A fully unsplit wave propagation algorithm for shallow water flows on GPUs

Donna Calhoun (Boise State University)

Carsten Burstedde, Univ. of Bonn, Germany

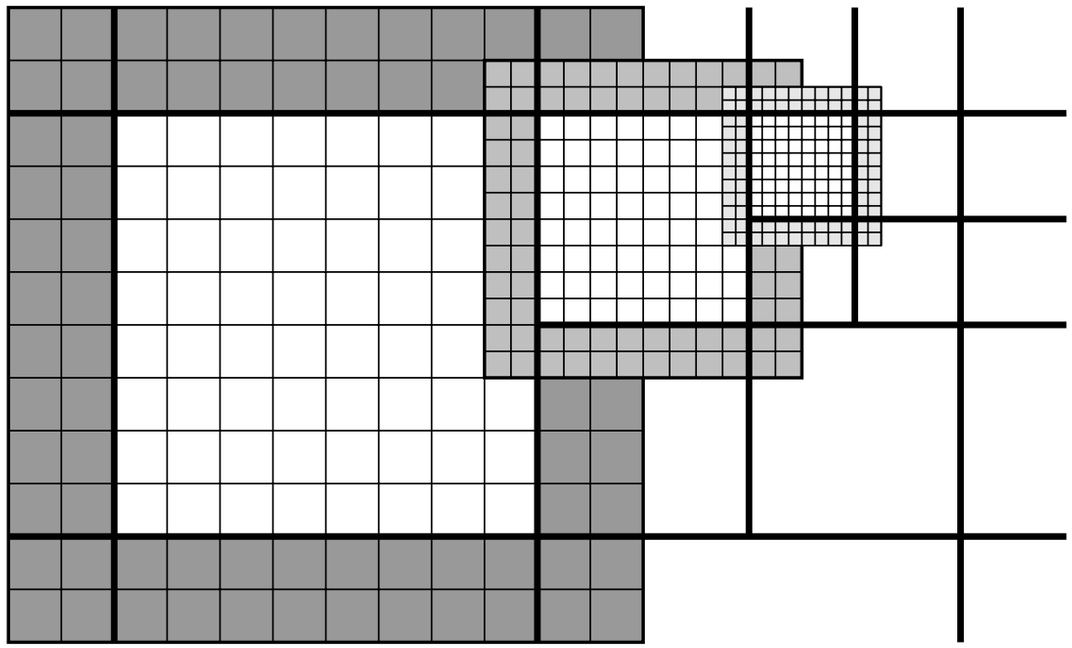
Other collaborators : M. Shih (NYU); S. Aiton (BSU); X. Qin (Univ. of Washington; Waymo); R. J. LeVeque (Univ. of Washington); K. Mandli (Columbia University) and many others.

SIAM Parallel Processing

Feb 11-12, 2020

Seattle, Washington

ForestClaw



In the “clawpatch” patch (used for finite volume solvers), each p4est quadrant is occupied by a single logically Cartesian grid, stored in contiguous memory, including ghost cells.

www.forestclaw.org

ForestClaw is a p4est PDE layer.

- Written mostly in object-oriented C
- Core routines are agnostic as to patch data, solvers used, etc.
- Most aspects of the PDE layer, including type of patch used, solver, interpolation and averaging, ghost-filling, can be customized
- Support for legacy codes
- Several extensions include Clawpack extension, GeoClaw, Ash3d and others.
- Current solvers are designed for FV meshes

ForestClaw

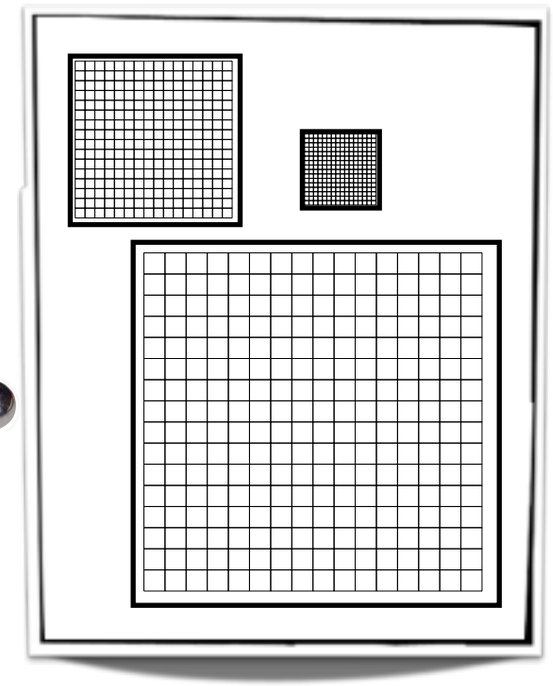
Core routines

p4est

ForestClaw

Time stepping, dynamic grid management, input/ output.

Patches



Tasks (tagging, building patches, etc) customized through use of function pointers stored in virtual tables

Extensions

clawpack4.6

clawpack5

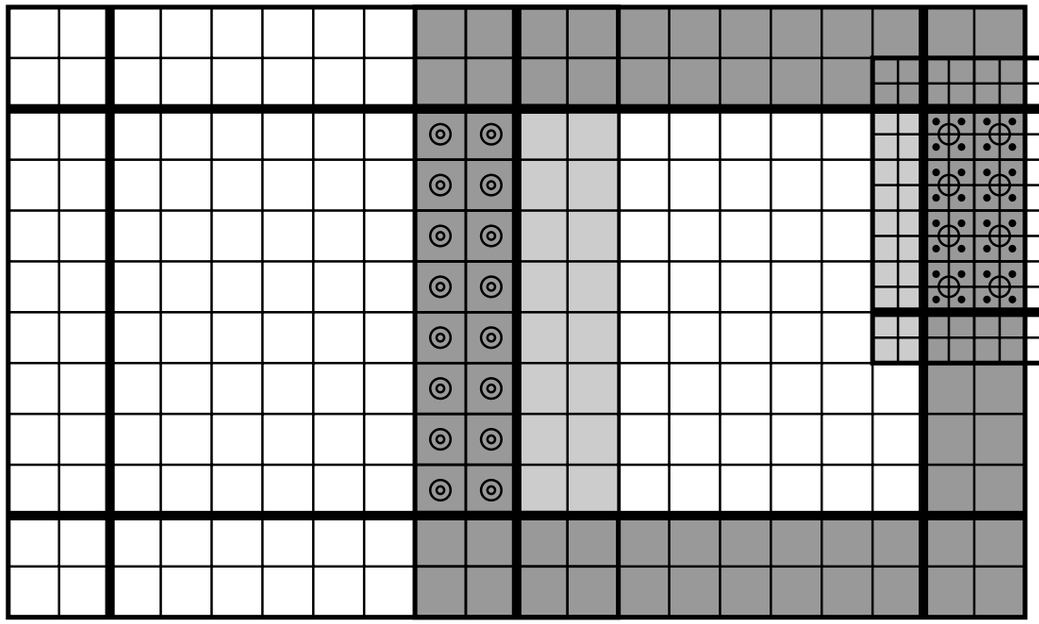
geoclaw

ash3d

ForestClaw is mostly C; solvers extension libraries left largely untouched (in original Fortran)

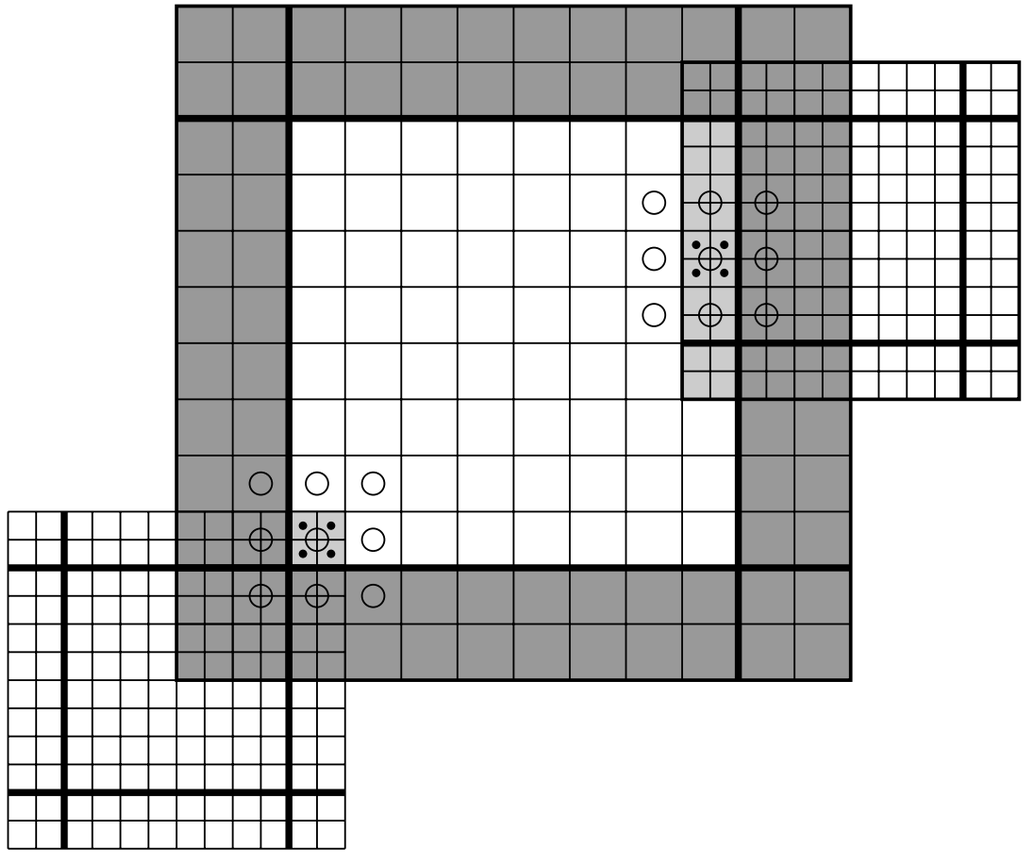
Filling ghost cells

Assume valid data in the interior of each patch



Step 1 : Averaging or copying to coarse ghost regions

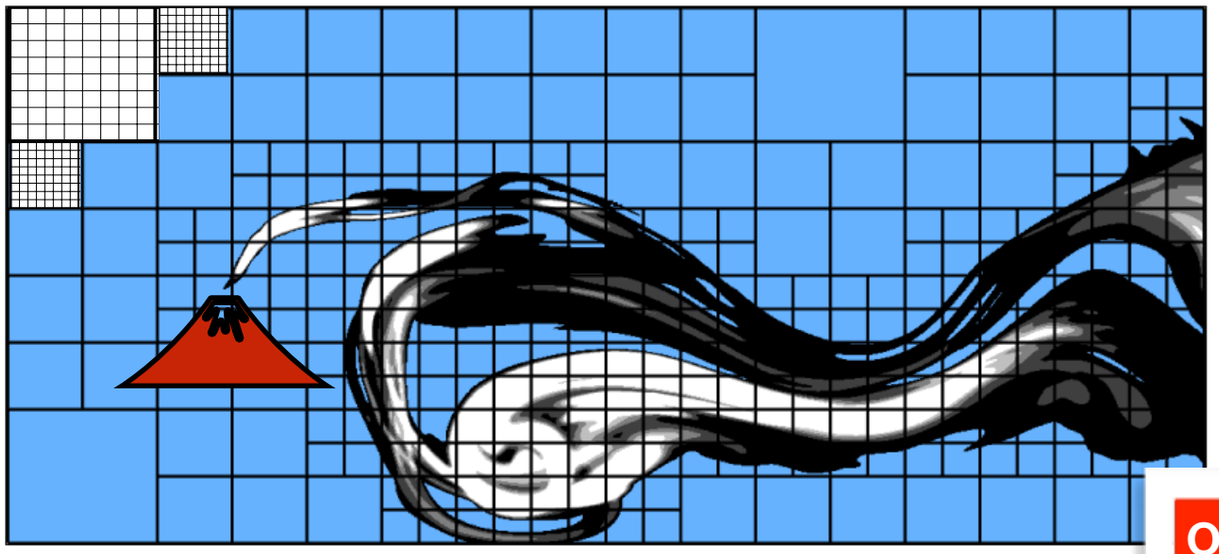
Note : 8x8 grids show here are too small to use in practice

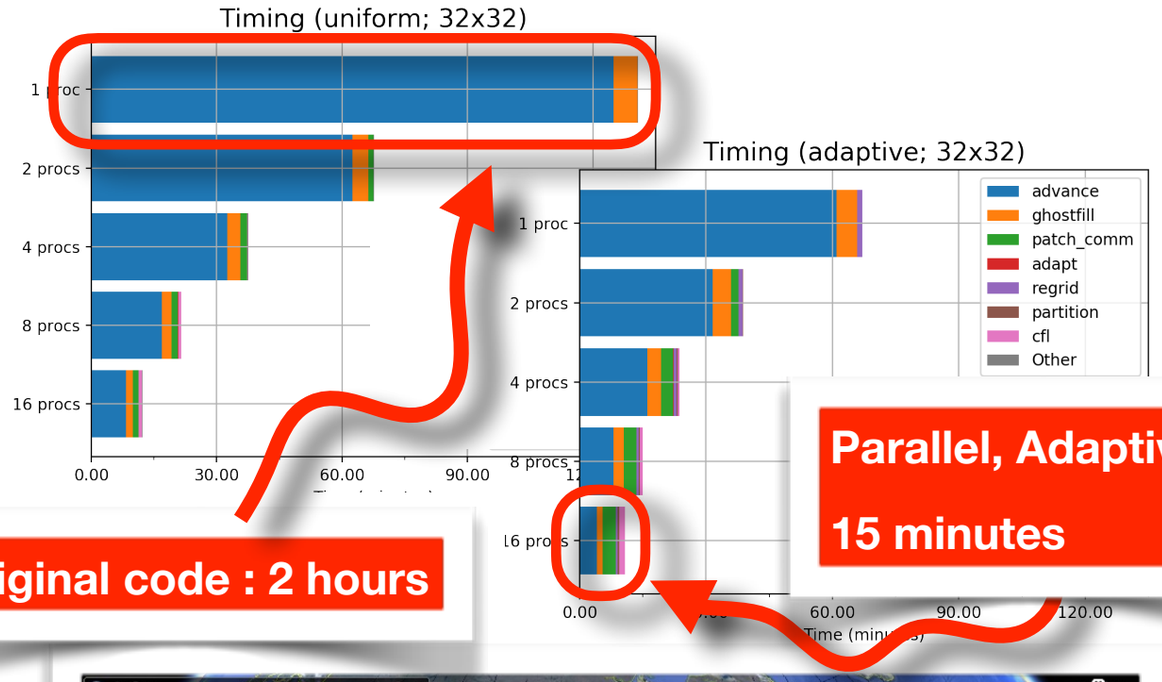


Step 2 : Interpolation to fine ghost regions, using coarse grid ghost regions

Each grid (or "leaf", in p4est terminology) has one of more layers of ghost cells used for communication between grids

Volcanic ash transport

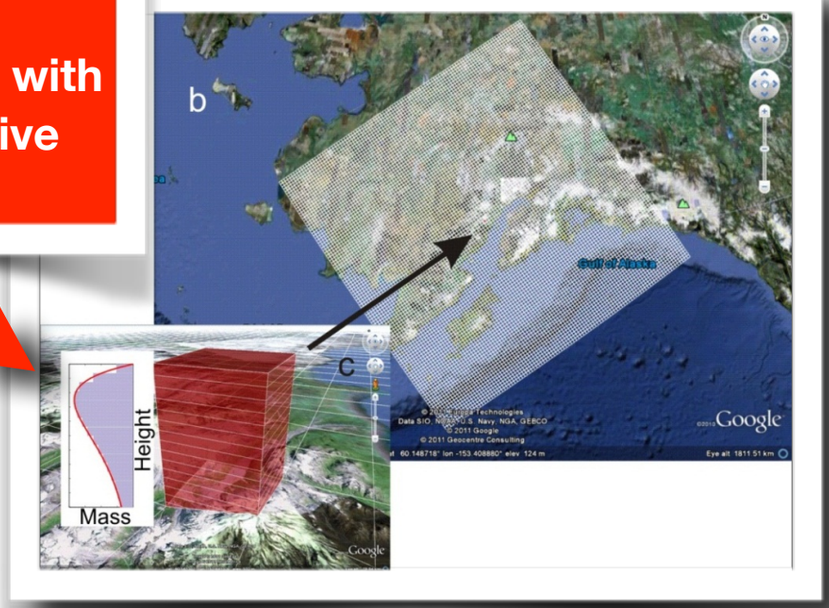


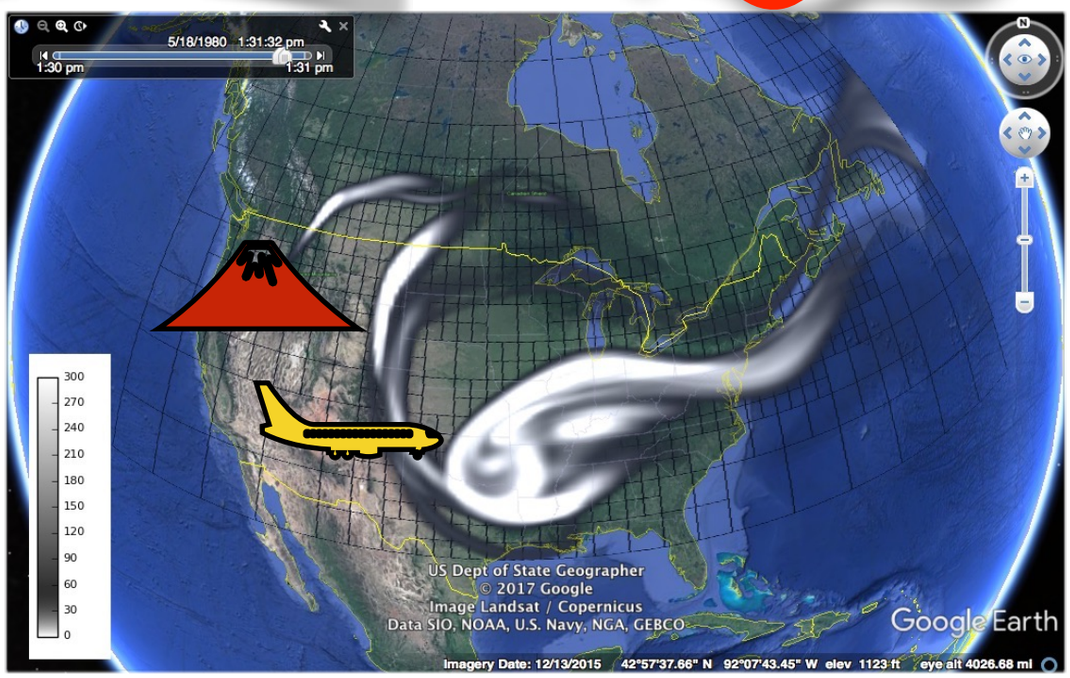


Original code : 2 hours

Parallel, Adaptive: 15 minutes

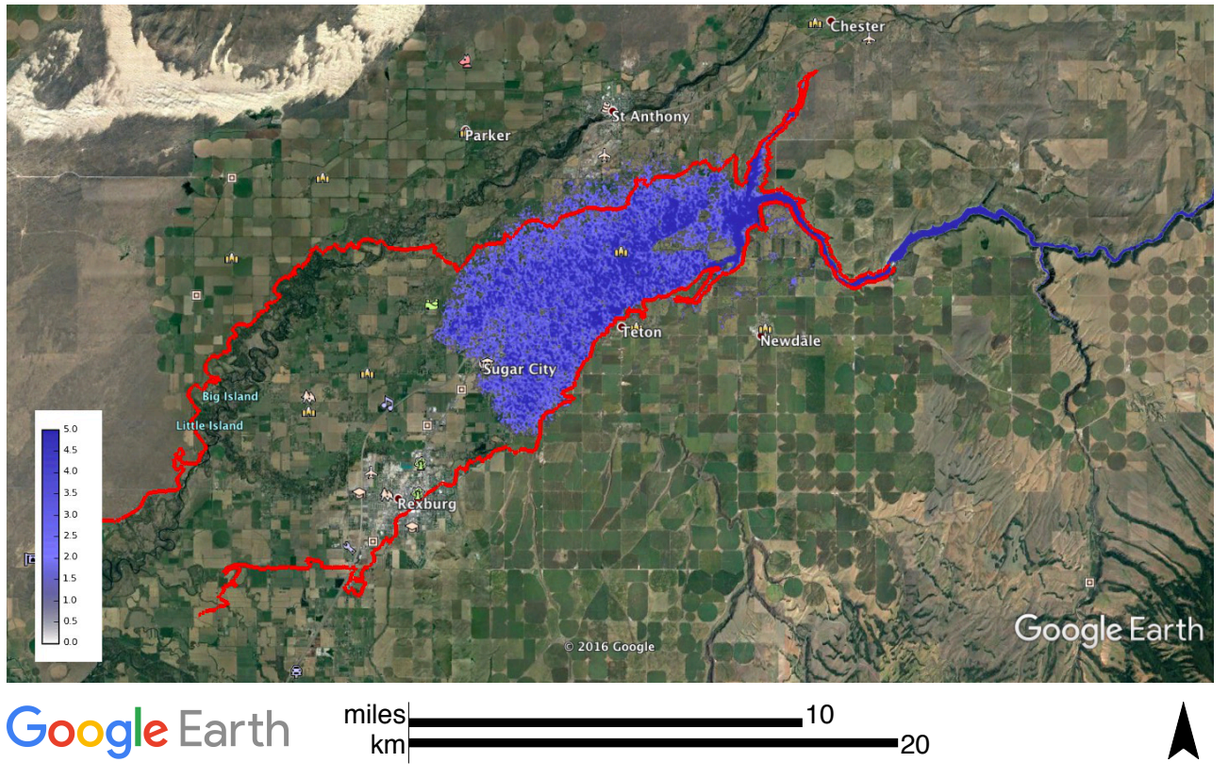
Extend existing volcanic ash transport model with a parallel, adaptive capabilities

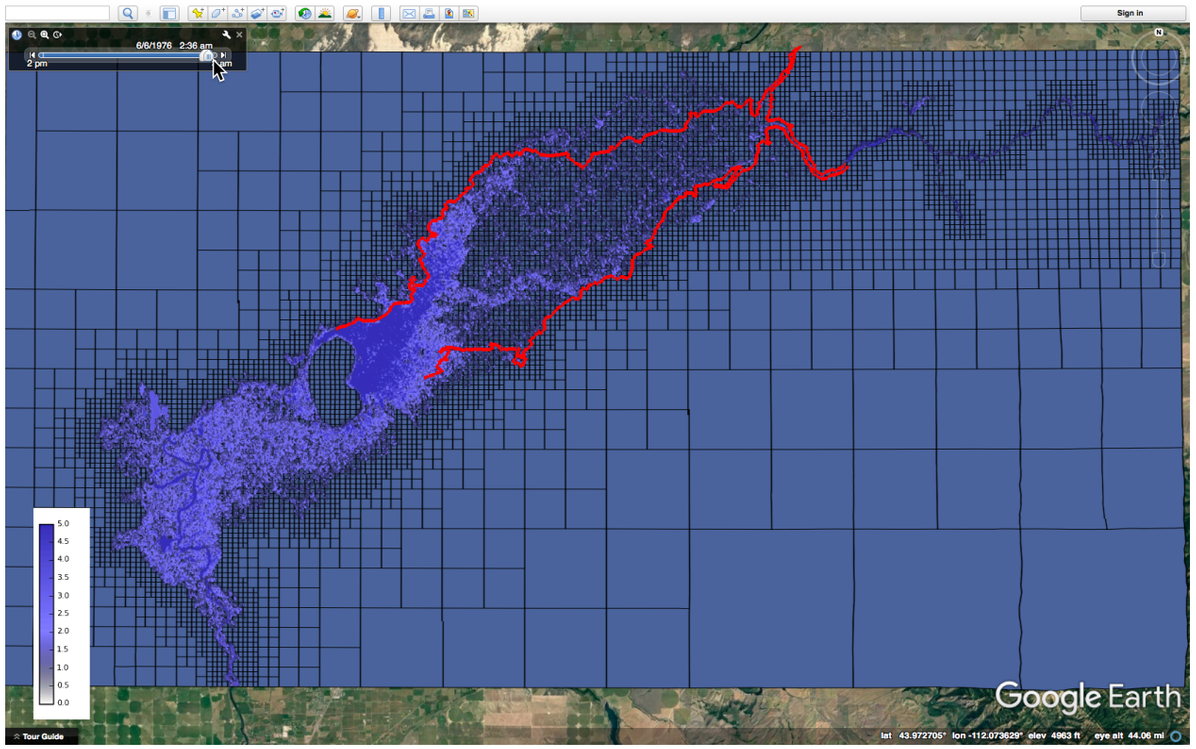




Volcanic ash transport using Ash3d (H. Schwaiger, USGS) extension of ForestClaw

Natural Hazards Modeling





Teton Dam failure (1976, Eastern Idaho) using GeoClaw (D. George, R. J. LeVeque, K. Mandli, M. Berger) extension of ForestClaw

Clawpack

- The wave propagation algorithm (R. J. LeVeque) is a second order, finite volume scheme for solving hyperbolic problems on logically Cartesian meshes
- Computes “waves” and “fluctuations at cell interfaces
- High resolution is achieved using wave limiters
- Scheme can be implemented as a dimensionally split method or as an unspilt method
- Unsplit method requires one normal Riemann solve and four transverse solves per direction.
- Solver is implemented in the Clawpack package, GeoClaw and ForestClaw, plus others (www.clawpack.org, www.geoclaw.org)

Why use GPUs?

- Migration of Clawpack 4.x to Clawpack 5.x was done in large part to re-order data layout for better cache re-use
- Polyhedral models for optimizing compilers for stencil calculations (Cathie Olschanowsky (BSU), Eddie Davis (BSU, Vulcan), ...)
- C. de Resende Ferreira, *Vectorization and Patch-Based Adaptive Mesh Refinement for Finite Volume Solvers*, PhD thesis, Technical University Munich, July 17 2019.
- Interface libraries such as Kokkos (LLNL) let the experts decide which data layout is best for which hardware platform
- GPUs are expensive ($$$). Just use OpenMP

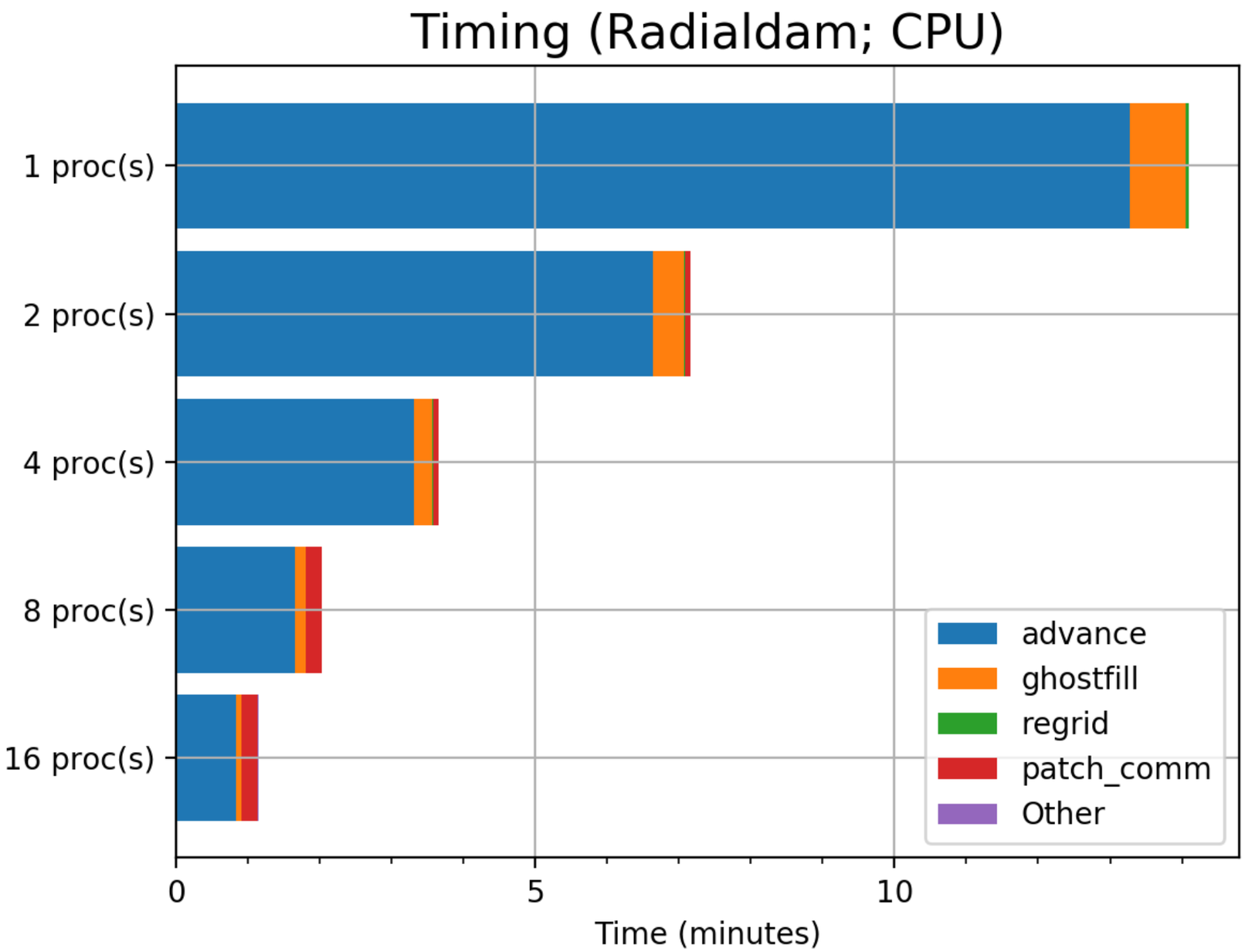
"You are leaving cache lines on the table"

Why use GPUs?

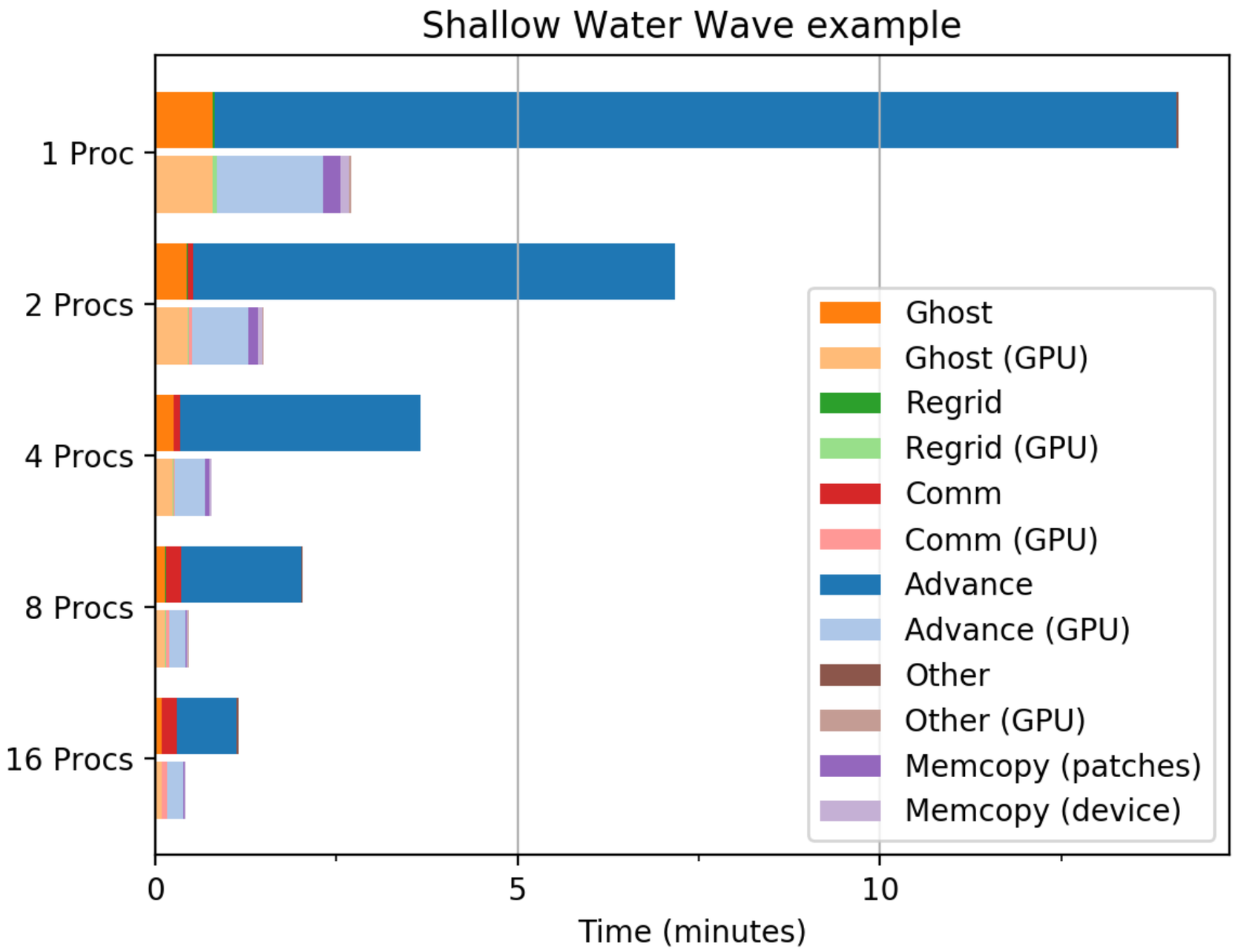
- Most supercomputers coming online have vast numbers of GPUs available.
- Codes which take advantage of GPU investments may get preference in queues
- GPUs on local clusters sit around unused
- Clean memory interface makes it clear how to optimally use memory hierarchies
- Remove dependence on third party libraries (and more groups that need to be on grants)

Our experience is that focusing effort on solvers has big pay-off without a large code base.

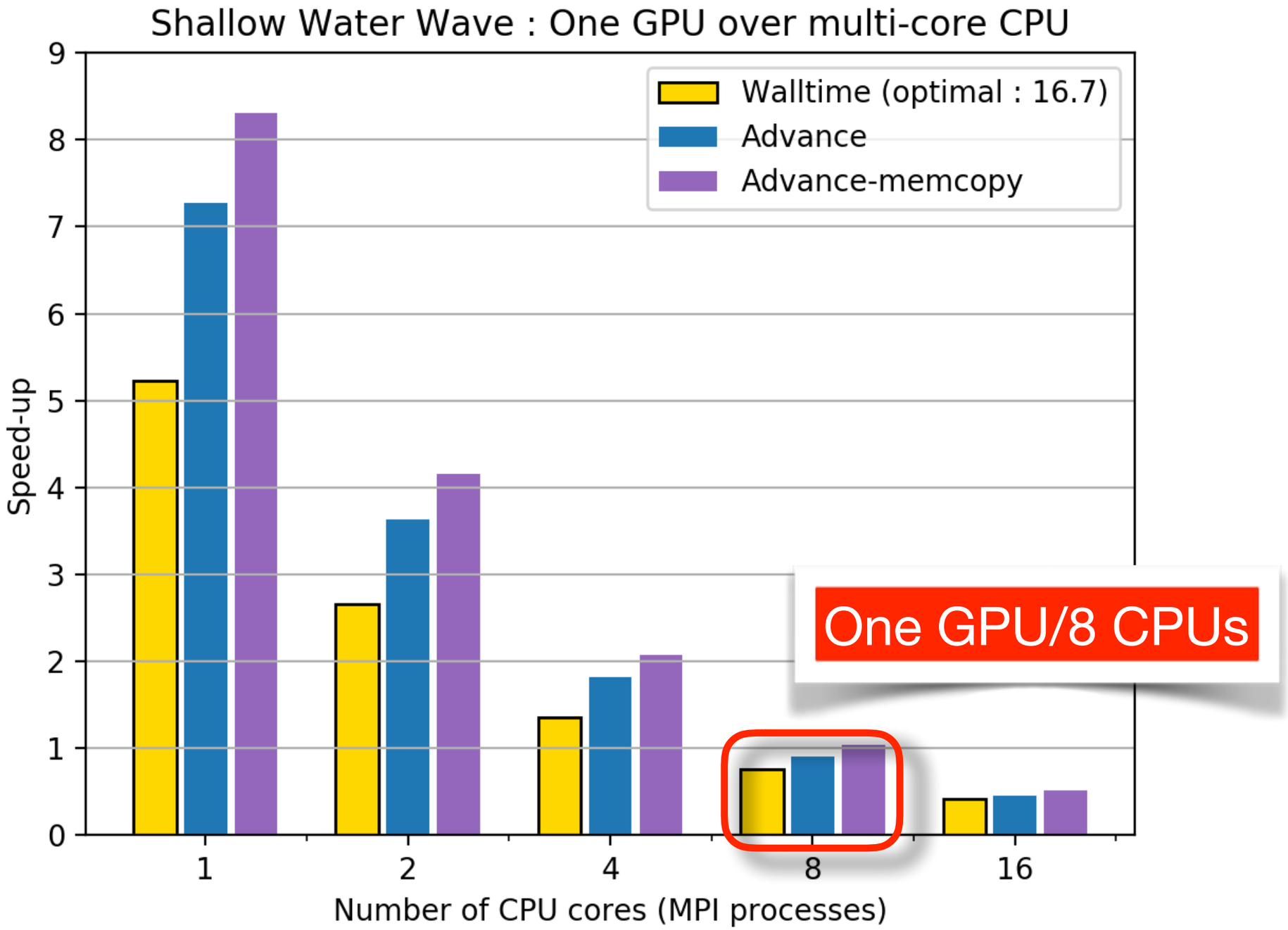
Shallow water



Shallow water



Shallow water



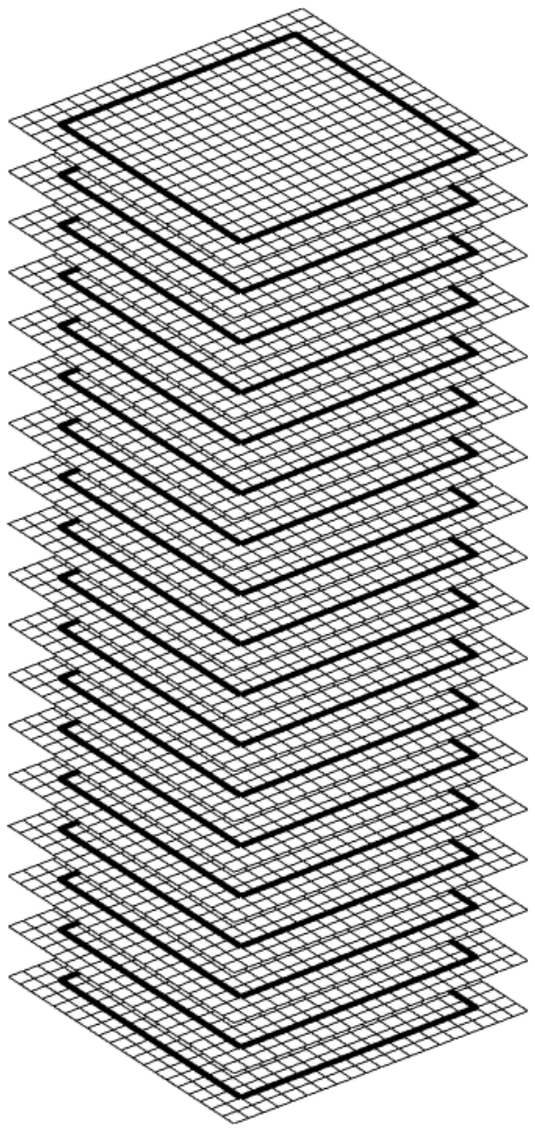
ForestClaw on GPUs

Ported fully unsplit wave propagation algorithm for hyperbolic conservation laws (implemented in Clawpack) to CUDA.

- Copy time level solution on all patches to single contiguous block of CPU memory
- Copy contiguous block of CPU memory to the GPU.
- Configure the GPU to assign one 1d thread block to each single ForestClaw patch
- Divide shared memory equally among thread blocks=patches
- All solution data resides in global memory; shared memory is only used for temporary data
- CUDA function pointers used to provide custom Riemann solvers.
- Best to use the 4.x (SOA) data layout

ForestClaw on GPUs

```
dim3 grid(1,1,batch_size);
```



One ForestClaw patch per CUDA block

```
block_size = 128;  batch_size = 4000;
mwork = 9*meqn + 9*maux + mwaves + meqn*mwaves;
bytes_per_thread = sizeof(double)*mwork;
bytes = bytes_per_thread*block_size;

dim3 block(block_size,1,1);
dim3 grid(1,1,batch_size);

claw_flux2<<<grid,block,bytes>>>(mx,my,meqn,..)
```

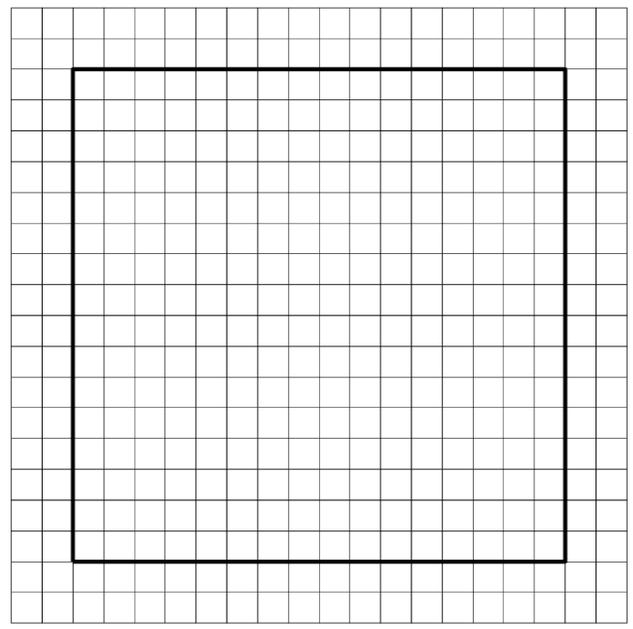
1d thread blocks

3d grid

~4000 patches in a batch

~128 threads per block

Patch layout with valid ghost cell data



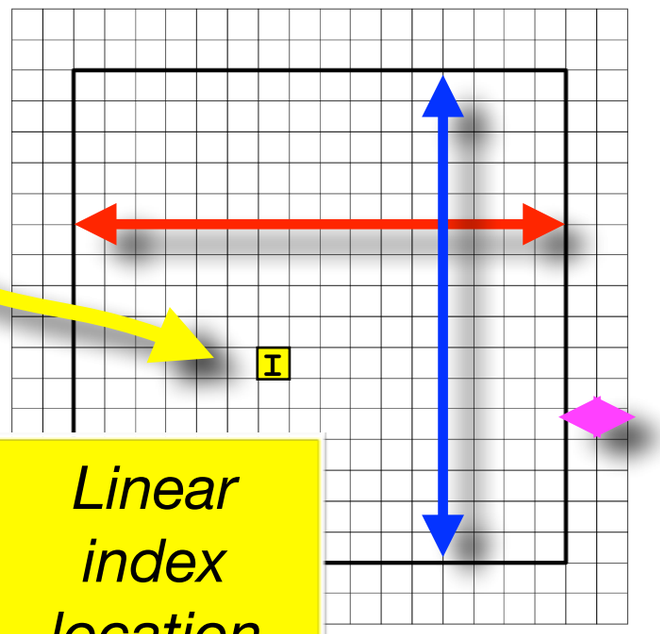
Thread block - loop over faces

```
ys = (2*mbc + mx);    /* Stride */
ifaces_x = mx + 2*mbc-1;
ifaces_y = my + 2*mbc-1;
num_cells = ifaces_x*ifaces_y;

for(ti = threadIdx.x; ti < num_ifaces; ti += blockDim.x)
{
  ix = ti % ifaces_x;
  iy = ti/ifaces_x;

  I = (iy + 1)*ys + (ix + 1);
    ....
```

Solve a normal Riemann problem at each face; include 1 ghost cell in each direction



Linear index location

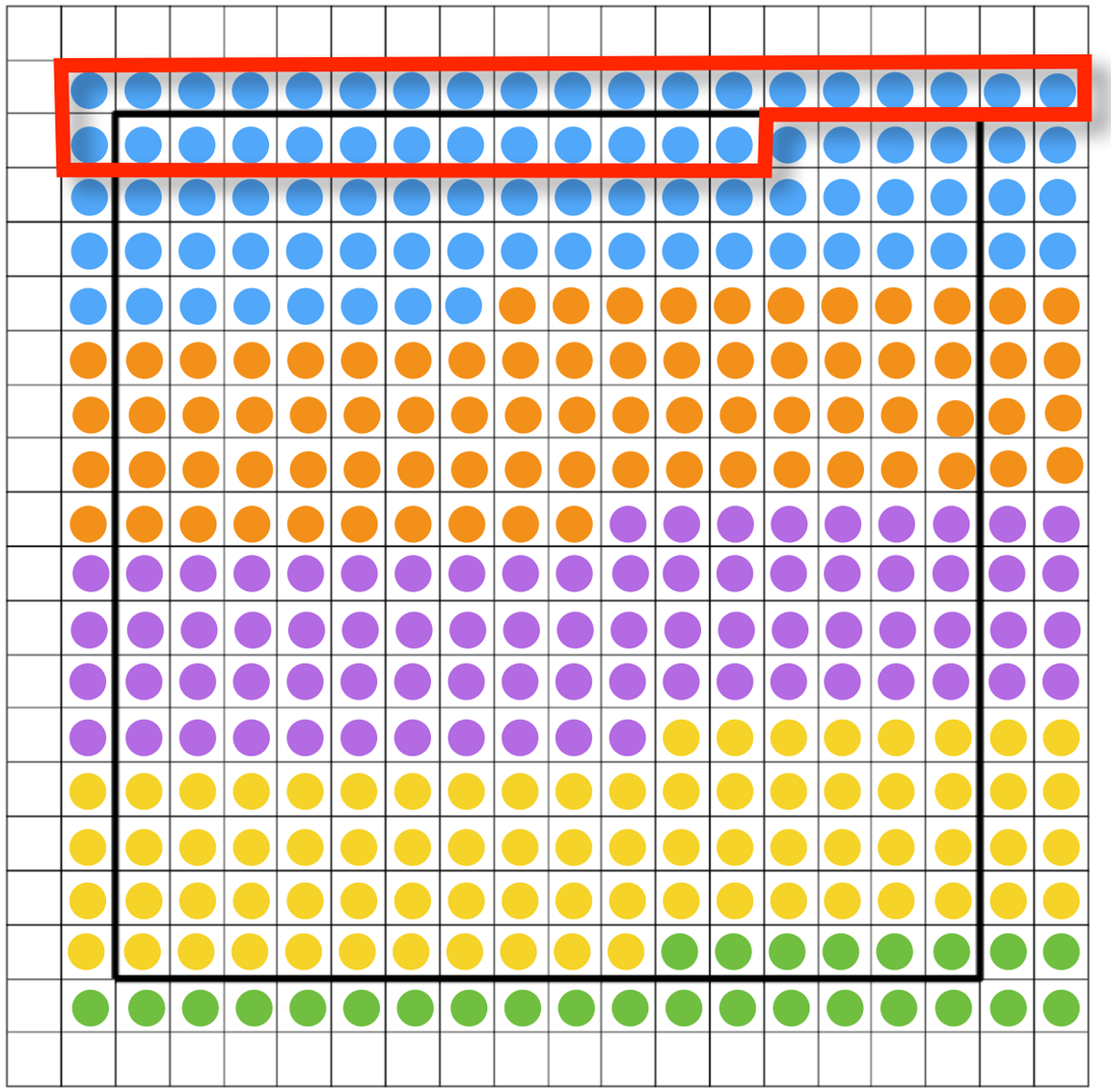
mx : Number of interior grid cells in x

my : Number of interior grid cells in y

mbc : Number of ghost cells

One dimensional thread block

- No block synchronization required
- Typical patch sizes are 32x32
- Number of threads per patch : ~128, depending on shared memory requirements



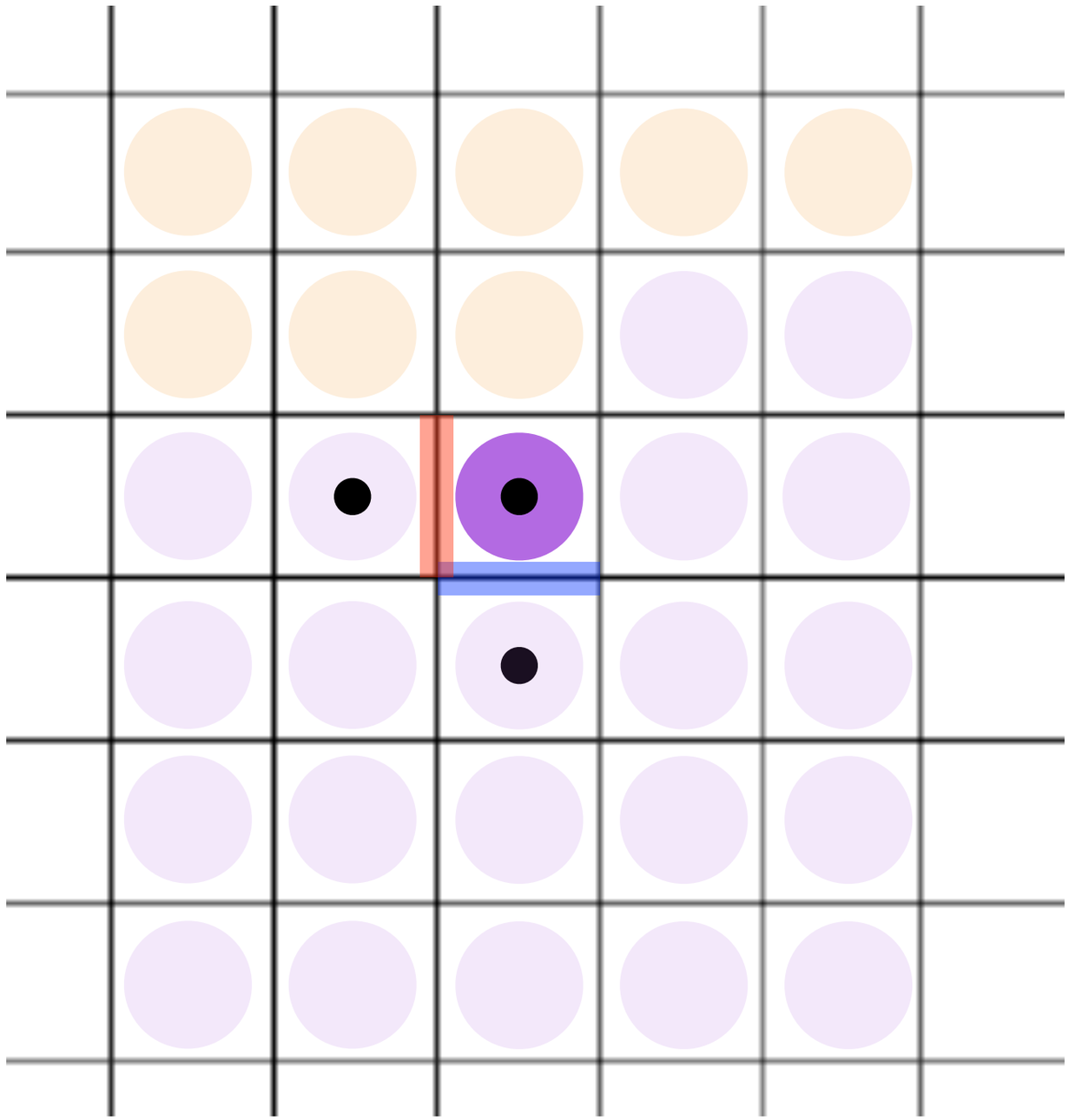
- *First pass*
- *Second pass*
- *Third pass*
- *Fourth pass*
- *Fifth pass*

Warp = 32 threads

Solve Riemann problems at x and y faces

Normal Riemann problems

- Each thread makes a local copy of global data and stores it in shared memory.
- Fluxes computed Riemann problems stored in global array



Thread

Global solution data accessed by thread

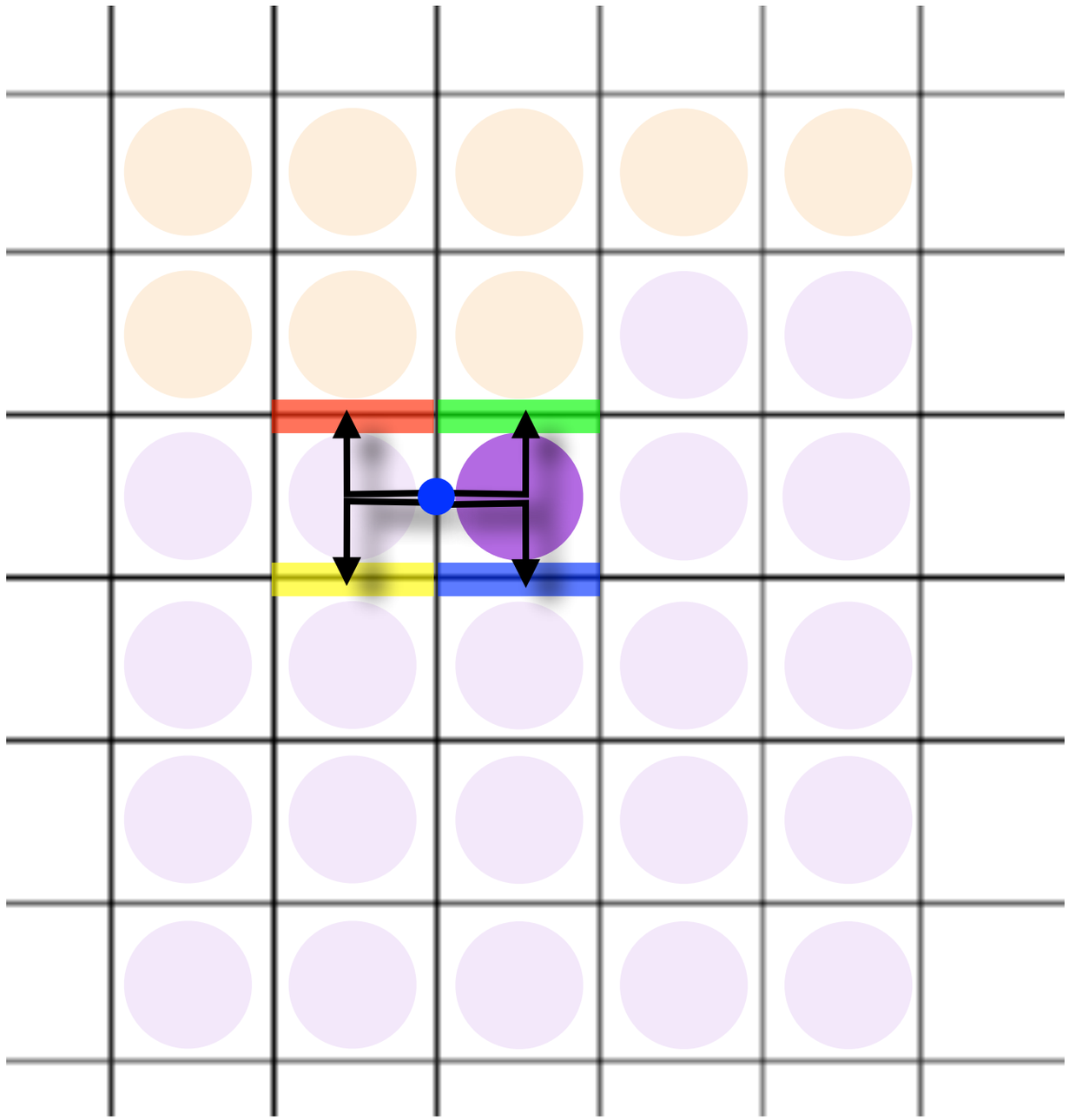
Fluxes computed at an x interface

Fluxes computed at a y interface

Fluxes are computed by solving Riemann problems

Unsplit algorithm

- Each transverse solve stores data in the same global memory space
- To avoid data collisions with other threads writing to the same global memory, four passes over all the global data are required, one for each “color”
- Sync threads between each pass



Thread

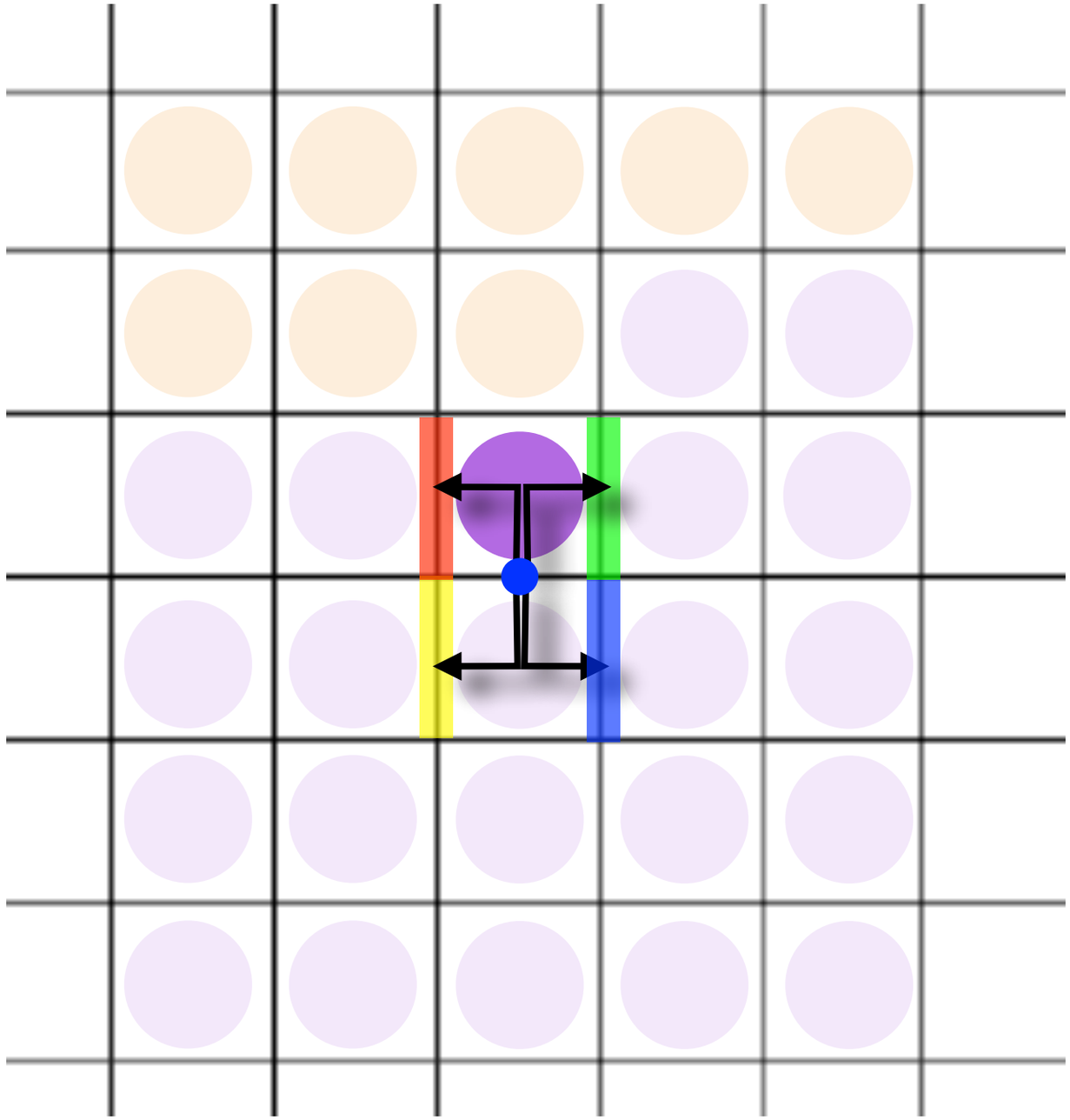
Global data stored at cell interfaces

Results from a horizontal normal Riemann problems are propagated in the vertical direction

Fluxes are computed by solving Riemann problems

Transverse Riemann problems

- Each transverse solve stores data in the same global memory space
- Four more passes over all the global data are required



Thread

Global data accessed by thread

Results from the vertical Riemann problems are then propagated in a horizontal direction

Fluxes are computed by solving Riemann problems

Unsplit wave propagation

Fully unsplit wave propagation algorithm is implemented in a single CUDA kernel.

- While more expensive than the dimensionally split version, the unsplit algorithm may be more suited to AMR.
- The cost in CUDA is that parts of the code that can be done together are are now split to avoid race conditions. *Maybe we can improve on this by using more global memory?*
- Our GPU configuration does not require any synchronization between thread blocks
- Since all patches are the same size, they can be processed in large batches (O(1000) per batch)
- All AMR tasks including filling ghost cells is done on the CPU
- Conservation requires extra memory copy from device.

GeoClaw extension of ForestClaw

- Developed by Melody Shih (NYU)
- GeoClaw Riemann solvers available in CUDA
- Timing results look very promising
- CPU Walltime : 46.90s (on four processors)
- GPU Walltime : 11.84s (on four processor; 2 GPUS/node)

Issues :

- Shared memory is very limited, so extensive use of auxiliary arrays reduces the number of threads that can be allocated per block.
- Current ForestClaw code for CUDA take advantage of 4.x layout, so data has to be transposed from GeoClaw layout before passing to GPU.

EuroHack 2018 - Lugano, Switzerland

Scott Aiton (BSU), Andreas Jocksch (CSCS), Xinsheng Qin (Univ. of Washington), D. Calhoun (BSU), Melody Shih (NYU)

Sponsored by : NVIDIA + Swiss National Computing Center

Ongoing work

- Direct solvers for AMR quad tree meshes (A. Gillman, CU Boulder)
- Extensions to full 3d (C. Burstedde)
- New and ongoing application areas including overland flooding (using GeoClaw extension in ForestClaw), volcanic ash transport (joint with USGS), wildland fires and fluid structure interactions.

Thank you!

www.forestclaw.org **www.github.com/ForestClaw**

Dispersive terms