Recent Parallel Results Using Dynamic Quad-Tree Refinement for Solving Hyperbolic Conservation Laws on 2d

Donna Calhoun (Boise State University)

Carsten Burstedde (Univ. of Bonn)

Collaborators : R. J. LeVeque (Univ. of Washington); M. J. Berger (NYU); D. George (USGS); D. Ketcheson (KAUST); K. Mandli (Columbia); Christiane Helzel (Univ. of Dusseldorf)

SIAM Parallel Processing

April 12-15, 2016

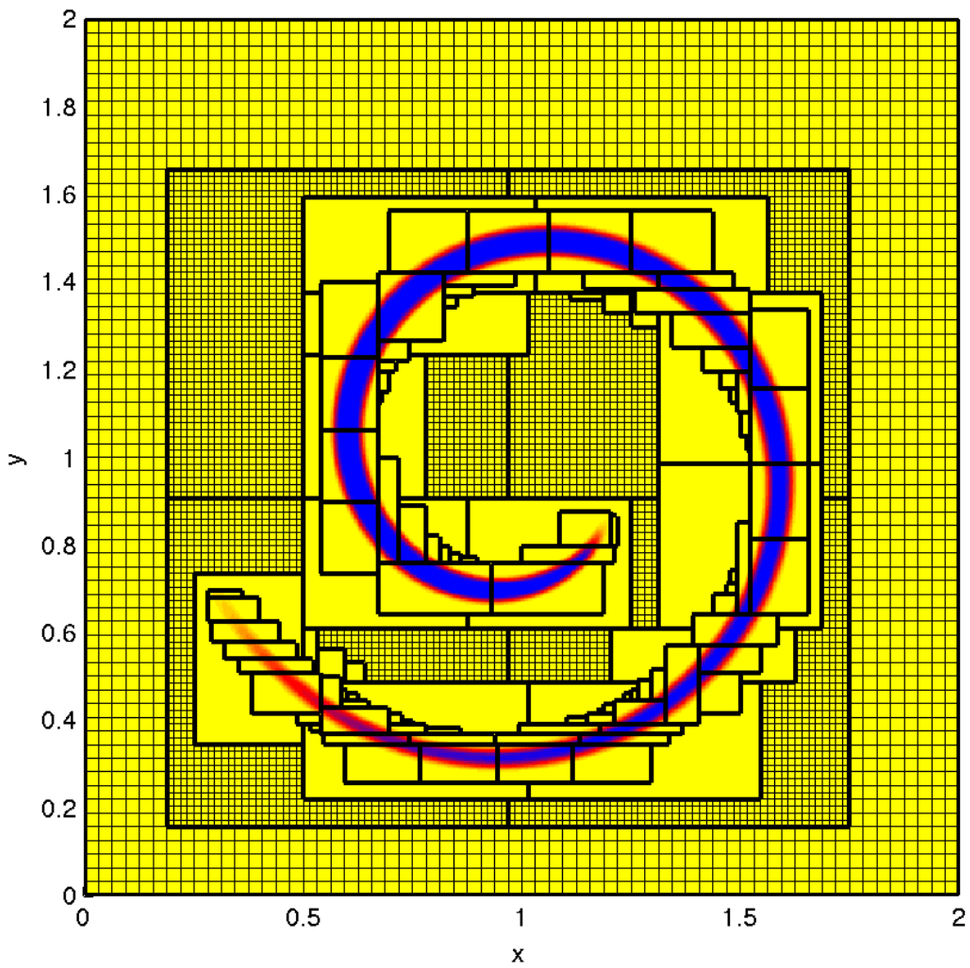
Universite Pierre et Marie Curie

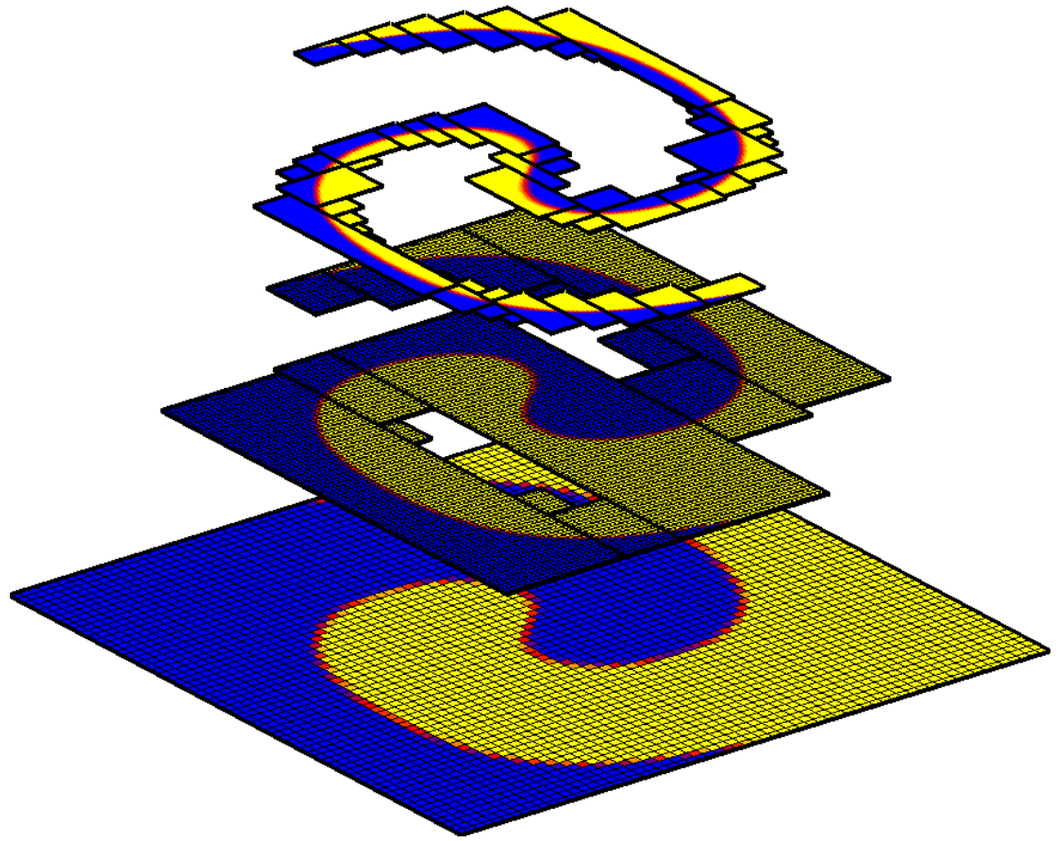
Paris, France

Adaptive Mesh Refinement (AMR)

Overlapping patch-based AMR

Original approach (Berger, 1984)





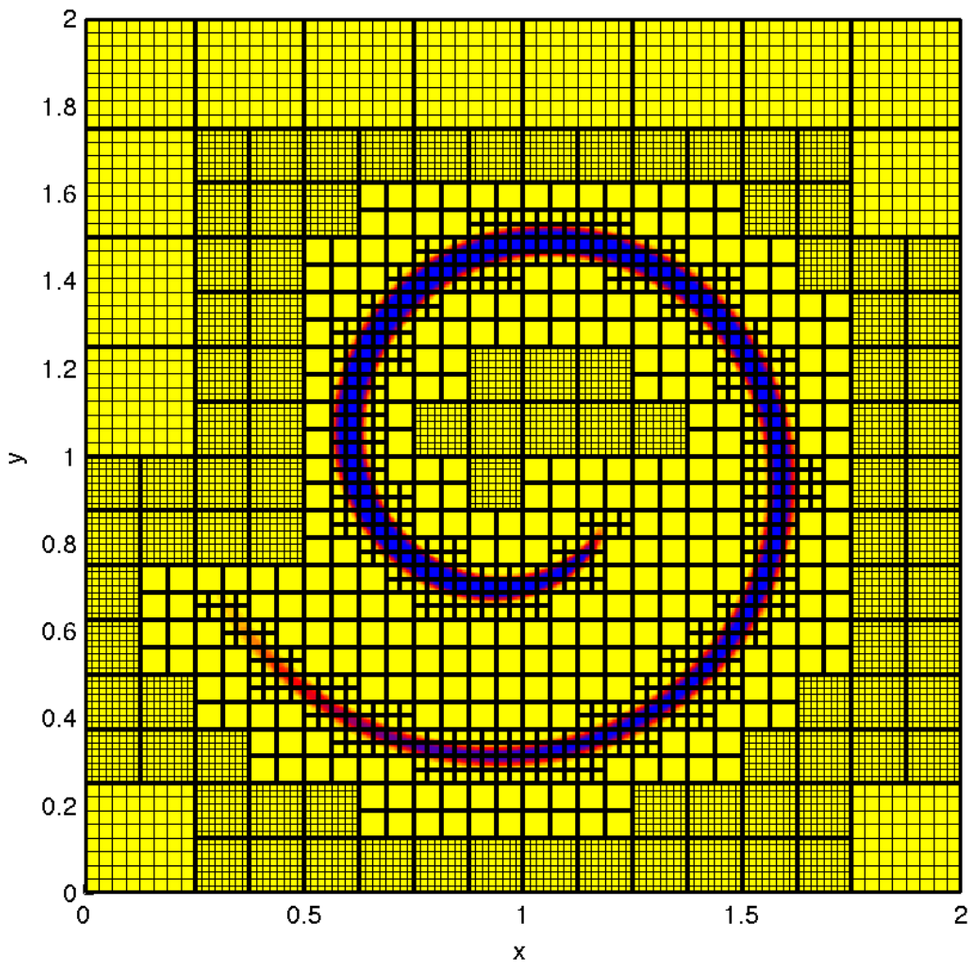
Chombo (LBL), AMRClaw (UW, NYU) , **Boxlib* (LBL)**, SAMRAI (LLNL), AMROC (S. Hampton)

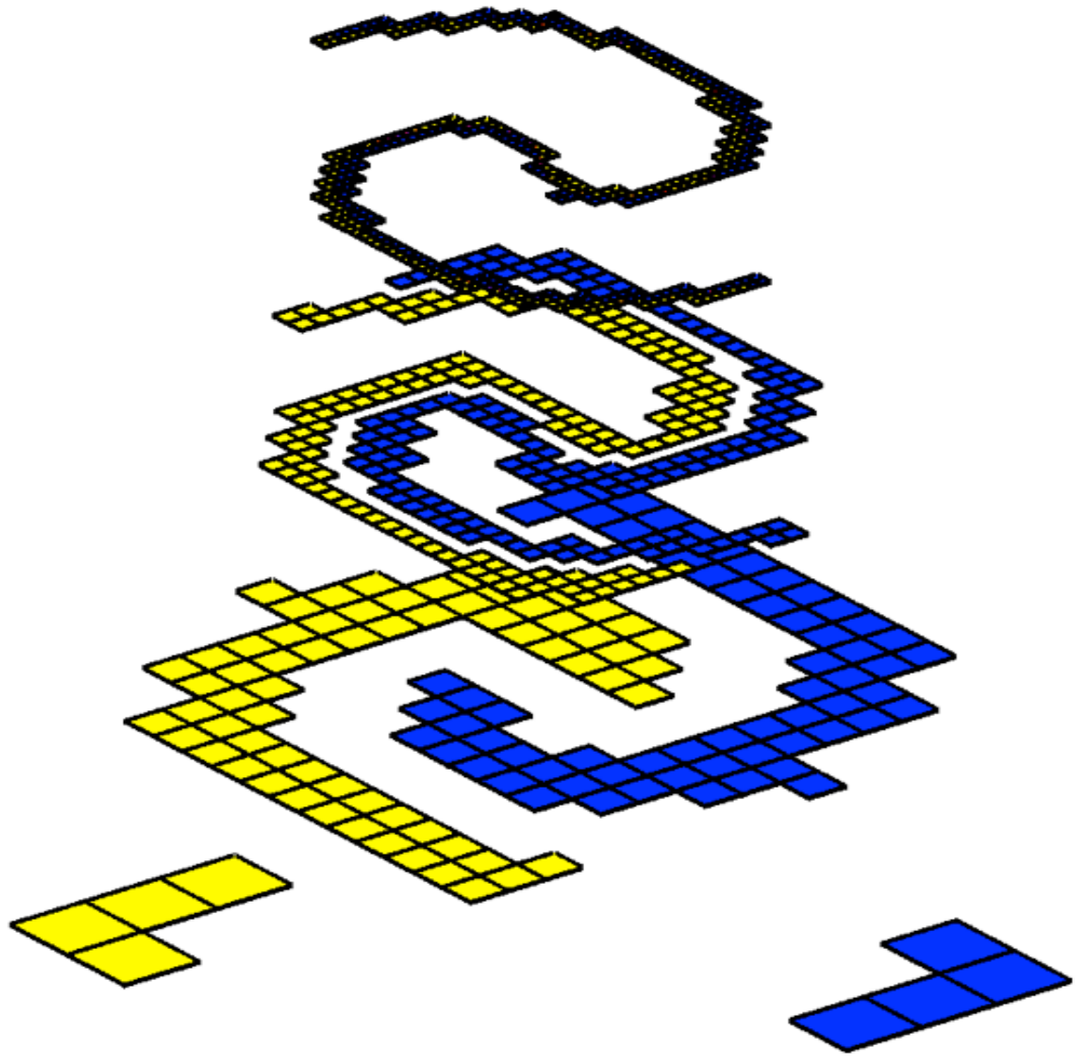
**See *IP4 : "Next Generation AMR (A. Almgren, Thursday 8:30)*

Adaptive Mesh Refinement (AMR)

Quadtree/Octree based AMR

Quad-tree approach





p4est (U. Bonn), PARAMESH (U. Chicago), **ForestClaw**, Gerris (Paris VI), Racoon II (U. Bochum), RAMSES (U Zurich), Nirvana (Potsdam), "Building Cubes" (Tohoku)

ForestClaw Project

A parallel, adaptive library for logically Cartesian, mapped, multi-block domains

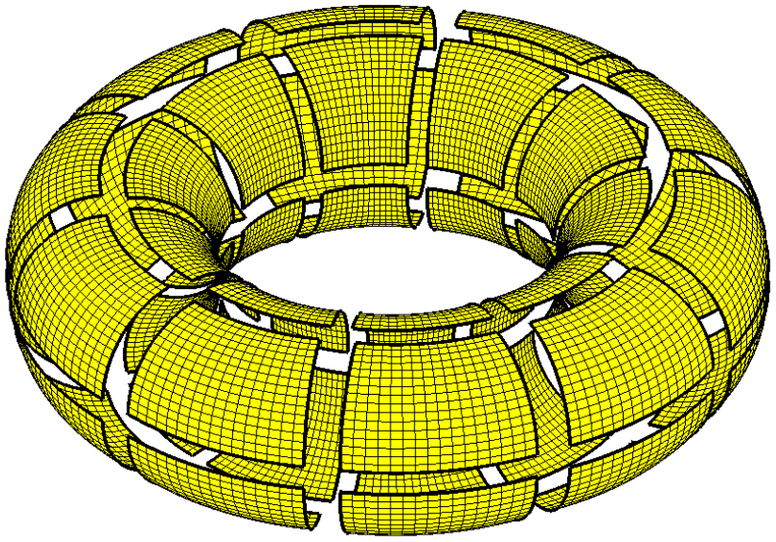
Features of ForestClaw include :

- Uses the **highly scalable p4est** dynamic grid management library (C. Burstedde, Univ. of Bonn, Germany)
- Each leaf of the quadtree contains a fixed, uniform grid,
- Optional multi-rate time stepping strategy,
- Has **mapped, multi-block** capabilities, (cubed-sphere, for example) to allow for flexibility in physical domains,
- Modular design gives user flexibility in including several solvers and packages.
- Uses essentially the same algorithmic components as patch-based AMR

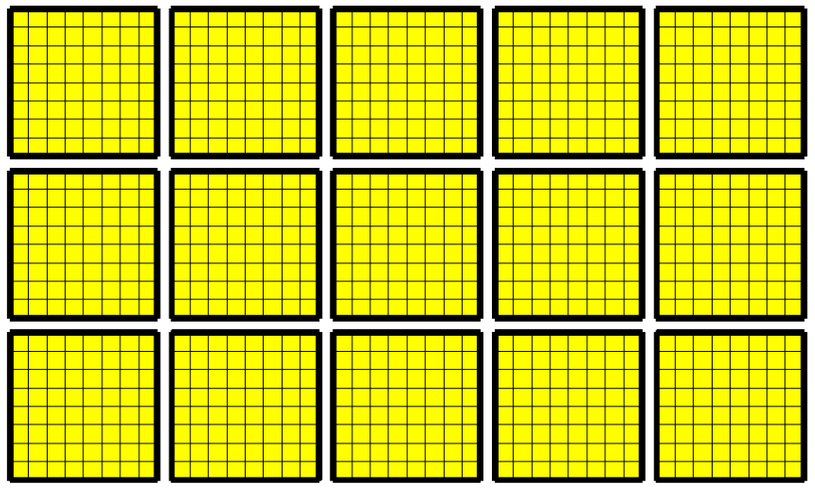
ForestClaw development supported by the National Science Foundation

www.forestclaw.org

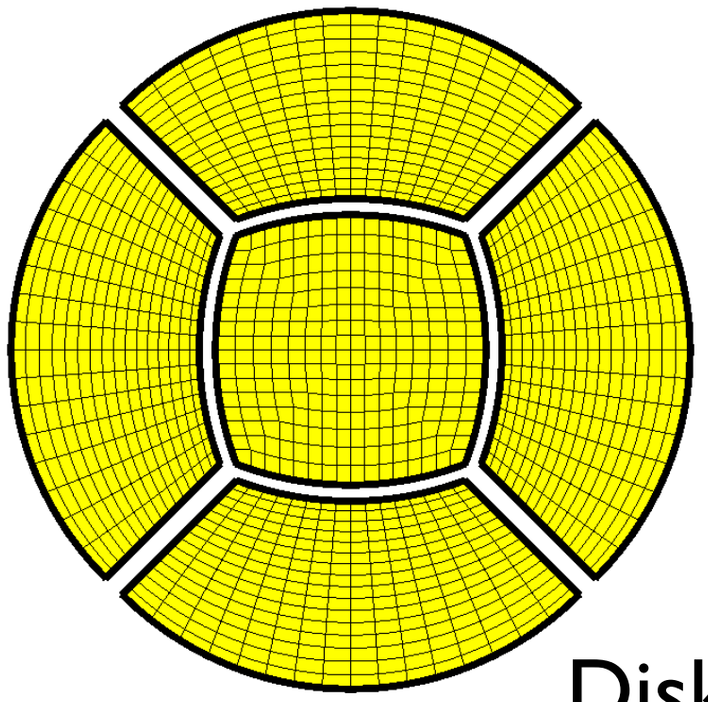
Mapped multi-block domains



Torus

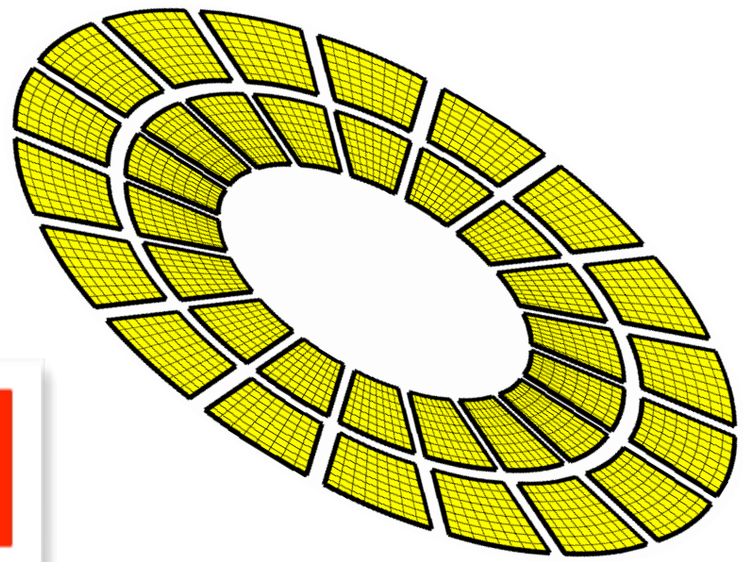


"Brick domain"



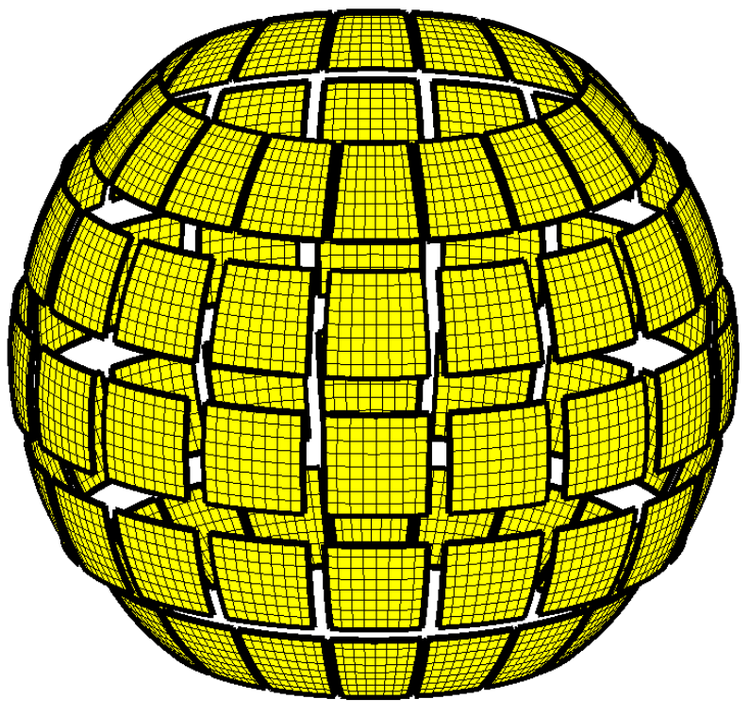
Disk

Each block is a quadtree

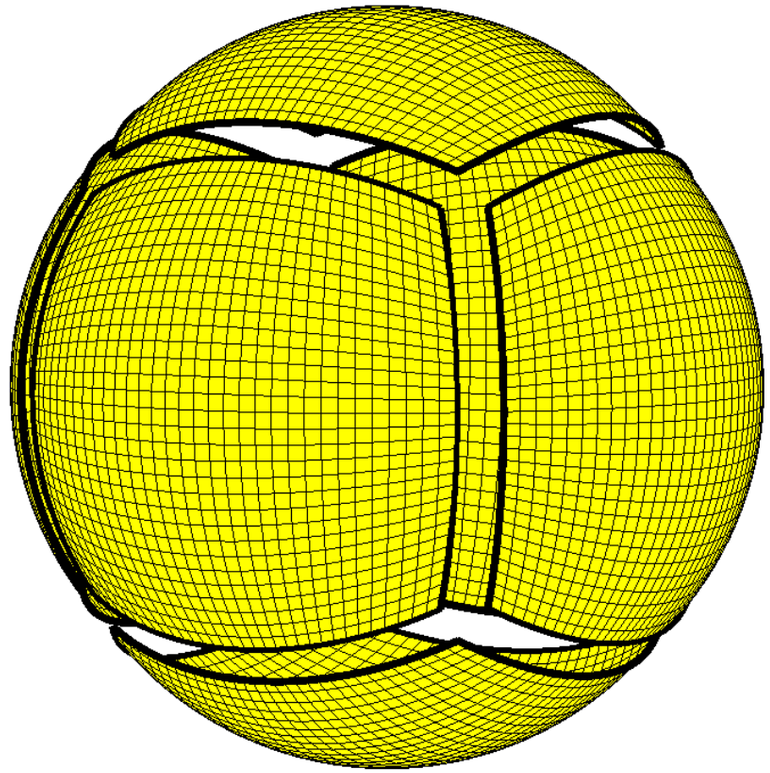


Annulus

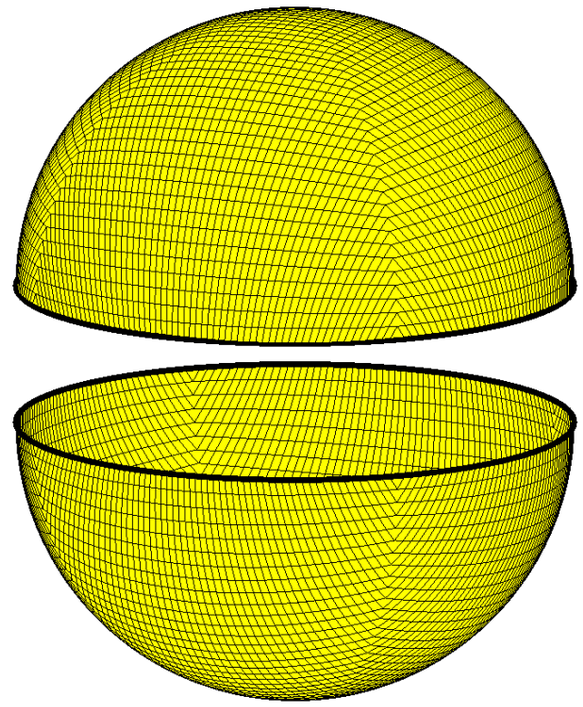
Mapped multi-block domains



Spherical coordinates



Cubed sphere



"Pillow grid" (with C. Helzel, R. LeVeque, SIAM Review, 2009)

Each block is a quadtree

Solvers?

Hyperbolic problems (currently available)

- Explicit, unsplit finite volume wave propagation methods in Clawpack (R. J. LeVeque, M. Berger, and many others)

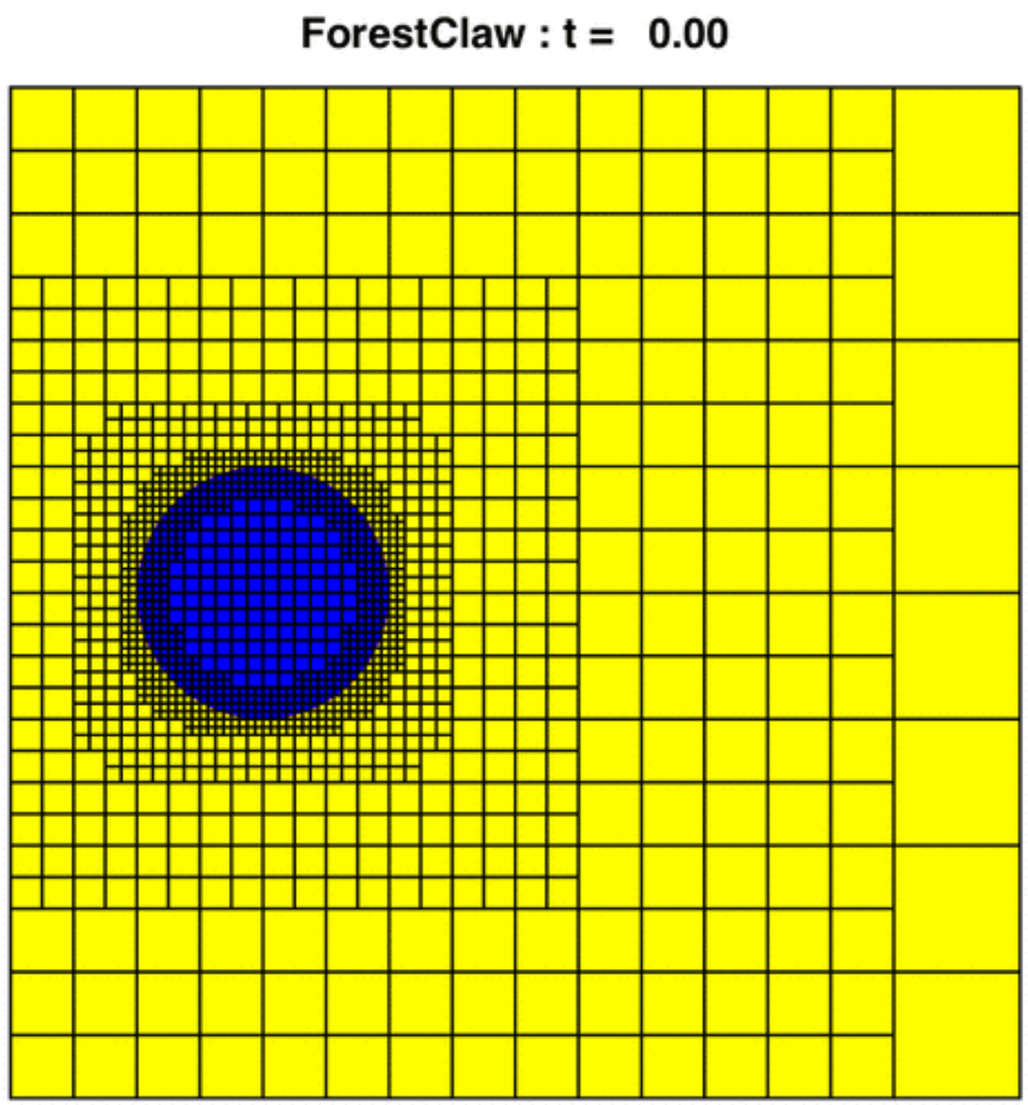
Parabolic problems (available soon)

- Stabilized, explicit, Runge-Kutta Chebyschev (RKC) methods (J. Verwer, B. Sommeijer, L. Shampine, A. Abdulle, …)

Elliptic problems (future work)

- Discrete Duality Finite Volume (DDFV) methods (F. Hermeline, P. Omnes, S. Delcourte, K. Domelevo, F. Hubert, Y. Coudiere, …)

Filament



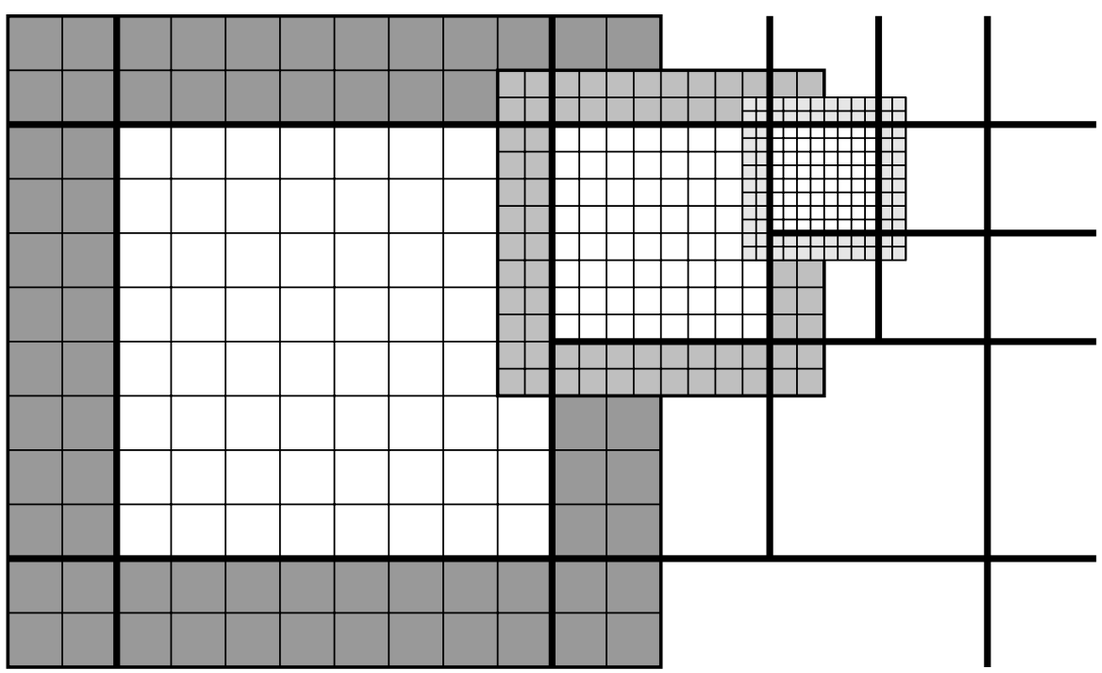
Tracer flow in a prescribed velocity field

Algorithmic components in AMR

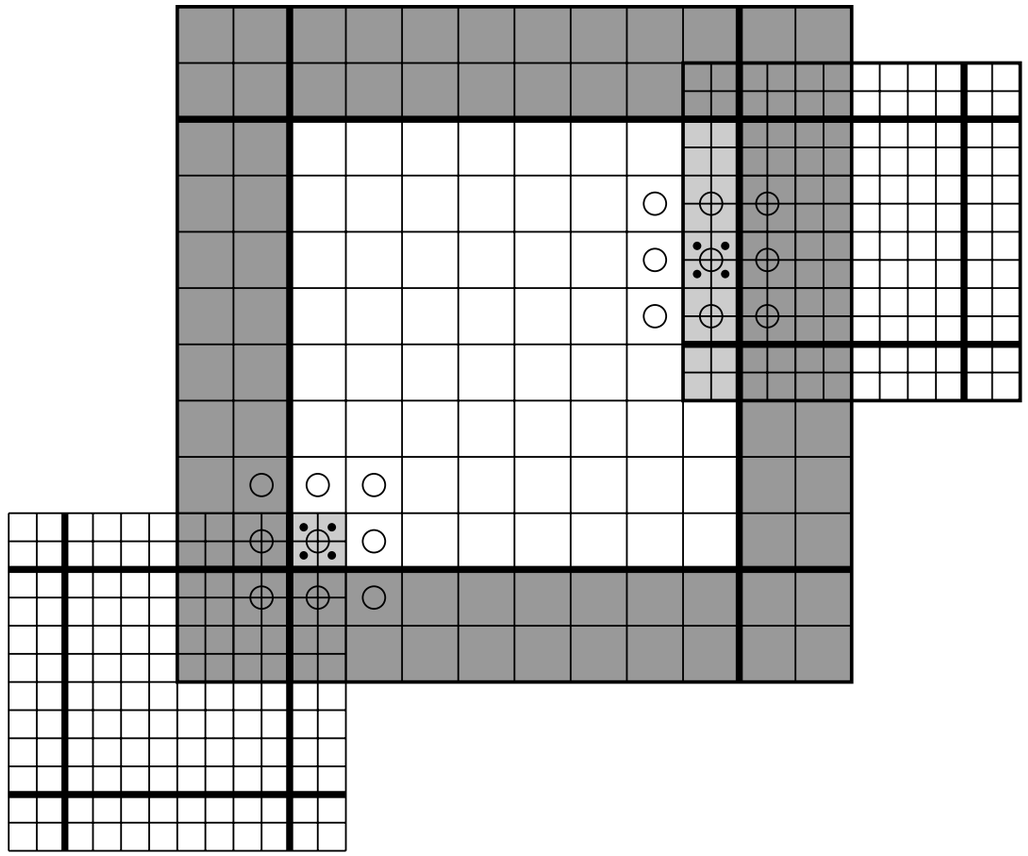
- Filling ghost cells
- Refinement criteria
- Adaptive mesh generation based on refinement criteria
- Parallel communication
- Multi-rate time stepping
- Grid mappings and grid transformations at multi-block seams
- Conservation at coarse/fine boundaries
- Accuracy at coarse/fine boundaries

Focus of this talk is on parallel performance

Filling ghost cells



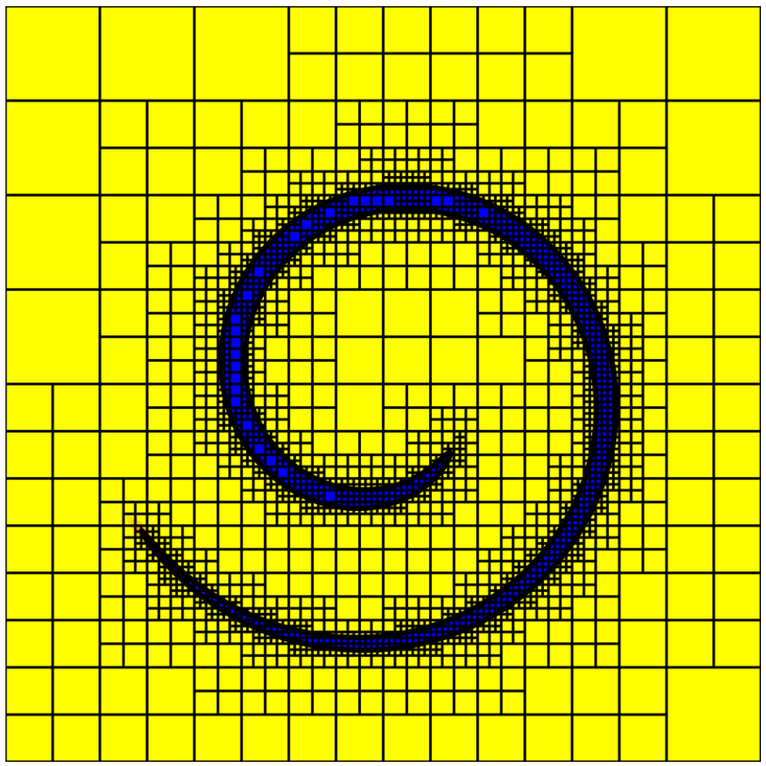
Each grid (or "leaf", in p4est terminology) has one of more layers of ghost cells used for communication between grids



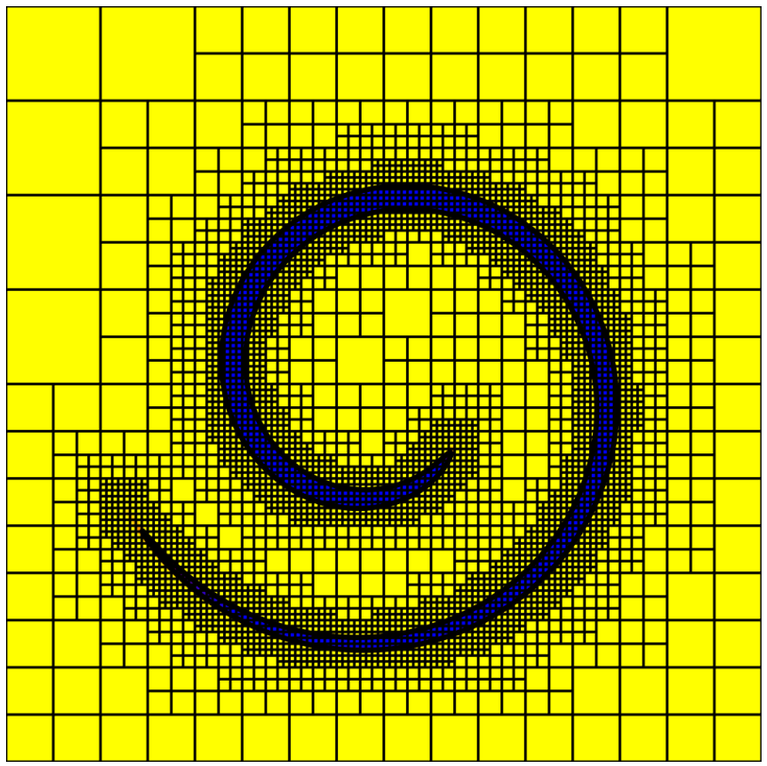
Ghost cells are filled by copying from same-size neighbors, averaging from fine grid neighbors or interpolation to fine grid neighbors.

Smooth Refinement

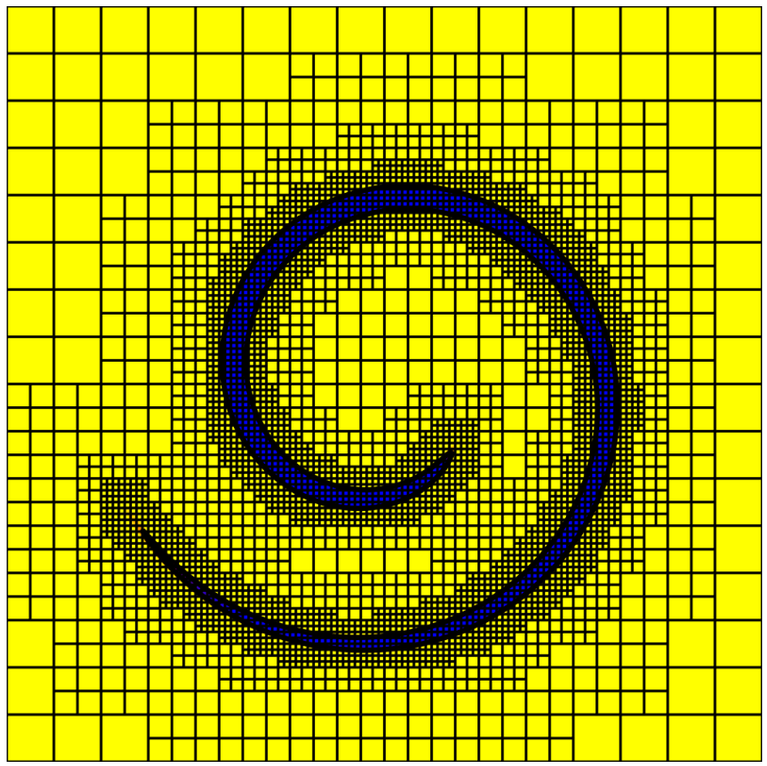
Provide buffer regions around specified levels



No grid buffering



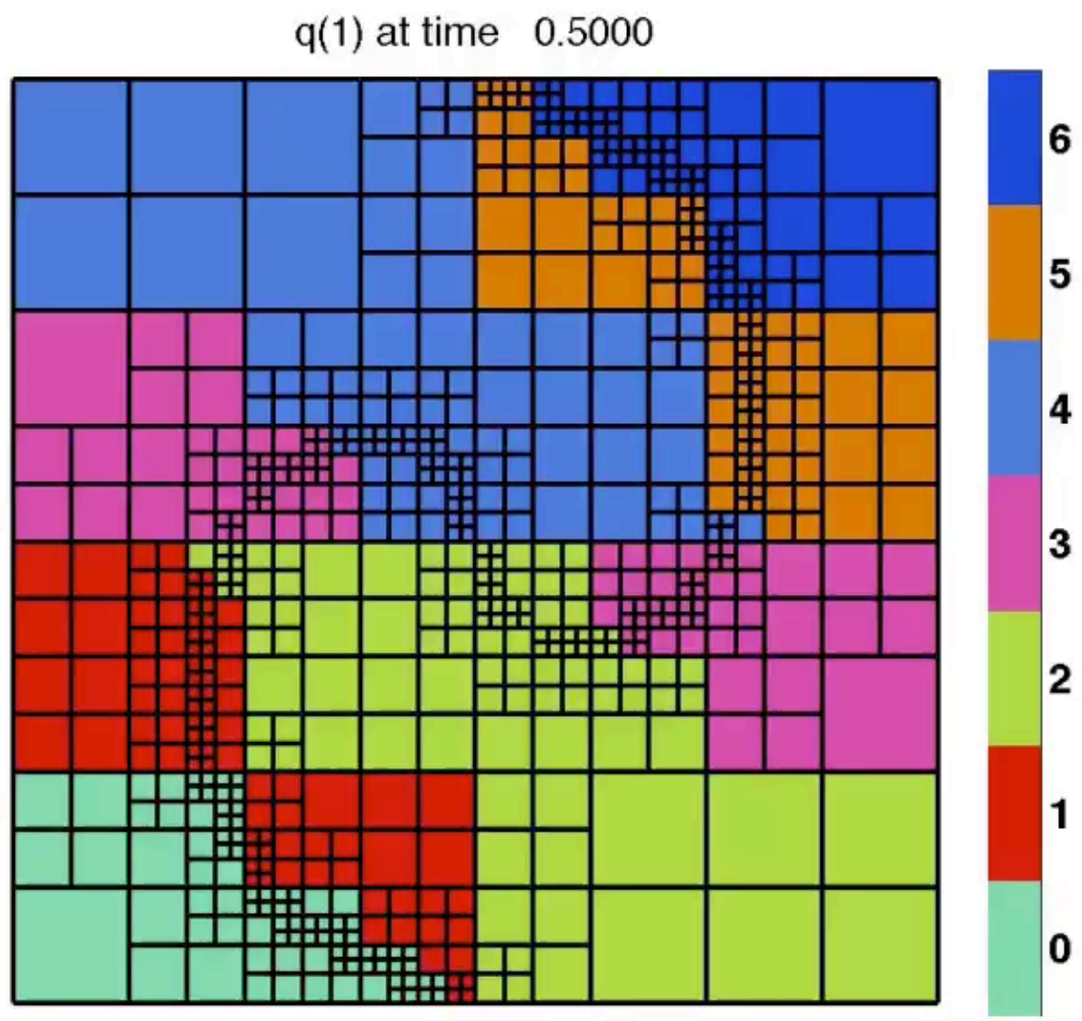
Buffering around finest level



All levels smoothly refined

Feature based refinement - easy to implement and often only practical choice

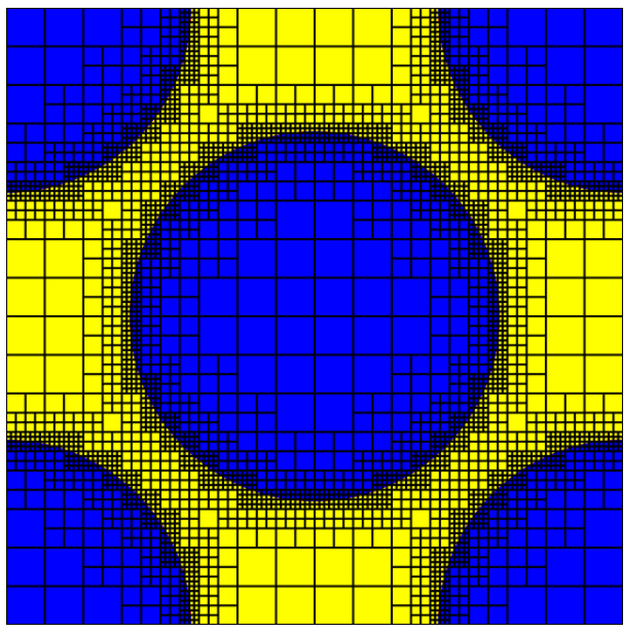
Distributing grids to processors

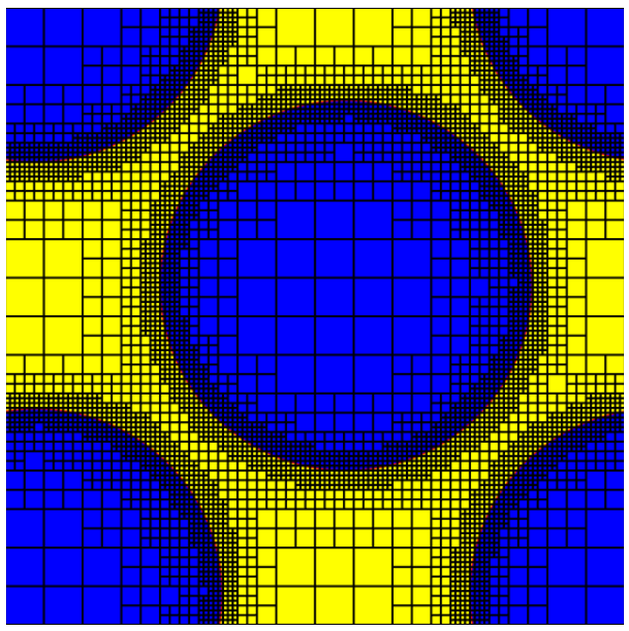


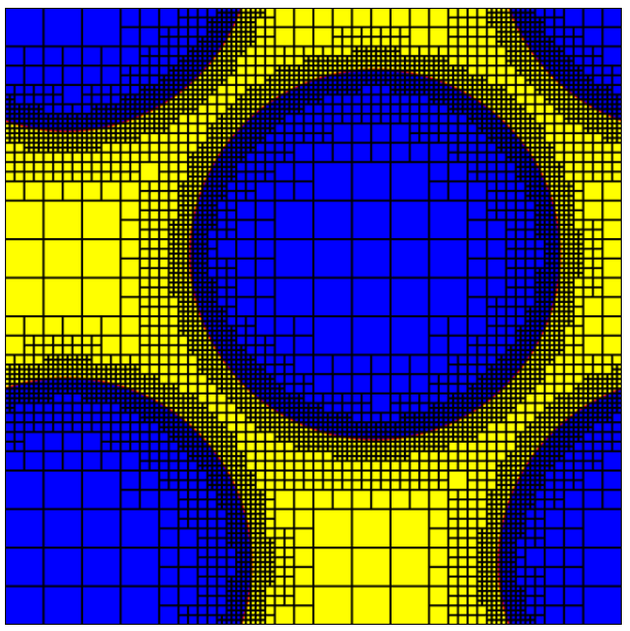
- A "grid" (i.e. leaf in the quadtree) is our minimum parallel unit
- Morton ordering space filling curve used to load balance and preserve data locality

Parallel Performance

- Scalar advection in a constant, periodic flow
- Uniform, replicated adaptive, and adaptive scenarios
- Fixed, global time step (160 steps total)
- Regrid every coarse grid time step (20 regrid steps)
- Four levels of refinement (levels 4-7)
- MPI processes only; no threading or tuning, and no attention paid to processor topology







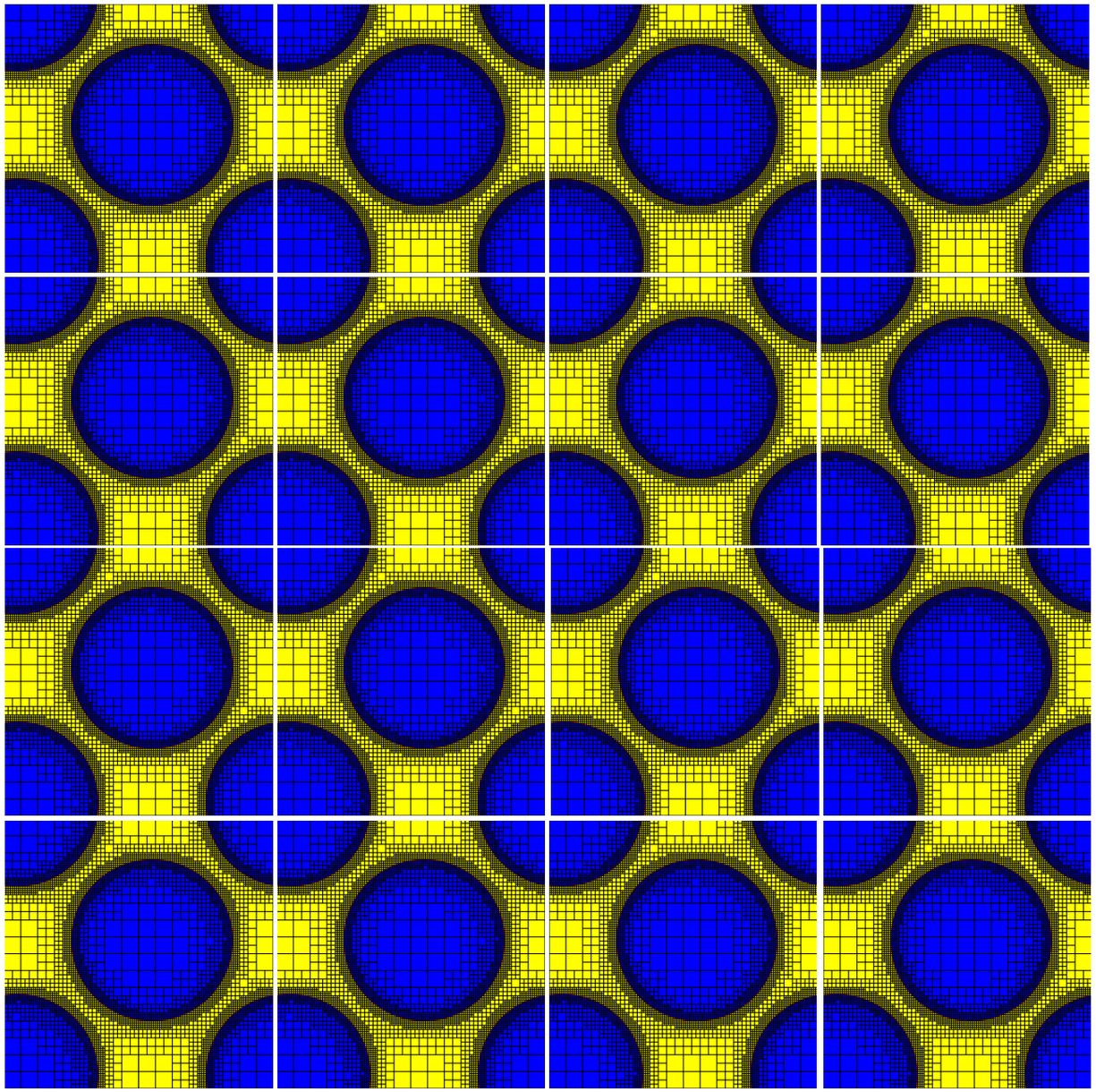
JUQUEEN - BlueGene/Q

Parallel tests were all done on JUQUEEN, the BlueGene/Q system at Juelich Super Computing Center (Germany)

- 28,672 nodes X 16 cores/nodes = 458,752 cores
- IBM PowerPC A2 @1.6 GHz
- 16GB RAM per node
- Quad floating point unit (FPU) for 4-wide double precision FPU (16, 32 or 64 ranks-per-node possible)
- 32 node = 512 core minimum billing unit
- 30 minute time limit on jobs that requiring fewer than 32 nodes

Many thanks to C. Burstedde for writing the proposal for time on JUQUEEN, and to the extremely helpful staff at Juelich, especially Kay Thust, who answered many of my emails.

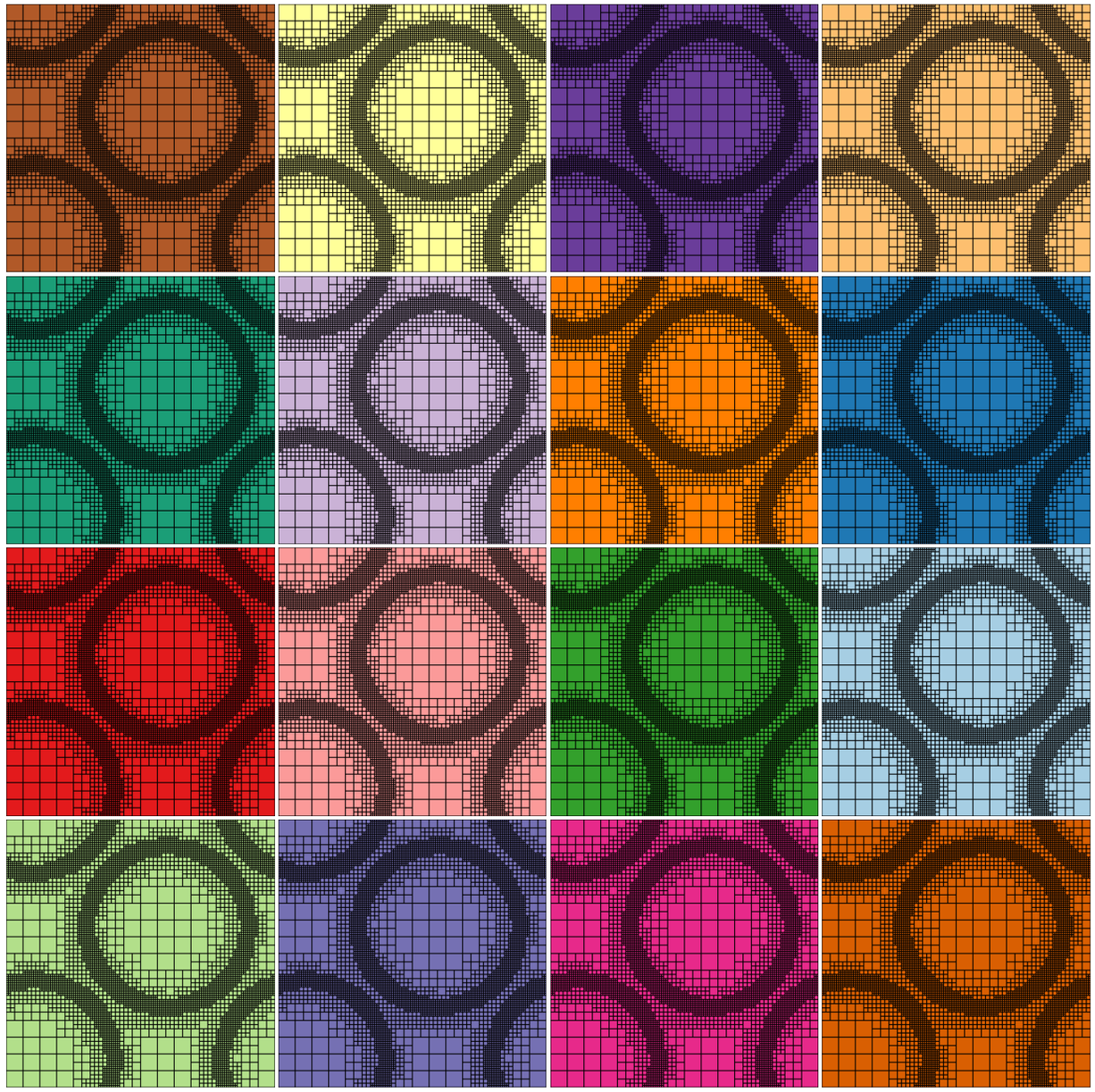
Weak scaling - replicated problem



Use "brick domain" to easily replicate the problem.

To show weak scaling, we replicate problem on each brick.
Also used in scaling studies by Colella, Bell, et al. (2007), and others.

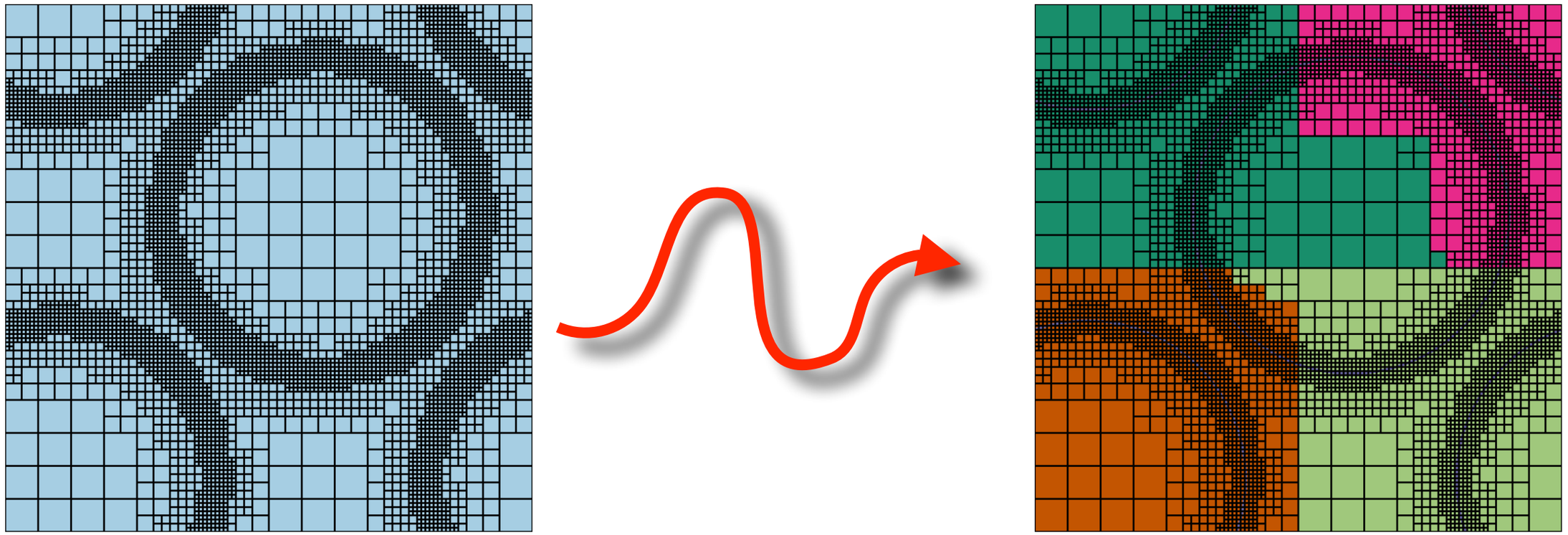
Weak scaling - replicated problem



16 processor run (one process per brick)

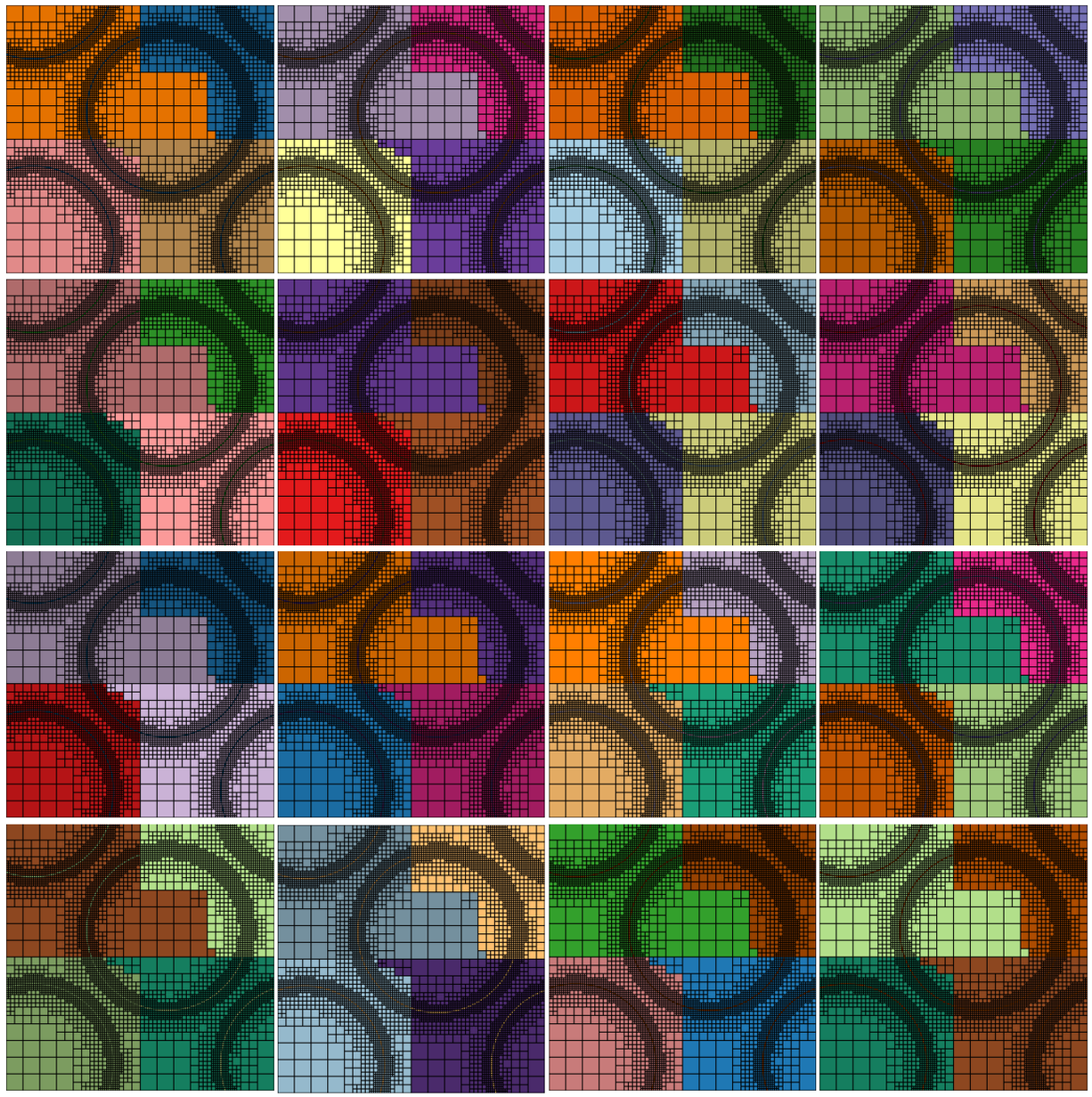
Equal numbers of grids on each processor

Strong scaling - replicated problem



Space filling curve used to equally distribute grids to processors

Strong scaling - replicated problem



64 processes
(four per brick)

Replicated problem

Grids per processor - **uniform**

Procs	1x1	2x2	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	4096	---	---	---	---	---	---	---	---
4	1024	4096	---	---	---	---	---	---	---
16	256	1024	4096	---	---	---	---	---	---
64	64	256	1024	4096	---	---	---	---	---
256	16	64	256	1024	4096	---	---	---	---
1024	---	16	64	256	1024	4096	---	---	---
4096	---	---	16	64	256	1024	4096	---	---
16384	---	---	---	16	64	256	1024	4096	---
65536	---	---	---	---	16	64	256	1024	4096

Grids per processor - **adaptive**

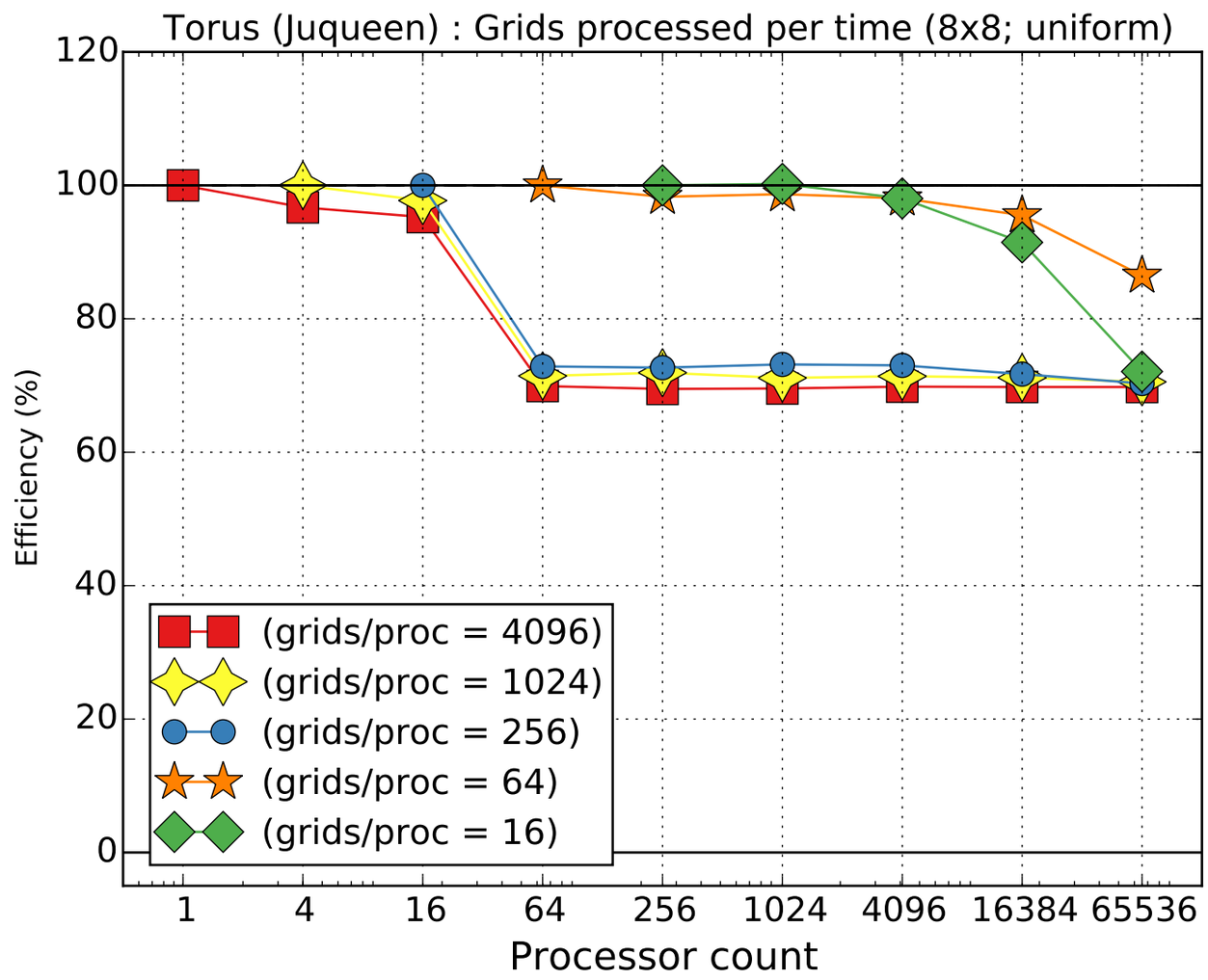
Procs	1x1	2x2	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	**5498**	---	---	---	---	---	---	---	---
4	1374	**5498**	---	---	---	---	---	---	---
16	343	1374	**5498**	---	---	---	---	---	---
64	85	343	1374	**5498**	---	---	---	---	---
256	21	85	343	1374	**5498**	---	---	---	---
1024	---	21	85	343	1374	**5498**	---	---	---
4096	---	---	21	85	343	1374	**5498**	---	---
16384	---	---	---	21	85	343	1374	**5498**	---
65536	---	---	---	---	21	85	343	1374	**5498**

Strong scaling

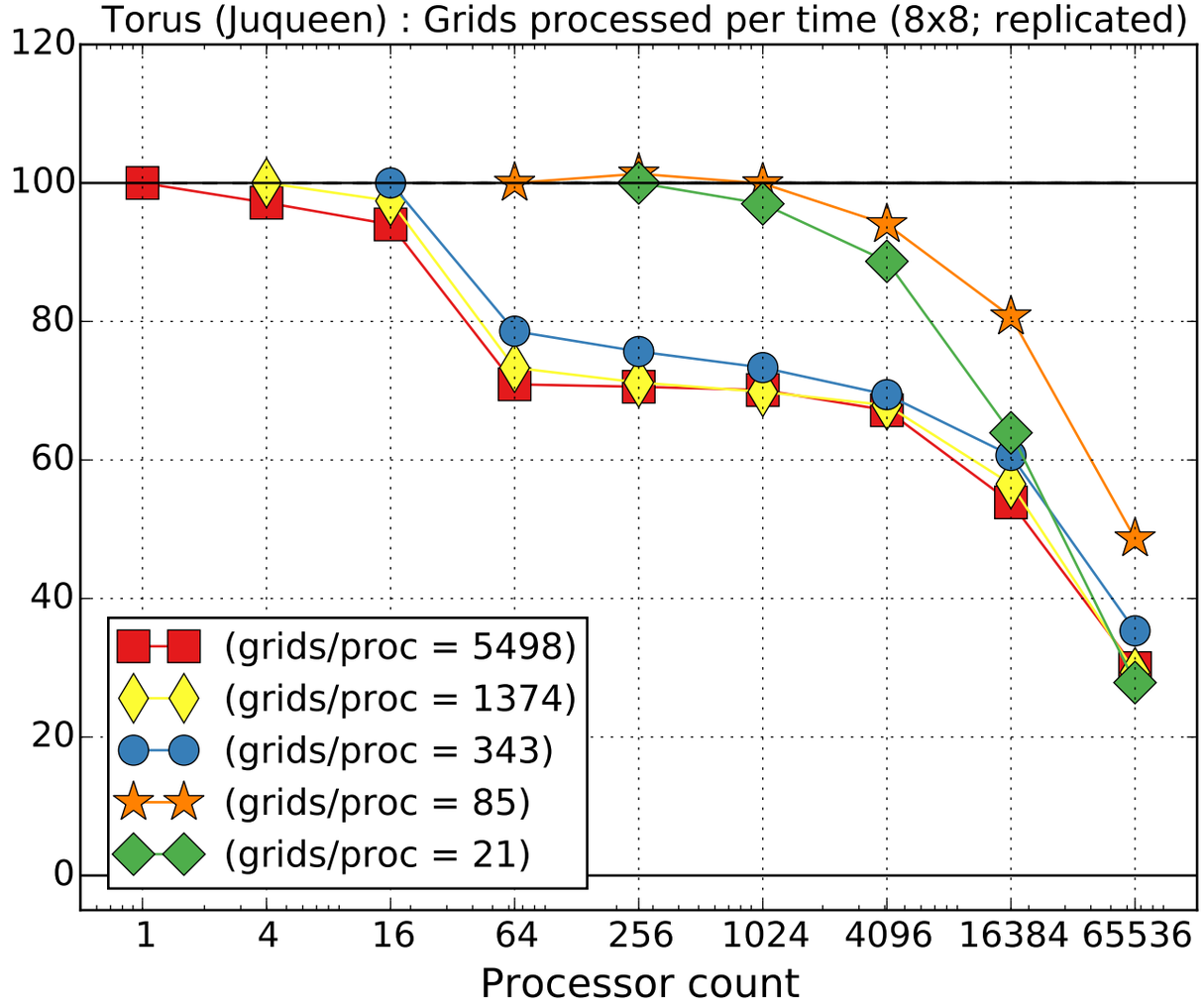
Weak scaling

Weak Scaling - 8x8 grids

Uniform



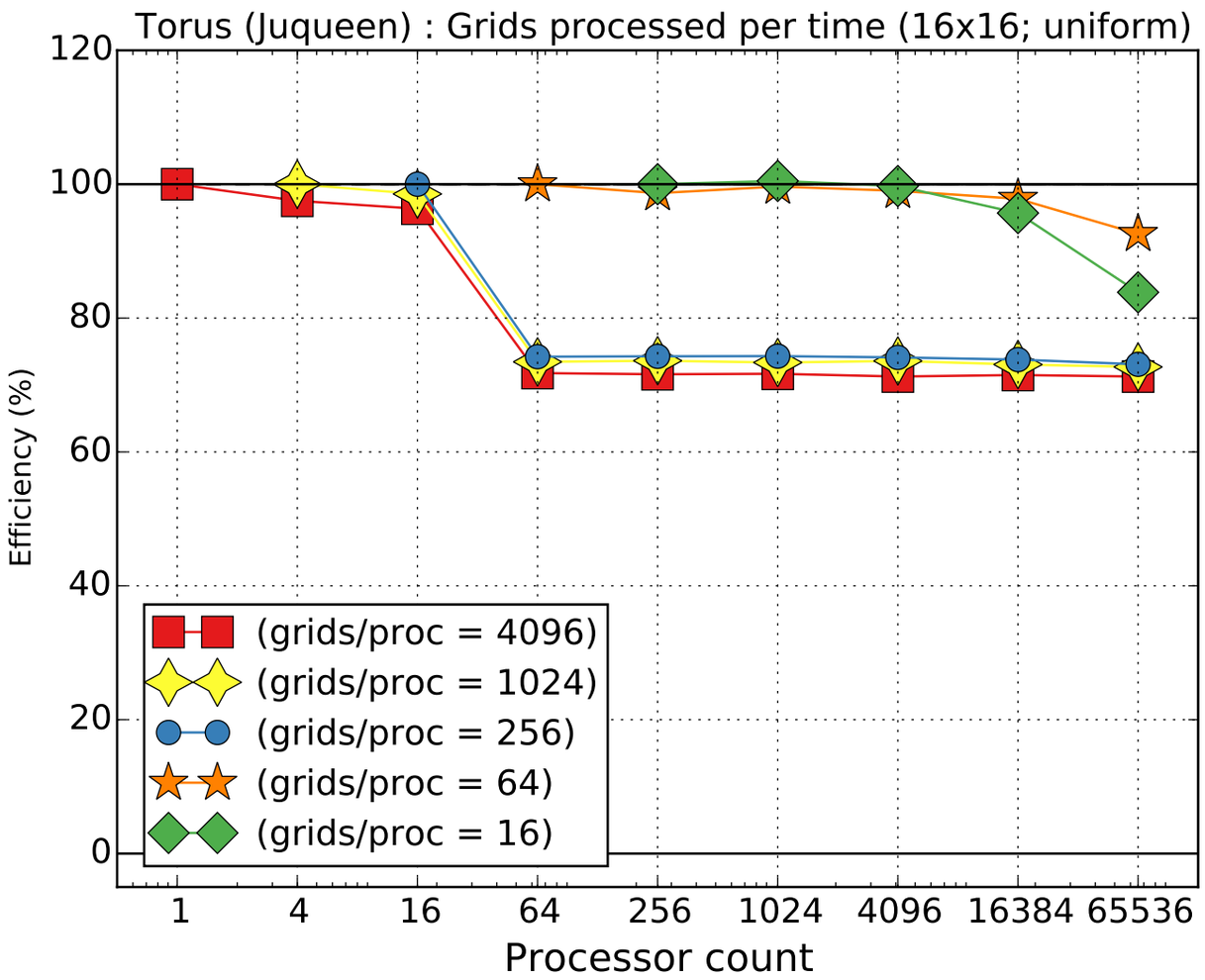
Adaptive



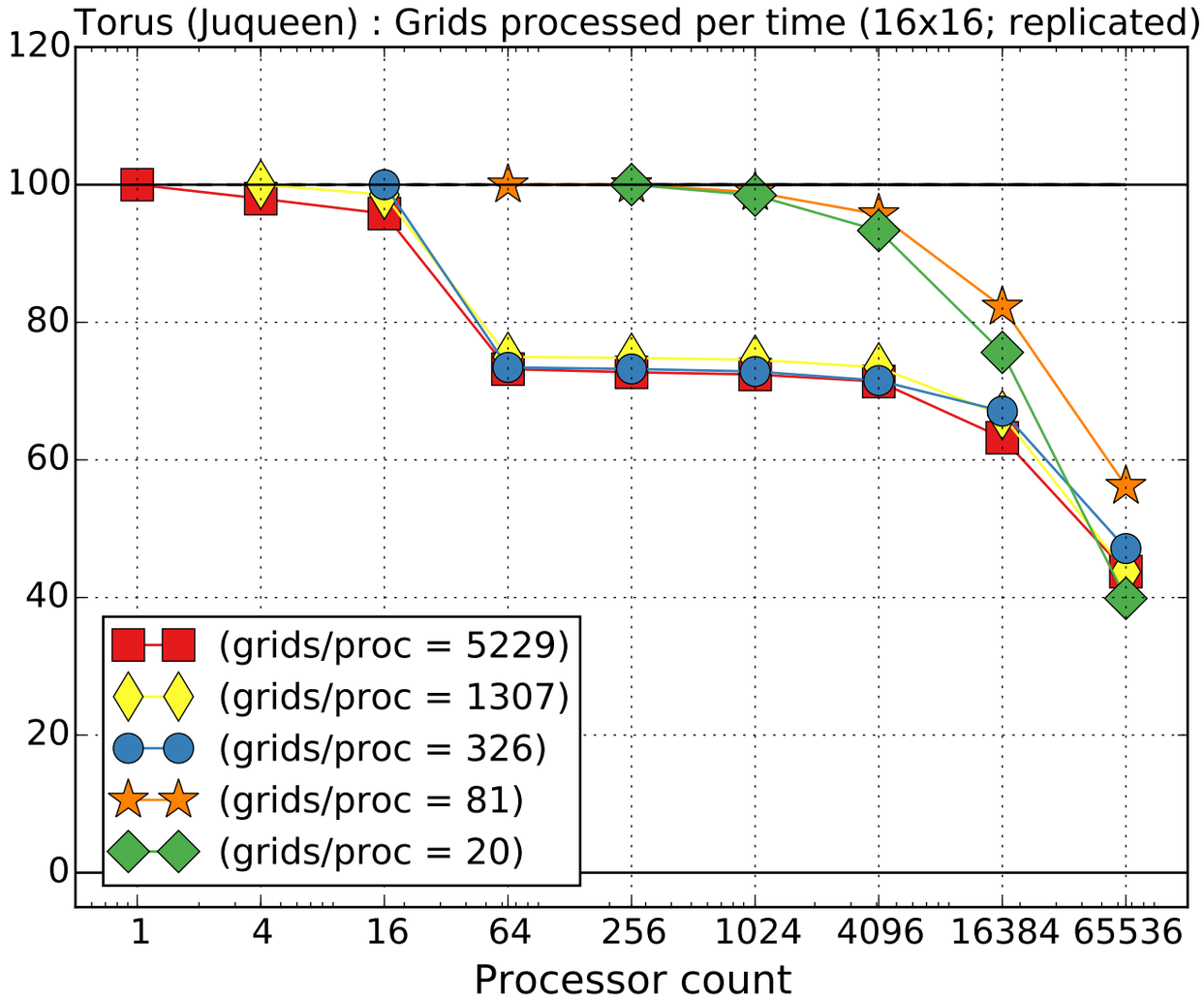
Grids processed per total wall time

Weak Scaling - 16x16 grids

Uniform



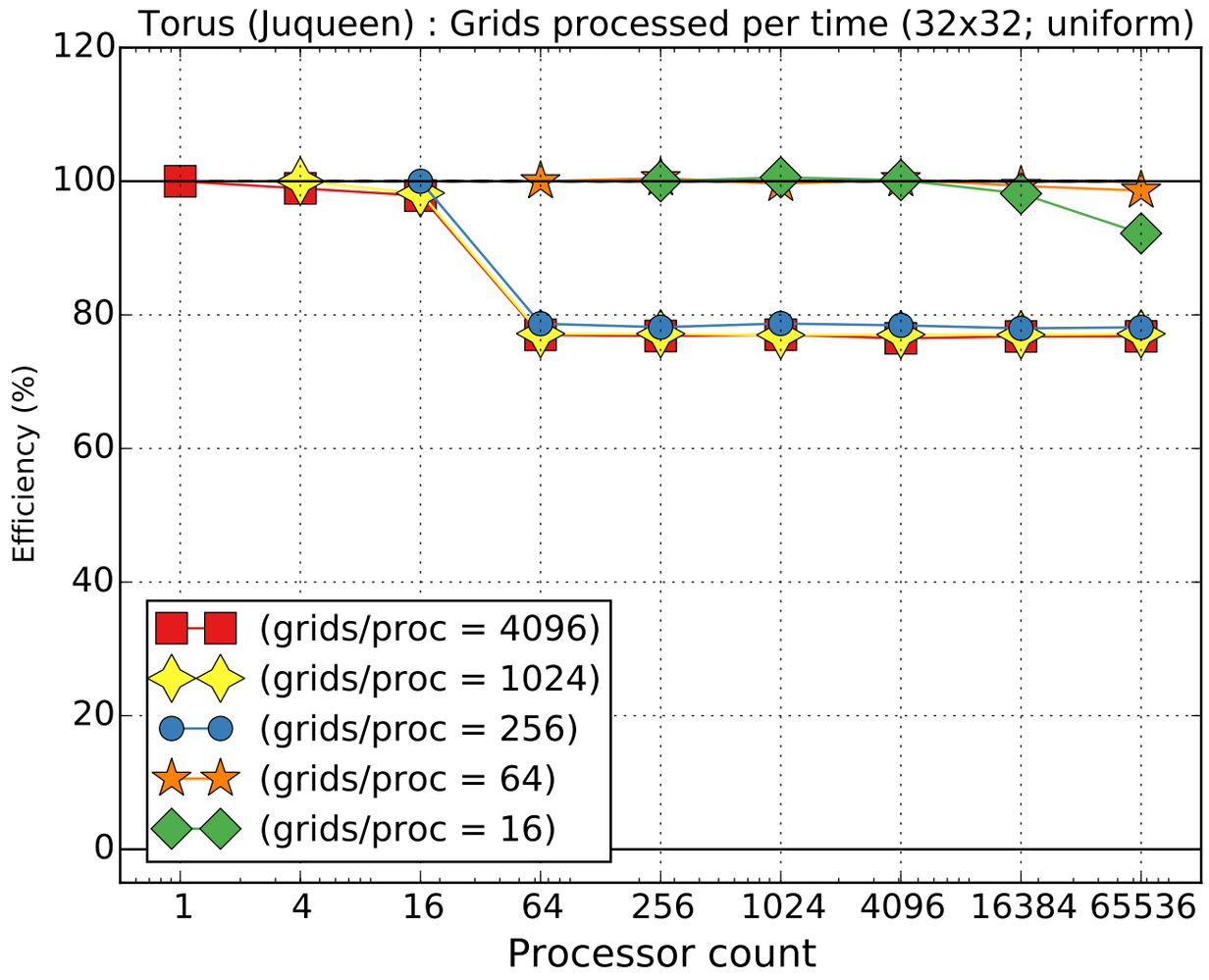
Adaptive



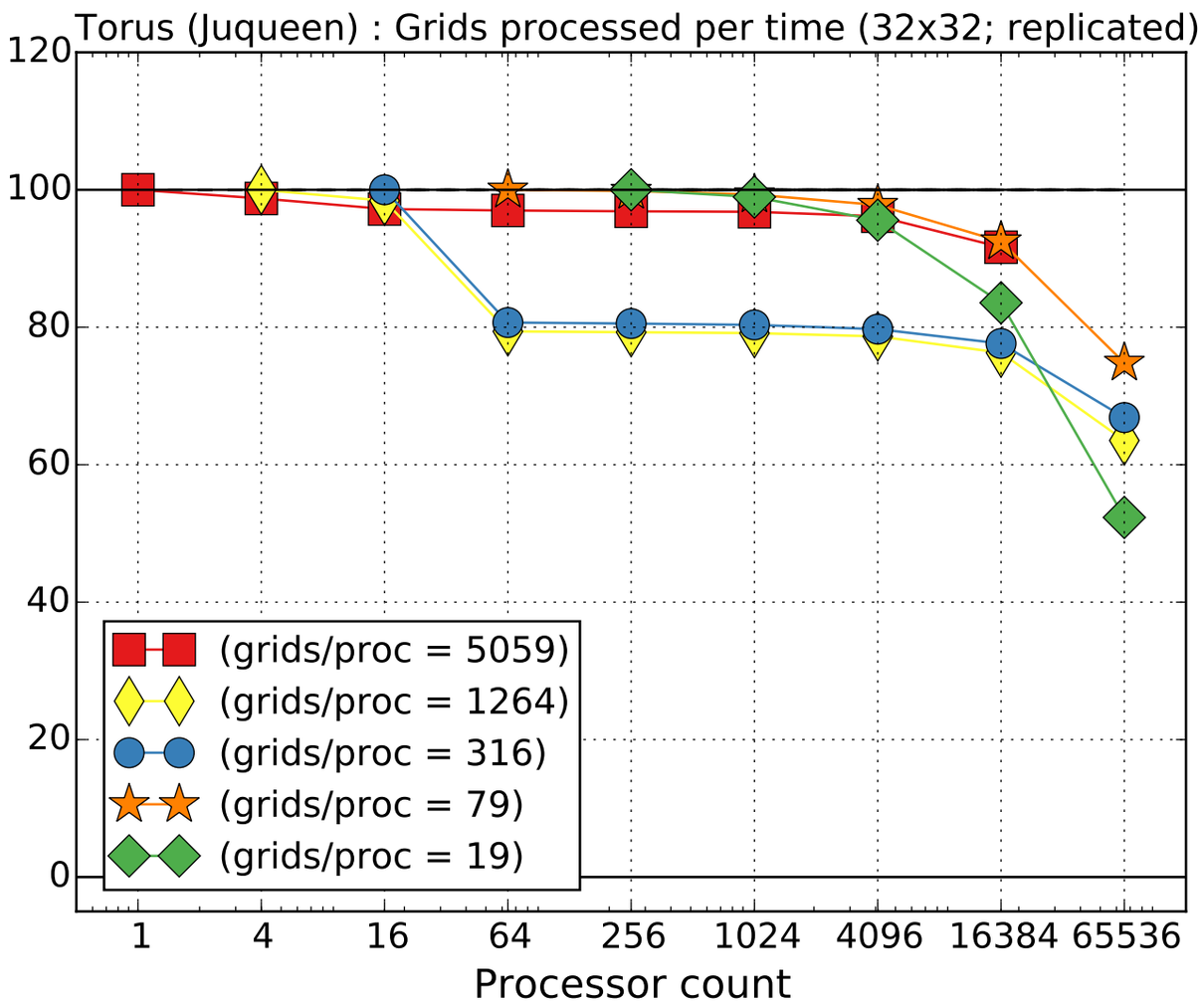
Grids processed per total wall time

Weak Scaling - 32x32 grids

Uniform



Adaptive



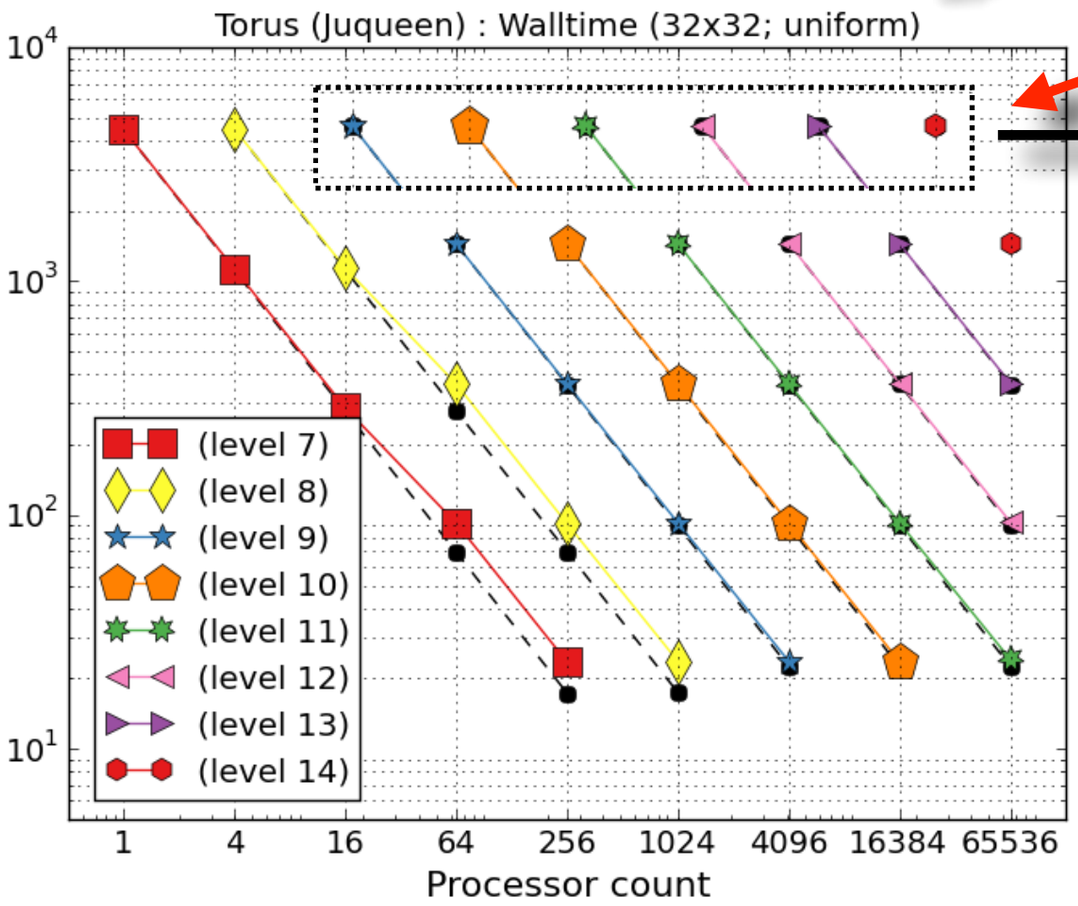
Grids processed per total wall time

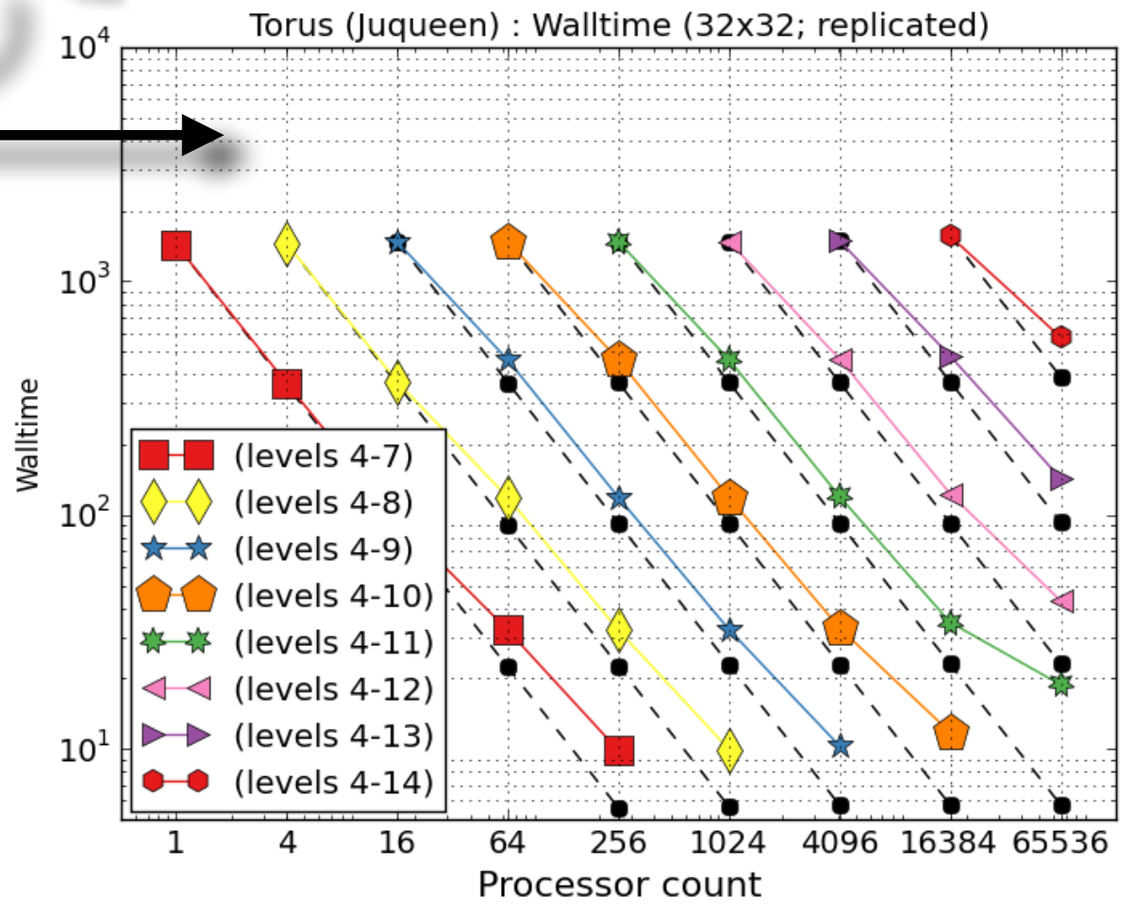
Strong scaling - 32x32

Uniform

not computed

Adaptive

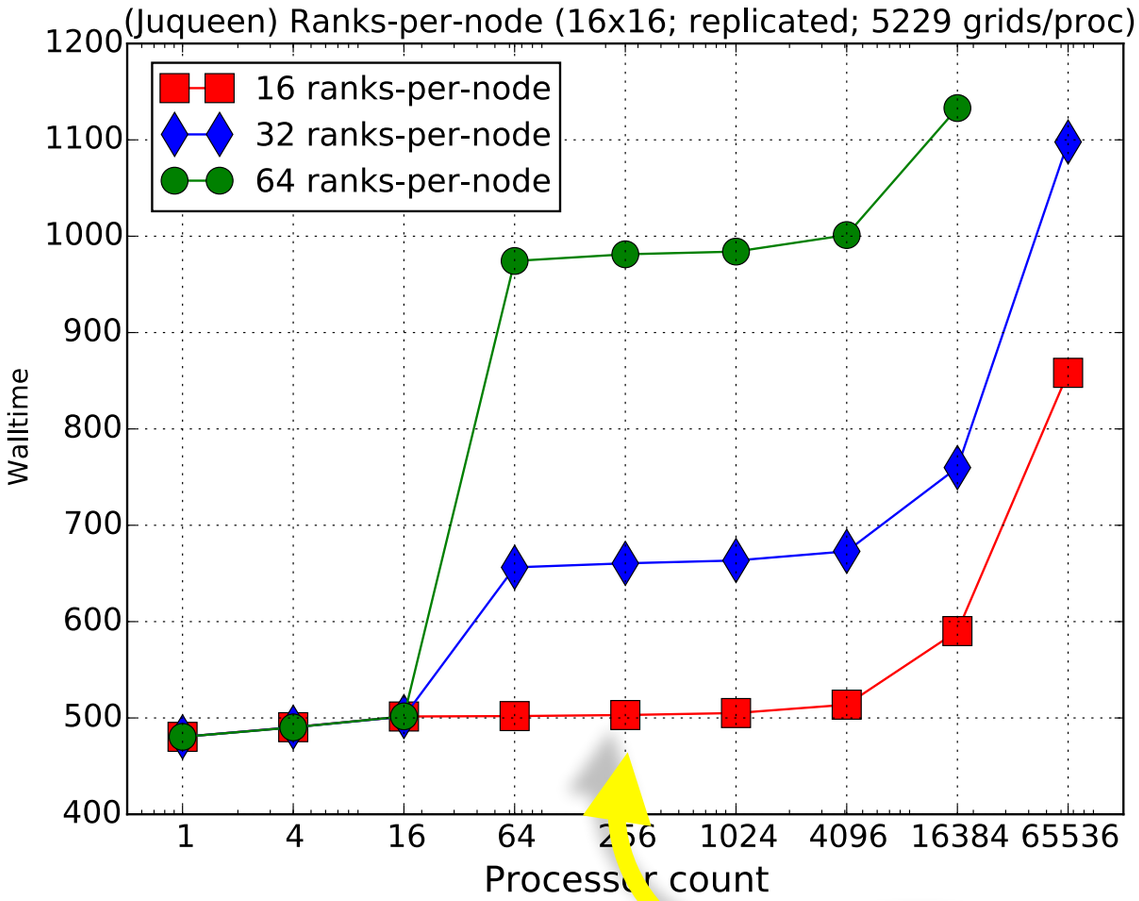




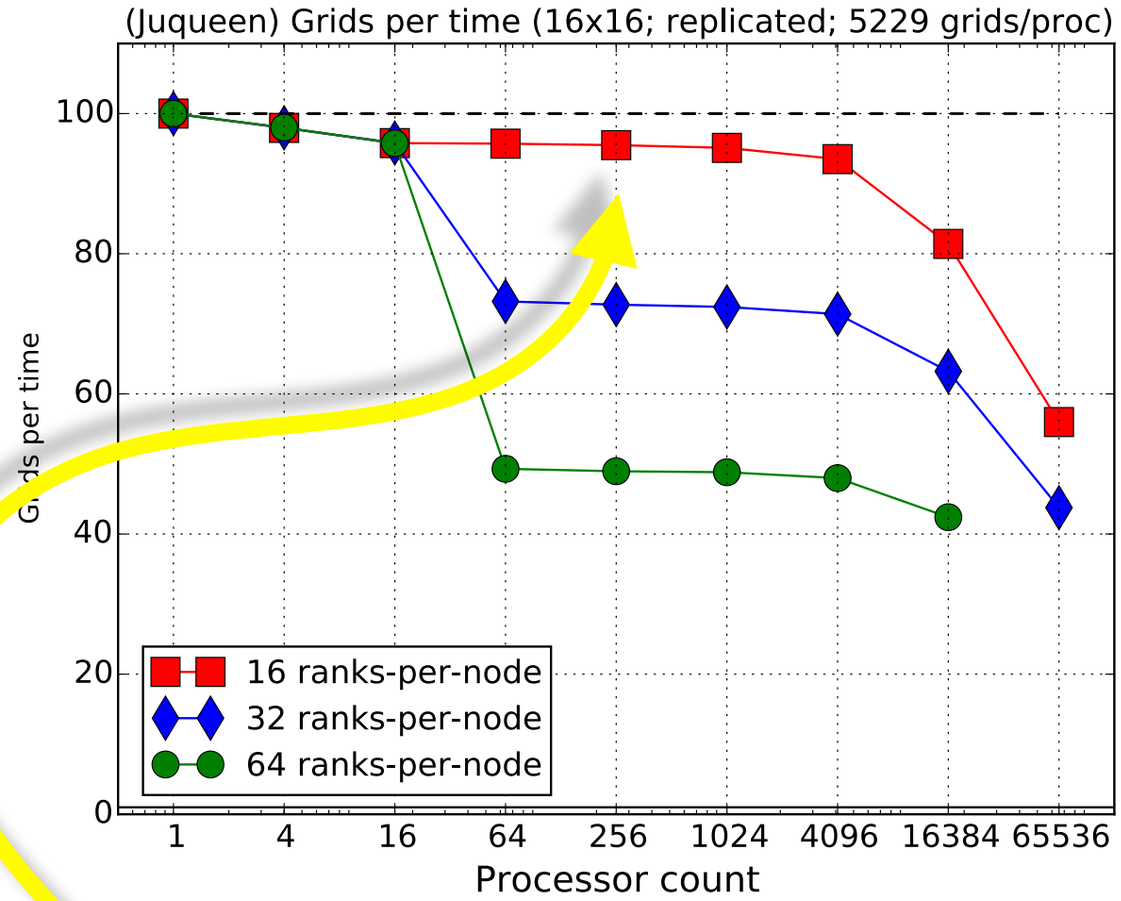
AMR is faster by about a factor of 10

Ranks-per-node

Walltime



Grids per time

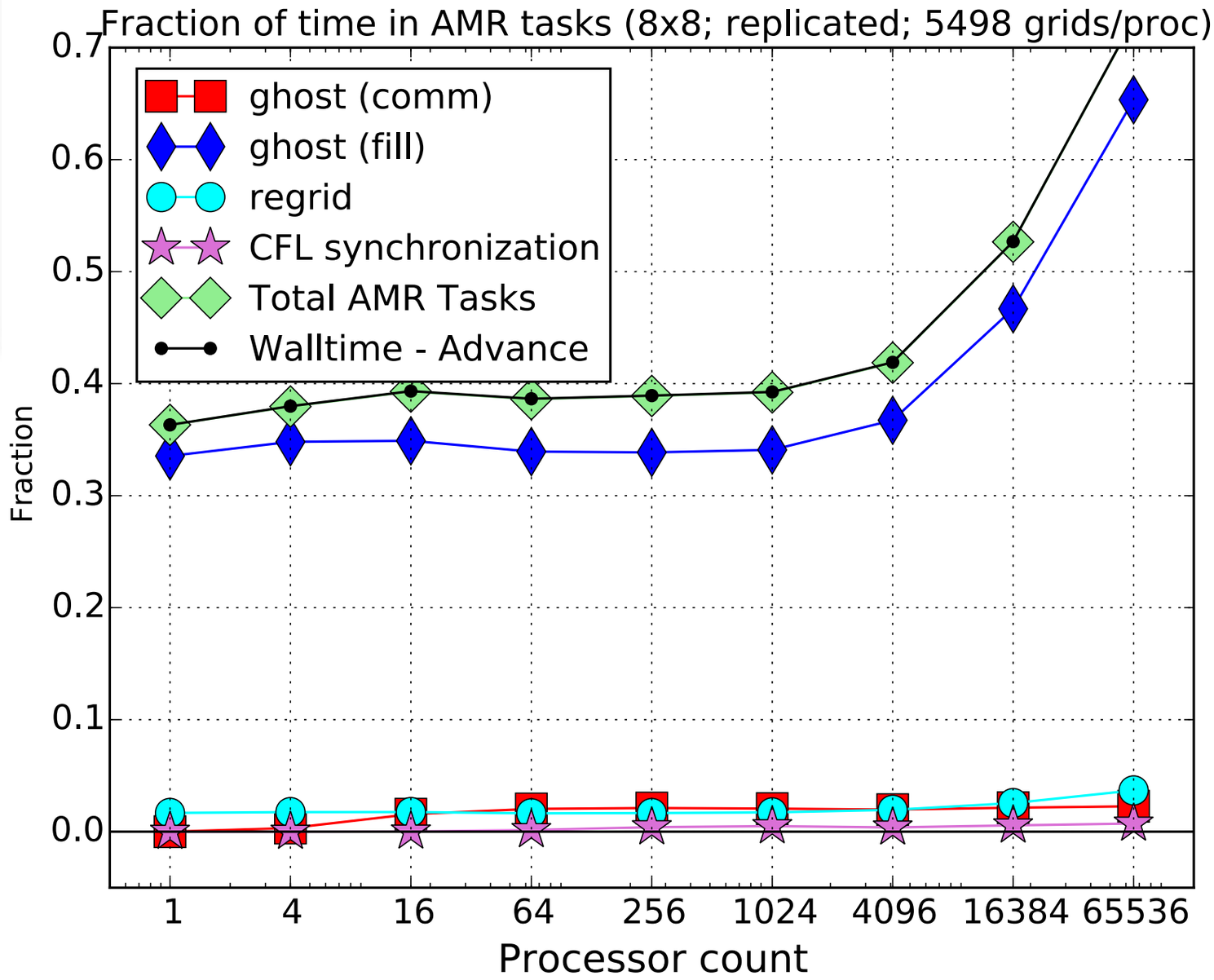


More ranks-per-node means fewer nodes are used, which is more efficient in terms of using allotted CPU hours. (Thanks to Kay Thust, Juelich SC Center)

Dip in performance disappears with when ranks-per-node equals number of cores per node.

What does AMR cost? (8x8 grids)

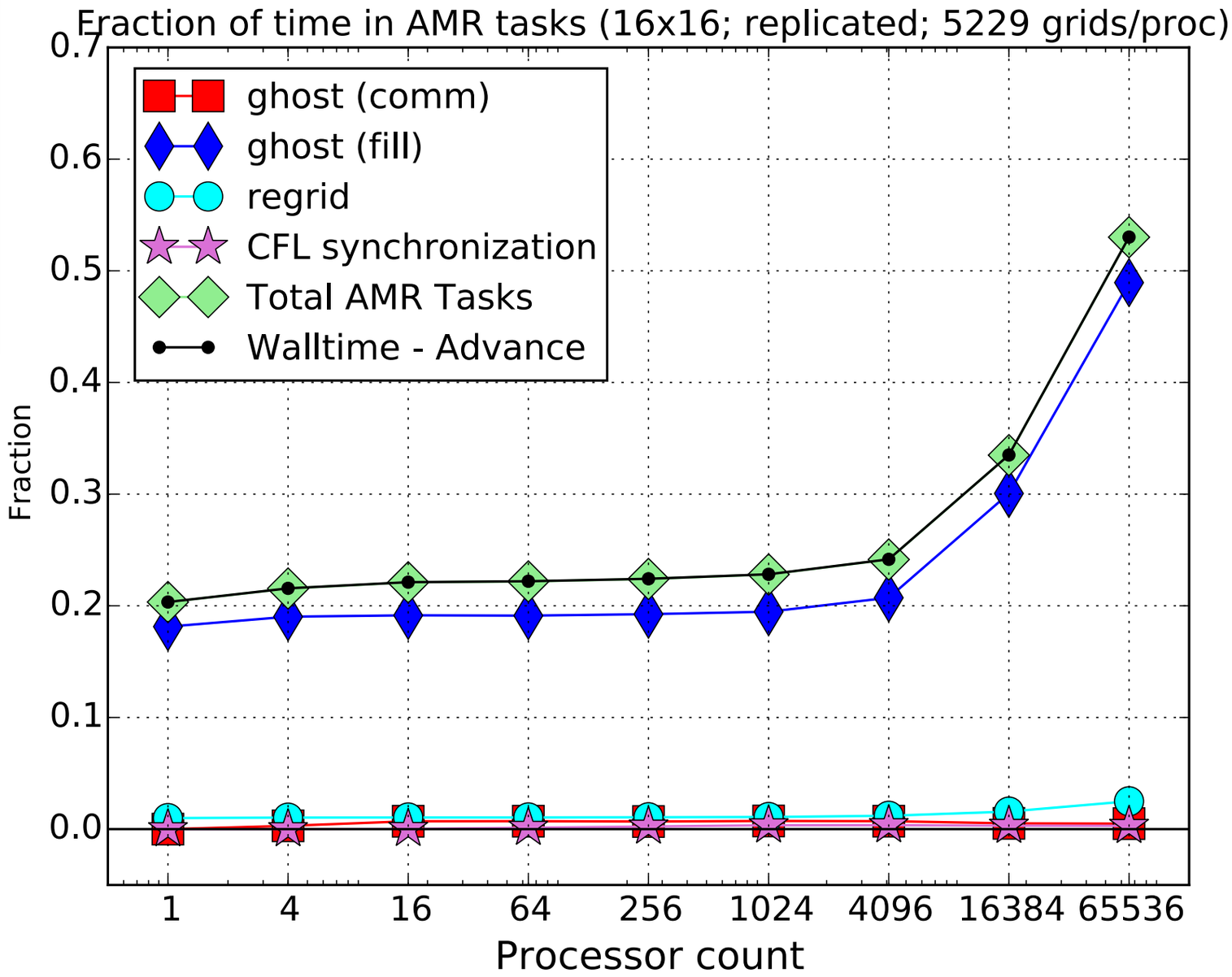
160 total time steps; regridding every coarse grid step (20 regrid steps in total.)



40% in amr related tasks, including communication

What does AMR cost? (16x16 grids)

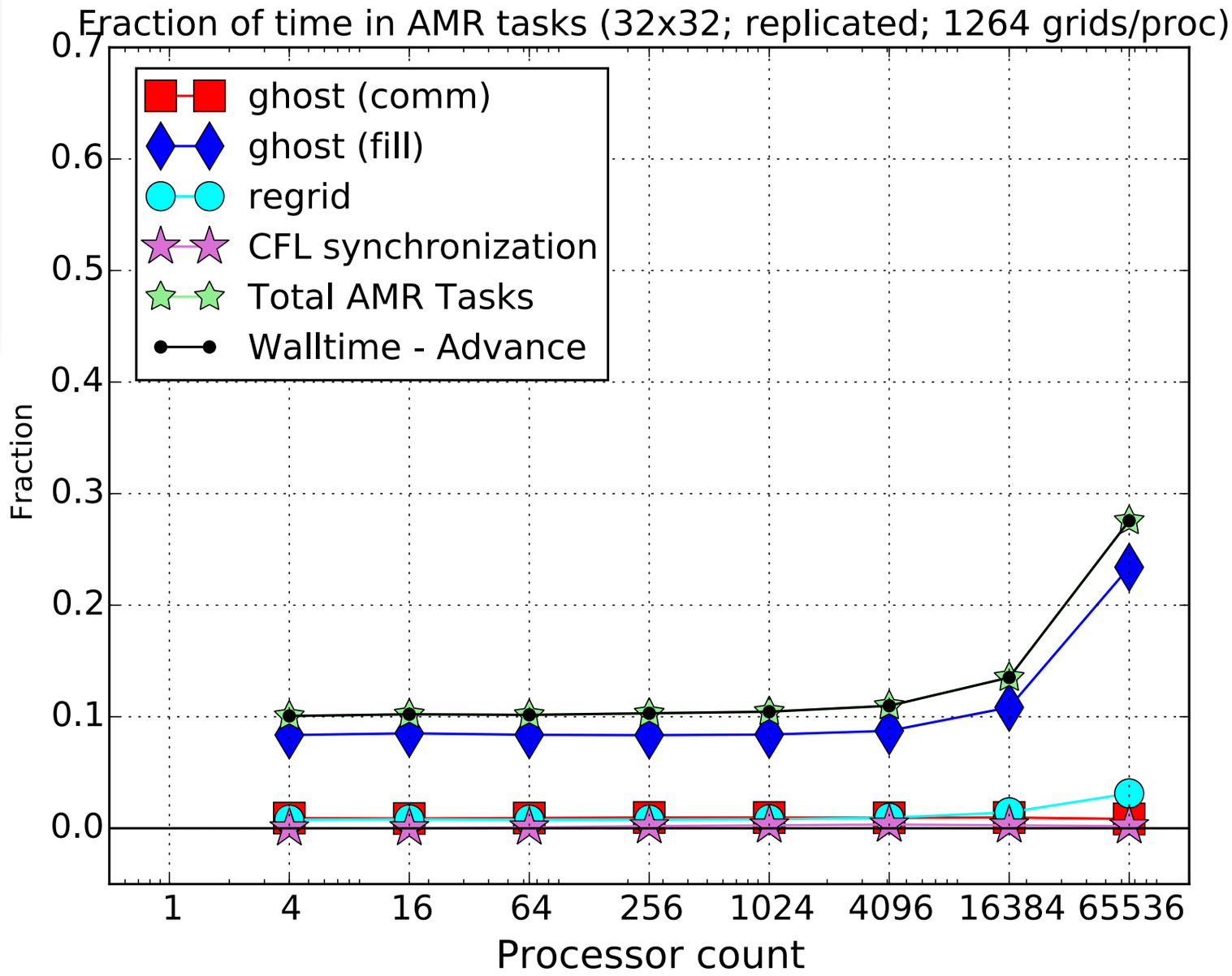
160 total time steps; regridding every coarse grid step (20 regrid steps in total.)



20% in amr related tasks, including communication

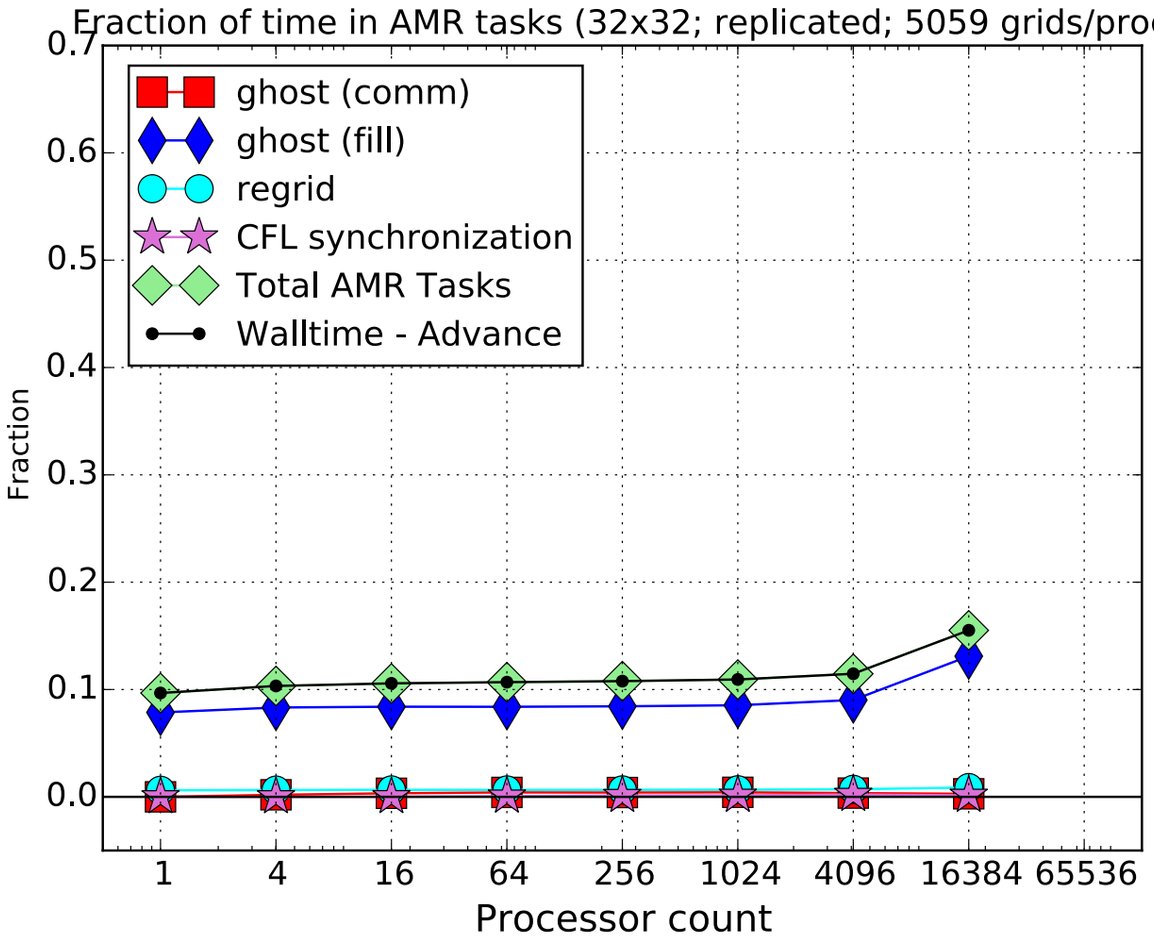
What does AMR cost? (32x32 grids)

160 total time steps; regridding every coarse grid step (20 regrid steps in total.)



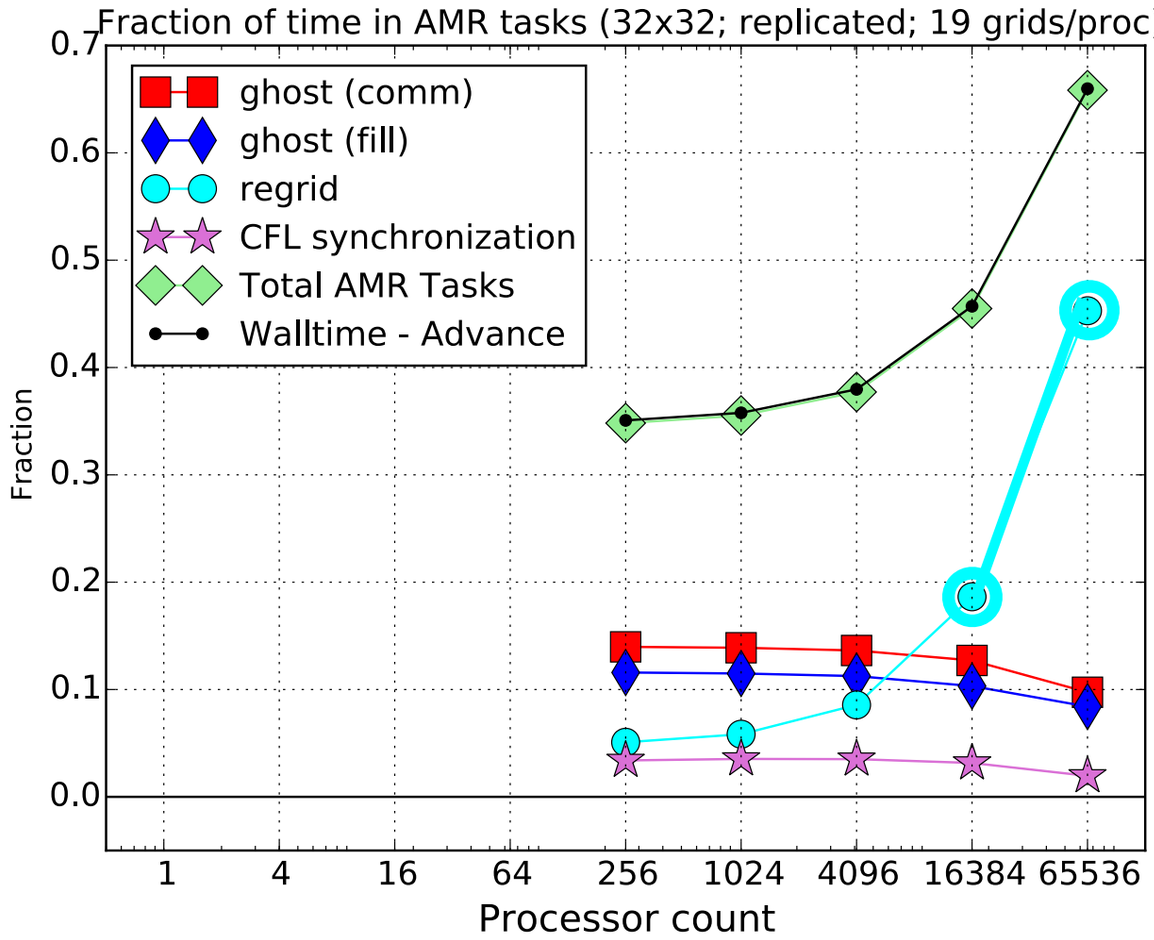
10% in amr related tasks, including communication

What does AMR cost? (32x32 grids)



5059 grids/proc

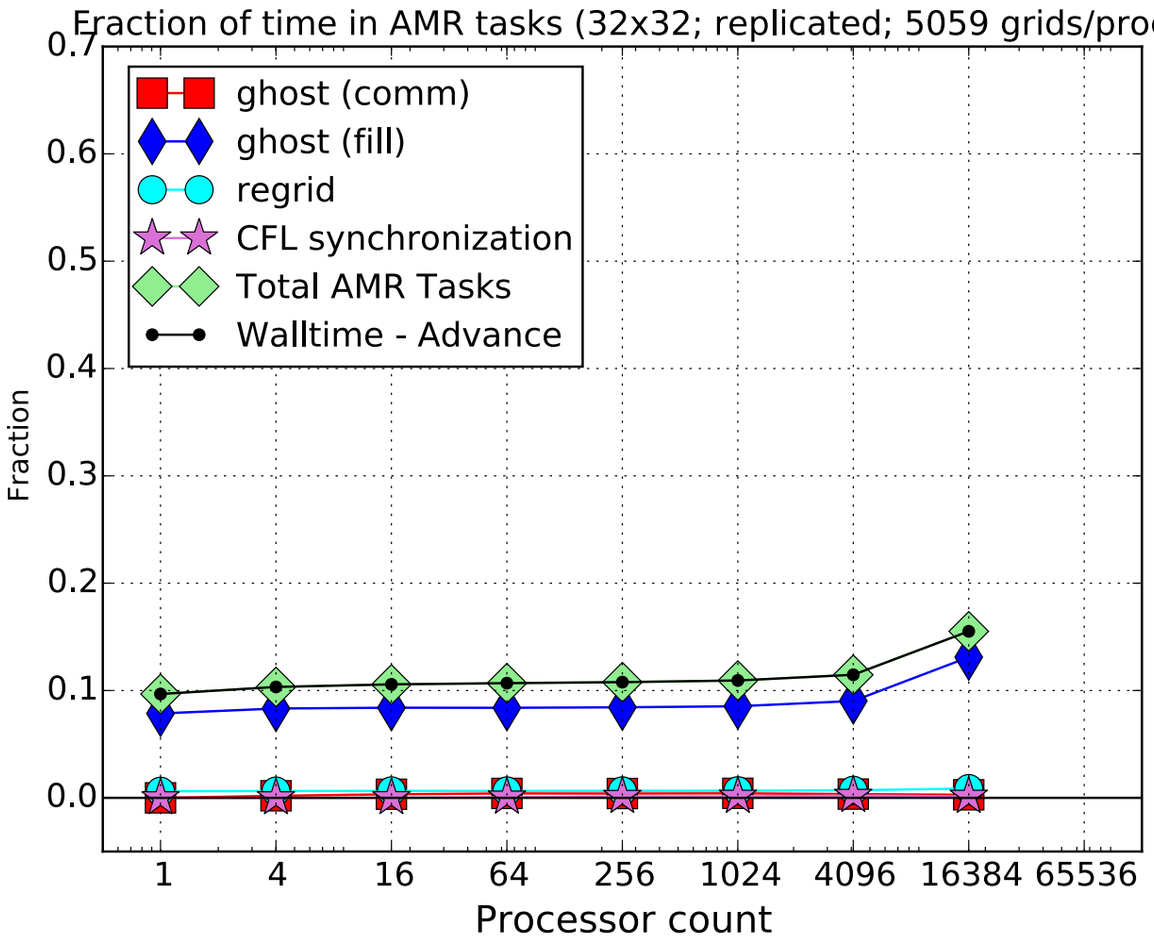
Ghost filling dominates cost



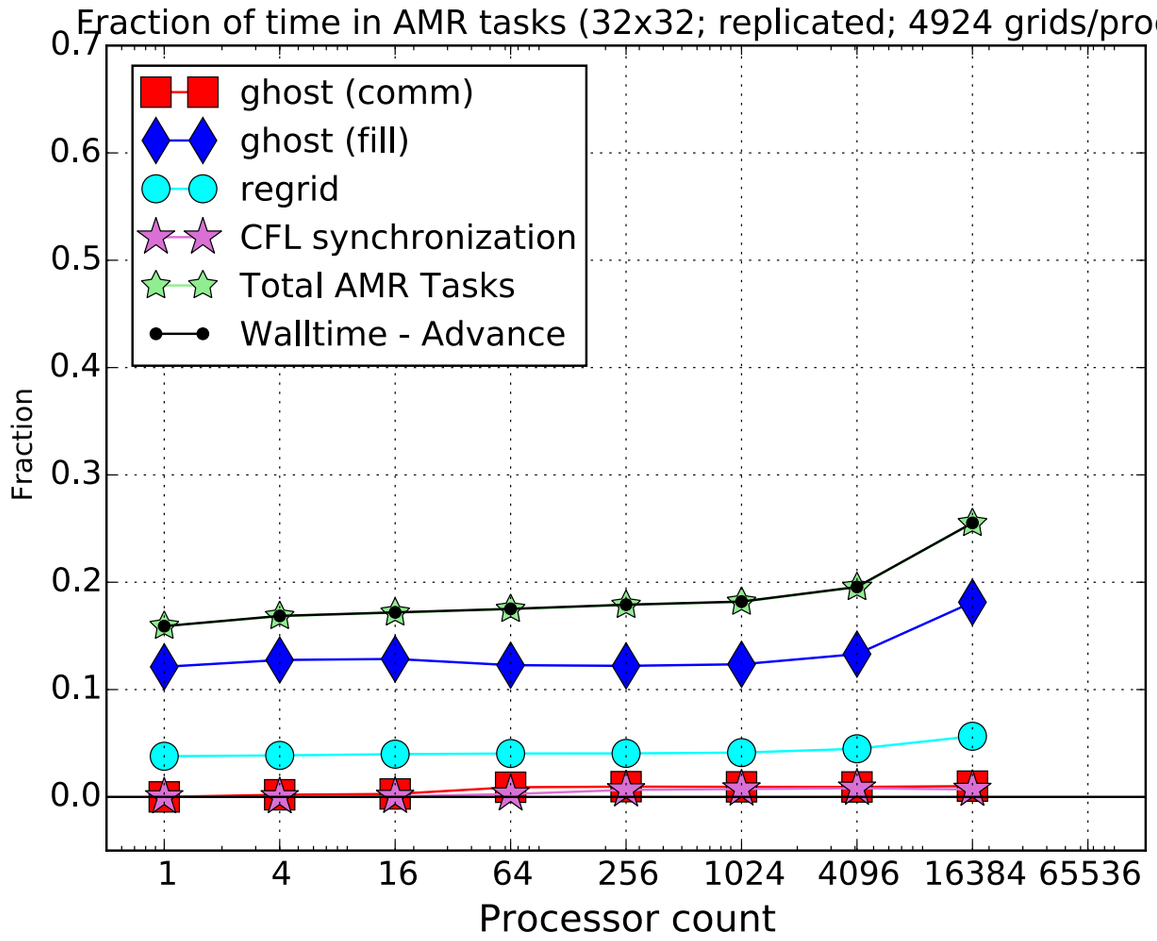
19 grids/proc

Regridding dominates cost

What does AMR cost? (32x32 grids)



Regridding every 8 time steps

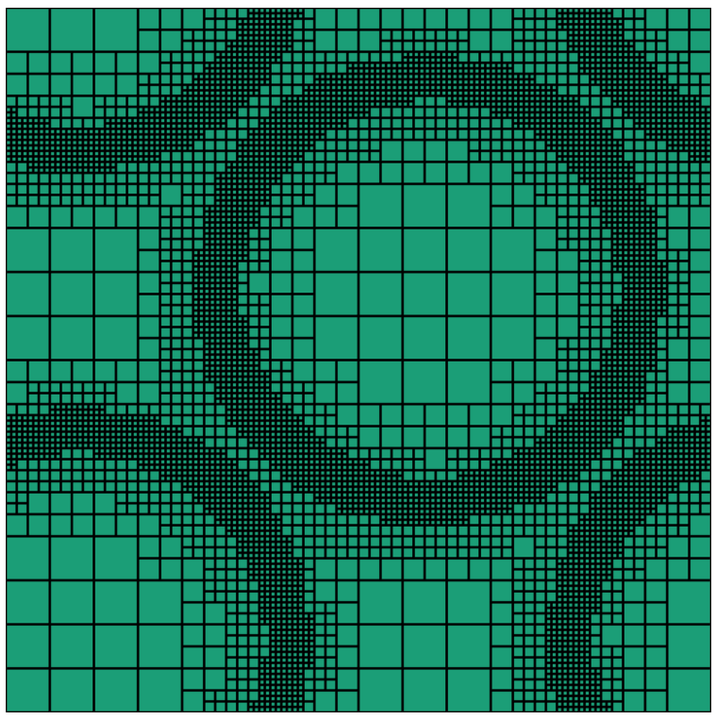


Regridding every time step*

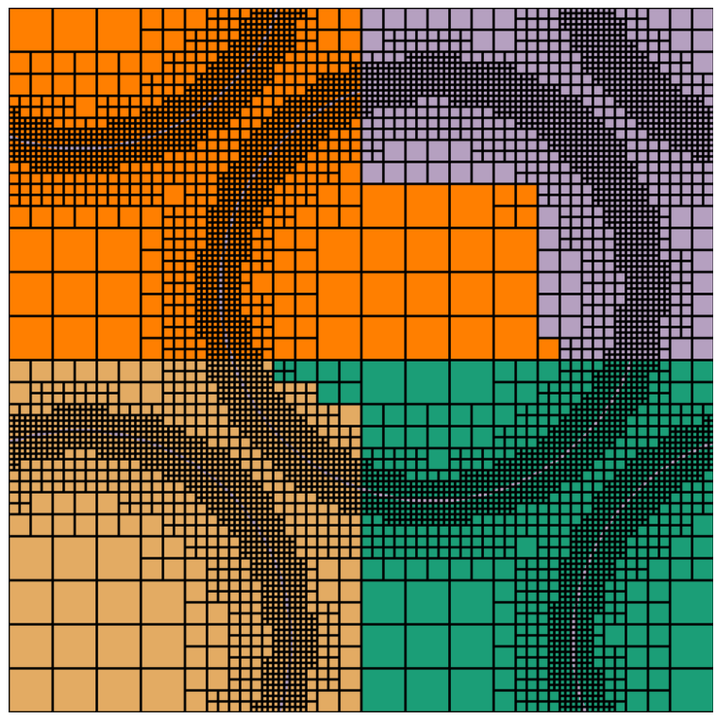
*Initial gridding not included

Scaling of single adaptive problem

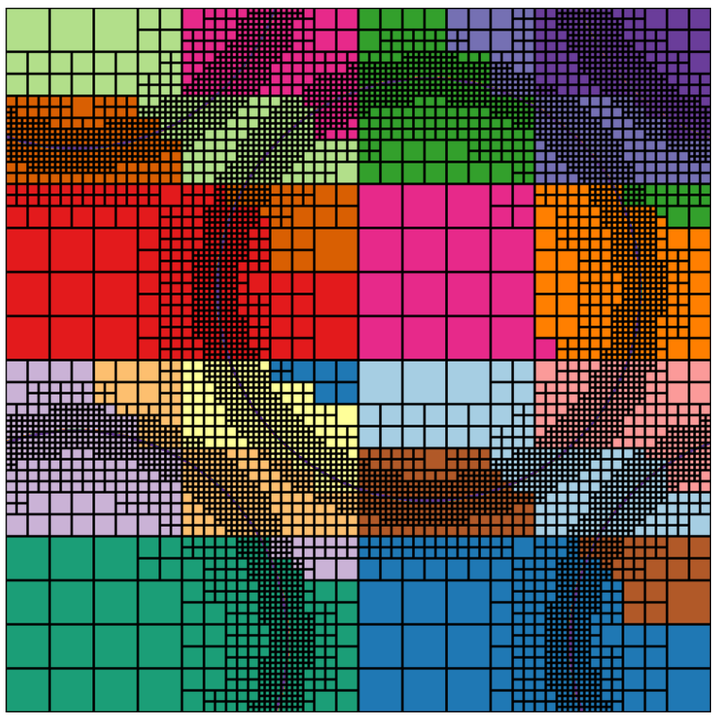
1 proc



4 procs



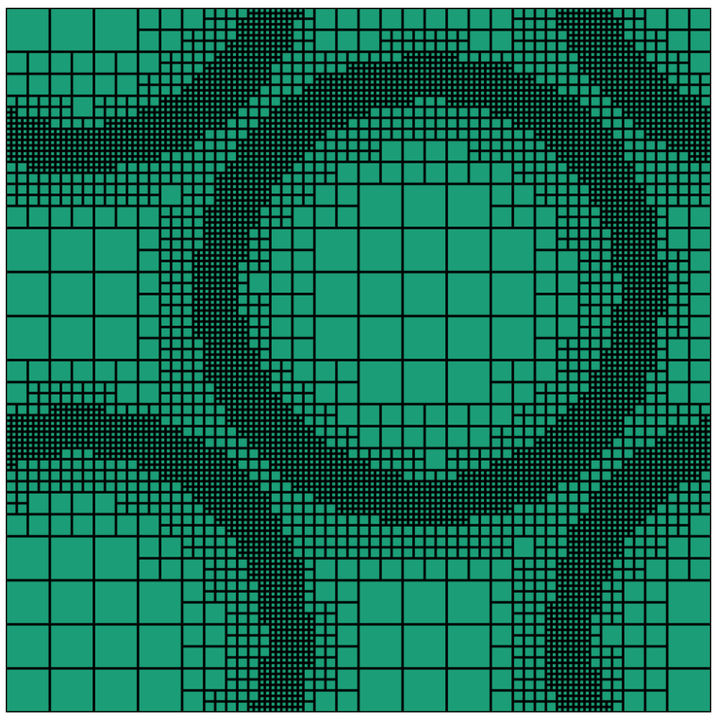
16 procs



…,64, 256, 16384, 65536

Scaling of single adaptive problem

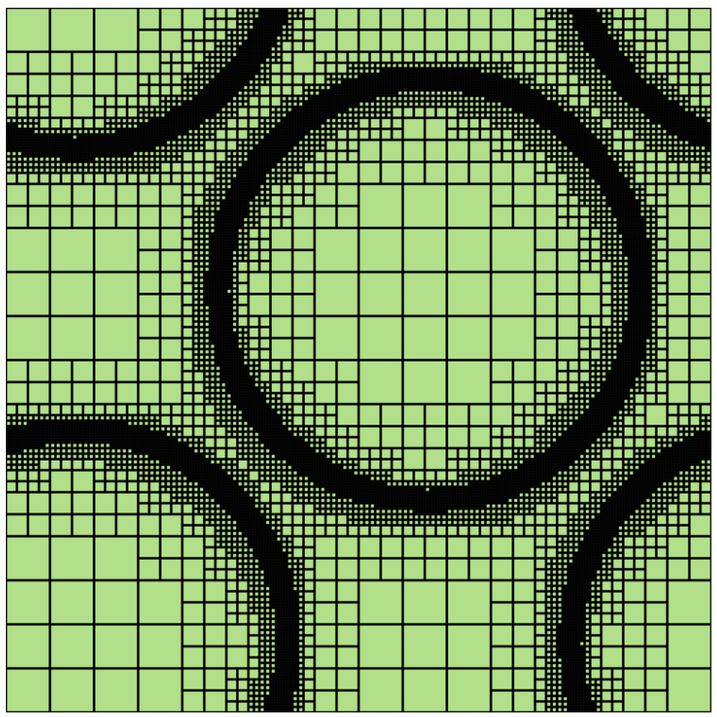
Levels 4-7



level 4 grids

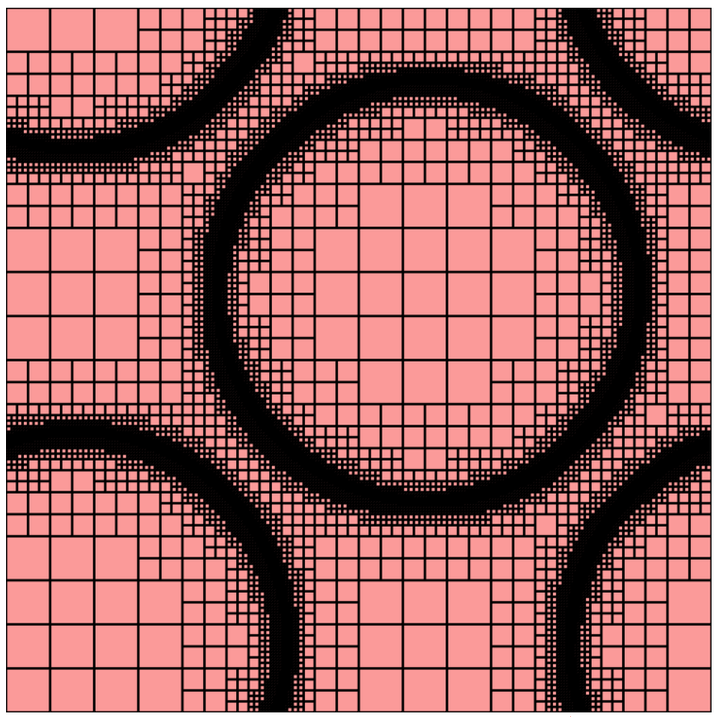
level 7 grids

Levels 4-8



level 8 grids

Levels 4-9



level 9 grids

Mimic scaling of uniform problem : Increase resolution and processor count simultaneously.

Adaptive scaling

Keep minimum level fixed;
Increase maximum refinement level

Strong scaling

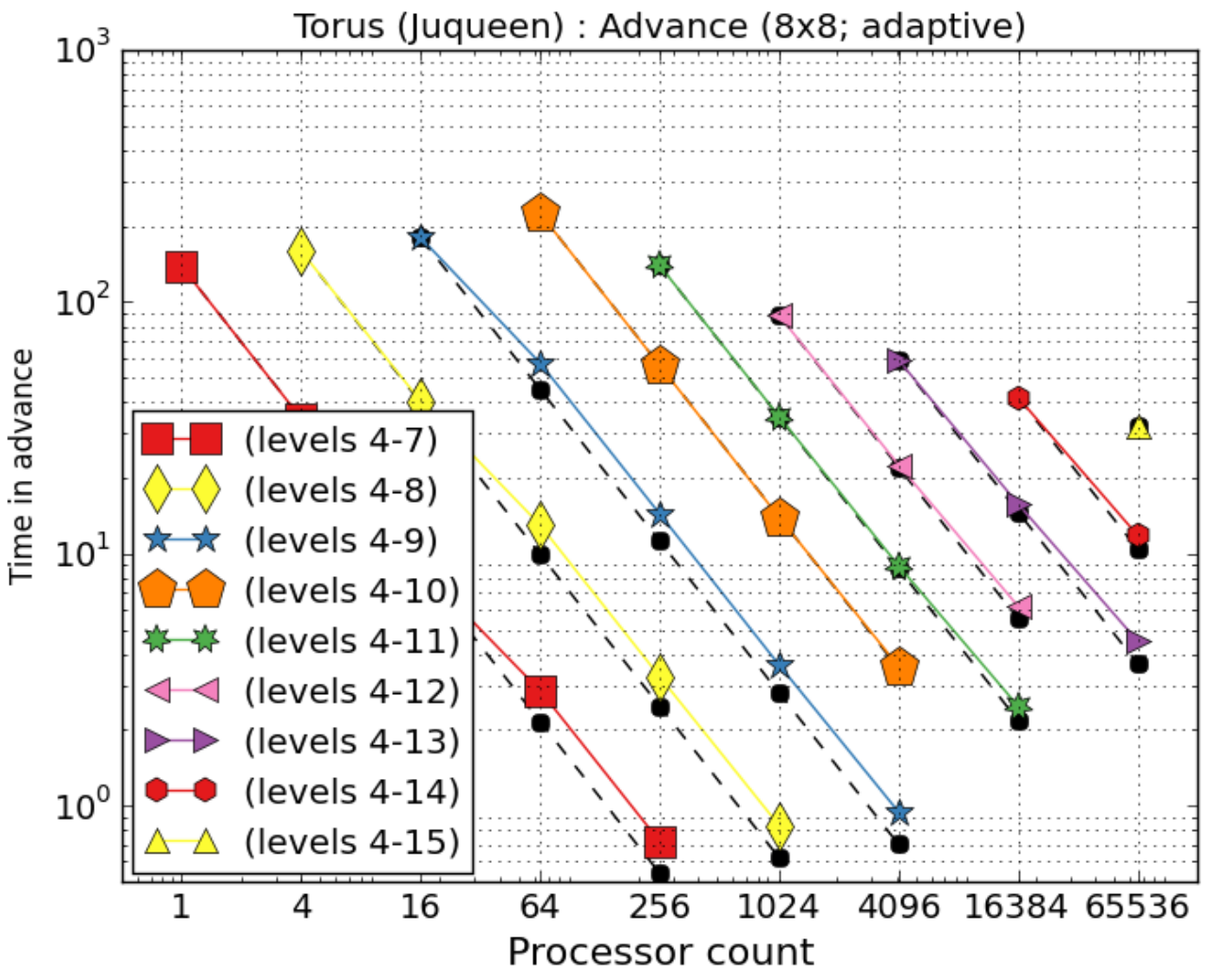
Weak scaling

mx = 8	4-7	4-8	4-9	4-10	4-11	4-12	4-13	4-14	4-15
1	5058	---	---	---	---	---	---	---	---
4	1264	2876	---	---	---	---	---	---	---
16	316	719	1579	---	---	---	---	---	---
64	79	179	394	764	---	---	---	---	---
256	19	44	98	191	239	---	---	---	---
1024	---	11	24	47	59	75	---	---	---
4096	---	---	6	11	14	18	24	---	---
16384	---	---	---	---	3	4	6	8	---
65536	---	---	---	---	---	---	1	2	2

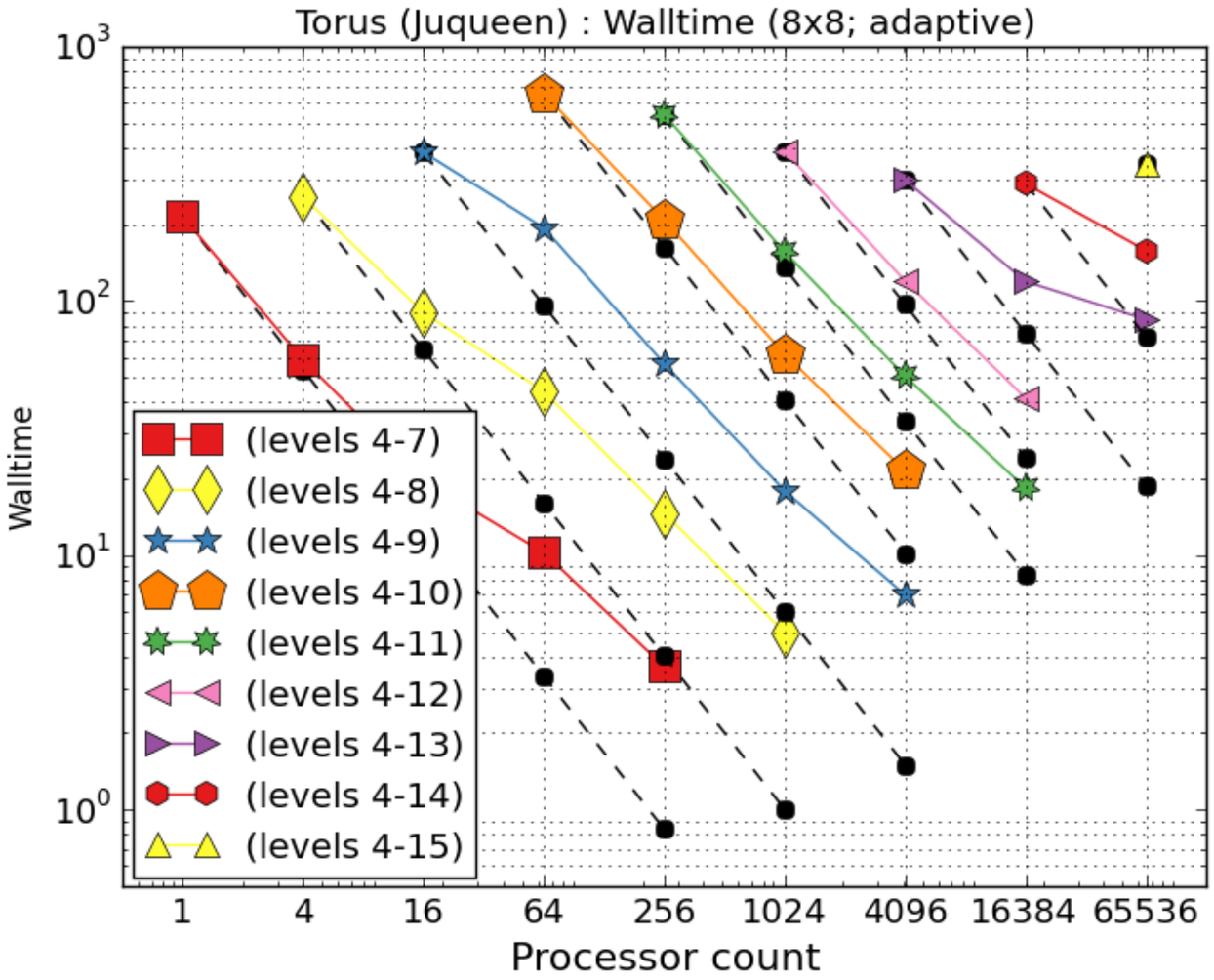
- Fixed number of coarse grid time steps; final time depends on grid size.
- Strong scaling results similar to those for replicated problem.

Strong scaling - adaptive - 8x8

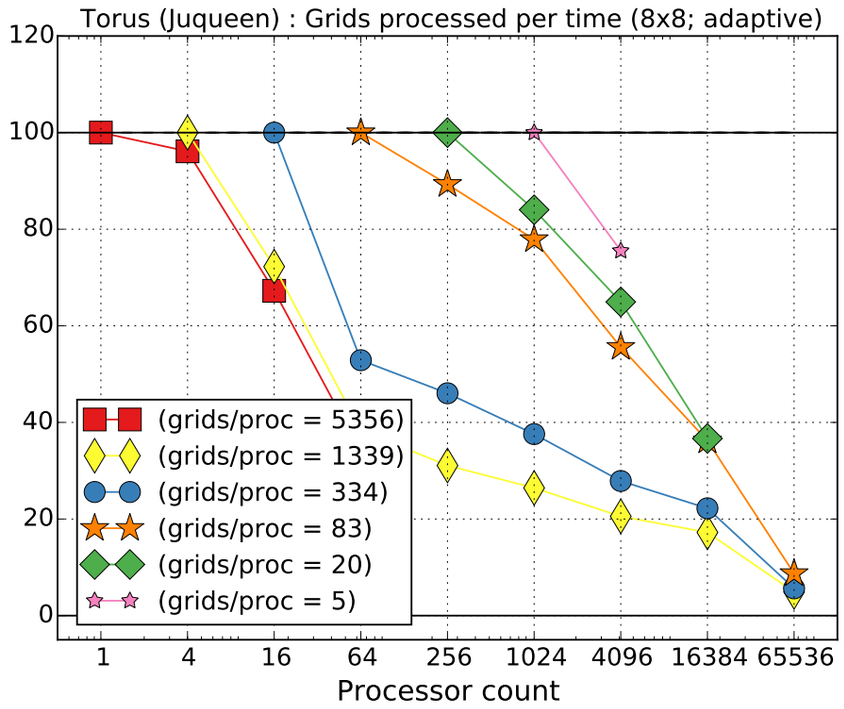
Time in advance



Total walltime



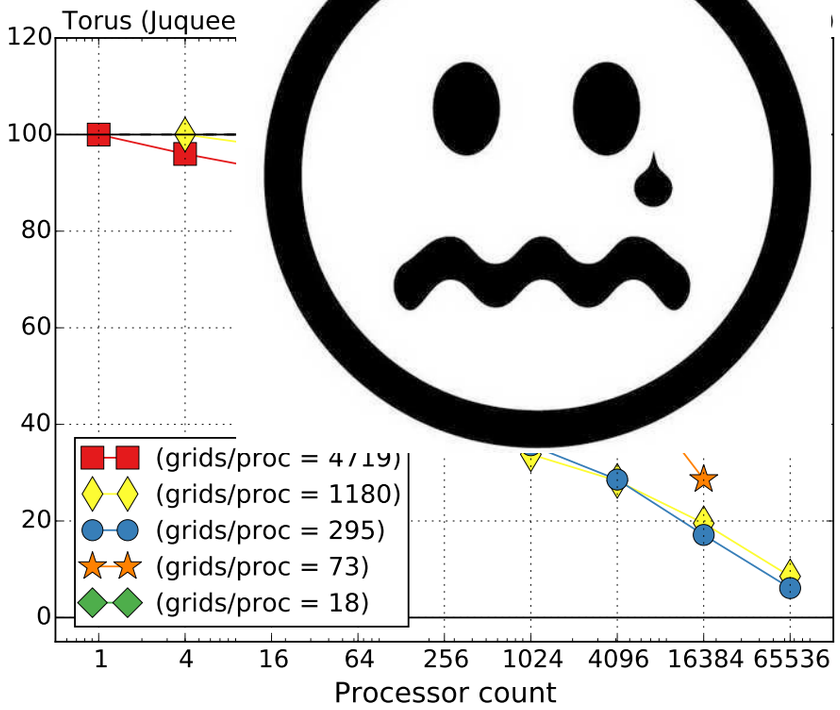
Weak scaling - adaptive



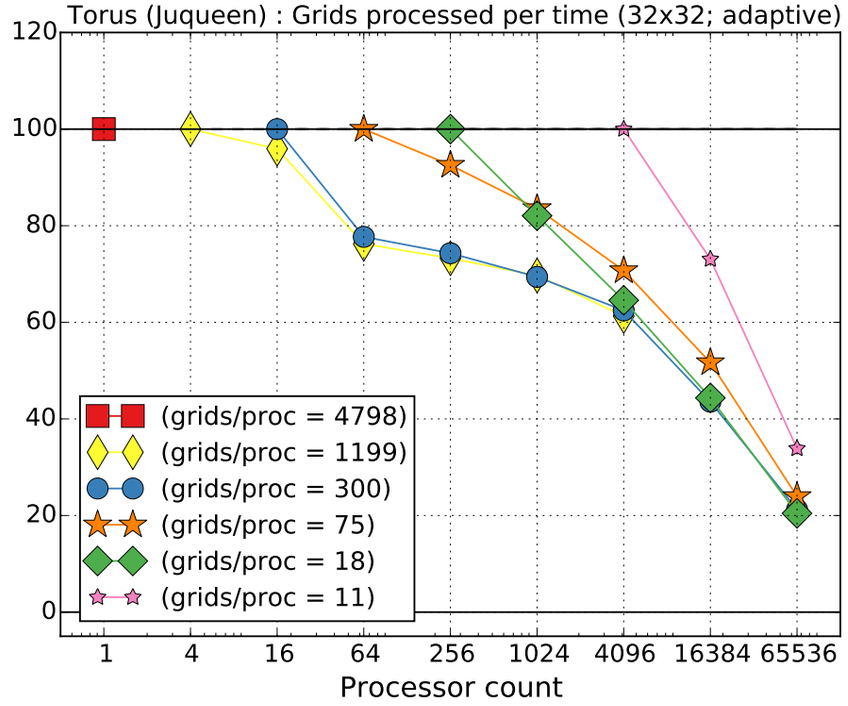
8x8

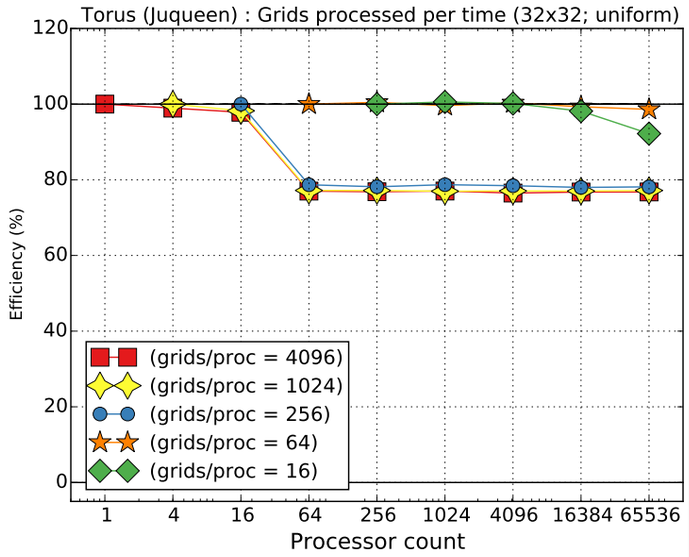
Adaptive (non-replicated) results

16x16



32x32



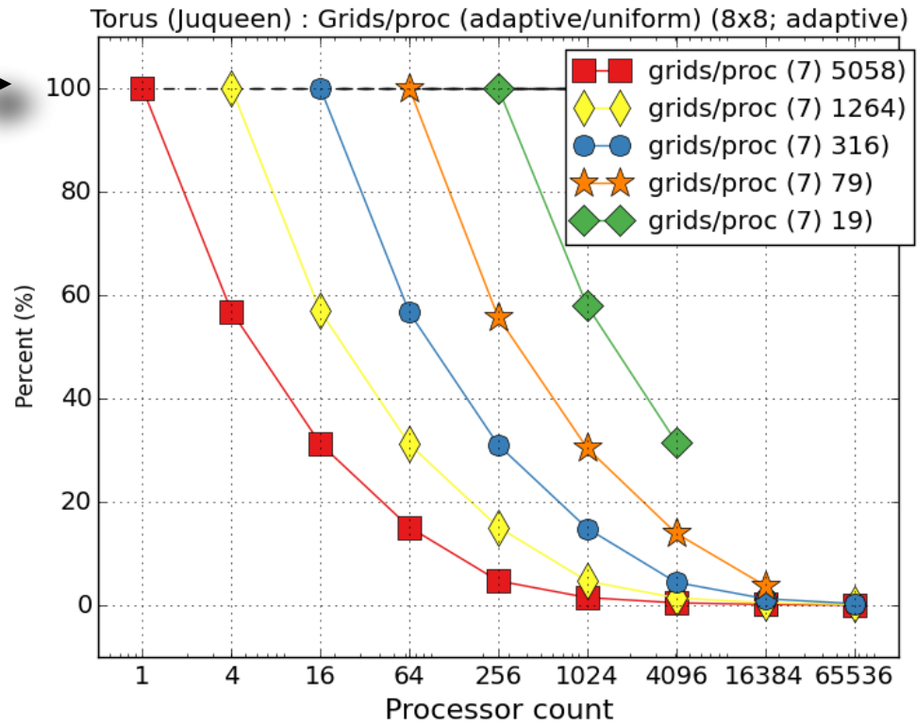


Uniform results

Weak scaling results

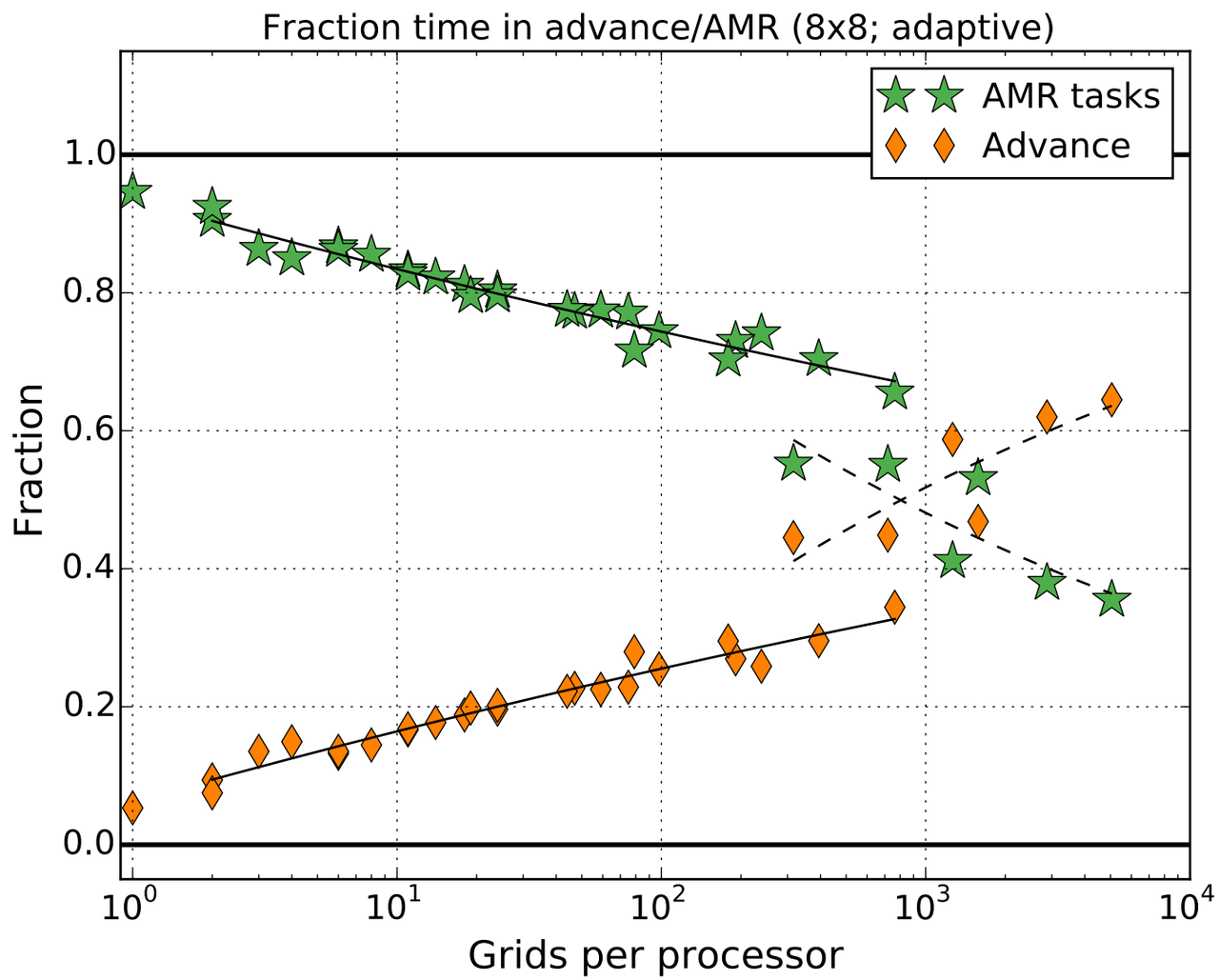
mx = 8	4-7	4-8	4-9	4-10	4-11	4-12	4-13	4-14	4-15
1	**5058**	---	---	---	---	---	---	---	---
4	1264	**2876**	---	---	---	---	---	---	---
16	316	719	**1579**	---	---	---	---	---	---
64	79	179	394	**764**	---	---	---	---	---
256	19	44	98	191	**239**	---	---	---	---
1024	---	11	24	47	59	**75**	---	---	---
4096	---	---	6	11	14	18	**24**	---	---
16384	---	---	---	---	3	4	6	**8**	---
65536	---	---	---	---	---	---	1	2	**2**

Uniform/replicated scaling →



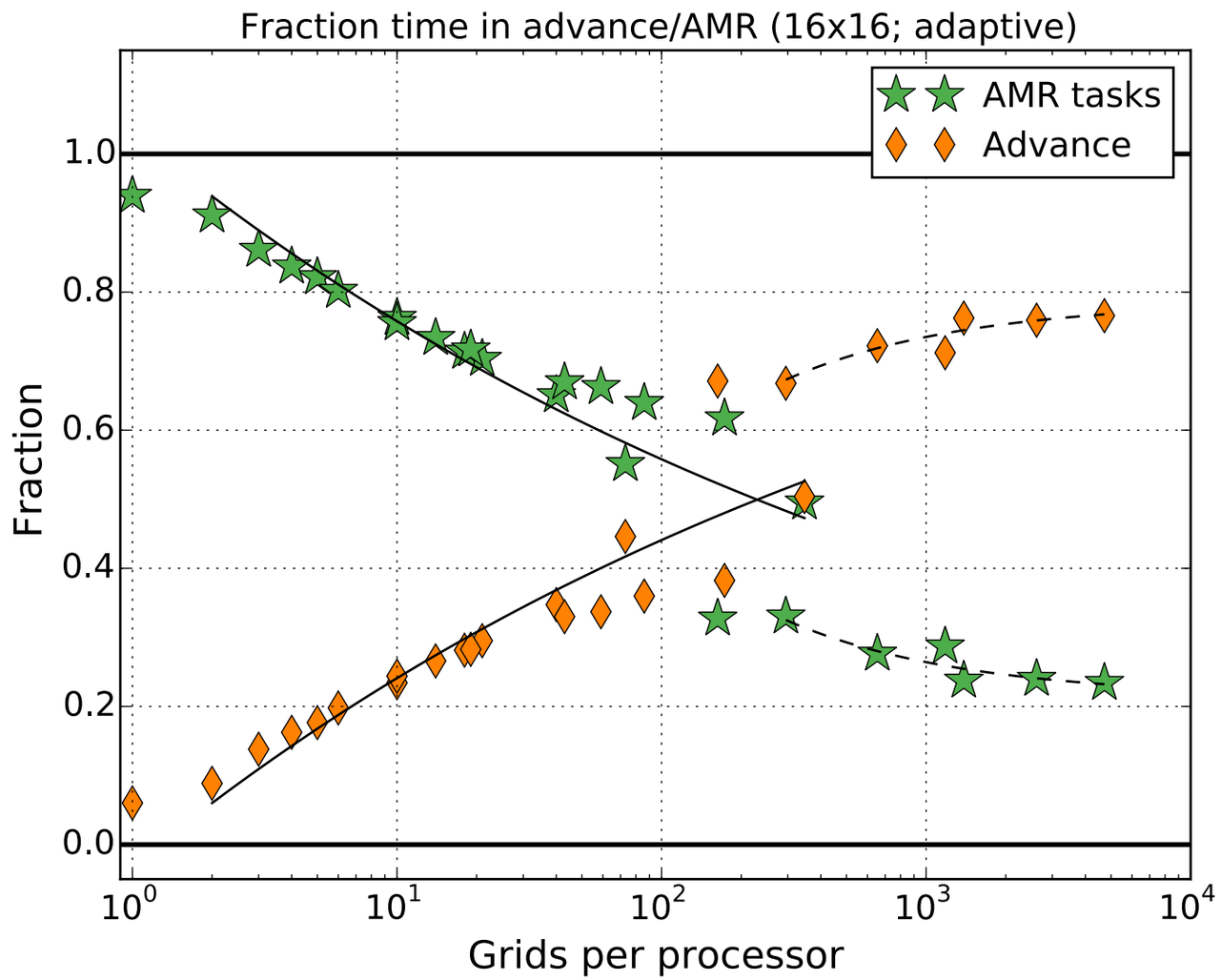
Grids per processor, as percent of grids on processor on original problem.

Adaptive scaling - 8x8



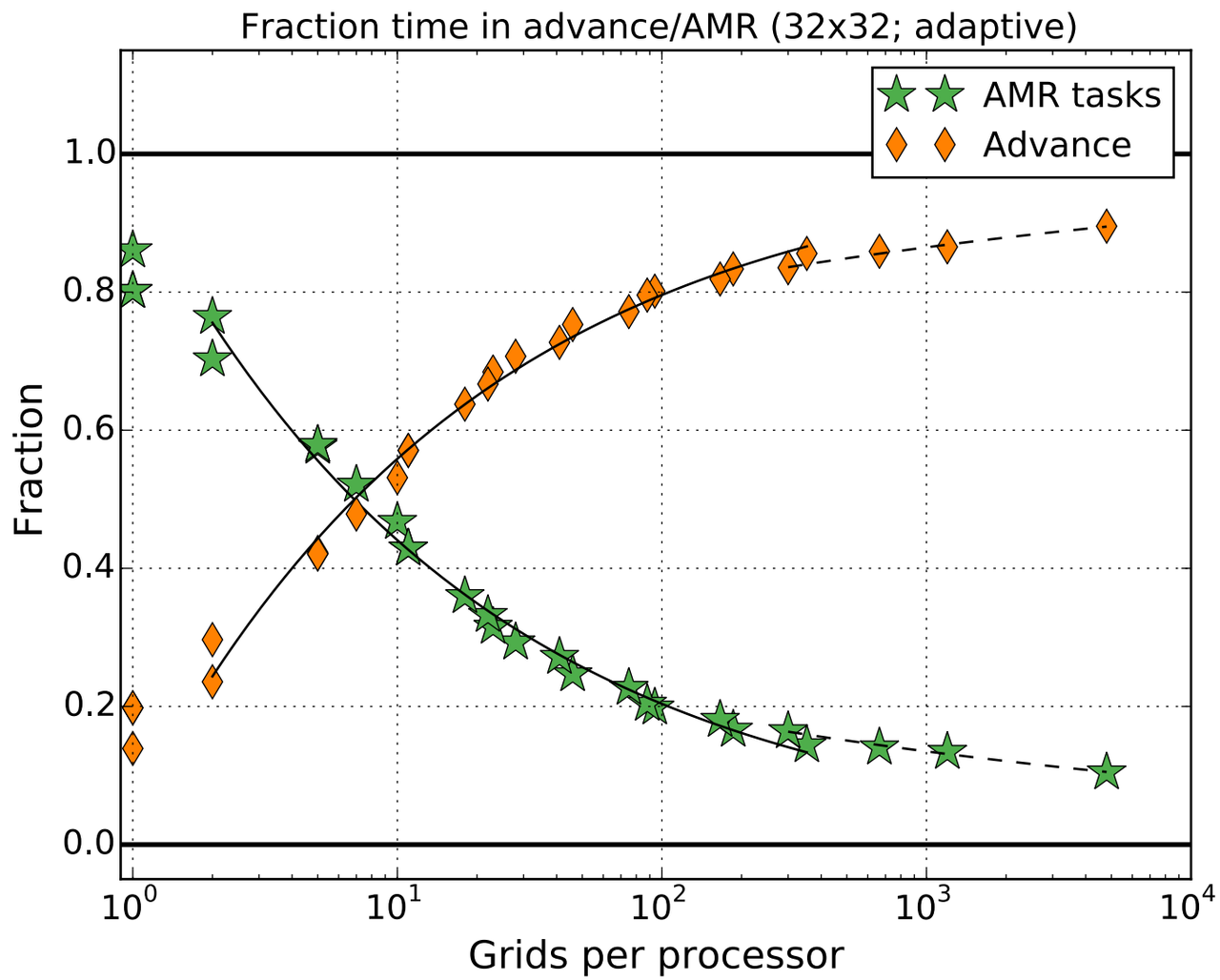
Plot timing results as function of granularity

Adaptive scaling - 16x16



Plot timing results as function of granularity

Adaptive scaling - 32x32



Plot timing results as function of granularity

Conclusions

- 32x32 grids more efficient than 8x8 or 16x16 in terms of parallel performance.
- Uniform problem scales at least to 65536 processors
- Replicated problem scales to at least 16384 processors
- Adaptive problem shows good "adaptive scaling"
- "Adaptive scaling" seems to be a good way to determine efficiency

What next?

- Develop approach for allocating processors based on characteristics of the problem,
- Make better use of latency hiding
- Look at scaling of harder problems
- Develop ForestClaw version of GeoClaw (M. Shih, Columbia University)
- Work towards full 3d simulations, with a stop along the way at 2.5d (refinement in horizontal only) for modeling ash cloud transport.

  ★ Collaboration with the USGS on developing a transport code for volcanic ash dispersion (Ash3d : L. Mastin, R. Denlinger, H. Schwaiger, 2012)

www.forestclaw.org

MS72 - Friday - 10:35 - Delarue

Friday, April 15

MS72
Scientific Computing with Parallel Adaptive Mesh Refinement Methods

10:35 AM - 12:15 PM
Room: Delarue

In the past years the popularization of important parallel computational resources have helped spreading parallel Adaptive Mesh Refinement (AMR) across a large range of applications that use scientific computing. We propose in this minisymposium to present a sample of these applications. We shall focus on the specific AMR method that was chosen, its implementation in a parallel framework and how these choices impact the discretization strategy.

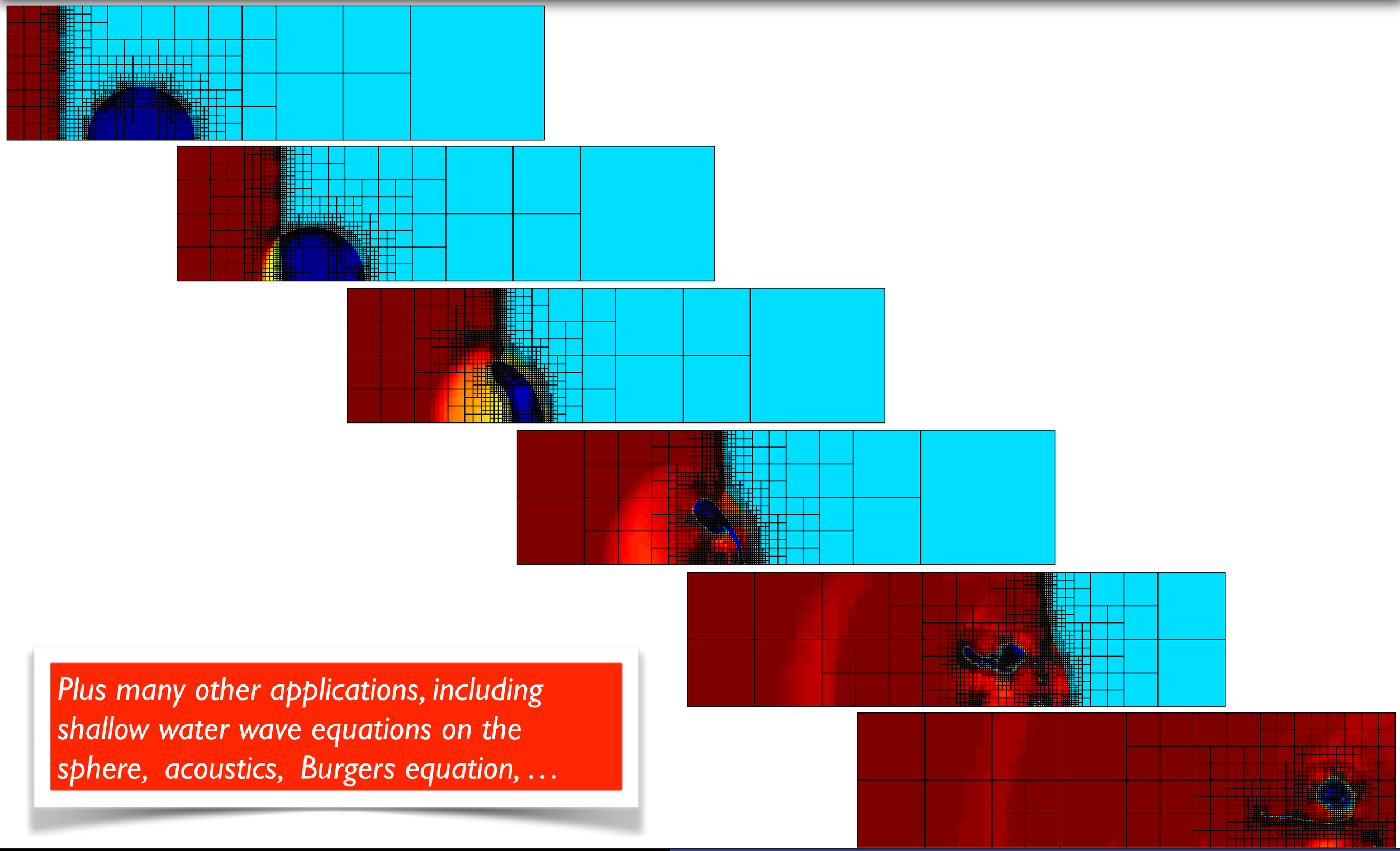
Organizer: Samuel Kokh
Maison de la Simulation, France

10:35-10:55 Bb-Amr and Numerical Entropy Production abstract
Frédéric Golay, Universite de Toulon, France
11:00-11:20 Title Not Available abstract
Pierre Kestener, Maison de la Simulation, France
11:25-11:45 Title Not Available abstract
Donna Calhoun, Boise State University, USA
11:50-12:10 Title Not Available abstract
Thierry Coupez, Ecole Centrale de Nantes, France

"Algorithmic components of the ForestClaw adaptive mesh library"

I'll talk about some details required to choreograph parallel communication, the ghost filling algorithm and a multi-rate time stepping strategy.

Shock-bubble interaction



Plus many other applications, including shallow water wave equations on the sphere, acoustics, Burgers equation, …

Backup Slides

Is AMR really worth it?

Skepticism about AMR

Still, there remains skepticism about how effective adaptive mesh refinement (AMR) can be

- Coarse/fine boundaries with abrupt resolution changes are regarded with suspicion,
- Lack of good refinement criteria dampens enthusiasm for trying out AMR,
- Not obvious how to extend sophisticated numerical algorithms and applications to the adaptive setting,

When AMR *is* used,

- Multi-rate time stepping is not always used,
- The goals are often modest :"Do no harm!"
- Grids are often only static; not dynamically refined.

Why are AMR codes difficult to write and use?

- Heterogeneous data structures associated with storing hierarchy of grids,
- Dynamically creating and destroying grids :
- Need a "factory" paradigm to create new grids and any auxiliary data arrays (material properties, metric terms, bathymetry, etc) that go with each new grid,
- Parallel load balancing
- Coupling of multiple solvers (e.g. advection + diffusion + elliptic solvers + source terms),
- Multi-physics,
- Data visualization and post-processing
- Computing diagnostics on a nested grid hierarchy,
- Error estimation, tuning for efficient use of grids, ...

What makes AMR codes difficult to use?

- Time stepping beyond just single step explicit schemes (IMEX, SSP, RK, ...)
- Understanding how overall time stepping interacts with dynamic grid creation, destruction and management.
- Implicit solvers,
- Coupling of multiple solvers (e.g. advection + diffusion + elliptic solvers + source terms),
- Multi-physics,
- Data visualization and post-processing
- Computing diagnostics on a nested grid hierarchy,
- Error estimation, tuning for efficient use of grids, ...

Scientific Software Development

Thanks!

*The Big Bang Theory

Scientific Software Development …

Backup slides

Existing AMR Codes

What if you have ideas about

- Multi-stage, multi-step, IMEX, SSP, parallel-in-time, exponential integrators, and other time stepping schemes in an adaptive setting,
- Accuracy of multi-rate schemes for PDEs with mixed elliptic/parabolic/hyperbolic terms.
- Elliptic and parabolic solvers (iterative? direct? Explicit? Fast multipole?)
- Parallelism in the AMR setting?
- Error estimation
- Higher order accuracy
- Complex physics

Should you write yet-another-AMR code?

What they might be thinking...

We'll just refine the grid everywhere

My grant doesn't pay for software development

I know somebody who tried it (in 1992) and it didn't work

It year(s) of effort it will take won't pay off

Or...

They've actually tried it...

Block-structured AMR codes

- General purpose (freely available) block-structured codes
  - PARAMESH (NASA/Goddard)
  - SAMRAI (Lawrence Livermore National Lab)
  - BoxLib (Lawrence Berkeley Lab)
  - Chombo (Lawrence Berkeley Lab)
  - AMRClaw (University of Washington/NYU)
- All are large frameworks, with many developers
- Mostly C++ and Fortran libraries (no GUIs) that started life as research codes.

See my website for a list of many more application specific codes

AMR Software

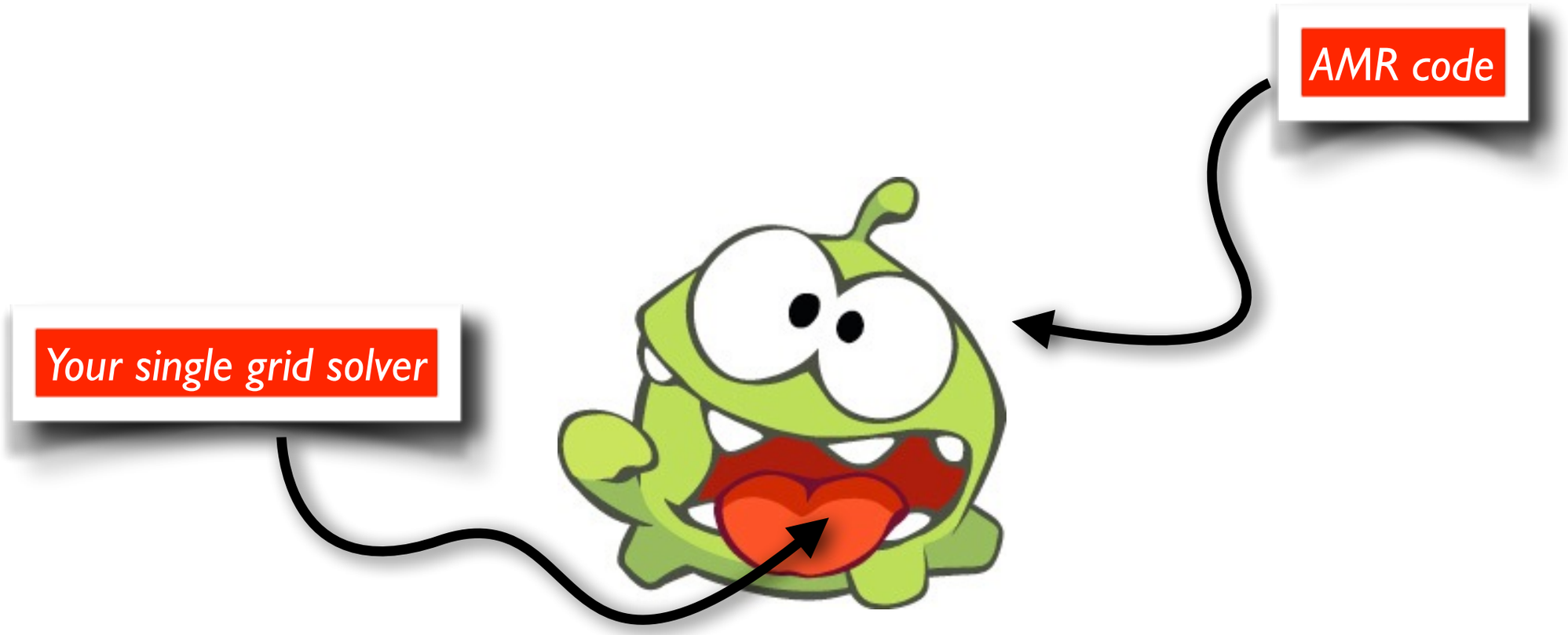
"PARAMESH is a package of Fortran 90 subroutines designed to provide an application developer with an easy route to extend an existing serial code which uses logically Cartesian structured mesh into a parallel code with adaptive mesh refinement"

SAMRAI - "Object oriented C++ library developed to provide algorithmic and software support to large scale multiphysics problems relevant to the US Department of Energy (DOE)"

Boxlib - "These libraries provide the software infrastructure for the computational activities (combustion, astrophysics, porous media flow) at the Center for Computational Sciences and Engineering (CCSE) at Lawrence Berkeley Labs"

Block-structured AMR

Mental model of how this might work :



* *Idea for code name : OmNum*

Block-structured AMR

The Dream

```
AMR.run(max_time, max_steps);
```

Your single grid solver is called from here.

Block-structured AMR

The Reality

```
Tuple< RefCountedPtr<AMRLevelOpFactory<
LevelData<FArrayBox> > >, SpaceDim> velTGAOpFactoryPtrs;

for (int idir = 0; idir < SpaceDim; idir++)
{velTGAOpFactoryPtrs[idir] =
    RefCountedPtr<AMRLevelOpFactory<LevelData
    <FArrayBox> > >
    ((AMRLevelOpFactory<LevelData<FArrayBox> >*)
    (new AMRPoissonOpFactory())); //.....
```

Your single grid solver

you

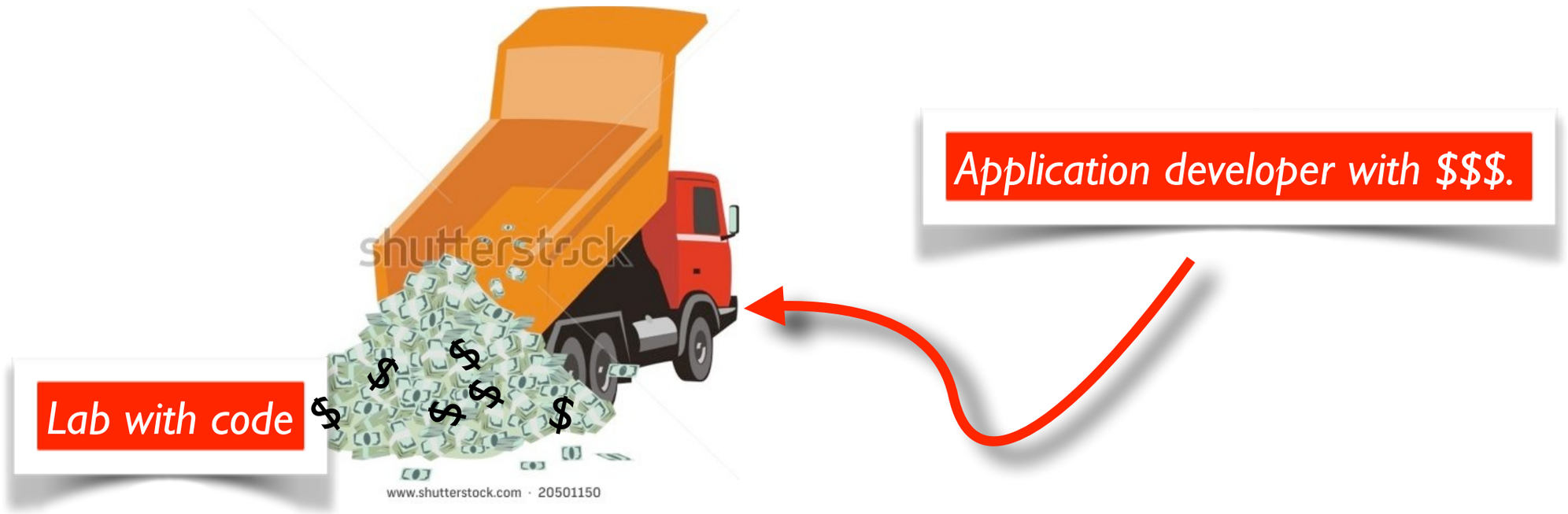
Retro...

```
node(ndjhi,mptrnx) = node(ndjhi,mptr)
node(ndjhi,mptr)   = node(ndjlo,mptr) + nyl - 1
node(ndjlo,mptrnx) = node(ndjhi,mptr) + 1
node(ndihi,mptrnx) = node(ndihi,mptr)
node(ndilo,mptrnx) = node(ndilo,mptr)

rnode(cornxlo,mptrnx)    = cxlo
rnode(cornylo,mptrnx)    = cymid
rnode(cornyhi,mptrnx)    = cyhi
rnode(cornxhi,mptrnx)    = cxhi
node(nestlevel,mptrnx)   = node(nestlevel,mptr)
rnode(timemult,mptrnx)   = rnode(timemult,mptr)
go to 10
```

Block-structured AMR codes

Most of these codes were designed mainly to support large scale applications developers and in-house scientific research.

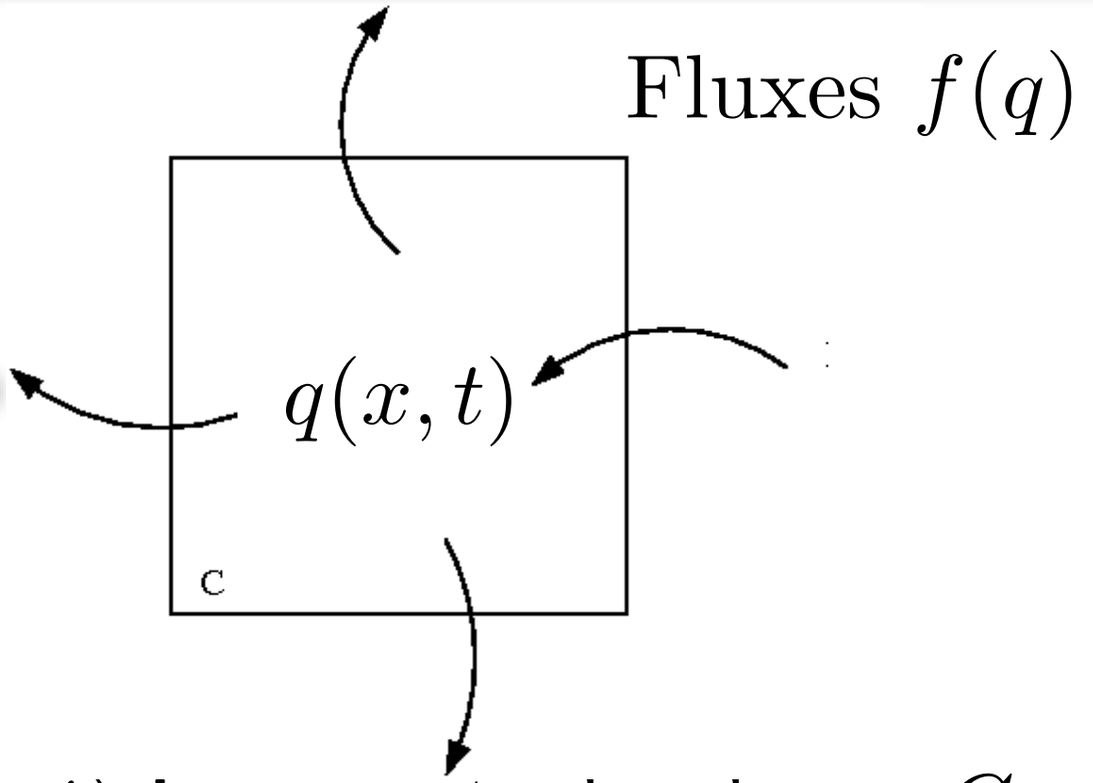


Also initially designed largely for hyperbolic PDEs and single step methods.

Natural Hazards Modeling

Finite volume discretization

Modeling physical processes - Finite volumes



A model of the transport of $q(\mathbf{x}, t)$ in a control volume C

$$\begin{aligned}\frac{d}{dt}\int_C q(x,t)\,dA &= -\int_C \nabla\cdot\mathbf{f}(q(x,t))\,dA \\ &= -\int_{\partial C}\mathbf{f}(q(x,t))\cdot\mathbf{n}\,dL\end{aligned}$$

Physical fluxes

Fluxes depend on underlying physical model and quantities being conserved, i. e.

- mass, tracer species (ash, smoke, …) $f(q) = uq$
- heat $f(q) = -\beta q_x$
- momentum $f(\rho, u, E) = u^2\rho + P(\rho, u, E)$
- energy $f(\rho, u, E) = u(E + P(\rho, u, E))$

By updating quantities using fluxes, we can guarantee numerical conservation of mass, energy, momentum, tracers, etc.

Finite Volume Update

Using numerical fluxes, we use the explicit update formula :

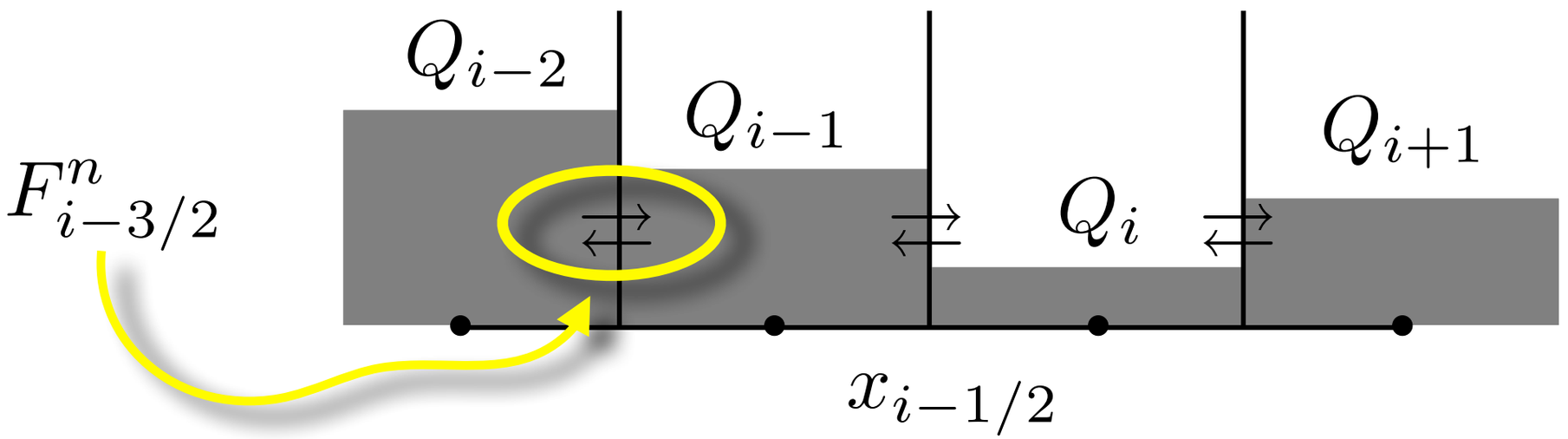
$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left[F_{i+1/2}^n - F_{i-1/2}^n\right]$$

Numerical fluxes

where

$$F_{i-1/2}^n \approx \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f(q(x_{i-1/2},t))\,dt$$

Average value in mesh cell



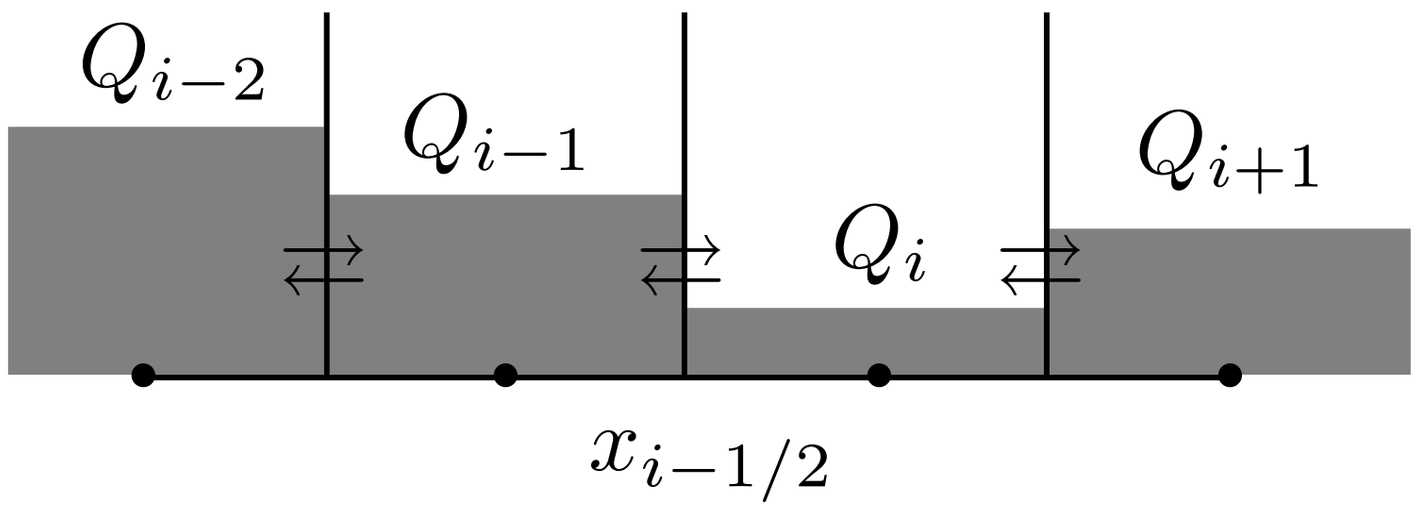
Finite volume schemes

We assume an explicit time stepping scheme

$$F^n_{i-1/2} \approx \int_{t_n}^{t_{n+1}} f(q(x_{i-1/2}, t))\, dt$$

so that we try to find formulas for the flux of the form

$$F^n_{i-1/2} = \mathcal{F}(Q^n_i, Q^n_{i-1})$$



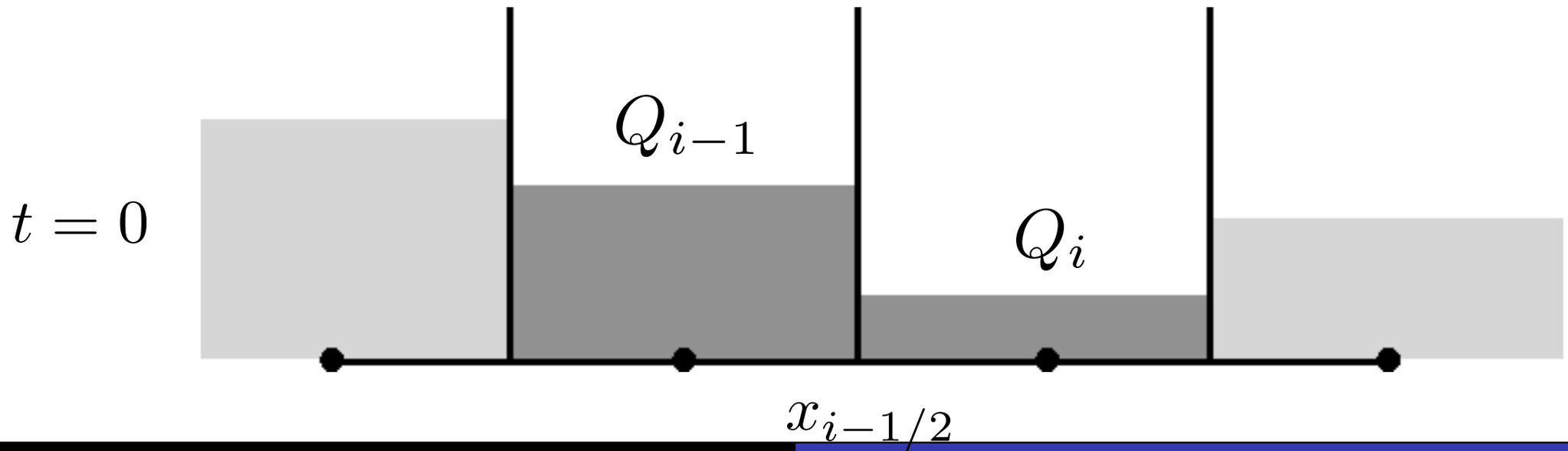
Riemann Problems

At each cell interface, solve the hyperbolic problem with special initial data, i.e.

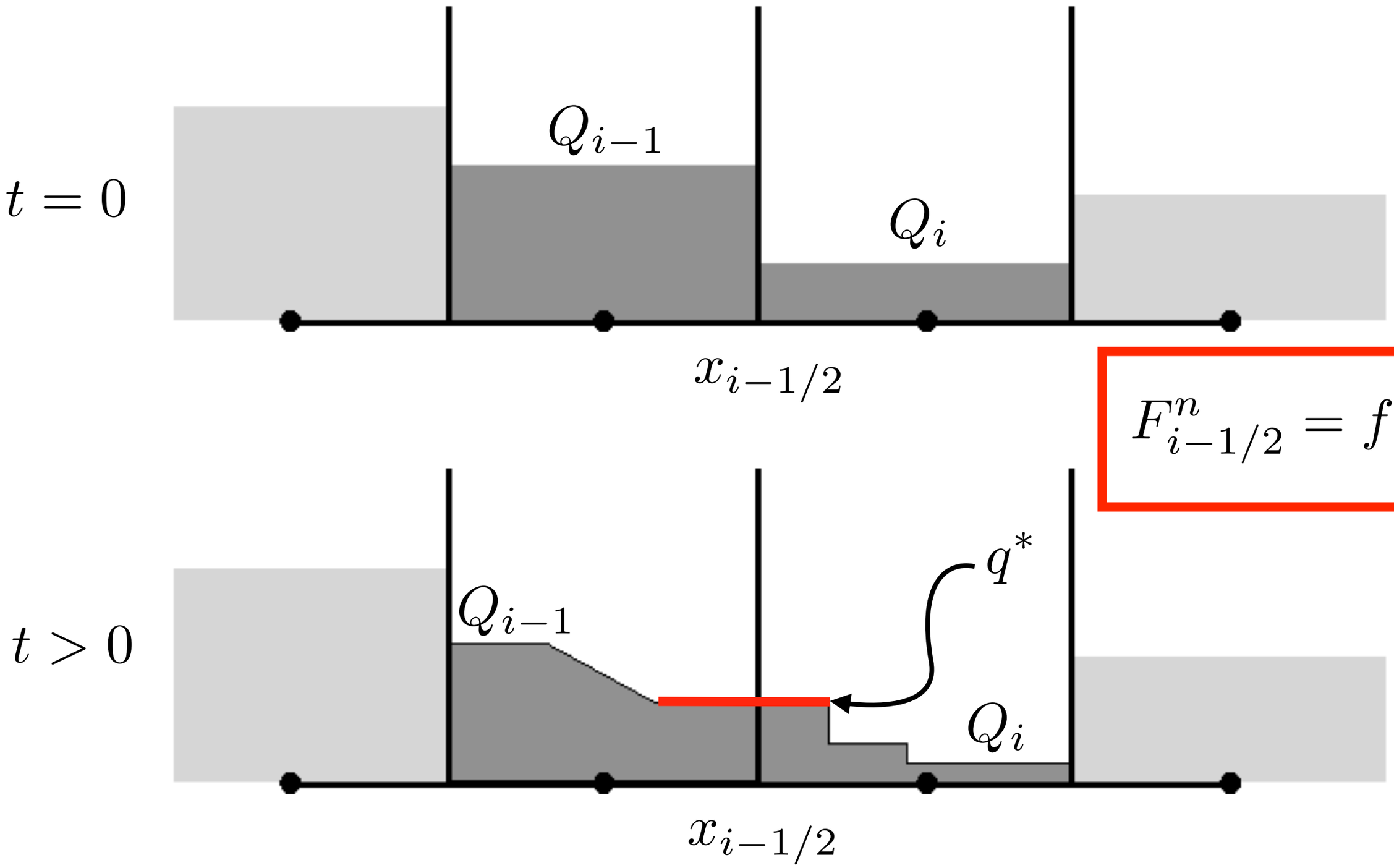
$$q_t + f(q)_x = 0$$

subject to

$$q(x,0) = \begin{cases} Q_{i-1} & x < x_{i-1/2} \\ Q_i & x > x_{x-1/2} \end{cases}$$

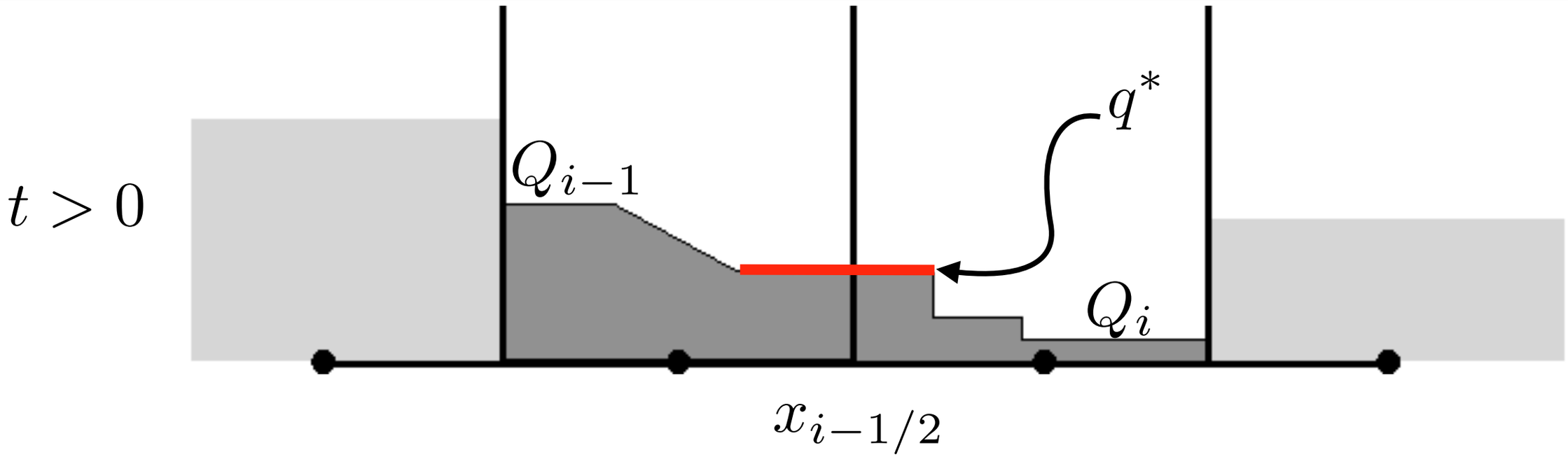


Finite volume update



$$F^n_{i-1/2} = f(q^*)$$

Godunov schemes



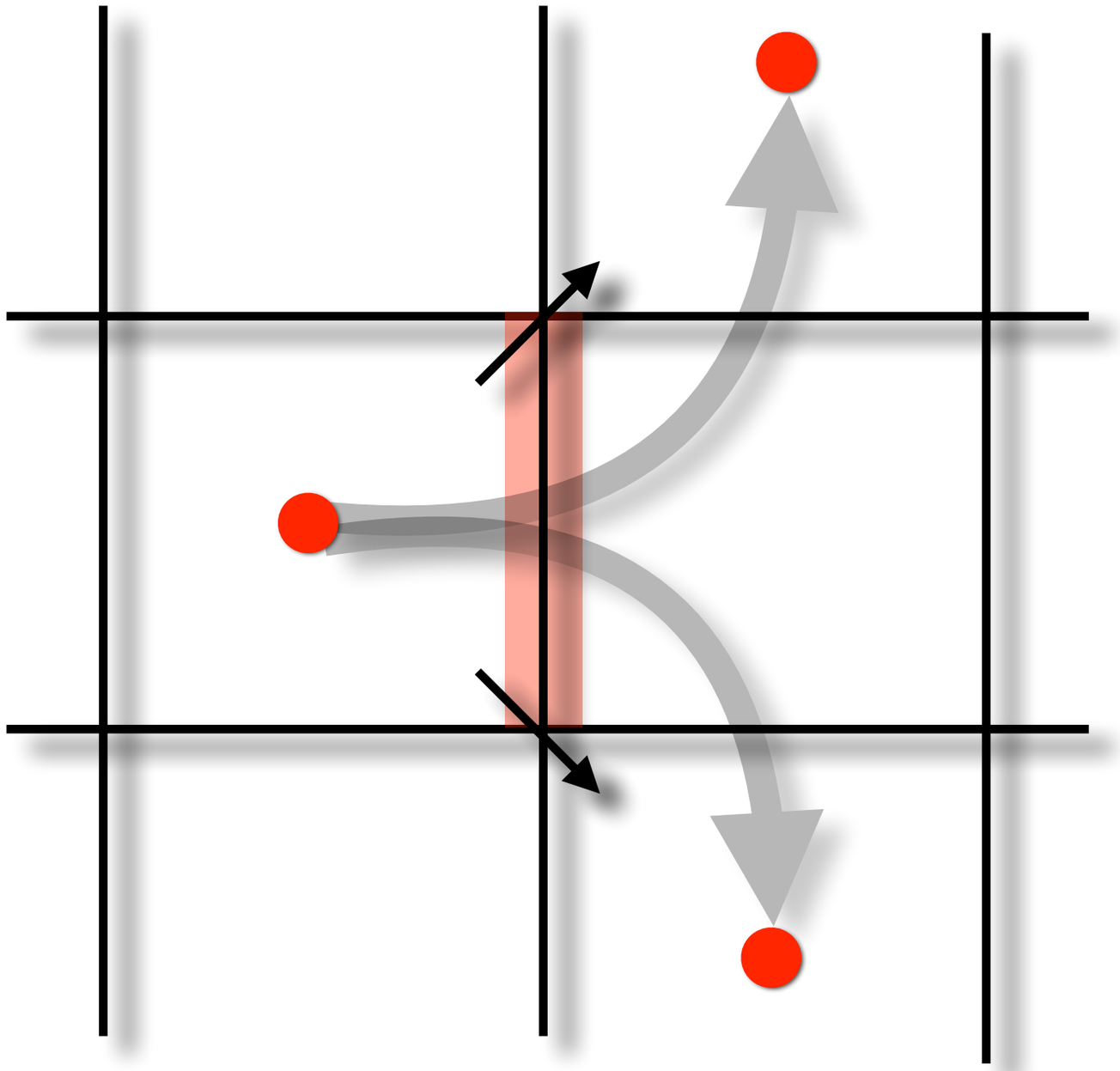
Numerical flux at cell interface is then approximated by

$$F_{i-1/2} = f(q^*)$$

This is the classical Godunov approach for solving hyperbolic conservation laws.

- Resolves shocks and rarefactions and is conservative

Unsplit algorithms in 2 dimensions



In one time step, conserved quantities can be transported across corners, so corner coupling must be taken into account.

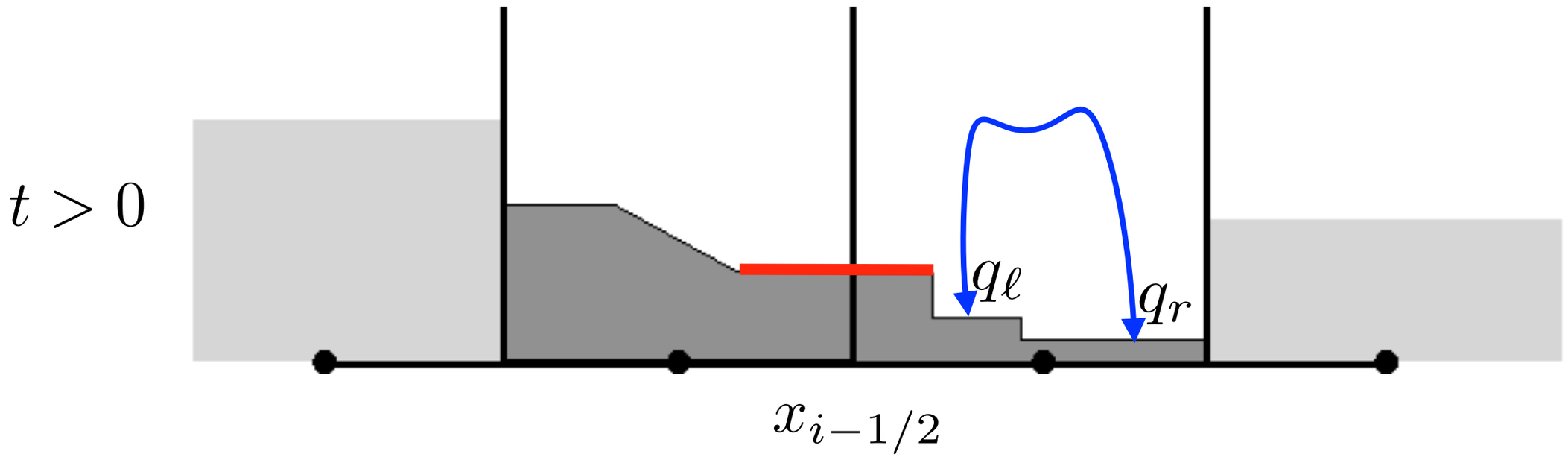
Approximate Riemann solvers

Conditions across shocks

Across shocks, the left and right states must satisfy the Rankine-Hugoniot conditions

$$f(q_r) - f(q_\ell) = s(q_r - q_\ell)$$

This leads to a nonlinear system for intermediate states.



Approximate Riemann solvers

$$q_t + f(q)_x = 0$$

We solve a linearized system at each cell interface, at each time step

$$q_t + f'(\hat{q})q_x = 0 \qquad \longleftrightarrow \qquad q_t + Aq_x = 0$$

for "Roe averaged" values $\widehat{q}$.

We use the eigenvalue/eigenvector decomposition of the flux Jacobian

$$A = R\Lambda R^{-1}$$

$$R = [r^1, r^2, \ldots r^m] \qquad \Lambda = \mathrm{diag}(\lambda^1, \lambda^2, \ldots \lambda^m)$$

Eigenvalue decomposition

$$q_t + A\,q_x = 0 \quad \rightarrow \quad q_t + R\Lambda R^{-1}\,q_x = 0$$

Define characteristic variables $\omega \in R^m$ as

$$\omega(x,t) = R^{-1}\,q(x,t), \qquad \omega(x,0) = R^{-1}\,q(x,0)$$

Characteristic equations decouple into m scalar equations :

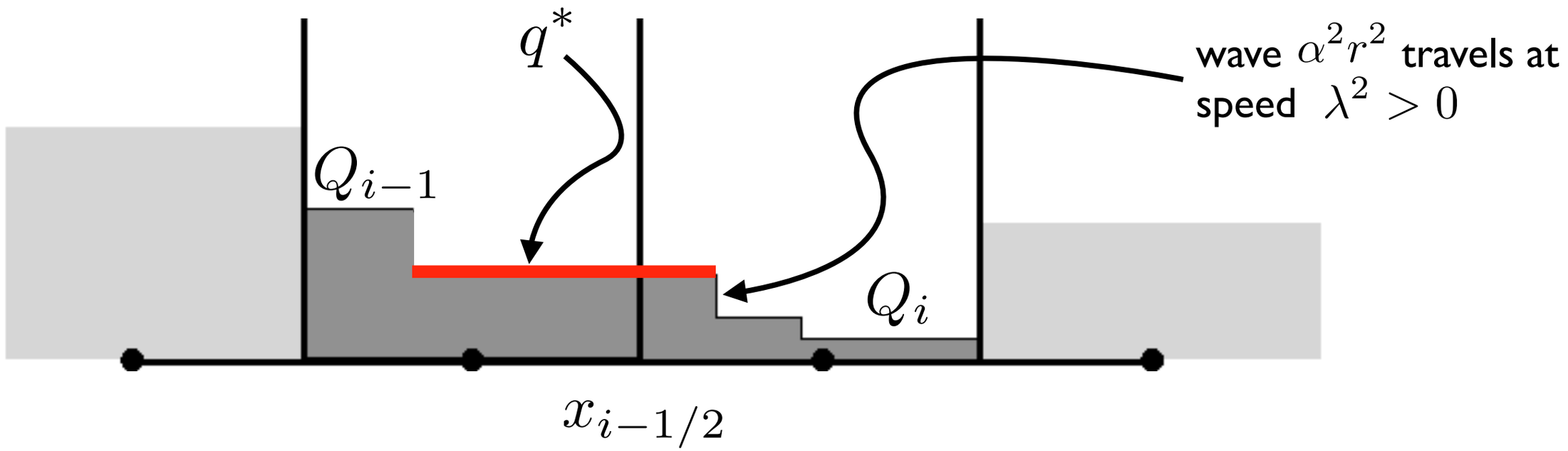
$$\omega^p_t + \lambda^p \omega^p_x = 0, \qquad p = 1, 2, \ldots, m$$

Solution for q(x,t) can be obtained from the solution to characteristic equations are given by

$$\omega^p(x,t) = \omega^p(x - \lambda^p t, 0)$$

Solution to linearized problem

For piecewise constant initial data, evolution looks like :



Decompose jump in Q at the interface into waves :

$$q^* = Q_{i-1} + \alpha^1 r^1 = Q_i - \alpha^3 r^3 - \alpha^2 r^2$$

where $\alpha = R^{-1}(Q_i - Q_{i-1})$

Roe solvers

- Scheme is conservative if linearization is chosen correctly
- Much more efficient then solving the Riemann problem exactly (assuming analytic eigenvalue information is known).
- Gives accurate results, although may require a fix for transonic rarefaction waves
- Use "wave limiters" to suppress spurious oscillations

Wave propagation algorithms implemented in Clawpack (R. J. LeVeque) using "fluctuations". This approach handles both conservative and nonconservative terms in a uniform fashion.

www.clawpack.org

Multirate time stepping

Mappings and multiblock transforms

Mapped, multiblock domains

Assume each quadrilateral mesh cell can be approximated by a ruled surface.

$$\mathbf{S}(\xi,\eta) = \mathbf{a}_{00} + \mathbf{a}_{01}\xi + \mathbf{a}_{10}\eta + \mathbf{a}_{11}\xi\eta, \qquad 0 \le \xi,\eta \le 1$$

where $\mathbf{a}_{ij} \in R^{2\times 1}$ are computed from mesh cell vertices.

- Covariant basis vectors : $\tau_{(1)} \approx \mathbf{S}_\xi$ and $\tau_{(2)} \approx \mathbf{S}_\eta$
- Mesh cell areas : $\| \mathbf{S}_\xi \times \mathbf{S}_\eta \|$
- Edge normals and tangents
- Mappings are supplied as functions

```
[xp,yp,zp] = mapc2m(xc,yc)
```

Mapping contexts

```
fclaw2d_map_context_t* fclaw2d_map_new_squareddisk(const double alpha)
{
    fclaw2d_map_context_t *cont;
    cont = FCLAW_ALLOC_ZERO (fclaw2d_map_context_t, 1);
    cont->query = fclaw2d_map_query_squareddisk;
    cont->mapc2m = fclaw2d_map_c2m_squareddisk;

    cont->user_double[0] = alpha;
}
```

No need to rely on a fixed argument list; arguments specific to the mapping can be passed into the mapping routine.

```
c       call mapc2m(xc,yc ,xd1,yd1,zd1)
        call fclaw2d_map_c2m(map_context,blockno,xc,yc,xd1,yd1,zd1)
```

```
static void
fclaw2d_map_c2m_squareddisk(fclaw2d_map_context_t *cont, int blockno,
                            double xc, double yc,
                            double *xp, double *yp, double *zp)
{
    double alpha = cont->user_double[0];
    mapc2m_squareddisk(&blockno,&xc,&yc,xp,yp,zp,&alpha);
}
```

Index transformations

```
c       # Exchange at left neighbor
        if (idir .eq. 0) then
           do j = 1,my
              do ibc = 1,mbc
                 do mq = 1,meqn
                  if (iface .eq. 0) then
c                       # Left face
                        i1 = 1-ibc
                        j1 = j
                     elseif (iface .eq. 1) then
c                       # Right face
                        i1 = mx+ibc
                        j1 = j
                     endif
                     call fclaw2d_transform_face(i1,j1,i2,j2,
     &                    transform_ptr)
                     qthis(mq,i1,j1) = qneighbor(mq,i2,j2)
                 enddo
              enddo
           enddo
```

Loop over all ghost cells at left edge

This is the code used for every exchange at a left edge even if it is not a block boundary.

transform_face maps (i1,j1) ghost cell values to neighbor values, using information about block boundary orientations encoded in ***transform_ptr****.*

Backup Slides

Misc