Improving the parallel scaling of the block-structured mesh adaptation framework AMROC

Ralf Deiterding

Computer Science and Mathematics Division

Oak Ridge National Laboratory

SIAM Conference on Parallel Data Processing

Seattle, February 24, 2010

In Collaboration with

- Sean Mauch and Daniel Meiron (Computational and Applied Mathematics, California Institute of Technology)
- David Hill, Manuel Lombardini, Dale Pullin (Graduate Aeronautical Laboratories, California Institute of Technology)
- Fehmi Cirak (Mechanical Engineering, University of Cambridge)
- Stuart Laurence (Institut für Aeroelastik, DLR Göttingen)

This work is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy (DOE) and was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. Part of this work was also performed at the California Institute of Technology and was supported by the ASC program of the Department of Energy under subcontract No. B341492 of DOE contract W-7405-ENG-48.

Outline of the talk

- AMROC
  - *Equations solved*
  - *Software overview*
  - *Adaptive mesh refinement*
  - *Embedded boundary method*
  - *Parallelization approach*
  - *Fluid-structure interaction*
  - *Typical example simulations*
- Scalability assessments for O(1000) CPUs
  - *Benchmarking of all algorithmic components*
  - *Elimination of bottlenecks*
    - *Partitioning algorithm*
    - *Global topological operations*
- Conclusions and outlook

Simulation of compressible flows

General convection-diffusion equation

$$\frac{\partial}{\partial t}\mathbf{q} + \sum_{k=1}^{d} \frac{\partial}{\partial x_k}\mathbf{f}(\mathbf{q}) + \sum_{k=1}^{d} \frac{\partial}{\partial x_k}\mathbf{g}(\mathbf{q}, \operatorname{grad} \mathbf{q}) = \mathbf{s}(\mathbf{q})$$

- Convective part $\mathbf{f}(\mathbf{q})$
  - *Conservative schemes with upwinding in all characteristic fields*
  - *Time-explicit treatment with Riemann solvers*
  - *Centered schemes in smooth solutions regions possible*
  - *Explicit method*

$$\mathbf{Q}_j(t_{n+1}) = \mathbf{Q}_j(t_n) - \frac{\Delta t}{\Delta x}\left[\mathbf{F}\left(\mathbf{Q}_j(t_n), \mathbf{Q}_{j+1}(t_n)\right) - \mathbf{F}\left(\mathbf{Q}_{j-1}(t_n), \mathbf{Q}_j(t_n)\right)\right]$$

stable for

$$\Delta t \leq \frac{\Delta x}{S^n_{max}}$$

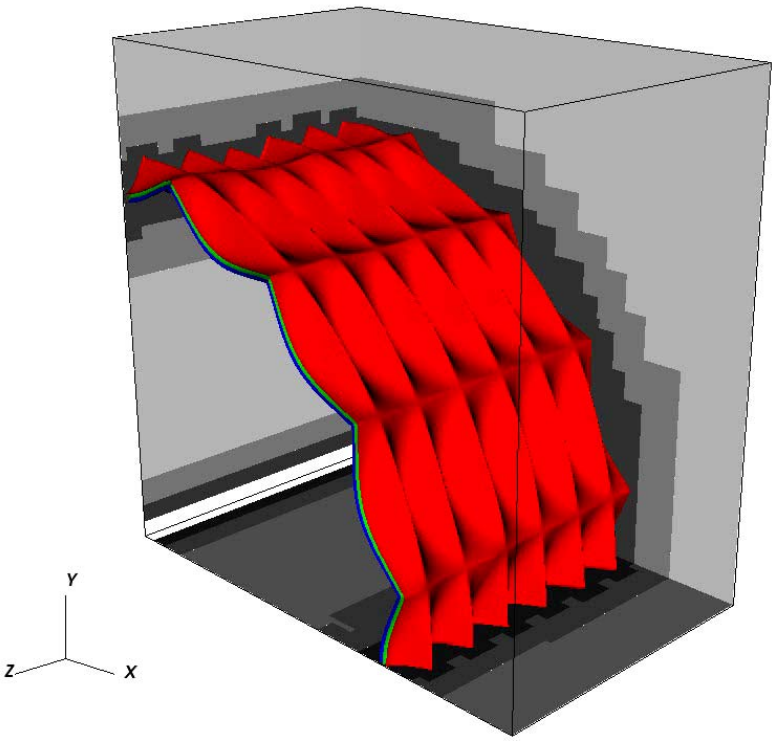
- Diffusion part $\mathbf{g}(\mathbf{q}, \operatorname{grad} \mathbf{q})$
  - *Conservative centered differences*

- Source term $\mathbf{s}(\mathbf{q})$: method of fractional steps

$$\mathcal{H}^{(\Delta t)}: \quad \partial_t \mathbf{q} + \nabla \cdot \mathbf{f}(\mathbf{q}) = 0\,, \quad \text{IC: } \mathbf{Q}(t_n) \overset{\Delta t}{\Longrightarrow} \tilde{\mathbf{Q}}$$

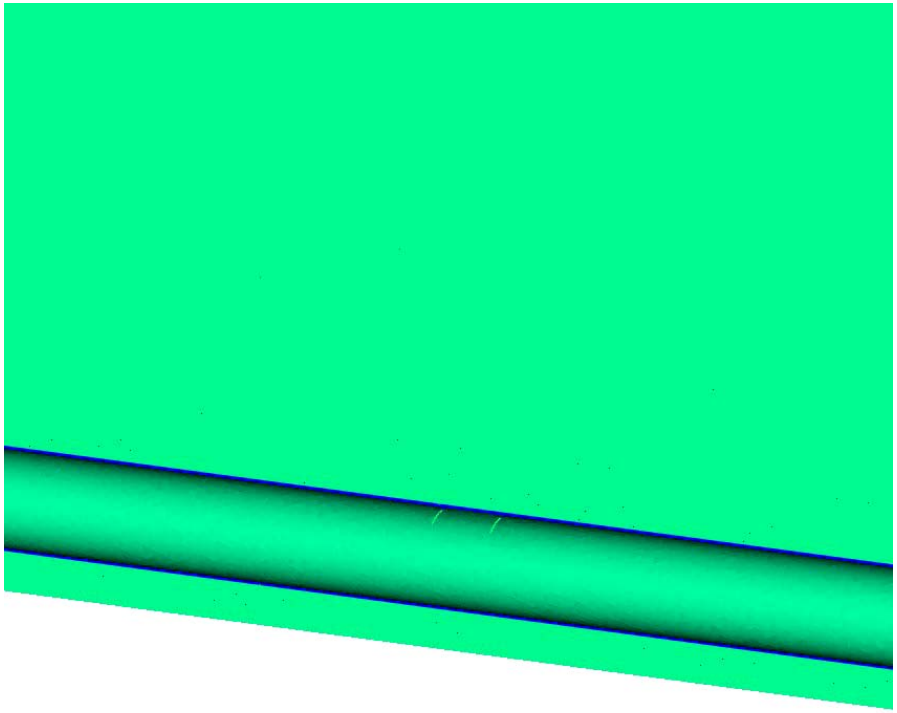
$$\mathcal{S}^{(\Delta t)}: \quad \partial_t \mathbf{q} = \mathbf{s}(\mathbf{q})\,, \quad \text{IC: } \tilde{\mathbf{Q}} \overset{\Delta t}{\Longrightarrow} \mathbf{Q}(t_n + \Delta t)$$

Dynamic meshes – examples



Dynamic adaptive simulation

M. Lombardini, RD, D. Pullin, in J. Meyers et al., *Quality and Reliability of LES*, p. 283-294, Springer, 2008



Fluid-Structure-Interaction

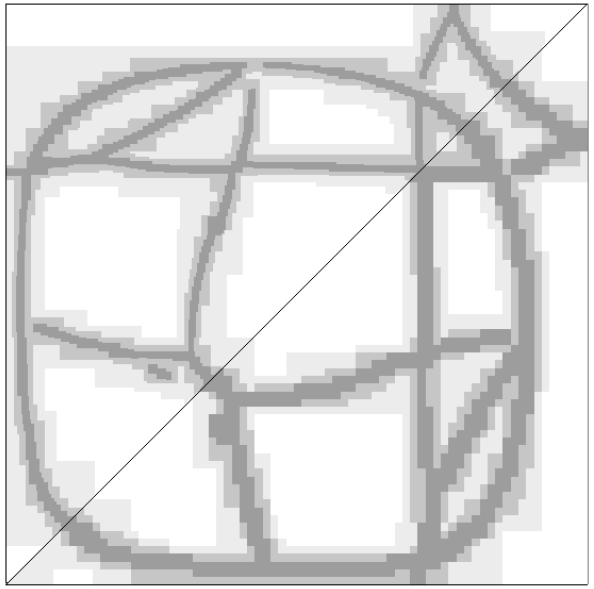
F. Cirak, RD, S. Mauch, Computers & Structures, 85 (11-14): 1049-1065, 2006

AMROC software

- Eulerian fluid solver framework primarily for hyperbolic equations (elliptic equations currently only in serial)
- Large suite of explicit shock-capturing finite volume fluid solvers for inviscid (reactive) flows
  - *Godunov-solvers for thermally perfect gas mixtures (in non-equilibrium) and gas-liquid flows*
  - *Hybridized TCD-WENO solvers (David Hill)*
  - *LES model for compressible flows by Dale Pullin (Caltech)*
- Dynamically adaptive structured Cartesian approach with complex boundary embedding
- MPI Parallelization with dynamic load balancing
- Parallel fluid-structure interaction coupling capability (the *Virtual Test Facility*)
  - *Interfaces to several explicit structure mechanics codes, e.g. LLNL Dyna3d*
- Used since ‘03 by routine at the DOE ASC Center for Simulation of Dynamic Response of Materials at Caltech
  - *20-30 papers indexed by ISI Web of Science*
  - *4 PhD theses completed, 3 PhD theses ongoing*
  - *AMROC V1.0 at* http://amroc.sourgeforge.net
  - *V2.0 within VTF 1.0 at http://www.cacr.caltech.edu/asc*

AMROC

- ~140,000 LOC C++, C, Fortran-77
- AMROC uses hierarchical data structures that have evolved from DAGH by M. Parashar and J.C. Browne to implement general Berger-Collela AMR
- Point explosion in box, 4 level Euler computation, 7 compute nodes



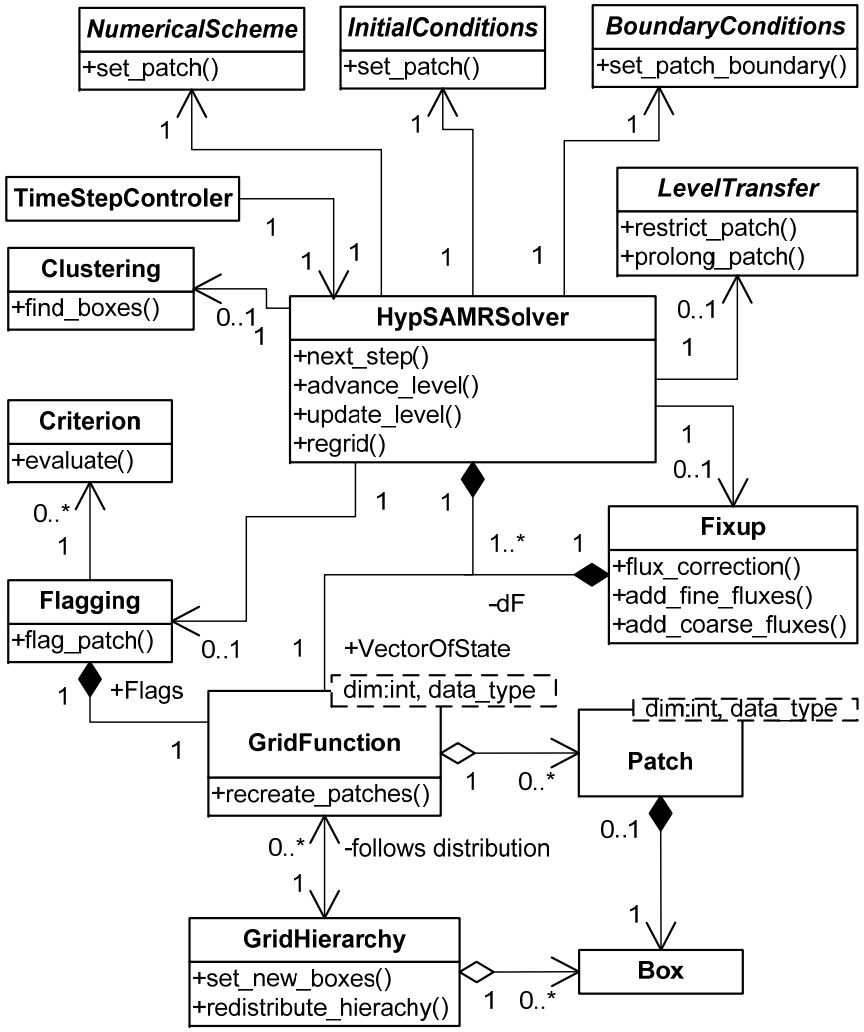
	l_{max}	Level 0	Level 1	Level 2	Level 3	Level 4
AMROC's DAGH grids/cells	1	43/22500	145/38696			
	2	42/22500	110/48708	283/83688		
	3	36/22500	78/54796	245/109476	582/165784	
	4	41/22500	88/56404	233/123756	476/220540	1017/294828
Original DAGH grids/cells	1	238/22500	125/41312			
	2	494/22500	435/48832	190/105216		
	3	695/22500	650/55088	462/133696	185/297984	
	4	875/22500	822/57296	677/149952	428/349184	196/897024

The Virtual Test Facility

- ~430,000 LOC C++, C, Fortran-77, Fortran-90
- autoconf / automake environment with support for typical parallel high-performance system

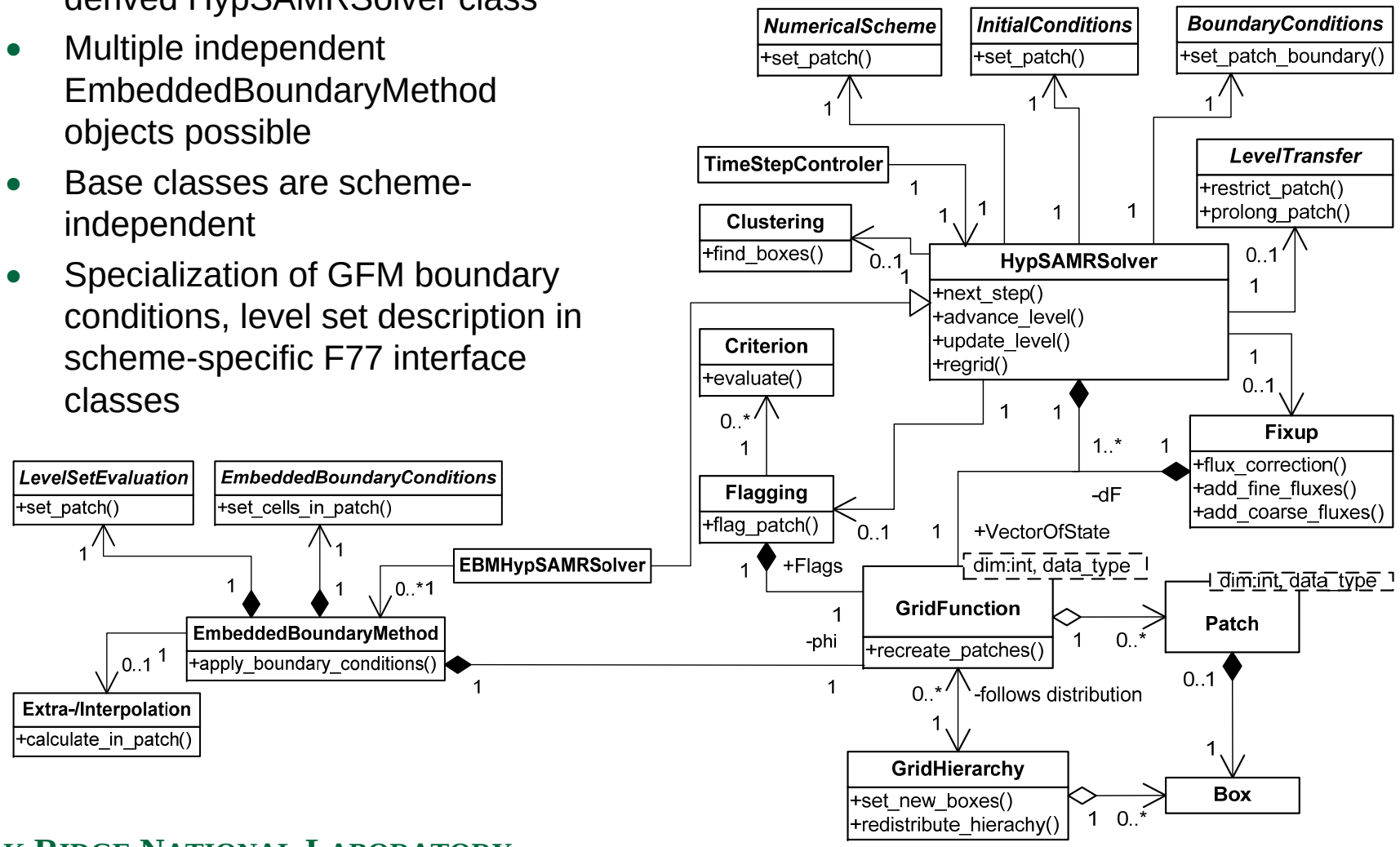
UML design of Amroc

- Classical framework approach with generic main program in C++
- Customization / modification in Problem.h include file by derivation from base classes and redefining virtual interface functions
- Predefined, scheme-specific classes (with F77 interfaces) provided for standard simulations
- Standard simulations require only linking to F77 functions for initial and boundary conditions, source terms. No C++ knowledge required.
- Interface mimics Clawpack
- Expert usage (algorithm modification, advanced output, etc.) in C++

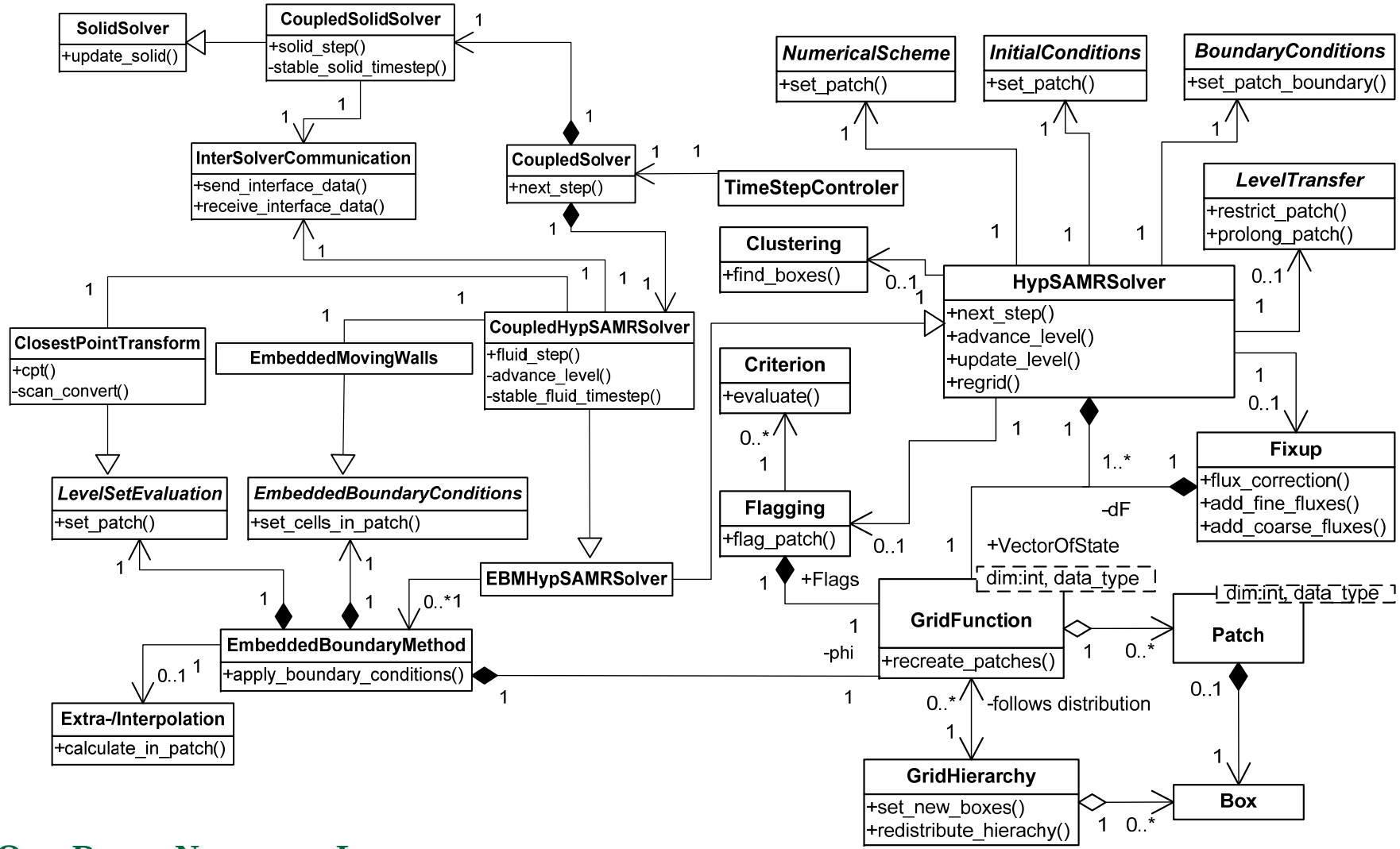


Ghost fluid method in Amroc

- Core algorithm implemented in derived HypSAMRSolver class
- Multiple independent EmbeddedBoundaryMethod objects possible
- Base classes are scheme-independent
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes



Amroc coupled to CSD solver (the VTF)

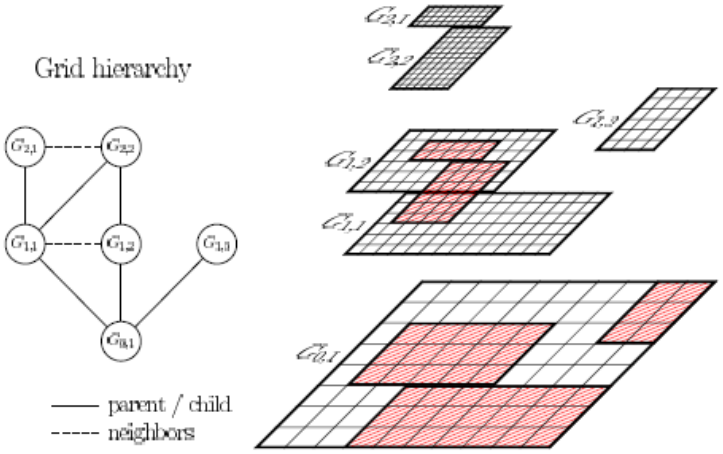


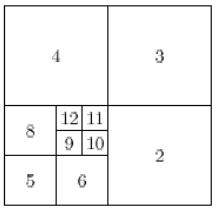
Block-based adaptive meshes

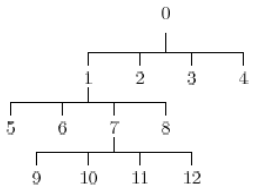
- Cells without marks get refined
- Suczessively embedded blocks of arbitrary size
  - *Special block generation algorithm necessary*
  - *Complex implementation (general tree)*
  - *Complex load balancing*
- Element-wise embedding of fine cell blocks
  - *Large number of ghost cells*
  - *Simple implementation (Quad-tree)*
  - *Simple load balancing*
- Numerical update

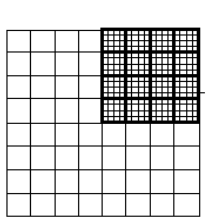
$$\mathbf{Q}_{jk}^{n+1} = \mathbf{Q}_{jk}^{n} - \frac{\Delta l}{\Delta x_1}\left[\mathbf{F}^1_{j+\frac{1}{2},k} - \mathbf{F}^1_{j-\frac{1}{2},k}\right] - \frac{\Delta l}{\Delta x_2}\left[\mathbf{F}^2_{j,k+\frac{1}{2}} - \mathbf{F}^2_{j,k-\frac{1}{2}}\right]$$

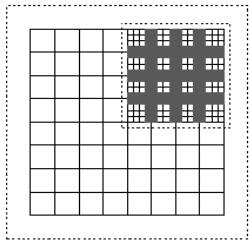
only necessary for single block





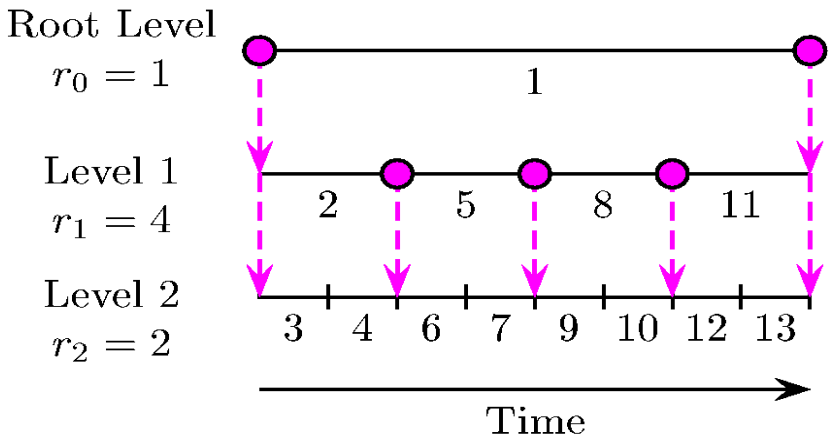






Berger-Collela-AMR-Algorithmus

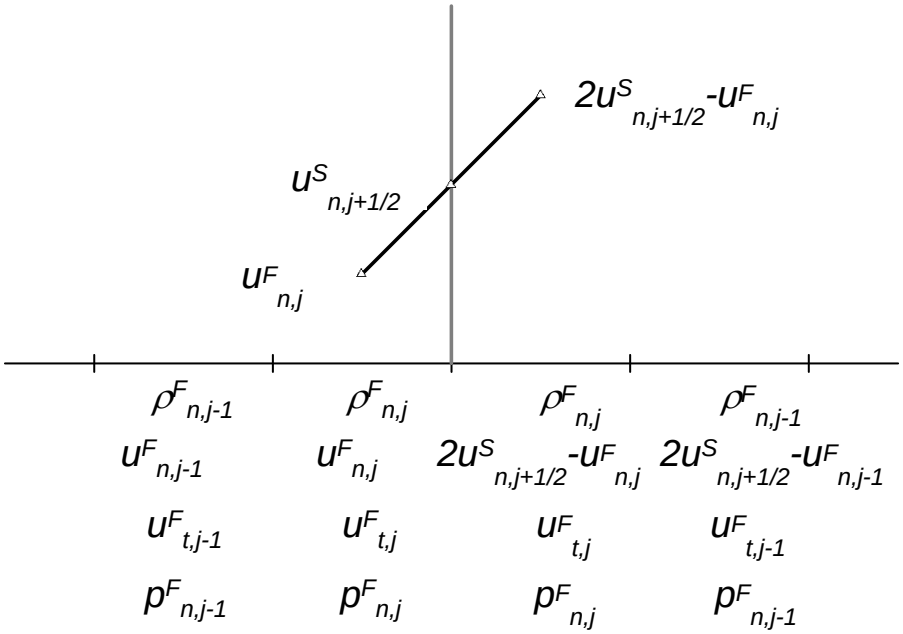
```
AdvanceLevel(l)

   Repeat r_l times
        Set ghost cells of Q^l(t)
        If time to regrid
             Regrid(l)
        UpdateLevel(l)
        If level l+1 exists
             Set ghost cells of Q^l(t + Δt_l)
             AdvanceLevel(l+1)
             Average Q^{l+1}(t + Δt_l) onto Q^l(t +
             Correct Q^l(t + Δt_l) with δF^{n,l+1}
        t := t + Δt_l
```



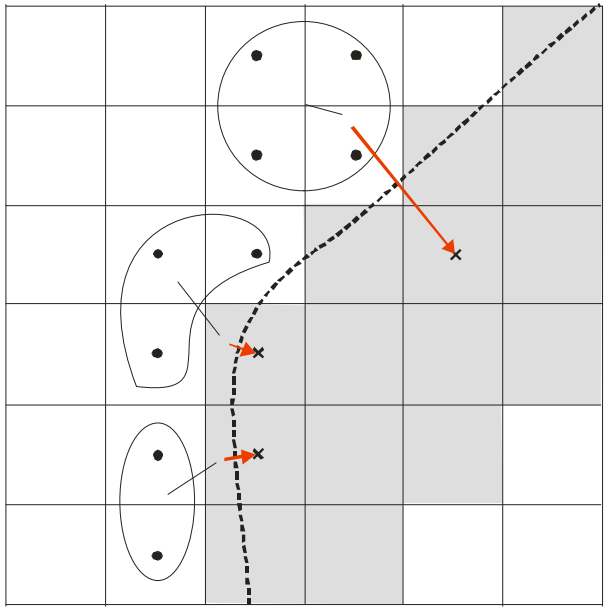
- FV scheme
- Interpolation
- Error indicator
- Grid manipulation
- Load balancing

Embedded boundary method

- Incorporate complex moving boundary/ interfaces into a Cartesian solver
- Implicit boundary representation via distance function φ, normal $n = \nabla\varphi / |\nabla\varphi|$
- Treat an interface as a moving rigid wall
- Construction of values in embedded boundary cells by interpolation / extrapolation



Velocity: $\boldsymbol{u}^F_{Gh} = 2((\boldsymbol{u}^S - \boldsymbol{u}^F_M)\cdot\boldsymbol{n})\,\boldsymbol{n} + \boldsymbol{u}^F_M$



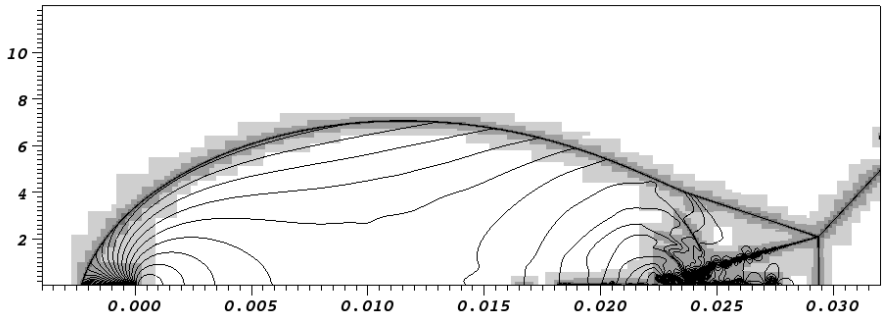
- Diffusive boundary approximation, non-conservative 1st order method
- Higher resolution at embedded boundary usually required than with first-order unstructured scheme
- Appropriate level-set-based refinement criteria required

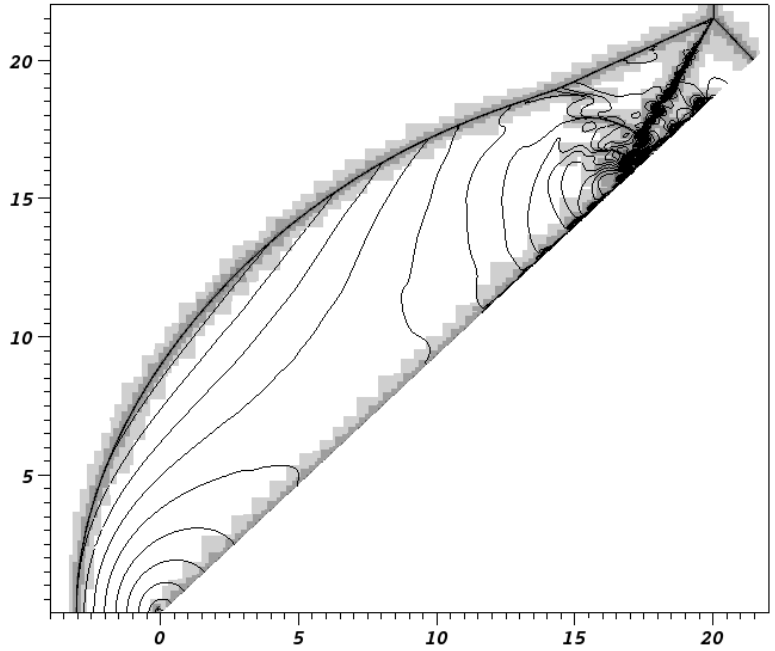
RD, Computers & Structures, 87: 769-783, 2009

Verification of embedded boundary method

- Reflection of a Mach 2.38 shock in nitrogen at 43^{o} wedge
- 2nd order MUSCL scheme with Roe solver, 2nd order multidimensional wave propagation

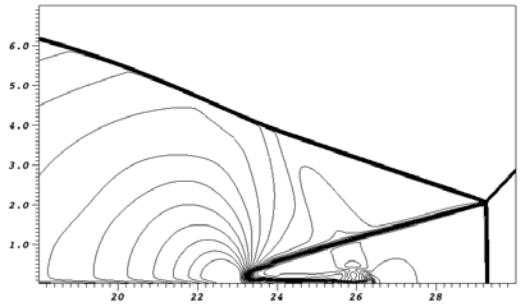
- Cartesian base grid 360x160 cells on domain of 36mm x 16mm with up to 3 refinement levels with refinement factors 2, 4, 4 $\rightarrow$ Δx=3.125 μm, 38h CPU



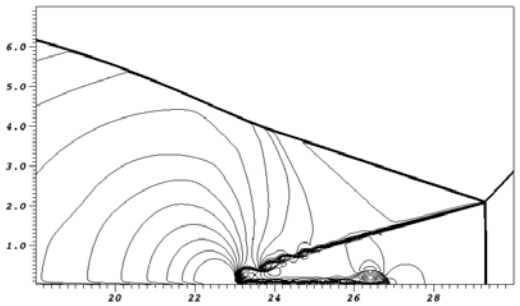


- GFM base grid 390x330 cells on domain of 26mm x 22mm with up to 3 refinement levels with refinement factors 2, 4, 4 $\rightarrow$ Δx_e=2.849 μm, 200h CPU

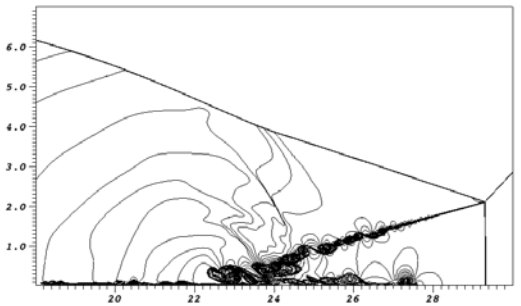
Verification of embedded boundary method, AMR



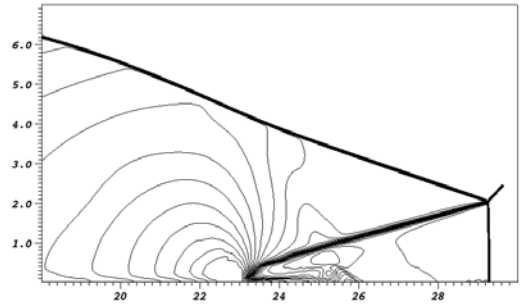
Δx=25μm



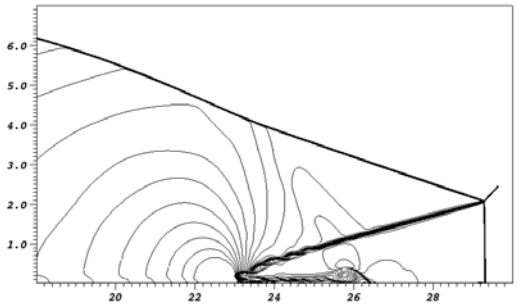
Δx=12.5μm



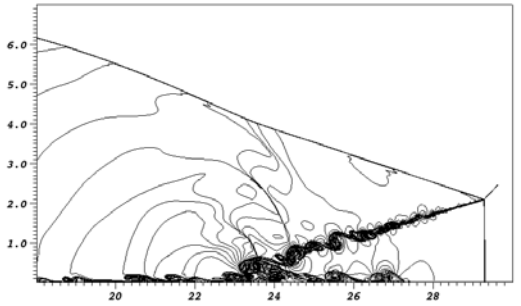
Δx=3.125μm



$\Delta x_e \approx 22.8\mu m$

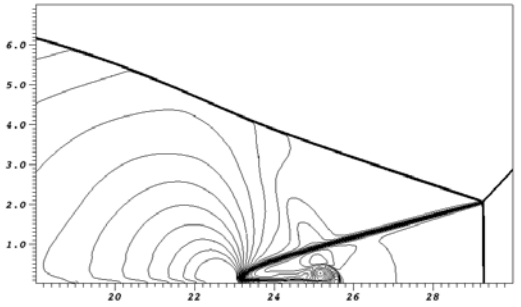


$\Delta x_e \approx 11.4\mu m$

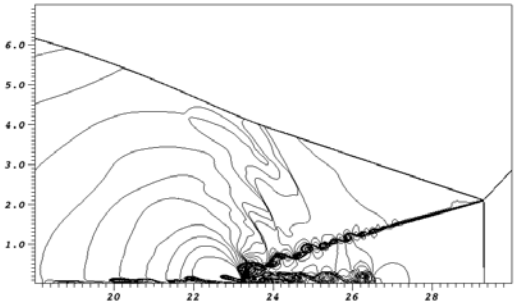


$\Delta x_e \approx 2.849\mu m$

- 2nd order MUSCL scheme with Van Leer FVS, dimensional splitting



Δx=12.5μm

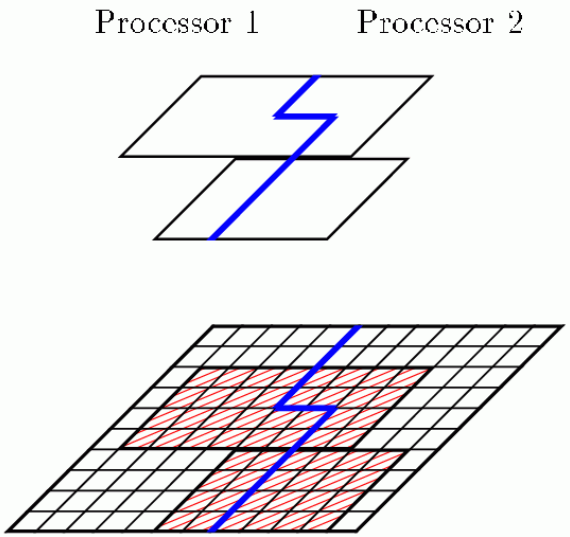


Δx=3.125μm

Parallelization

- Possibilities:
  - *Distribution of individual patches within level $\rightarrow$ only suitable for shared memory (high synchronization costs)*
  - *Independent domain decomposition of each level $\rightarrow$ high inter-level communication costs*
- Rigorous domain decomposition of base mesh based on overall workload with generalized space filling curve
  - *Some load imbalance accepted*
  - *Inter-Level operations strictly local*
  - *Parallel operations:*
    - *Synchronization of ghost cells*
    - *Repartitioning of patches within regridding operation*
    - *Flux correction on coarse grid cells*

$$\mathcal{W}(\Omega) = \sum_{l=0}^{l_{\max}} \left[\mathcal{N}_l(G_l \cap \Omega) \prod_{\nu=0}^{l} r_\nu \right]$$



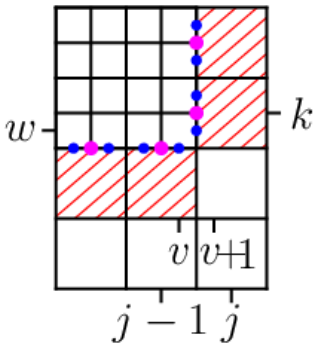
Flux correction

Correction pass:

1. $\delta \mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} := -\mathbf{F}^{1,l}_{j-\frac{1}{2},k}$

2. $\delta \mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} := \delta \mathbf{F}^{1,l+1}_{j-\frac{1}{2},k} + \frac{1}{r_{l+1}} \sum_{\nu=0}^{r_{l+1}-1} \mathbf{F}^{1,l+1}_{v+\frac{1}{2},w+\nu}(t + \mu \Delta t_{l+1})$

3. $\check{\mathbf{Q}}^{l}_{jk}(t + \Delta t_l) := \mathbf{Q}^{l}_{jk}(t + \Delta t_l) + \frac{\Delta t_l}{\Delta x_{1,l}} \delta \mathbf{F}^{1,l+1}_{j-\frac{1}{2},k}$

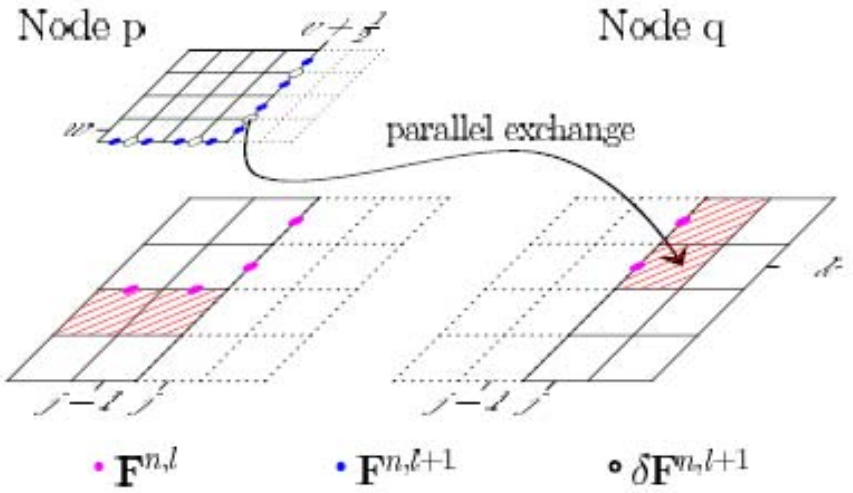


Example: Cell j, k

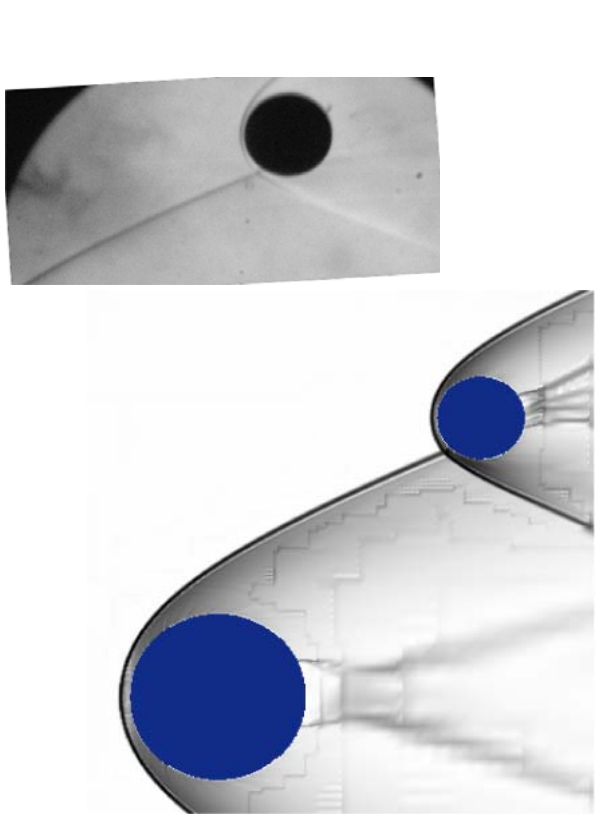
Only Step 3 needs to be parallelized!

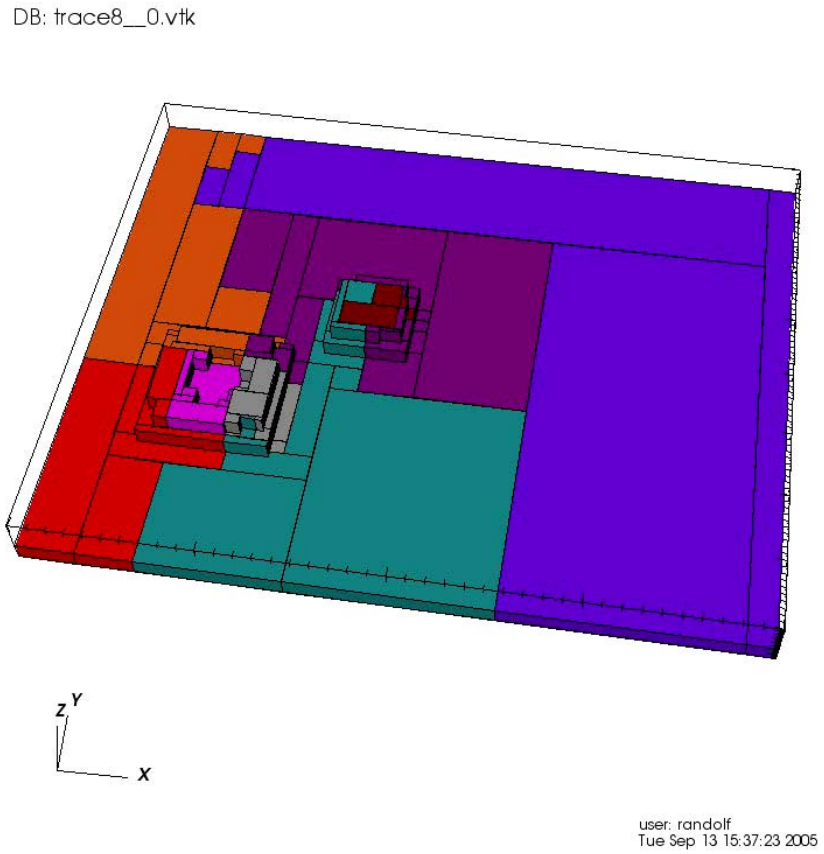
Communication of lower-dimensional data on coarse mesh

RD, in T. Plewa et al., *Adaptive Mesh Refinement – Theory and Applications*, p. 361-372, Springer, 2005



Partitioning example



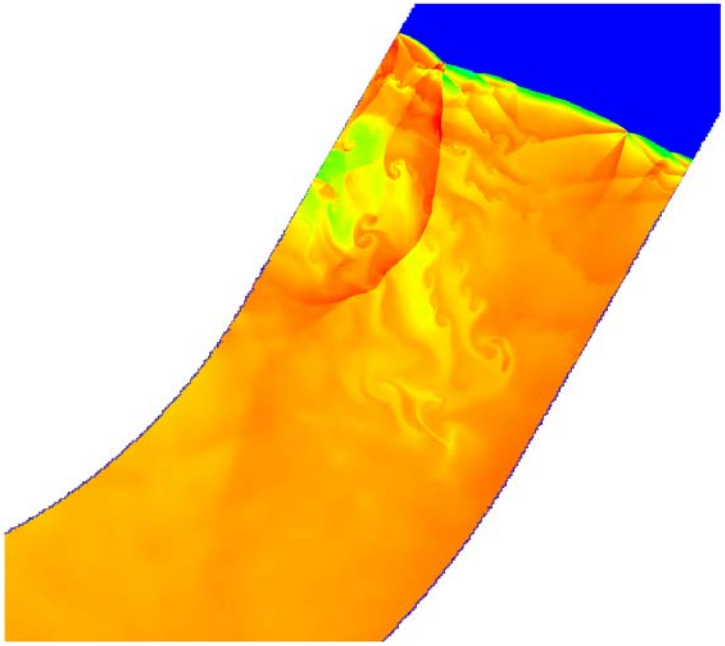


- Cylinders of spheres in supersonic flow
- Predict force on secondary body
- Right: 200x160 base mesh, 3 Levels, factors 2,2,2, 8 CPUs

S. Laurence, RD, H. Hornung, Journal of Fluid Mechanics, 590: 209-237, 2007

Large scale AMR example

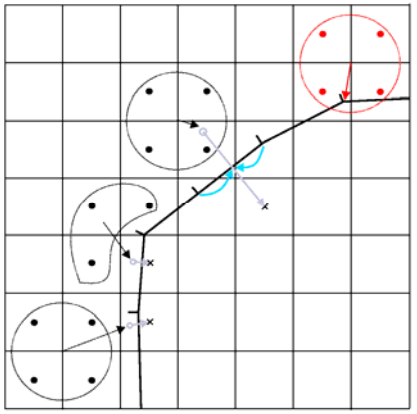
- Detonation propagation through a 60 degree elbow
- 67.6 Pts within induction length. AMR base grid 1200x992. 4 additional refinement levels (2,2,2,4). Adaptive computations use up to $7.1 \cdot 10^6$ cells ($4.8 \cdot 10^6$ on highest level) instead of $1.22 \cdot 10^9$ cells (uniform grid)
- ~70,000h CPU on 128 CPUs Pentium-4 2.2GHz



RD, Computers & Structures, 87: 769-783, 2009

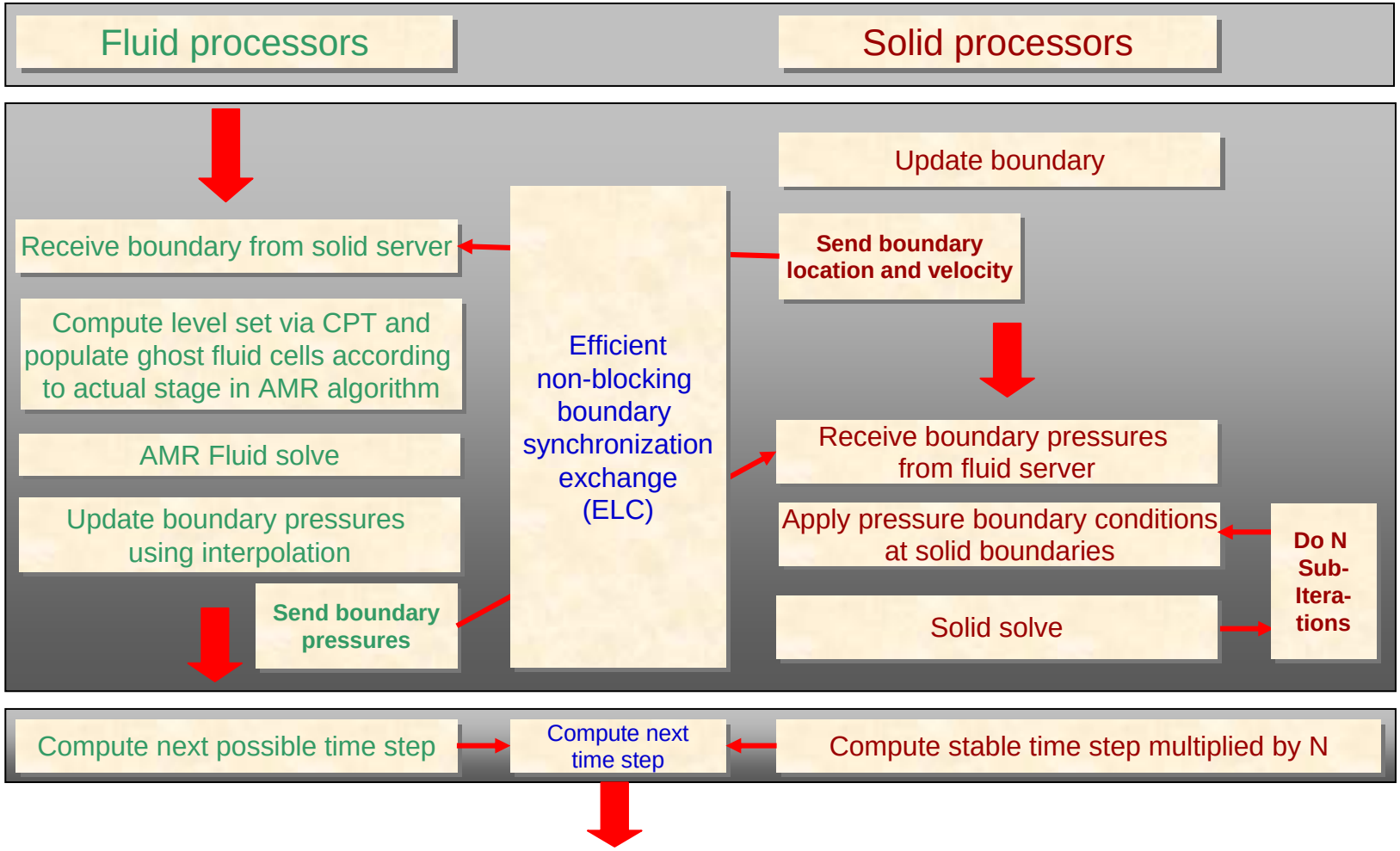
Fluid-structure coupling

- Couple compressible Euler equations to Lagrangian structure mechanics
- Compatibility conditions between inviscid fluid and solid at a slip interface
  - *Continuity of normal velocity:* $u^S_n = u^F_n$
  - *Continuity of normal stresses:* $\sigma^S_{nn} = -p^F$
  - *No shear stresses:* $\sigma^S_{n\tau} = \sigma^S_{n\omega} = 0$



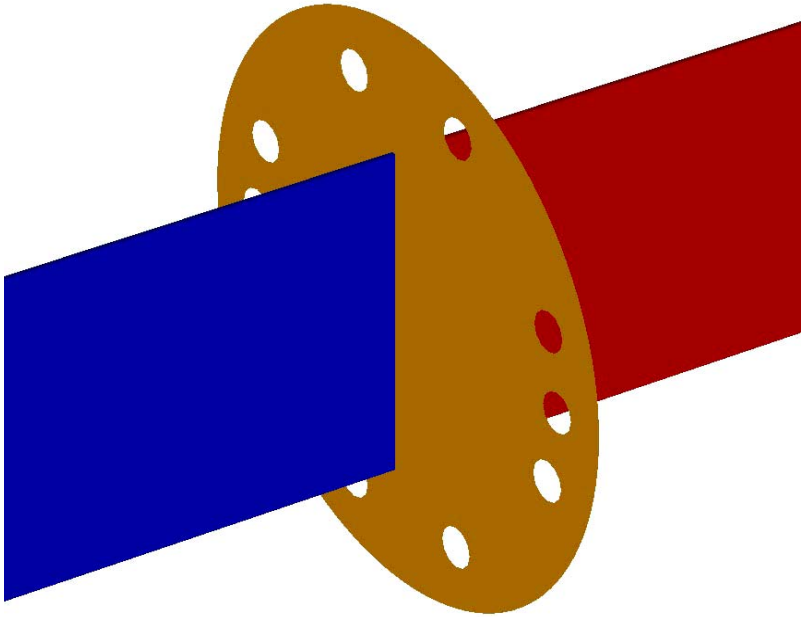
- Time-splitting approach for coupling
  - *Fluid:*
    - *Treat evolving solid surface with moving wall boundary conditions in fluid*
    - *Use solid surface mesh to calculate fluid level set*
    - *Use nearest velocity values* $\boldsymbol{u}^S$ *on surface facets to impose* u^F_n *in fluid*
  - *Solid:*
    - *Use interpolated hydro-pressure* p^F *to prescribe* σ^S_{nn} *on boundary facets*
- Ad-hoc separation in dedicated fluid and solid processors

Algorithmic approach for coupling

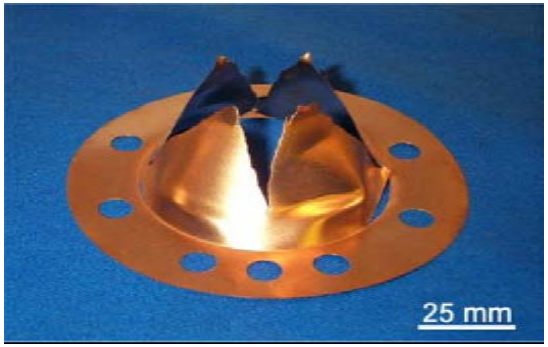


FSI example: plate fracture by water hammer

- Level set with unsigned distance function allows for topology changes
- Thin-shell finite element solver with fracture and cohesive interface model
- AMR base grid: 374x20x20, 2 levels, refinement factors 2,2, FEM: 8896 triangles, ~1250 time steps to t=1.0 ms
- 6+6 nodes 3.4-GHz-Intel-Xeon-dual-processor, ~800h CPU



Initial pressure
p_0=64 MPa



RD, F. Cirak, S. Mauch, Proc. Int. Workshop FSI, Herrsching, 2008

Benchmarking of fluid solver

- Calculation of coupled 50 time steps on 3.4 GHz Intel Xeon dual processor
- *F* processors used for fluid solver, *S* processors for solid solver
- AMR base level: 350x20x20, 2 additional levels, refinement factor 2,2
- FSI coupling at level *2*, 5 sub-iterations in solid solver

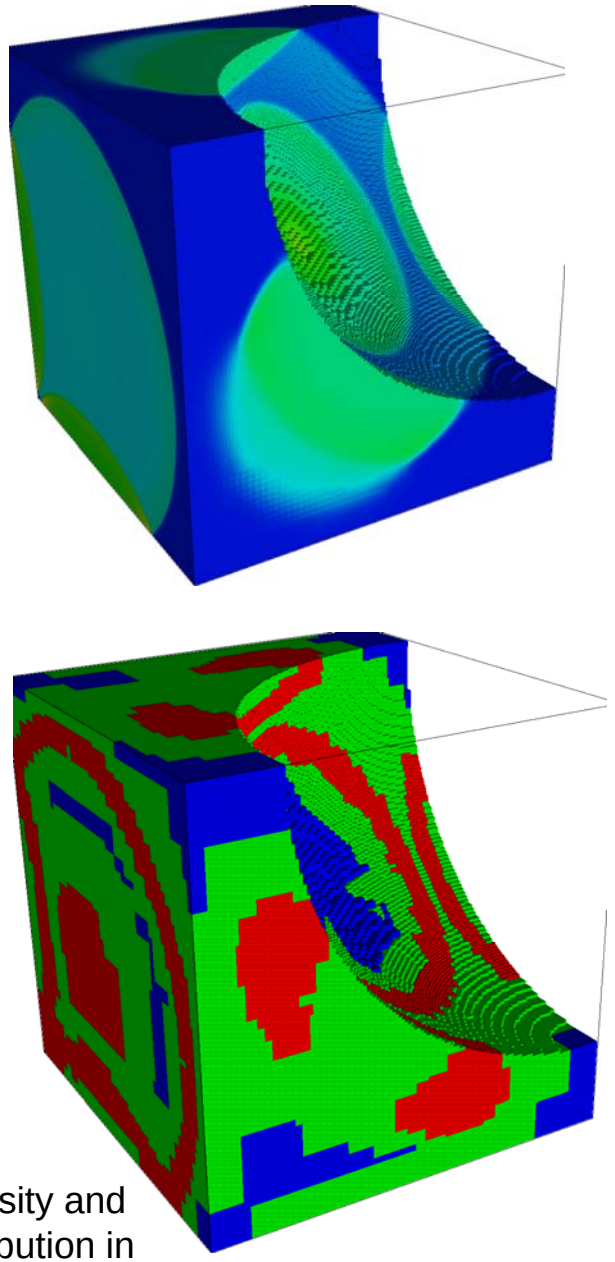
Task	F=6, S=2	F=12, S=4	F=24, S=8	F=48, S=16	F=96, S=32
	%	%	%	%	%
Integration	27.3	22.4	16.1	12.5	8.8
Boundary sync	41.3	39.6	48.1	50.1	47.3
Recomposition	3.3	5.5	6.4	8.5	10.4
Interpolation	1.0	1.0	0.7	0.5	0.4
Regridding	0.7	0.9	0.6	0.4	0.4
GFM Find cells	3.2	2.7	2.0	1.6	1.2
GFM Interpolation	10.0	8.6	6.1	4.7	3.5
GFM Overhead	1.6	0.7	0.3	0.5	0.3
CPT	0.5	0.6	0.7	0.9	1.3
Level set sync	2.6	7.3	8.6	9.1	11.3
ELC	5.7	7.4	7.6	8.0	10.8
Coupling data calc	0.5	0.3	0.3	0.3	0.2
Misc	2.3	3.0	2.5	2.9	4.1
Time per step [s]	**23**	**14**	**11**	**7**	**5**

Performance test

- Test run on 2.2 GHz AMD Opteron quad-core cluster connected with Infiniband
- Cartesian test configuration
  - *Spherical blast wave, Euler equations, 3rd order WENO scheme, 3-step Runge-Kutta update*
  - *AMR base grid 64^3, 2 levels with factors 2, 2, 89 time steps on coarsest level*
    - *Redistribute in parallel every 2nd level 0 step*
  - *Uniform grid $256^3=16.8\cdot 10^6$ cells, 355 time steps*
- Embedded boundary method
  - *AMR: 96 time steps on coarsest level*
  - *Uniform: 383 time steps*

Level	Grids	Cells
0	115	262,144
1	373	1,589,808
2	2282	5,907,064

Grid and cells used on 16 CPUs



Right: density and level distribution in test simulation

Unigrid – costs of embedded boundary method

- Embedded boundary method requires 3 ghost cells
- GFM Interpolation is comparably inefficient C++ code
- Overall overhead costs are constant around 16-21%

CPUs	**16**	**32**	**64**
Time per step	*59.65 s*	*29.70 s*	*15.15 s*
Integration	90.33%	90.90%	88.97%
Boundary Setting	4.53%	4.19%	4.84%
Output	0.33%	0.68%	1.36%
Misc.	4.81%	4.23%	4.76%

CPUs	**16**	**32**	**64**
Time per step	*69.09 s*	*35.94 s*	*18.24 s*
Integration	75.46%	72.94%	71.33%
Boundary Setting	9.42%	11.51%	12.61%
GFM Find Cells	2.51%	2.41%	2.37%
GFM Interpolation	7.64%	7.30%	7.05%
GFM Overhead	0.60%	0.57%	0.57%
GFM Calculate	0.96%	0.92%	0.87%
Output	0.31%	0.60%	1.26%
Misc.	3.08%	3.47%	3.67%

Costs of adaptive mesh refinement

CPUs	16	32	64
Time per step	*32.44 s*	*18.63*	*11.87 s*
Uniform	*59.65 s*	*29.70 s*	*15.15 s*
Integration	73.46%	64.69%	50.44%
Flux Correction	1.30%	1.49%	2.03%
Boundary Setting	13.72%	16.60%	20.44%
Regridding	10.43%	15.68%	24.25%
Clustering	0.34%	0.32%	0.26%
Output	0.29%	0.53%	0.92%
Misc.	0.46%	0.44%	0.47%

- Flux correction is negligible
- Clustering is negligible (already local approach)
- Costs for GFM constant around ~36%
- Main costs: Regrid(l) operation and ghost cell synchronization

CPUs	16	32	64
Time per step	*43.97 s*	*25.24 s*	*16.21 s*
Uniform	*69.09 s*	*35.94 s*	*18.24 s*
Integration	59.09%	49.93%	40.20%
Flux Correction	0.82%	0.80%	1.14%
Boundary Setting	19.22%	25.58%	28.98%
Regridding	7.21%	9.15%	13.46%
Clustering	0.25%	0.23%	0.21%
GFM Find Cells	2.04%	1.73%	1.38%
GFM Interpolation	6.01%	10.39%	7.92%
GFM Overhead	0.54%	0.47%	0.37%
GFM Calculate	0.70%	0.60%	0.48%
Output	0.23%	0.52%	0.74%
Misc.	0.68%	0.62%	0.58%

AMROC scalability tests

- Basic configuration test
  - *Spherical blast wave, Euler equations, 3D wave propagation method*
  - *AMR base grid 32^3, 2 levels with factors 2, 4, 5 time steps on coarsest level*
  - *Uniform grid $256^3=16.8\cdot 10^6$ cells, 19 time steps*
- Weak scalability test
  - *Reproduction of configuration each 64 CPUs*

Level	Grids	Cells
0	606	32,768
1	575	135,312
2	910	3,639,040

  - *On 1024 CPUs: 128x64x64 base grid, >33,500 Grids, ~$61\cdot 10^6$ cells, uniform 1024x512x512=$268\cdot 10^6$ cells*

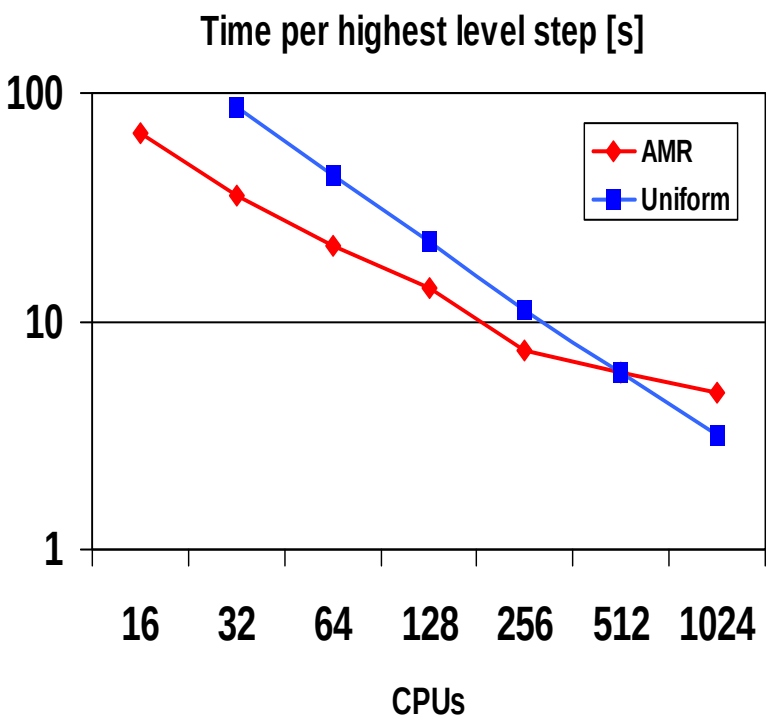
- Flux correction deactivated
- No volume I/O operations
- Tests run IBM BG/P Eugene at NCCS
  - *Mode –VN until otherwise indicated*

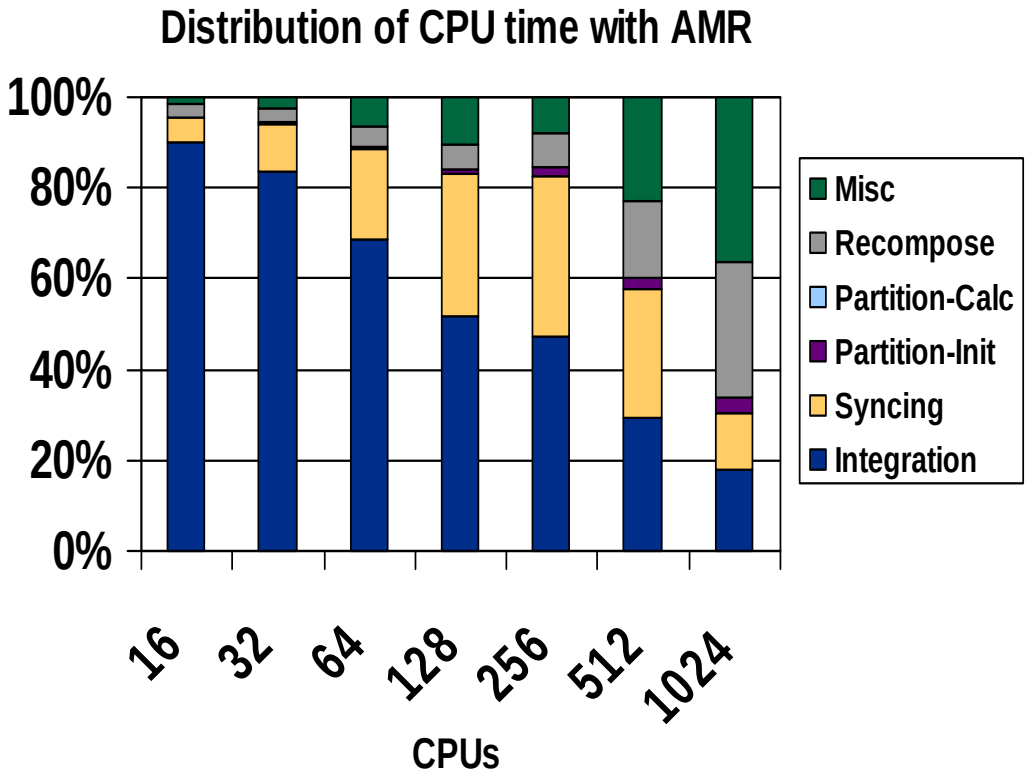
- Strong scalability test
  - *64x32x32 base grid*

Level	Grids	Cells
0	1709	65,536
1	1735	271,048
2	2210	7,190,208

  - *Uniform 512x256x256= $33.6\cdot 10^6$ cells*

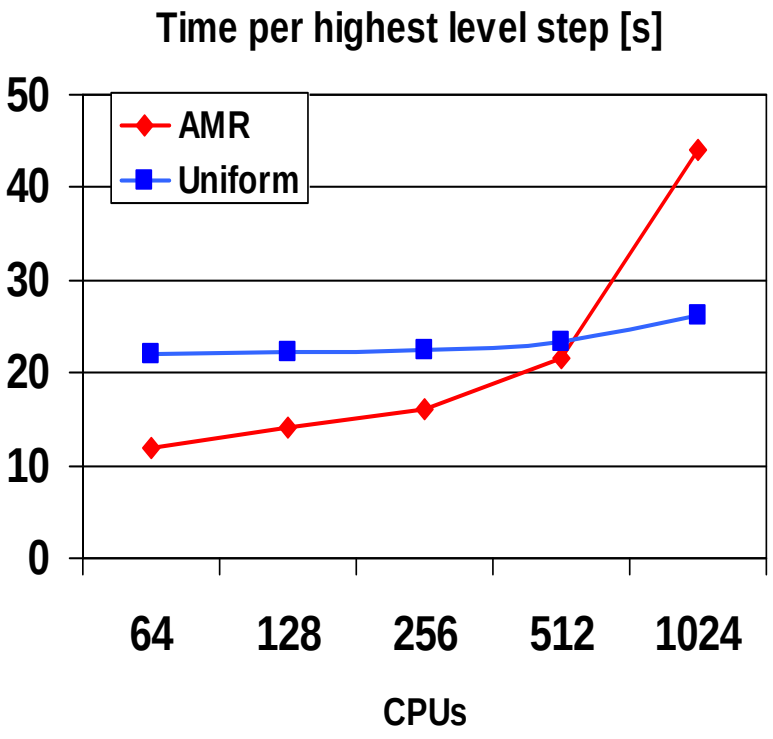
Strong scalability test – old code

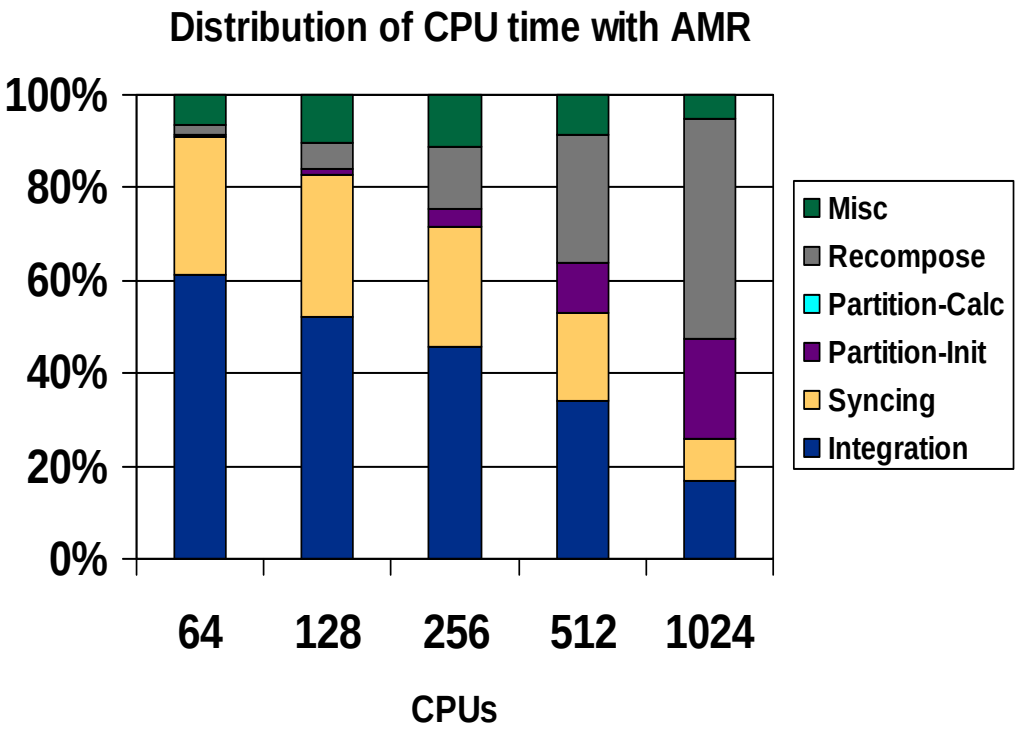




- Misc: time not measured (often waiting times), not interpolation or clustering
- Partition-Init, Partition-Calc: construction of space filling curve
- Syncing: Parallel communication portion of Boundary setting
- Recompose: topological list operations, construction of boundary info, redistribution of data blocks

Weak scalability test – old code





- Costs for Partition-Init and Recompose increase dramatically for large problem size

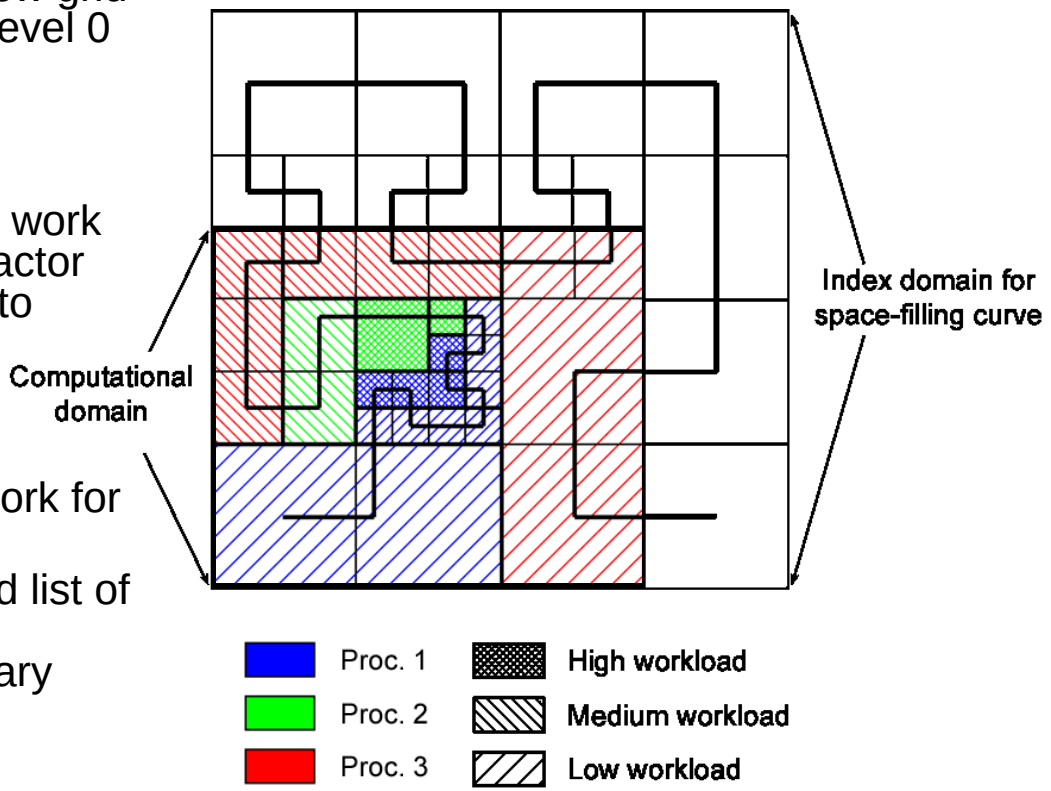
Construction of space-filling curve

Partition-Init:

1. Compute aggregated workload for new grid hierarchy G_l and project result onto level 0

$$\mathcal{W}(\Omega) = \sum_{l=0}^{l_{\max}} \left[\mathcal{N}_l(G_l \cap \Omega) \prod_{\nu=0}^{l} r_\nu \right]$$

2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid



Partition-Calc:

1. Compute entire workload and new work for each processor
2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible

- Ensure scalability of Partition-Init by creating SFC-units strictly local
- Currently still use of MPI_allgather() to create globally identical input for Partition-Calc

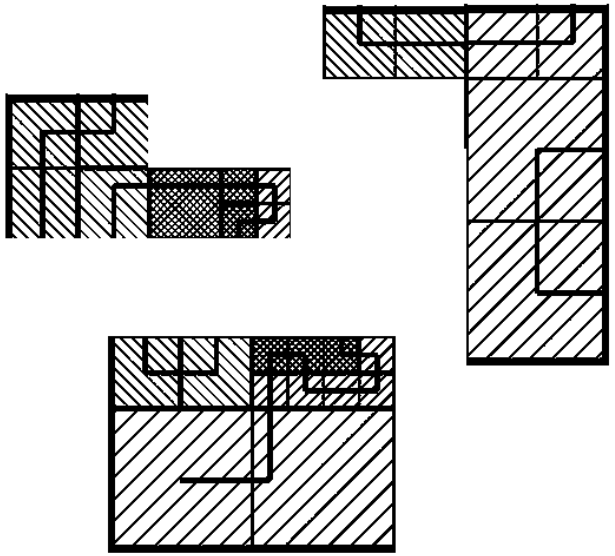
Construction of space-filling curve

Partition-Init:

1. Compute aggregated workload for new grid hierarchy G_l and project result onto level 0

$$\mathcal{W}(\Omega) = \sum_{l=0}^{l_{\max}} \left[\mathcal{N}_l(G_l \cap \Omega) \prod_{\nu=0}^{l} r_\nu \right]$$

2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid



Partition-Calc:

1. Compute entire workload and new work for each processor
2. Go sequentially through SFC-ordered list of partitioning units and assign units to processors, refine partition if necessary and possible

- Ensure scalability of Partition-Init by creating SFC-units strictly local
- Currently still use of MPI_allgather() to create globally identical input for Partition-Calc

Elimination of global list operations

Recompose(l)

For $\iota = l$ To l_c Do
$C\bar{G}_\iota := G_0 \backslash \bar{G}_\iota$, $\bar{G}_{\iota+1} := \bar{G}_{\iota+1} \backslash C\bar{G}_\iota^1$

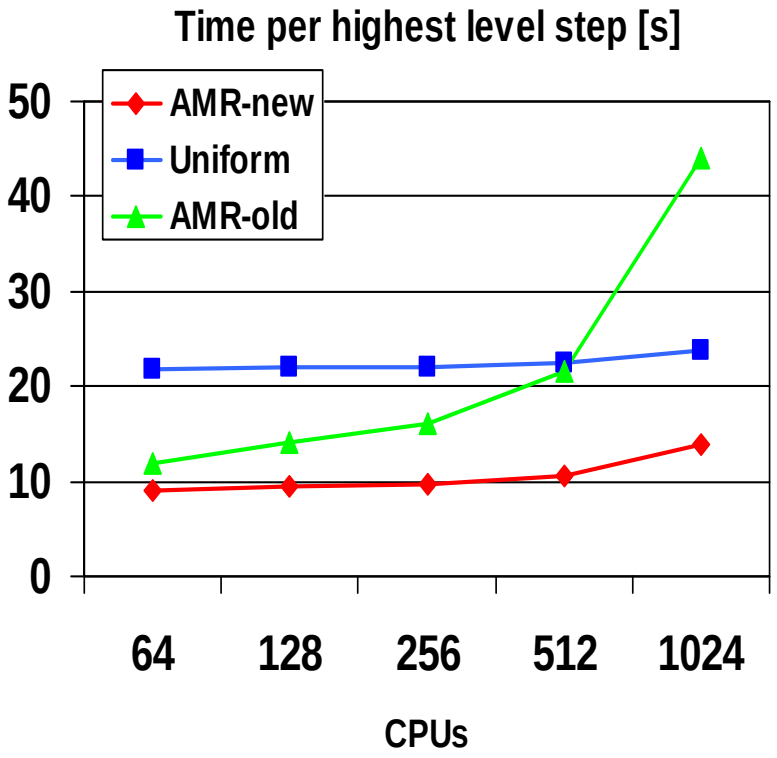
Generate G_0^p from $\{G_0, ..., G_l, \bar{G}_{l+1}, ..., \bar{G}_{l_c+1}\}$
For $\iota = 0$ To $l_c + 1$ Do
If $\iota > l$?
$\bar{G}_\iota^p := \bar{G}_\iota \cap G_0^p$
Interpolate $\mathbf{Q}^{\iota-1}(t)$ onto $\bar{\mathbf{Q}}^\iota(t)$
else
$\bar{G}_\iota^p := G_\iota \cap G_0^p$
If $\iota > 0$?
Copy $\delta\mathbf{F}^{n,\iota}$, onto $\delta\bar{\mathbf{F}}^{n,\iota}$
$\delta\mathbf{F}^{n,\iota} := \delta\bar{\mathbf{F}}^{n,\iota}$
If $\iota \geq l$? $\kappa_\iota = 0$ else $\kappa_\iota = 1$
For $\kappa = 0$ To κ_ι Do
Copy $\mathbf{Q}^\iota(t + \kappa\Delta t_\iota)$ onto $\bar{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
Set ghost cells of $\bar{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
$\mathbf{Q}^\iota(t + \kappa\Delta t_\iota) := \bar{\mathbf{Q}}^\iota(t + \kappa\Delta t_\iota)$
$G_\iota^p := \bar{G}_\iota^p$, $G_\iota := \bigcup_p G_\iota^p$

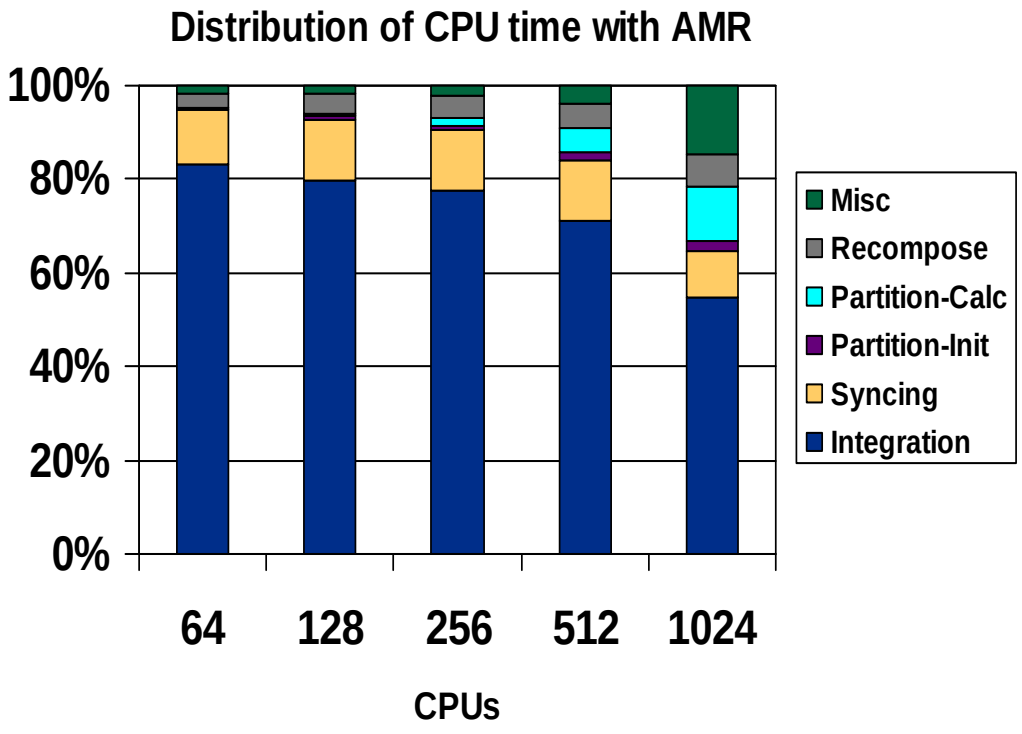
- Operations $\backslash, \cap$ on two box lists have complexity O(N M)
- Costs of operations in Recompose(l), that use global box lists G_l increase quadratically (problem pointed out by SAMRAI team)

Solution:

- Clip G_l with *properly chosen* quadratic bounding box around G_l^p before using $\backslash, \cap$
- *All* topological operations involving global lists can be reduced to local ones
- Present code still uses MPI_allgather() to communicate global lists to all nodes
- Global view is particularly useful to evaluate new local portion of hierarchy and for data redistribution

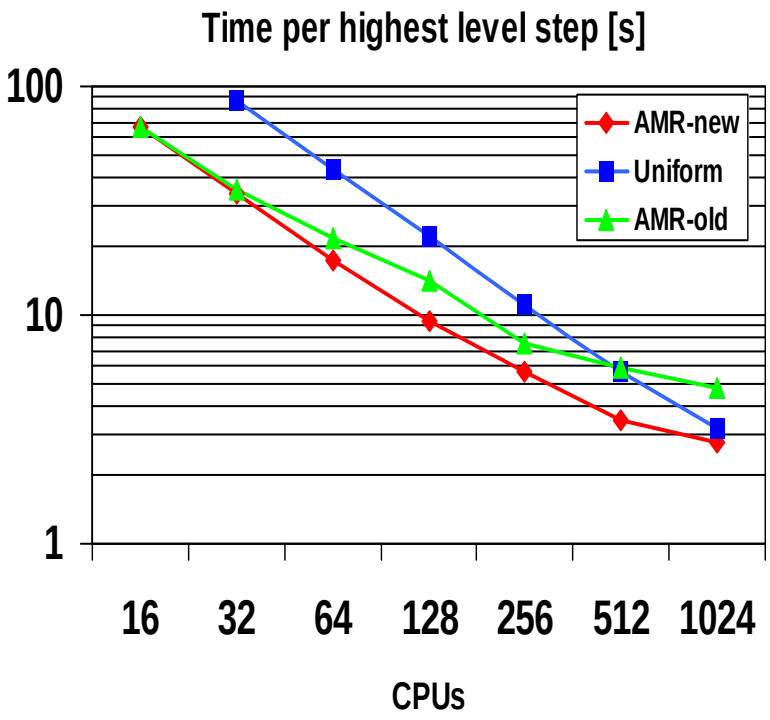
Weak scalability test – new code

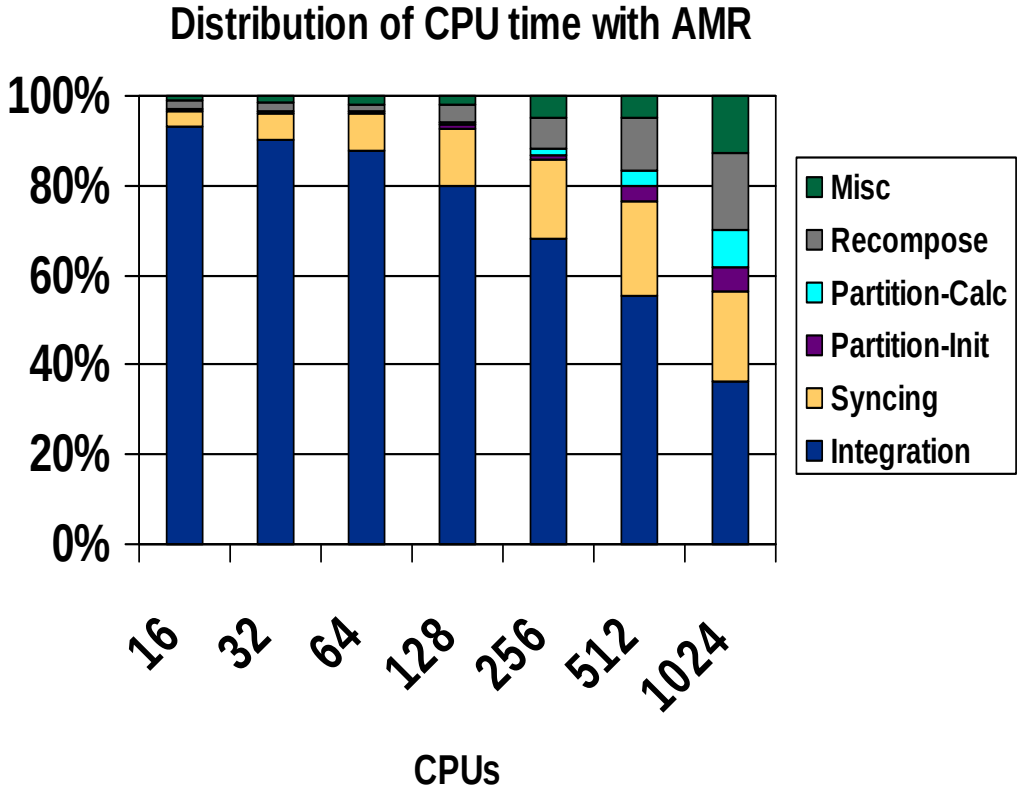




- Overall performance improvement for 1024 CPUs by ~69%
- Absolute costs for Syncing are almost constant
- 1024 required usage of -DUAL option due to usage of global lists data structures in Partition-Calc and Recompose

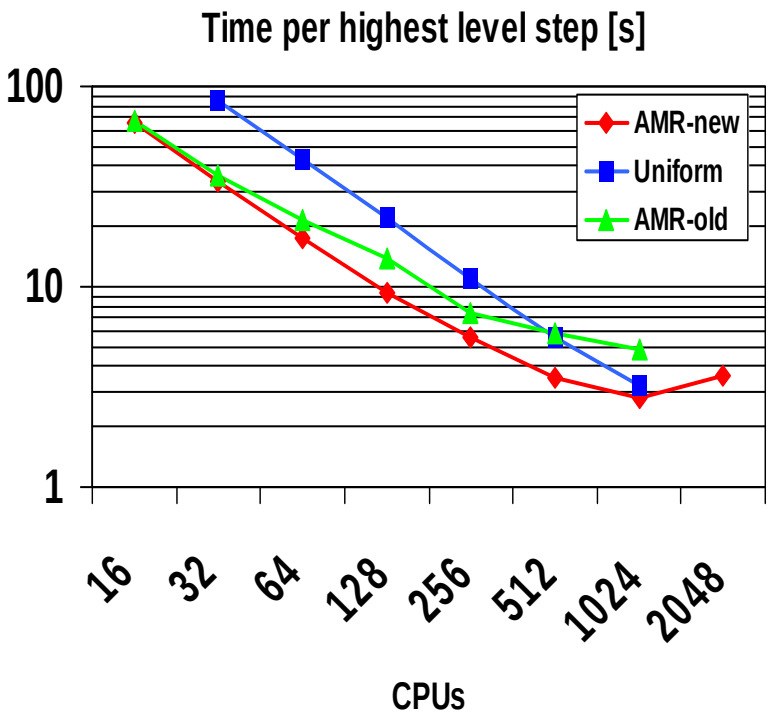
Strong scalability test – new code

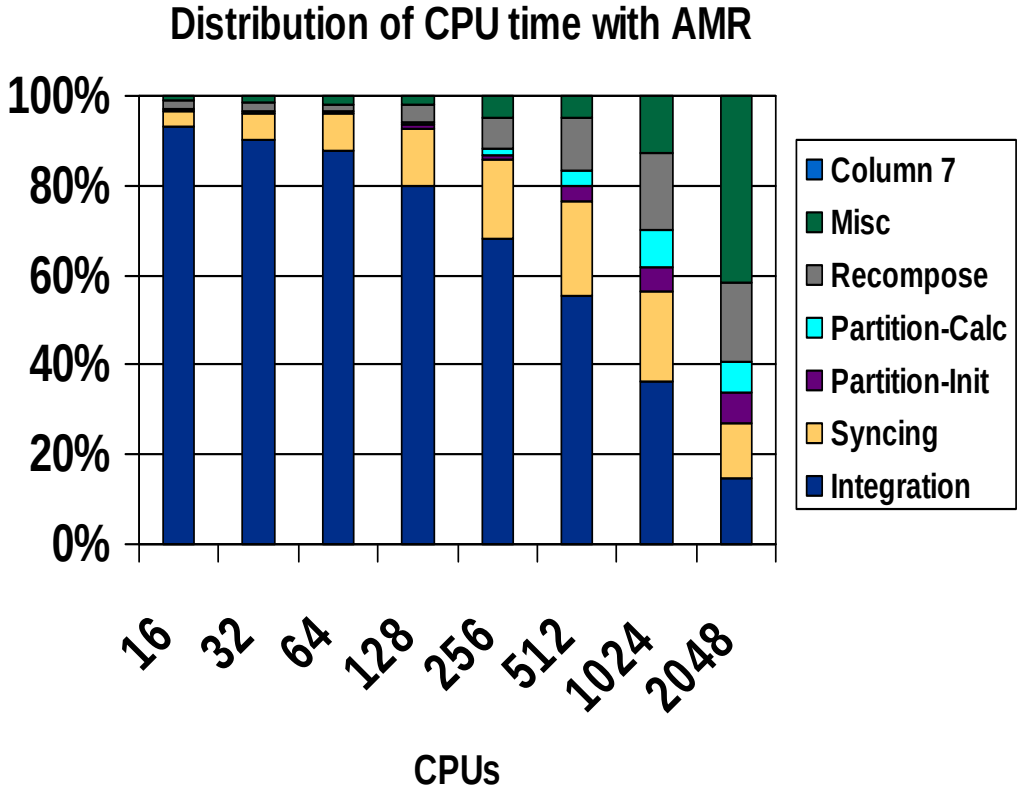




- Overall performance improvement for 1024 CPUs by 43%
- Improved partitioning algorithm allowed usage of GuCFactor=1 instead of 2 before and full parallel data redistribution in every Regrid(l) instead of every 2nd level-0 step

Strong scalability test – new code





- 2048 required usage of -DUAL option due to usage of global lists data structures in Partition-Calc and Recompose

Conclusions and outlook

- Achieved major improvements in strong and weak scalability for block-structured AMR framework by
  - *Local recursive generation of space filling curve*
  - *Reduction of topological list operations to local domains*
- Next step is to eliminate aggregation of global box list data (that currently uses simply MPI_allgather)
  - *Crucial for reducing the memory foot print*
- Two operations need to be considered
  - *Partition-Calc: assignment of SFC-ordered sequence and refinement could be executed sequentially on each node*
    - *Avoid global data but costs grow linearly with node count*
  - *Global topology lists: assemble only those portions of global lists on each node that are relevant for the subsequent operations*
    - *use Cartesian bounding box information to construct irregular point-to-point communication pattern for list data between nodes*
    - *A larger number and more complex communication operations would be required but the data volume will be greatly reduced*
- Weak scalability to O(10,000) nodes should be easily achievable