\ 2, & 1\,\mathrm{m} < h \le 2\,\mathrm{m}, \\ 3, & 2\,\mathrm{m} < h \le 4\,\mathrm{m}, \\ 4, & 4\,\mathrm{m} < h \le 8\,\mathrm{m}, \\ 5, & h > 8\,\mathrm{m}. \end{cases} \tag{26}$$

For a target minimum $\Delta(R-I) = 3\,\mathrm{mm/h}$ at $\Delta t = 0.2\,\mathrm{s}$ ($\Delta h_{\min} \approx 1.67 \times 10^{-7}\,\mathrm{m}$), this ladder maintains $2^{s(h)}\Delta h_{\min} \ge 2.8\,\mathrm{ULP}(h)$, guaranteeing truncation-free float accumulation up to $h \le 16\,\mathrm{m}$. The ladder in Eq. (26) approximates a $\lfloor \log_2 h \rfloor$ dependence but is not identical to it. It deliberately brackets the mantissa-boundary exponent from above: the ladder steps from 0 directly to 2, so it never under-strides. It is evaluated by simple conditional checks to avoid an expensive logarithm in the kernel.

A cell at spatial index (i, j) applies its pulse at step k if and only if

$$(k + i + j) \equiv 0 \pmod{2^{s(h)}}. \tag{27}$$

Phase-staggering the trigger by $(i + j)$ prevents grid-wide synchronized pulsing. Because the pulse scales the instantaneous forcing $2^{s(h)}\Delta h(t_k)$ rather than accumulating intermediate states, the scheme trades mantissa truncation for a minor temporal sampling error whenever R or Δt varies across the $2^{s(h)}$ interval.

Interior discharge and prescribed-water-level sources. Discharge sources (Q, fixed or time-varying) add depth $\Delta h = Q\,\Delta t/A$ distributed evenly over their cells, and are sum-reduced where cells coincide. Prescribed-water-level sources (η, fixed or time-varying) set $h = \max(0, \eta_{\text{target}} - z)$ and are datum-shifted by z_{offset} to match the centered datum (Sect. 2.2); where cells coincide they are max-reduced. Time-varying values are linearly interpolated from the stored series. These sources are not confined to the domain boundary: like point and line sources they may be placed anywhere in the interior, which is why they are treated here rather than in Sect. 2.8. The signed water-level term is one of the inputs tracked by the water-balance diagnostic (Sect. 4.4).

Drainage inlets. Drainage inlets (gully gutters, curb drains) are state-dependent point sinks governed by the orifice equation,

$$Q_{\text{inlet}} = C_d\,A\,\sqrt{2g\,H}, \qquad H = \min\big(\max(h - z_{\text{inv}}, 0),\, H_{\max}\big), \tag{28}$$

where h is the local surface depth, C_d the discharge coefficient, A the effective throat area, z_{inv} the inlet invert relative to the ground, and $H_{\max}$ a cap on the driving head. The extracted volume per time step is $V = \min(Q_{\text{inlet}}\,\Delta t, \phi\,h\,A_{\text{cell}})$, with the positivity factor $\phi = 0.99$ preventing negative depths, and an optional per-inlet discharge cap representing the connecting-pipe capacity. Unlike a prescribed fixed-capacity sink, the extracted flow responds dynamically to the surface state: it vanishes on dry cells and grows with ponding depth. Absent detailed inlet metadata, we adopt a street-gully default ($C_d = 0.40$, $A = 0.01\,\mathrm{m}^2$, $z_{\text{inv}} = 0$, $H_{\max} = 0.30\,\mathrm{m}$), consistent with experimental work on surcharged grated inlets reporting effective discharge

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

coefficients in the range 0.14–0.41 and recommending revision of the commonly used 0.6 value (Tellez-Alvarez et al., 2021); per-inlet geometry can be supplied where available. Co-located inlets are evaluated as a grouped per-cell operation, so they share one depth-limited extraction without GPU atomic locks. A coupled 1-D pipe network providing tailwater and backflow feedback is out of scope for the core engine and is exposed through the plugin interface (Sect. 4).

Culverts. Sub-grid conduits are modeled as inlet/outlet cell pairs governed by a three-regime rating curve, keyed on the inlet-cell water depth h_{inlet} relative to the conduit opening height h_k:

$$Q = \begin{cases} 0 & \text{if } h_{\text{inlet}} \le h_{\text{wet}} \quad \text{(dry)}, \\ \min\!\left(C_w\, w\, h_{\text{inlet}}^{3/2},\; v_{\text{barrel}} A\right) & \text{if } h_{\text{inlet}} < h_k \quad \text{(weir flow)}, \\ C_o\, A \sqrt{2g\left(h_{\text{inlet}} - \tfrac{1}{2} h_k\right)} & \text{if } h_{\text{inlet}} \ge h_k \quad \text{(orifice flow)}, \end{cases} \tag{29}$$

where w is the opening width (or diameter $2r$ for circular pipes), A is the full cross-sectional area, v_{barrel} is the open-channel design velocity, and the default coefficients are $C_w = 1.8$ and $C_o = 0.60$, following standard weir/orifice culvert-hydraulics practice (Chow et al., 1988). The volume extracted from the inlet cell per time step is clamped to $V_{\text{in}} = \min(Q\,\Delta t,\; h_{\text{inlet}}\, A_{\text{cell}})$ to preserve positivity.

The transit time along the barrel is represented without integrating the full 1-D Saint-Venant equations. Extracted volume is enqueued into a lumped-parameter first-in-first-out (FIFO) queue with timestamp t; if the buffer reaches capacity, excess volume is merged into the latest slot, conserving the enqueued volume. Water is dequeued into the downstream cell when

$$t_{\text{dequeue}} \ge t + t_{\text{travel}}, \qquad t_{\text{travel}} = \frac{L}{v_{\text{barrel}}}, \tag{30}$$

where L is the culvert length and v_{barrel} follows the Manning–Strickler formula (Chow et al., 1988), $v_{\text{barrel}} = \frac{1}{n} R_h^{2/3} S_0^{1/2}$ (default $n = 0.011$, with full-section hydraulic radius R_h from the conduit geometry). Both circular and rectangular geometries are supported in the hydraulic radius.

This captures the phase lag and volume storage of sub-grid conduit transport at minimal cost, but the surrogate implies three explicit operational boundaries: (i) flow is strictly unidirectional from inlet to outlet; (ii) tailwater pressure at a submerged outlet exerts no backwater feedback on inlet extraction; and (iii) exchange occurs purely as a mass source/sink without momentum transport. This places WarpFlow's culvert model close to the head-driven 1D-element representation benchmarked by Oberauer et al. (2026), who compare terrain-opening, explicit 1D-culvert and coupled-sewer treatments of short crossings and conclude that the head-difference culvert element is a practical baseline that captures inlet/outlet losses and pressurised/free-surface transitions, while a coupled-sewer element is required where internal conduit transients and outlet backwater dominate.

Extending the source terms. The terms above constitute the solver core. Each can be extended or replaced at runtime through the plugin layer (Sect. 4) without editing the flux kernels. The constant infiltration, for instance, is replaced by a dynamic capacity model in the bundled infiltration plugin (Sect. 4.2). The plugin architecture, the four bundled plugins, and the scope boundary between this paper and its follow-up are set out in Sect. 4.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

3 Software design and high-performance implementation

3.1 Design goals and core engineering principles

Traditional 2D shallow-water solvers are often monolithic, closed-source executables, academic testbeds or file-driven legacy binaries. Such tools are numerically sound, but they are rarely built to sit inside a modern automated pipeline, where models must be generated ad hoc from live municipal GIS repositories, re-conditioned on the fly, or driven by real-time sensor feeds such as radar rainfall nowcasts and stream-gauge telemetry. WarpFlow is designed from the ground up as an API-first, Python-native hydrodynamic engine for the operational demands of Urban Digital Twins (UDTs), coupling a high-level geospatial interface to low-level GPU kernels.

The numerical formulation of Sect. 2 is computationally lightweight for modern hardware. Execution speed is governed instead by memory bandwidth, streaming-multiprocessor (SM) occupancy, and latency. The engineering of WarpFlow therefore follows two pragmatic laws: do less work, and then do the remaining work faster.

Do less work. The solver spends near-no cycles on inactive cells or stagnant flow. Threads are notlaunched or early return for non-terrain cells, and quiescent ponded regions are skipped during momentum updates by active-tile tracking (Sect. 3.4). Expensive global reductions, such as the characteristic CFL speed check, are amortized by adaptive recompute cadences (Sect. 3.5). Optional modules and kernel parameters also follow a zero-tax principle: a disabled feature incurs near-zero cost during the simulation loop.

Do the work faster. Compute kernels maximize hardware utilization through memory coalescing, warp-aligned block geometries (16×16 and 256-thread launches), and bit-addressed tile layouts. Because the numerical scheme guarantees non-negative depth ($h \geq 0$) by construction (Sect. 2.6), the kernels never produce floating-point exceptions or `NaN` values. This makes aggressive compiler optimizations, `fastmath`, denormals-as-zero, and fused multiply–add, safe to enable on both the LLVM and PTX compilers.

Software architecture.

Figure 2 shows how these principles map onto the software architecture, which is divided into three functional phases.

1. **Initialization and strategy dispatch (Fig. 2A):** a stateless `SimulationRunner` manages workspace preparation, declarative TOML parsing, and domain setup. It instantiates an abstract `Simulation` base through a Strategy pattern, dispatching to one of three backends (`SimulationCPU`, `SimulationCUDA`, or `SimulationCUDATiled`).

2. **Main time step pipeline (Fig. 2B):** the core loop (`run_timestep`) runs a phase-gated sequence: update boundary conditions, evaluate flux updates, apply hydrological losses and sub-grid sources, and close with active-tile detection and adaptive time step control.

3. **Asynchronous streams and decoupled I/O (Fig. 2C):** hot-loop stalls are removed by offloading blocking operations to asynchronous side-streams. The CFL reduction overlaps on a dedicated CUDA stream, and frame outputs pass through a zero-copy shared-memory pool to a background daemon writer process.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

To stay accessible in both municipal and academic environments, WarpFlow builds entirely on distributed `conda-forge` channels and relies on pre-compiled Python GIS and NVIDIA CUDA bindings, which removes complex build chains and speeds up dataset initialization.

WarpFlow ships with built-in hydraulic statistics and hazard diagnostics (Sect. 4.3), but it encourages a user-supplied plugin methodology. A hook-based extensibility layer lets users attach custom lifecycle observers or substitute core stages, such as grid-source rain and infiltration handling, through runtime kernel overrides, without modifying the core flux solver (Sect. 4). During initial architectural evaluation, cross-accelerator frameworks such as Taichi (Hu et al., 2019) were tested to assess backend portability across multi-vendor accelerators. Hand-tuned Numba-CUDA (Lam et al., 2015) and CuPy (Okuta et al., 2017) kernels gave superior memory bandwidth, register discipline, and control over shared-memory reductions on commodity NVIDIA hardware. We therefore standardized on Numba-CUDA as the primary GPU stack. A related route to the same single-source goal is SYCL, which Büttner et al. (2025) use to run a 2D shallow-water solver across CPUs, GPUs and FPGAs from one code base while analyzing the resulting performance-portability trade-offs. WarpFlow shares that single-source objective but makes a different technology choice, targeting the commodity NVIDIA hardware of a municipal IT department.

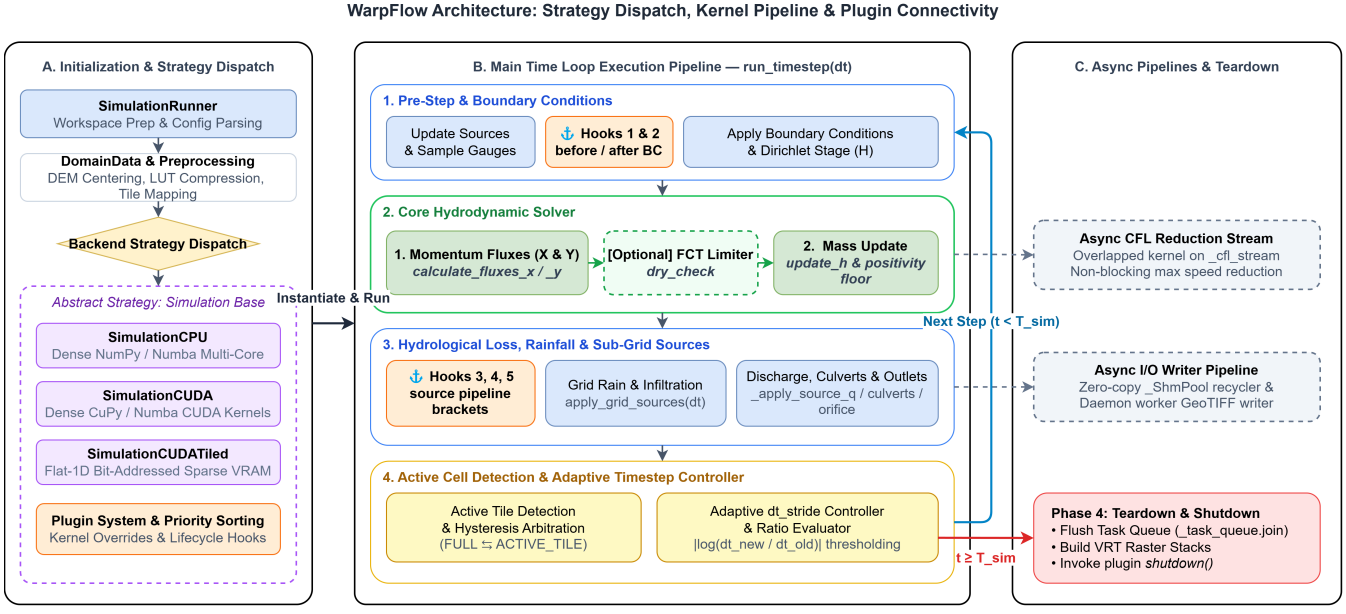


Figure 2. WarpFlow software architecture and execution pipeline: (a) initialization and backend strategy dispatch; (b) per-step execution loop with phase-gated plugin hooks (1–5); and (c) asynchronous CFL and I/O side-streams.

3.2 Precision discipline

All constants and array identifiers are explicitly typed with the canonical NumPy dtypes (Harris et al., 2020) (`float32`, `int8`, `int32`) to prevent implicit promotion to `float64` when values flow into CUDA kernel arguments. Such promotion

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

would double per-thread register usage, roughly halve SM occupancy, and raise shared-memory pressure during tile-boundary reductions. The CPU backend can run the full solver in `float64` without a severe throughput penalty, but GPU execution is enforced to single precision: when a CUDA backend is requested with `precision = double`, the configuration layer force-downgrades to single with a warning.

Half precision is supported selectively, for storage only. Auxiliary and diagnostic buffers with relaxed accuracy requirements, for example the statistics-plugin hazard fields (Sect. 4.3), may be stored as `float16` to save bandwidth. The numerical core remains pure `float32`: its accumulation-sensitive flux and friction terms cannot tolerate half-precision round-off, and the configuration layer never lets the safe compute type fall to `float16`. This single-precision state is a deliberate trade-off between numerical fidelity, memory bandwidth and cross-platform portability, and it is validated rather than assumed (Sect. 5).

3.3 Data model and static-field compression

Sparse terrain indexing and multi-semantic DEM sentinels. Only terrain cells carry computational work. Cell type is an `int8` derived from DEM sentinel values, which encode hydraulic interface semantics directly in the elevation raster: solid walls and buildings (-9999) and drainage sinks (-9998). The flat indices of the terrain cells are precomputed once at domain preparation and row-major or tile sorted, so that each worker receives contiguous, cache-line-aligned chunks. This single terrain-index list is the substrate for both execution modes of Sect. 3.4. How it is traversed, densely over all terrain cells, or sparsely over active tiles, is described there and is identical in semantics on CPU and GPU.

LUT compression. Static scalar fields (e.g., Manning n^2 roughness and infiltration rates) use a compressed representation: a `uint8` class map coupled with a `float32` lookup table (LUT) of at most 256 entries. Depending on the input raster, the LUT is built under one of three regimes: uniform (single entry), exact bijection (≤ 256 distinct values, fully lossless), or quantile-representative (nearest-neighbor snapping). During flux calculations, kernels evaluate parameters through `lut[class[j,i]]`. This cuts per-cell static-field bandwidth nearly fourfold and keeps a branch-free execution flow.

Dynamic source buffers. Interior hydraulic forcing follows a build-once, lightweight-update lifecycle (Sect. 2.9). Complex structural preprocessing, such as rasterizing spatial points and lines to ghost-padded indices, collapse-and-fold grouping of co-located sources, and compressed-sparse-row (CSR) mapping, runs once at load time and is cached in VRAM. During the simulation loop, only lightweight scalar buffers (e.g., interpolated discharge `d_q_val` and water level `d_h_val`) are refreshed and transferred to the device, with no array reallocation or geometry transfer. Structural rebuilds fire only when sources are added, removed, or toggled at runtime, signaled by a single state flag. Spatially distributed rainfall uses the same two-stage index mapping and uploads only coarse rain-rate arrays per step.

3.4 Execution modes: full, active tiles and auto-switching

WarpFlow separates how the domain is stored from how work is scheduled over it. The two axes are orthogonal (Fig. 3): the storage layout is either dense or tiled (Sect. 3.6), while the execution mode, the subject of this section, decides which cells are launched each step. The same single-source kernels run under any execution-mode combination.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Full mode. In full mode (Fig. 3A) the solver launches every terrain cell of the row-major index list (Sect. 3.3) each step, accessing memory in long contiguous 1-D runs. It carries no activity bookkeeping: no detection, no compaction, no per-step reorganization, and non-terrain cells (buildings, out-of-domain) are never launched. This is the fastest mode when most of the domain is active.

Active-tile mode. In active-tile mode (Fig. 3B) the solver launches only the cells of active tiles and early-returns elsewhere, at the price of an explicit bookkeeping pipeline that full mode does not pay. A tile is active when all of its cells satisfy $h > h_{\text{wet}}$ together with $|h - h_{\text{prev}}| > \Delta t \cdot 10^{-4}$ or $|\mathbf{v}_c|^2 > 10^{-4}$. A lake at rest is deliberately not active, yet still receives rain and infiltration until it wakes.

A two-kernel detect/dilate/compact pipeline rebuilds the launch list: a shared-memory reduction flags active tiles, and an atomic-append compaction collects them. All active tiles are dilated by one tile, so every neighbor of a truly active cell is also launched and owns the shared face it must write (Sect. 3.9). Skipping inactive terrain in this way generalizes the block-deactivation optimization of GPU finite-volume SWE solvers, where dry blocks are excluded from the update and dry neighbors of wet blocks are pre-activated to admit the advancing front (Dazzi et al., 2018).

The activity tile is a separate tiling from the storage tile of Sect. 3.6. Its side length is configurable (default 32) and can be made smaller to target slowly filling urban depressions more tightly, without touching the fixed 32×32 VRAM storage layout. The two tilings are fully decoupled.

Auto mode. In `auto` mode a hysteresis state machine switches between the two. It enters active-tile execution when the launched-work fraction falls below 0.55 and returns to full above 0.75, with a minimum dwell of 50 steps and periodic probing, all normalized against terrain-capable tiles rather than total storage tiles. The trade-off is stated plainly and, we believe, is rarely quantified. A nearly fully wet domain (e.g. under spatially uniform rain) is faster in full mode, precisely because the detect/dilate/compact pipeline is then pure overhead, bookkeeping for a skip that never happens. A locally wetted domain (a one-sided storm, a constricted floodplain) is faster in active-tile mode. Smaller activity tiles skip more cells but cost more bookkeeping. We measure both sides of this balance directly in Sect. 6.3.

3.5 Amortizing the CFL reduction: async side-stream and adaptive stride

The CFL reduction Eq. (25) is the one intrinsic hot-loop reduction, and it is amortized twice over. First, it runs on a dedicated side-stream (GPU only): the characteristic-speed kernel and its readback launch on `_cfl_stream`, and the loop consumes the previous async result (`block=False`), accepting a max-speed that is stale by about one recompute window. Flash-flood Δt drift is smooth enough that this is acceptable, and a slightly lower configured `CFL` absorbs it. The side-stream hides the synchronization behind main-loop kernels and prevents an immediate implicit-synchronization stall. Second, the recompute cadence itself is adaptive. A multiplicative-ratio controller on $|\log(\Delta t_{\text{new}}/\Delta t_{\text{old}})|$ loosens the stride when Δt is stable and shrinks it (down to every step, on a $\geq 15\%$ jump) when it is not. On a calm domain the costly global reduction runs rarely, and on a rapidly evolving one it runs often. A $\Delta t_{\max}$ headroom boost fires only after several stable samples away from calm, which prevents overshoot during a dry-start onset where $u_{\max} \approx 0$. Outside this, host synchronization is essentially absent from the critical hot path.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

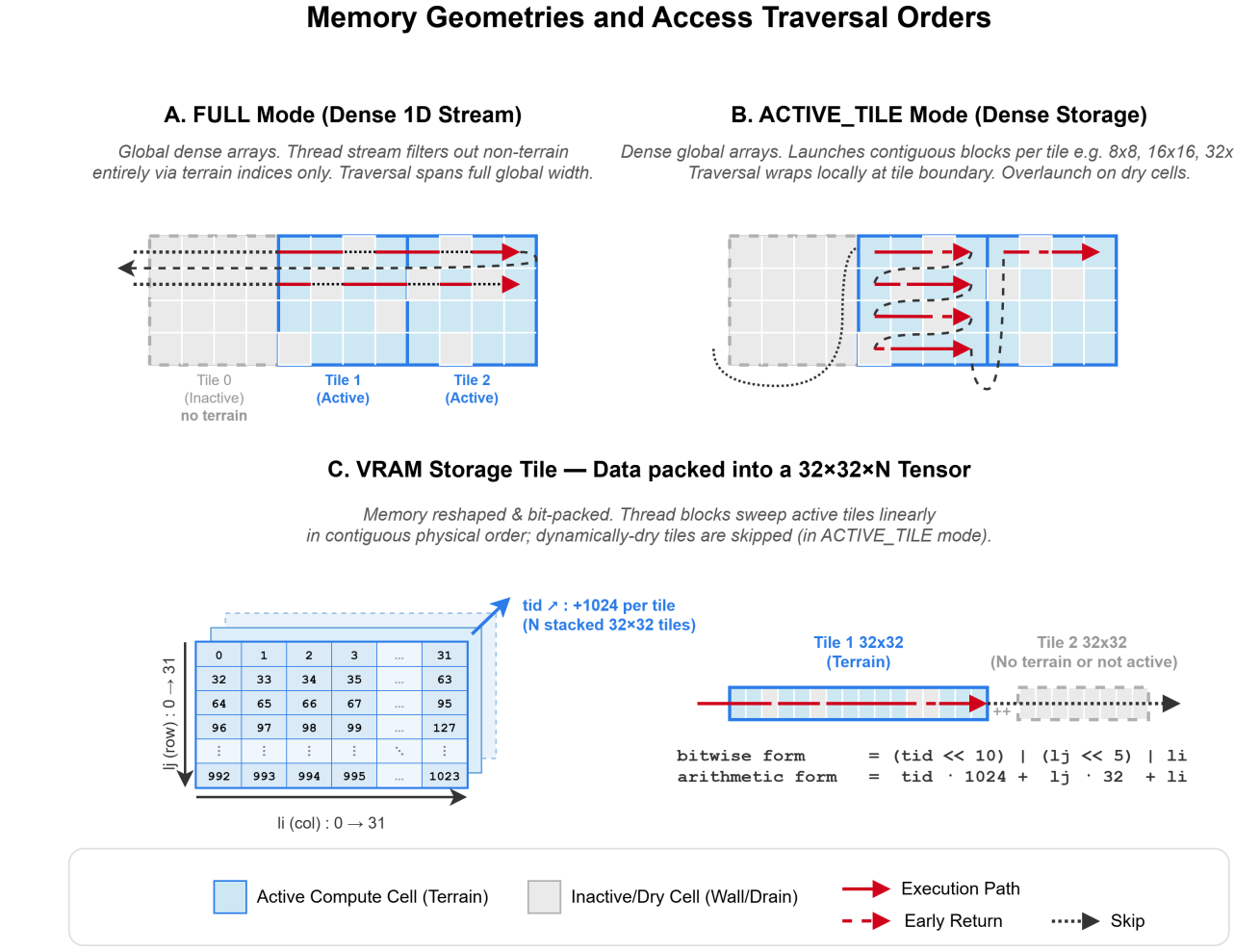


Figure 3. The two orthogonal axes of WarpFlow's execution. *Execution mode* (this section): (a) full mode launches every terrain cell in contiguous 1-D runs with no bookkeeping; (b) active-tile mode launches only active tiles and early-returns elsewhere, maintained by a detect/dilate/compact pipeline. *Storage layout* (Sect. 3.6): (c) tiled VRAM packs each 32×32 tile into a contiguous slab of an N_T-tile tensor with bit-addressed indexing. Any execution mode runs on either storage layout.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

3.6 Tiled storage geometry and bit-addressing

The tiled storage layout is the second axis of Fig. 3 (panel C). It is a pure remapping of memory; the numerics are identical to the dense layout (Sect. 3.9). The layout is chosen so that a cell and its stencil neighbors sit close together in the address space. The motivation is locality rather than coalescing alone. In the dense row-major layout a cell's north/south neighbor lies n_x elements away, far enough on a large grid that it is routinely evicted from L2, the on-chip cache shared across all SMs, before the stencil reuses it. In a 32×32 tile the whole 3×3 stencil footprint occupies a small contiguous region that stays resident. The tiled layout therefore raises the L2 hit rate on the vertical ($j \pm 1$) neighbor reads, at the price of a few extra integer operations per access for the bit-addressing (Eq. (31)). On regular, fully wet domains these two effects very nearly cancel, so the tiled backend tracks dense throughput to within measurement scatter (Sect. 6.2). On such domains the tiled layout is therefore not a throughput optimization but a throughput-neutral one. Its decisive benefits are the terrain-sparse allocation and row-pitch independence described below, with any residual stencil-locality gain expected only on irregular urban rasters.

State fields are held as a single contiguous device allocation of $N_T \times \text{TILE_SIZE}^2$ elements, with $\text{TILE_SIZE} = 32$. The tile dimension is 32 deliberately: it is the warp size, so one tile row maps to one warp for coalesced loads, and it tiles cleanly under the 16×16 (256-thread) launch geometry of the compute kernels. Array indexing is a division-free bit-packing,

$$\text{address} = (\texttt{tid} \ll 10) \,|\, (\texttt{lj} \ll 5) \,|\, \texttt{li}, \qquad \texttt{li} = \text{packed} \,\&\, 31, \;\; \texttt{tid} = \text{packed} \gg 10. \tag{31}$$

Interior-neighbor accesses stay within the local tile and bypass the global tile-map lookup entirely. This is the common fast path for the 900 of 1024 interior cells per tile ($\approx 88\%$); accesses fall back to a cross-tile lookup (`tr_global`/`tw_global`) only at the tile perimeter. This flat, bit-addressed sparse layout follows the same design goal as spatially sparse GPU data structures (Hu et al., 2019): store and launch only what carries work.

Terrain-sparse VRAM allocation. Memory is allocated only for terrain-bearing tiles, so a sparse urban domain consumes VRAM in proportion to its physical extent rather than its rectangular bounding box. The per-cell byte budget is identical in the two layouts, 9 `float32`, 1 `int8` and 2 `uint8`, i.e. 39 B/cell, so the saving does not come from packing bytes more tightly. It comes entirely from not allocating the non-terrain fraction. The dense layout pays 39 B for every cell of the padded rectangle (plus a terrain-index array), whereas the tiled layout pays only for tiles that contain terrain, plus sparse auxiliary arrays (tile map, rain index maps, LUT scalars, culvert metadata). For a sparse city raster this terrain-sparse allocation is the dominant contribution to the footprint reduction quantified in Sect. 6.4.

Row-pitch independence. A further benefit of the tiled layout is immunity to unaligned row pitches. In a standard 2D GPU array, if the row stride $(n_x + 2)$ is not a multiple of 32 `float32` elements (128 bytes), warp loads cross L2 cache-line boundaries and degrade throughput. WarpFlow's tiled VRAM instead packs each 32×32 tile into a contiguous 4 KB block. Because every row within a tile is exactly 32 `float32` elements (128 bytes), every warp load achieves full coalescing and cache-line alignment, regardless of global grid dimensions or unpadded DEM boundaries.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

3.7 CPU backend

The CPU kernels are `prange`-parallel over the terrain-index list and run on the dense storage layout only. Tiled storage is pointless on CPU, whose kernels write back through dense arrays regardless. Every kernel is parallel: the single-donor face ownership and the single-donor limiter make it race-free without serialization. Numba's `@njit(fastmath=True)` enables reassociation and fused multiply–add, `cache=True` persists the compiled binaries so only the first run pays the JIT cost, and the CFL scratch buffers are sized once to the maximum work count and never reallocated as the active set changes.

The CPU backend is not the performance target, and it is retained deliberately for three reasons. It is the deterministic reference against which the GPU backends are validated (Sect. 3.9). It is a fallback for environments without an NVIDIA GPU. It also lowers the barrier to extension, letting plugin authors develop and test new physics against the same kernels on a regular PC without writing CUDA. Because all three backends share one kernel formulation, a plugin verified on CPU behaves identically once moved to the GPU.

3.8 GPU backends

Both GPU backends compile the same device functions with `@cuda.jit(fastmath=True, cache=True)` and share several optimizations. The CFL and maximum-depth diagnostics use a block-level binary-tree reduction in shared memory, followed by a single `atomicMax`/`atomicAdd`. The tiled backend fuses maximum speed and maximum depth into one tree pass. Atomics are used sparingly and only where a genuine race may exist: active-tile counting (Sect. 3.4), the final aggregation of the shared-memory CFL and max-depth reductions, and culvert inlet/outlet exchange. The co-located source folding of Sect. 2.9 means ordinary source forcings need no atomics at all.

Host–device interaction is minimized. Host buffers are page-locked (`cuda.pinned_array`) so transfers proceed by DMA without page faults, and CuPy supplies NumPy-compatible zero-copy views into device memory for host-facing arithmetic that allocates nothing. We use CuPy only for fills and zero-copy views, never in the hot loop: many of its own algorithms (e.g., reductions, `cumsum`) draw on an internally managed memory pool that makes them unfit for per-iteration use. Launch geometry is 16×16 2-D blocks (a warp multiple, optimal for coalesced tile-local access) and 256-thread 1-D blocks, with launcher wrappers computing the grid size at call time so the grid tracks the work count without recompilation.

3.9 CPU–GPU parity by design

Parity between the three backends is not reconciled after the fact. It falls out of the way the solver is built. There is a single kernel formulation and a single abstract `Simulation` base. The CPU, dense-CUDA and tiled-CUDA backends differ only in their memory layout and their launch mechanics, never in their arithmetic. Every feature is implemented and maintained on all three at equal priority. There is no second-class backend and no GPU-only physics, so a run configured identically produces the same result, within single-precision tolerance, whichever backend executes it.

Backend agreement and kernel parity. Equivalence rests on identical arithmetic by construction. To keep uniform kernel signatures across backends and preserve a clean, zero-overhead plugin API, the dense CUDA backend intentionally retains

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

standard 2D indexing that matches the CPU path and omits row-pitch padding. Pitch padding could align horizontal memory transactions in dense mode, but it would alter kernel signatures and boundary conditions, and it would not resolve the vertical 2D stencil strides ($j \pm 1$), which force multi-kilobyte VRAM jumps on large dense grids. The tiled CUDA backend is a pure memory remap of the dense backend: it addresses both horizontal coalescing and vertical 2D stencil locality in the storage layout, yielding better cache locality while executing the same operations, without altering the kernel math across backends.

Jacobi evaluation and race freedom. Two complementary properties guarantee traversal-order independence and race freedom. First, the momentum update is evaluated in Jacobi form (Sect. 2.4), reading a frozen start-of-step velocity state so that results are independent of cell traversal order, the one place where CPU SIMD, dense-CUDA and tiled-CUDA thread orders would otherwise diverge. The Jacobi form uses two staggered velocity buffers (≈ 8 B/cell): flux kernels write u^{n+1}, v^{n+1} into a separate buffer, made current by a pointer swap after the full update sequence. Because flux kernels are ALU-bound (`sqrt`/`cbrt`/friction), removing the write-after-read hazard adds no net bandwidth penalty while restoring directional isotropy. Second, single-writer face ownership ensures race-free flux writes without atomics: each interior face is written by exactly one cell (east/north unconditionally; west/south only if the neighbor is inactive). In active-tile execution (Sect. 3.4), the detect/dilate/compact pipeline launches the owning neighbor of every active cell, so every shared face is updated deterministically.

Equivalence, not bit-identity. The backends are held to numerical equivalence within single-precision tolerance, not to bitwise identity. The kernels are identical only up to compilation: the CPU path is lowered through LLVM and the CUDA paths through NVVM/PTX, and `fastmath` reassociation is not guaranteed to be the same across those compilers. SIMD and warp reductions further associate floating-point operations in different orders. The residual difference is therefore pure `float32` rounding, of order 10^{-8}–10^{-7} in max-depth (Sect. 5.3). Because the GPU backends track the CPU reference to this tolerance, the deterministic CPU path serves as the audit for small cases, and large problems can be tested safely on GPU. One caveat: the asynchronous CFL reduction (Sect. 3.5) slightly alters the time step trajectory, so equivalence is reported only for `dt_stride=1` or a fixed time step. In practice this is a non-issue.

3.10 Asynchronous output

Output is decoupled from computation. In the tiled backend a frame is first reassembled from the flat tile storage into a dense raster by `prange`-parallel scatter kernels, compiled with the same Numba `@njit` backend as the CPU solver, independently of the chosen compute backend, and its ghost ring is stripped. A daemon writer process then consumes frames and writes LZW-compressed GeoTIFFs while the solver advances, lowering its scheduling priority (`os.nice`) to yield the machine when idle. Frame buffers are passed by a single `memcpy` into shared memory with descriptors. A best-fit shared-memory pool recycles segments, amortizing the cost of creating a mapping down to a steady-state copy. A bounded in-flight queue provides back-pressure with a watchdog timeout, and the main loop synchronizes only when it needs a write barrier before assembling a final virtual-raster stack.

Point read-back, an inherently device-to-host operation, is likewise kept minimal in the critical path by a single batched sampler (Fig. 4). A consumer (e.g., a stage gauge, a culvert inlet) registers as a watcher of cell (j, i) under a strict build-

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

then-sample lifecycle (Fig. 4A) and receives a slice index (`handle = slice(start, stop)`). The manager bakes these indices into 1D device arrays and allocates contiguous pinned host buffers. At each time step a single 1D GPU kernel (`sample_2d_at_locations`) extracts all requested cells across all consumers into device memory, followed by one batched device-to-host transfer per step (Fig. 4B). Downstream consumers then slice their assigned ranges directly from the host buffer by zero-copy array views (Fig. 4C). This is the one place a small device-to-host transfer is intrinsic, and slice batching is what keeps it cheap. On the CPU backend the same interface is a trivial array view. Registration order is fixed and deterministic, so the sampled series are reproducible across backends (Sect. 3.9). One caveat: on Windows, which lacks `fork()`, the inter-process handoff still requires the memcpy, giving a small transient dip in GPU utilization at each write interval.

Point-gauge output follows the same design but is trivial by comparison: a `StageRecorder` appends one crash-safe CSV row per record, keyed on gauge ID for column stability.

3.11 Scripting API and configuration

WarpFlow is built to be driven, not only operated. Model construction, execution (start/stop/step), boundary and source updates, sampling and output are exposed as a Python API. Geospatial input is handled through the standard geodata stack with on-the-fly resampling and unit detection, and LISFLOOD-FP file formats (outside of `.bci` and `.bdy`) are read natively, so WarpFlow can act as a drop-in replacement in many ACC-style workflows. Configuration is declarative TOML, loaded through a strict validate-cast-execute pipeline. In our operational Urban Digital Twin this API is what lets models be generated ad hoc from live municipal data and run unattended. Here it is simply the interface of the package.

4 Extensibility: the plugin architecture

4.1 The plugin system

Every add-on, including infiltration, diagnostics, water accounting, performance measurement and, beyond this paper, sediment and drainage coupling, is a self-contained `Plugin` subclass. This yields three properties useful to a reviewer: the core is verifiable in isolation, user physics is composable and reorderable, and the three backends are transparent to plugin authors. Plugins are registered through the `[plugins]` TOML section and sorted by an ascending `priority` to give a deterministic init and hook-execution order.

Plugins attach at phase-gated lifecycle hooks whose correspondence to the water-column state h is fixed and observable (Fig. 5): before and after the boundary bracket, after the core flux update, after grid sources (rain + infiltration), after point sources (gullies + culverts), and at output/finalization. A second mechanism, the kernel override registry, is what makes the system able to change the physics rather than only observe it. `register_kernel_overrides(sim)` returns launcher callables keyed by core kernel name, which are merged into the active dispatch table, so a plugin can replace a computational stage without editing the core. Because the core dispatches kernels by name, the override inherits the same full/active-tile work

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

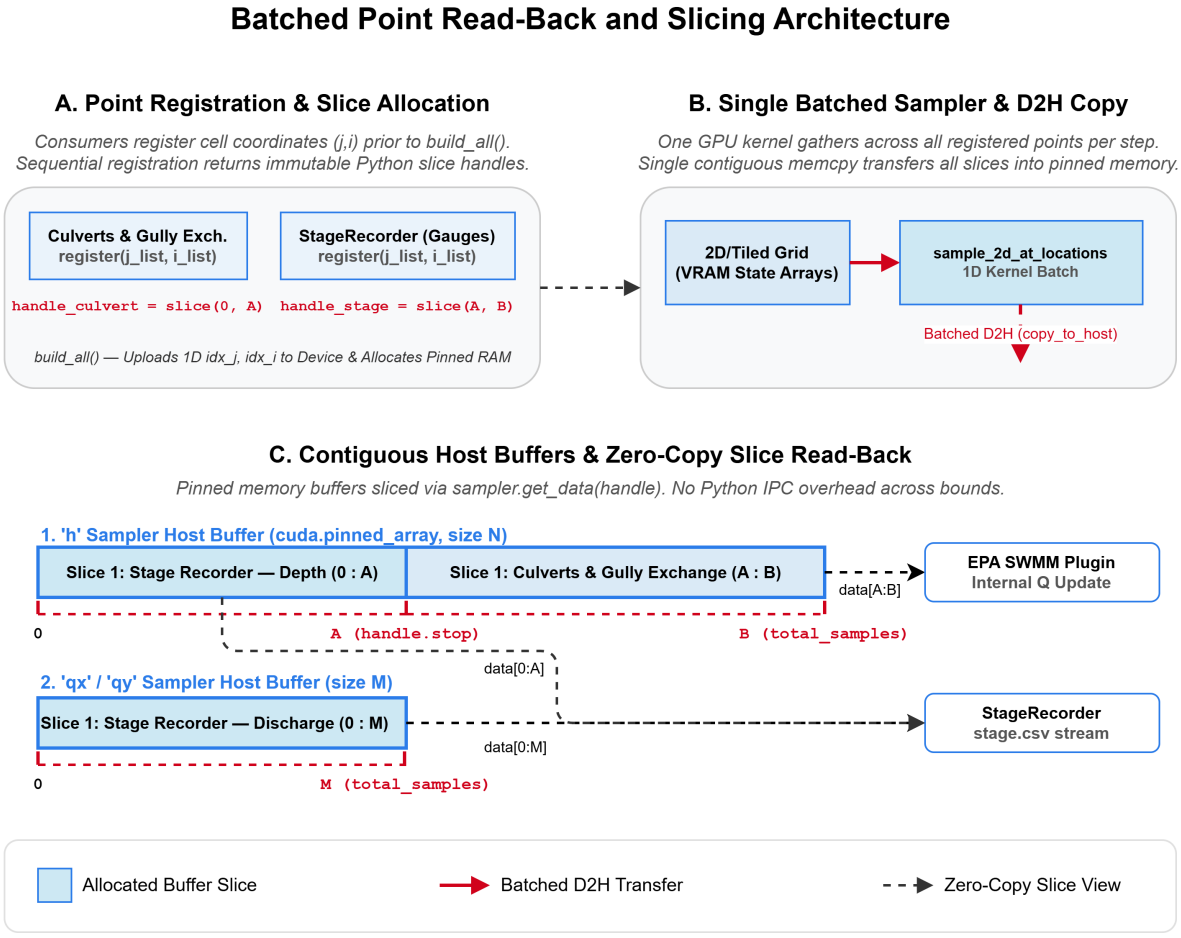


Figure 4. Batched point read-back and slice allocation architecture. *Lifecycle & Registration*: (a) Consumers register cell coordinates prior to initialization and receive deterministic slice handles before `build_all()` allocates pinned host memory. *Kernel Execution*: (b) A single 1D GPU kernel gathers all registered point samples per step, followed by one batched device-to-host memory transfer into pinned RAM. *Sliced Read-Back*: (c) Sampled values are laid out contiguously in host buffers, allowing downstream consumers (e.g., culverts, stage hydrograph recorders) to access their assigned data views by zero-copy slice indexing without inter-process overhead.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

dispatch and resolves the same three-way backend dispatch, reusing the tiled bit-addressing (Sect. 3.6) and LUT-compressed fields (Sect. 3.3). Two patterns follow: pure observers, which only read state, and kernel overrides, which substitute a stage. This user-extensible, plugin-based source-term design is a direct benefit of a Python-native architecture. We ship four plugins with the release described here, and all are used in the remainder of the work.

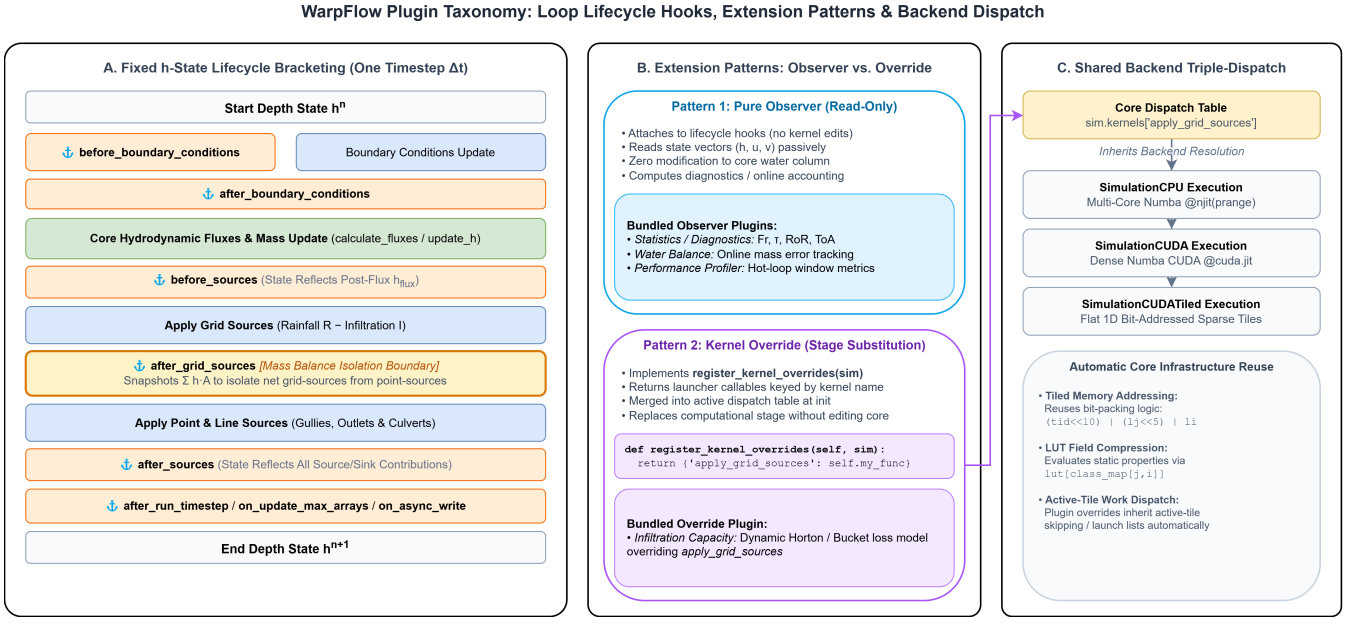


Figure 5. The architecture of WarpFlow's plugin system. *Time step integration* (this section): (a) phase-gated hooks bracket the h-state across one time step to isolate distinct flux contributions; (b) extension patterns allow plugins to either passively observe state or actively substitute core kernels. *Execution dispatch* (Sect. 3): (c) overridden kernels transparently inherit the core's triple-backend resolution and memory infrastructure.

4.2 Infiltration capacity (kernel override)

This plugin overrides the fused `apply_grid_sources` stage (Sect. 3.4) and replaces the constant abstraction of Sect. 2.9 with either a simple storage bucket or a Horton exponential-decay infiltration model with dry-period recovery (Horton, 1941). It supplements the core infiltration data arrays with additional capacity or saturation arrays. Recharge is dispatched by parameter type (disabled/uniform/raster/per-class LUT).

4.3 Statistics and hydraulic diagnostics (observer)

An observer derives, from the primitive state, the cell-centered velocity and magnitude, unit discharge $q = |\mathbf{V}|h$, Froude number $\mathrm{Fr} = |\mathbf{V}|/\sqrt{gh}$, and bed shear $\tau = \rho g n^2 |\mathbf{V}|^2 / h^{1/3}$ (using a `cbrtf` on CUDA and the LUT roughness). It also derives the temporal hazard fields rate-of-rise, time-of-arrival (with a $0.5\,a_t$ hysteresis on the arrival threshold to prevent wet/dry oscilla-

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

tion), flood duration and integrated flow direction. Like all grid-wide passes it sweeps the full terrain set at the output stride (Sect. 3.4), and it offers three memory paths, including a mapped-memory option that trades VRAM for PCIe latency when memory is tight.

4.4 Water balance (observer)

A pure observer closes a global mass budget online by bracketing h between the lifecycle hooks, on a dedicated stream, with all host-side accumulation in double precision. Each term is measured as the change in $\sum h\,A$ across its hook bracket rather than from the mechanism that caused it, so the observer composes with any physics plugin that modifies h between snapshots. The budget is

$$\text{mass_error} = \underbrace{\left(V_{\text{initial}} + V_{\text{rain}} + V_{\text{source h}}\right)}_{\text{inputs}} - \underbrace{\left(V_{\text{infil}} + V_{\text{sources}} + V_{\text{drain}}\right)}_{\text{outputs}} - \underbrace{\left(V_{\text{current}} + V_{\text{transit}}\right)}_{\text{stored}}. \tag{32}$$

Here $V_{\text{current}} = \sum h\,A$ is the ponded volume, V_{drain} the integrated flux over terrain–drain faces, and V_{transit} the water currently held inside the culvert travel-time queues (Sect. 2.9). Transit water is in neither the ponded depth nor the outlet cell until released, so it is carried on the stored side beside V_{current}. This lets the budget close at every step rather than only once the barrels drain at end of run. Two terms are signed, so the side on which they appear in Eq. (32) is nominal. $V_{\text{source h}}$, the prescribed-water-level (Dirichlet) contribution, may be positive or negative, because imposing a level can raise or lower h. V_{sources} is the net point-source exchange, so a discharge source (`QFIX`/`QVAR`) that adds water enters it negatively, acting as an input, while gully/orifice extraction (Eq. (28)) enters positively as an output.

By design Eq. (32) does not cancel to zero: the residual equals the positivity floor's mass creation, $\text{mass_error} = -V_{\text{clamp bias}}$ (Sect. 2.6), making the plugin a direct probe of the only conservation defect in the default configuration. Each tracked term launches its reductions only when the corresponding feature is present: stage tracking only when prescribed-level sources exist, the transit snapshot only when culverts exist. Inter-plugin communication is natively supported, but the water-balance observer accounts only for the core's effect on h. It does not track any other plugin's internal state (e.g. the infiltration-capacity bucket). The mass-conservation test of Sect. 5.4 therefore uses the core infiltration.

4.5 Performance profiler (observer)

The fourth bundled plugin is a pure observer that instruments the runs of Sect. 6. It overrides no kernel and self-reports its own overhead, so its measurements are effectively internally free. The authoritative timing is the end-to-end wall clock, and the plugin deliberately does not force a per-window device synchronization, so it does not distort that number. Instead it records one lightweight CUDA event per step and drains it once per window, yielding a pure device time whose ratio to wall time, $\text{gpu_util} = t_{\text{device}}/t_{\text{wall}}$, is a direct launch-bound-versus-compute-bound indicator (excluding host launch jitter and the CFL side-stream overlap by construction). It aggregates per window rather than per step, because instantaneous throughput is dominated by output stalls and asynchronous launch and queue time. Each window emits one CSV row: iterations per second, cell-updates per second $\Sigma(\text{active cells})/t_{\text{wall}}$, the simulation phase (rain vs. dry-off), the active-cell and active-tile

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

work fractions, the current core mode and adaptive `dt_stride`, and the window's Δt, maximum depth and maximum speed. Host per-step percentiles are recorded only on the CPU backend, where loop-to-loop host time is the true iteration time.

The plugin also quantifies the two adaptivity costs directly by bucketing host step-time. Recompute versus non-recompute steps expose the CFL-stride reduction tax (Sect. 3.5), while per-core-mode buckets plus a switch counter expose the active-tile bookkeeping tax and confirm that mode switches are infrequent (Sect. 3.4). Sweeping `dt_stride` or `active_mode` with the other pinned, and emitting one mode/stride-tagged CSV plus a one-line matrix summary per run, is what produces the adaptivity-cost figures of Sect. 6.3 and the throughput figures of Sect. 6.2.

4.6 Further plugins

The same hooks and triple-dispatch already carry a further suite: a bidirectionally coupled 1D Saint-Venant/EPA-SWMM drainage network, depth-averaged advection–diffusion transport, bedload morphodynamics via an Exner update, and nature-based-solution elements burned into the terrain, roughness and infiltration fields, implemented on all three backends without modifying the flux kernels. These are deliberately held back for a follow-up publication, both to keep the present contribution focused on the solver core and because each warrants its own validation. Their existence is the clearest evidence that the plugin boundary is drawn in the right place.

5 Verification and validation

We separate verification from validation. Verification tests whether the implemented method solves the discrete balances and equations it claims to solve. Validation tests agreement with independent observations or established benchmark data. We also report an inter-model comparability exercise against LISFLOOD-FP ACC GPU. This is not strict validation of WarpFlow's native configuration, because WarpFlow retains advective momentum whereas ACC is a local-inertial model. Performance and scalability are treated separately in Sect. 6.

5.1 Test-suite configuration and reproducibility

The verification suite lives under `/tests`. Every case is run and evaluated through the reproducible harness `verification.py`, with case generation in `verification_generate.py` and execution and metric collation in `verification_run.py` and `verification_evaluate.py`. Each case has a stable identifier (`VV-01`, `VV-02`, ...), a `case.json` metadata file, a TOML configuration, and case-local data notes. Only the analytical and synthetic cases generate their own input data. The applied and inter-model cases, UK Environment Agency, Upper Lee, Malpasset and Dresden, are preconfigured and read shared input data from directories under `data/`, emitting placeholder markers when the externally supplied files are absent.

All measurements in this section were conducted on one workstation: a Windows 11 host, Ubuntu 26.04 LTS under WSL, an AMD Ryzen 9 3900X CPU with 24 hardware threads, 32 GiB system RAM, and an NVIDIA GeForce RTX 4080 with 16 GiB GDDR6X VRAM and 64 MiB L2 cache. CUDA execution used Python 3.14 in the provided `warpflow` conda environment. All speed, throughput and turnaround claims in this section are therefore specific to this single hardware and software configu-

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

ration. All tests were also cross-checked on an older RTX 2080 Ti GPU in the same host; as expected, performance gains from the tiled memory layout were smaller on real-world geometry due to its significantly lower L2 cache (5.5 MiB).

Single precision is treated as a design constraint. Analytical checks use `float32`-safe grids and refinement ratio $r = 2$. For analytically checked cases `compute_max_steps = 1`, so `depth_max` is the exact running maximum of the simulated sequence. For applied and performance-oriented cases, where `compute_max_steps = -1` may be used, maxima are updated at output cadence. We state this where it affects a comparison. Unless noted otherwise, analytical tests use $g = 9.806\,65\,\mathrm{m/s^2}$ and conservative time step settings, while applied cases use operational solver settings.

The resulting verification, validation and inter-model suite is summarized in Table 2.

5.2 Analytical and controlled verification

Dam breaks: Ritter and Stoker. One-dimensional dam-break propagation over dry (`VV-01 ritter_dry`) and wet (`VV-02 stoker_wet`) beds (Ritter, 1892; Stoker, 1957) was evaluated on a $L = 10\,\mathrm{m}$ domain across five spatial resolutions ($n \in \{50, 100, 200, 400, 800\}$, corresponding to $\Delta x \in [0.20, 0.0125]\,\mathrm{m}$). Initial conditions were $h_L = 0.005\,\mathrm{m}$ and $h_R = 0$ (Ritter dry), and $h_R = 0.001\,\mathrm{m}$ (Stoker wet). Evaluation logs confirm that neither enabling the FCT flux limiter (`drycheck`) nor switching to double precision (`float64`) altered the numerical behavior or error trajectories relative to single precision (`float32`). These variants are therefore omitted from the plot for clarity.

Convergence metrics and spatial profiles at $t = 6\,\mathrm{s}$ are presented in Table 3, Fig. 6, and Fig. 7. Water depth error $L_2(h)$ exhibits monotonic convergence across all grid levels: reducing from 2.381×10^{-3} to $9.995 \times 10^{-4}\,\mathrm{m^{3/2}}$ for Ritter dry (p_h decreases from 0.421 to 0.194), and from 2.358×10^{-3} to $9.275 \times 10^{-4}\,\mathrm{m^{3/2}}$ for Stoker wet (p_h decreases from 0.397 to 0.212). The sub-linear observed order $p_h = \log_2(L_2^n / L_2^{2n})$ is dictated by first-order donor-cell numerical dissipation at non-smooth wave fronts.

Conversely, velocity error $L_2(u)$ stalls near $p_u \approx 0$, remaining plateaued at $L_2(u) \approx 0.463\,\mathrm{m^{3/2}/s}$ (Ritter dry) and $L_2(u) \approx 0.058\,\mathrm{m^{3/2}/s}$ (Stoker wet). As detailed in Fig. 6, this error stagnation arises from localized front mechanics rather than interior flow inaccuracies:

1. **Ritter dry-front formula mismatch:** The solver accurately tracks the analytical parabolic depth profile and the rising velocity branch across the interior rarefaction. However, the first-order donor-cell flux and semi-implicit velocity update do not reproduce the exact Ritter dry-front characteristic relation all the way to $h \to 0$. Instead, the numerical solution forms a finite-depth advancing front near $x \approx 6.0\,\mathrm{m}$, while the analytical thin-film branch continues to $x \approx 7.7\,\mathrm{m}$ with velocities tending to $2\sqrt{gh_L} \approx 0.44\,\mathrm{m/s}$. This is not a wet-depth-threshold artifact: the dam-break tests use $h_{\mathrm{wet}} = 1 \times 10^{-8}\,\mathrm{m}$ and the same value for the minimum dam-break flow depth. Because the analytical solution contains a finite spatial interval that is not represented by the discrete front formula, the integrated velocity error $L_2(u)$ is bounded by a spatial constant independent of Δx.

2. **Stoker shock displacement:** The wet-bed profile captures the expansion fan but exhibits a slight spatial shock lag and an intermediate velocity plateau deficit. This yields a localized $O(1)$ velocity discrepancy over the shock jump. Refining

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

the grid sharpens the transition, but it does not remove the finite shock-position and plateau-speed mismatch produced by the first-order flux and momentum update.

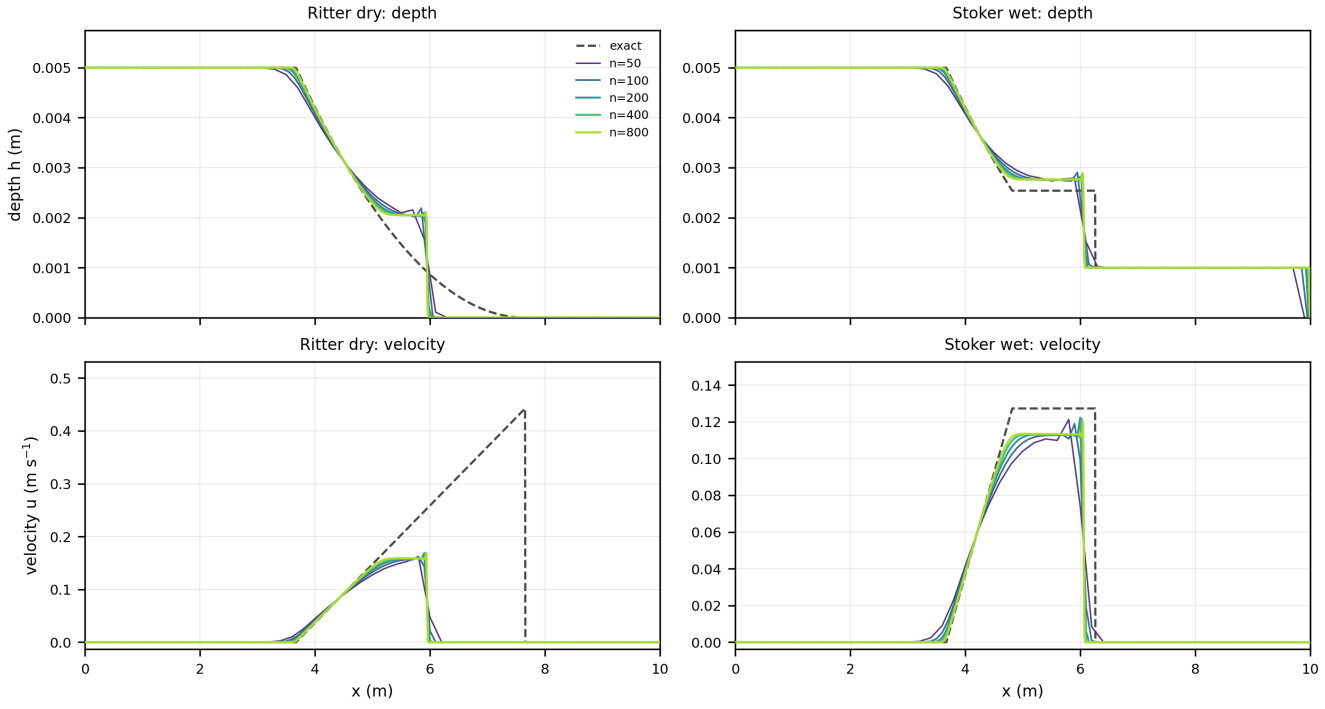


Figure 6. Spatial profile comparisons at $n = 800$ ($t = 6\,\mathrm{s}$) against exact analytical solutions. Left: Ritter dry bed illustrating rarefaction fan agreement and the finite-depth numerical dry front near $x \approx 6.0\,\mathrm{m}$; Right: Stoker wet bed showing shock-front lag at $x \approx 6.0\,\mathrm{m}$.

Lake at rest. `VV-03 lake_at_rest_immersed` and `VV-04 lake_at_rest_emerged` verify the C-property of the hydrostatic reconstruction. Both cases use a 50×50 grid, a random bed and a constant initial free surface $\eta = 0.5\,\mathrm{m}$ over a $5\,\mathrm{s}$ run. `VV-03` is fully immersed, while `VV-04` contains an emergent bump. The checker evaluates the maximum wet-cell velocity and the maximum wet-cell free-surface deviation over the full run. The all-cell velocity maximum is kept as a diagnostic in the emergent case, but the pass gate uses wet cells only.

Both cases preserve the free surface at the single-precision floor (Table 4). The immersed case gives $\max|v|_{\mathrm{wet}} = 4.236 \times 10^{-6}\,\mathrm{m/s}$ and $\max|\Delta\eta| = 1.118 \times 10^{-6}\,\mathrm{m}$. The emergent case gives $\max|v|_{\mathrm{wet}} = 1.122 \times 10^{-5}\,\mathrm{m/s}$ and $\max|\Delta\eta| = 7.246 \times 10^{-7}\,\mathrm{m}$, passing the wet/dry-interface velocity tolerance while retaining free-surface perturbations at the `float32` floor.

Synthetic smooth-grid GCI. Before applying grid analysis to real urban terrain, `VV-05 synthetic_smooth_gci` uses a generated smooth case that satisfies the assumptions of a scalar Richardson/Roache GCI study (Roache, 1994; Celik et al., 2008). The case is fully wet, uses a flat bed, and starts from a smooth Gaussian depth perturbation on a $256\,\mathrm{m} \times 256\,\mathrm{m}$ square domain. The three-grid sequence is $4\,\mathrm{m}$, $2\,\mathrm{m}$, and $1\,\mathrm{m}$, with fixed time steps refined in the same ratio ($0.1\,\mathrm{s}$, $0.05\,\mathrm{s}$ and $0.025\,\mathrm{s}$).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

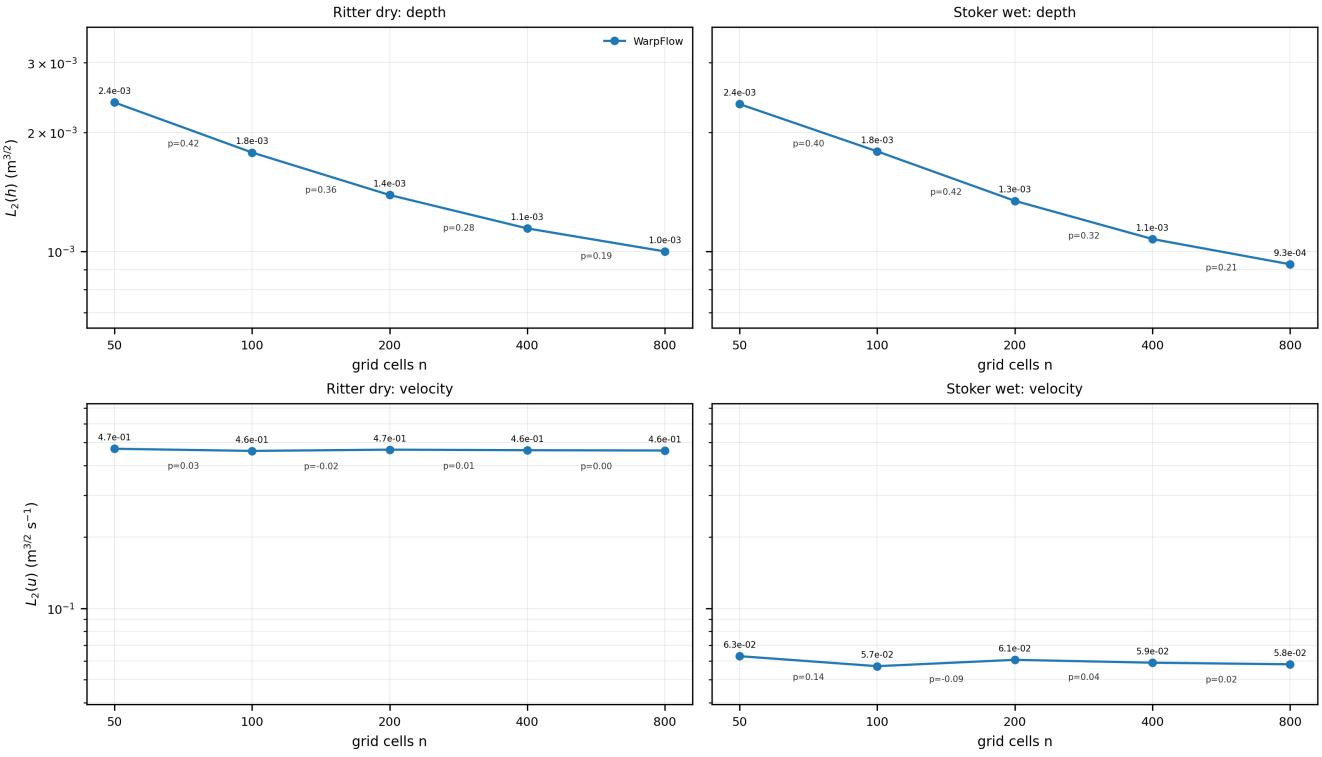


Figure 7. L_2 error norm convergence against spatial resolution $n \in \{50, \ldots, 800\}$. Depth error $L_2(h)$ converges sub-linearly ($p_h \in [0.19, 0.42]$), whereas velocity error $L_2(u)$ plateaus due to dry-front formula mismatch in the Ritter case and shock-front displacement in the Stoker case.

The GCI is evaluated on smooth integral functionals of the final instantaneous depth field, not on `depth_max` or thresholded flood extent.

The controlled sequence gives monotone behavior for the primary smooth QoIs (Table 5 and Fig. 8). Fine-grid GCI is $0.002\,\%$ for the Gaussian-weighted mean depth, $0.283\,\%$ for the perturbation-energy proxy, $0.141\,\%$ for the corresponding L_2 perturbation norm, and $0.018\,\%$ for the central-patch mean depth. The full-field Cauchy check also shrinks: final-depth RMS decreases from $2.1806 \times 10^{-5}\,\mathrm{m}$ to $1.1477 \times 10^{-5}\,\mathrm{m}$. Global volume and mean depth remain conserved across the hierarchy, confirming that the sequence is not contaminated by a domain-area or boundary-cell artifact. The difference panels in Fig. 8 show smooth, structured coarse–medium and medium–fine residuals at the expected $1 \times 10^{-5}\,\mathrm{m}$ scale.

5.3 CPU–GPU parity and reproducibility

`VV-06 parity_dambreak` quantifies backend reproducibility on a fully wet radial dam break. The case uses an 80×80 grid over a $40\,\mathrm{m} \times 40\,\mathrm{m}$ square domain with a central water column. It avoids a dry front so that residual differences between

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

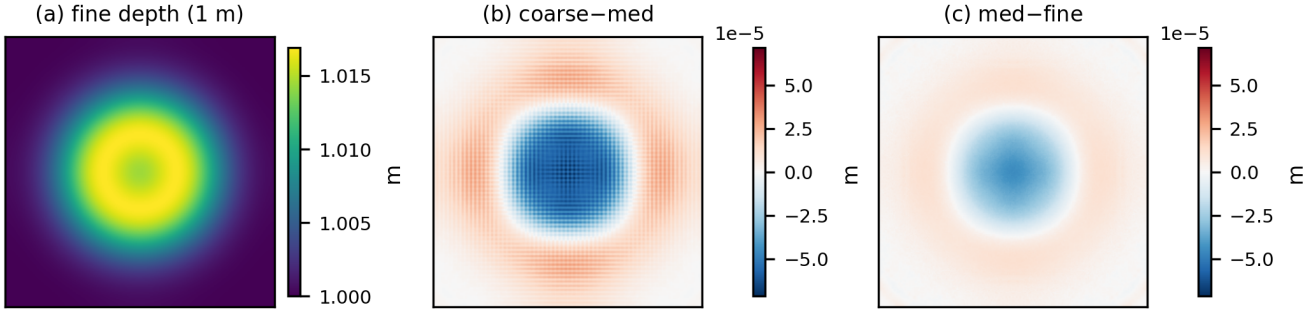


Figure 8. Synthetic smooth-grid GCI on the generated all-wet Gaussian-depth perturbation. Panel (a) shows the fine-grid final depth at 1 m resolution. Panels (b) and (c) show the signed coarse–medium and medium–fine final-depth differences resampled to the fine grid.

CPU, dense CUDA and tiled CUDA are attributable to single-precision arithmetic ordering, compiler lowering and memory layout, rather than wet/dry activation ambiguity. The comparison is made on the final `depth_max` field.

The time step is fixed, not adaptively reduced: `dt_limiter = "f"` with $\Delta t = 0.1\,\mathrm{s}$. This removes the adaptive-CFL side stream from the comparison. A tight output-time snapping tolerance, `dt_error`$=1\times10^{-8}$, ensures that all backends advance through the same sequence of physical times.

The three backends agree at single-precision round-off scale (Table 6 and Fig. 9). CPU and dense CUDA differ by at most $1.490\times10^{-7}\,\mathrm{m}$ in `depth_max`, with $L_2 = 2.095\times10^{-8}\,\mathrm{m}$. Dense CUDA and tiled CUDA agree similarly, with $\max|\Delta| = 8.941\times10^{-8}\,\mathrm{m}$ and $L_2 = 1.396\times10^{-8}\,\mathrm{m}$. The residuals are confined to the moving front and are symmetric at round-off level, supporting the Jacobi double-buffered velocity update and confirming that tiled storage does not alter the numerical solution.

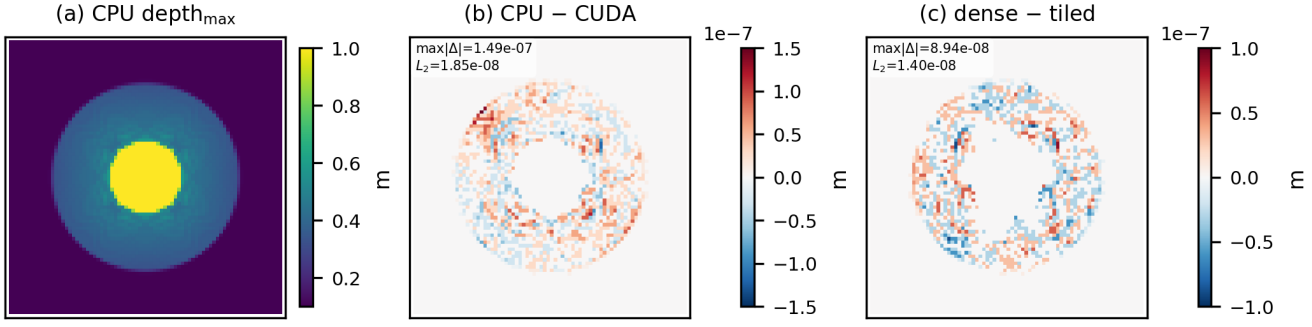


Figure 9. Backend parity on the fully wet radial dam break. Panel (a) shows the CPU `depth_max` reference. Panel (b) shows the signed CPU–dense-CUDA residual, and panel (c) the signed dense-CUDA–tiled-CUDA residual. Difference fields are plotted with symmetric color scales of order $1\times10^{-7}\,\mathrm{m}$.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

5.4 Mass conservation and source-term accounting

The water-balance tests verify Eq. (32) independently of the analytical flow tests. All four cases use a gently sloped 200×200 grid with rainfall, infiltration and drain-boundary outflow. The online water-balance plugin observes the actual numerical state: terrain storage is obtained from reductions of $\sum hA$, source stages are bracketed by before/after snapshots, rainfall is accumulated from the prescribed rate, drain loss is integrated from computed drain-face fluxes, and culvert in-transit storage is read from the live FIFO buffers. The budget therefore measures what the kernels did to the water column and auxiliary culvert storage, not merely what a source routine intended to apply.

`VV-07 mass_dryon` exercises the normal dry-check path. `VV-08 mass_dryoff` disables that path and verifies the default positivity-floor behavior: the remaining residual is the bounded clamp bias at draining fronts. `VV-09 mass_hvar` activates an interior prescribed-level `HVAR` line and checks the signed $V_{\text{source h}}$ term. `VV-10 mass_sources_culverts` combines ordinary point sources/sinks, an `HVAR` line and a culvert. It is checked while water is present in the culvert queue, so $V_{\text{transit}} > 0$ must be accounted for.

All four cases close far below the configured relative tolerance of 1×10^{-4} (Table 7 and Fig. 10). The two plain rainfall/infiltration/drainage cases have a signed residual of $-1.992 \times 10^{-3}\,\mathrm{m}^3$, corresponding to a relative closure error of 2.0×10^{-7}. The `HVAR` case activates $V_{\text{source h}} = 1.105 \times 10^{3}\,\mathrm{m}^3$ and reports a residual of $1.100 \times 10^{-2}\,\mathrm{m}^3$, corresponding to a relative closure error of 1.2×10^{-6}. The combined source/culvert case activates all non-rainfall pathways, including $V_{\text{sources}} = -2.390 \times 10^{2}\,\mathrm{m}^3$, $V_{\text{source h}} = 1.106 \times 10^{3}\,\mathrm{m}^3$ and $V_{\text{transit}} = 5.356 \times 10^{-1}\,\mathrm{m}^3$, while retaining a relative closure error of 1.3×10^{-6}. Figure 10 shows the cumulative budget terms and the residual on a secondary axis, making the separation between kilocubic-metre storage/source terms and sub-cubic-metre residuals explicit.

5.5 Self-referenced grid consistency on Dresden-Striesen

The public real-terrain consistency case, `VV-11 dresden_striesen`, complements the controlled smooth-grid GCI of Table 5. It uses a $2\,\mathrm{km} \times 2\,\mathrm{km}$ Dresden-Striesen district subset. This is not a formal pointwise convergence proof: real urban rasters contain curbs, walls, depressions, road crowns and narrow conveyance features whose representation changes with resolution. The purpose is therefore to quantify self-consistency and grid sensitivity on a public urban surface.

The public test case consumes preprocessed raster inputs at $4\,\mathrm{m}$, $2\,\mathrm{m}$, and $1\,\mathrm{m}$. For transparency, the supplied rasters were derived from main $1\,\mathrm{m}$ DEM, Manning and infiltration raster using a common $4\,\mathrm{m}$ analysis footprint and active-cell block aggregation for the coarser grids. A unified active mask gives the same active computational footprint at $4\,\mathrm{m}$, $2\,\mathrm{m}$, and $1\,\mathrm{m}$ ($A_{\text{active}} = 2\,955\,920\,\mathrm{m}^2$, 73.90 %). The case uses a 100-year, $1\,\mathrm{h}$ Euler-II design storm (DWA, 2025), with the rainfall depth taken from the KOSTRA-DWD-2020 return-period statistics for the site (Junghänel et al., 2023), followed by $1\,\mathrm{h}$ runoff. Gullies, culverts and restricted drainage-network data are excluded.

We report scalar quantities of interest above the flood threshold $h > 0.1\,\mathrm{m}$ using a three-grid Richardson/Roache GCI ($r = 2$, $F_s = 1.25$) (Roache, 1994; Celik et al., 2008), and a full-field Cauchy diagnostic on `depth_max`. The scalar hazard metrics are well behaved (Table 8 and Fig. 11): flooded volume gives a fine-grid GCI of 1.18 %, and flooded area gives 1.87 %.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

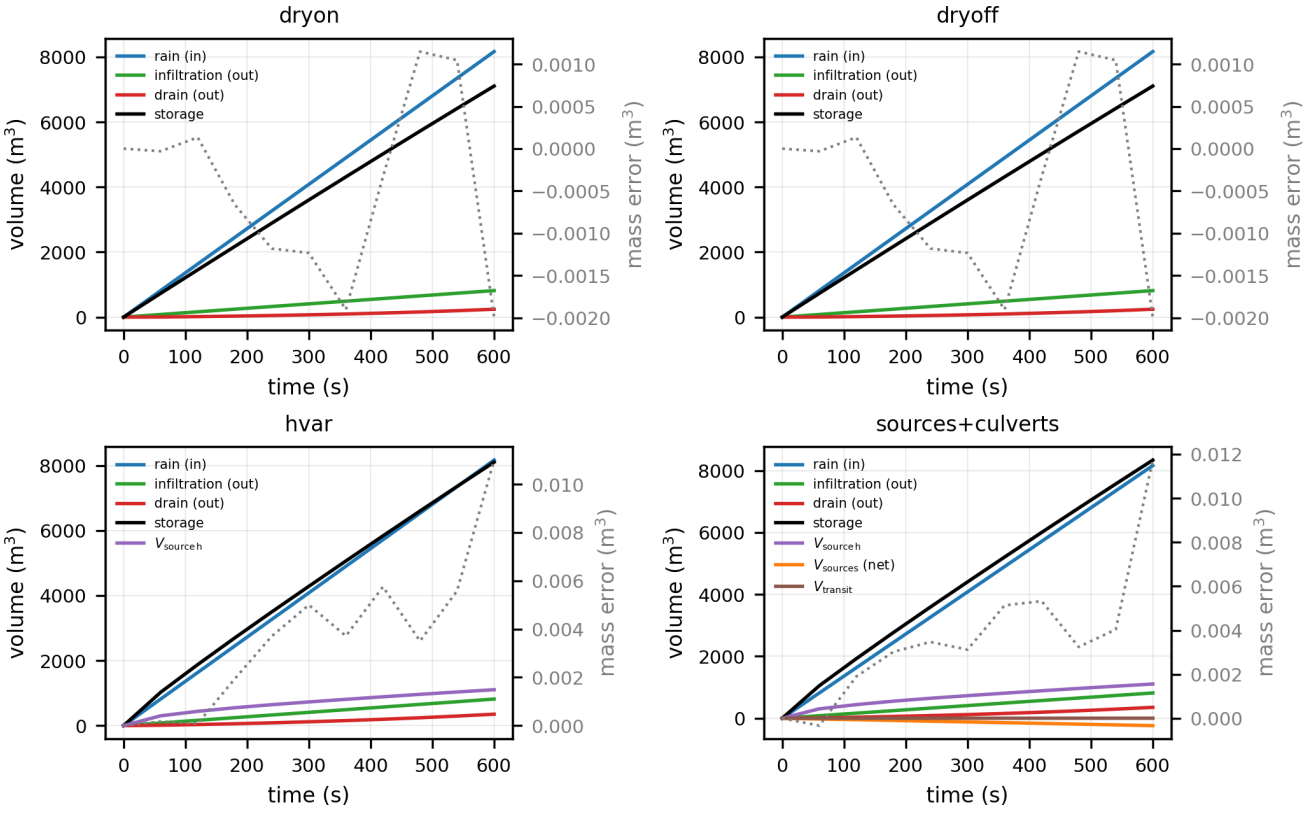


Figure 10. Cumulative online water-budget closure and source-term accounting. The four panels separate the normal dry-check path, the dry-check-disabled clamp-bias path, prescribed-level exchange, and combined source/culvert transit storage. Main cumulative terms are plotted on the left axes; the gray dotted curve shows the residual mass error on the secondary axis.

Peak depth is oscillatory, as expected for an urban maximum controlled by individual pits, curbs or wall-edge cells. The field RMS shrinks only weakly, from $0.0454\,\mathrm{m}$ to $0.0445\,\mathrm{m}$. We therefore treat the field result as a real-terrain sensitivity diagnostic rather than as a failed formal convergence result. Figure 11 shows that the residuals concentrate along urban micro-topographic structures, road edges, depressions and narrow conveyance features, exactly where raster-resolution changes are expected to matter most.

5.6 UK Environment Agency 2D benchmark subset

We use the Environment Agency / Neelz–Pender 2D benchmark suite (Neelz and Pender, 2013) as an external validation reference. From it we retain two cases that each exercise a distinct regime. Test 5 (`VV-12 ea_05_valley_momentum`) is momentum-sensitive valley flow, the regime in which retaining advective momentum is expected to matter. Test 8A (`VV-13`

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

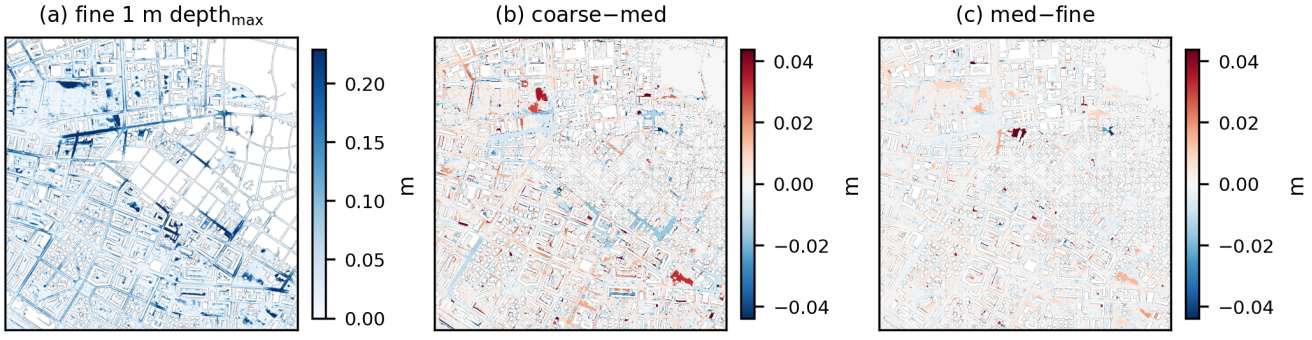


Figure 11. Self-referenced grid consistency on the Dresden-Striesen public sub-domain. The left panel shows the fine-grid 1 m `depth_max` field. The center and right panels show signed coarse–medium and medium–fine differences after resampling to the common comparison grid. Residuals concentrate along urban micro-topographic structures and narrow conveyance features, while scalar flooded-volume and flooded-area QoIs remain well behaved under refinement.

`ea_08a_pluvial`) is rainfall-driven urban pluvial flow, the operational target regime. Test 5 is reconstructed for the WarpFlow suite. Test 8A is converted from the LISFLOOD-FP 8.1 dataset (Sharifian et al., 2023) with `lisflood_fp_to_warpflow.py`.

Test 5: valley momentum. The inflow propagates down a sinuous valley (Fig. 12). WarpFlow gives a monotone wave arrival from 27 min at gauge g0 to 150 min at g6, peak channel depths up to ≈ 7 m, and peak velocities up to ≈ 5 m/s. The gauge water-surface levels rise sharply on wave arrival and settle to stable backwater stages, and the velocity hydrographs peak near 2.8 m/s (g1) before receding smoothly. The clean arrival ordering and bounded velocities are the expected signature of a momentum-resolving scheme on this benchmark.

Test 8A: pluvial and resolution consistency. The Test 8A case is run at 0.5 m and 2 m (Fig. 13). It is a report-only benchmark in the present suite: the checker samples the nine supplied gauge locations and summarizes the model hydrographs, but no independent EA observation hydrographs are configured for scoring. The forcing combines a 400 mm/h rain-on-grid input of 4 min duration with a sewer overflow point source, so the gauges respond from the beginning of the event.

The largest gauge response is ≈ 0.7 m at g2, while g0 peaks near 0.56 m. The 0.5 m and 2 m hydrographs reproduce the same event timing and recession pattern, with resolution differences largest at gauges controlled by curbs, road crowns and local depressions. The signed 2 m minus 0.5 m peak-depth field is mostly within ±0.10 m and is concentrated along urban micro-topographic features. This resolution sensitivity is consistent with the self-referenced Dresden-Striesen result (Sect. 5.5). Runtime is reported only as context: on the RTX 4080 testbed the 0.5 m run completed in 256.4 s, and the 2 m run in 56.8 s. The 0.5 m case is too small to saturate the GPU, while the local CFL time step drops to ≈ 0.10 s, so its wall time is governed mainly by step count and launch/orchestration cost rather than by large-domain stencil throughput.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

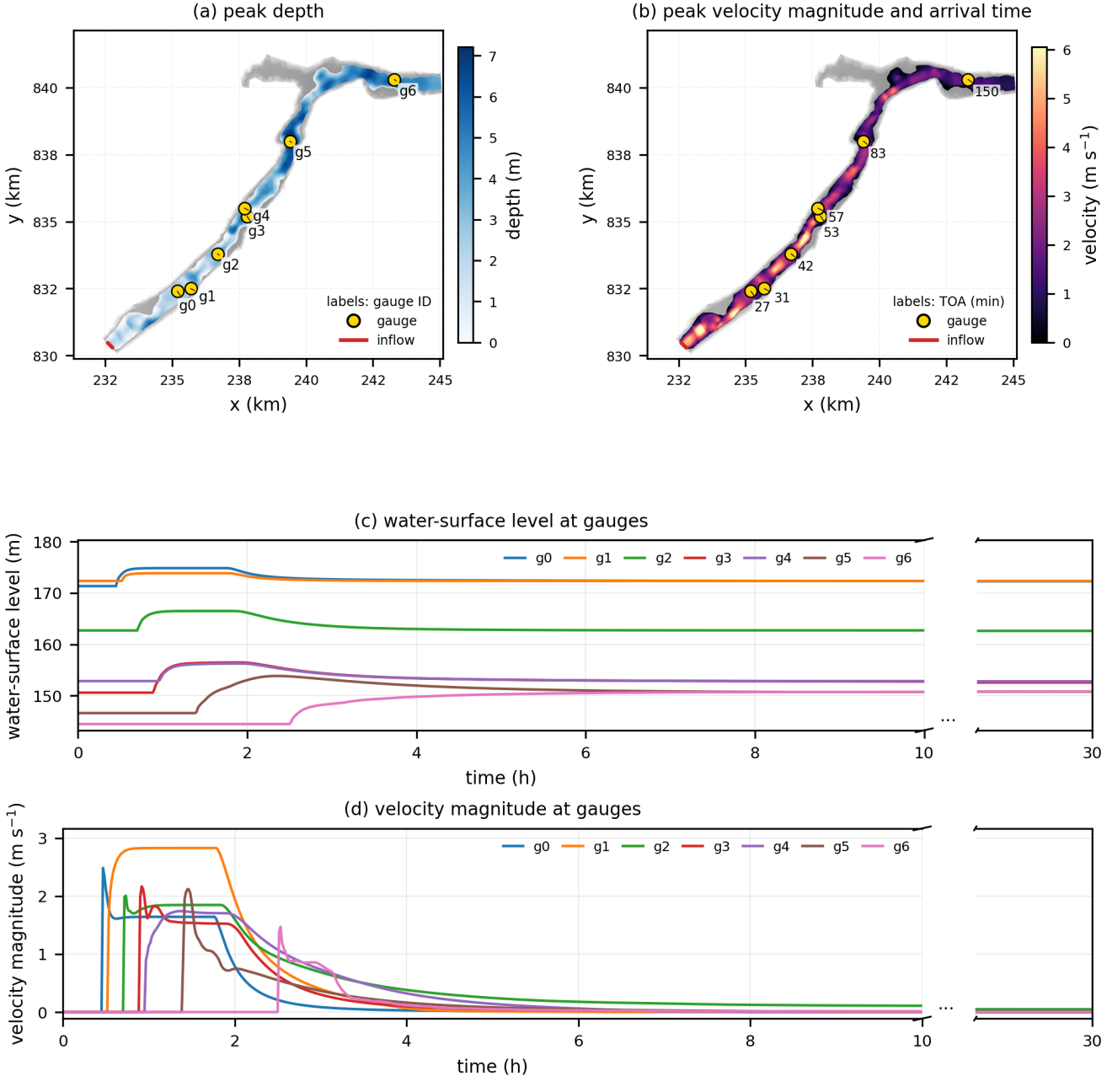


Figure 12. Environment Agency Test 5 (valley momentum), `VV-12`. (a) peak depth and (b) peak velocity magnitude with per-gauge time of arrival (TOA, min); (c) water-surface level and (d) velocity magnitude at gauges g0–g6. Wave arrival is monotone from 27 min (g0) to 150 min (g6).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

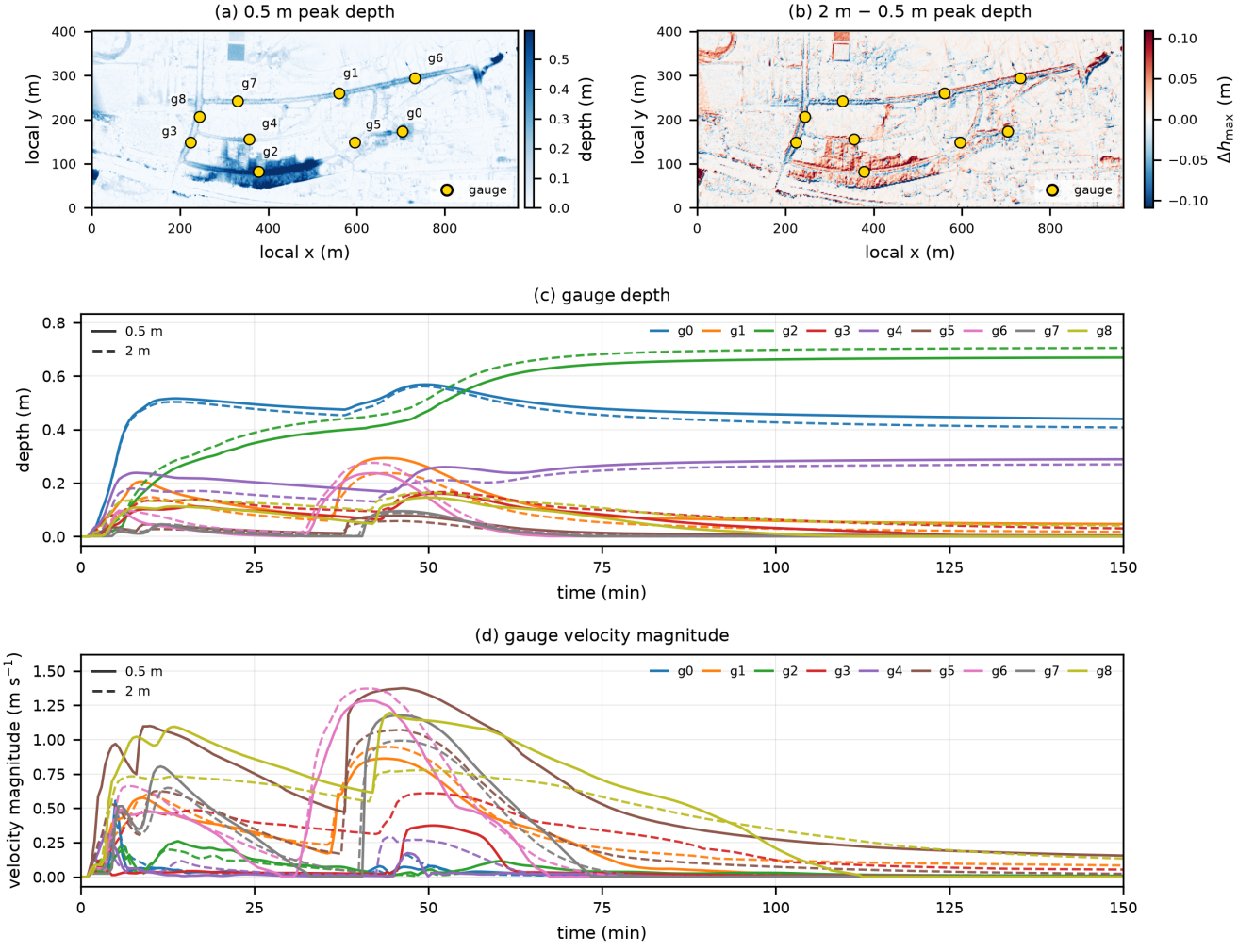


Figure 13. Environment Agency Test 8A (pluvial), VV−13, at 0.5 m and 2 m. (a) 0.5 m peak depth with gauge locations; (b) signed 2 m minus 0.5 m peak-depth difference, concentrated on road, curb and depression-scale features; (c) gauge depth and (d) gauge velocity magnitude at gauges g0–g8, solid 0.5 m, dashed 2 m.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

5.7 Upper Lee inter-model comparability

The Upper Lee case is an inter-model comparability context against the LISFLOOD-FP ACC solver, not a strict validation. ACC is a local-inertial solver recommended for gradually varying, subcritical flooding over sufficiently rough surfaces, whereas WarpFlow retains advective momentum and advances the full-SWE velocity-form balance (Shaw et al., 2021). The two schemes are therefore expected to differ, especially in steep, rapidly varying or locally supercritical reaches. Comparisons use the published LISFLOOD-FP reference dataset (Sharifian et al., 2023). Two native WarpFlow configurations are run: `VV-14 upperlee/wf_compare`, the dense-CUDA baseline (full-domain, `dt_stride=1`, advection enabled), and `VV-15 upperlee/wf_optimized`, tiled-CUDA execution with `active_mode="auto"` and adaptive CFL striding. A third diagnostic variant, `upperlee/wf_optimized_n=0.8xn`, repeats the optimized run after scaling the Manning raster to $0.8 \times n$. The converter `lisflood_fp_to_warpflow.py` translates the existing ACC configuration and input data into drop-in WarpFlow input files.

The published Upper Lee GPU loop times (Sharifian et al., 2023) are 2557 s for uniform ACC (ACC GPU) and 820 s for non-uniform ACC (NU_ACC GPU, $\epsilon = 10^{-3}$). These values were obtained on different hardware and with a different numerical scheme, so direct speed ratios are not meaningful. On the RTX 4080 testbed, WarpFlow completed the dense-CUDA baseline in 514.470 s. The adjusted-roughness optimized tiled-CUDA configuration completed in 361.376 s. Thus, relative to the dense-CUDA baseline, this optimized configuration reduces wall time by approximately 30 %. This is an optimized-configuration comparison rather than a pure tiling speedup: the optimized runs also use adaptive CFL striding. In this case `auto` execution remains in full mode, so the runtime reduction is not an active-tile sparsity result. The domain has only 2 621 409 terrain cells and is too small to represent the large-domain throughput ceiling of Sect. 6.2; we therefore de-emphasize runtime here.

Inter-model field agreement between the published ACC GPU output and WarpFlow is shown in Fig. 14. Against the ACC GPU optimized maximum-depth field, the dense WarpFlow baseline gives RMSE 0.0345 m, bias 0.0029 m and Critical Success Index (CSI; Schaefer, 1990) 0.882. The optimized WarpFlow configuration gives RMSE 0.0184 m, bias 0.0013 m and CSI 0.922. The $0.8 \times n$ variant gives RMSE 0.0187 m, bias 0.0010 m and CSI of 0.917. These domain-wide metrics are descriptive rather than a hydrological ranking criterion, because much of the raster is off-channel upland and does not control the hydraulic comparison.

The informative differences are in the stream network and the gauge hydrographs. At the stage 1 gauge, scaling the WarpFlow Manning coefficient to $0.8 \times n$ aligns the WarpFlow depth hydrograph more closely with the ACC GPU record along the main rising limb and near peak stage (Fig. 14a). The outlet hydrographs show the corresponding roughness sensitivity in discharge (Fig. 14b): the lower-roughness variant produces a sharper, earlier response, while the unscaled configuration is more attenuated. In the maximum-depth residual field (Fig. 14c), WarpFlow is generally shallower than ACC along parts of the channel network. This is consistent with the retained advective momentum holding back and redistributing flow relative to the local-inertial ACC approximation, rather than being a simple roughness error.

The original observed discharge series is not available to us, so we describe the $0.8 \times n$ WarpFlow hydrograph shape in numerical terms rather than scoring it against observations. It rises to a single peak of $\approx 105\,\mathrm{m^3/s}$ near $t = 75\,\mathrm{h}$, then recedes

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

on a smooth limb without secondary oscillation. The published ACC runs (Sharifian et al., 2023) peak lower on the same forcing: $93\,\mathrm{m^3/s}$ for the uniform grid (a premature cutoff) and $63\,\mathrm{m^3/s}$ to $82\,\mathrm{m^3/s}$ under non-uniform adaptivity (attenuation).

Different schemes give different results, so the exercise is comparability, not validation. The main outcome is practical: WarpFlow ingests the ACC input data directly, reproduces the broad maximum-depth field and inundation pattern, and exposes the expected stream-network differences caused by retaining advective momentum. With a simple roughness rescaling, the stage hydrograph at the reported gauge is close to the ACC GPU optimized result, but the result should not be read as a universal Manning correction or as a proof of equivalence between the two schemes.

5.8 Malpasset dam break

The Malpasset dam break is retained as an applied severe-bore benchmark. It is a stress regime for WarpFlow, not a design target. We use it to characterize behavior in steep, rapidly propagating, initially dry flow, rather than to claim this regime as the numerical target.

Two Manning roughness variants are evaluated, `VV-16` ($n = 0.033$) and `VV-17` ($n = 0.025$), at laboratory gauges G6–G14. Maximum water-surface levels are similar for the two values, with slightly smaller errors for the lower-roughness variant: the mean absolute error (MAE) is $2.99\,\mathrm{m}$ (RMSE $3.53\,\mathrm{m}$) for $n = 0.033$ and $2.86\,\mathrm{m}$ (RMSE $3.41\,\mathrm{m}$) for $n = 0.025$. Arrival times are more sensitive to roughness: $n = 0.025$ lowers the arrival-time MAE from $89.12\,\mathrm{s}$ (RMSE $122.03\,\mathrm{s}$) to $37.84\,\mathrm{s}$ (RMSE $51.59\,\mathrm{s}$), indicating better bore-speed agreement for the lower-roughness variant. The 2D inundation field with gauge profiles is shown in Fig. 15. The police high-water marks are used only for a visual comparison of maximum flood extent against the historical observation.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

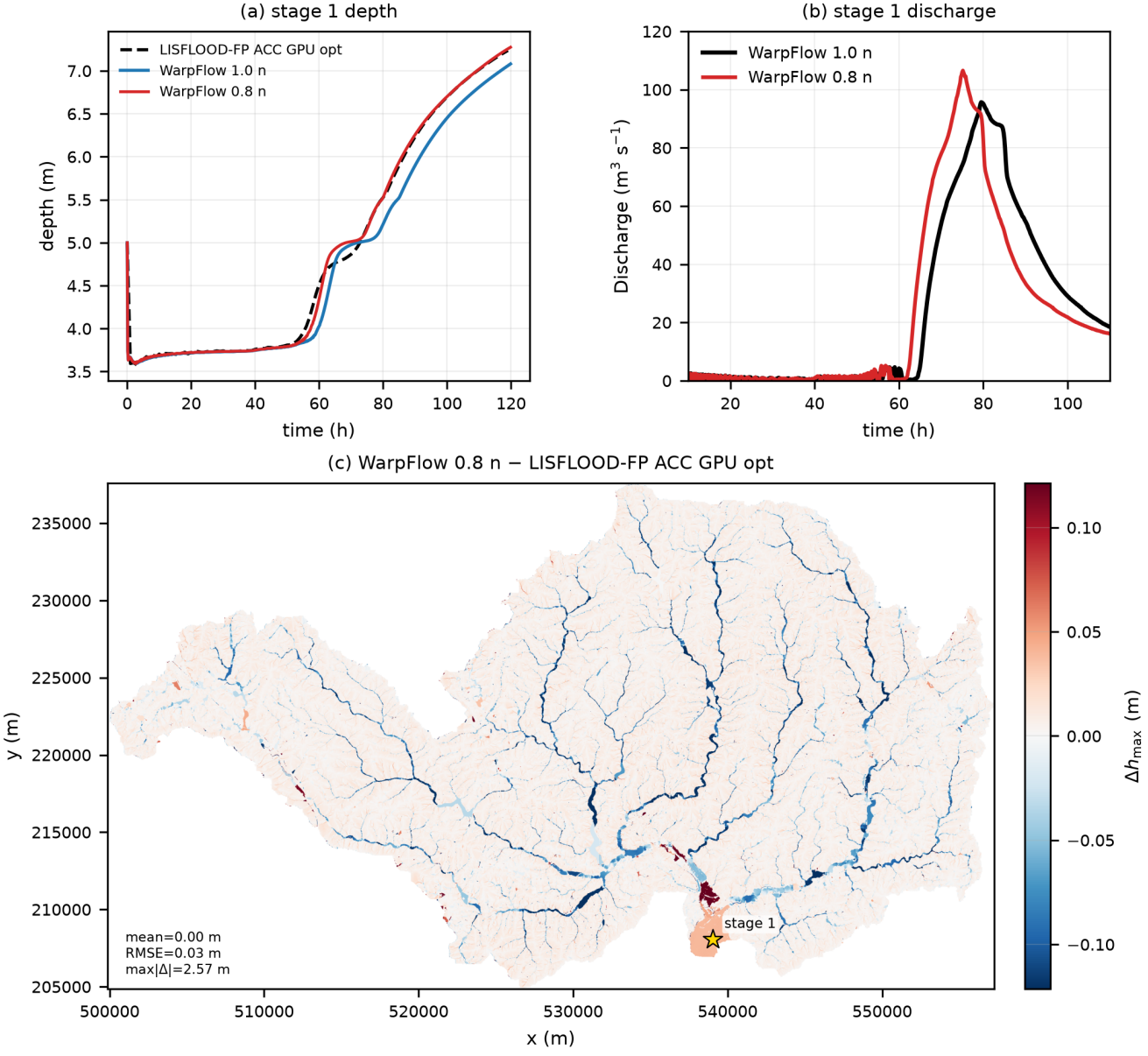


Figure 14. Upper Lee inter-model comparison. Panel (a) presents stage 1 depth hydrographs comparing ACC GPU, WarpFlow ($1.0 \times n$), and the adjusted-roughness variant ($0.8 \times n$). Panel (b) shows discharge-hydrograph sensitivity to Manning roughness. Panel (c) shows maximum-depth residuals ($\Delta h_{\max}$) between WarpFlow ($0.8 \times n$) and the published ACC GPU result, alongside the stage 1 location. Negative residuals mark reaches where WarpFlow is shallower than ACC.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

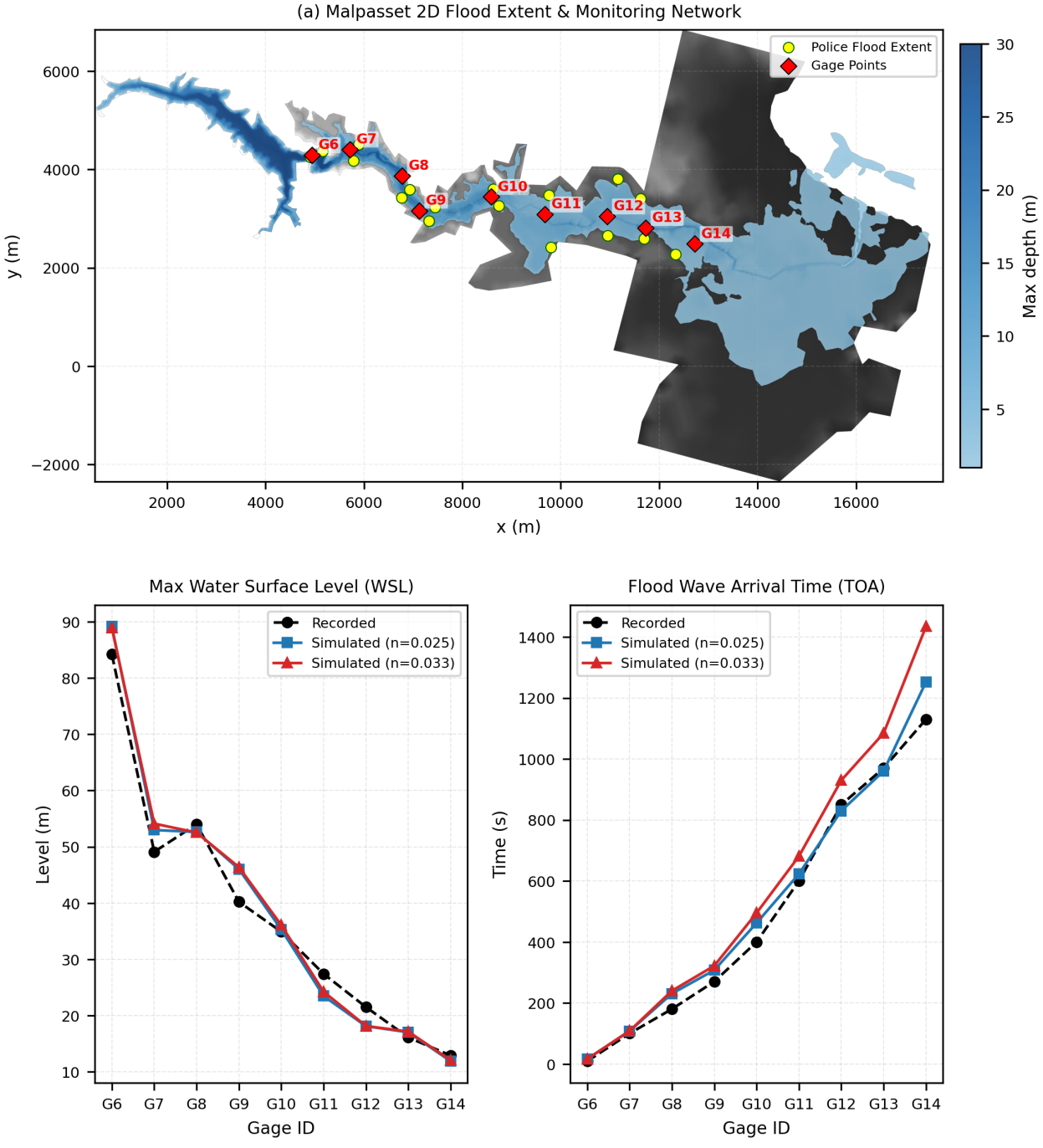


Figure 15. Applied Malpasset dam-break validation. The top panel shows the `depth_max` field overlaid on the terrain raster, with police survey high-water marks (for visual comparison of maximum flood extent) and laboratory gauges G6–G14. The bottom panels show longitudinal gauge comparisons for maximum water-surface level (left) and flood-wave arrival time (right). The lower-roughness variant ($n = 0.025$) improves arrival-time agreement, while maximum water-surface level errors stay similar for both roughness settings.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 2. Verification, validation and inter-model comparability suite. Each row is a generated public test case or generated variant. External-data cases are preconfigured and become strict failures only when strict checking mode is requested.

ID	Case	Type / grid	Primary check
VV-01	`ritter_dry`	analytical, 100/200/400 cells	L_1/L_2 depth error and monotone reduction against Ritter dry-bed solution
VV-02	`stoker_wet`	analytical, 100/200/400 cells	L_1/L_2 depth error and monotone reduction against Stoker wet-bed solution
VV-03	`lake_at_rest_immersed`	C-property, 50^2	$\max\|v\|$ and $\max\|\Delta\eta\|$ at the `float32` floor
VV-04	`lake_at_rest_emerged`	C-property, 50^2	same, with emergent bed cell and wet-cell velocity tolerance
VV-05	`synthetic_smooth_gci`	controlled GCI, 4 m, 2 m, and 1 m	scalar-QoI GCI and full-field Cauchy consistency on a smooth all-wet case
VV-06	`parity_dambreak`	CPU / CUDA / tiled CUDA	`depth_max` agreement within single-precision tolerance
VV-07	`mass_dryon`	budget, 200^2	relative water-budget closure on the normal dry-check path
VV-08	`mass_dryoff`	budget, 200^2	bounded positivity-floor clamp-bias identity when dry checking is disabled
VV-09	`mass_hvar`	budget, 200^2	closure with signed prescribed-level term $V_{\text{source h}}$
VV-10	`mass_sources_culverts`	budget, 200^2	closure with V_{sources}, $V_{\text{source h}}$ and V_{transit}
VV-11	`dresden_striesen`	self-referenced, 4/2/1 m	scalar QoI GCI and field-RMS consistency on the public Striesen subset
VV-12	`ea_05_valley_momentum`	EA benchmark Test 5	gauge-level depth, velocity and arrival time in a momentum-sensitive valley
VV-13	`ea_08a_pluvial`	EA benchmark Test 8A	rainfall-driven pluvial response and 0.5/2 m resolution consistency
VV-14	`upperlee/wf_compare`	inter-model, dense SWE CUDA	LISFLOOD-FP ACC/NU_ACC field and runtime context; WarpFlow native SWE, full-domain
VV-15(b)	`upperlee/wf_optimized`	inter-model, tiled SWE CUDA	diagnostic comparison of optimized WarpFlow SWE output and runtime
VV-16	`malpasset/n033`	applied validation	Malpasset arrival and maximum-level errors, Manning $n = 0.033$
VV-17	`malpasset/n025`	applied validation	same data and configuration family, Manning $n = 0.025$

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 3. Dam-break discretization convergence metrics for Ritter dry and Stoker wet cases across five grid levels ($L = 10\,\mathrm{m}$). Observed local order is $p = \log_2(L_2^n / L_2^{2n})$.

Case	Grid (n)	Δx (m)	$L_2(h)$ ($\mathrm{m}^{3/2}$)	p_h	$L_2(u)$ ($\mathrm{m}^{3/2}/\mathrm{s}$)	p_u
Ritter dry bed	50	0.2000	2.381×10^{-3}	—	4.707×10^{-1}	—
	100	0.1000	1.779×10^{-3}	0.421	4.610×10^{-1}	0.030
	200	0.0500	1.389×10^{-3}	0.357	4.666×10^{-1}	-0.018
	400	0.0250	1.143×10^{-3}	0.281	4.641×10^{-1}	0.008
	800	0.0125	9.995×10^{-4}	0.194	4.628×10^{-1}	0.004
Stoker wet bed	50	0.2000	2.358×10^{-3}	—	6.286×10^{-2}	—
	100	0.1000	1.790×10^{-3}	0.397	5.704×10^{-2}	0.140
	200	0.0500	1.342×10^{-3}	0.416	6.067×10^{-2}	-0.089
	400	0.0250	1.074×10^{-3}	0.321	5.900×10^{-2}	0.040
	800	0.0125	9.275×10^{-4}	0.212	5.809×10^{-2}	0.022

Table 4. Lake-at-rest well-balancing over the full run (50×50, 5 s). Velocity is the wet-cell maximum; the emergent case uses a larger velocity tolerance at the wet/dry interface.

Variant	$\max\|v\|_{\mathrm{wet}}$ ($\mathrm{m\,s^{-1}}$)	$\max\|v\|_{\mathrm{all}}$ ($\mathrm{m\,s^{-1}}$)	$\max\|\Delta\eta\|$ (m)
Immersed bed	4.236×10^{-6}	4.236×10^{-6}	1.118×10^{-6}
Emergent bump	1.122×10^{-5}	1.122×10^{-5}	7.246×10^{-7}

Table 5. Synthetic smooth-grid GCI on final instantaneous depth ($r = 2$, $F_s = 1.25$). Scalar GCI is reported for smooth integral QoIs; field RMS is a companion Cauchy-consistency diagnostic.

Quantity	Coarse	Medium	Fine	p	$\mathrm{GCI}_{\mathrm{fine}}$
Gaussian-weighted mean depth (m)	1.01295	1.01298	1.01299	0.835	0.002%
Perturbation energy proxy (m^4)	2.17353	2.18275	2.18748	0.966	0.283%
L_2 perturbation norm (m^2)	1.47429	1.47741	1.47901	0.968	0.141%
Central-patch mean depth (m)	1.01593	1.01597	1.01600	0.285	0.018%
Field RMS `depth_max` (m)	2.1806×10^{-5}		1.1477×10^{-5}		r=0.526 (shrinking)

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 6. Backend parity on the fully wet radial dam break (80×80). Differences are computed on `depth_max`. The case uses a fixed time step $\Delta t = 0.1\,\mathrm{s}$ and `dt_error`=1×10^{-8} to avoid backend-dependent adaptive-CFL and output-time snapping differences. All comparisons pass the configured max-norm tolerance $\max|\Delta| \leq 1 \times 10^{-4}\,\mathrm{m}$.

Comparison	$\max\|\Delta\|$ (m)	L_2 (m)	$\max_{\mathrm{rel}}$
CPU vs dense CUDA	1.490×10^{-7}	2.095×10^{-8}	5.741×10^{-7}
CPU vs tiled CUDA	1.490×10^{-7}	2.004×10^{-8}	7.382×10^{-7}
dense CUDA vs tiled CUDA	8.941×10^{-8}	1.396×10^{-8}	2.815×10^{-7}

Table 7. Online water-budget closure at the checked row ($t = 600\,\mathrm{s}$, $V_{\mathrm{rain}} = 8.168 \times 10^3\,\mathrm{m}^3$). Volumes are in m^3. The percent closure column is $100\,|\text{relative closure}|$. The combined source/culvert case demonstrates simultaneous activation of ordinary sources, prescribed-level exchange and culvert transit storage.

Variant	V_{sources}	$V_{\mathrm{source\,h}}$	V_{transit}	**mass_error**	**rel. closure**	**closure (%)**
`dryon`	0	0	0	-1.992×10^{-3}	2.0×10^{-7}	2.0×10^{-5}
`dryoff`	0	0	0	-1.992×10^{-3}	2.0×10^{-7}	2.0×10^{-5}
`hvar`	0	1.105×10^3	0	1.100×10^{-2}	1.2×10^{-6}	1.2×10^{-4}
`sources_culverts`	-2.390×10^2	1.106×10^3	5.356×10^{-1}	1.174×10^{-2}	1.3×10^{-6}	1.3×10^{-4}

Table 8. Dresden-Striesen self-referenced consistency ($r = 2$, $F_s = 1.25$, threshold $h > 0.1\,\mathrm{m}$). Scalar $\mathrm{GCI}_{\mathrm{fine}}$ is reported only for monotone QoIs. Field RMS is computed on `depth_max` over the flooded area and interpreted as a real-terrain Cauchy-consistency diagnostic.

Quantity	**Coarse** (4 m)	**Medium** (2 m)	**Fine** (1 m)	**$\mathrm{GCI}_{\mathrm{fine}}$ / verdict**
Peak depth (m)	1.948	2.221	2.187	Oscillatory
Flooded volume (m^3)	61 430	64 904	66 097	1.18 % ($p = 1.54$)
Flooded area (m^2)	3.320×10^5	3.468×10^5	3.534×10^5	1.87 % ($p = 1.17$)
Field RMS `depth_max` (m)	0.0454		0.0445	r=0.980 (shrinks)
Relative field RMS	23.78 % of mean fine-grid flood depth			above 5 % target

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

6 Performance and scalability

Section 5 verifies numerical behavior; this section measures computational behavior. We separate five claims: fixed-step solver-core throughput, scaling with problem size, the cost and benefit of adaptive execution, memory footprint, and output overhead. The public performance cases are generated by the benchmark harness and use either synthetic fully wet domains or the shared public Dresden 5 m benchmark dataset without Elbe River bathymetry. The full operational Dresden 2 m run uses restricted municipal infrastructure data. It is therefore reported separately as an operational showcase rather than as a public reproducibility case.

Table 9 gives the stable performance-case identifiers used throughout this section. The identifiers are analogous to the `VV-*` labels of Sect. 5. `PERF-01` and `PERF-02` isolate solver-core throughput and scaling on synthetic domains. `PERF-03` is a synthetic adaptivity sanity sweep. `PERF-04`–`PERF-07` use the public Dresden 5 m benchmark data. `PERF-08` is a metadata-only restricted Dresden 2 m operational showcase.

Table 9. Performance and scalability benchmark suite. Public cases are generated by the performance benchmark generator. Dresden 5 m cases use the shared public/releasable Dresden benchmark dataset. The restricted Dresden 2 m operational case is metadata-only in the public suite.

ID	Generator sweep	Type / data	Primary performance question
PERF-01	`throughput`	synthetic, fully wet, fixed time step	Raw solver-core cell-update throughput for CPU, dense CUDA and tiled CUDA with `output_off=true`.
PERF-02	`scaling`	synthetic, unaligned rectangular domains	Scaling from launch-bound small domains to large bandwidth-dominated domains.
PERF-03	`synthetic_adaptivity`	synthetic sanity sweep	Qualitative cost of active modes and CFL-reduction cadence on a controlled synthetic domain.
PERF-04	`dresden5_active_rain`	public Dresden 5 m, uniform rain	Worst-case active-tile overhead when most of the real urban domain becomes active.
PERF-05	`dresden5_active_elbe`	public Dresden 5 m, Elbe river QFIX line source	Active-tile benefit for sparse/localized forcing.
PERF-06	`dresden5_cfl`	public Dresden 5 m, uniform rain	Cost of every-step, fixed-stride and adaptive CFL reductions.
PERF-07	`output_overhead`	public Dresden 5 m	Overhead of statistics plugins, production rasters and dense time-slice output.
PERF-08	`dresden2_restricted`	restricted Dresden 2 m, metadata only	Operational city-domain showcase with restricted municipal infrastructure data; no public runnable config.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

6.1 Hardware, software and measurement methodology

Performance cases are generated, run and evaluated through the reproducible harness `performance.py`. The generator implementation is `performance_generate.py`; execution and collation are handled by `performance_run.py` and `performance_evaluate.py`. The public benchmark families are those listed in Table 9. Synthetic cases generate their rasters inside the benchmark folder. Dresden 5 m cases reference shared public/releasable data under `data/dresden_5m` and shared rainfall under `data/rain`. The restricted Dresden 2 m case (`PERF-08`) emits no public runnable configuration.

The main timing quantity is end-to-end wall time for the solver run. Throughput P is reported as effective terrain-cell updates per second,

$$P = \frac{\sum_k N_{\mathrm{work},k}}{t_{\mathrm{wall}}}, \tag{33}$$

where $N_{\mathrm{work},k}$ is the number of terrain cells advanced in step k. For the fixed-step, full-domain synthetic benchmarks of Sects. 6.2–6.2, this is simply the number of terrain cells multiplied by the number of steps. Boundary wall cells and ghost padding are not counted as cell updates.

All timings use the bundled `performance_measure` plugin (Sect. 4.5). Unless a run is explicitly intended to measure output cost, regular GeoTIFF output is disabled with `output_off=true`. The lightweight plugin CSV and summary files remain enabled. This separates solver throughput from filesystem, raster assembly and compression cost.

The authoritative timing is the wall-clock time reported by the performance plugin at the end of each run. On CUDA backends, the plugin also records CUDA event time per sampling window without forcing a full all-stream synchronization. The diagnostic ratio

$$\mathrm{gpu_util} = \frac{t_{\mathrm{device}}}{t_{\mathrm{wall}}} \tag{34}$$

is used only to distinguish launch-bound from compute-bound regimes. It is not used for speed-up claims, because WarpFlow deliberately overlaps CFL-side-stream work and background output with the main loop. Wall time is therefore the consistent denominator for all tables.

Each benchmark row records the case identifier, backend, storage layout, execution mode, `dt_limiter`, `dt_stride`, output policy, simulated duration, work count, step count, wall time and throughput. The `performance_measure` overhead is small in the reported throughput and scaling runs. Across the synthetic CPU/CUDA throughput and scaling rows, measured overhead is at most 0.541 %; in the largest GPU scaling rows it is 0.035 % for both CUDA layouts. We therefore treat the plugin as effectively measurement-only for these comparisons.

6.2 Solver-core throughput and scaling

The synthetic throughput and scaling benchmarks, `PERF-01 throughput` and `PERF-02 scaling` (Table 9), measure the same underlying quantity: fixed time step, full-domain, output-disabled solver-core cell updates on fully wet, spatially uniform domains. In all rows, the time step limiter is fixed with `dt_limiter="f"`, adaptive CFL reductions are inactive,

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

regular output is disabled with `output_off=true`, and each run advances 1001 fixed steps over a 1000 s synthetic interval. These tests therefore isolate the hot-loop stencil kernels. They deliberately exclude the benefits of active-tile execution, adaptive CFL cadence and asynchronous raster output.

Table 10 combines two related sweeps, and Fig. 16 is the log–log visualization of the same data. The first block is the square throughput sweep, reported by terrain-work grid size: 254^2, 1022^2 and 4094^2, corresponding to 64 516, 1 044 484 and 16 760 836 terrain cells. Boundary and ghost cells are excluded from the throughput denominator. This block gives the cleanest comparison of CPU, dense CUDA and tiled CUDA on regular square domains under the solver's normal boundary layout. The second block is the rectangular scaling sweep. It uses deliberately unaligned rectangular domains from 128×80 to 12500×8000 to expose launch overhead, bandwidth saturation, memory-capacity effects and row-pitch sensitivity.

The square throughput block shows the transition from small-domain overhead to large-domain stencil throughput. At 254^2 terrain cells, all backends are still dominated by fixed overheads, and the CPU reference remains competitive (0.0358 Gcells/s) with dense CUDA (0.0417 Gcells/s) and tiled CUDA (0.0563 Gcells/s). At 1022^2 terrain cells, the CUDA backends reach approximately 1 Gcells/s, giving a 13.0–13.5× speedup over the CPU reference. On the largest square case, dense CUDA reaches 5.350 Gcells/s and tiled CUDA reaches 5.257 Gcells/s, compared with 0.0873 Gcells/s on CPU. The corresponding GPU/CPU speedups are 61.2× for dense CUDA and 60.2× for tiled CUDA. The two CUDA layouts are therefore effectively tied on regular square domains; the measured difference at the largest size is only about 2 %.

The rectangular scaling block shows the launch-bound to bandwidth-dominated transition over a wider size range. At 9828 terrain cells, the CPU reference is faster than either GPU backend, because fixed host orchestration and kernel-launch costs dominate the GPU wall time. By 98 704 terrain cells, the CUDA backends are already faster than CPU. Around one million terrain cells, throughput rises to 0.941 Gcells/s for dense CUDA and 0.935 Gcells/s for tiled CUDA, about 12.5× the CPU reference. Around ten million terrain cells, the solver enters the large-domain regime, with dense CUDA at 4.798 Gcells/s and tiled CUDA at 4.817 Gcells/s, roughly 55–56× the CPU reference.

The largest rectangular scaling row is the most representative synthetic stress test. On 99 959 004 terrain cells, dense CUDA reaches 6.222 Gcells/s, and tiled CUDA reaches 6.112 Gcells/s, compared with 0.0901 Gcells/s on CPU. The corresponding GPU/CPU speedups are 69.1× and 67.8×. Dense CUDA is therefore slightly faster on this fully terrain-filled rectangular case, but the difference between CUDA layouts is again small ($\approx 1.8\,\%$). The appropriate conclusion is not that one layout universally dominates the other on dense synthetic domains. Both CUDA layouts converge to approximately 6 Gcells/s on the measured RTX 4080 testbed. The tiled layout's main advantage is therefore not dense-domain throughput, but terrain-sparse allocation, row-pitch independence and stable performance on irregular urban rasters.

The small-grid rows show the launch-overhead tax on the GPU. For domains below roughly 10^4–10^5 terrain cells, fixed orchestration dominates the measured GPU wall time, and CPU execution can be faster. From 10^6 to 10^7 terrain cells, throughput rises rapidly as the GPU becomes better occupied. At 10^8 terrain cells, wall time scales nearly linearly with work and the solver reaches its bandwidth-dominated large-domain regime. The large square and large rectangular rows are therefore the relevant solver-core throughput results.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 10. Solver-core throughput and scaling (1001 fixed time steps over 1000 s, full-domain execution, `output_off=true`). Throughput is computed from terrain-cell updates and end-to-end wall time. Boundary and ghost cells are excluded from the throughput denominator.

sweep	terrain grid	backend	terrain cells	wall time (s)	throughput (Gcells/s)
sq. throughput	254^2	CPU	64 516	1.803	0.035823
sq. throughput	1022^2	CPU	1 044 484	14.013	0.074611
sq. throughput	4094^2	CPU	16 760 836	192.084	0.087345
sq. throughput	254^2	dense CUDA	64 516	1.550	0.041674
sq. throughput	1022^2	dense CUDA	1 044 484	1.077	0.970782
sq. throughput	4094^2	dense CUDA	16 760 836	3.136	5.349806
sq. throughput	254^2	tiled CUDA	64 516	1.146	0.056335
sq. throughput	1022^2	tiled CUDA	1 044 484	1.038	1.007177
sq. throughput	4094^2	tiled CUDA	16 760 836	3.191	5.257325
rect. scaling	128×80	CPU	9828	0.652	0.015082
rect. scaling	400×250	CPU	98 704	2.448	0.040361
rect. scaling	1250×800	CPU	995 904	13.322	0.074832
rect. scaling	4000×2500	CPU	9 987 004	115.602	0.086477
rect. scaling	12500×8000	CPU	99 959 004	1110.497	0.090103
rect. scaling	128×80	dense CUDA	9828	1.537	0.006402
rect. scaling	400×250	dense CUDA	98 704	1.075	0.091886
rect. scaling	1250×800	dense CUDA	995 904	1.059	0.940976
rect. scaling	4000×2500	dense CUDA	9 987 004	2.084	4.798103
rect. scaling	12500×8000	dense CUDA	99 959 004	16.081	6.222187
rect. scaling	128×80	tiled CUDA	9828	1.140	0.008628
rect. scaling	400×250	tiled CUDA	98 704	1.080	0.091525
rect. scaling	1250×800	tiled CUDA	995 904	1.066	0.935174
rect. scaling	4000×2500	tiled CUDA	9 987 004	2.075	4.817244
rect. scaling	12500×8000	tiled CUDA	99 959 004	16.370	6.112275

6.3 Cost and benefit of adaptive execution

WarpFlow provides two independent adaptivity mechanisms: active-tile work scheduling (Sect. 3.4) and an amortized adaptive CFL reduction cadence (Sect. 3.5). Both trade runtime bookkeeping overhead for avoided computational work. We evaluate both on the public Dresden 5 m benchmark domain using paired benchmark runs (`PERF-04`–`PERF-06`, Table 9). Figure 17 shows the dynamic active-footprint evolution and the adaptive time step behavior.

Active-tile execution dynamics. Active-tile scheduling is evaluated under two contrasting spatial forcing regimes: a spatially uniform, one-hour Euler-II design storm across the entire domain (`PERF-04`), the worst case where surface wetting is ubiquitous, and an Elbe river localized line source (`PERF-05`), a case of highly localized inundation propagation.

Table 11 summarizes execution timings across activity-tile block sizes (8×8, 16×16 and 32×32). Under uniform rainfall, full-domain execution completes in 57.150 s. Pure active-tile mode is counterproductive across all block sizes (83.436 s for 8×8, $+46.0\%$; 72.697 s for 16×16, $+27.2\%$; and 73.200 s for 32×32, $+28.1\%$), because the shared-memory detect–compact

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

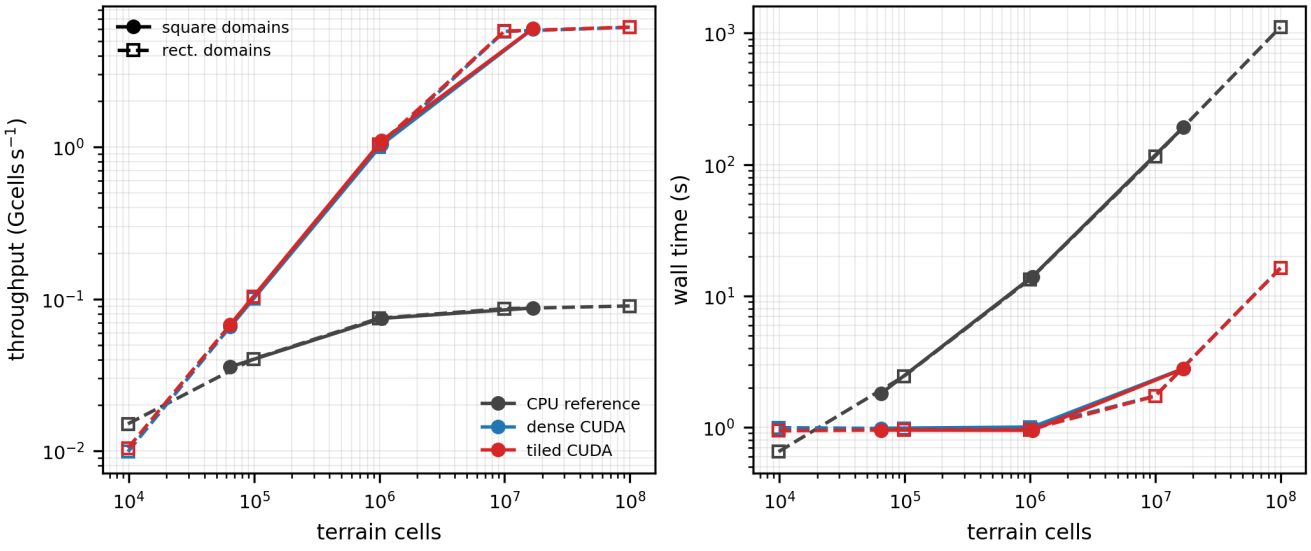


Figure 16. Solver-core throughput and scaling over regular square terrain-work grids and unaligned rectangular synthetic domains, generated from Table 10. Left: effective terrain-cell throughput versus terrain-cell count; right: end-to-end wall time. The log–log view shows the transition from launch-bound small cases, where CPU execution can remain competitive, to bandwidth-dominated large cases, where both CUDA layouts converge to approximately 6 Gcells/s.

pipeline overhead yields little avoided work when most terrain-capable tiles become active. The heuristic `auto` mode remains in full mode in this case and is within measurement scatter of the full baseline.

For the localized Elbe river QFIX line source, active-tile scheduling instead yields substantial savings. Figure 17(a) tracks the active-domain footprint over a 120-minute window. Every curve stays below 1% of the terrain domain throughout. The true inundated cell fraction (dashed curve) grows from near zero to only $\approx 0.18\%$ at $t = 120\,\mathrm{min}$, and the launched active-tile footprint tracks it with a block-size-dependent dilation: 8×8 tiles bound the wetting front most tightly ($\approx 0.38\%$ active domain), 16×16 tiles reach $\approx 0.56\%$, and 32×32 tiles dilate furthest, to $\approx 0.9\%$, as halo padding and coarser spatial quantization pad each active tile with a larger dry margin. Because every configuration launches well under 1% of the domain, the avoided flux compute is near-total in all three cases. The residual wall-time differences are then governed by the detect/dilate/compact bookkeeping and by kernel-launch efficiency, not by the small differences in launched cell count. The coarsest 32×32 tiling is therefore the fastest despite launching the most cells: its fewer, larger tiles give better streaming-multiprocessor occupancy and a cheaper compaction pass, reducing total wall time from $33.168\,\mathrm{s}$ (full mode) to $18.613\,\mathrm{s}$, a 43.9% reduction ($1.78\times$ speedup). The `auto` controller reproduces this automatically ($18.669\,\mathrm{s}$, 43.7% reduction), after one initial switch from full to active-tile execution.

Adaptive CFL reduction controller. The adaptive CFL stride controller (`dt_stride = -1`) replaces fixed-interval reductions by monitoring relative shifts in the characteristic wave speed $S_{\max}$ (Eq. (25)). The controller evaluates the logarithmic

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 11. Active execution on the public Dresden 5 m benchmark. All runs use tiled CUDA, `dt_limiter="cv"`, `dt_stride=1`, and `output_off=true`. Ratios are relative to the full-mode row of the same forcing pattern.

forcing	mode / tile size	steps	wall time (s)	ratio to full	change
uniform rain	full	17 830	57.150	1.000	baseline
	active tile, tile 8	17 830	83.436	1.460	+46.0 %
	active tile, tile 16	17 830	72.697	1.272	+27.2 %
	active tile, tile 32	17 830	73.200	1.281	+28.1 %
	auto, tile 8	17 830	56.405	0.987	-1.3 %
	auto, tile 16	17 830	55.593	0.973	-2.7 %
	auto, tile 32	17 830	56.101	0.982	-1.8 %
Elbe river QFIX	full	12 061	33.168	1.000	baseline
	active tile, tile 8	12 061	25.968	0.783	-21.7 %
	active tile, tile 16	12 061	19.429	0.586	-41.4 %
	active tile, tile 32	12 061	18.613	0.561	-43.9 %
	auto, tile 8	12 061	26.039	0.785	-21.5 %
	auto, tile 16	12 061	19.560	0.590	-41.0 %
	auto, tile 32	12 061	18.669	0.563	-43.7 %

time step variation $r = |\log(\Delta t^{n+1}/\Delta t^n)|$. When $r > 0.15$ or the flow exits a calm state, the stride contracts to preserve stability during rapid wave acceleration. When the relative change remains small over consecutive samples, the stride expands, bounded by the allowed simulated time gap between CFL reductions.

Figure 17(b) details this control loop under the two-hour design storm. During the dry and nearly calm start, the controller can increase the recompute interval while Δt remains capped at 5 s. Once surface runoff accelerates, Δt collapses from 5.0 s to below 1 s, and the controller repeatedly contracts the stride to near-every-step sampling. During the main storm response, Δt continues to fall toward ≈ 0.31 s, but the controller expands only when consecutive samples indicate a stable characteristic speed. After the peak transition, the stride grows much further than in the earlier benchmark configuration, reaching a maximum of 965 steps. It does not simply stay at that maximum: rainfall-index updates and evolving wet/dry boundaries trigger temporary contractions, after which the stride re-expands and settles near 800 steps through the recession.

As quantified in Table 12, the adaptive controller reduces global CFL reduction passes from 17 830 in the every-step baseline to 161, a 99.1 % reduction. The wall time decreases from 56.318 s to 47.736 s, a 15.2 % reduction. A fixed stride of 10 captures most of the wall-time benefit (48.529 s, 13.8 % reduction), but still performs 1780 reductions and cannot adapt to the changing storm phases.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Table 12. CFL-reduction cadence on Dresden 5 m. All runs use tiled CUDA, full mode, uniform rain without Q sources, gullies, culverts or restricted drainage infrastructure, and `output_off=true`.

`dt_stride`	steps	CFL reductions	stride mean	stride max	wall time (s)	ratio	change
1	17830	17830	1.0	1	56.318	1.000	baseline
10	17795	1780	10.0	10	48.529	0.862	-13.8 %
adaptive −1	17780	161	595.8	965	47.736	0.848	-15.2 %

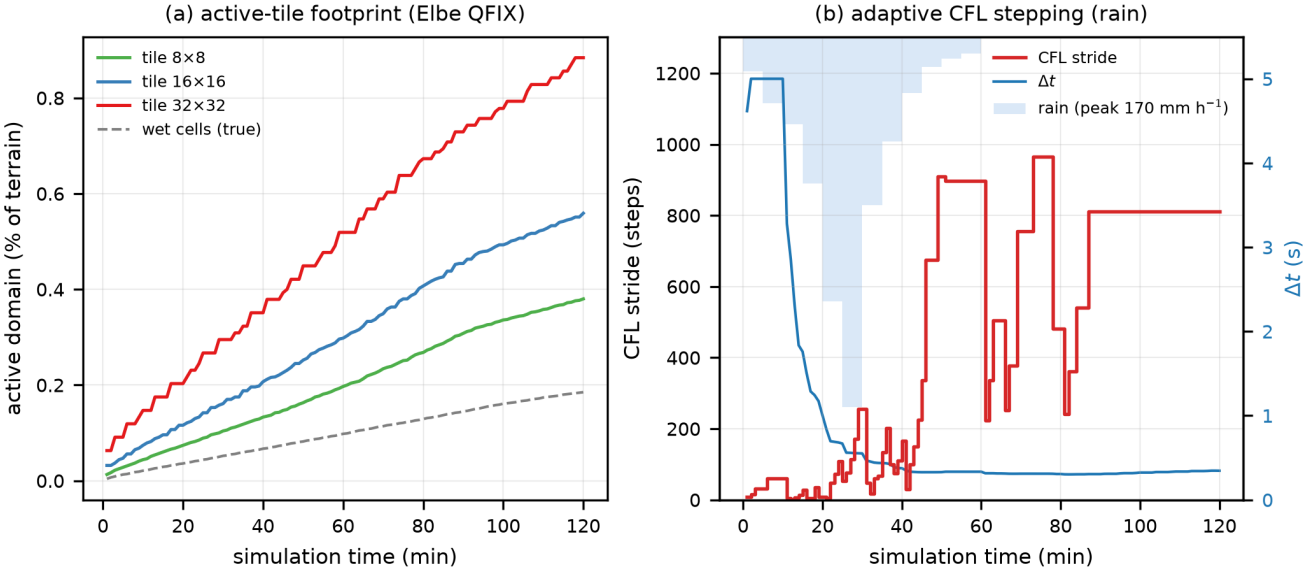


Figure 17. Measured adaptivity mechanics on the Dresden 5 m benchmark. (a) Temporal expansion of the active-domain footprint for the localized Elbe river QFIX source across tile sizes (8, 16, 32) compared to true wet cells, all remaining below 1% of the terrain domain. (b) Adaptive CFL controller stepping (`dt_stride`, red) and time step (Δt, blue) under the two-hour hyetograph. The stride contracts during storm onset and rainfall-transition phases, expands to nearly 1000 steps once the characteristic speed stabilizes, and then settles below the maximum during the recession as wet/dry boundaries continue to evolve.

6.4 Memory footprint and hardware utilization

The core CUDA solver state uses a 39 B per allocated-cell budget: nine `float32` fields, one `int8` cell-type field, and two `uint8` class-map fields. This is the core hydrodynamic storage only; the runtime allocator report also includes auxiliary arrays such as terrain indices, sparse tile maps, rainfall coordinate maps, and LUT scalars.

In dense CUDA, the 39 B/cell core budget is applied to the full padded bounding rectangle. In tiled CUDA, it is applied to all cells contained in active 32×32 storage tiles. Therefore the tiled allocated-cell count is not equal to the number of terrain cells: partially filled tiles still allocate complete 32×32 slabs.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

For the public Dresden 5 m benchmark, the padded dense domain has 26 590 272 cells, of which 14 600 702 are terrain cells (54.9 %). The tiled layout activates 15 427 storage tiles, corresponding to 15 797 248 allocated tiled cells (59.4 % of the dense padded domain). Using the allocator estimate, dense CUDA requires 1044.7 MiB, whereas tiled CUDA requires 668.7 MiB, giving a 375.9 MiB, or 36.0 %, reduction in solver-side device storage.

Table 13 projects the maximum domain bounding-box capacity across standard GPU VRAM sizes. If capacities are projected from the core 39 B/cell budget alone, the capacity is 27.53×10^6 cells per GiB. If the allocator-style dense estimate is used instead, including the terrain-index array at approximately 55 % terrain coverage, the effective capacity is about 26.1×10^6 dense bounding-box cells per GiB.

Table 13. Estimated maximum domain capacity (in millions of cells, 10^6) across standard GPU VRAM sizes. Dense layout allocates the full bounding rectangle; tiled CUDA capacity scales with active terrain percentage. No additional statistics/plugin memory allocations.

		Tiled CUDA (% terrain)			
VRAM (GiB)	**Dense**	**100 %**	**75 %**	**50 %**	**25 %**
4	104	104	139	208	417
8	208	208	278	417	833
12	313	313	417	625	1250
16	417	417	556	833	1667
24	625	625	833	1250	2500
32	833	833	1111	1667	3333
48	1250	1250	1667	2500	5000
80	2083	2083	2778	4167	8333
96	2500	2500	3333	5000	10000

6.5 Output and plugin overhead

The throughput and scaling benchmarks disable regular GeoTIFF output, because large raster writes can dominate wall time. Operational runs, however, need statistics rasters, maximum-depth fields, time of arrival, duration, hazard diagnostics and selected time slices. We therefore measure output and statistics overhead separately in `PERF-07 output_overhead` (Table 9) on the public Dresden 5 m benchmark.

All rows in Table 14 use tiled CUDA, full mode, `dt_limiter="cv"`, `dt_stride=10`, uniform rain without Q sources, gullies, culverts or restricted drainage infrastructure, and the same 7200 s simulated duration. The output-disabled performance baseline takes 49.670 s. Enabling the statistics plugin without GeoTIFF output takes 49.212 s, and production-interval rasters with statistics take 49.219 s. These two rows are slightly faster than the output-disabled baseline in this run, so we do not interpret the sign of the difference physically; all three timings are within measurement scatter. Dense time-slice output is

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

the one costly policy: computing statistics every iteration and writing dense historical rasters takes 78.615 s, or 1.583× the baseline, because it moves far more memory bandwidth per step and lowers measured GPU utilization during output.

Table 14. Output and statistics overhead on the public Dresden 5 m benchmark. Ratios are relative to the output-disabled performance baseline. The statistics-only and production-interval rows are within timing scatter of the baseline; dense time-slice output is the only clearly costly policy.

configuration	GeoTIFF output	statistics plugin	steps	wall time (s)	ratio
performance output off	off	off	17795	49.670	1.000
statistics only	off	on	17795	49.212	0.991
production interval rasters	on	on	17795	49.219	0.991
dense time-slice output	on	on	17802	78.615	1.583

The practical conclusion is that the statistics plugin and production-style interval rasters are not the limiting cost in this benchmark; dense time-slice output is. Solver-core throughput should therefore be quoted from output-disabled rows, while production turnaround estimates must include the chosen raster-output policy.

6.6 Operational showcase: Dresden city-domain run

The Dresden city-domain run demonstrates the restricted operational deployment for a city-wide, high-intensity pluvial scenario in full mode (Sect. 3.4). Both resolutions use the same municipal input data: a spatially uniform 100-year, one-hour Euler-II design storm (DWA, 2025) with a peak 5 min intensity of ≈170 mm/h from the KOSTRA-DWD-2020 statistics (Junghänel et al., 2023), roughness and abstraction fields, drain/free-flow boundaries, 110523 gully/orifice sinks, and 301 hydraulically resolved culverts. Roughness is split into 14 Manning classes and abstraction into 22 infiltration classes, with $\mathrm{CFL} = 0.95$. The storm forces the first hour, and the model runs a 7200 s (two-hour) window to capture the runoff recession. Operational outputs are written every 300 s. Figure 18 shows the varying (a) landuse, (b) slope, (c) critical infrastructure including culverts and gully drainages, and (d) maximum water depth across the Dresden administrative area.

At 5 m resolution the domain is 5622 by 4726 cells, of which 14600702 are terrain cells, a terrain-only planimetric area of 365.02 km^2. The full run completes in 51.68 s on the RTX 4080, against 3156.26 s (≈ 52.6 min) for the same solver's 24-thread CPU backend, a 61.1× speedup.

The separate public output-overhead sweep without Q sources, gullies, culverts or restricted drainage infrastructure gives a production-interval raster time of 49.219 s on the same 5 m benchmark family (Table 14). The restricted operational 5 m run with 110523 gully sinks and 301 culverts completes in 51.68 s, only 2.46 s slower (1.05×, or 5.0 %). This small increment shows that the additional software complexity of hydraulically resolved gully/orifice sinks and culvert transport remains minor in the end-to-end city-scale runtime. The source-term kernels are therefore not the limiting cost; the operational turnaround remains governed primarily by the solver stencil, CFL cadence and production raster output.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

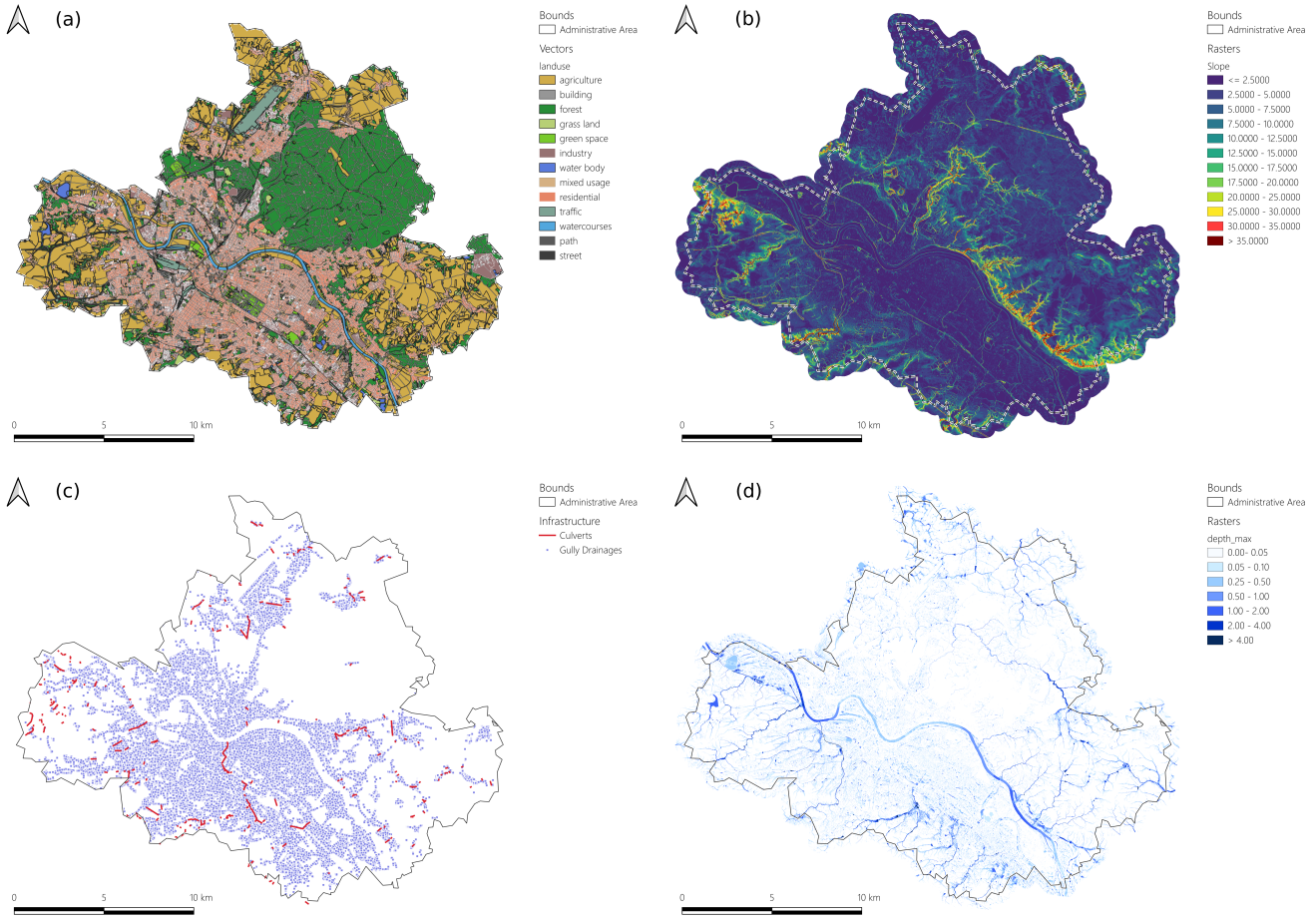


Figure 18. Spatial characteristics and simulation results for the Dresden benchmark domain: (a) Landuse classification including agricultural, forest, residential, and infrastructure vectors; (b) Terrain slope distribution in degrees; (c) Location of critical drainage infrastructure (culverts and gully drainages); (d) Raster distribution of maximum inundation water depth (`depth_max`).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

At 2 m resolution the same scenario uses a 14 056 by 11 816 grid with 90 803 396 terrain cells (363.21 km^2). It completes in 863.60 s (≈ 14.4 min) on the GPU, against 55 502.17 s (≈ 15.42 h) on the CPU, a 64.3× speedup. Separate water-balance runs close to a mass error of 0.0003 % at 5 m and −0.0261 % at 2 m without the flux limiter.

For context, Buttinger-Kreuzhuber et al. (2022) apply an essentially identical forcing, a spatially uniform, approximately 100-year one-hour rainfall (53 mm/h) over a two-hour window, to the entire city of Cologne on a single NVIDIA Titan RTX, completing a 220×10^6-cell 1.5 m full-SWE run just faster than real time but using up to 23.4 GB of the card's 24 GB. The comparable 2 m Dresden run uses only 6.5 GB with all diagnostics and source terms enabled, about 40 % of the RTX 4080's 16 GiB. WarpFlow's terrain-sparse tiling and 39 B-per-cell state thus place a Cologne-scale domain within a single commodity 16 GiB card rather than a 24 GB device.

These operational speedups sit below the synthetic ceiling of Sect. 6.2, and the reason is informative. The fully wet, output-disabled synthetic benchmark reaches approximately 68–69× over the CPU reference on the largest rectangular domain. The Dresden runs reach 61.1× (5 m) and 64.3× (2 m), because the real runs also evaluate the 110 523 gully sinks, 301 culverts, and periodic raster output, none of which the uniform synthetic case includes. These per-step source-term and output costs take a larger share of the shorter GPU wall time than of the long CPU wall time, so the measured end-to-end ratio falls short of the pure stencil ceiling. The numbers are therefore the realistic operational figure, not a best case.

The practical outcome is that a city-wide, 100-year-intensity pluvial flood is computed in under a minute at operational 5 m output resolution, which supports timely flood nowcasting. A spatiotemporally varying rainfall field using active-tile mode can run considerably faster (Sect. 6.3). The high-detail 2 m simulation completes in about 14.4 min, fast enough for automated daily recomputation of multiple scenarios whenever new or altered data sources arrive, which keeps the city flood assessment up to date.

7 Discussion

The results support one consistent reading: conservative numerics paired with deliberate engineering. We discuss what this buys, and where it stops.

Numerical scope. WarpFlow updates mass conservatively and advances momentum in advective, non-conservative form (Sect. 2.4). It is not a Riemann shock-capturing solver, and the first-order donor-cell dissipation is visible in the dam-break convergence: depth converges sub-linearly, and the velocity error plateaus because of the dry-front formula mismatch in Ritter flow and the localized shock displacement in the Stoker case (Sect. 5.2). The scheme therefore behaves exactly as its first-order construction predicts, without reproducing the analytical dam-break solution pointwise. This is the expected cost of a monotone, first-order flux. It is acceptable for the target regime, urban pluvial and fluvial inundation, where robustness and positivity matter more than the exact speed of a severe bore. The Malpasset case confirms this boundary: WarpFlow reproduces the arrival and peak-level trends but is not intended as a dam-break code (Sect. 5.8). The critical-ejection drain boundary (Sect. 2.3) suits flash-flood domains where edge backwater is not of interest, and is not appropriate where a true downstream stage controls the solution.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Roughness calibration on coarse grids. Two independent comparisons favored lower effective roughness in hydraulically informative parts of the domain. In Upper Lee, reducing Manning roughness to $0.8 \times n$ aligned the WarpFlow stage hydrograph more closely with the ACC GPU comparison record at the reported gauge, while the domain-wide RMSE remained a descriptive metric dominated by off-channel uplands (Sect. 5.7). In Malpasset, the lower value $n = 0.025$ improved arrival-time agreement and slightly reduced maximum-level RMSE (Sect. 5.8). We read both consistently: a coarse raster DEM underresolves conveyance geometry and adds effective damping, so a lower Manning n can compensate for grid-scale conveyance loss. This is a resolution-dependent calibration, not a defect of the friction closure, and it is consistent with the grid sensitivity seen in the self-referenced consistency study (Sect. 5.5).

Well-balancing and conservation. The scheme preserves the still-water C-property at the single-precision floor (Sect. 5.2), and its only conservation defect is the positivity-floor clamp bias. That defect is not hidden. The online water balance reports it directly, and it closes below 1.3×10^{-6} relative error on the source-accounting test cases and below $0.03\,\%$ on the operational Dresden runs (Sect. 5.4, Sect. 6.6). Where exact conservation across draining fronts is required, the parallel-safe FCT limiter reduces the bias further (Sect. 2.5).

Single source, three backends. The engineering claim is the stronger one. One kernel formulation runs on CPU, dense CUDA and tiled CUDA, and the three agree to single-precision round-off (Sect. 5.3). This is a design outcome, not a calibration: the Jacobi velocity update and single-writer face ownership make the result independent of traversal order. The practical value is a deterministic CPU reference for small cases and safe GPU execution for large ones, with no second-class backend and no GPU-only physics.

Adaptivity is a trade-off, not a free win. We report both sides of the adaptive machinery. Active-tile scheduling is counterproductive under near-uniform wetting, where pure active-tile execution is about 27–46 % slower than full mode, and it saves about 44 % on a localized Elbe line source (Sect. 6.3). Amortizing the CFL reduction is a smaller, more uniform 15 % gain. Both mechanisms serve the target user: a municipal operator who should not have to tune the solver by hand. The `auto` controller is what delivers this. It stays within measurement scatter of full mode in the bad case, captures nearly all of the benefit in the localized-source case, and needs no manual configuration.

Positioning. Against the solver landscape of Table 1, WarpFlow does not claim the highest-order numerics or the largest domain. Its contribution is the combination that the operational use case needs: full SWE, a structured grid generated directly from municipal rasters, an API-first Python package, city-scale domains on a single consumer GPU, and a plugin boundary that already carries a coupled drainage, transport and morphodynamics suite (Sect. 4.6). Recent reviews of flood and urban-water digital-twin applications identify precisely this implementation gap: the main barriers are no longer only conceptual model design, but heterogeneous data integration, real-time two-dimensional hydrodynamic cost, and the difficulty of turning prototypes into systems that can be run routinely by municipalities (Hlal et al., 2025; Lo et al., 2026). WarpFlow addresses the hydrodynamic-engine part of that gap. The Upper Lee comparison shows the immediate consequence: WarpFlow reads LISFLOOD-FP input near-natively and reproduces the broad ACC inundation pattern while exposing the expected stream-network differences caused by retaining advective momentum. It is therefore usable as a full-SWE alternative in many ACC-style data workflows, but not as a proof of equivalence to the local-inertial ACC scheme (Sect. 5.7).

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Limitations. The limitations follow from the same design choices that make WarpFlow useful for an operational flood twin. First, the raster workflow makes model generation automatable, but it also makes the result dependent on DEM conditioning, resolution, cell-type encoding and effective roughness calibration. The Dresden-Striesen consistency test therefore should be read as an application-scale sensitivity check on an urban raster model, not as a formal proof of resolution-independent flood depths. Second, the core source and boundary representations are intentionally lightweight. Critical-ejection drain edges, orifice-based gully sinks and lumped-delay culverts are appropriate for rapid city-scale pluvial screening and nowcasting, but cases controlled by downstream stage, backwater, pressurized pipe hydraulics or bidirectional sewer–surface exchange require more explicit boundary condition handling or coupled-network physics. Third, the performance results are operational workstation benchmarks, not universal hardware constants. Absolute turnaround depends on domain wetness, source-term load, output policy, adaptive time-step trajectory and GPU generation; the transferable result is the measured scaling behaviour and the demonstrated ability to run the target city-scale workflow on a single commodity GPU.

8 Conclusions

We presented WarpFlow, an open-source solver for the two-dimensional shallow-water equations, built for operational, full-scale urban flood modeling. It solves the full SWE on a staggered Arakawa C-grid with a well-balanced hydrostatic reconstruction, an upwind mass flux, and a semi-implicit momentum update, and it guarantees non-negative depth by construction. A single kernel formulation compiles to CPU, dense CUDA and memory-sparse tiled CUDA, and the three backends agree to single-precision round-off.

The headline result meets the three requirements set in the introduction. On one consumer GPU, a 100-year design-storm simulation over the administrative area ($\approx 365\,\mathrm{km}^2$) of Dresden completes in $\approx 14.4\,\mathrm{min}$ at $2\,\mathrm{m}$ resolution ($\approx 1.66 \times 10^8$ total cells) and in $51.68\,\mathrm{s}$ at $5\,\mathrm{m}$ resolution ($\approx 2.66 \times 10^7$ total cells). Against the same solver's 24-thread CPU backend, this is a $64.3\times$ speedup at $2\,\mathrm{m}$ and a $61.1\times$ speedup at $5\,\mathrm{m}$ (Sect. 6.6). The runs stay within the memory of a commodity card, and the scripting API drives them without hand editing. City-scale runs therefore complete in minutes, on hardware a municipality can own, inside an automated pipeline.

These three properties change what a municipality can do. A city-wide, 100-year pluvial scenario at $5\,\mathrm{m}$ now computes in under a minute, which brings hydrodynamic flash-flood nowcasting and rapid "what-if" scenario testing into operational reach. The $2\,\mathrm{m}$ run is fast enough for unattended daily recomputation, so a flood assessment can track the city's continual change instead of ageing between multi-year mapping cycles. Because the model is generated directly from municipal GIS data, authority over a piece of critical infrastructure stays with the city that operates it, rather than moving to an external consultancy.

WarpFlow is aimed at three groups. Practitioners get minute-scale city runs, a native reader for existing LISFLOOD-FP ACC inputs, and modern GIS input and output. Researchers get an inspectable, single-source Python solver whose four bundled plugins show how to add new physics on all three backends without touching the flux kernels. Municipalities get an operational engine that runs on commodity hardware without an HPC cluster or a per-seat license.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

This is the accessibility argument: high-resolution large-scale urban flood modeling does not need to depend on specialist infrastructure. It can be installed with a single conda environment. A QGIS plugin is under development. The Python-native, plugin-based design also positions WarpFlow to interoperate with the wider open-source earth-surface ecosystem, for example similar to how Landlab (Hobley et al., 2017) integrated the LISFLOOD-FP ACC formulation, rather than to stand alone.

WarpFlow is the hydrodynamic engine behind the operational Urban Digital Flood Twin for Dresden. This paper documented the solver core and its verification. Follow-up publications will validate the coupled drainage-network, transport and morphodynamics plugins that the same architecture already carries (Sect. 4.6). The code is released under the AGPLv3 license, so the community can test, extend and build on it.

Code and data availability. WarpFlow v1.0 is released under the GNU Affero General Public License v3 (AGPLv3). The exact source-code version described in this paper is archived on Zenodo at https://doi.org/10.5281/zenodo.22275530 (Backhaus and Stamm, 2026a). The archive contains the source tree, installation instructions, environment specification, user documentation, configuration reference and test harness. A public git repository will be made available after publication.

The verification, validation and performance reproducibility package is archived separately on Zenodo at https://doi.org/10.5281/zenodo.22276867 (Backhaus and Stamm, 2026b). It contains the generated test cases, public case data, TOML configurations, simulation outputs, and figure-generation scripts. Performance wall times are machine-local benchmark results; raw CLI logs are not treated as invariant regression targets.

The Environment Agency cases follow the Neelz–Pender benchmark specification (Neelz and Pender, 2013). The Upper Lee comparison uses the published LISFLOOD-FP dataset of Sharifian et al. (2023). Public Dresden $5\,\mathrm{m}$ and Dresden-Striesen inputs are included in the reproducibility archive with case-local source and license notes.

The Dresden operational showcase uses municipal gully, culvert and drainage data owned by the City of Dresden and treated by the data holder as security-sensitive critical-infrastructure information. These restricted data and tests are not redistributed.

Author contributions. Lars Backhaus: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, visualization, writing – original draft, funding acquisition, project administration. Jürgen Stamm: supervision, funding acquisition, writing – review.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. This work was supported by the German Federal Ministry for Housing, Urban Development and Building (BMWSB) and KfW under the Smart Cities Model Projects programme (Grant No. 14675631 – Landeshauptstadt Dresden). We thank the Environmental Office, Geoservice Office and Eigenbetrieb IT-Dienstleistungen of the City of Dresden for project support and for providing the environmental and municipal data used in the operational Dresden showcase.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

AI-use declaration. Large-language-model tools (Claude Opus 4.8 and GPT-5.5 Thinking) were used to support development of the software and automated test suite, and to assist with manuscript structuring, language editing and consistency checks. All scientific claims, implementation details, numerical results and conclusions were independently checked and validated by the authors.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

References

Altındaş, G., Dane, G., and Baba, A.: Urban digital twins and water sensitive urban design: A systematic literature review, Sustainable Cities and Society, 145, 107 456, https://doi.org/10.1016/j.scs.2026.107456, 2026.

Apel, H., Benisch, J., Helm, B., Vorogushyn, S., and Merz, B.: Fast Urban Inundation Simulation with RIM2D for Flood Risk Assessment and Forecasting, Frontiers in Water, 6, 1310 182, https://doi.org/10.3389/frwa.2024.1310182, 2024.

Arakawa, A. and Lamb, V. R.: Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model, Methods in Computational Physics, 17, 173–265, 1977.

Audusse, E., Bouchut, F., Bristeau, M.-O., Klein, R., and Perthame, B.: A Fast and Stable Well-Balanced Scheme with Hydrostatic Reconstruction for Shallow Water Flows, SIAM Journal on Scientific Computing, 25, 2050–2065, https://doi.org/10.1137/S1064827503431090, 2004.

Backhaus, L. and Stamm, J.: WarpFlow v1.0: A Single-Source CPU/GPU Shallow-Water Solver for Operational City-Scale Urban Flood Modeling, https://doi.org/10.5281/zenodo.22275530, computer software, 2026a.

Backhaus, L. and Stamm, J.: WarpFlow v1.0: Test Suite Data and Reproducibility Package, https://doi.org/10.5281/zenodo.22276867, dataset, 2026b.

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, Journal of Hydrology, 387, 33–45, https://doi.org/10.1016/j.jhydrol.2010.03.027, 2010.

Bermúdez, A. and Vázquez, M. E.: Upwind Methods for Hyperbolic Conservation Laws with Source Terms, Computers & Fluids, 23, 1049–1071, https://doi.org/10.1016/0045-7930(94)90004-3, 1994.

Bundesamt für Kartographie und Geodäsie: WMS Hinweiskarte Starkregengefahren (wms_starkregen), 2026.

Buttinger-Kreuzhuber, A., Konev, A., Horváth, Z., Cornel, D., Schwerdorf, I., Blöschl, G., and Waser, J.: An Integrated GPU-accelerated Modeling Framework for High-Resolution Simulations of Rural and Urban Flash Floods, Environmental Modelling & Software, 156, 105 480, https://doi.org/10.1016/j.envsoft.2022.105480, 2022.

Büttner, M., Alt, C., Kenter, T., Köstler, H., Plessl, C., and Aizinger, V.: Analyzing Performance Portability for a SYCL Implementation of the 2D Shallow Water Equations, The Journal of Supercomputing, 81, 772, https://doi.org/10.1007/s11227-025-07063-7, 2025.

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, Geoscientific Model Development, 16, 977–1008, https://doi.org/10.5194/gmd-16-977-2023, 2023.

Celik, I. B., Ghia, U., Roache, P. J., et al.: Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications, Journal of Fluids Engineering, 130, 078 001, https://doi.org/10.1115/1.2960953, 2008.

Chow, V. T., Maidment, D. R., and Mays, L. W.: Applied Hydrology, McGraw-Hill, New York, 1988.

Chowdhury, A. A. and Kesserwani, G.: LISFLOOD-FP 8.2: GPU-accelerated Multiwavelet Discontinuous Galerkin Solver with Dynamic Resolution Adaptivity for Rapid, Multiscale Flood Simulation, Geoscientific Model Development, 18, 9827–9854, https://doi.org/10.5194/gmd-18-9827-2025, 2025.

Chowdhury, A. A., Kesserwani, G., Rougé, C., and Richmond, P.: GPU-parallelisation of Haar Wavelet-Based Grid Resolution Adaptation for Fast Finite Volume Modelling: Application to Shallow Water Flows, Journal of Hydroinformatics, 25, 1210–1234, https://doi.org/10.2166/hydro.2023.154, 2023.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Dazzi, S., Vacondio, R., Dal Palù, A., and Mignosa, P.: A Local Time Stepping Algorithm for GPU-accelerated 2D Shallow Water Models, Advances in Water Resources, 111, 274–288, https://doi.org/10.1016/j.advwatres.2017.11.023, 2018.

De Almeida, G. A. M. and Bates, P.: Applicability of the Local Inertial Approximation of the Shallow Water Equations to Flood Modeling, Water Resources Research, 49, 4833–4844, https://doi.org/10.1002/wrcr.20366, 2013.

DWA: Arbeitsblatt DWA-A 531: Starkregen in Abhängigkeit von Wiederkehrzeit Und Dauer, Tech. rep., DWA, 2025.

European Parliament and Council: Directive 2007/60/EC of the European Parliament and of the Council of 23 October 2007 on the Assessment and Management of Flood Risks, 2007.

Fuenzalida Alarcón, J., Kusanovic, D., Meneses, R., and Catalán, P. A.: SWEpy: An Open-Source GPU-accelerated Solver for near-Field Inundation and Far-Field Tsunami Modeling, Geoscientific Model Development, 19, 3953, https://doi.org/10.5194/gmd-19-3953-2026, 2026.

Guo, K., Guan, M., and Yu, D.: Urban Surface Water Flood Modelling – a Comprehensive Review of Current Models and Future Challenges, Hydrology and Earth System Sciences, 25, 2843–2860, https://doi.org/10.5194/hess-25-2843-2021, 2021.

Harris, C. R., Millman, K. J., van der Walt, S. J., et al.: Array Programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Hlal, M., Baraka Munyaka, J.-C., Chenal, J., Azmi, R., Diop, E. B., Bounabi, M., Ebnou Abdem, S. A., Almouctar, M. A. S., and Adraoui, M.: Digital Twin Technology for Urban Flood Risk Management: A Systematic Review of Remote Sensing Applications and Early Warning Systems, Remote Sensing, 17, 3104, https://doi.org/10.3390/rs17173104, 2025.

Hobley, D. E. J., Adams, J. M., Nudurupati, S. S., Hutton, E. W. H., Gasparini, N. M., Istanbulluoglu, E., and Tucker, G. E.: Creative Computing with Landlab: An Open-Source Toolkit for Building, Coupling, and Exploring Two-Dimensional Numerical Models of Earth-Surface Dynamics, Earth Surface Dynamics, 5, 21–46, https://doi.org/10.5194/esurf-5-21-2017, 2017.

Horton, R. E.: An Approach toward a Physical Interpretation of Infiltration-Capacity, Soil Science Society of America Journal, 5, 399–417, https://doi.org/10.2136/sssaj1941.036159950005000C0075x, 1941.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F.: Taichi: A Language for High-Performance Computation on Spatially Sparse Data Structures, ACM Transactions on Graphics (TOG), 38, 201:1–201:16, https://doi.org/10.1145/3355089.3356506, 2019.

Junghänel, T., Ostermöller, J., and Deutschländer, T.: Grids of Return Periods of Heavy Precipitation (Design Precipitation) over Germany (KOSTRA-DWD), https://doi.org/10.5676/DWD/KOSTRA-DWD-2020, 2023.

Kundzewicz, Z. W., Kanae, S., Seneviratne, S. I., Handmer, J., Nicholls, N., Peduzzi, P., Mechler, R., Bouwer, L. M., Arnell, N., Mach, K., Muir-Wood, R., Brakenridge, G. R., Kron, W., Benito, G., Honda, Y., Takahashi, K., and Sherstyukov, B.: Flood Risk and Climate Change: Global and Regional Perspectives, Hydrological Sciences Journal, 59, 1–28, https://doi.org/10.1080/02626667.2013.857411, 2014.

Kuzmin, D.: Flux-Corrected Transport, in: Flux-Corrected Transport: Principles, Algorithms, and Applications, edited by Kuzmin, D., Löhner, R., and Turek, S., pp. 1–32, Springer, Dordrecht, https://doi.org/10.1007/978-94-007-4038-9_1, 2012.

Lam, S. K., Pitrou, A., and Seibert, S.: Numba: A LLVM-based Python JIT Compiler, in: Proc. Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM'15), https://doi.org/10.1145/2833157.2833162, 2015.

LeVeque, R. J.: Finite Volume Methods for Hyperbolic Problems, Cambridge University Press, https://doi.org/10.1017/CBO9780511791253, 2002.

Lo, T., Malmström, A., and Juhola, S.: Critical assessment on digital twins implementation for flood risk management and governance: A systematic literature review, International Journal of Disaster Risk Reduction, 140, 106 165, https://doi.org/10.1016/j.ijdrr.2026.106165, 2026.

https://doi.org/10.5194/egusphere-2026-5446
Preprint. Discussion started: 5 November 2026
© Author(s) 2026. CC BY 4.0 License.

Lukáčová-Medvid'ová, M. and Teschke, U.: Comparison Study of Some Finite Volume and Finite Element Methods for the Shallow Water Equations with Bottom Topography and Friction Terms, ZAMM, 86, 874–891, https://doi.org/10.1002/zamm.200510280, 2006.

Mandal, D., Zou, L., Wadhwa, A., Wilkho, R. S., Cai, Z., Zhou, B., Ye, X., Newman, G., Gharaibeh, N., and Güneralp, B.: FlowsDT: A geospatial digital twin for navigating urban flood dynamics, Computers, Environment and Urban Systems, 126, 102 414, https://doi.org/10.1016/j.compenvurbsys.2026.102414, 2026.

Morales-Hernández, M., Sharif, M. B., Kalyanapu, A., et al.: TRITON: A Multi-GPU Open Source 2D Hydrodynamic Flood Model, Environmental Modelling & Software, 141, 105 034, https://doi.org/10.1016/j.envsoft.2021.105034, 2021.

Neelz, S. and Pender, G.: Benchmarking the Latest Generation of 2D Hydraulic Modelling Packages, Tech. Rep. SC120002, Environment Agency, Bristol, UK, 2013.

Oberauer, M., Lehmann, B., Buttinger-Kreuzhuber, A., and Waser, J.: Methodological Approaches for Numerical Integration of Culverts in 2D Surface Models in Urban Flood Analysis, Journal of Hydrology, 665, 134 728, https://doi.org/10.1016/j.jhydrol.2025.134728, 2026.

Okuta, R., Unno, Y., Nishino, D., Hido, S., and Loomis, C.: CuPy: A NumPy-compatible Library for NVIDIA GPU Calculations, in: Proc. Workshop on ML Systems, NeurIPS, 2017.

Ritter, A.: Die Fortpflanzung Der Wasserwellen, Zeitschrift des Vereines Deutscher Ingenieure, 36, 947–954, 1892.

Roache, P. J.: Perspective: A Method for Uniform Reporting of Grid Refinement Studies, Journal of Fluids Engineering, 116, 405–413, https://doi.org/10.1115/1.2910291, 1994.

Sampson, C. C., Bates, P. D., Neal, J. C., and Horritt, M. S.: An Automated Routing Methodology to Enable Direct Rainfall in High Resolution Shallow Water Models, Hydrological Processes, 27, 467–476, https://doi.org/10.1002/hyp.9515, 2013.

Schaefer, J. T.: The Critical Success Index as an Accuracy Measure, Weather and Forecasting, 5, 570–575, https://doi.org/10.1175/1520-0434(1990)005<0570:TCSIAA>2.0.CO;2, 1990.

Schubert, J. E., Sanders, B. F., Smith, M. J., and Wright, N. G.: Unstructured Mesh Generation and Landcover-Based Resistance for Hydrodynamic Modeling of Urban Flooding, Advances in Water Resources, 31, 1603–1621, https://doi.org/10.1016/j.advwatres.2008.07.012, 2008.

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, Geoscientific Model Development, 16, 2391–2413, https://doi.org/10.5194/gmd-16-2391-2023, 2023.

Shaw, J., Kesserwani, G., Neal, J., Bates, P., and Sharifian, M. K.: LISFLOOD-FP 8.0: The New Discontinuous Galerkin Shallow-Water Solver for Multi-Core CPUs and GPUs, Geoscientific Model Development, 14, 3577–3602, https://doi.org/10.5194/gmd-14-3577-2021, 2021.

Stoker, J. J.: Water Waves: The Mathematical Theory with Applications, Interscience, 1957.

Tellez-Alvarez, J., Gómez, M., Russo, B., and Amezaga-Kutija, M.: Numerical and Experimental Approaches to Estimate Discharge Coefficients and Energy Loss Coefficients in Pressurized Grated Inlets, Hydrology, 8, https://doi.org/10.3390/hydrology8040162, 2021.

Toro, E. F.: Shock-Capturing Methods for Free-Surface Shallow Flows, Wiley, 2001.

Xia, X. and Liang, Q.: A New Efficient Implicit Scheme for Discretising the Stiff Friction Terms in the Shallow Water Equations, Advances in Water Resources, 117, 87–97, https://doi.org/10.1016/j.advwatres.2018.05.004, 2018.

Zalesak, S. T.: Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids, Journal of Computational Physics, 31, 335–362, https://doi.org/10.1016/0021-9991(79)90051-2, 1979.