

The SERGHEI model and its core shallow water solver

Mario **Morales-Hernández** (Universidad de Zaragoza, Spain)

Ilhan **Özgen-Xian** (Lawrence Berkeley National Lab, USA)

Daniel **Caviedes-Voullième** (Forschungszentrum Jülich, Germany)



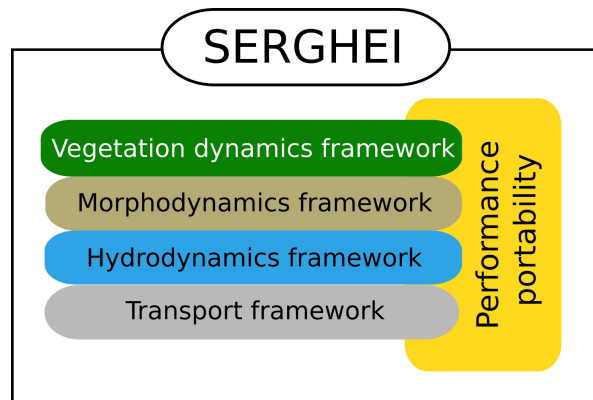
1542

Universidad
Zaragoza



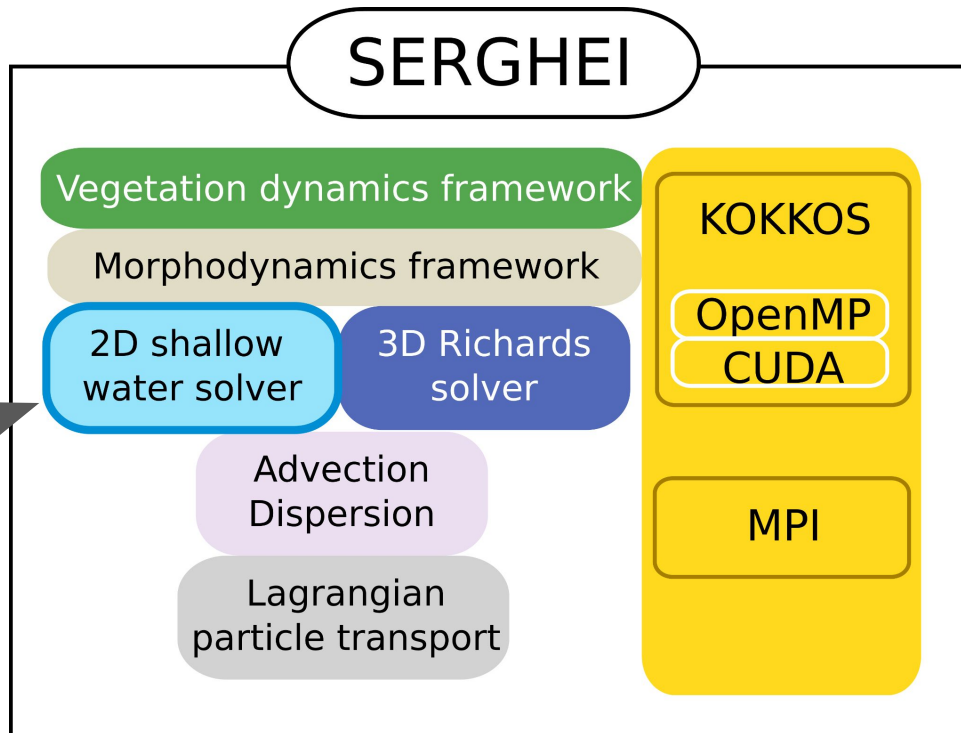
Introducing SERGHEI

Simulation Environment for Geomorphology, Hydrodynamics, and Ecohydrology in Integrated form



Fully operational

Most modules currently
under different stages
of development



Why SERGHEI?

- Simple and easy to configure and use
- Well documented, maintained and (in the near future) open access in GitLab
- Able to run on different HPC platforms and/or personal computers/laptops

Ensures **performance portability** (Kokkos)
and future-proofing

Why is this interesting for geoscientists?

Performance Portability

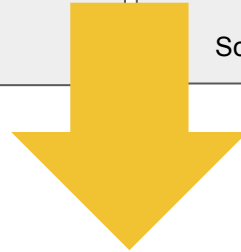
“An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required.”

Source: <https://performanceportability.org/>

Future-proofing

“To design software, a computer, etc. so that it can still be used in the future, even when technology changes.”

Source: Cambridge Dictionary



- Keep code readable, maintainable, and compact, avoiding code duplications.
- Use your code on CPUs and GPUs, from laptops to workstations, clusters and HPC systems.
- No need to re-code nor significantly adapt your code.
- Ensure that development efforts will be useful in incoming systems, with Nvidia, AMD (e.g. El Capitan @ LLNL) and Intel GPU cards.



- Open Source C++ library
- Developed at Sandia National Lab (USA)
- Programming model that targets all major HPC platforms
- Avoid the cost of porting codes
- It combines data structures with parallel execution in a fashion way
 - Memory spaces and layouts
 - Execution patterns (for, reduction, atomic...)
 - Data allocation, transfer and execution spaces
- Math libraries and tools based on Kokkos
 - Profiling
 - Debugging

Kokkos

Serial C/C++ (single CPU execution space)

```
inline void computeNewState(State &state , const Domain &dom, const SourceSinkData &ss) {  
  for (int j=0; j<dom.ny; j++) {  
    for (int i=0; i<dom.nx; i++) {  
      int ii = dom.getHaloExtension(i,j,dom.nx);  
  
      real z=state.z(ii);  
      real hold=state.h(ii);  
      real huold=state.hu(ii);  
      real hvold=state.hv(ii);  
      bool nodata=state.isnodata(ii);  
  
      // lots of computationally intensive code  
    }  
  }  
}
```

This code works for single
CPUs, OpenMP (multi-CPU)
and CUDA (GPUs)



Implementing Kokkos is
minimally invasive (if parallelism
is already well-exposed).

```
inline void computeNewState(State &state , const Domain &dom, const SourceSinkData &ss) {  
  Kokkos::parallel_for( dom.nCellDomain , KOKKOS_LAMBDA (int iGlob) {  
    int ii = dom.getIndex(iGlob);  
  
    real z=state.z(ii);  
    real hold=state.h(ii);  
    real huold=state.hu(ii);  
    real hvold=state.hv(ii);  
    bool nodata=state.isnodata(ii);  
  
    // lots of computationally intensive code  
  })  
}
```

Kokkos C++ (execution space determined at compilation)

For multi-GPUs and/or multi-nodes, we still rely on MPI

HPC systems for testing



NeRSC



OAK RIDGE
National Laboratory



JÜLICH
Forschungszentrum

SERGHEI 2D shallow water solver

2D SWE: Mass and momentum in x,y-directions

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S}_s + \mathbf{S}_b + \mathbf{S}_f$$

$$\mathbf{U} = \begin{pmatrix} h \\ q_x \\ q_y \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} q_x \\ \frac{q_x^2}{h} + \frac{1}{2}gh^2 \\ \frac{q_x q_y}{h} \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} q_y \\ \frac{q_x q_y}{h} \\ \frac{q_y^2}{h} + \frac{1}{2}gh^2 \end{pmatrix}$$

$$\mathbf{S}_s = \begin{pmatrix} s + r - f \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{S}_b = \begin{pmatrix} 0 \\ -gh \frac{\partial z}{\partial x} \\ -gh \frac{\partial z}{\partial y} \end{pmatrix} \quad \mathbf{S}_f = \begin{pmatrix} 0 \\ -\frac{gn^2}{h^{7/3}} q_x \sqrt{q_x^2 + q_y^2} \\ -\frac{gn^2}{h^{7/3}} q_y \sqrt{q_x^2 + q_y^2} \end{pmatrix}$$

ARoe finite volume explicit upwind scheme

$$\mathbf{U}_i^* = \mathbf{U}_i^n - \frac{\Delta t}{\Delta x} \sum_{k=1}^4 \sum_{m=1}^3 \frac{\tilde{\lambda}^-}{\tilde{\lambda}} \left[(\tilde{\lambda} \tilde{\alpha} - \tilde{\beta}_b) \tilde{\mathbf{e}} \right]_{m,k}^n$$

$$\mathbf{U}_i^{n+1} = \mathcal{F}(\mathbf{U}_i^n, \mathbf{U}_i^*) + (s + r - f)_i^n \Delta t$$

$$\mathcal{F}^1 = h^* \quad \mathcal{F}^2 = -(q_x^*) \left(\frac{1 - \sqrt{1 + 4S_f}}{2S_f} \right) \quad \mathcal{F}^3 = -(q_y^*) \left(\frac{1 - \sqrt{1 + 4S_f}}{2S_f} \right)$$

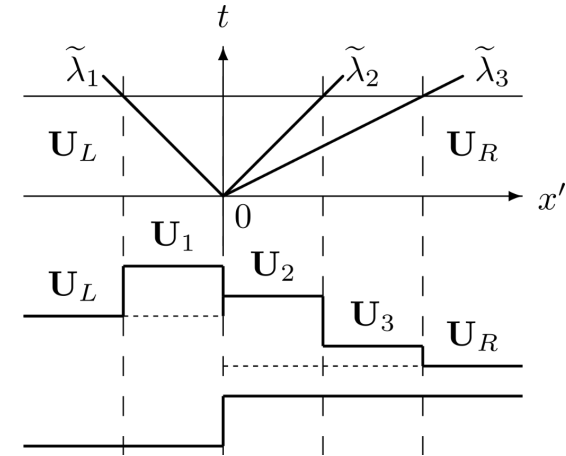
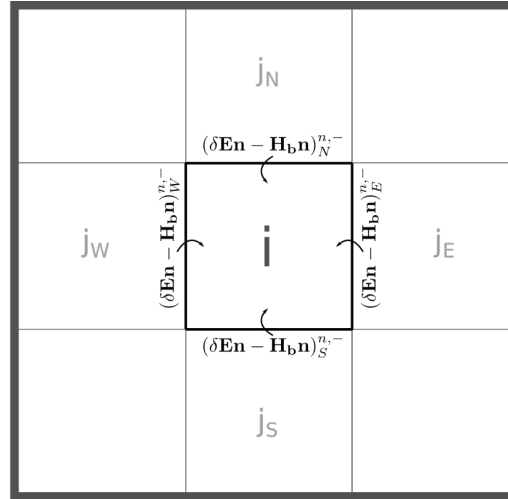
$$S_f = \frac{\Delta t g n^2 \sqrt{(q_x^*)^2 + (q_y^*)^2}}{(h^n)^{7/3}}$$

Time-step restricted by CFL condition

$$\Delta t = \text{CFL} \min_i \left\{ \frac{\Delta x}{\left| \frac{q_x}{h} \right|_i + \sqrt{gh_i}, \left| \frac{q_y}{h} \right|_i + \sqrt{gh_i}} \right\} \quad \text{CFL} \leq 0.5$$

SERGHEI shallow water solver

- Cartesian (squared) mesh
- Mass conservative
- Exact balance between fluxes and source terms
- Accurate wet/dry treatment using the RP states

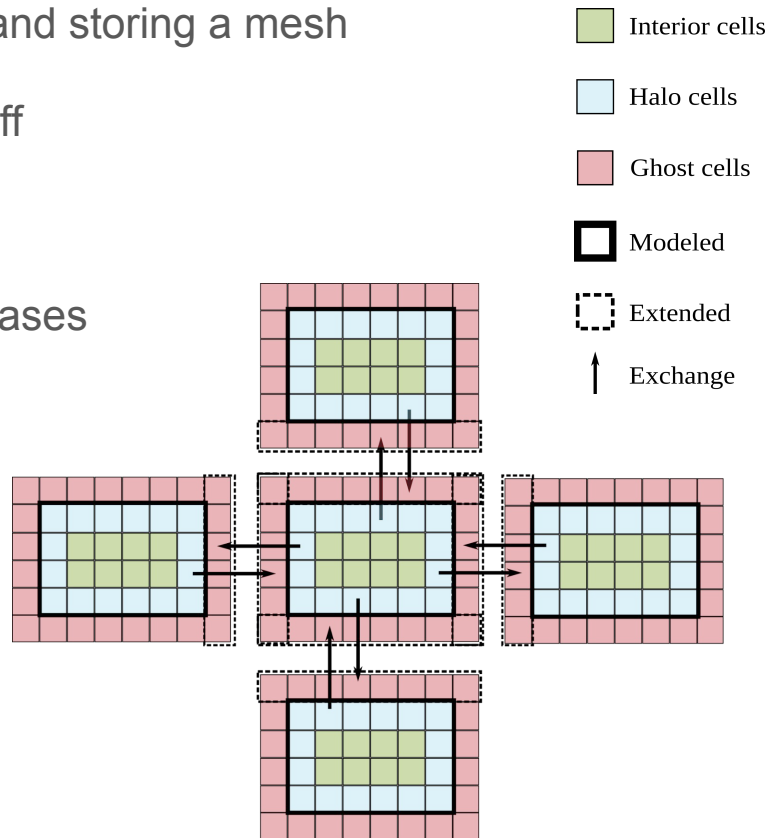


SERGHEI shallow water solver features

- DEM as computational mesh → avoid building and storing a mesh
- Roughness map → spatially distributed
- Temporally and spatially distributed rainfall/runoff
- Temporally and spatially distributed infiltration
- Source/sinks
- Typical boundary conditions
- Tested against most available benchmark test cases

MPI domain decomposition

- 2D block-wise domain decomposition
- Ghost cells containing neighbour information
- Exchange information every time step
- Non-blocking MPI

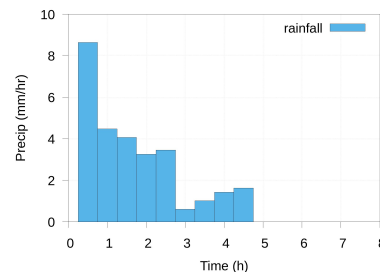
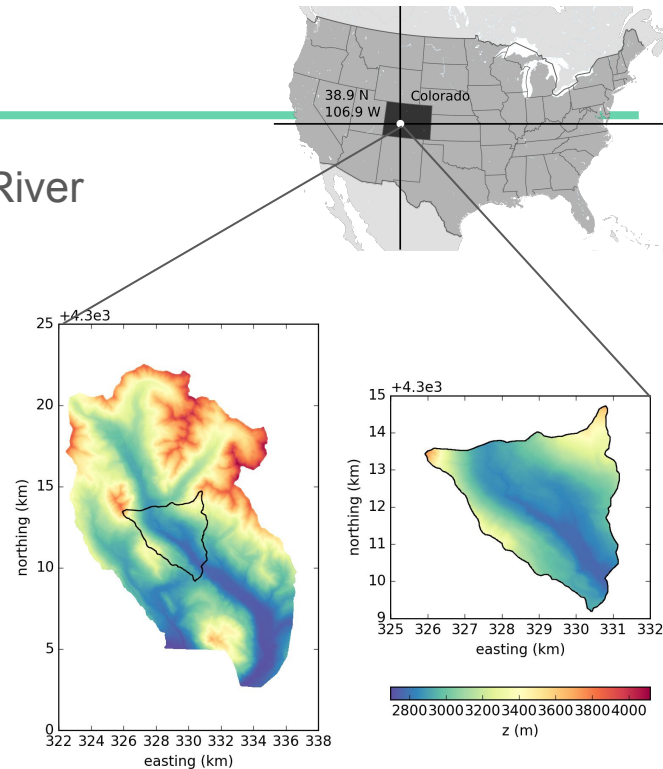


Examples: Lower Triangle catchment

- High-elevation, high-gradient subcatchment of the East River watershed
- 14.84 km² with 0.5 DEM resolution → 118M grid cells
- 3h rainfall
- Constant Manning's roughness: 0.03 m/s^{1/3}
- Critical depth at outlet boundary

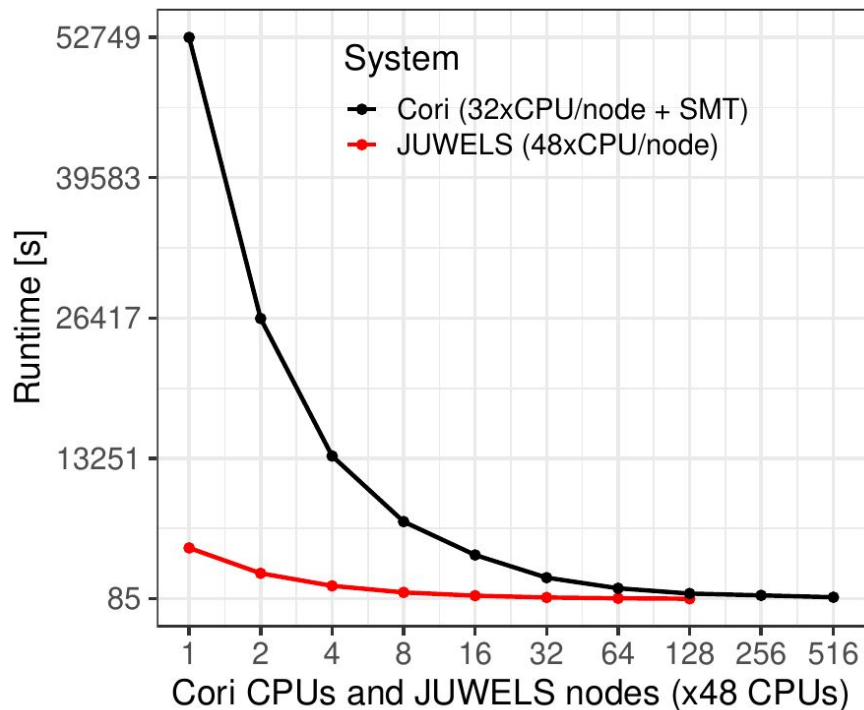
OBJECTIVE: Evaluate SERGHEI performance

- ❑ Multiple CPUs (MPI + OpenMP)
- ❑ Multiple GPUs (MPI + CUDA)

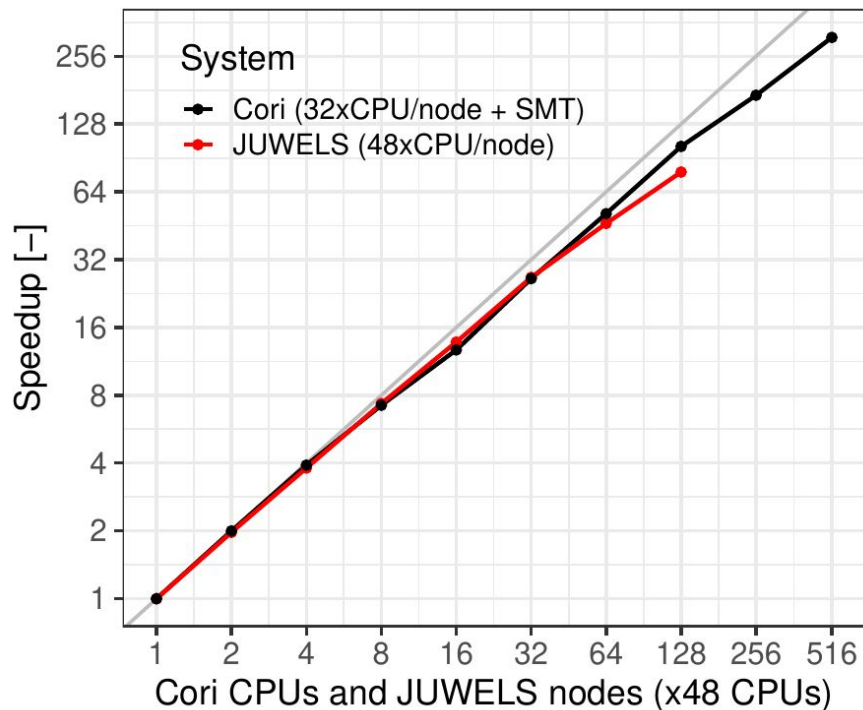


Data provided by work funded by the US DOE (BER), Award no. DE-AC02-05CH11231.

Lower Triangle catchment: CPU scaling

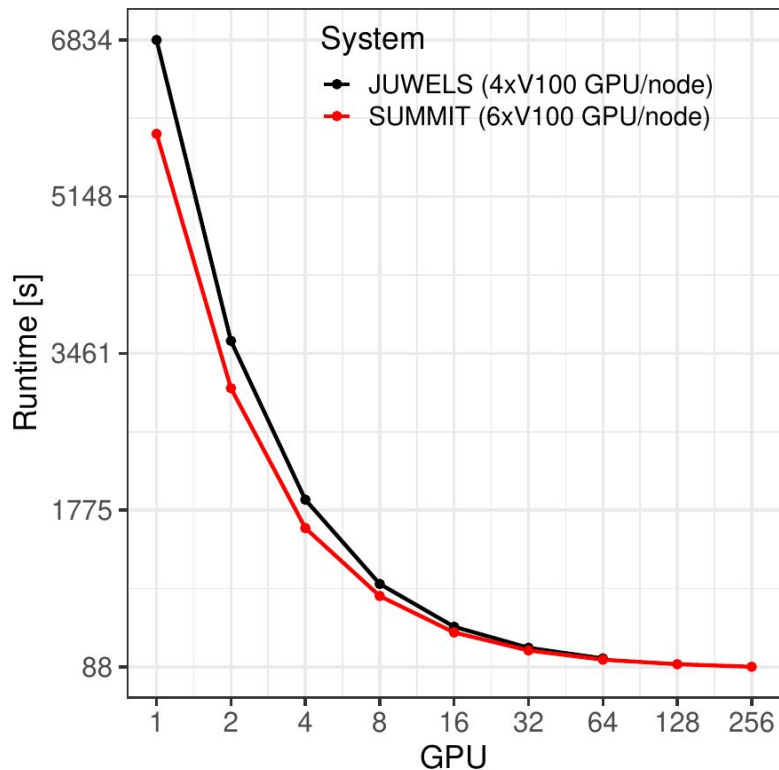


Runs in NERSC Cori with 1 to 516 CPUs
Runs in Juwels with 48 to 6144 CPUs

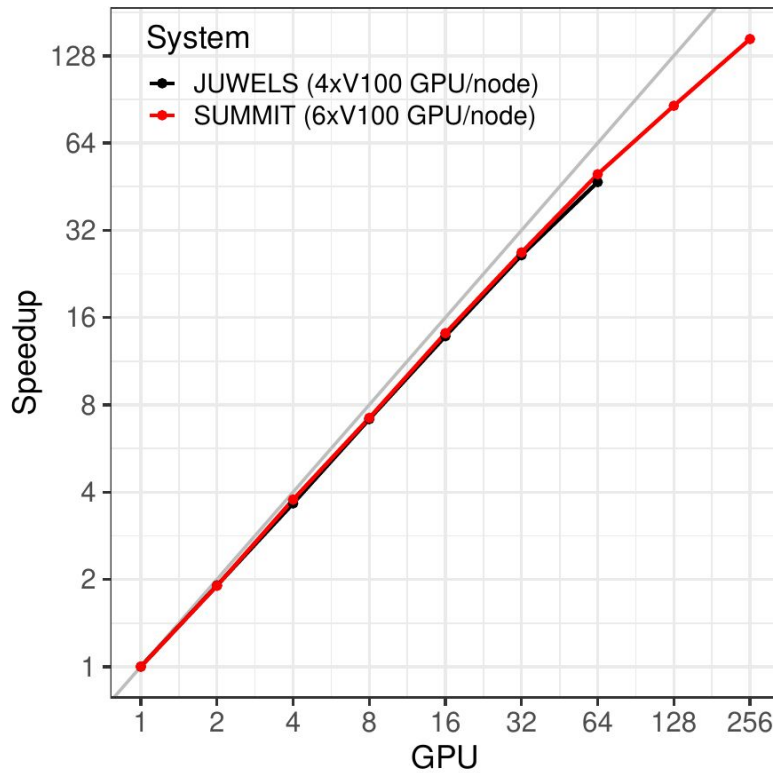


Practically the same strong scaling
without any tuning

Lower Triangle catchment: GPU scaling



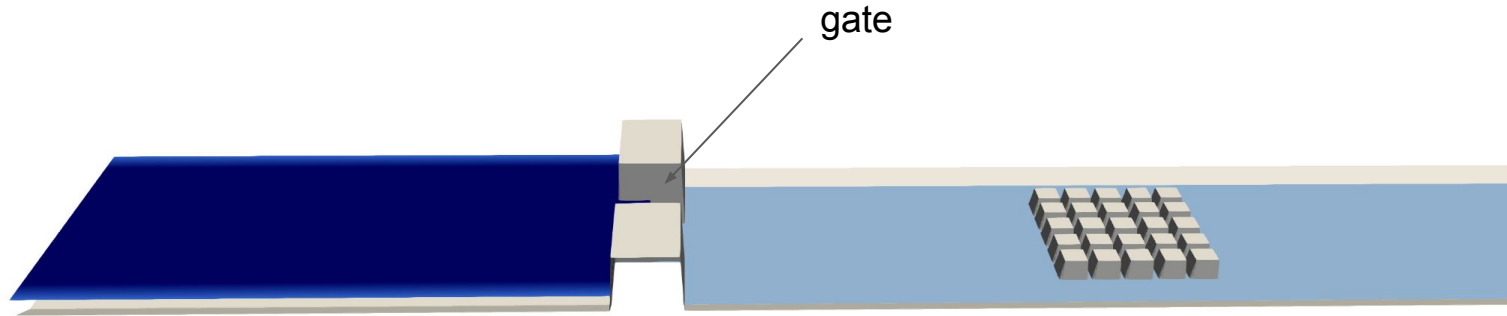
Summit is somewhat faster,
likely due to node configuration



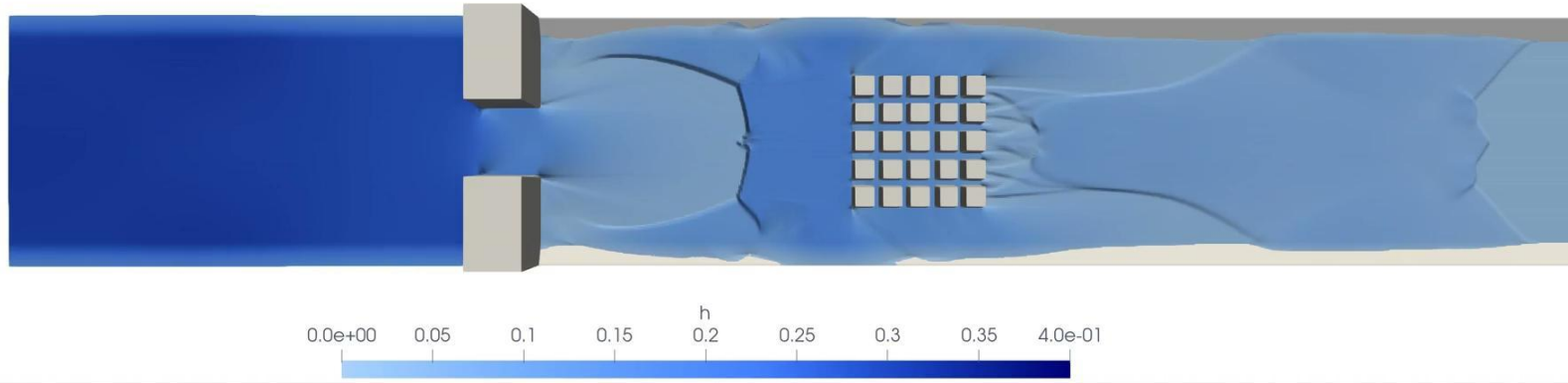
Practically the same strong
scaling without any tuning

Examples: Experimental dam-break in urban area

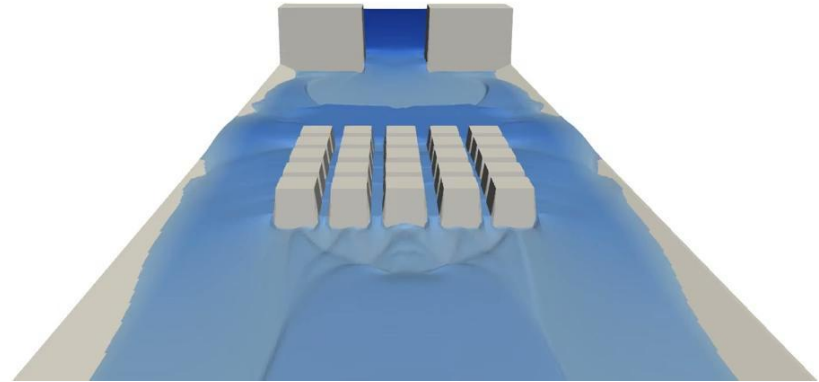
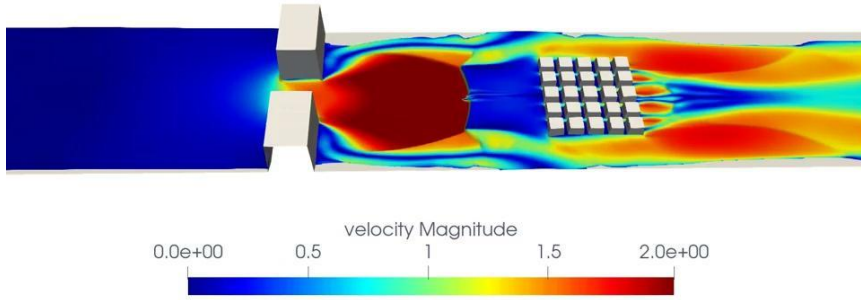
- Experiment carried out at UCL (Belgium)
- 36m long channel with a rectangular cross section (except near the banks)
- 25 (5x5) square blocks located in the channel of size 0.3x0.3 aligned with the flow
- A gate separates upstream level of 0.40m from a downstream level of 0.011m
- Constant Manning's roughness: 0.01 m/s^{1/3}
- Simulation time: 15s
- Low resolution: 324000 cells, runtime: 9.2 s using a Nvidia GeForce RTX 3070 (500 €)
- High resolution: 20736000 cells, runtime: 445 s using Nvidia Volta100 (10000 €)



Examples: Experimental dam-break in urban area



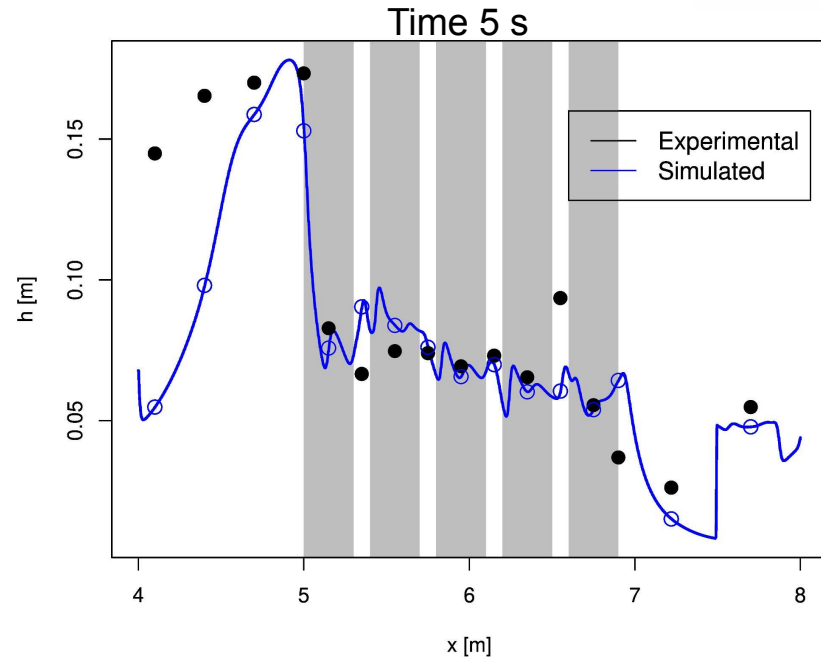
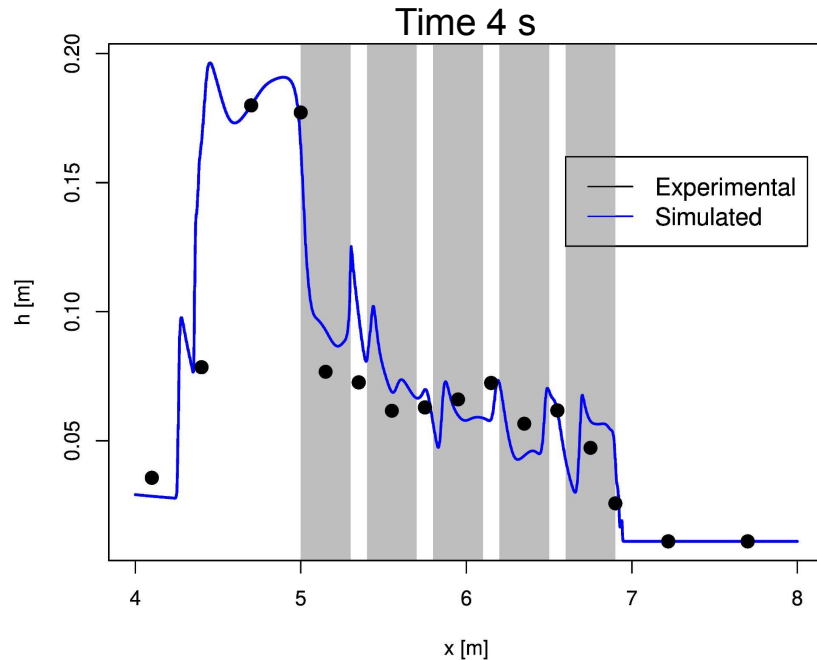
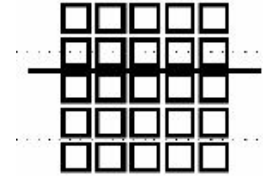
Time: 9.2 s



Click on the figure to launch video

Examples: Experimental dam-break in urban area

Comparison against experimental data along a profile



The SERGHEI model and its core shallow water solver

Mario **Morales-Hernández** (Universidad de Zaragoza, Spain)

Ilhan **Özgen-Xian** (Lawrence Berkeley National Lab, USA)

Daniel **Caviedes-Voullième** (Forschungszentrum Jülich, Germany)



1542

Universidad
Zaragoza

