Adaptive mesh refinement for solving PDEs on logically Cartesian mapped, multiblock quadtree domains

Donna Calhoun (Boise State University)

Collaborators : C. Burstedde (Univ. of Bonn)

and

R. J. LeVeque (Univ. of Washington), D. George (USGS), K. Mandli (Columbia University); M. Berger (NYU); C. Helzel (Univ. Dusseldorf); Melody Shih (Columbia University); Talin Mirzakhanian (Boise State University); D. Ketcheson (KAUST University)

Applied Math Seminar

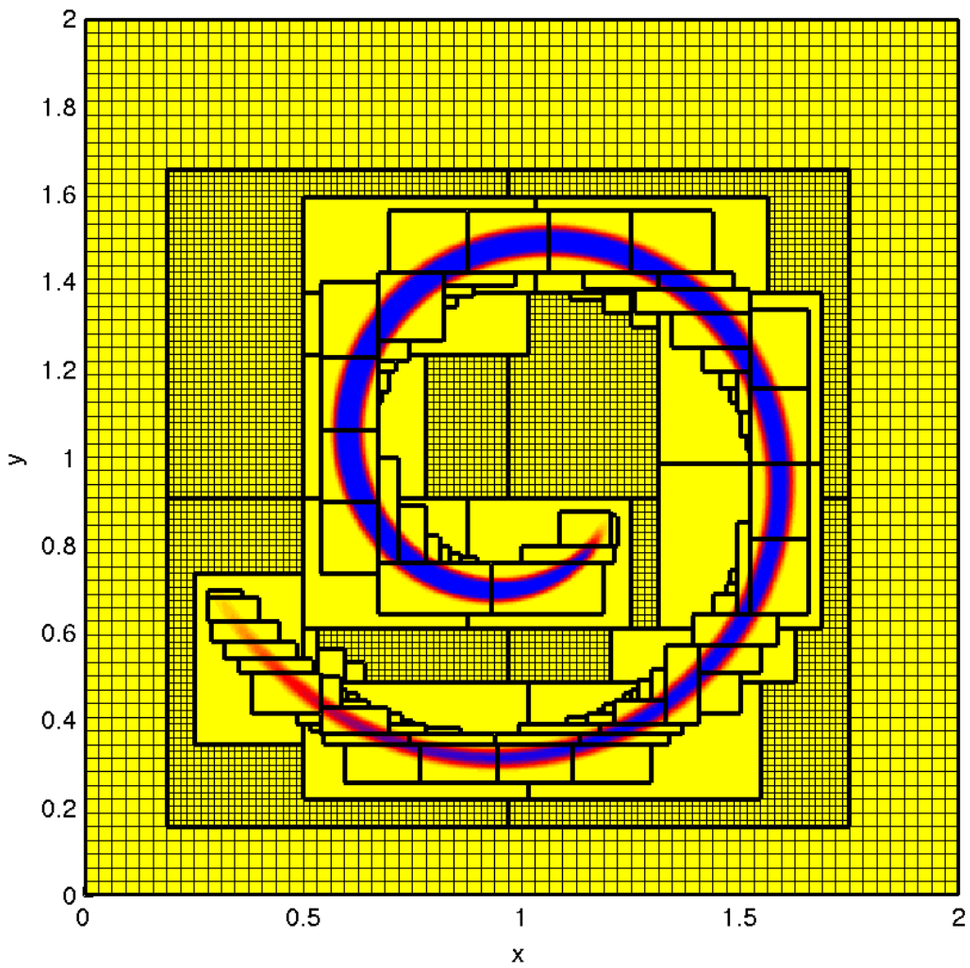
September 22,2016

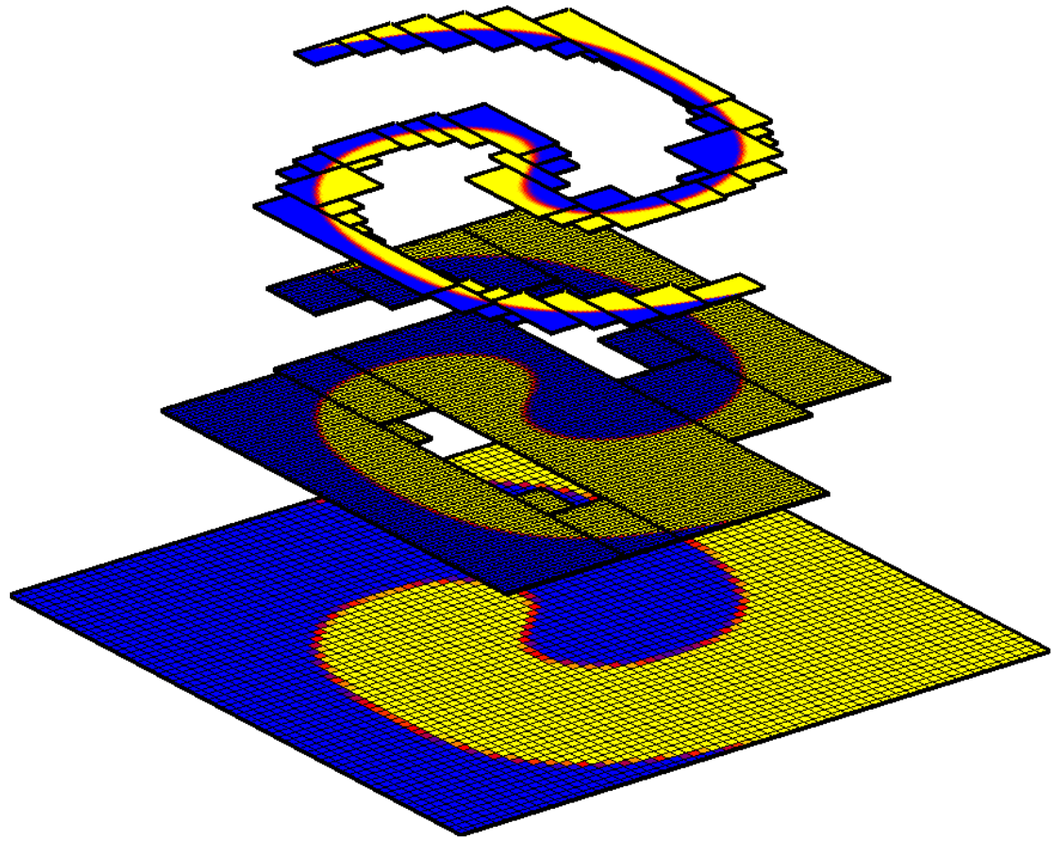
Colorado State University

Adaptive Mesh Refinement (AMR)

Overlapping patch-based AMR (Structured AMR or SAMR)

Original approach (Berger, 1984)





Codes : Chombo (LBL), AMRClaw and GeoClaw (UW, NYU) , **Boxlib* (LBL)**, SAMRAI (LLNL), AMROC (Univ. of South Hampton) and many others

Brief history of AMR*

Hyperbolic conservation laws with explicit time, local time stepping

- 1985 : M. Berger and J. Oliger publish "Adaptive mesh refinement for **hyperbolic** partial differential equations" (from M. Berger's thesis at Stanford)
- 1989 : M. Berger and P. Colella publish "Local adaptive mesh refinement for **shock hydrodynamics**" (NYU + Berkeley effort)
- 1987 : W. Skamarock, "Adaptive Grid Refinement for Numerical Weather Prediction" (another Stanford thesis; uses M. Berger's code)
- 1991 : M. Berger and I. Rigoustos publish "An algorithm for point clustering and grid generation" - NYU
- 1991 : A. Almgren - "A fast adaptive vortex method using local corrections" (thesis; Berkeley)
- 1998 : M. Berger and R. J. LeVeque publish "Adaptive mesh refinement using wave-propagation algorithms for **hyperbolic** systems" - (AMRClaw; University of WA + NYU)

1985 ⟵ ⟶ 1998

* and incomplete and lab biased

Brief history of AMR

Elliptic solvers, Navier-Stokes and incompressible Euler equations

- 1996 : D. Martin and K. Cartwright write technical report "Solving Poisson's Equation using Adaptive Mesh Refinement (LBL)
- 1997 : L. Howell and J. Bell, "An Adaptive Mesh Projection Method for Viscous Incompressible Flow" (LBL)
- 1998 : A. Almgren, J. Bell, P. Colella, et al "A Conservative Adaptive Projection Method for the Variable Density Incompressible Navier-Stokes Equations" (LBL)
- 2000 : D. Martin and P. Colella, "A Cell-Centered Adaptive Projection Method for the Incompressible Euler Equations" (LBL)
- 2000 : M. Day and J. Bell, "Numerical Simulation of Laminar Reacting Flows with Complex Chemistry" (LBL)
- 2000 : J. Huang and L. Greengard, "A Fast Direct Solver for Elliptic Partial Differential Equations on Adaptively Refined Meshes" (NYU)
- 2008 : D. Martin, P. Colella, D. Graves, "A cell-centered adaptive projection method for the incompressible Navier-Stokes equations in three dimensions"

1996 ⟷ 2008

Brief history of AMR

Parallel scaling and performance

- 1999 : C. Rendelman, V. Beckner, et al, "Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms" (p-Boxlib)
- 2001 : A. M. Wessink, R. D. Hornung, et al, "Large scale parallel structured AMR calculations using the SAMRAI framework" (LLNL)
- 2006 : M. Welcome, C. Rendleman, et al, "Performance Characteristics of an Adaptive Mesh Refinement Calculation on Scalar and Vector Platforms" (LBL)
- 2007 : P. Colella, J. Bell, N. Keen et al, "Performance and Scaling of Locally-Structured Grid Methods for Partial Differential Equations" (LBL)
- 2007 : T. Wen, J. Su, P. Colella, et al, "An adaptive mesh refinement benchmark for modern parallel programming languages" (LBL)
- 2010 : J. Luitjens and M. Berzins, "Improving the performance of Uintah: A large-scale adaptive meshing computational framework" (Univ. of Utah)

1999 ⟷ present

Brief history of AMR

Higher order and mapped grids

- 2005 : D. Calhoun, R. LeVeque "An accuracy study of mesh refinement on mapped grids" (Univ. of Washington)
- 2011 : P. McCorquodale and P. Colella, "A high-order finite volume method for conservation laws on locally refined grids" (LBL)
- 2011 : C. Shen and JM Qiu and A. Christlieb, et al, "Adaptive mesh refinement based on high order finite difference WENO scheme for multi-scale simulations" (Michigan State Univ.)
- 2011 : S. Guzik, P. McCorquodale, P. Colella, "A Freestream-Preserving High-Order Finite-Volume Method for Mapped Grids with Adaptive-Mesh Refinement" (LBL)
- 2012 : Q. Zhang and H. Johansen and P. Colella, "A fourth-order accurate finite-volume method with structured adaptive mesh refinement for solving the advection-diffusion equation" (LBL)
- 2016 : X. Gao, L. Owen and S. Guzik, "A parallel adaptive numerical method with generalized curvilinear coordinate transformation for compressible Navier--Stokes equations". (CSU)

2000 ⟷ present

Block-structured AMR codes

- General purpose (freely available) block-structured codes
  - AMROC (Univ. of South Hampton)
  - SAMRAI (Lawrence Livermore National Lab)
  - BoxLib (Lawrence Berkeley Lab)
  - Chombo (Lawrence Berkeley Lab)
  - AMRClaw (University of Washington/NYU)
  - Uintah (University of Utah)
  - plus many others designed for specific application areas

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

Brief history of AMR

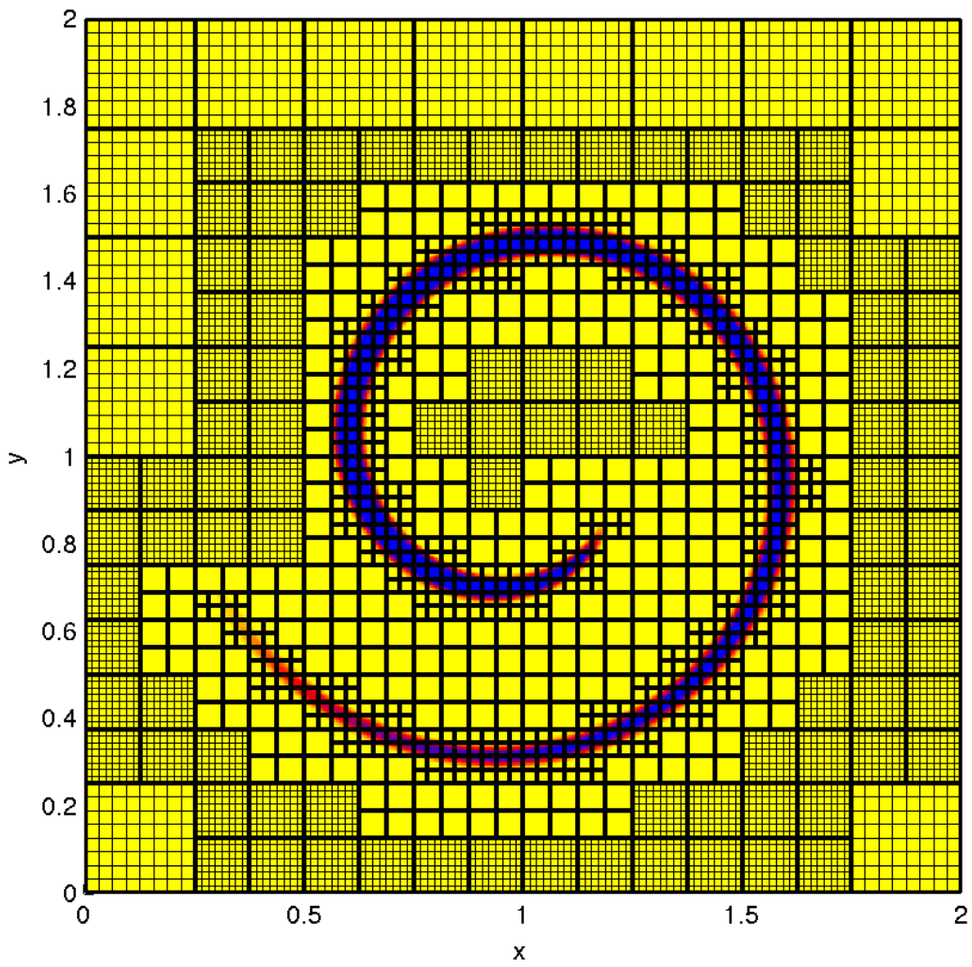
Excellent survey of widely used codes currently available (Chombo, Cactus, Boxlib, Uintah, FLASH)

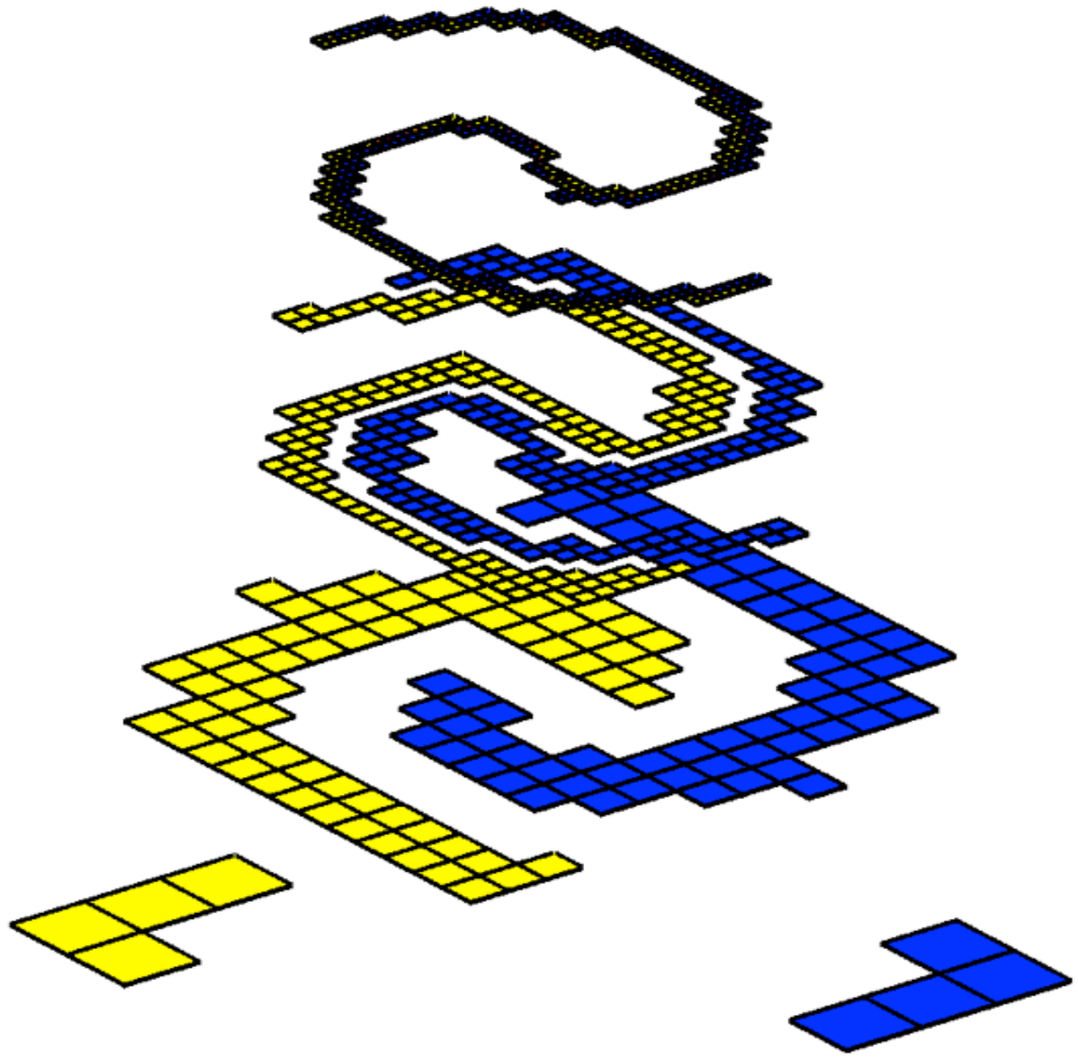
A. Dubey, A. Almgren, J. B. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Laeffler, B. O'Shea, E. Schnetter, B.V. Straalen, and K. Weide, "A survey of high level frameworks in block-structured adaptive mesh refinement packages" , *Journal of Parallel and Distributed Computing*, (2014).

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)

Brief history of AMR

Refinement based on quadtree and octree grid layouts

- 2000 : P. MacNiece, K. Olson et al, "**PARAMESH**: A parallel adaptive mesh refinement community toolkit" (FLASH code based on PARAMESH)
- 2002 : R. Tessyier, "Cosmology Hydrodynamics with adaptive mesh refinement. A new high resolution code called **RAMSES**" (Lausanne, Switzerland)
- 2003 : S. Popinet, "**Gerris**: A tree-based adaptive solver for the incompressible Euler equations in complex geometries" (Paris IV, France)
- 2004 : U. Ziegler, "An ADI-based adaptive mesh Poisson solver for the MHD code **NIRVANA**" (Potsdam, Germany)
- 2005 : J. Dreher and R. Grauer, "**Racoon**: A parallel mesh-adaptive framework for hyperbolic conservation laws" (Bochum, Germany)
- 2008 : C. Burstedde, O. Ghattas, G. Stadler, et al "Towards Adaptive Mesh PDE Simulations on Petascale Computers" (Univ. of Texas)
- 2011 : C. Burstedde, L. Wilcox, O. Ghattas, "**p4est**: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees" (Univ. Texas)
- 2011 : K. Komatsu, T. Soga et al "Parallel processing of the **Building-Cube Method** on a GPU platform" (Tohoku, Japan)

2000 ⟷ present

Brief survey of AMR codes

Codes now being built are using quadtree/octree layouts :

- Quadtree and octree layouts for simplicity and ease of use, and more straightforward path to parallelization.
- Space filling curves for load balancing
- Often with mapped, multiblock capabilities.

But with the possible exception of PARAMESH (which is supported only as part of the FLASH astrophysics code) there are no general purpose easily available quadtree/octree codes based on the Berger-Oliger-Colella patch based algorithms.

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

Goal of patch-based AMR codes

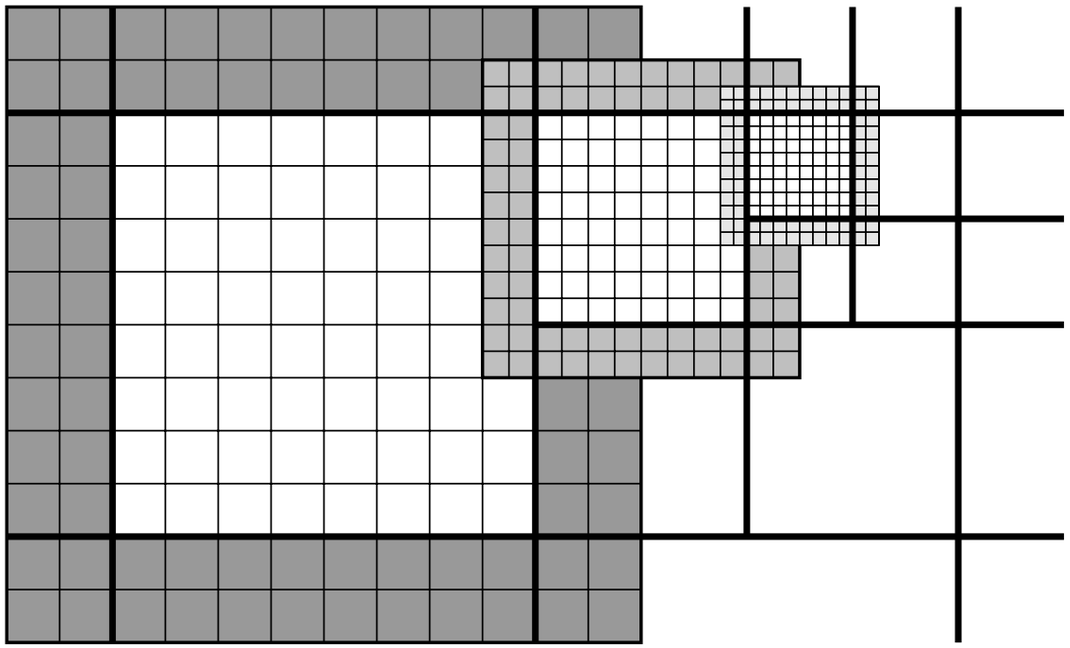
- Make full use of existing solvers for Cartesian grids
- Operate locally on patches whenever possible
- Have the same order of accuracy as the single grid algorithm.
- Maintain conservation where appropriate
- Optional use of local time stepping to maintain a constant CFL number across refinement levels,
- Fully couple solution between grids,
- Operate efficiently on latest hardware.

Goal is to do this without significant overhead associated with managing the grid hierarchy.

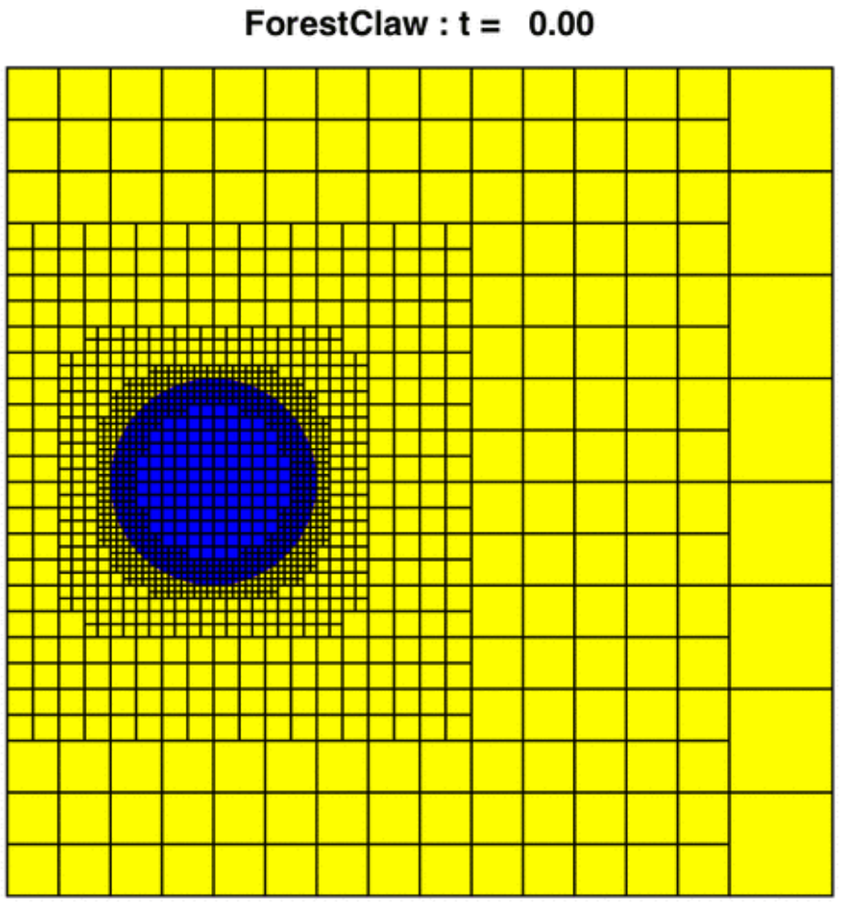
AMR for the computational mathematician

- Support for grid management that is separate from the numerics, that is intuitive, with easily manageable data structures,
- Support for multi-rate time stepping with flexibility to include new time stepping schemes (MOL solvers, for example),
- Easy to add diagnostics for convergence studies,
- Natural code for iterating over arrays (*in Fortran?*),
- Flat data structures - little reliance on templates, and exotic object oriented data structures,
- Parallelism should happen automatically.
- Simple build system

ForestClaw



Each quadrant is a grid, where single grid Cartesian grid solvers are used



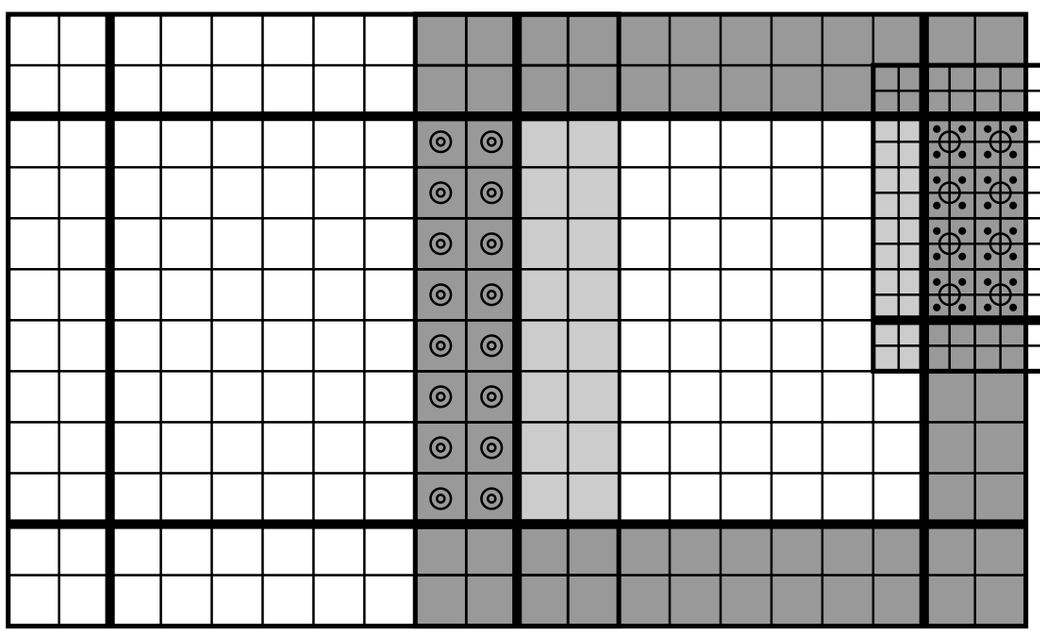
Tracer advected in a swirling flow field

Algorithmic components in AMR

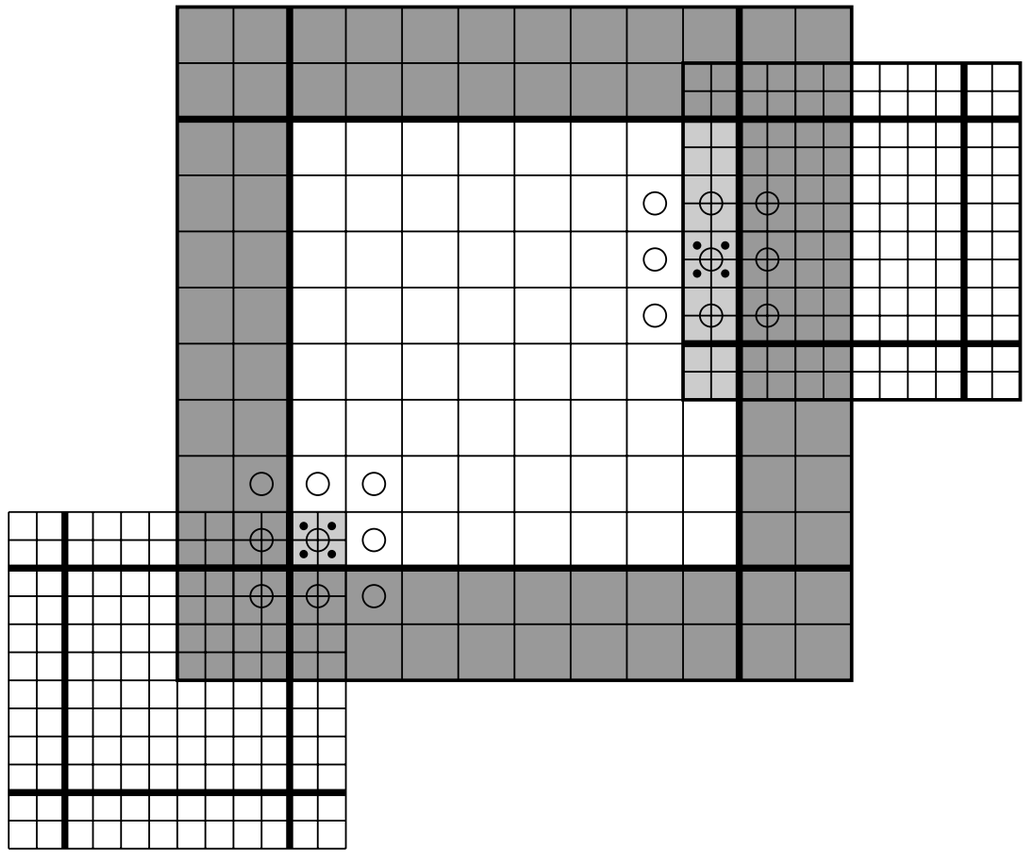
- Filling ghost cells and parallel communication
- Refinement criteria and tagging cells
- Adaptive mesh generation
- Multi-rate time stepping
- Grid mappings
- Index transformations at multi-block seams
- Conservation at coarse/fine boundaries
- Accuracy at coarse/fine boundaries
- Solvers (e.g. Clawpack)

Filling ghost cells

Assume valid data in the interior of each patch



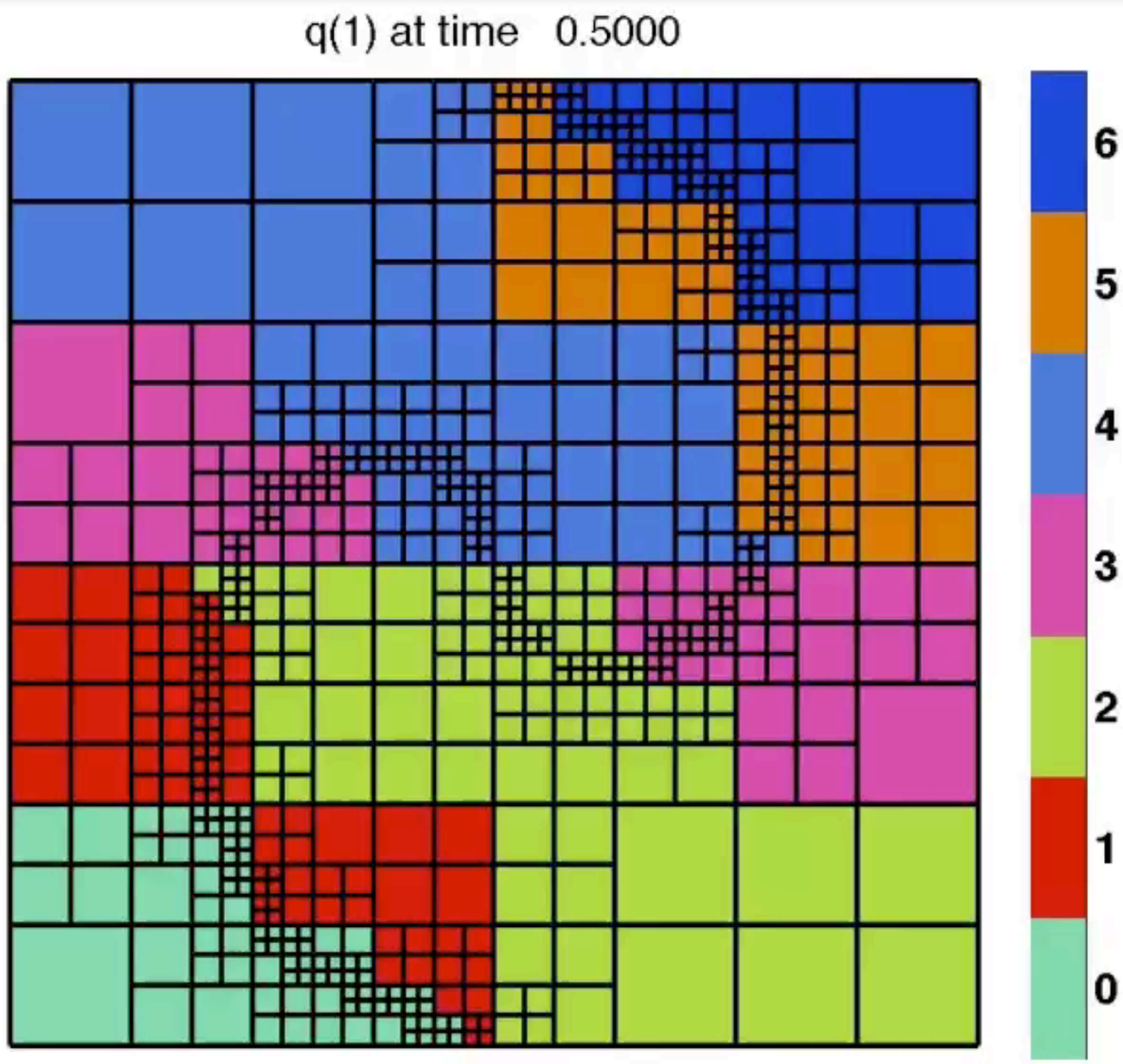
Step 1 : Averaging or copying to coarse ghost regions



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

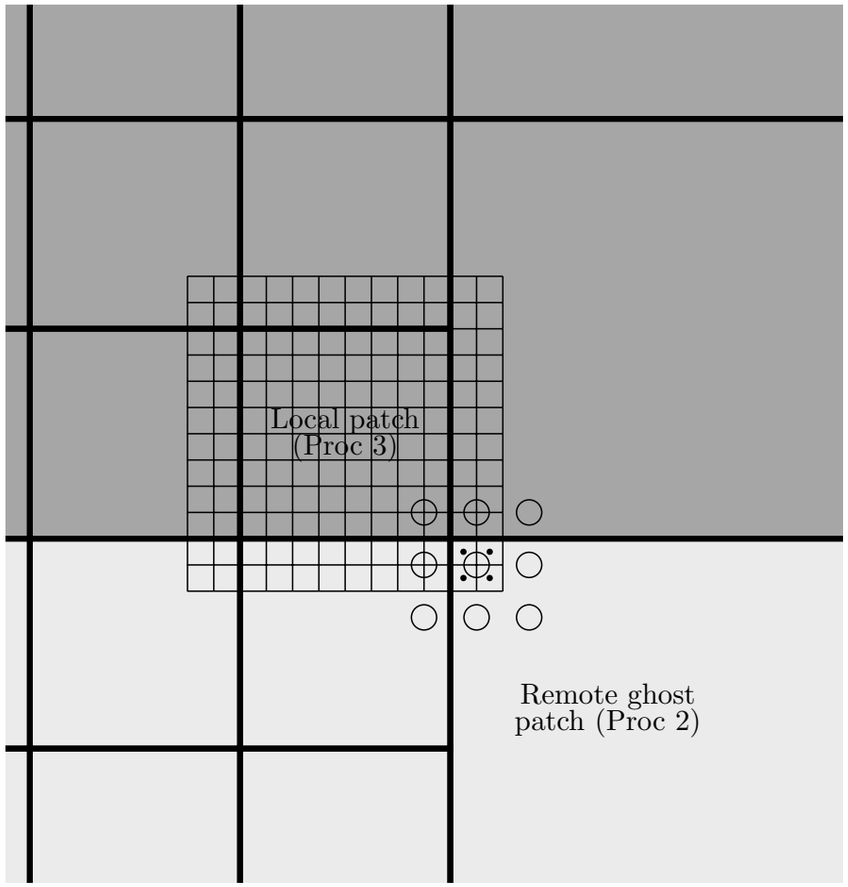
Each grid (or "leaf", in p4est terminology) has one of more layers of ghost cells used for communication between grids

Parallel ghost filling algorithm

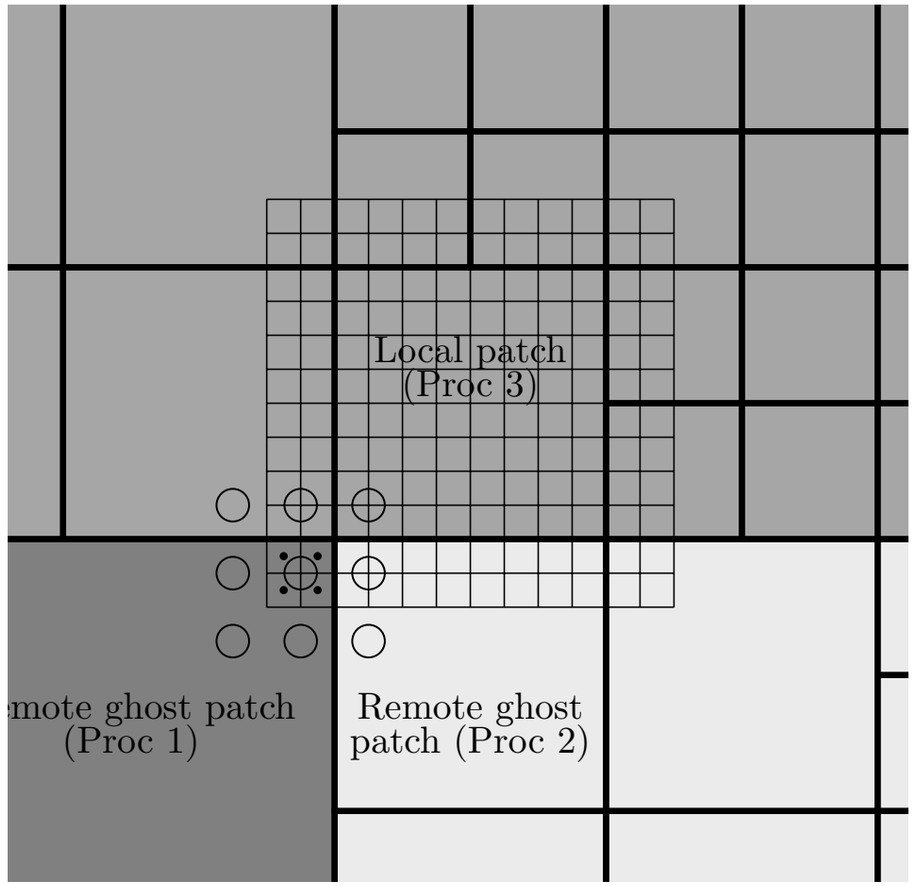


- A "grid" (i.e. leaf in the quadtree) is our minimum parallel unit
- Morton ordering space filling curve used to load balance and preserve data locality

Parallel ghost filling algorithm



Remote patches on processor boundary must exchange ghost cells before being sent to local processor



An lightweight indirect exchange is required

- *Remote patches must have valid coarse grid ghost data so that corners on local patches can be filled in.*
- *Requires one communication pass per ghost cell update*

Algorithmic components

Parallel Performance and cost of AMR

Scalar advection

Focus of this talk will be on a simple scalar advection problem.

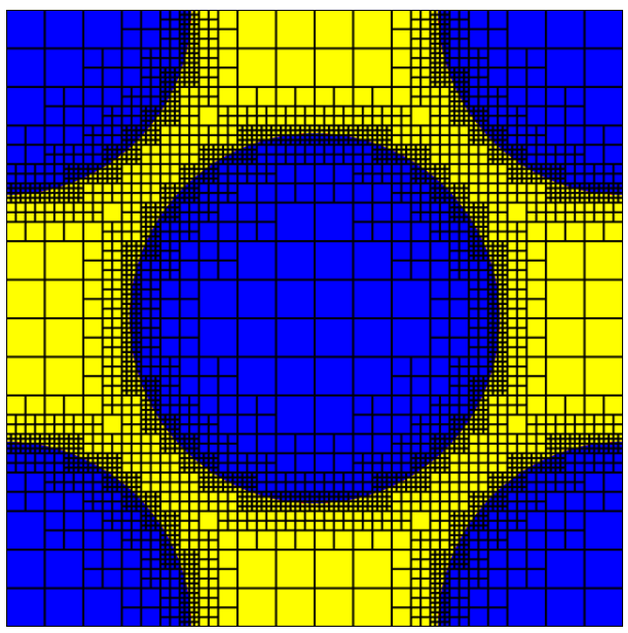
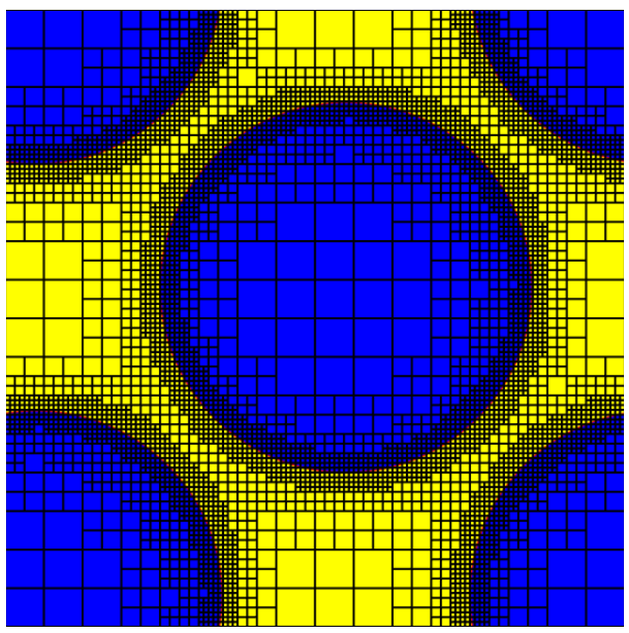
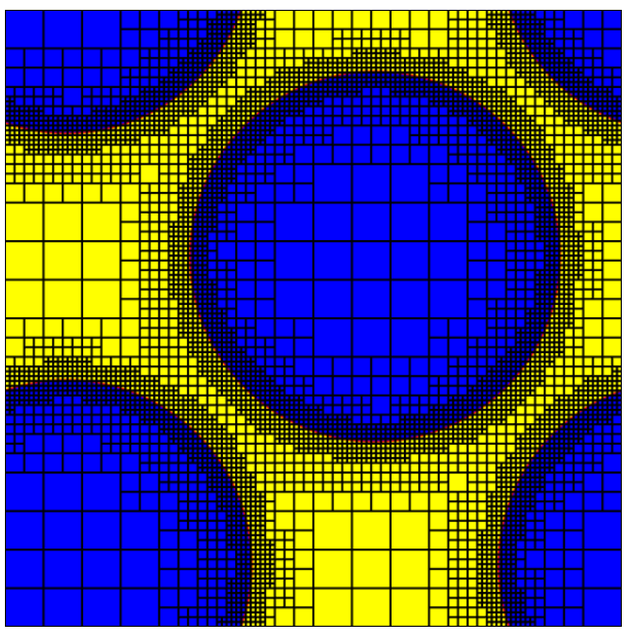
$$q_t + \mathbf{u} \cdot \nabla q = 0$$

and a discontinuous tracer field q(x,t).

- Constant flow field
- Riemann solver is cheap, so communication costs cannot be hidden.
- Easy to choose time step that scales across several resolutions and simulations times
- Discontinuous tracer fields are convenient because refinement is triggered at all levels.

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

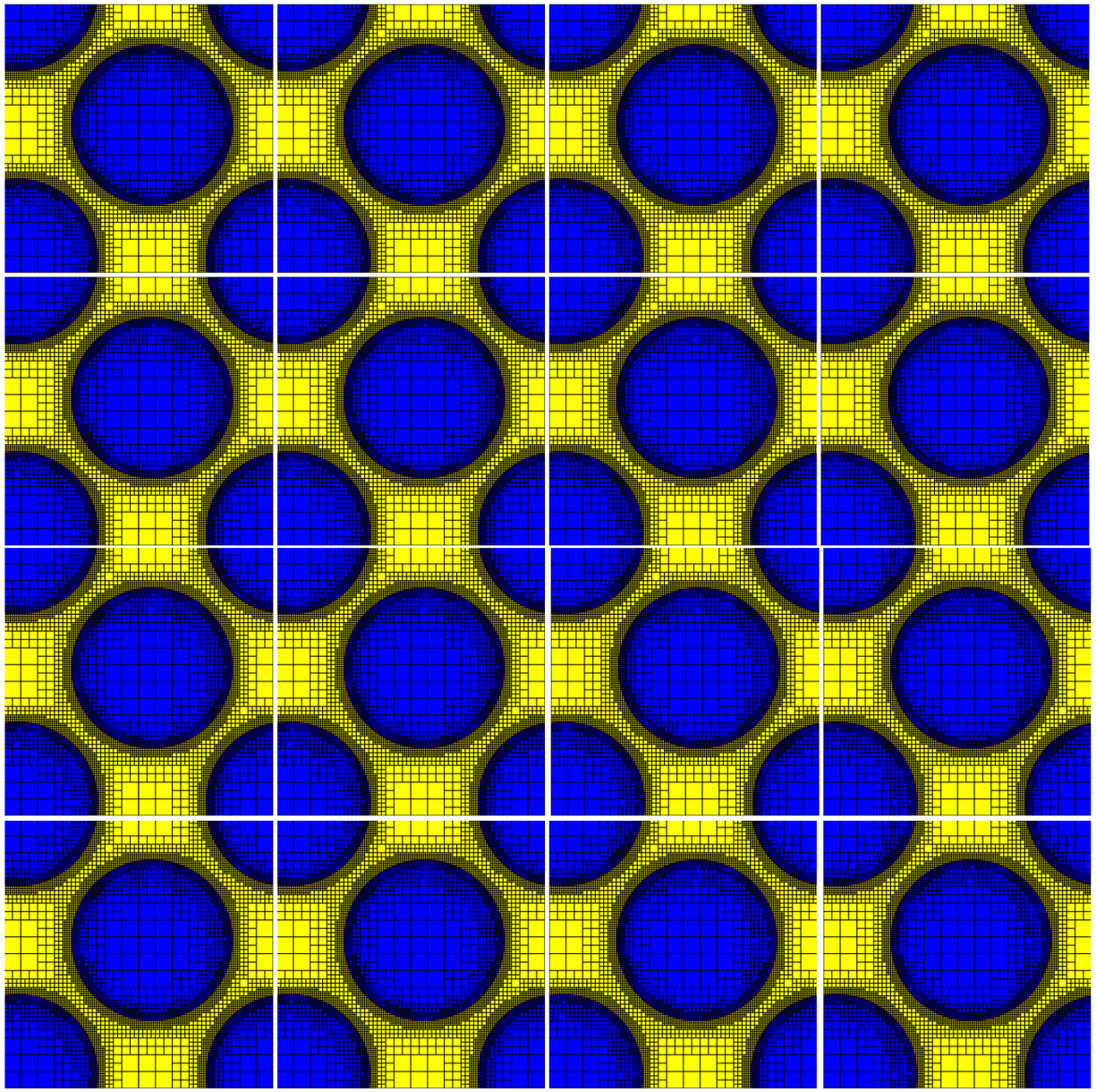
JUQUEEN - BlueGene/Q

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 (or 1024 MPI processes)

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.

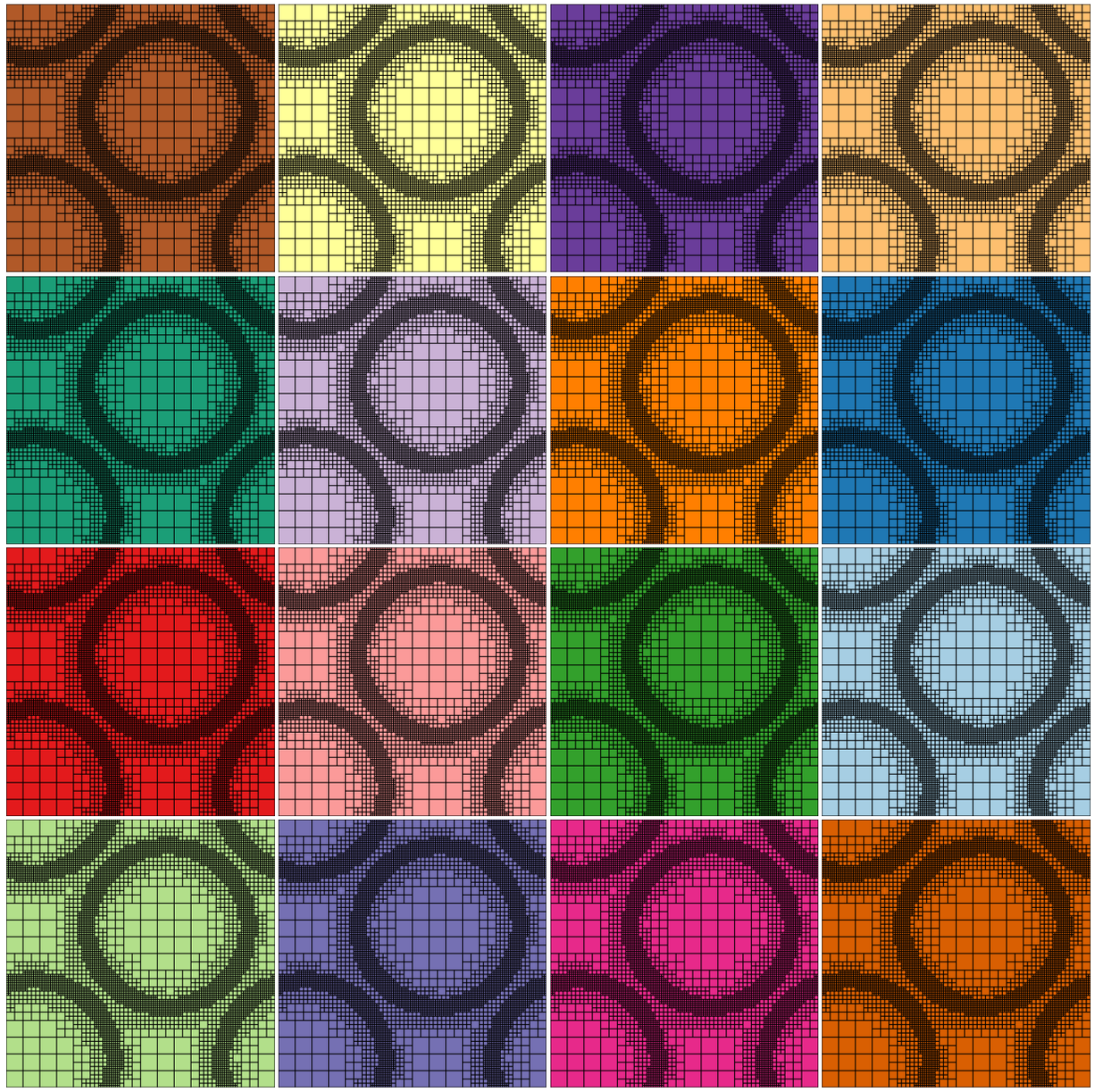
Weak scaling - replicated problem



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.

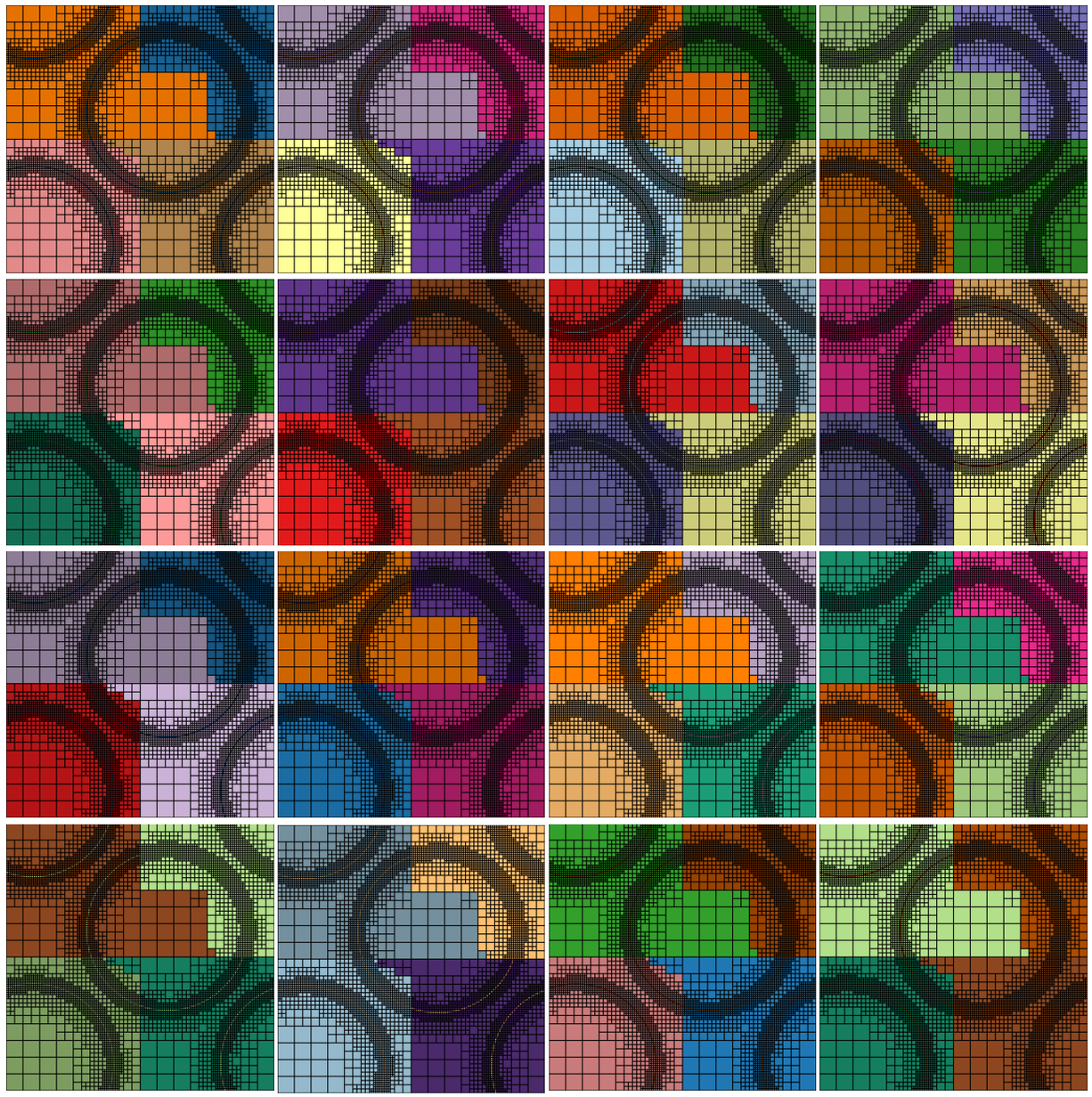
Weak scaling - replicated problem



16 processor run (one process per brick)

Equal numbers of grids on each processor

Strong scaling - replicated problem



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**

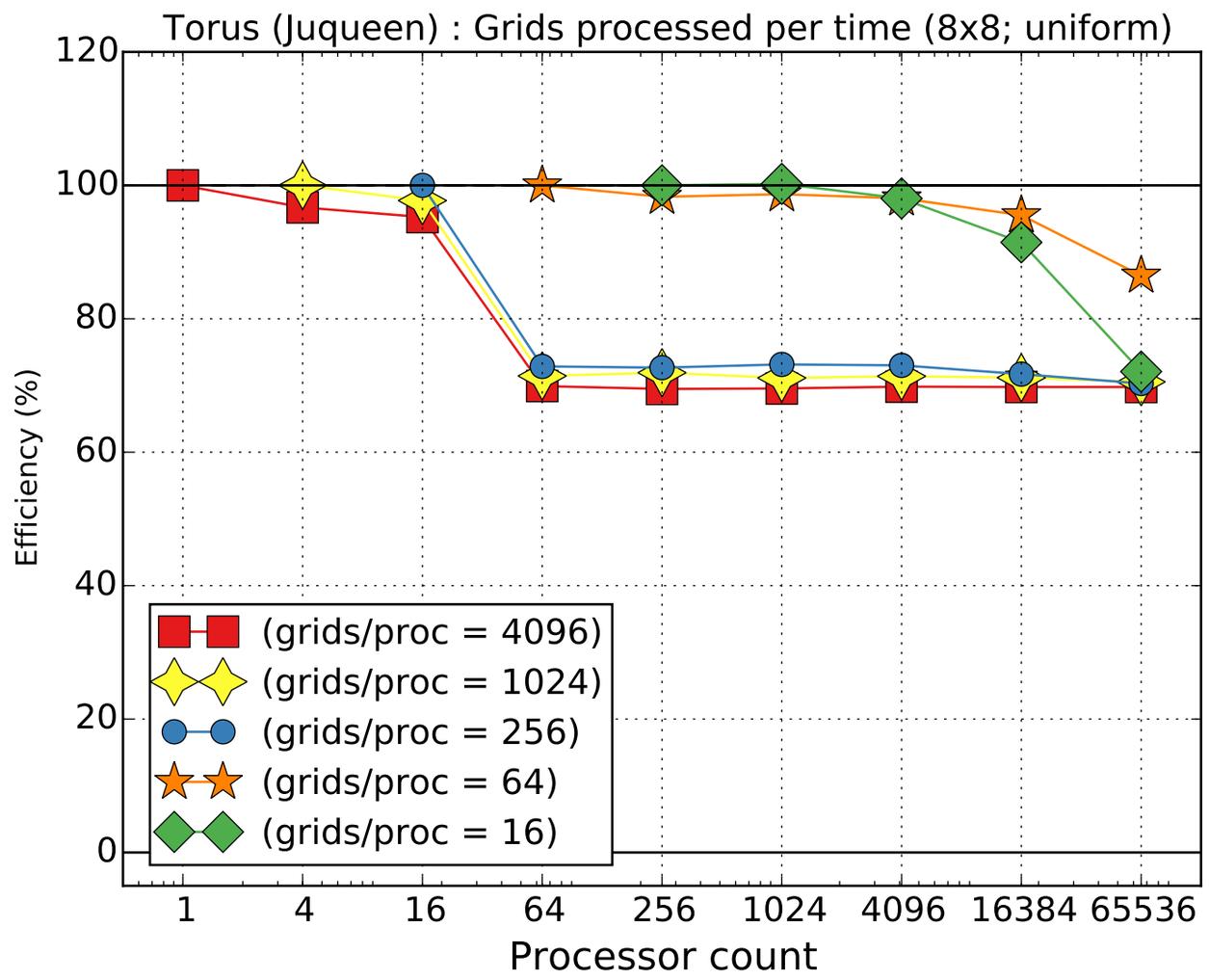
Procs	1x1	2x2	4x4	8x8	16x16	32x32	64x64	128x128	256x256
1	**5498**	---	---	---	---	---	---	---	---
4	1374	**5498**	---	---	---	---	---	---	---
16	343	1374	**5498**	---	---	---	---	---	---
64	85	343	1374	**5498**	---	---	---	---	---
256	21	85	343	1374	**5498**	---	---	---	---
1024	---	21	85	343	1374	**5498**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

Strong scaling

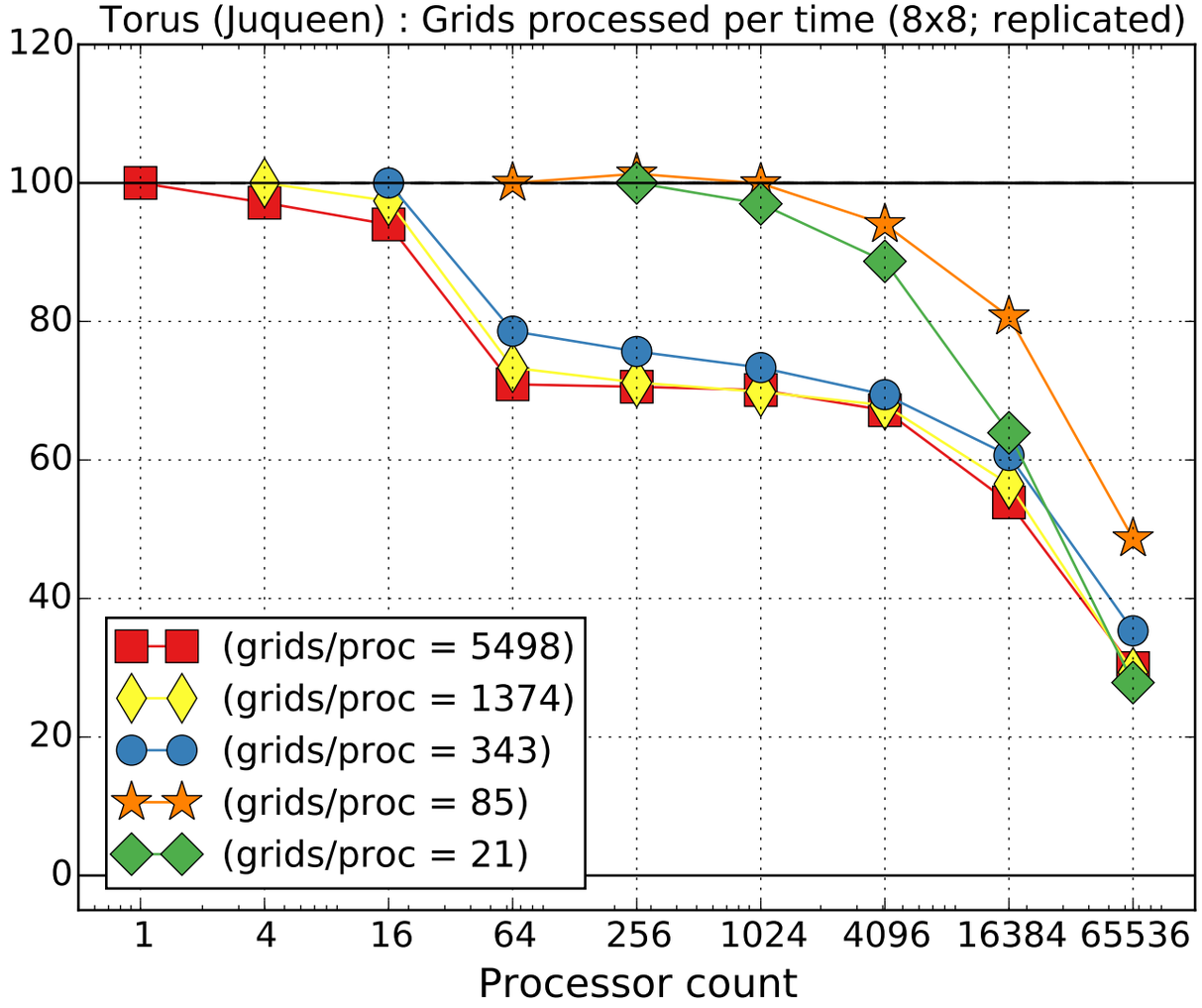
Weak scaling

Weak Scaling - 8x8 grids

Uniform



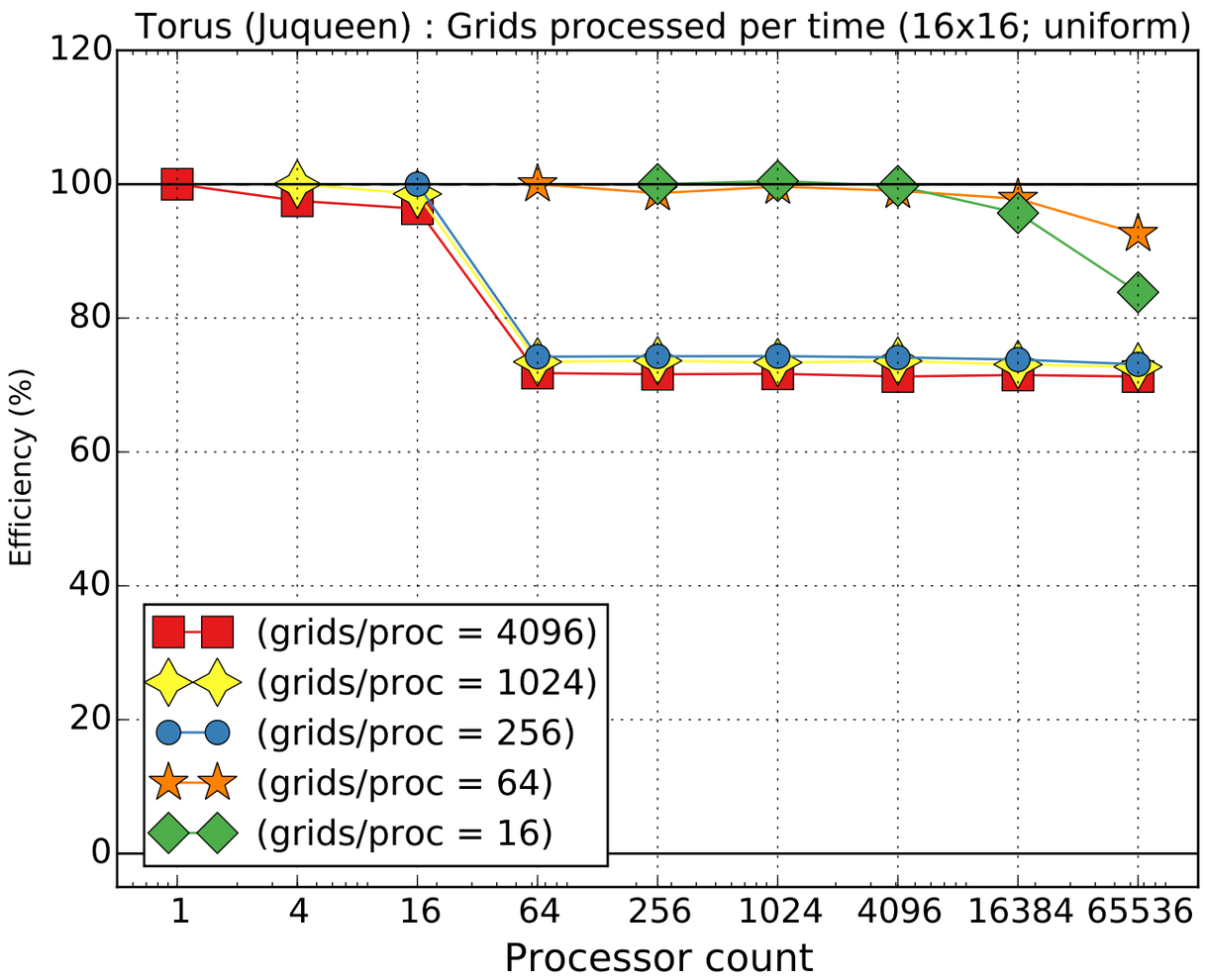
Adaptive



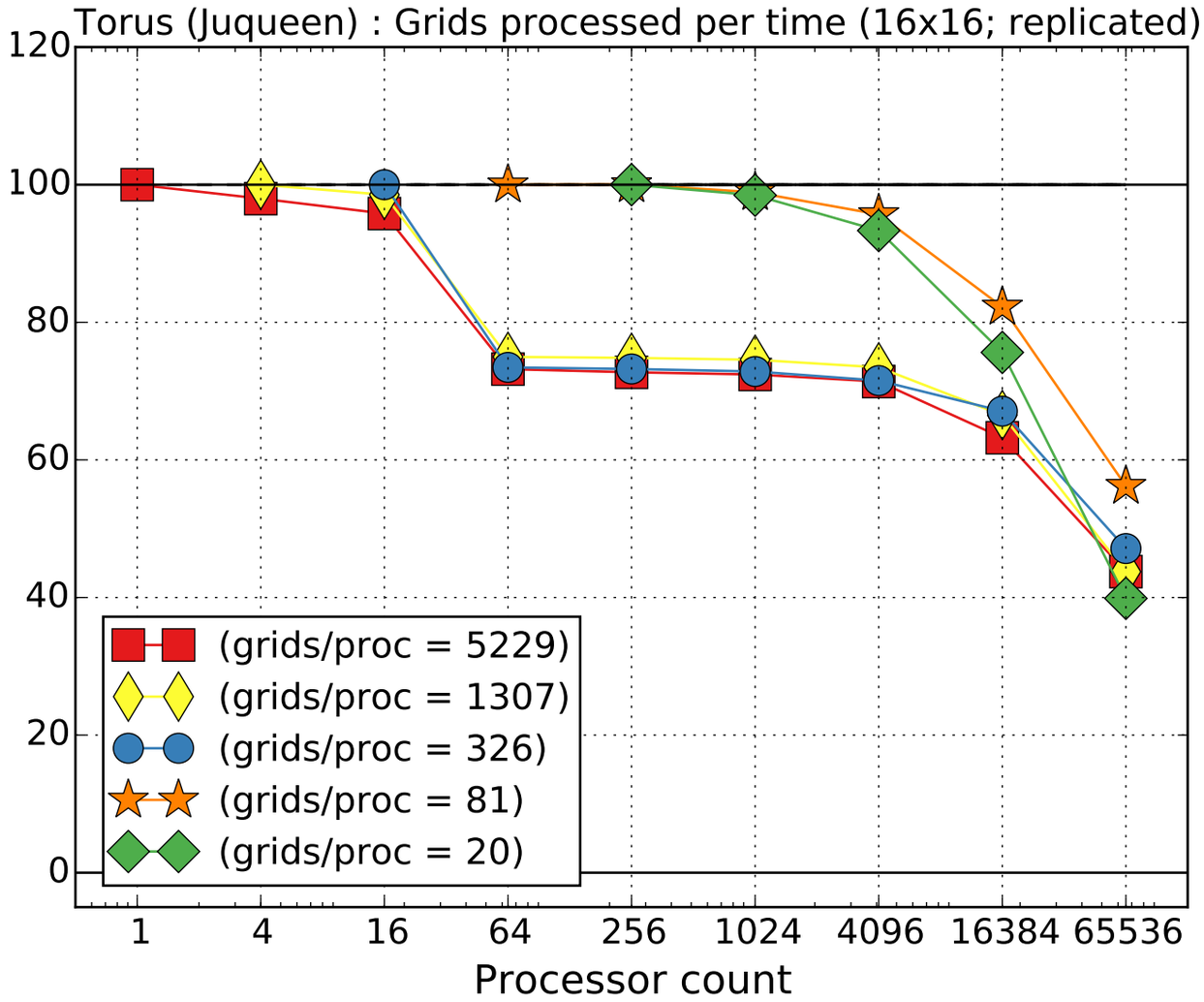
Grids processed per total wall time

Weak Scaling - 16x16 grids

Uniform



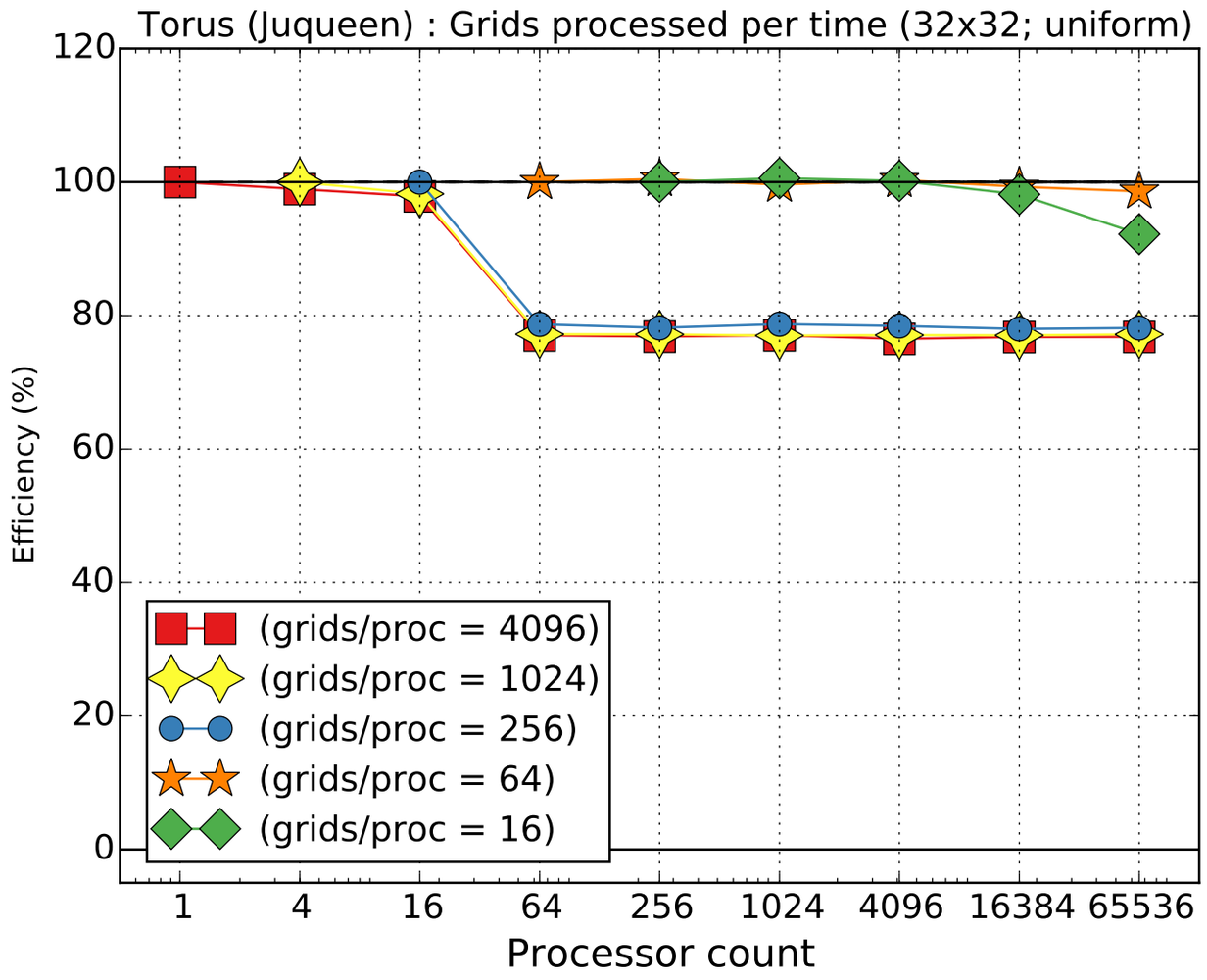
Adaptive



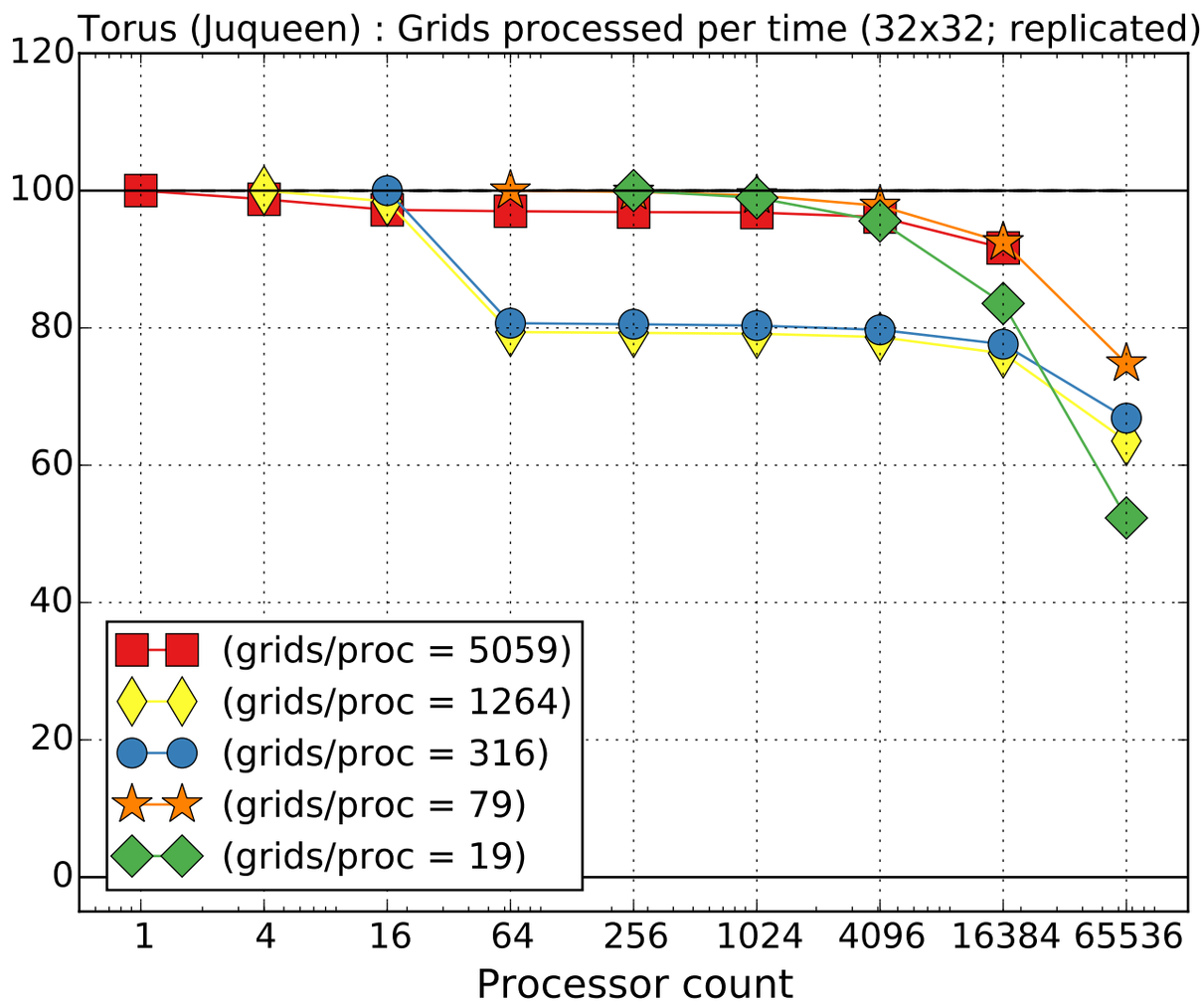
Grids processed per total wall time

Weak Scaling - 32x32 grids

Uniform



Adaptive



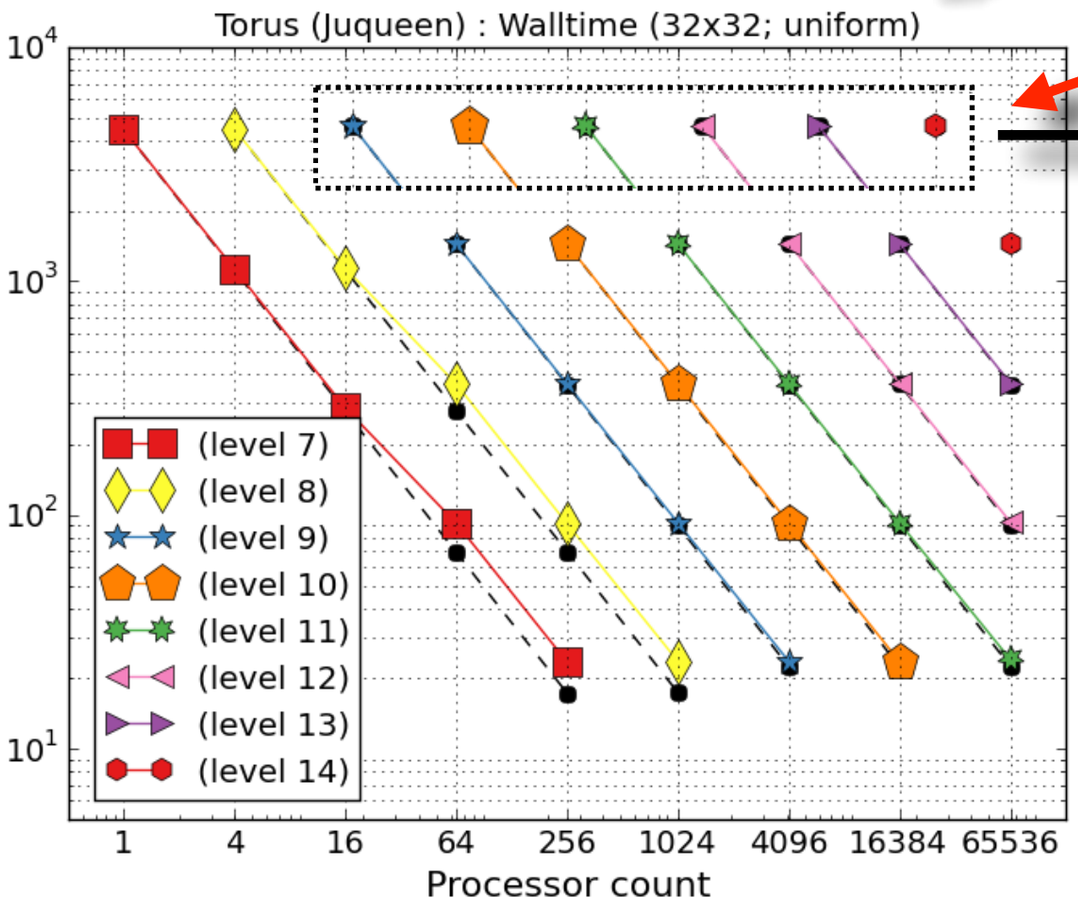
Grids processed per total wall time

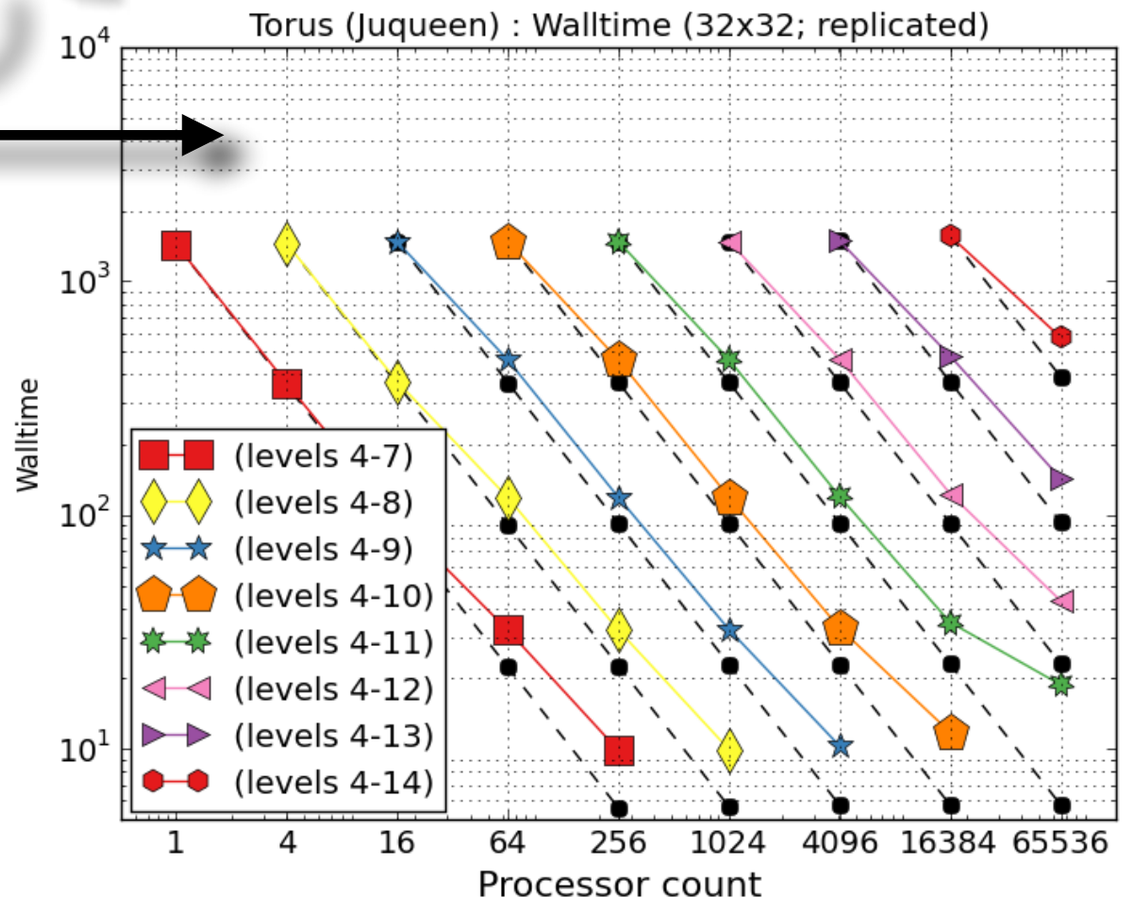
Strong scaling - 32x32

Uniform

not computed

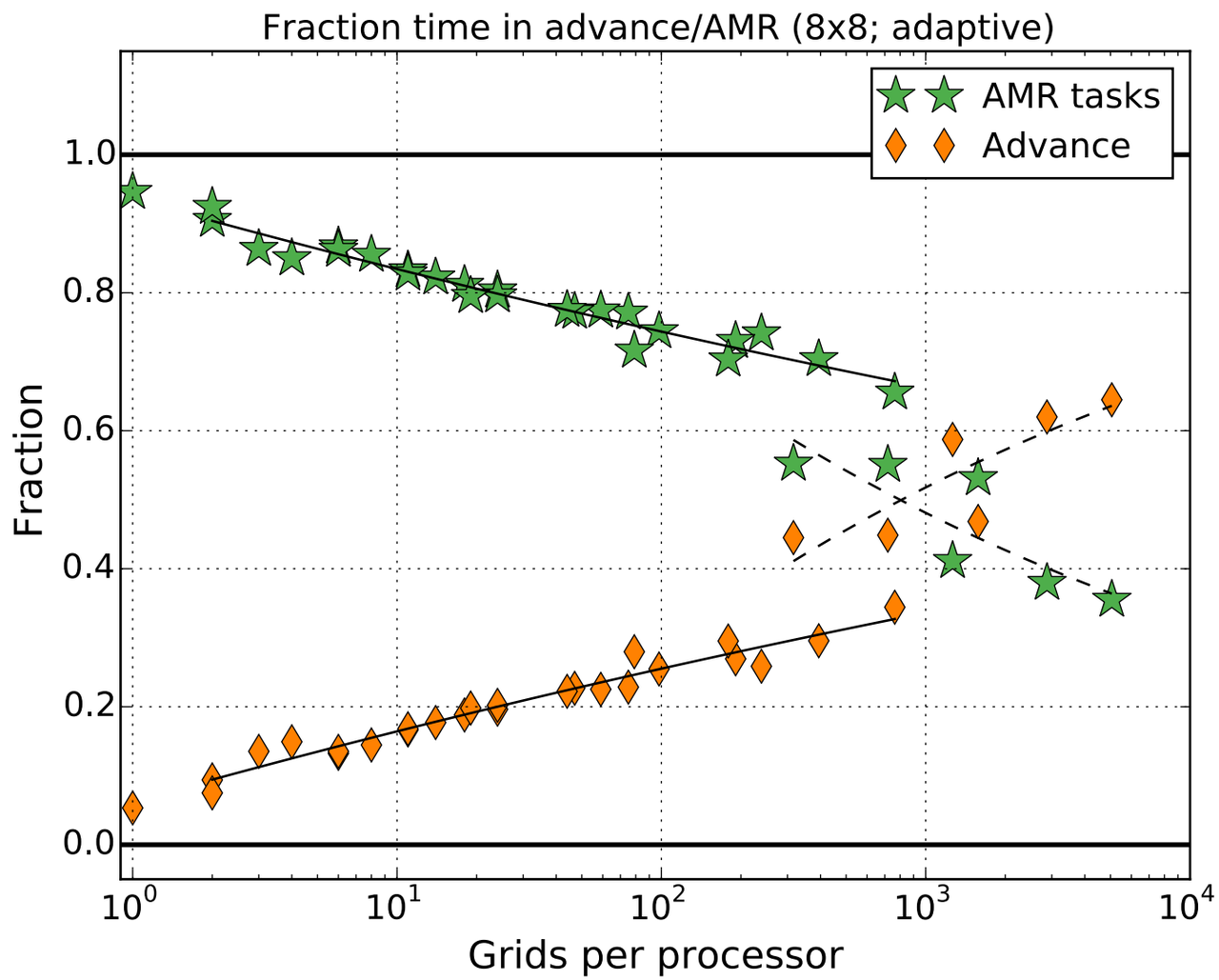
Adaptive





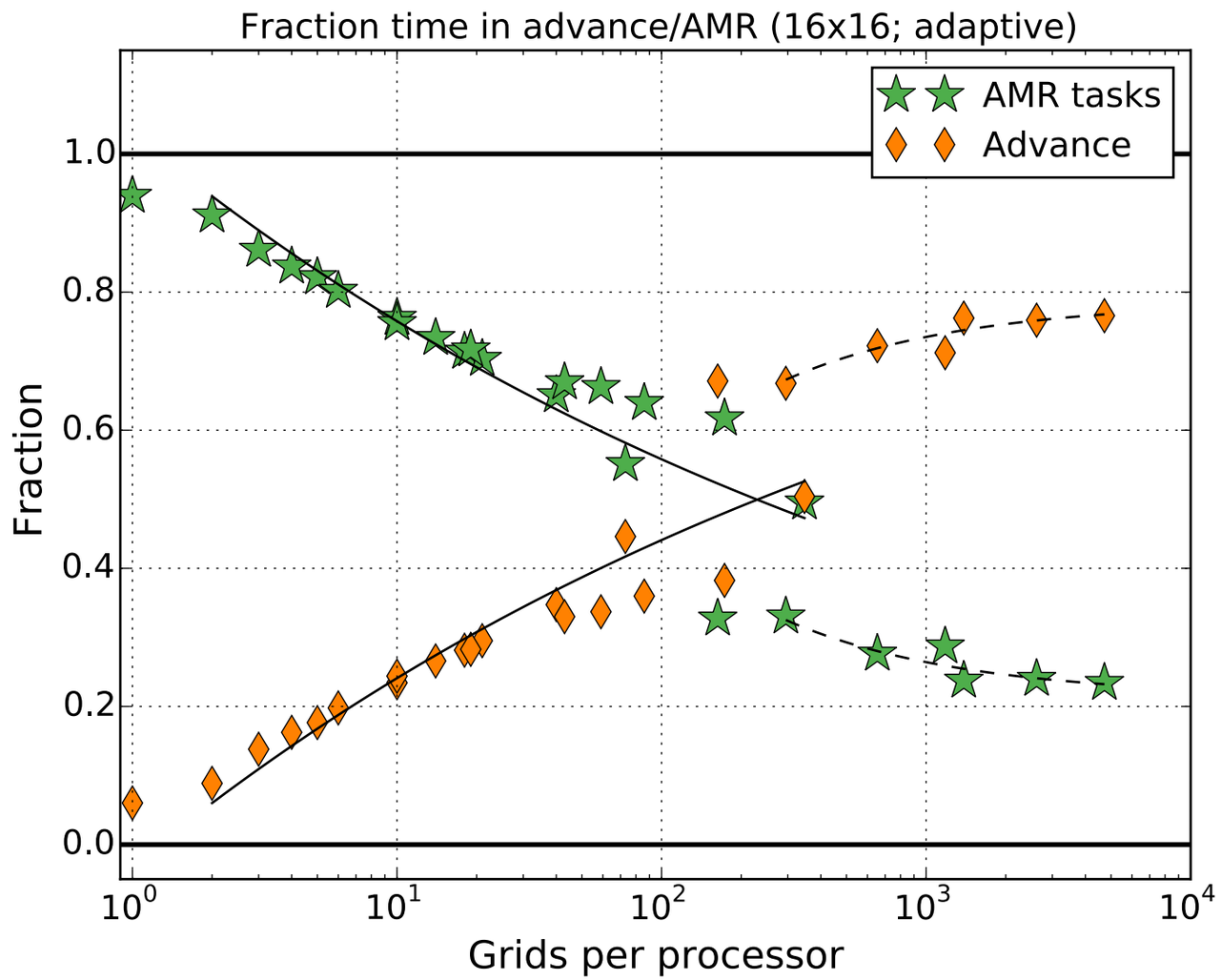
AMR is faster by about a factor of 10

Adaptive scaling - 8x8



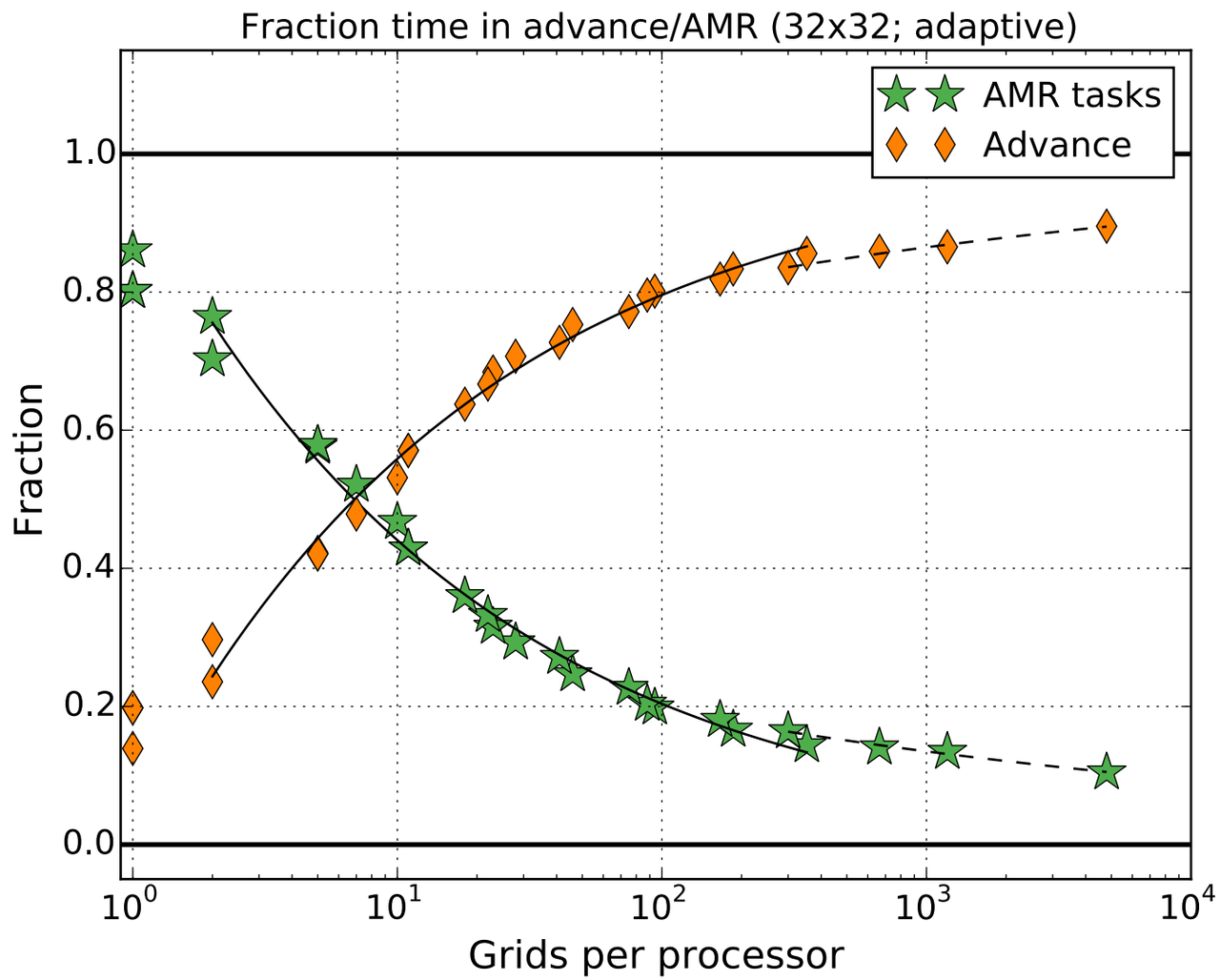
Plot timing results as function of granularity

Adaptive scaling - 16x16



Plot timing results as function of granularity

Adaptive scaling - 32x32



Plot timing results as function of granularity

Natural Hazards Modeling

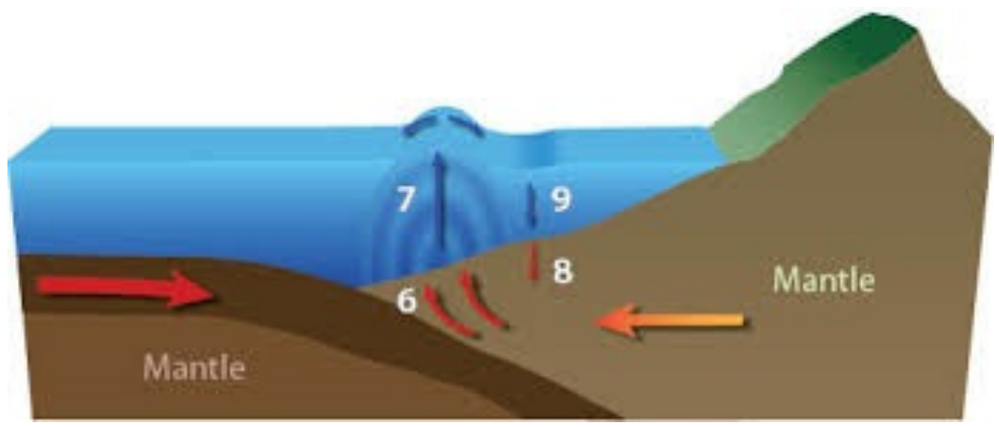
Many natural hazards are well suited to numerical modeling and simulation

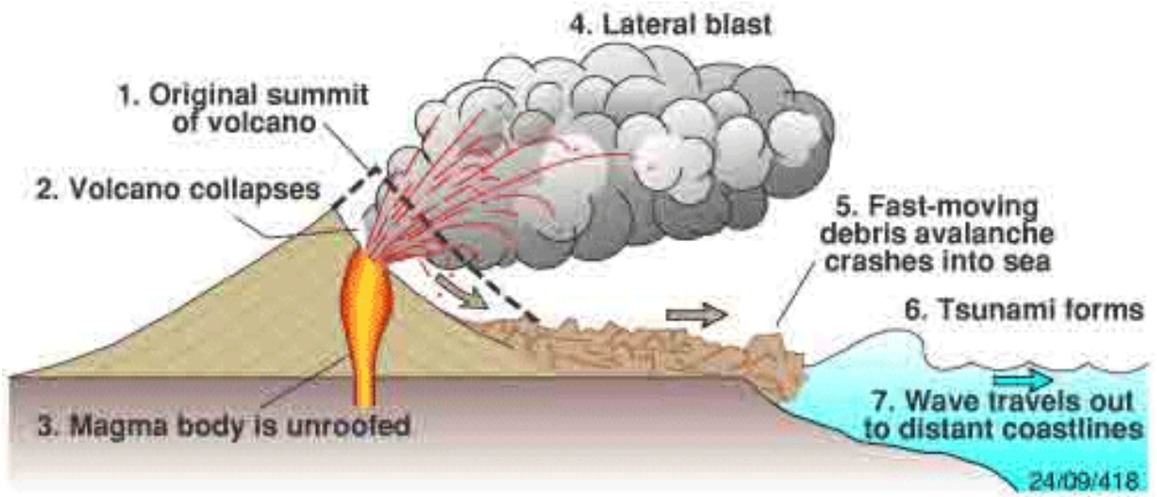
- Tsunamis
- Volcanic ash plumes
- Earthquakes
- Wildfires
- Storm surges
- Debris flows and landslides
- Flooding

Mathematical models are now routinely used to understand these phenomena and predict their behavior

Challenges

- Unknown initial conditions
- Complicated geometry that is not well known or understood.
- Coupled events requiring several different models
- Many temporal and spatial scales are involved
- Simulations should be done in real time to be most useful

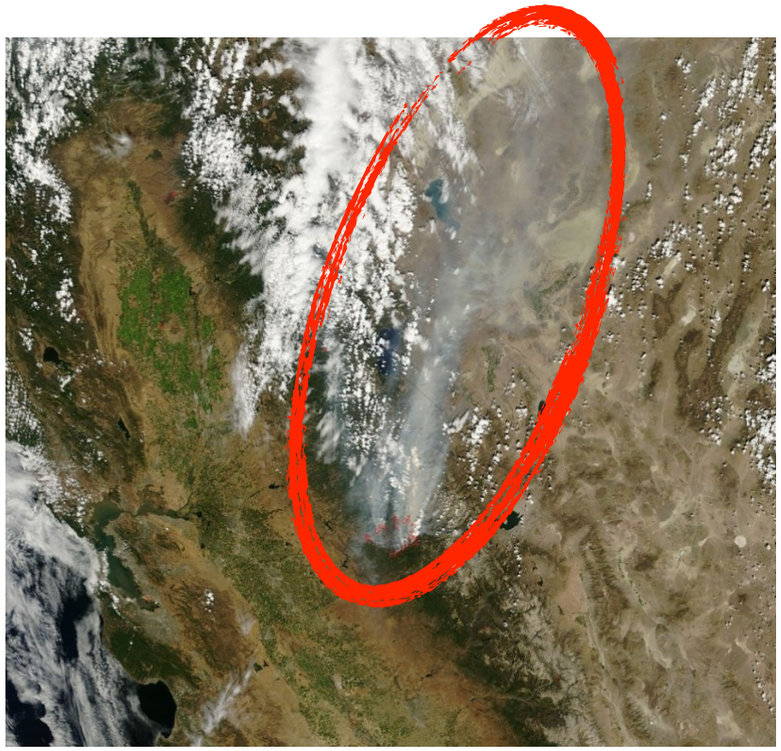




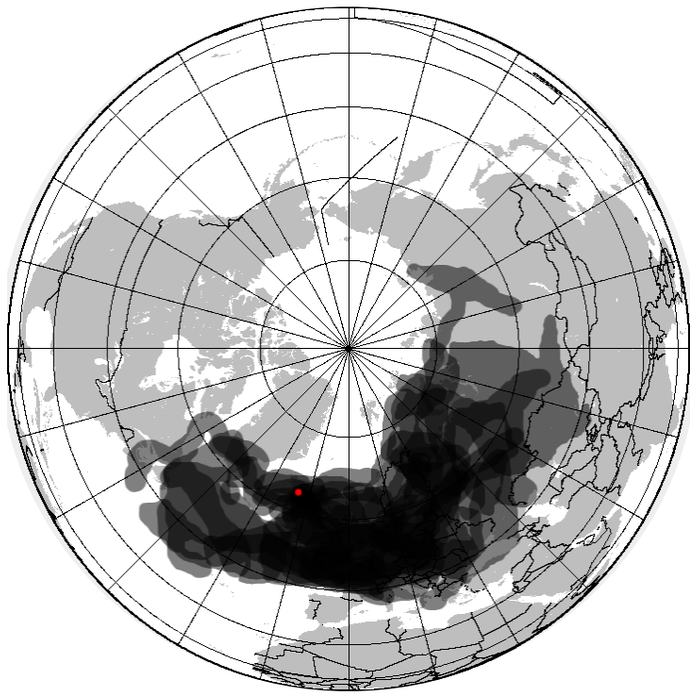
Challenges

- Small scales must be tracked for a long time over long distances
- Domains must be large enough to allow for numerical modeling of the entire event.

Computations can become very expensive without dynamic, multi-resolution capabilities.

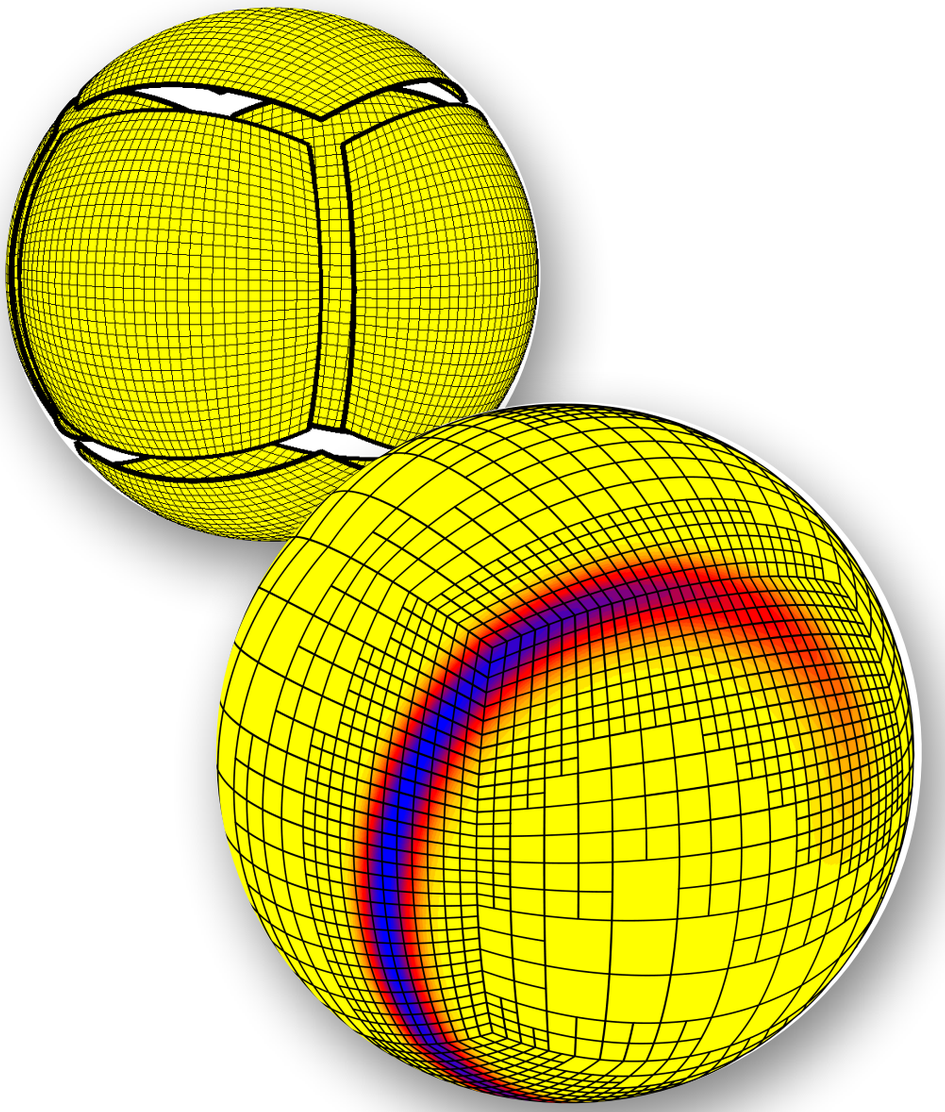


Smoke from the California Rim Fire, Sept 2013

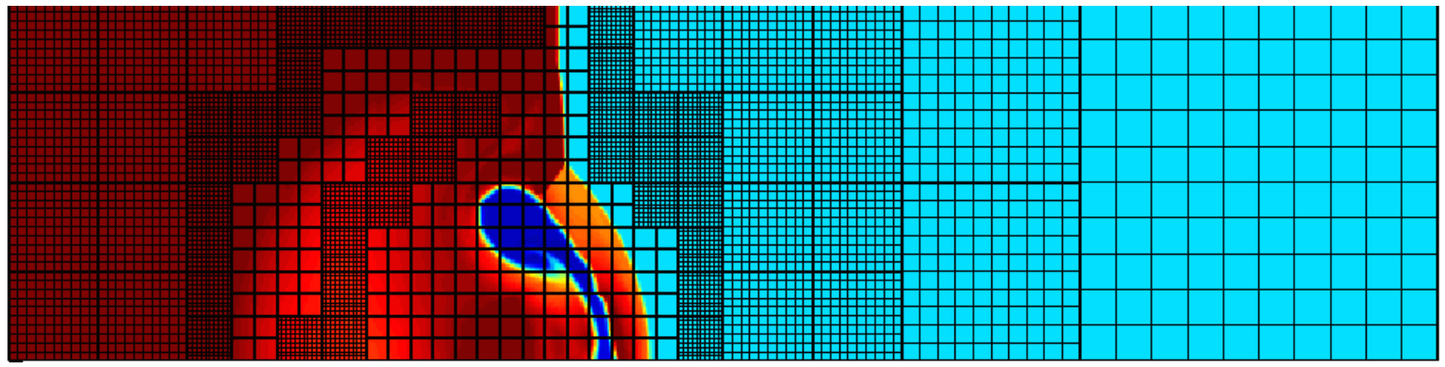


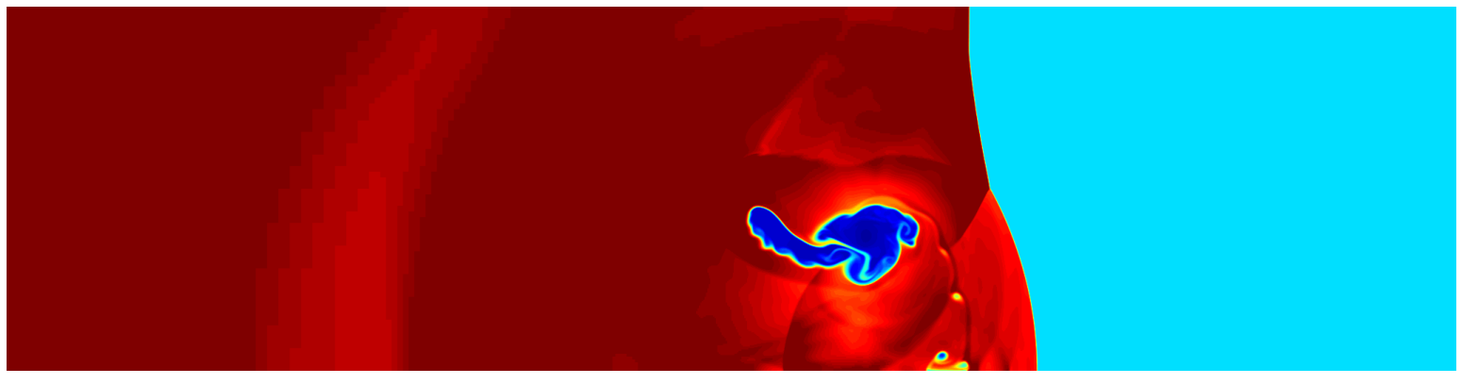
2010 Eruption in Iceland

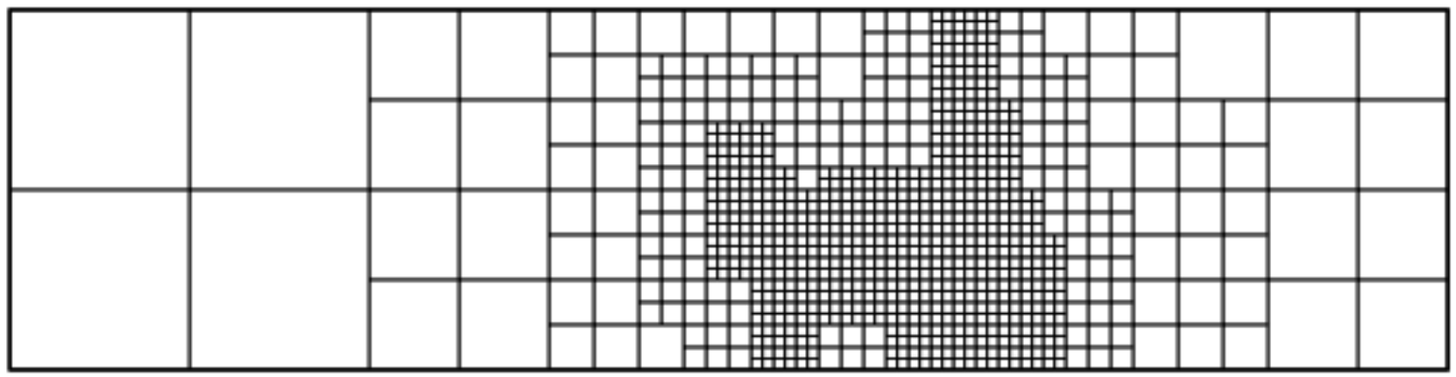
Examples using Clawpack



Correlated Cosine bell on a cubed sphere grid (Tracer Transport Benchmark, Lauritzen, Skamarock et al)







Shock hitting low density bubble

What next?

- Develop ForestClaw version of GeoClaw (Melody Shih, Columbia University)

```
../forestclaw/configure \
    –enable-mpi –enable-geoclaw –without-blas \
    CXXFLAGS="-O2 -Wall" FFLAGS="-O2 -cpp” \
    FCFLAGS=“-O2 -cpp" CFLAGS="-O2 -Wal FC=gfortran
```

- 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)

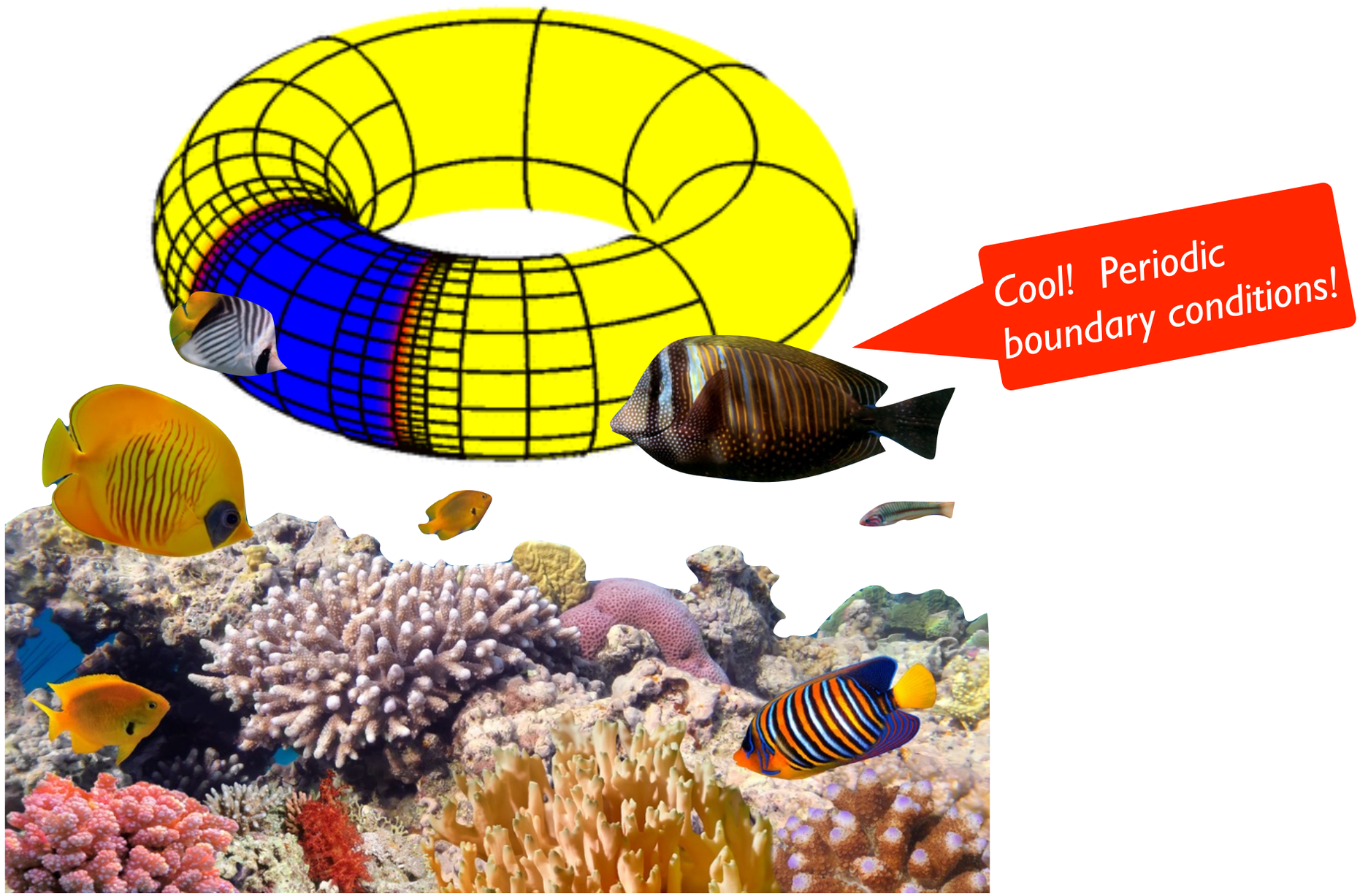
www.forestclaw.org

ForestClaw - stats

```
SLOC  Directory  SLOC-by-Language (Sorted)
55778    p4est              ansic=53965,sh=1547,python=226,awk=40
46062    applications       fortran=30556,cpp=6956,ansic=3971,f90=2270,python=2222,
                            perl=44,csh=43
26816    src_solvers        fortran=12776,cpp=6834,f90=6621,ansic=585
17378    sc                 ansic=17322,sh=56
14252    src_top_dir        cpp=7137,ansic=7115

945      src_fortran_source2d fortran=945
927      src_mappings       fortran=850,ansic=77

466      docs               sh=335,perl=131

Totals grouped by language (dominant language first):
ansic:           83117 (48.92%)
fortran:         45163 (26.58%)
cpp:             21312 (12.5
f90:              9490 (5.59
sh:               6437 (3.79
python:           3908 (2.30
```

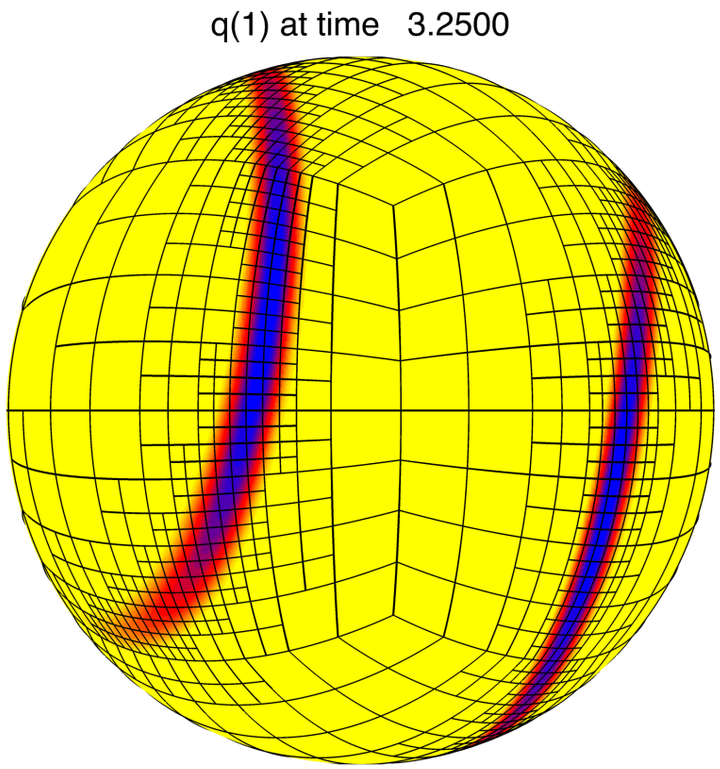
```
Total Physical Source Lines of Code (SLOC)                = 169,905
Development Effort Estimate, Person-Years (Person-Months) = 43.93 (527.14)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.25 (27.06)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 19.48
Total Estimated Cost to Develop                           = $ 5,934,144
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Flow on a torus



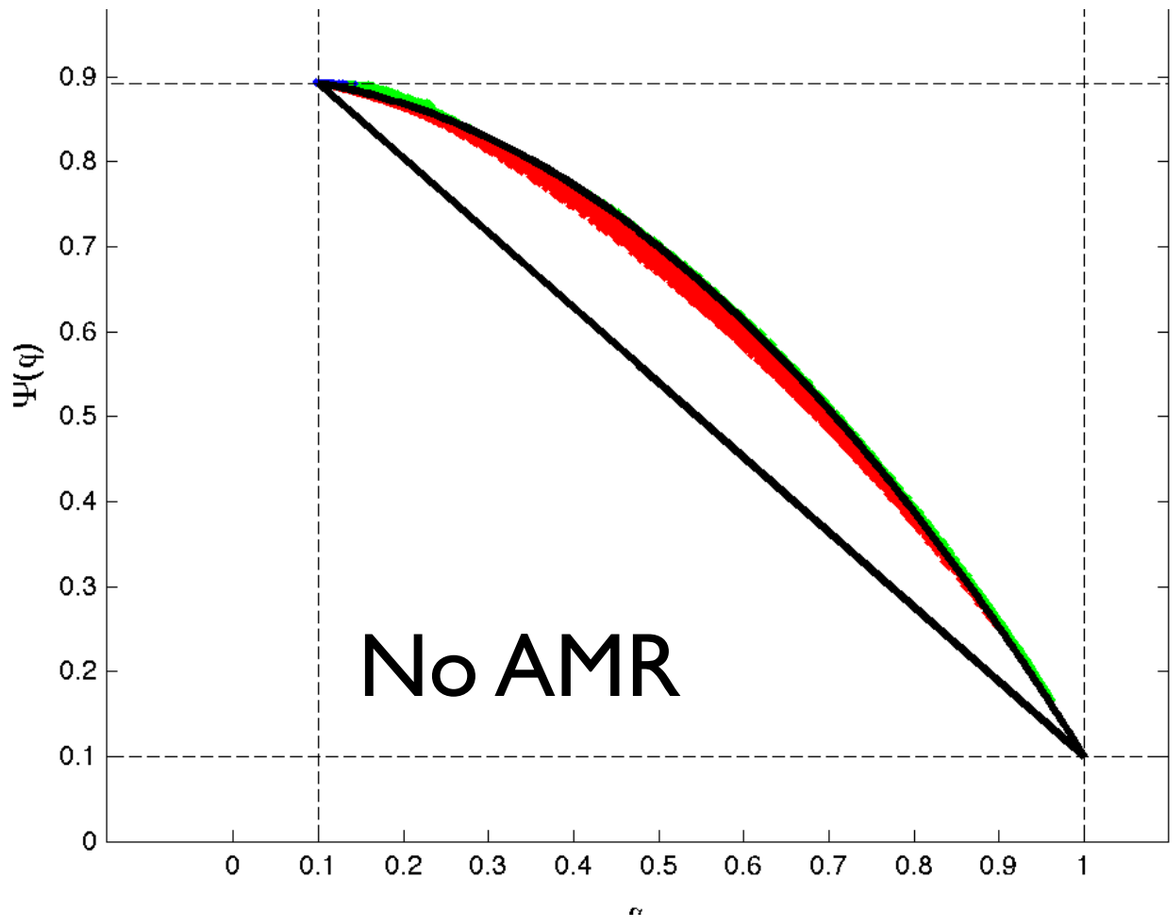
Mixing diagnostic

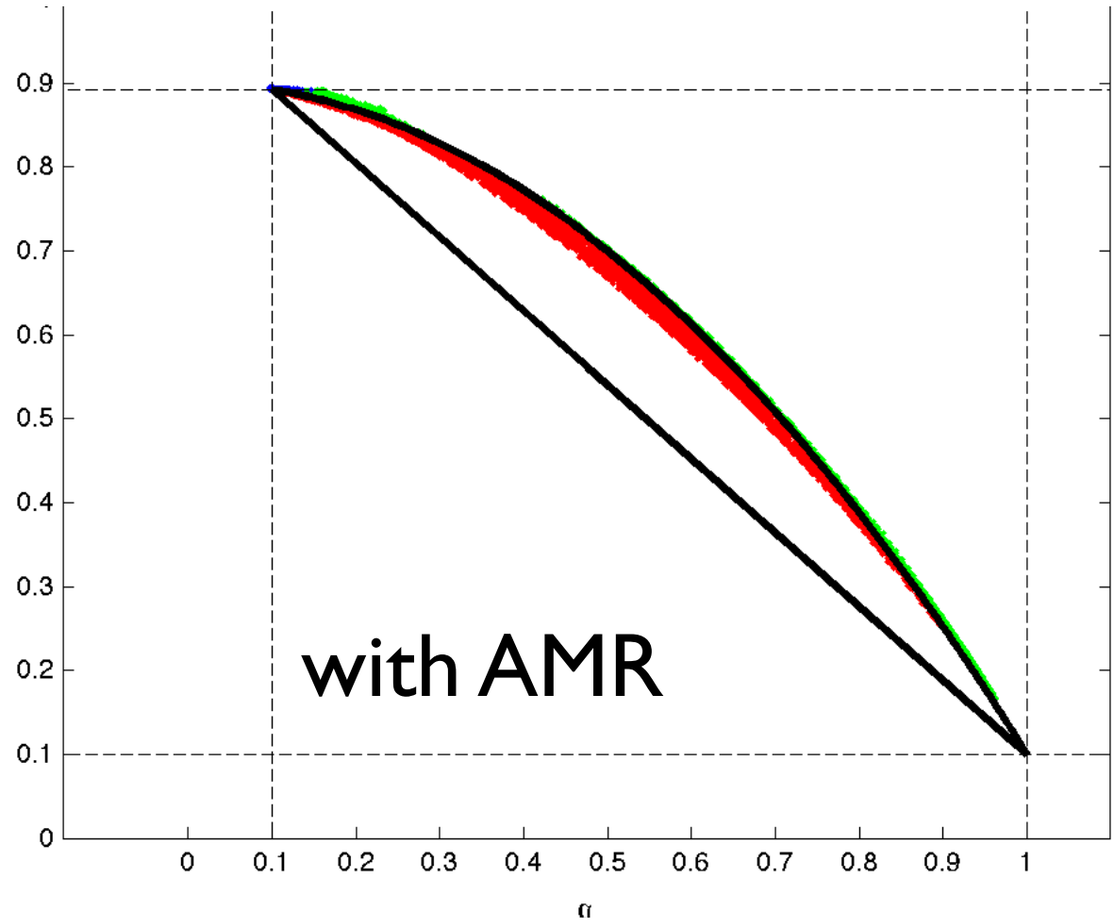
Preservation of functional relationship between tracers.



Lauritzen, P. H., Skamarock, W. C., Prather, M. J., and and, M. A. Taylor. A standard test case suite for two-dimensional linear transport on the sphere. Geoscientific Model Development 5 (2012), 887–901.

Mixing diagnostics (256 x 256)

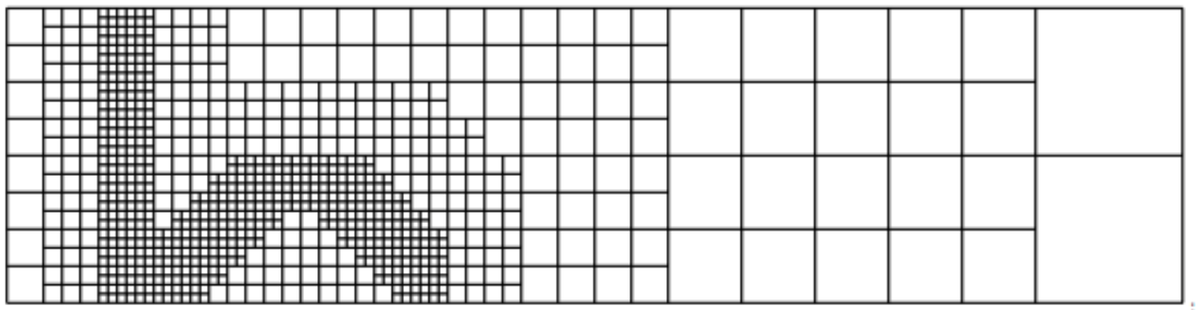


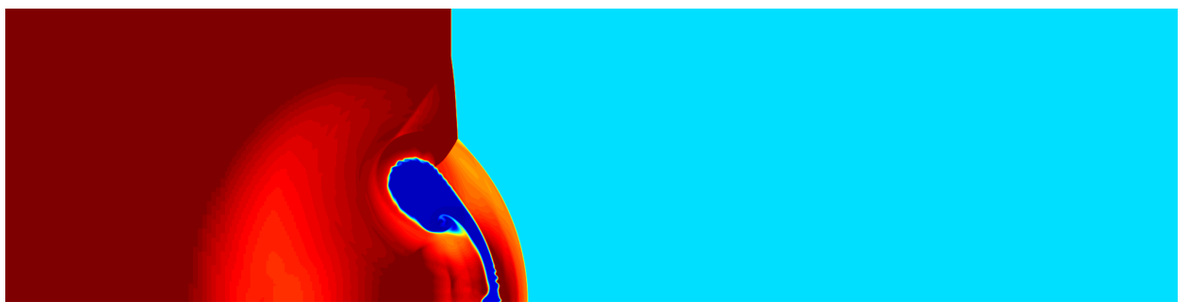


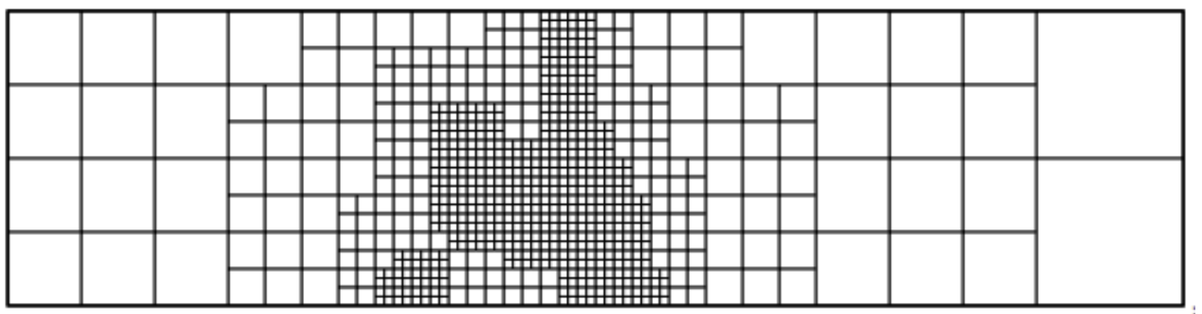
	Diagnostic	Fraction	Diagnostic	Fraction
Real mixing (r)	5.45E-04	0.7565	1.68E-04	0.7709
Range preserving mixing (g)	1.49E-04	0.2070	4.2E-05	0.1925
Under and over shoots (b)	2.63E-05	0.0365	7.98E-06	0.0366

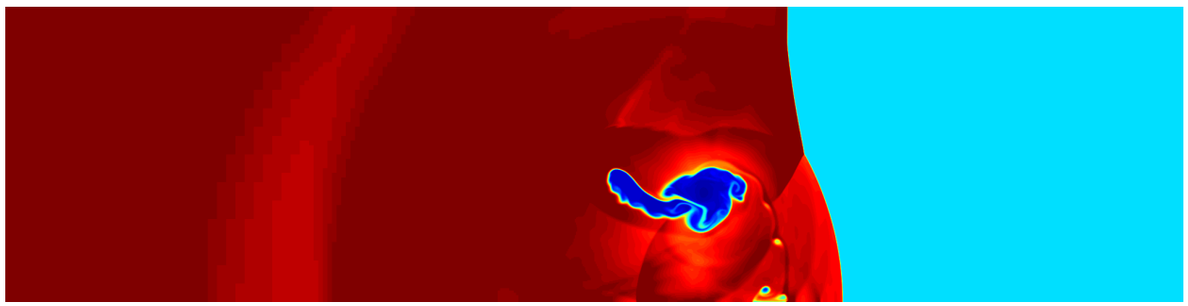
Thanks!

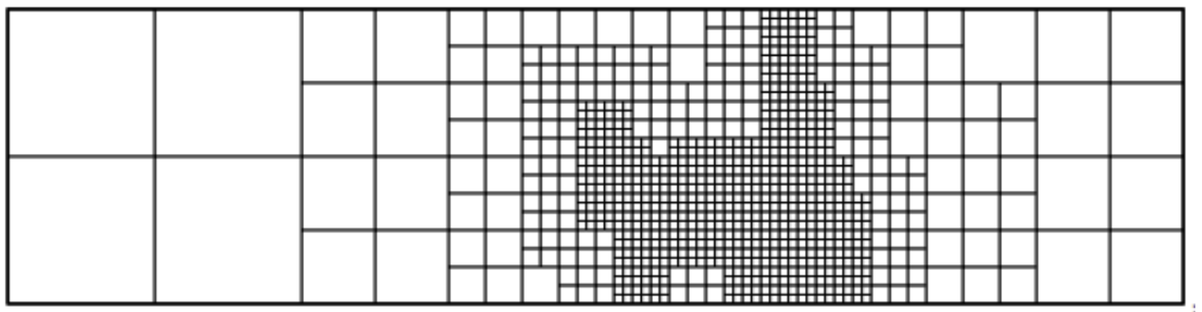
Shockbubble problem

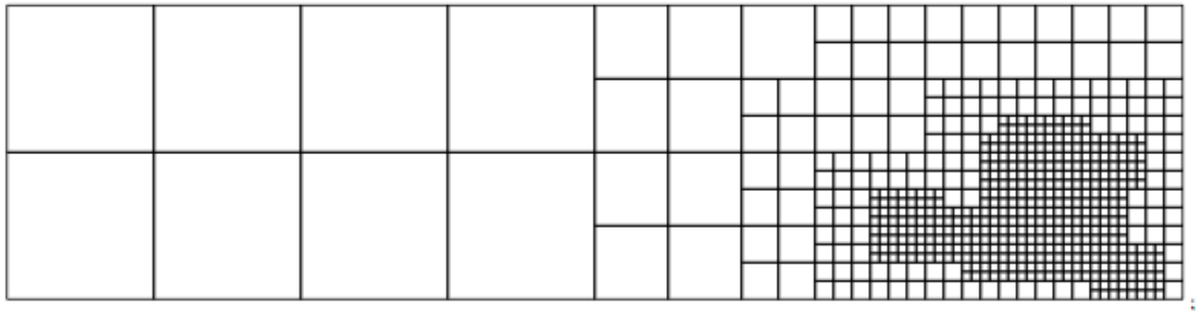












ForestClaw - P > 4

ForestClaw (8×8, $T = 0.1$)							
P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	5	664	553.3	501.4 (90.6)	51.2 (9.3)	0.0 (0.0)	0.4×10^5
4	6	368	646.8	532.8 (82.4)	66.8 (10.3)	46.3 (7.2)	0.4×10^5
16	7	202	834.1	559.3 (67.1)	83.6 (10.0)	189.5 (22.7)	4.7×10^5
64	8	93	1026.7	515.4 (50.2)	104.4 (10.2)	402.4 (39.2)	12.7×10^5
256	9	41	1289.9	435.7 (33.8)	102.9 (8.0)	742.1 (57.5)	33.9×10^5
1024	10	19	1524.8	402.1 (26.4)	108.6 (7.1)	1000.3 (65.6)	105.5×10^5

Increase processor count and resolution simultaneously

More results

ForestClaw (8×8, $T = 0.1$)

P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	5	664	553.3	501.4 (90.6)	51.2 (9.3)	0.0 (0.0)	0.4×10^5
4	6	368	646.8	532.8 (82.4)	66.8 (10.3)	46.3 (7.2)	0.4×10^5
16	7	202	834.1	559.3 (67.1)	83.6 (10.0)	189.5 (22.7)	4.7×10^5
64	8	93	1026.7	515.4 (50.2)	104.4 (10.2)	402.4 (39.2)	12.7×10^5
256	9	41	1289.9	435.7 (33.8)	102.9 (8.0)	742.1 (57.5)	33.9×10^5
1024	10	19	1524.8	402.1 (26.4)	108.6 (7.1)	1000.3 (65.6)	105.5×10^5

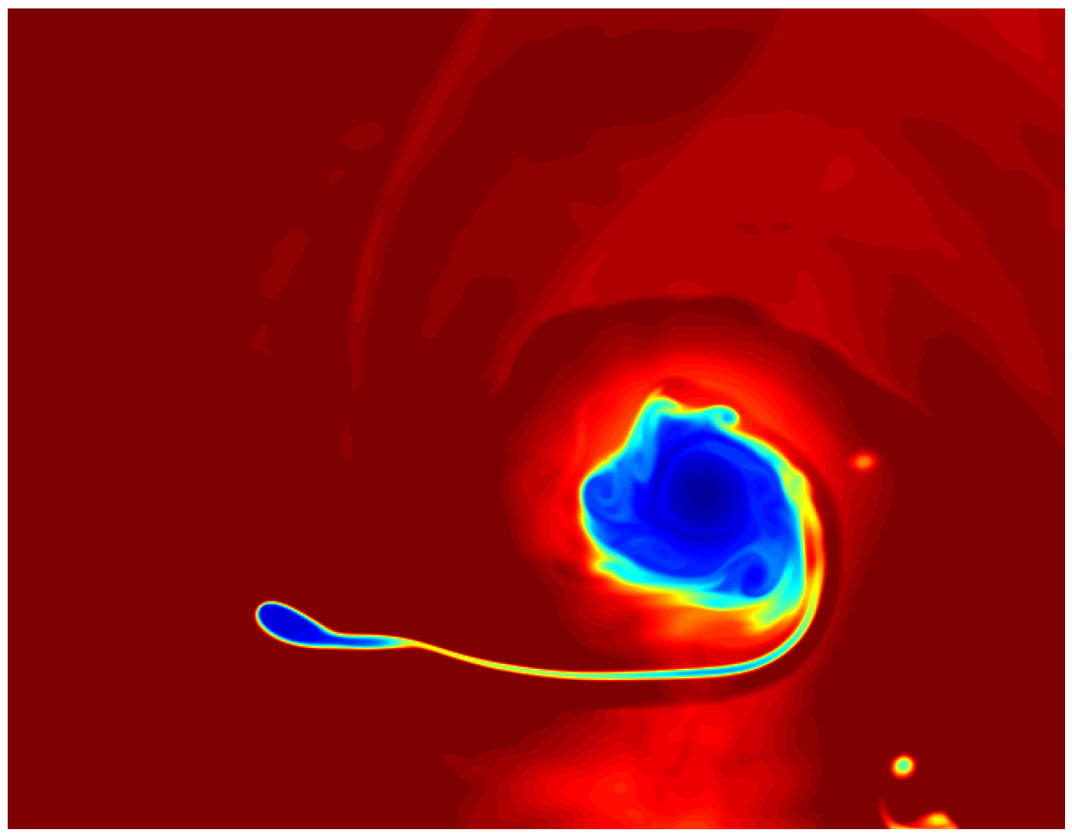
ForestClaw (16×16, $T = 0.1$)

P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	4	290	738.2	706.5 (95.7)	31.2 (4.2)	0.0 (0.0)	0.5×10^5
4	5	163	920.4	763.7 (83.0)	48.9 (5.3)	106.9 (11.6)	1.9×10^5
16	6	89	1098.2	796.7 (72.5)	64.9 (5.9)	234.2 (21.3)	6.6×10^5
64	7	49	1456.6	938.6 (64.4)	96.0 (6.6)	416.7 (28.6)	21.2×10^5

ForestClaw (32×32, $T = 0.1$)

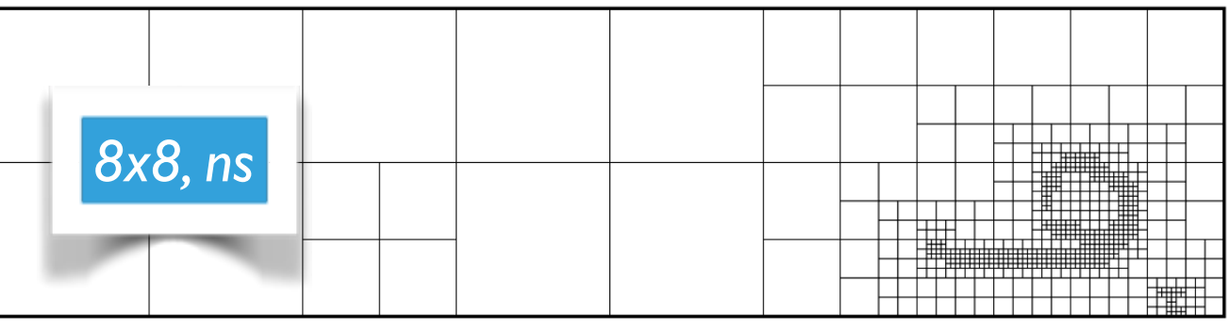
P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
4	3	27	303.8	242.0 (79.6)	8.9 (2.9)	52.6 (17.3)	2.1×10^5
16	4	18	444.2	307.5 (69.2)	17.8 (4.0)	116.0 (26.1)	7.2×10^5
64	5	10	614.7	366.9 (59.7)	26.9 (4.4)	215.7 (35.1)	22.3×10^5
256	6	5	837.6	377.1 (45.0)	32.6 (3.9)	416.1 (49.7)	67.6×10^5

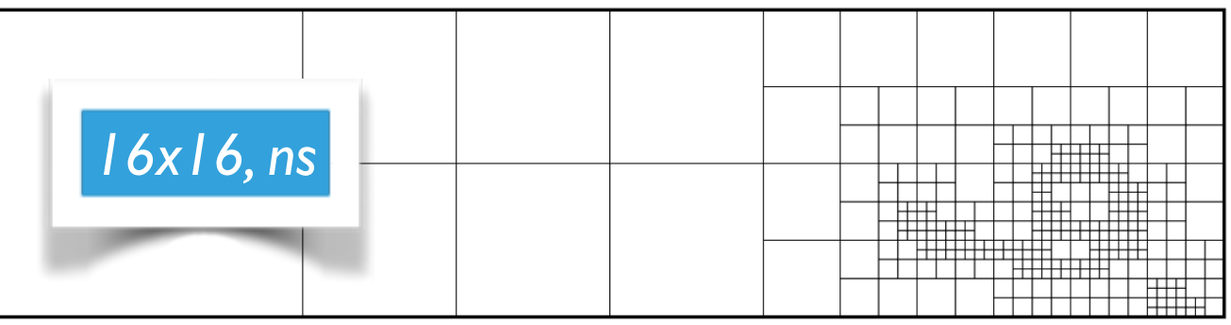
Uniform solution

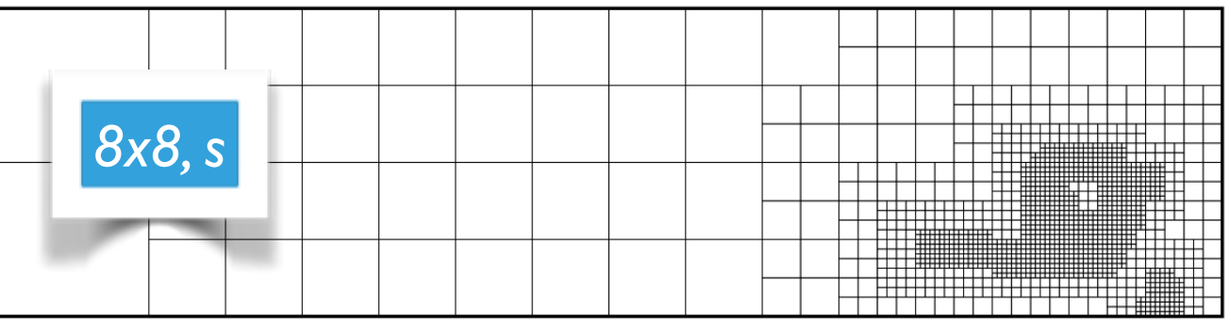


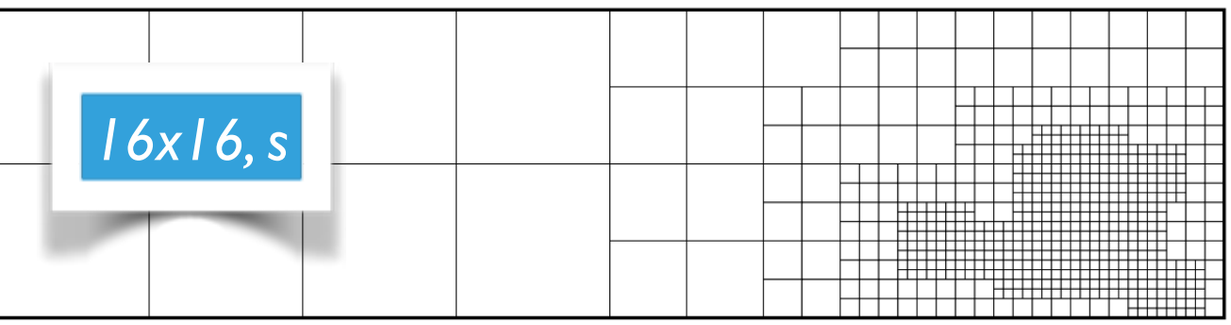
Meshes

ForestClaw

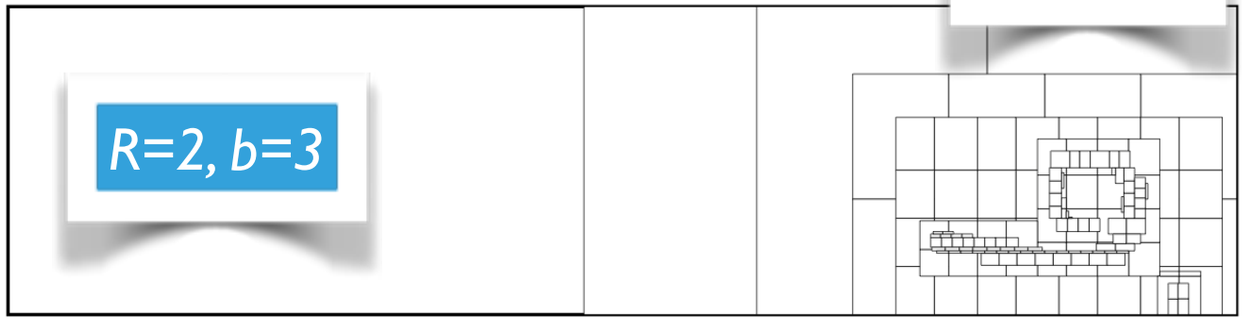


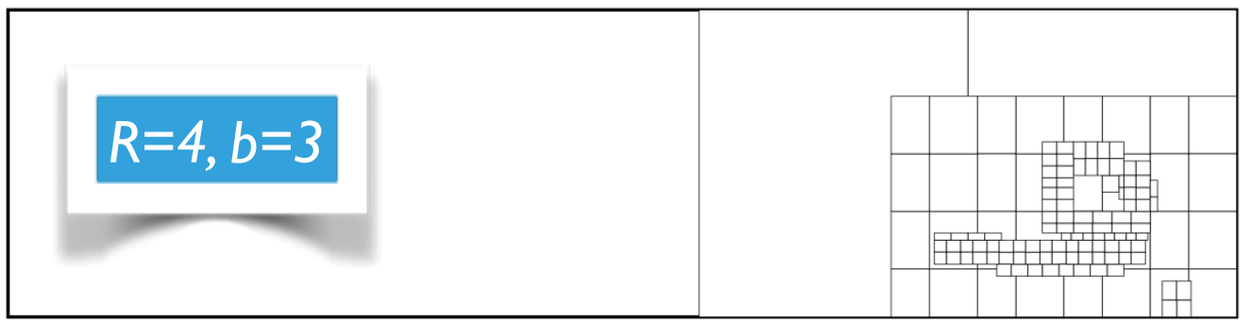


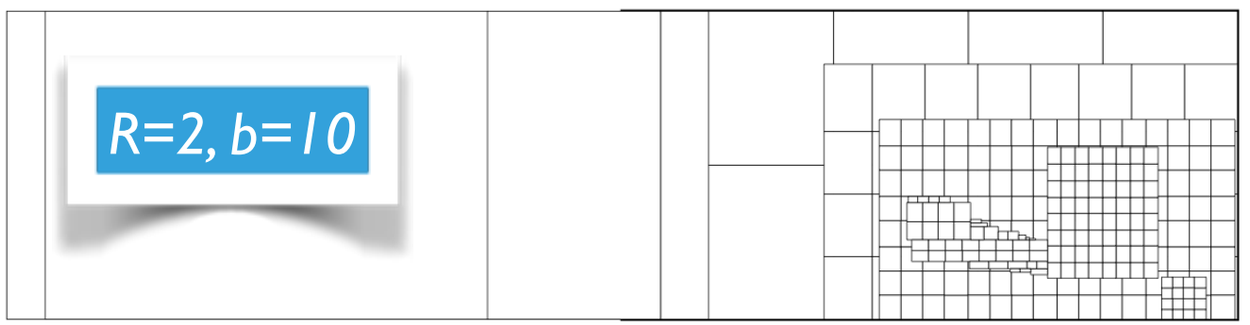




AMRClaw

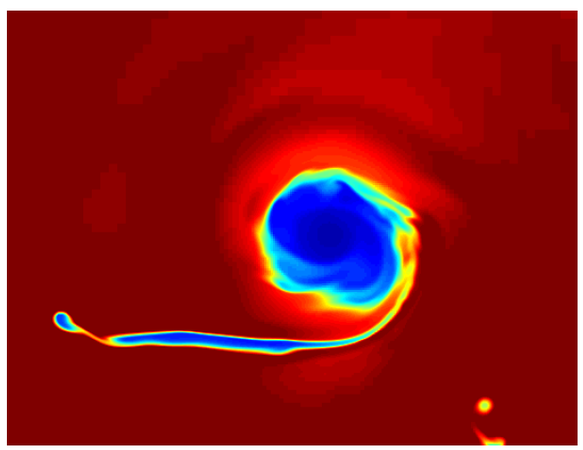




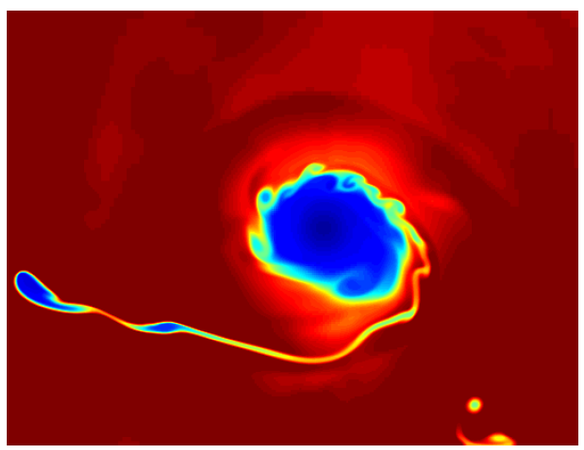


ForestClaw and AMRClaw

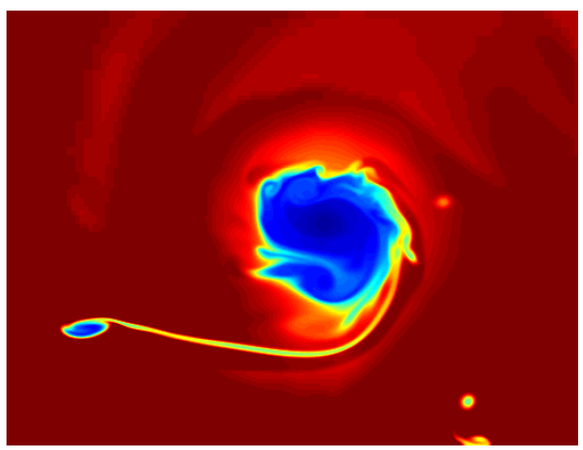
ForestClaw



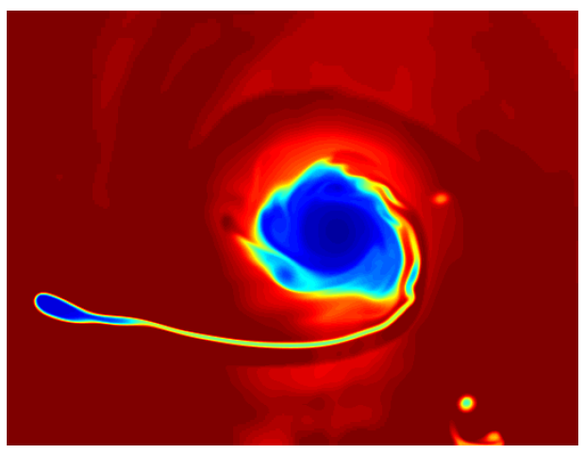
$M = 8,$ non-smooth
$t = 228.9$



$M = 16,$ non-smooth
$t = 304.6$

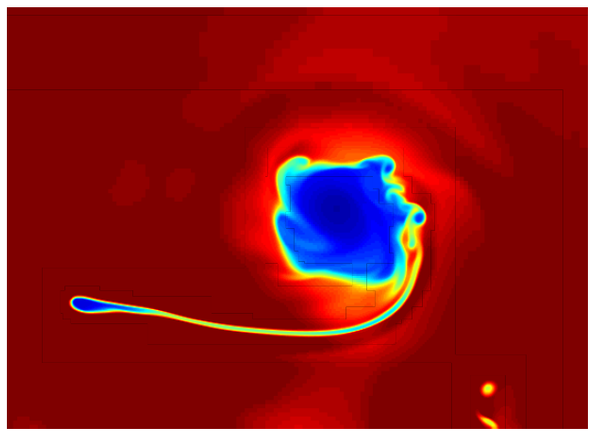


$M = 8,$ smooth
$t = 511.5$

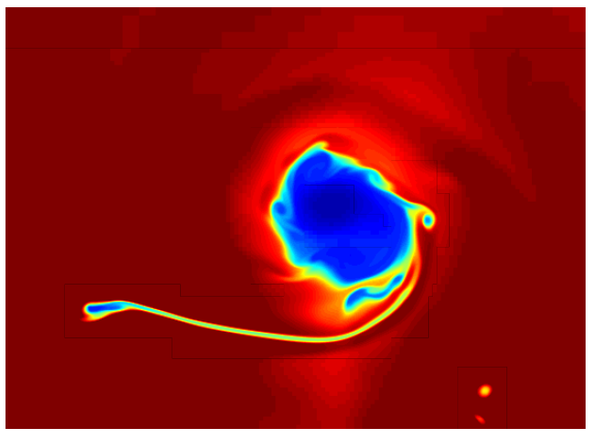


$M = 16,$ smooth
$t = 688.8$

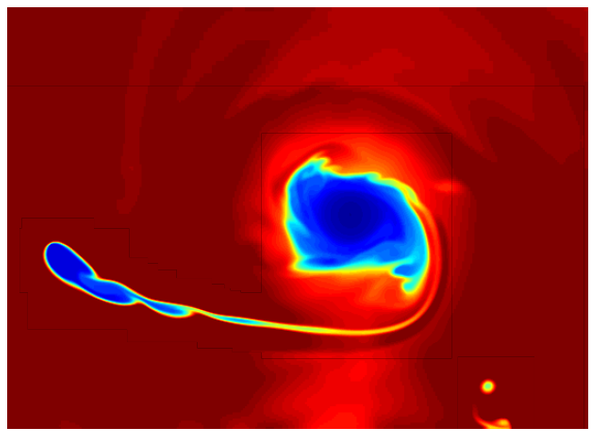
AMRClaw



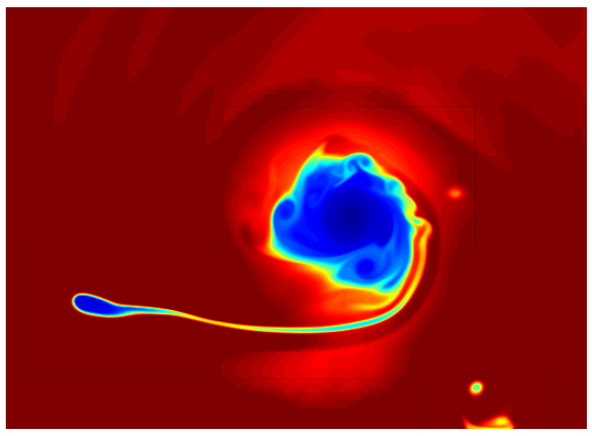
$R = 2,\ b = 3$
$t = 253.9$



$R = 4,\ b = 3$
$t = 293.8$



$R = 2,\ b = 10$
$t = 553.4$



$R = 4,\ b = 10$
$t = 620.2$

Timing results (4-cores)

ForestClaw							
M	S	Wall	Adv. (%)	Fill (%)	Comm. (%)	Regrid (%)	Rate
8	(ns)	228.9	169.8 (74.2)	24.6 (10.8)	33.4 (14.6)	0.2 (0.1)	16.3×10^5
16	(ns)	304.6	232.1 (76.2)	20.4 (6.7)	50.7 (16.6)	0.2 (0.1)	22.0×10^5
8	(s)	511.5	397.1 (77.6)	45.8 (9.0)	66.7 (13.0)	0.3 (0.1)	16.8×10^5
16	(s)	688.8	550.8 (80.0)	37.6 (5.5)	97.8 (14.2)	0.5 (0.1)	21.5×10^5
Uniform		3300.6	2825.2 (85.6)	25.3 (0.8)	246.4 (7.5)	0.0 (0)	30.5×10^5

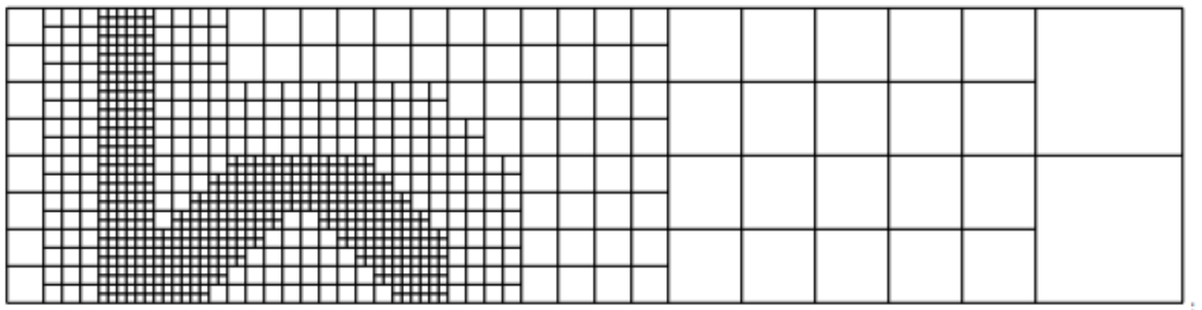
AMRClaw						
R	b	Wall	Adv. (%)	Fill (%)	Regrid (%)	Rate
2	3	253.9	210.1 (82.7)	6.3 (2.5)	33.3 (13.1)	18.4×10^5
4	3	293.8	279.9 (95.3)	4.8 (1.6)	7.3 (2.5)	20.9×10^5
2	10	553.4	467.2 (84.4)	7.3 (1.3)	69.9 (12.6)	19.7×10^5
4	10	620.2	597.0 (96.3)	6.2 (1.0)	13.2 (2.1)	22.9×10^5
Uniform		3716.3	3684.3 (99.1)	3.8 (0.1)	0.0 (0)	27.2×10^5

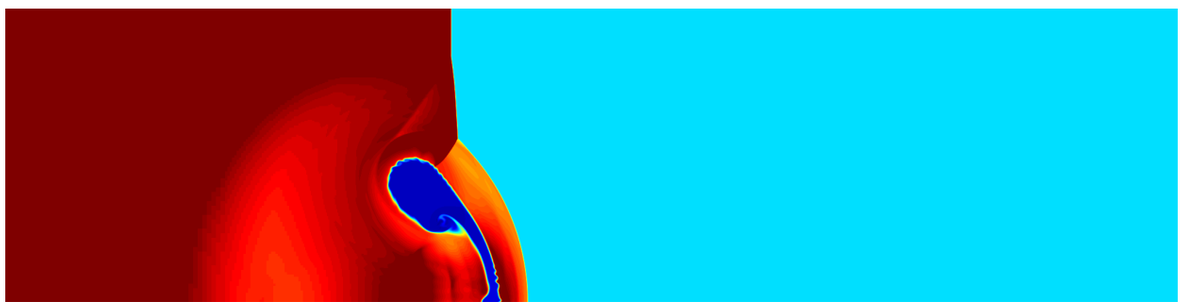
Rate = number of cells processed per second

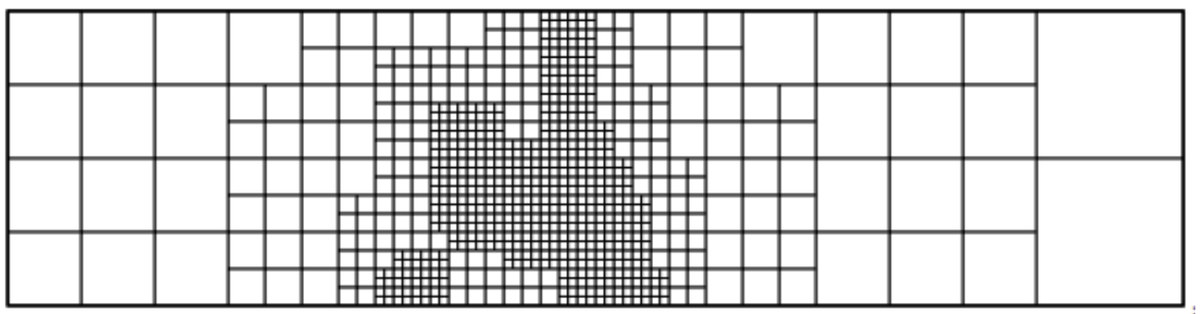
Backup slides

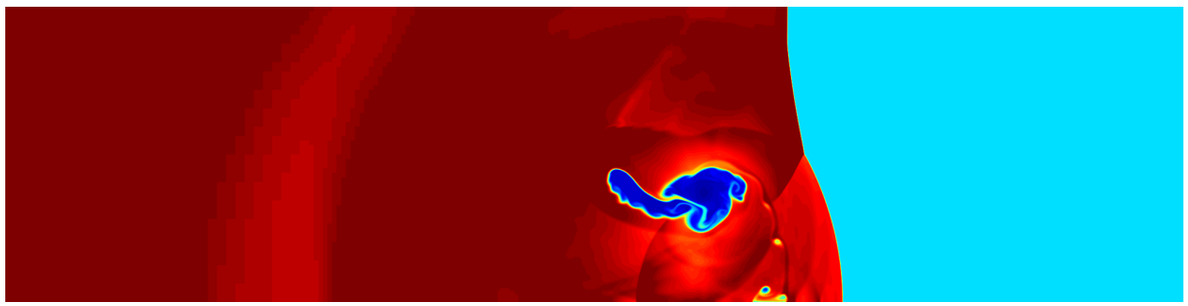
Shockbubble Problem

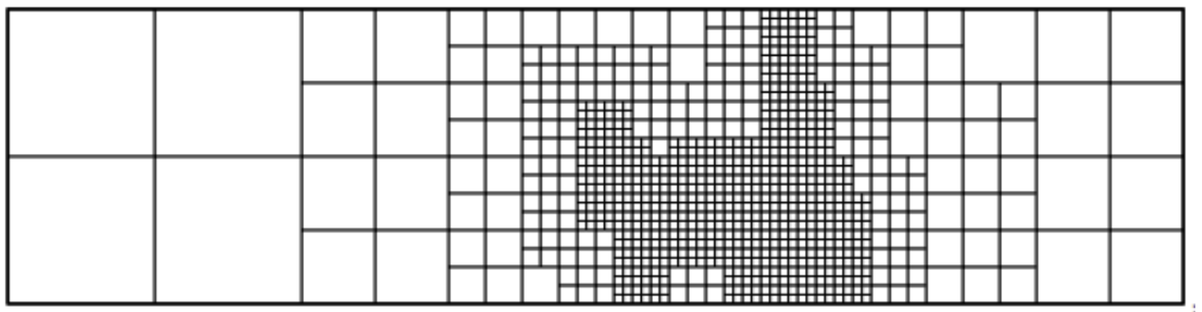
Shockbubble problem

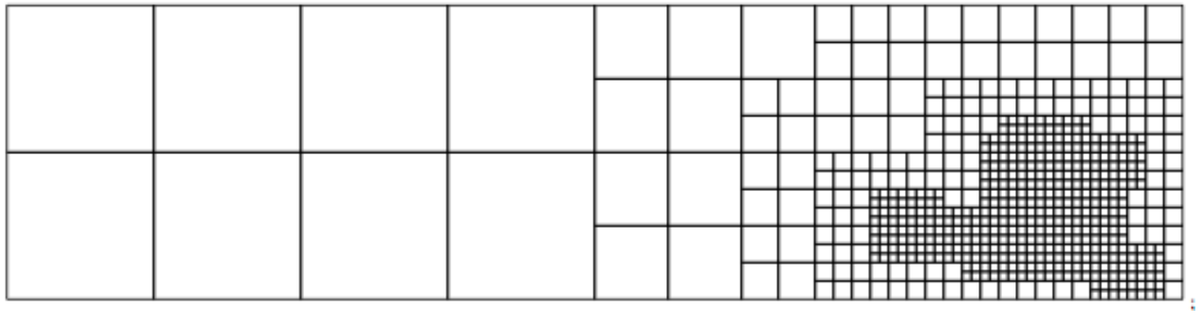












ForestClaw - P > 4

ForestClaw (8×8, $T = 0.1$)							
P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	5	664	553.3	501.4 (90.6)	51.2 (9.3)	0.0 (0.0)	0.4×10^5
4	6	368	646.8	532.8 (82.4)	66.8 (10.3)	46.3 (7.2)	0.4×10^5
16	7	202	834.1	559.3 (67.1)	83.6 (10.0)	189.5 (22.7)	4.7×10^5
64	8	93	1026.7	515.4 (50.2)	104.4 (10.2)	402.4 (39.2)	12.7×10^5
256	9	41	1289.9	435.7 (33.8)	102.9 (8.0)	742.1 (57.5)	33.9×10^5
1024	10	19	1524.8	402.1 (26.4)	108.6 (7.1)	1000.3 (65.6)	105.5×10^5

Increase processor count and resolution simultaneously

More results

ForestClaw (8×8, $T = 0.1$)

P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	5	664	553.3	501.4 (90.6)	51.2 (9.3)	0.0 (0.0)	0.4×10^5
4	6	368	646.8	532.8 (82.4)	66.8 (10.3)	46.3 (7.2)	0.4×10^5
16	7	202	834.1	559.3 (67.1)	83.6 (10.0)	189.5 (22.7)	4.7×10^5
64	8	93	1026.7	515.4 (50.2)	104.4 (10.2)	402.4 (39.2)	12.7×10^5
256	9	41	1289.9	435.7 (33.8)	102.9 (8.0)	742.1 (57.5)	33.9×10^5
1024	10	19	1524.8	402.1 (26.4)	108.6 (7.1)	1000.3 (65.6)	105.5×10^5

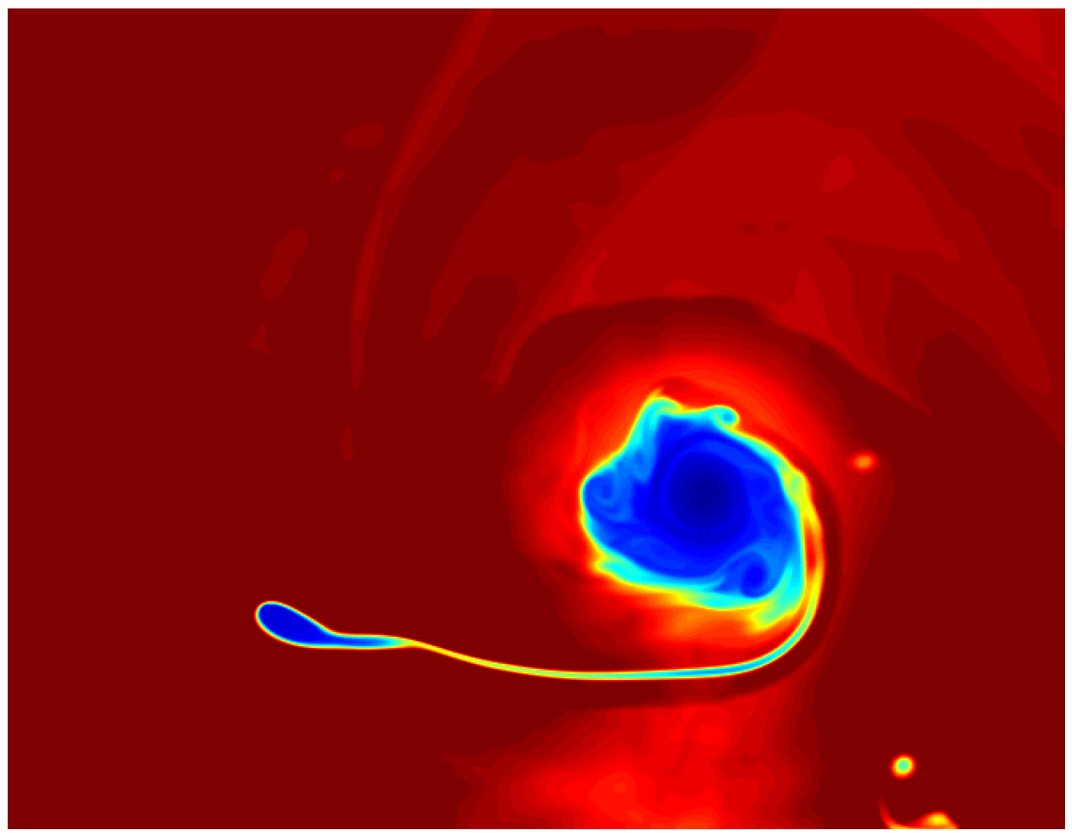
ForestClaw (16×16, $T = 0.1$)

P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
1	4	290	738.2	706.5 (95.7)	31.2 (4.2)	0.0 (0.0)	0.5×10^5
4	5	163	920.4	763.7 (83.0)	48.9 (5.3)	106.9 (11.6)	1.9×10^5
16	6	89	1098.2	796.7 (72.5)	64.9 (5.9)	234.2 (21.3)	6.6×10^5
64	7	49	1456.6	938.6 (64.4)	96.0 (6.6)	416.7 (28.6)	21.2×10^5

ForestClaw (32×32, $T = 0.1$)

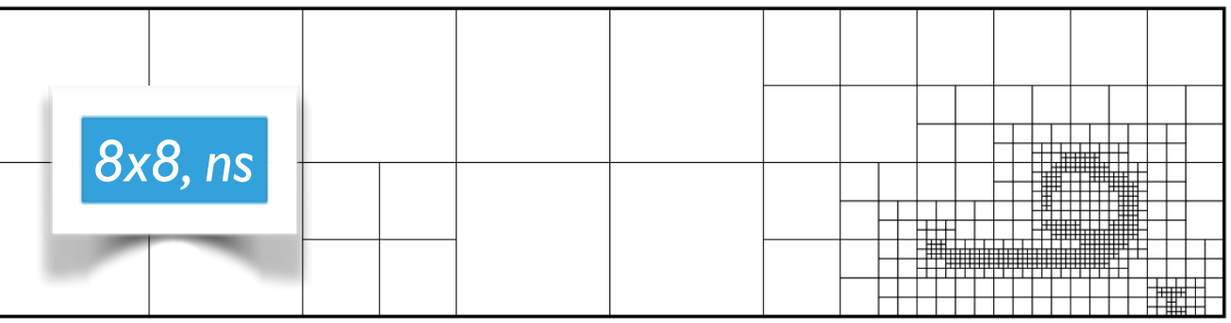
P	ℓ	G	Wall	Adv. (%)	Fill (%)	Comm. (%)	Rate
4	3	27	303.8	242.0 (79.6)	8.9 (2.9)	52.6 (17.3)	2.1×10^5
16	4	18	444.2	307.5 (69.2)	17.8 (4.0)	116.0 (26.1)	7.2×10^5
64	5	10	614.7	366.9 (59.7)	26.9 (4.4)	215.7 (35.1)	22.3×10^5
256	6	5	837.6	377.1 (45.0)	32.6 (3.9)	416.1 (49.7)	67.6×10^5

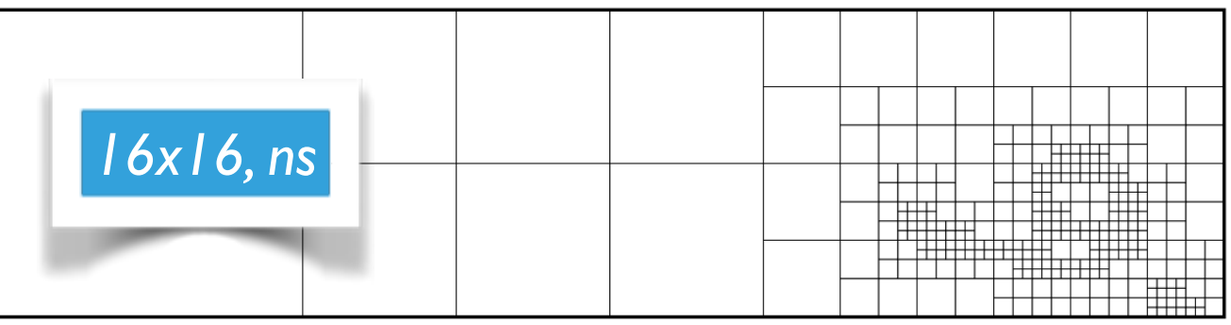
Uniform solution

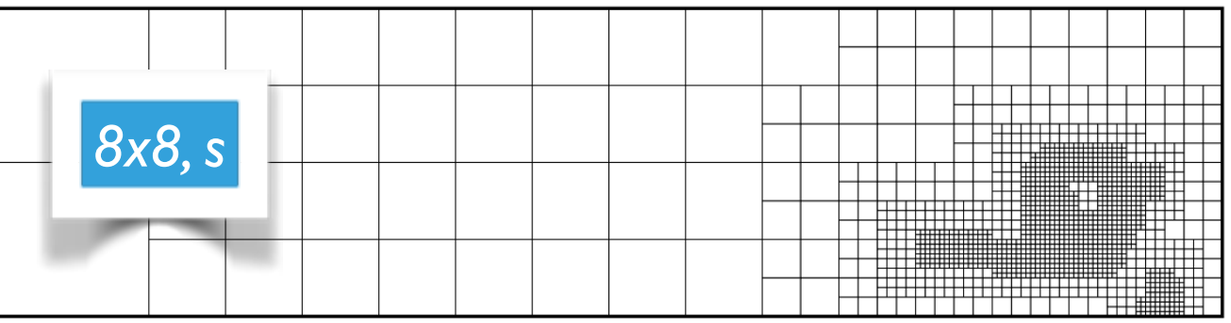


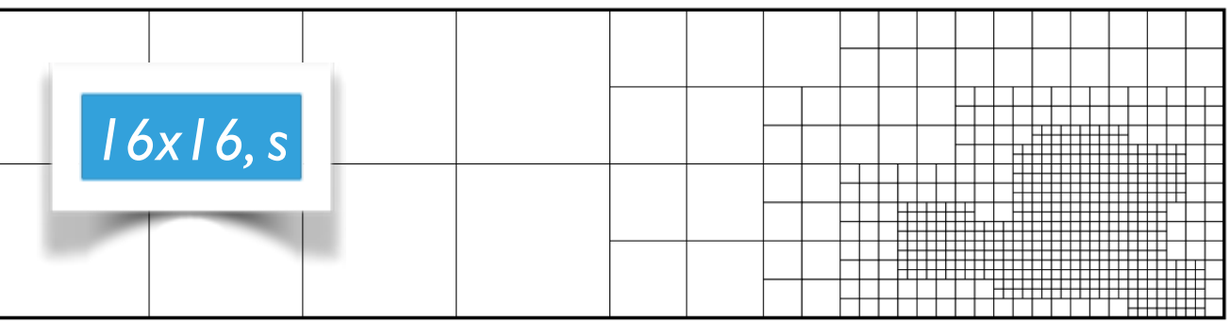
Meshes

ForestClaw

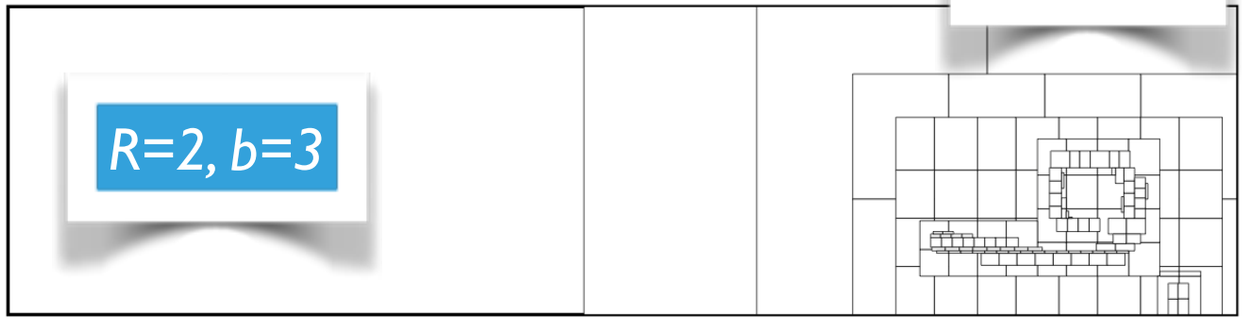


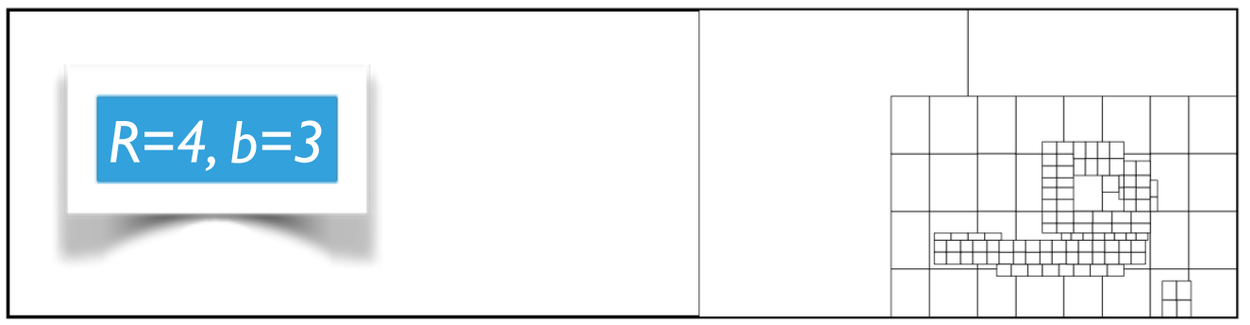


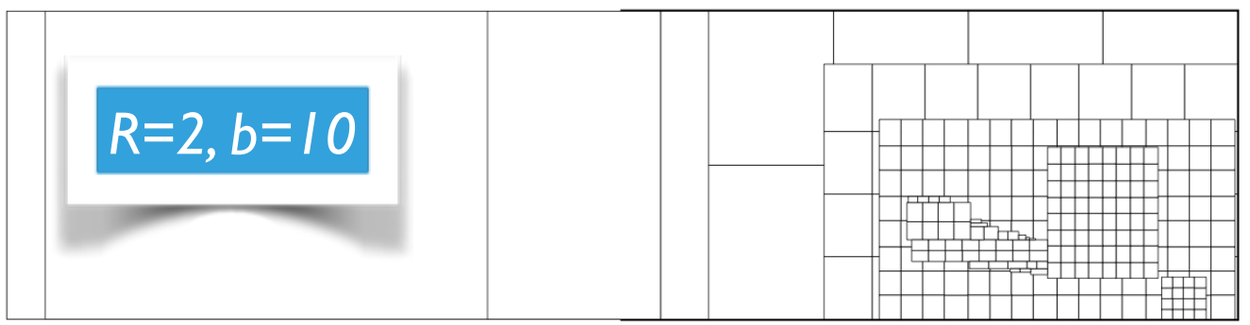




AMRClaw

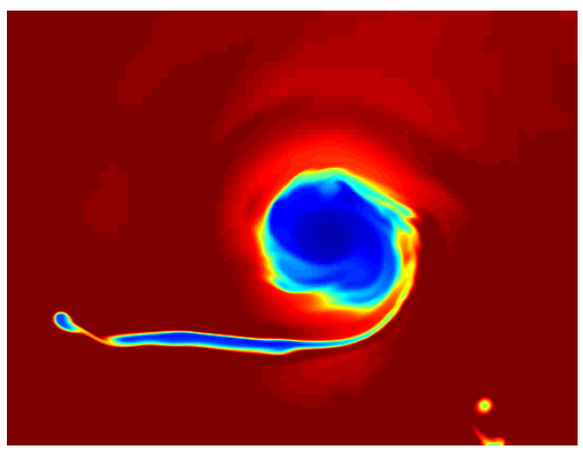




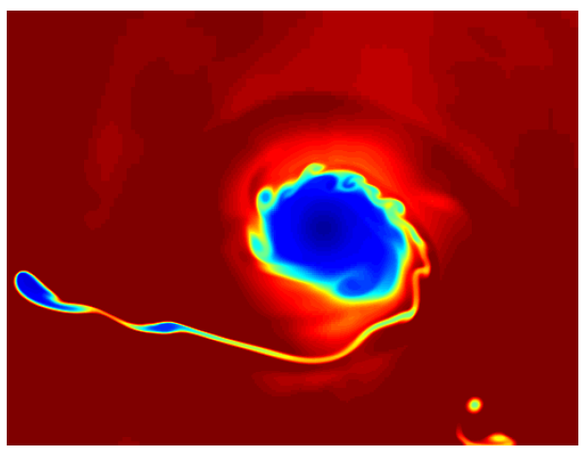


ForestClaw and AMRClaw

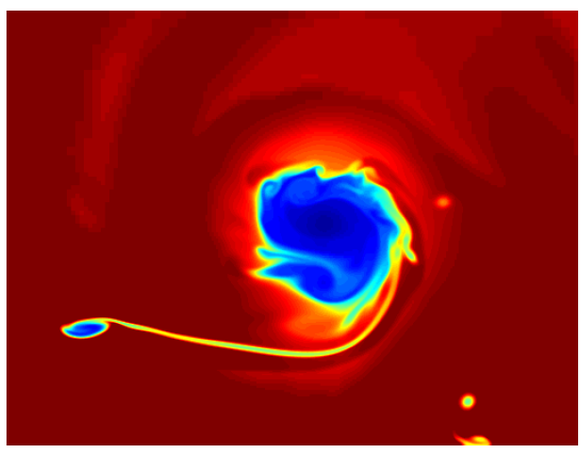
ForestClaw



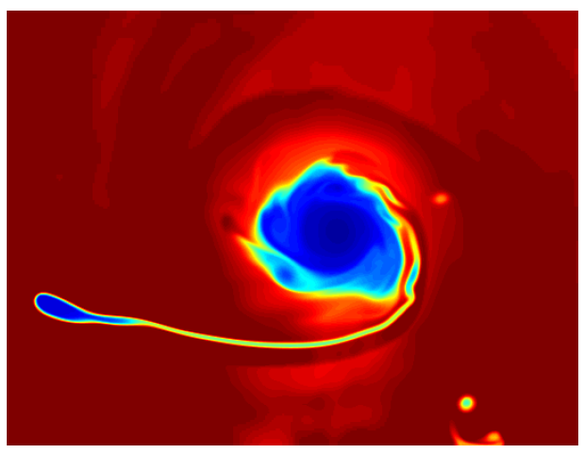
$M = 8,$ non-smooth
$t = 228.9$



$M = 16,$ non-smooth
$t = 304.6$

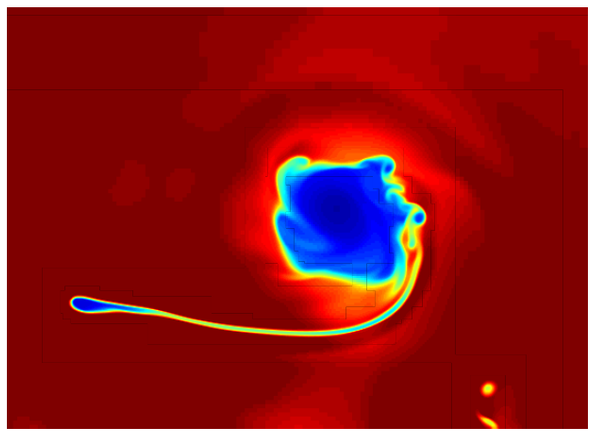


$M = 8,$ smooth
$t = 511.5$

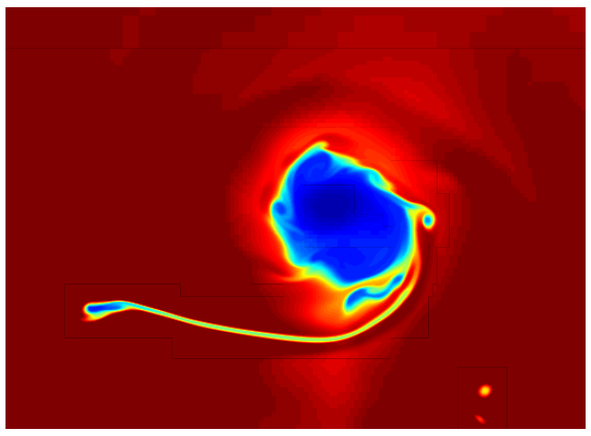


$M = 16,$ smooth
$t = 688.8$

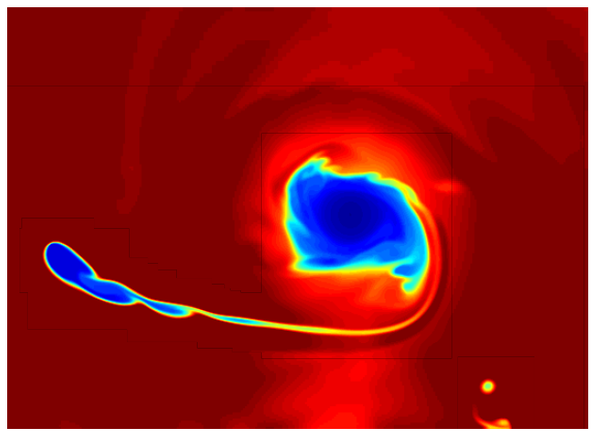
AMRClaw



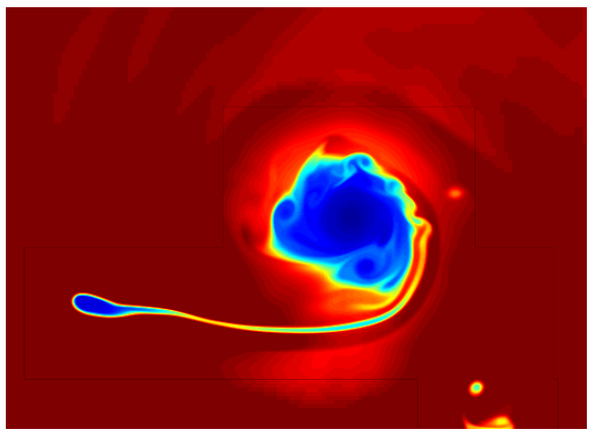
$R = 2,\ b = 3$
$t = 253.9$



$R = 4,\ b = 3$
$t = 293.8$



$R = 2,\ b = 10$
$t = 553.4$



$R = 4,\ b = 10$
$t = 620.2$

Timing results (4-cores)

ForestClaw							
M	S	Wall	Adv. (%)	Fill (%)	Comm. (%)	Regrid (%)	Rate
8	(ns)	228.9	169.8 (74.2)	24.6 (10.8)	33.4 (14.6)	0.2 (0.1)	16.3×10^5
16	(ns)	304.6	232.1 (76.2)	20.4 (6.7)	50.7 (16.6)	0.2 (0.1)	22.0×10^5
8	(s)	511.5	397.1 (77.6)	45.8 (9.0)	66.7 (13.0)	0.3 (0.1)	16.8×10^5
16	(s)	688.8	550.8 (80.0)	37.6 (5.5)	97.8 (14.2)	0.5 (0.1)	21.5×10^5
Uniform		3300.6	2825.2 (85.6)	25.3 (0.8)	246.4 (7.5)	0.0 (0)	30.5×10^5

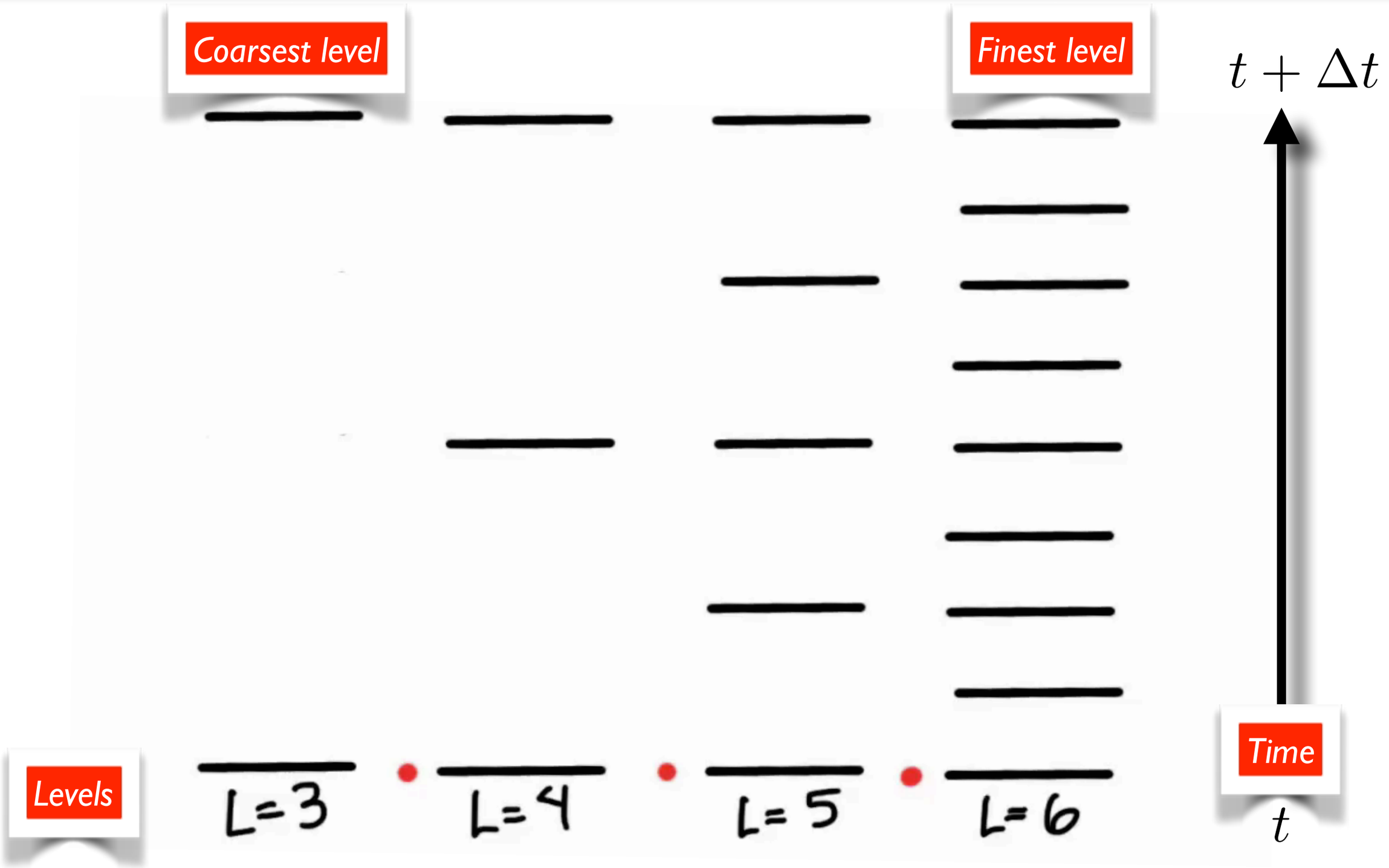
AMRClaw						
R	b	Wall	Adv. (%)	Fill (%)	Regrid (%)	Rate
2	3	253.9	210.1 (82.7)	6.3 (2.5)	33.3 (13.1)	18.4×10^5
4	3	293.8	279.9 (95.3)	4.8 (1.6)	7.3 (2.5)	20.9×10^5
2	10	553.4	467.2 (84.4)	7.3 (1.3)	69.9 (12.6)	19.7×10^5
4	10	620.2	597.0 (96.3)	6.2 (1.0)	13.2 (2.1)	22.9×10^5
Uniform		3716.3	3684.3 (99.1)	3.8 (0.1)	0.0 (0)	27.2×10^5

Rate = number of cells processed per second

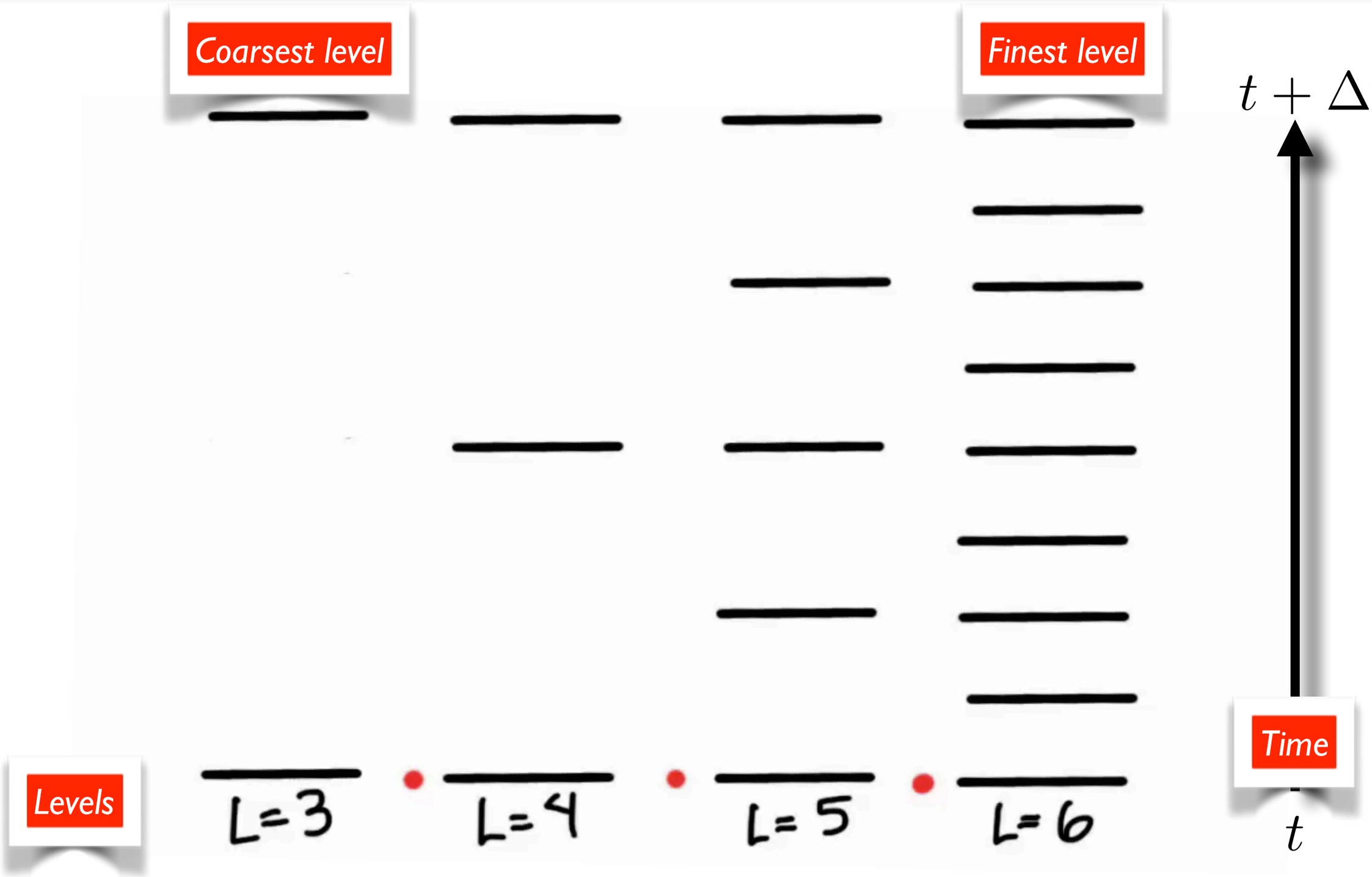
Multirate time stepping

How are ghost cells filled?

Multi-rate time stepping



Parallel multirate time stepping



Multirate algorithm

Require: Grids at all levels at time t must have valid ghost cells values.
  for $k = 1$ to $2^{\ell_{max}-\ell_{min}}$ **do**
    ADVANCE_SOLUTION$(\ell_{max}, (\Delta t)_{\ell_{max}})$ **Advance solution on finest level**
    if multirate **then** **Multirate**
      if $k < 2^{\ell_{max}-\ell_{min}}$ **then**
        Find largest integer $p \geq 0$ such that 2^p divides k.
        $\ell_{time} = \ell_{max} - p - 1$ **Intermediate synchronization**
        UPDATE_GHOST$(\ell_{time} + 1)$
      end if
    else **Global time stepping**
      UPDATE_GHOST(ℓ_{min})
    end if
  end for
  UPDATE_GHOST(ℓ_{min}).

procedure ADVANCE_SOLUTION(level $= \ell$, dt_stable $= \Delta t$)
  for all grids g on level ℓ **do**
    Update solution $Q^{n+1} = Q^n + \Delta t\, F(Q^n, t_n)$.
  end for
  if $\ell > \ell_{min}$ **then**
    if multirate **then**
      if levels ℓ and $\ell - 1$ are time synchronized **then**
        ADVANCE_SOLUTION$(\ell - 1, 2\Delta t)$
        TIME_INTERPOLATE$(\ell - 1, t + 2\Delta t)$
      end if
    else
      ADVANCE_SOLUTION$(\ell - 1, \Delta t)$
    end if
  end if
end procedure

Recursive advance, followed by a time interpolation

Is multirate time stepping worth it?

Assuming an equal number of grids at each level :

$$(1 + 2 + 4 + 8 + \ldots + 2^{\ell_{max} - \ell_{min}}) = (2^{\ell_{max} - \ell_{min} + 1} - 1)$$

verses

$$2^{\ell_{max} - \ell_{min}} (\ell_{max} - \ell_{min} + 1)$$

on a uniformly refined grid, for a speed-up of

$$\frac{\text{multi rate}}{\text{global time step}} = \frac{2}{\ell_{max} - \ell_{max} + 1} - \frac{1}{2^{\ell_{max} - \ell_{min}} (\ell_{max} - \ell_{min} + 1)}$$

$$\approx \frac{2}{\ell_{max} - \ell_{min} + 1}, \qquad \text{for large number of levels}$$

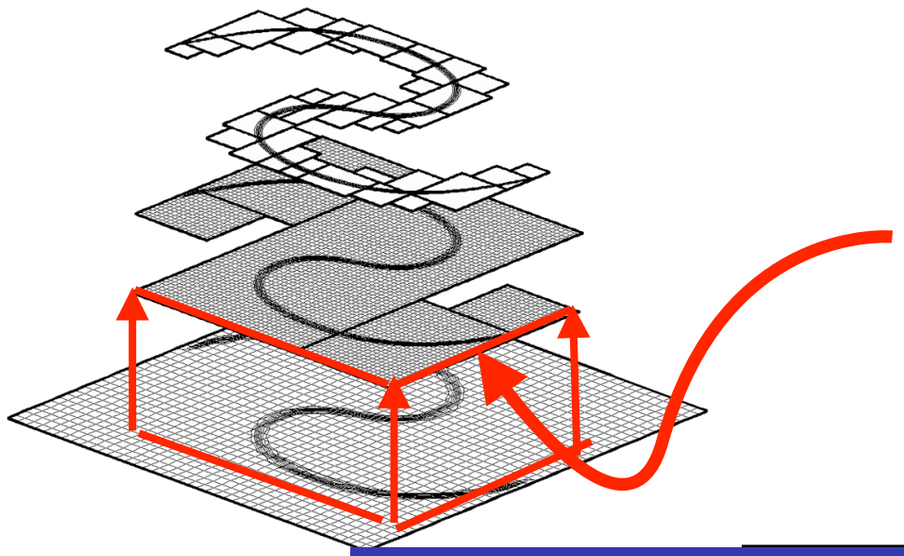
A multi-rate strategy also requires fewer parallel exchanges

Multi-rate time stepping

A single time step advance, assuming a refinement factor of R.

1. Advance at the coarsest level by time step Δt
2. Interpolate coarse grid solution to fine grid ghost cells
3. Advance fine grid R time steps, by a time step $\Delta t/R$
4. Average solution from fine grids to coarse grid,
5. Adjust coarse grid solution to assure flux continuity at the coarse/fine boundaries,
6. Tag cells for refinement and regrid

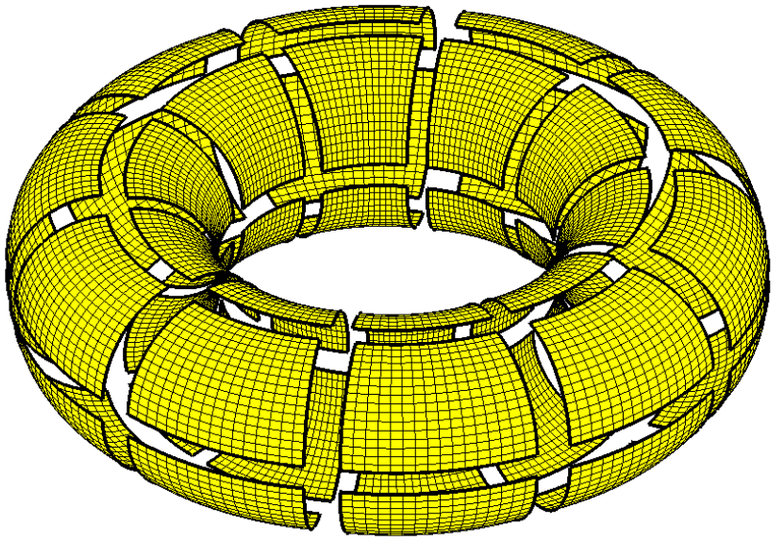
Grids at the same level exchange ghost cell values directly



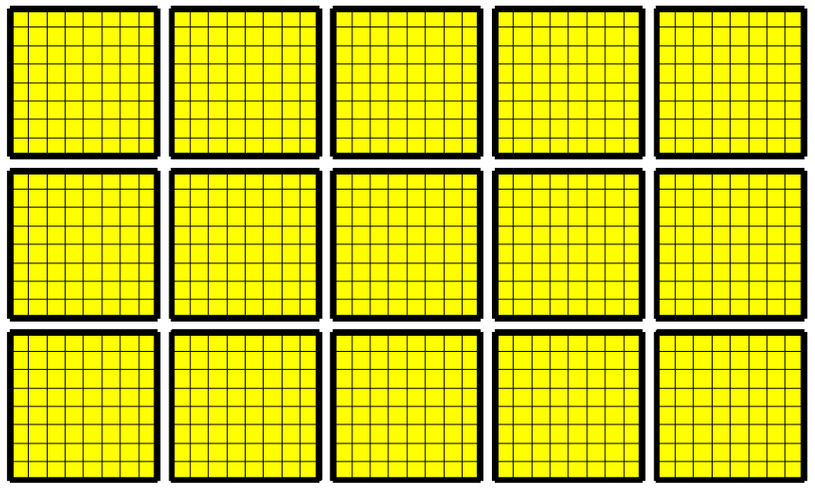
Fine grid boundary conditions interpolated in space and time from coarse grid

Mappings and multiblock transforms

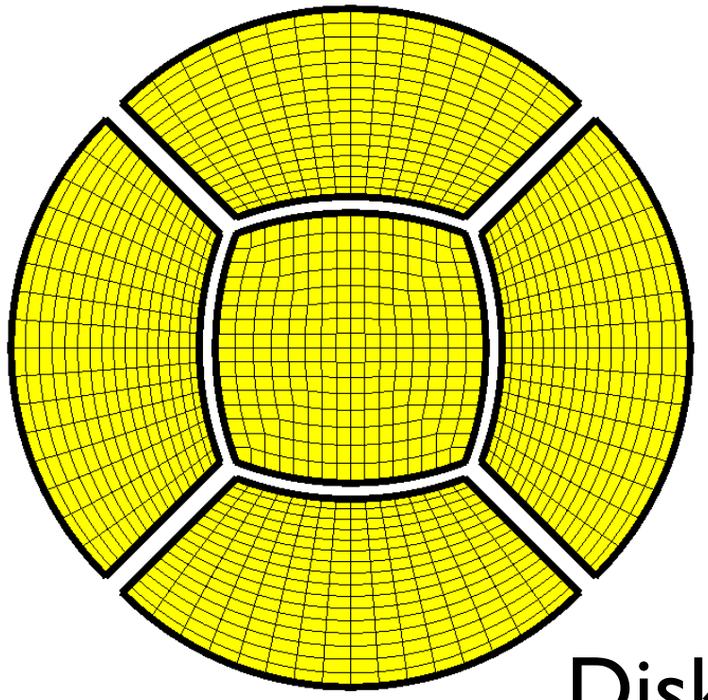
Mapped multi-block domains



Torus

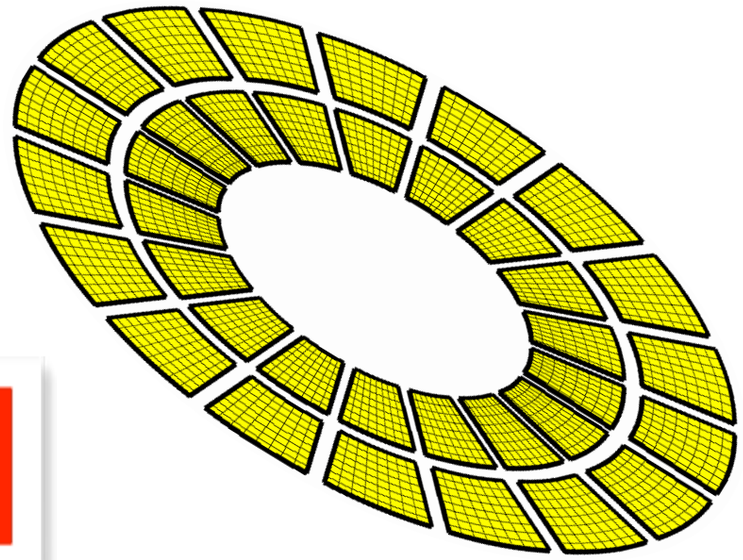


"Brick domain"



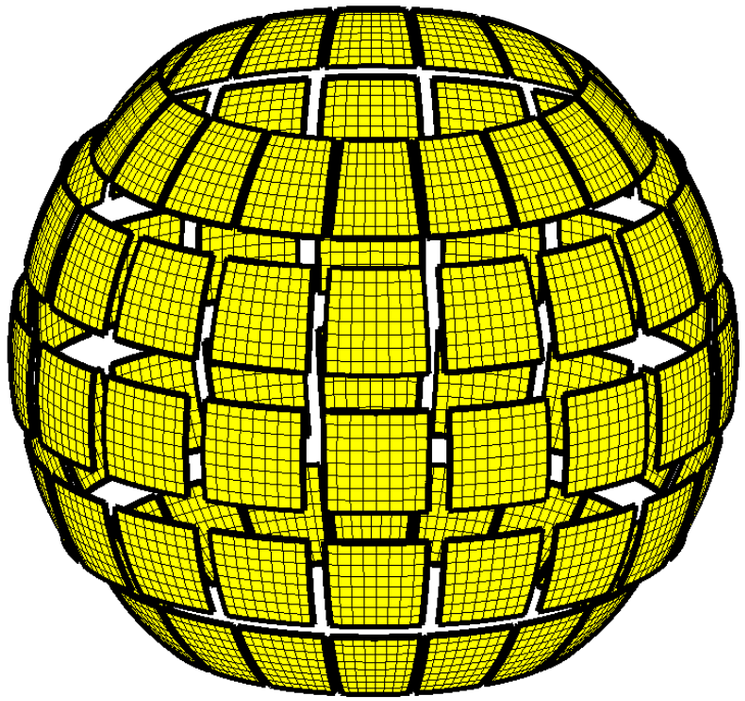
Disk

Each block is a quadtree

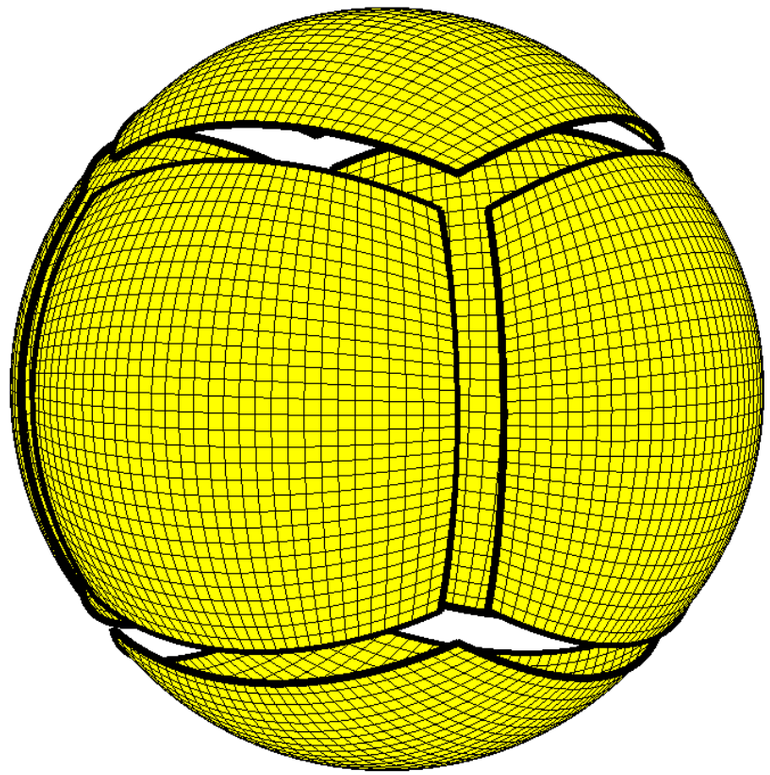


Annulus

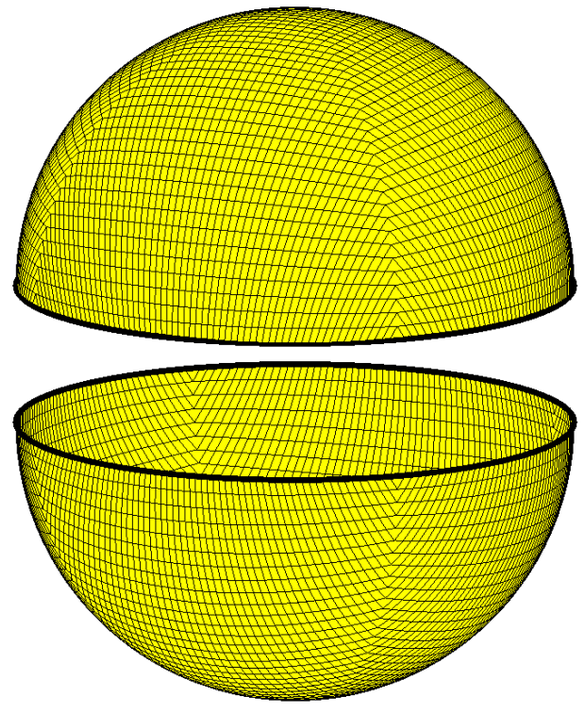
Shallow water equations the sphere



Spherical coordinates



Cubed sphere



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

Each block is a quadtree

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.

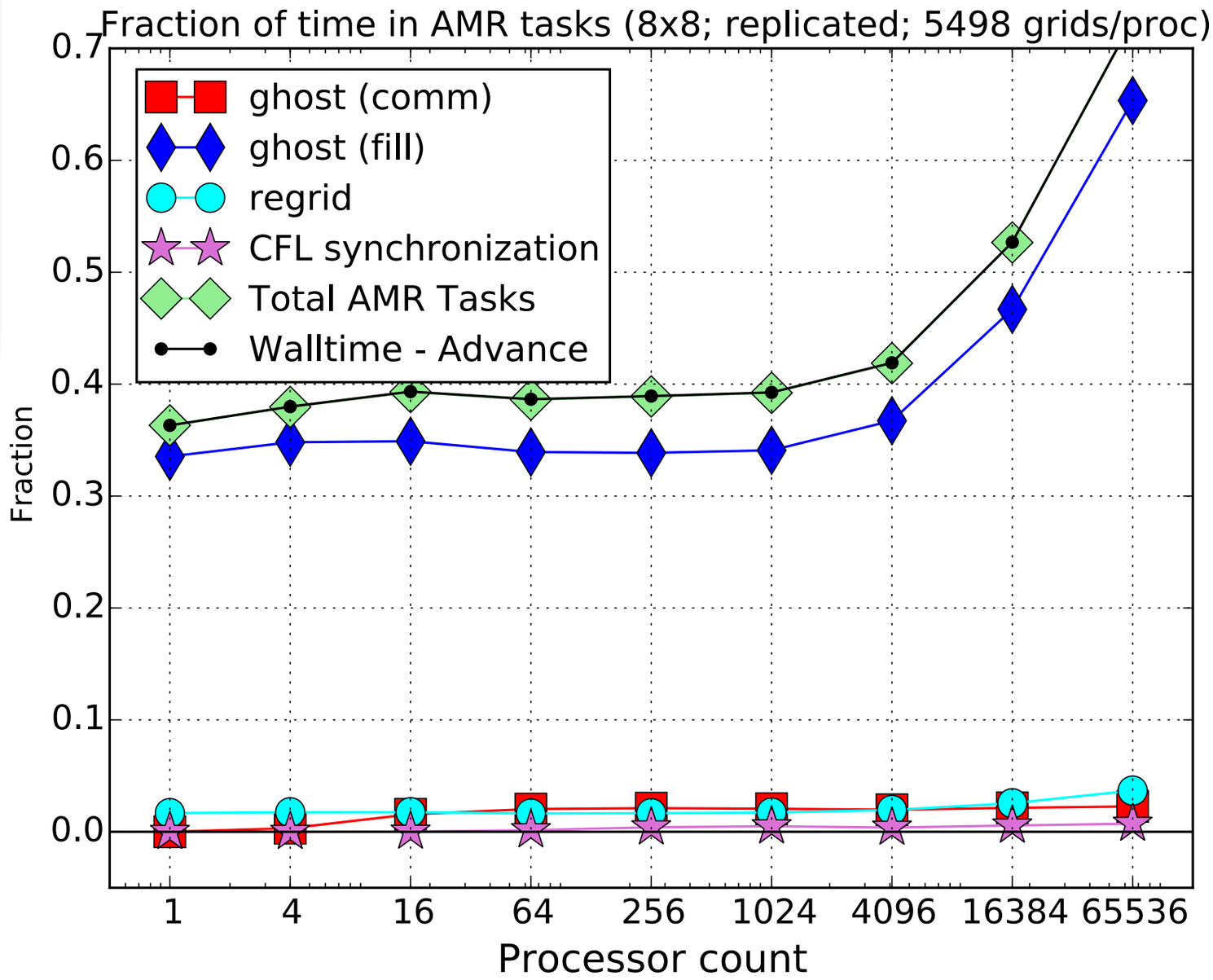
transform_face maps (i1,j1) ghost cell values to neighbor values, using information about block boundary orientations encoded in ***transform_ptr****.*

Backup slides

Cost of AMR

What does AMR cost? (8x8 grids)

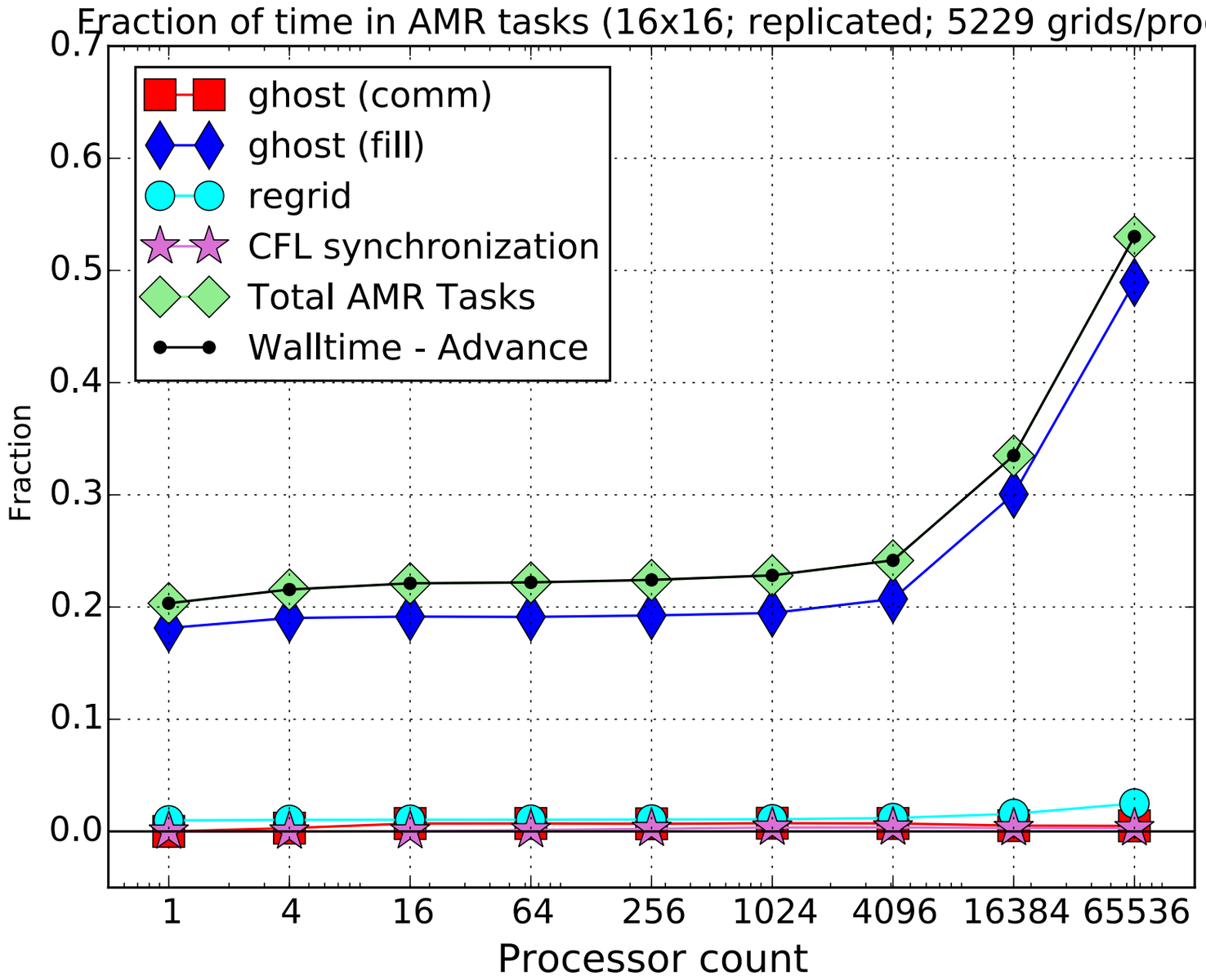
160 total time steps; regridding every coarse grid step (20 regrid steps in total.)



40% in amr related tasks, including communication

What does AMR cost? (16x16 grids)

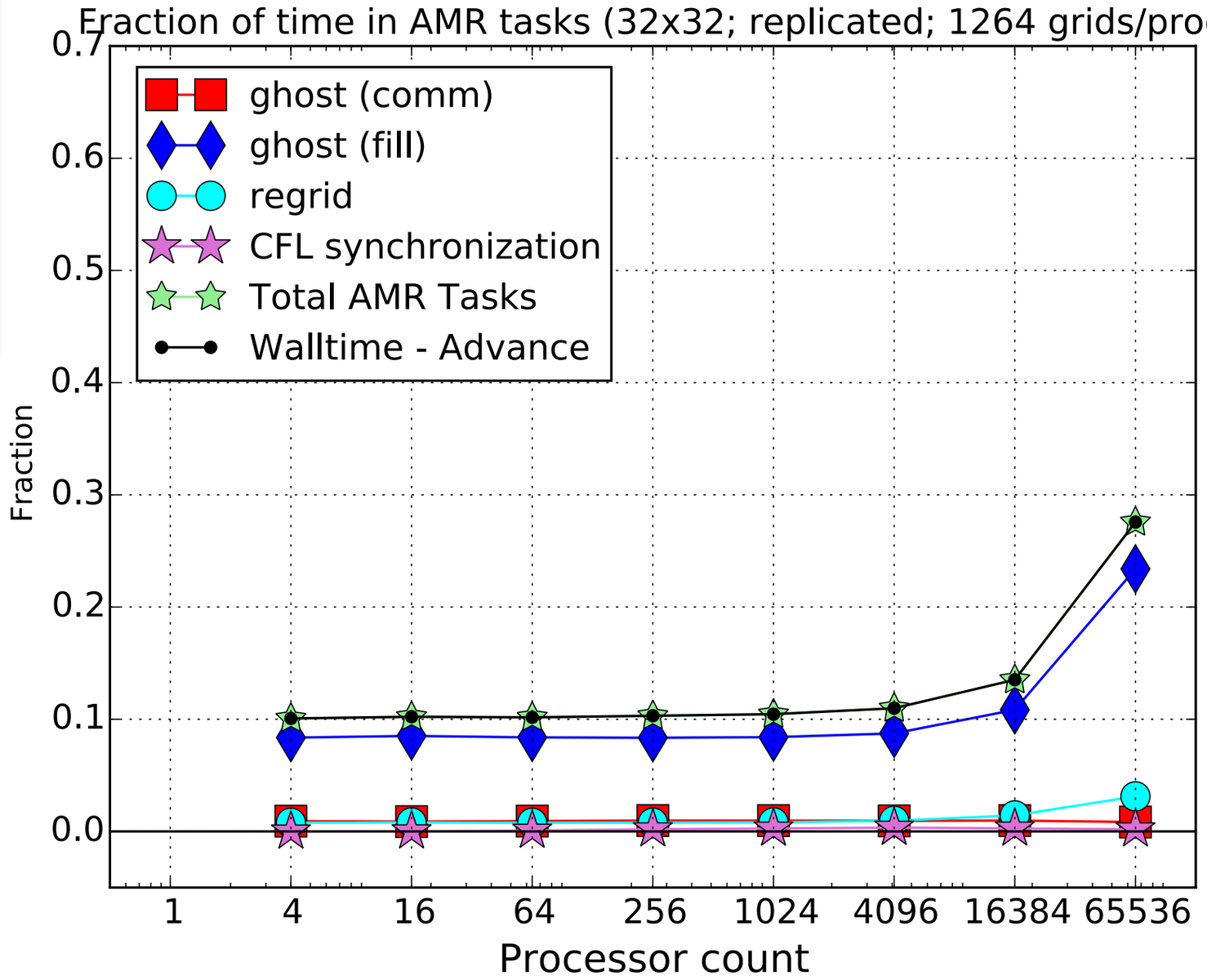
160 total time steps; regridding every coarse grid step (20 regrid steps in total.)



20% in amr related tasks, including communication

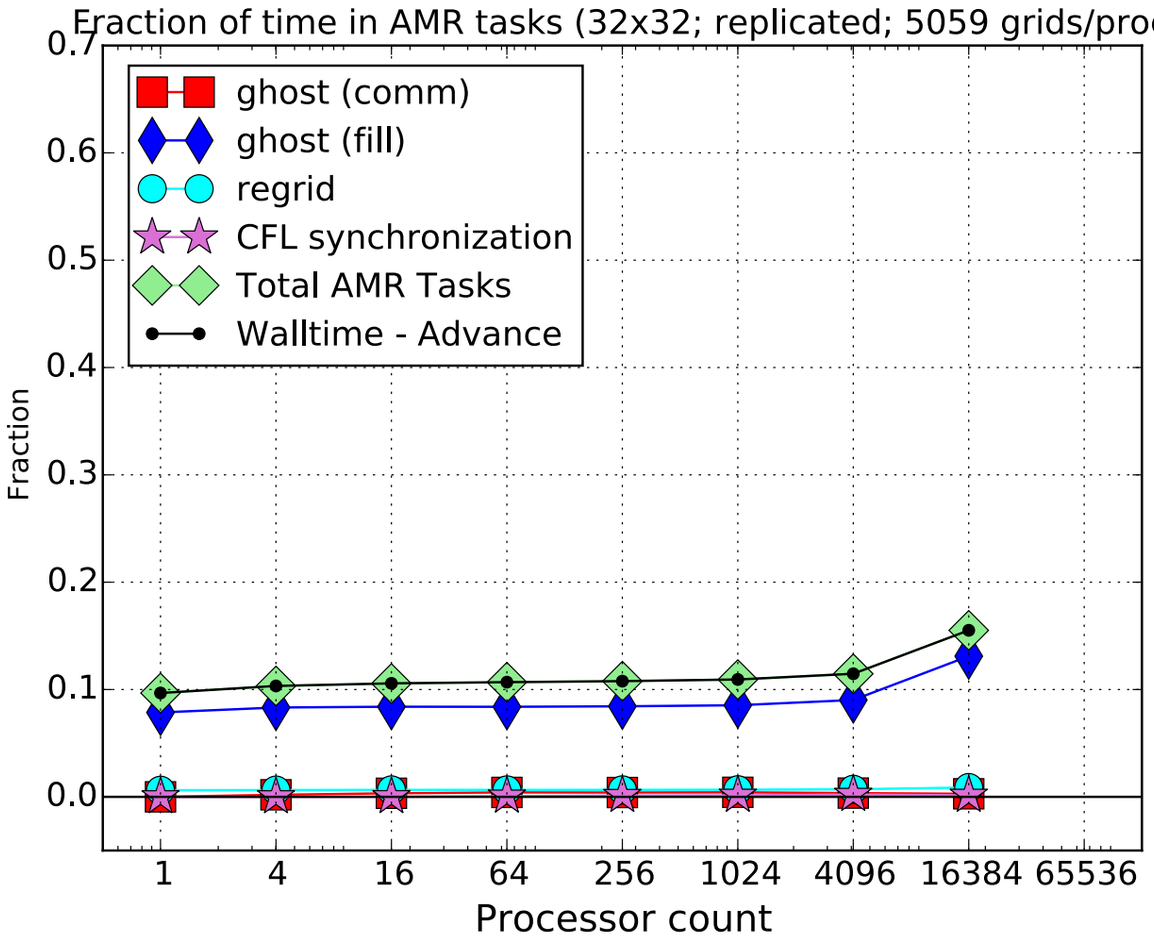
What does AMR cost? (32x32 grids)

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



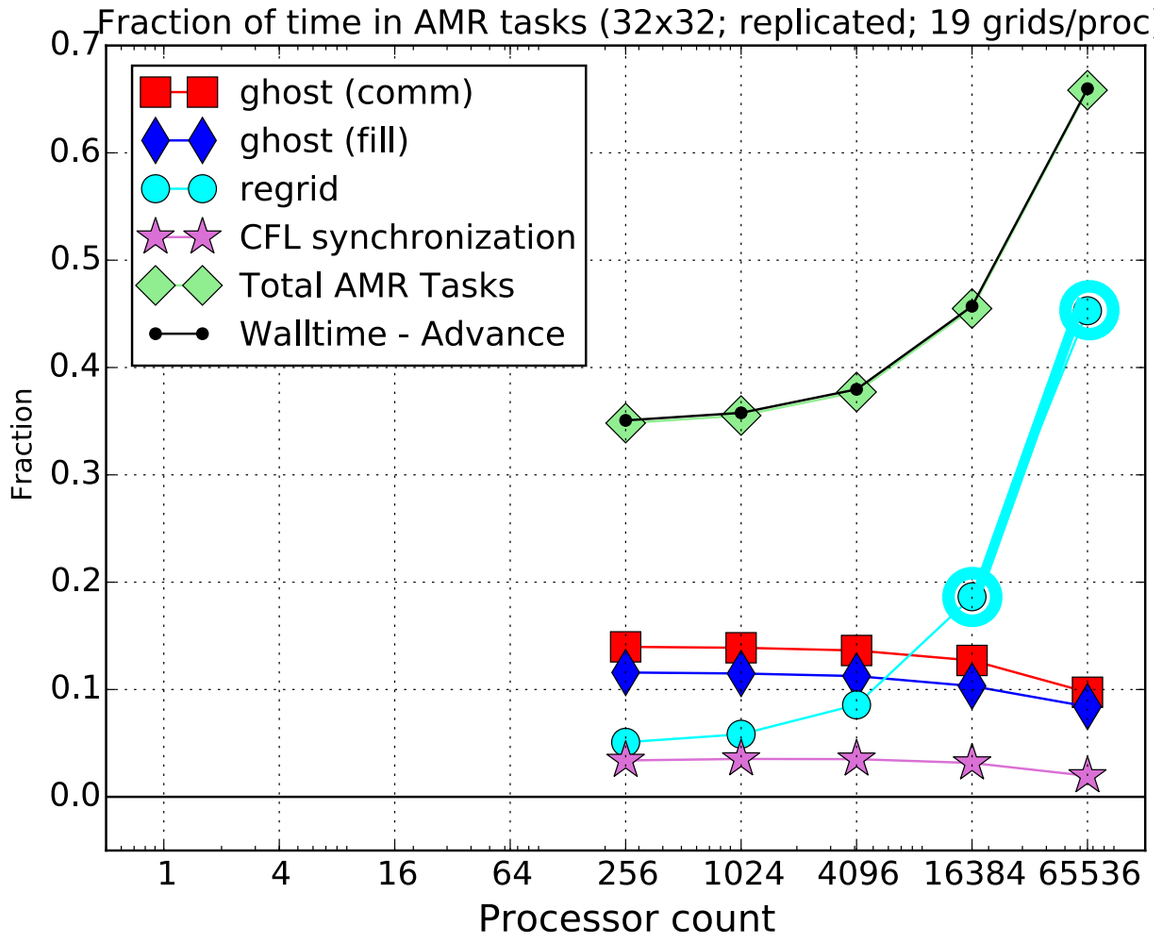
10% in amr related tasks, including communication

What does AMR cost? (32x32 grids)



5059 grids/proc

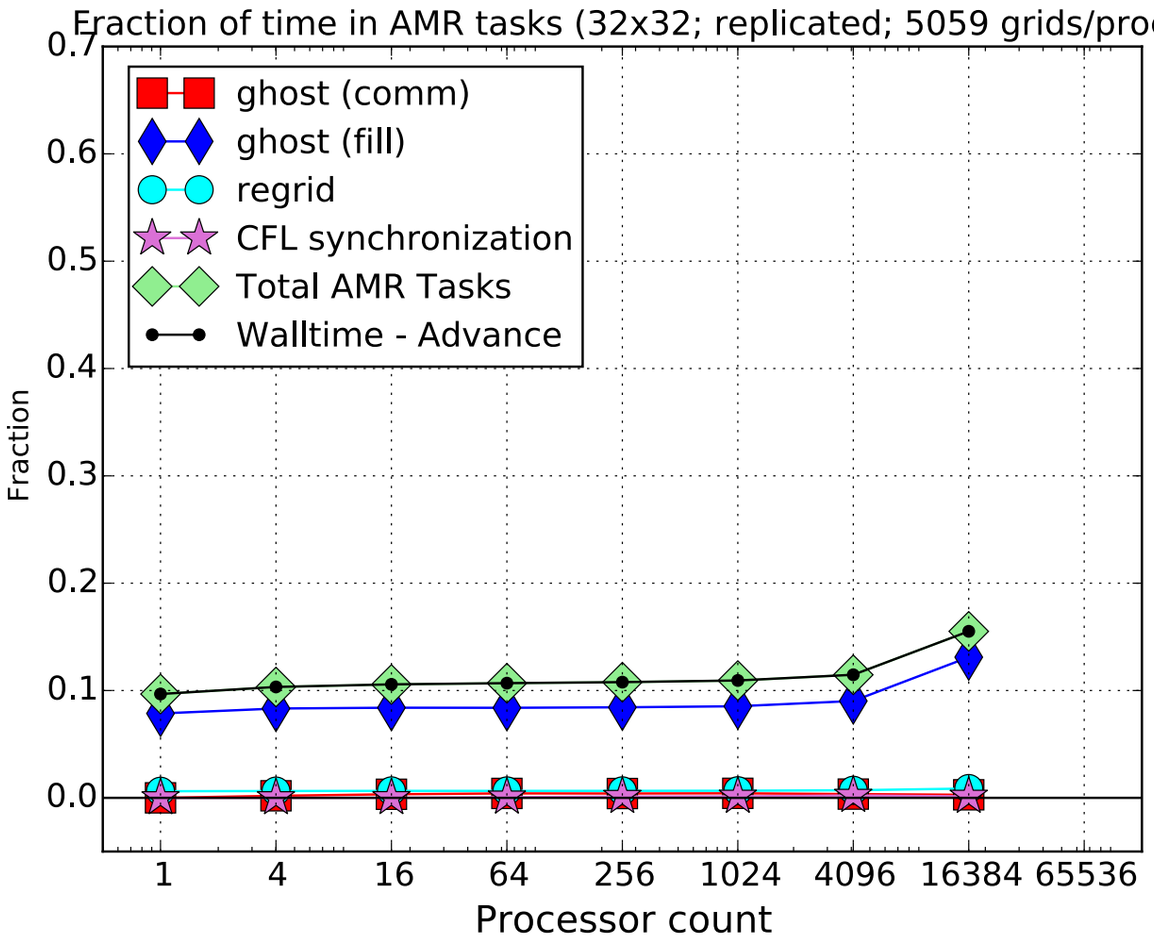
Ghost filling dominates cost



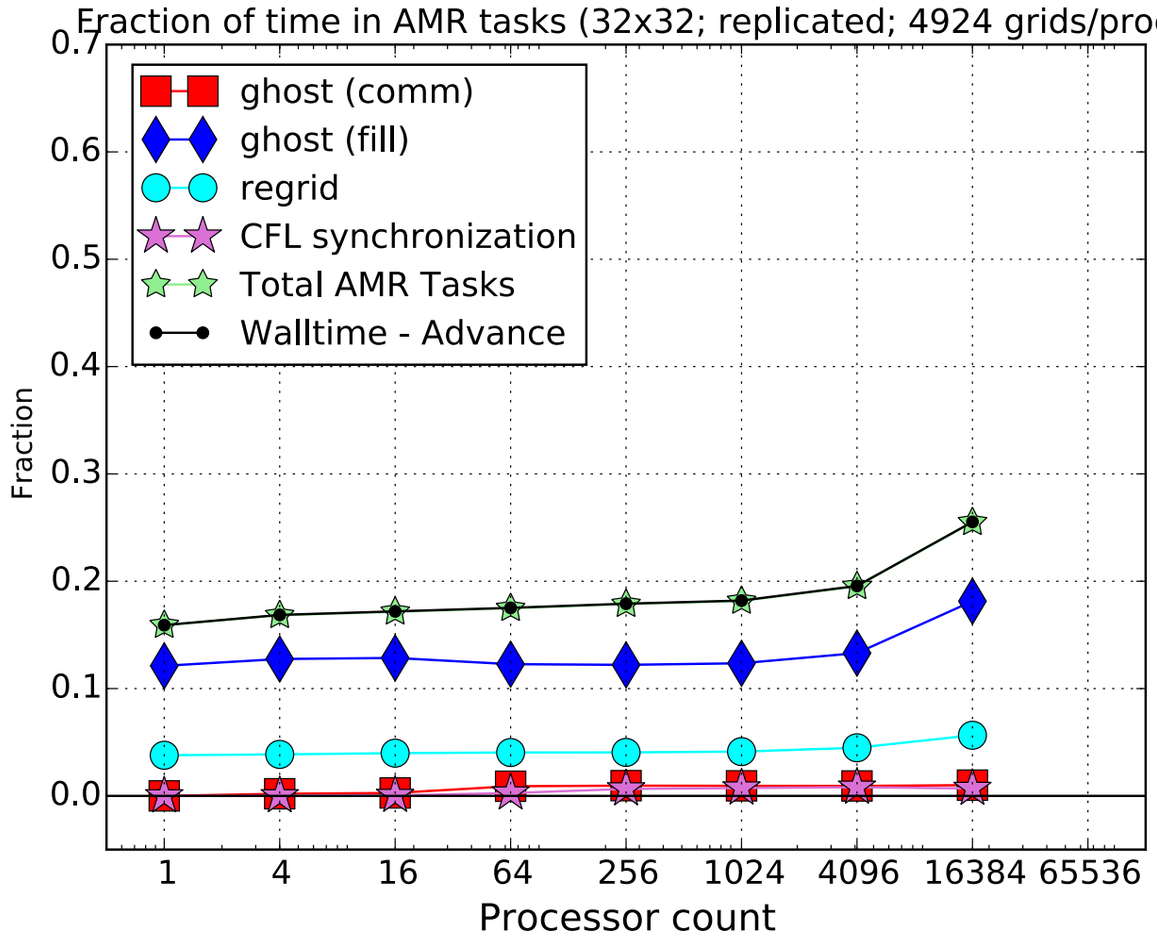
19 grids/proc

Regridding dominates cost

What does AMR cost? (32x32 grids)



Regridding every 8 time steps



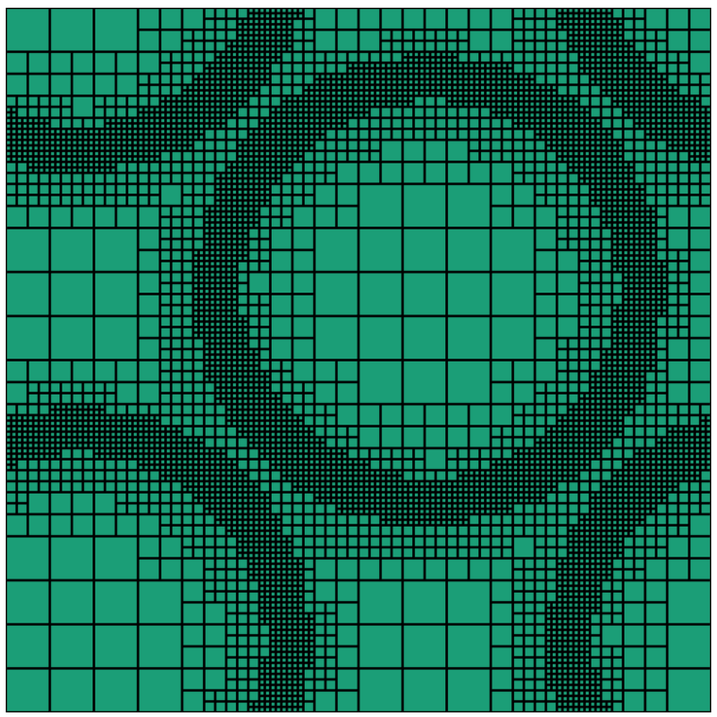
Regridding every time step*

*Initial gridding not included

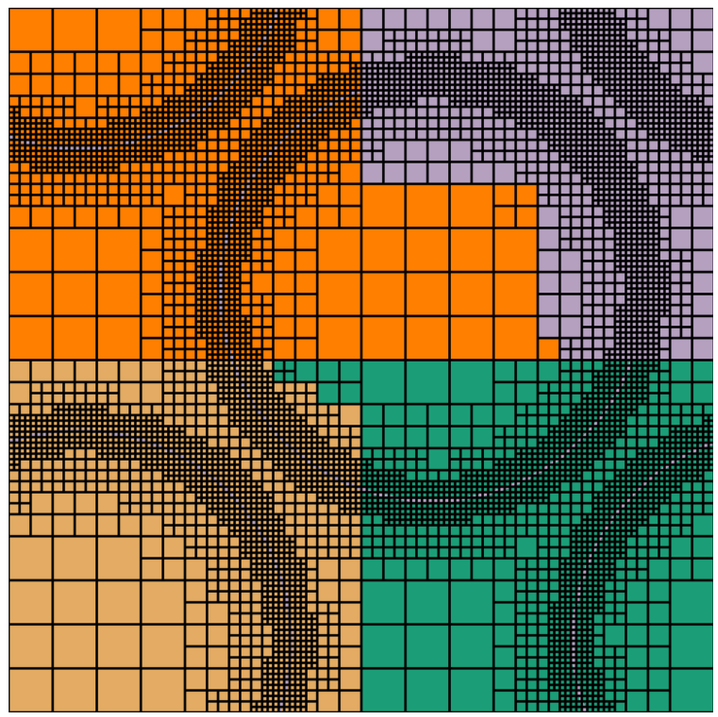
Single adaptive grid scaling

Scaling of single adaptive problem

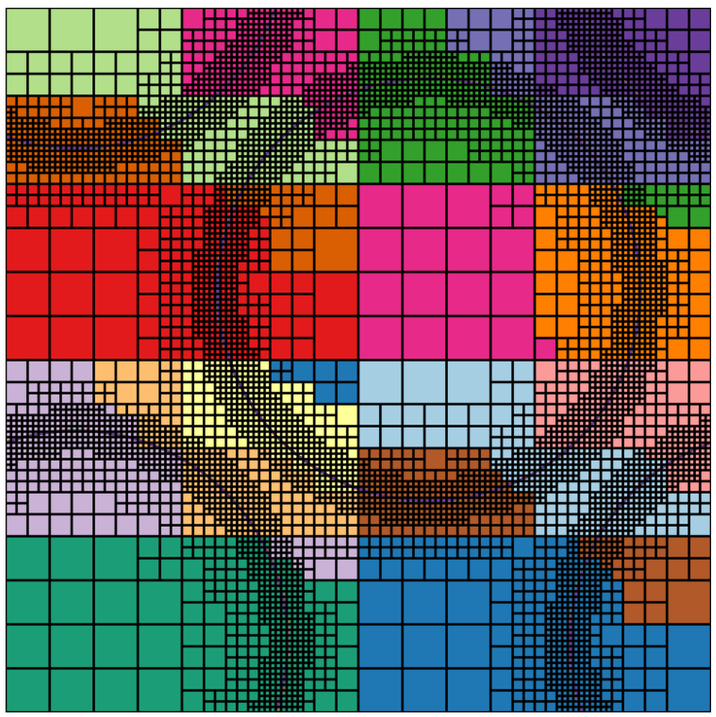
1 proc



4 procs



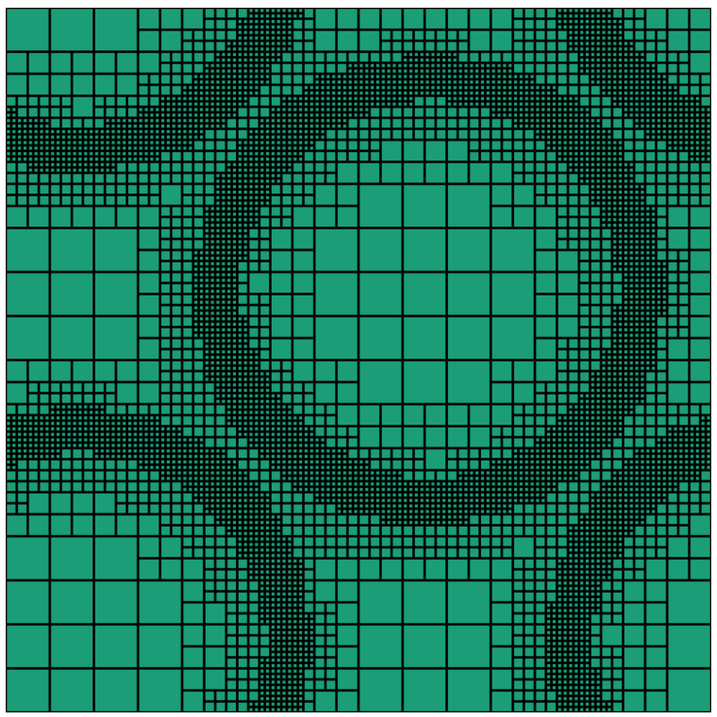
16 procs



…,64, 256, 16384, 65536

Scaling of single adaptive problem

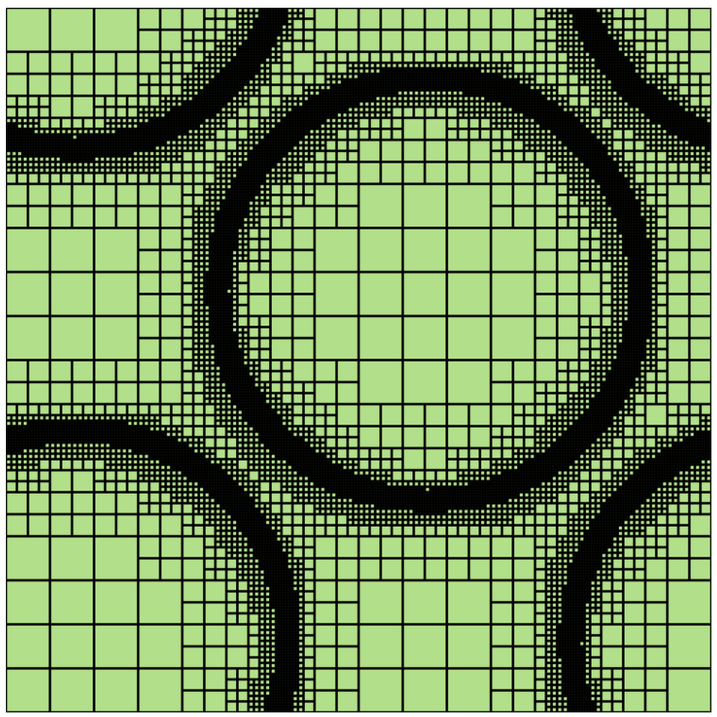
Levels 4-7



level 4 grids

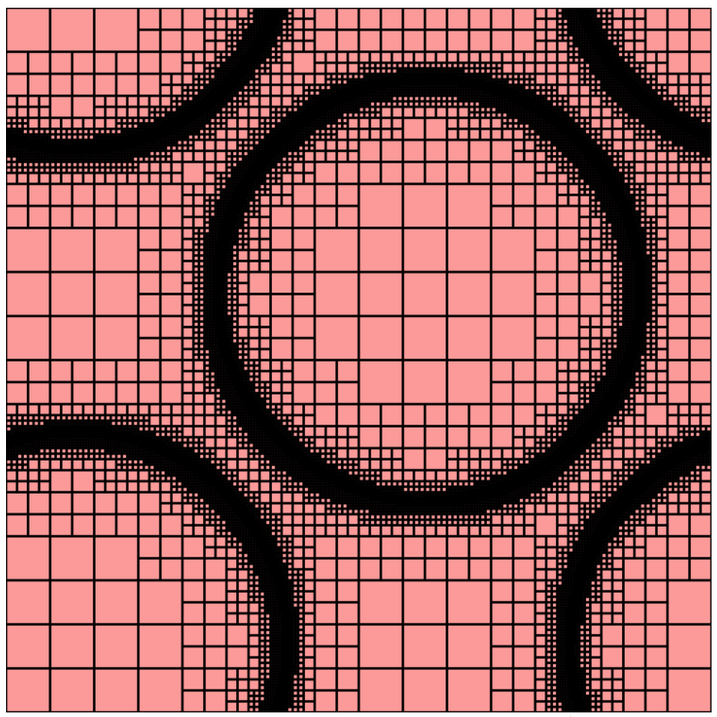
level 7 grids

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

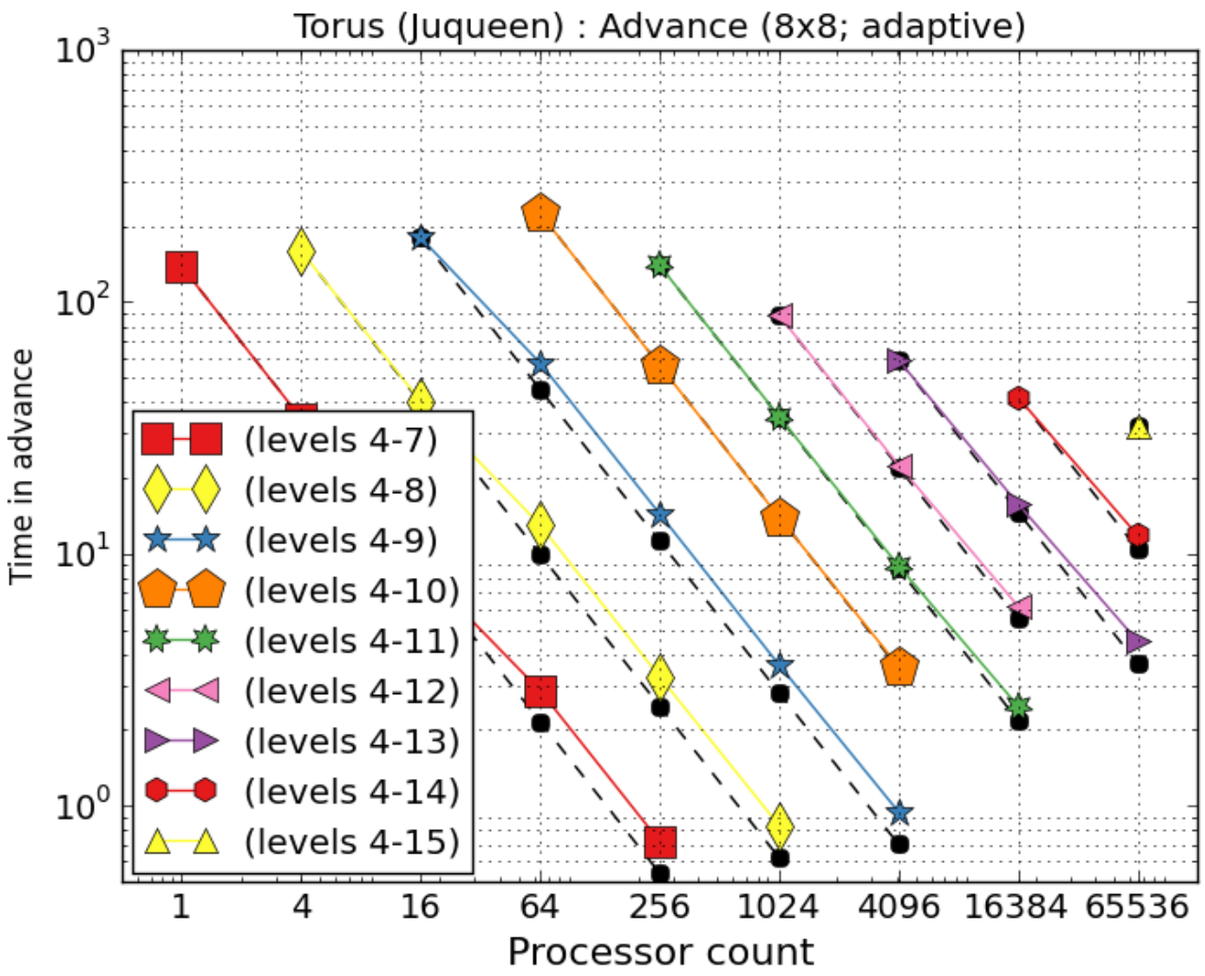
Weak scaling

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

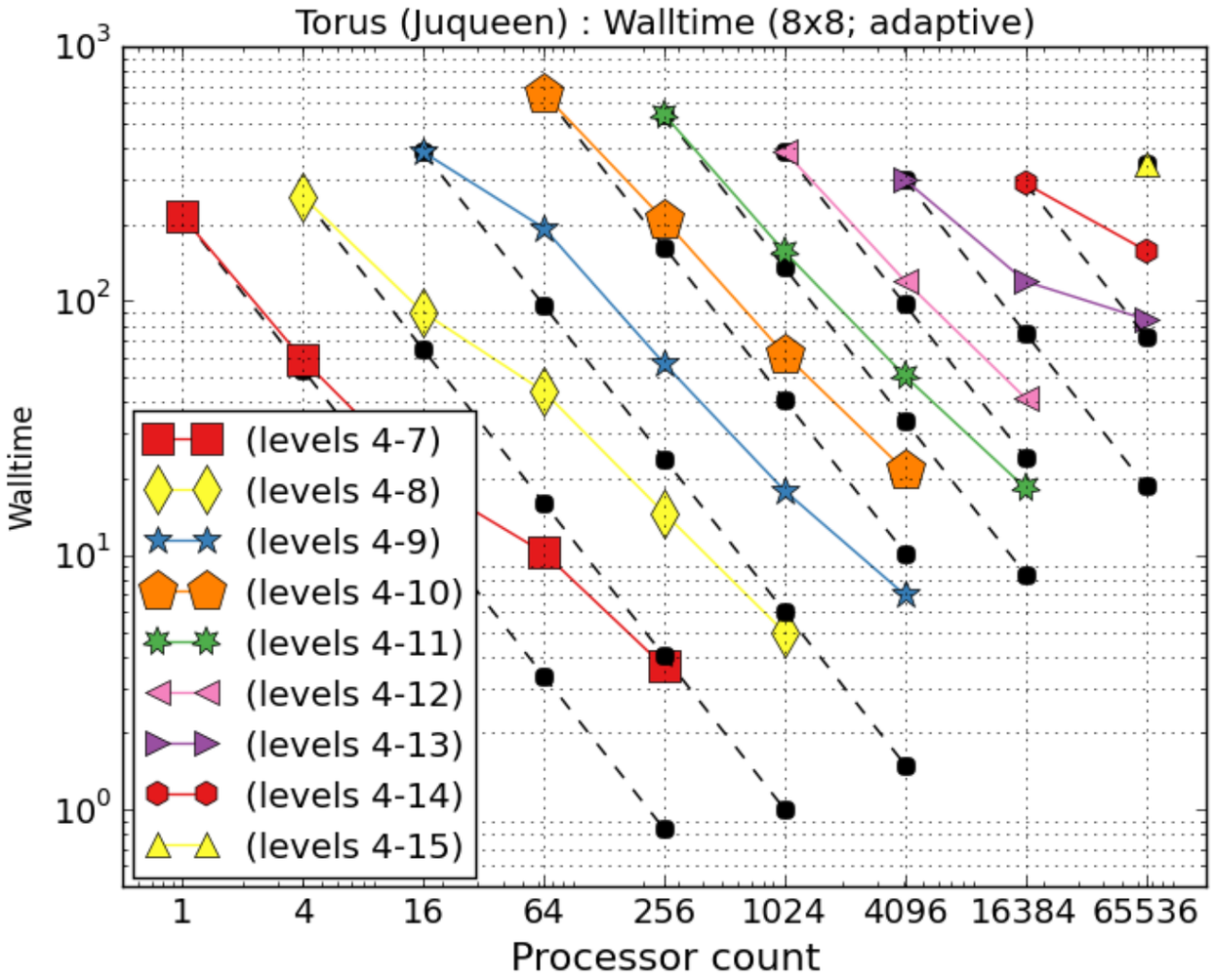
- Fixed number of coarse grid time steps; final time depends on grid size.
- Strong scaling results similar to those for replicated problem.

Strong scaling - adaptive - 8x8

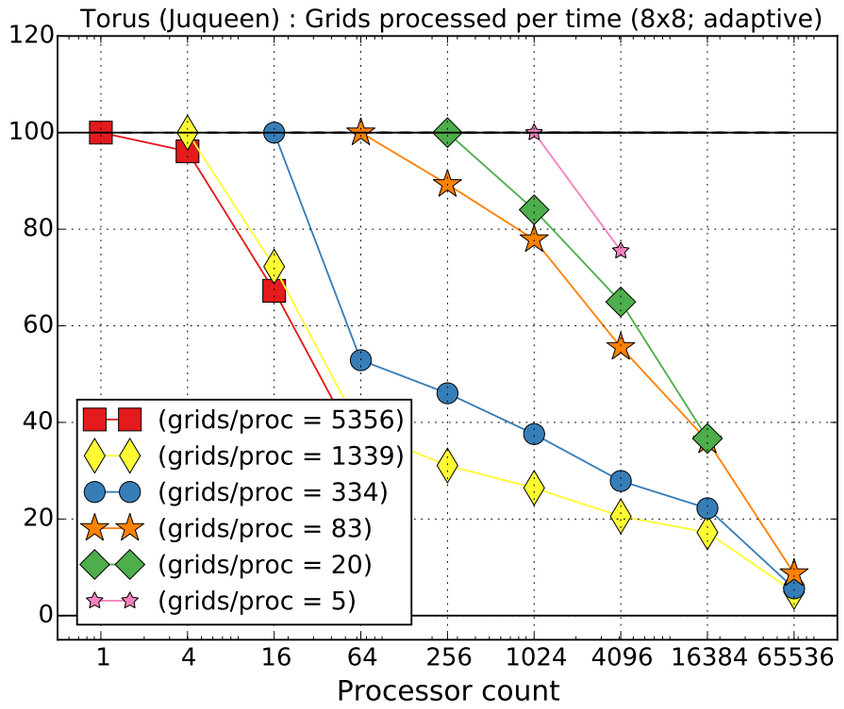
Time in advance



Total walltime



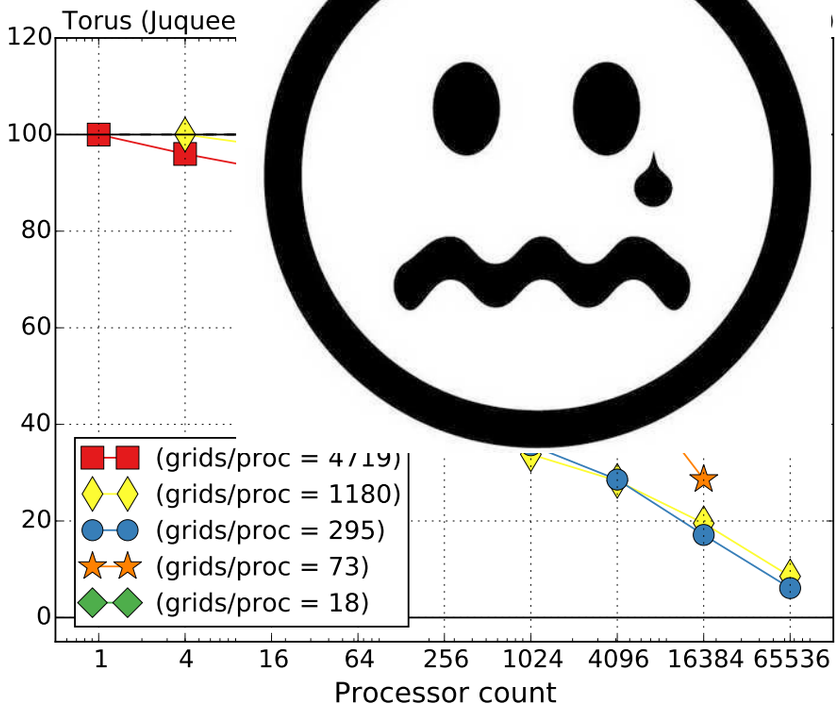
Weak scaling - adaptive



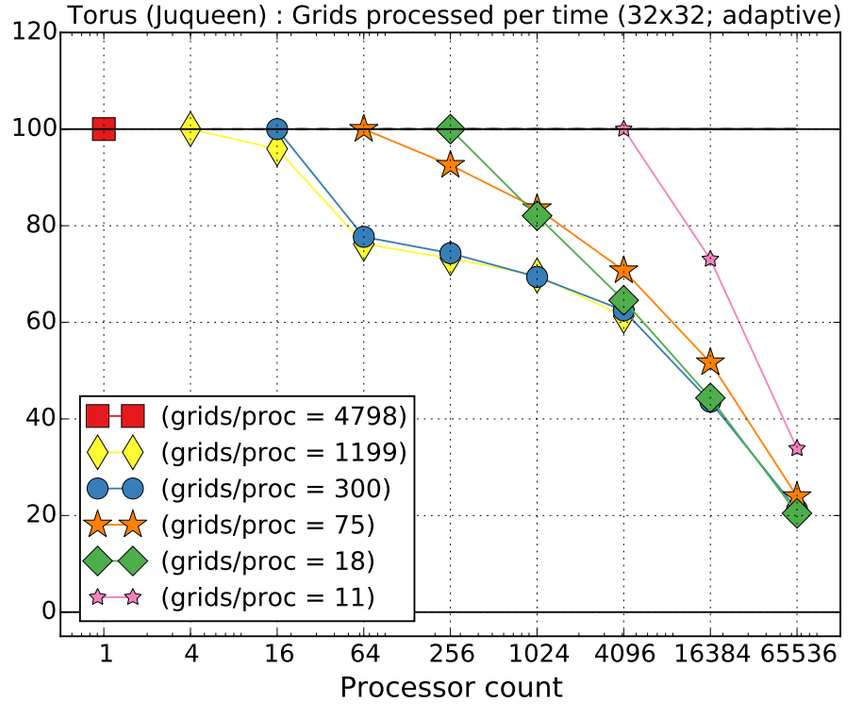
8x8

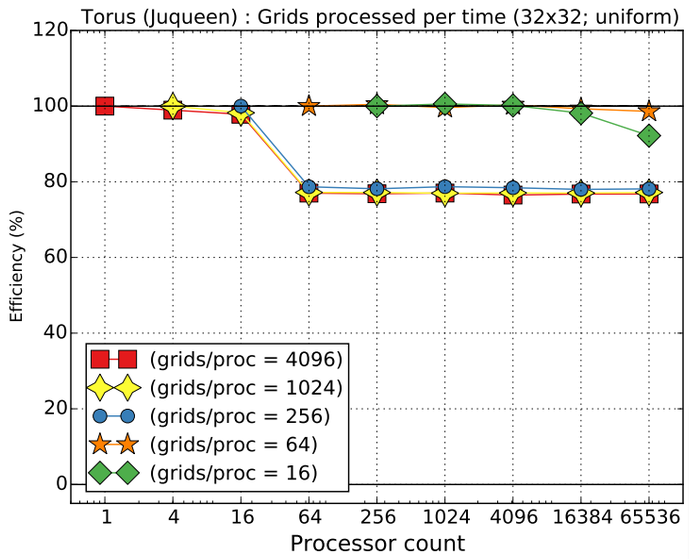
Adaptive (non-replicated) results

16x16



32x32



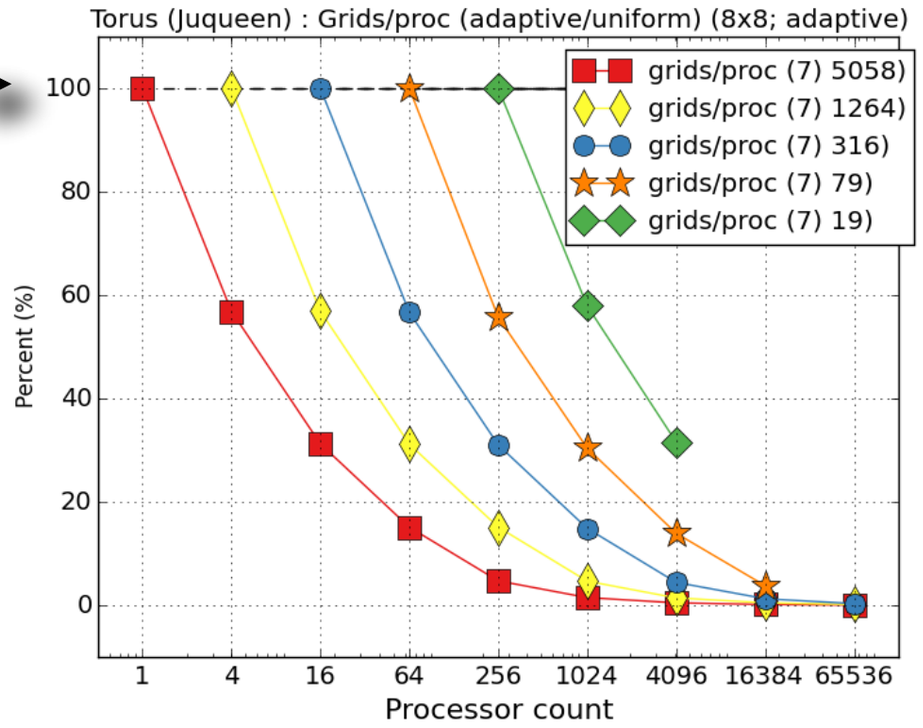


Uniform results

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.

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, ...