Lecture 6
The AMROC software system

Course *Block-structured Adaptive Finite Volume Methods for Shock-Induced Combustion Simulation*

Ralf Deiterding
German Aerospace Center (DLR)
Institute for Aerodynamics and Flow Technology
Bunsenstr. 10, Göttingen, Germany

E-mail: ralf.deiterding@dlr.de
Available SAMR software
- Simplified block-based AMR
- General patch-based SAMR
Outline

Available SAMR software
  Simplified block-based AMR
  General patch-based SAMR

AMROC
  Overview
  Layered software structure
Outline

Available SAMR software
   Simplified block-based AMR
   General patch-based SAMR

AMROC
   Overview
   Layered software structure

Massively parallel SAMR
   Performance data from AMROC
Simplified structured designs

* Distributed memory parallelization fully supported if not otherwise stated. *
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- **Uintah** (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)

- **DAGH/Grace** [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to bases mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- **DAGH/Grace** [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to bases mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Simplified structured designs

*Distributed memory parallelization fully supported if not otherwise stated.*

- **PARAMESH** (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- **Flash code** (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- **Uintah** (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)
Simplified structured designs

* Distributed memory parallelization fully supported if not otherwise stated.*

- PARAMESH (Parallel Adaptive Mesh Refinement)
  - Library based on uniform refinement blocks [MacNeice et al., 2000]
  - Both multigrid and explicit algorithms considered
  - [http://sourceforge.net/projects/paramesh](http://sourceforge.net/projects/paramesh)

- Flash code (AMR code for astrophysical thermonuclear flashes)
  - Built on PARAMESH
  - Solves the magneto-hydrodynamic equations with self-gravitation
  - [http://www.flash.uchicago.edu/site/flashcode](http://www.flash.uchicago.edu/site/flashcode)

- Uintah (AMR code for simulation of accidental fires and explosions)
  - Only explicit algorithms considered
  - FSI coupling Material Point Method and ICE Method (Implicit, Continuous fluid, Eulerian)
  - [http://www.uintah.utah.edu](http://www.uintah.utah.edu)

- DAGH/Grace [Parashar and Browne, 1997]
  - Just C++ data structures but no methods
  - All grids are aligned to bases mesh coarsened by factor 2
  - [http://userweb.cs.utexas.edu/users/dagh](http://userweb.cs.utexas.edu/users/dagh)
Available SAMR software

AMROC

Massively parallel SAMR

References

General patch-based SAMR

Systems that support general SAMR

AMROC

AMROC is a software system that supports general SAMR. It is a massively parallel system and is very mature [Hornung et al., 2006]. Explicit algorithms are directly supported, with implicit methods through an interface to the Hypre package. It supports mapped geometry and some embedded boundary support. For more information, visit https://computation-rnd.llnl.gov/SAMRAI/software.php.

BoxLib, AmrLib, MGLib, HGProj

BoxLib is a Berkeley-Lab-AMR collection of C++ classes by J. Bell et al. It supports both multigrid and explicit algorithms and has 50,000 LOC [Rendleman et al., 2000]. Embedded boundary support is also provided. For more information, visit https://ccse.lbl.gov/Downloads/index.html.

Chombo

Chombo is a redesign and extension of BoxLib by P. Colella et al. It supports both multigrid and explicit algorithms and has some embedded boundary support. For more information, visit https://commons.lbl.gov/display/chombo.
Systems that support general SAMR

- SAMRAI - Structured Adaptive Mesh Refinement Application Infrastructure
  - Very mature SAMR system [Hornung et al., 2006]
  - Explicit algorithms directly supported, implicit methods through interface to Hypre package
  - Mapped geometry and some embedded boundary support
Systems that support general SAMR

- **SAMRAI** - Structured Adaptive Mesh Refinement Application Infrastructure
  - Very mature SAMR system [Hornung et al., 2006]
  - Explicit algorithms directly supported, implicit methods through interface to Hypre package
  - Mapped geometry and some embedded boundary support

- **BoxLib, AmrLib, MGLib, HGProj**
  - Berkley-Lab-AMR collection of C++ classes by J. Bell et al., 50,000 LOC [Rendleman et al., 2000]
  - Both multigrid and explicit algorithms supported
Systems that support general SAMR

- **SAMRAI - Structured Adaptive Mesh Refinement Application Infrastructure**
  - Very mature SAMR system [Hornung et al., 2006]
  - Explicit algorithms directly supported, implicit methods through interface to Hypre package
  - Mapped geometry and some embedded boundary support

- **BoxLib, AmrLib, MGLib, HGProj**
  - Berkley-Lab-AMR collection of C++ classes by J. Bell et al., 50,000 LOC [Rendleman et al., 2000]
  - Both multigrid and explicit algorithms supported

- **Chombo**
  - Redesign and extension of BoxLib by P. Colella et al.
  - Both multigrid and explicit algorithms demonstrated
  - Some embedded boundary support
  - [https://commons.lbl.gov/display/chombo](https://commons.lbl.gov/display/chombo)
Further SAMR software
Further SAMR software

▶ Overture (Object-oriented tools for solving PDEs in complex geometries)
  ▶ Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  ▶ Explicit and implicit algorithms supported
  ▶ http://www.overtureframework.org
Further SAMR software

- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - http://www.overtureframework.org
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://depts.washington.edu/clawpack
Further SAMR software

- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - http://www.overtureframework.org

- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://depts.washington.edu/clawpack

- Amrita by J. Quirk
  - Only 2D explicit finite volume methods supported
  - Embedded boundary algorithm
  - http://www.amrita-cfd.org
Further SAMR software

- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - http://www.overtureframework.org
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://depts.washington.edu/clawpack
- Amrita by J. Quirk
  - Only 2D explicit finite volume methods supported
  - Embedded boundary algorithm
  - http://www.amrita-cfd.org
- Cell-based Cartesian AMR: RAGE
  - Embedded boundary method
  - Explicit and implicit algorithms
  - [Gittings et al., 2008]
AMROC

▶ “Adaptive Mesh Refinement in Object-oriented C++”

▶ ~ 46,000 LOC for C++ SAMR kernel, ~ 140,000 total C++, C, Fortran-77

▶ uses parallel hierarchical data structures that have evolved from DAGH
AMROC

▶ “Adaptive Mesh Refinement in Object-oriented C++”
▶ ~ 46,000 LOC for C++ SAMR kernel, ~ 140,000 total C++, C, Fortran-77
▶ uses parallel hierarchical data structures that have evolved from DAGH
▶ Right: point explosion in box, 4 level, Euler computation, 7 compute nodes
AMROC

▶ “Adaptive Mesh Refinement in Object-oriented C++”

▶ ~ 46,000 LOC for C++ SAMR kernel, ~ 140,000 total C++, C, Fortran-77

▶ uses parallel hierarchical data structures that have evolved from DAGH

▶ Right: point explosion in box, 4 level, Euler computation, 7 compute nodes

▶ V1.0: http://amroc.sourceforge.net

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

Comparison of number of cells and grids in DAGH and AMROC
The Virtual Test Facility

- Implements all described algorithms beside multigrid methods
- AMROC V2.0 plus solid mechanics solvers
- Implements explicit SAMR with different finite volume solvers
- Embedded boundary method, FSI coupling
- $\sim 430,000$ lines of code total in C++, C, Fortran-77, Fortran-90
- autoconf / automake environment with support for typical parallel high-performance system
- http://www.cacr.caltech.edu/asc
- [Deiterding et al., 2006][Deiterding et al., 2007]
The AMROC software system

UML design of AMROC

- Classical framework approach with generic main program in C++
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

The AMROC software 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
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
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
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++
Commonalities in software design

- Index coordinate system based on \( \Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa \) to uniquely identify a cell within the hierarchy
Layered software structure

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \frac{l_{\text{max}}}{\prod_{\kappa=\substack{l+1}}^{\infty}} r_\kappa$ to uniquely identify a cell within the hierarchy.

- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap$, $\cup$, $\setminus$. 

The AMROC software system
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \approx \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa$ to uniquely identify a cell within the hierarchy
- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap$, $\cup$, $\setminus$
- Patch<dim,type> class that assigns data to a rectangular grid $G_{m,l}$
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \leq \frac{L_{\text{max}}}{\prod_{\kappa=l+1}^{1+\kappa} r_{\kappa}}$ to uniquely identify a cell within the hierarchy

- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lowerleft, upperright, stepsize and specify topological operations $\cap, \cup, \setminus$

- Patch<dim,type> class that assigns data to a rectangular grid $G_{m,l}$

- A class, here GridFunction<dim,type>, that defines topological relations between lists of Patch objects to implement synchronization, restriction, prolongation, re-distribution
Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \equiv \prod_{\kappa=l+1}^{l_{\text{max}}} r_\kappa$ to uniquely identify a cell within the hierarchy
- Box<dim>, BoxList<dim> class that define rectangular regions $G_{m,l}$ by lower left, upper right, step size and specify topological operations $\cap$, $\cup$, $\setminus$
- Patch<dim,type> class that assigns data to a rectangular grid $G_{m,l}$
- A class, here GridFunction<dim,type>, that defines topological relations between lists of Patch objects to implement synchronization, restriction, prolongation, re-distribution
- Hierarchical parallel data structures are typically C++, routines on patches often Fortran
Embedded boundary method / FSI coupling

- Multiple independent EmbeddedBoundaryMethod objects possible
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes
Embedded boundary method / FSI coupling

- Multiple independent EmbeddedBoundaryMethod objects possible
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes

Coupling algorithm implemented in further derived HypSAMRSolver class
- Level set evaluation always with CPT algorithm
- Parallel communication through efficient non-blocking communication module
- Time step selection for both solvers through CoupledSolver class
Parallelized construction of space-filling curve

Computation of space filling curve

- Partition-Init
Parallelized construction of space-filling curve

Computation of space filling curve

- **Partition-Init**
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
Parallelized construction of space-filling curve

Computation of space filling curve

- **Partition-Init**
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  2. Construct recursively SFC-units until work in each unit is homogeneous, GuCFactor defines minimal coarseness relative to level-0 grid
Parallelized construction of space-filling curve

Computation of space filling curve

- **Partition-Init**
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  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
Parallelized construction of space-filling curve

Computation of space filling curve

- **Partition-Init**
  1. Compute aggregated workload for new grid hierarchy and project result onto level 0
  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
Parallelized construction of space-filling curve

Computation of space filling curve

▸ Partition-Init

1. Compute aggregated workload for new grid hierarchy and project result onto level 0
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
Parallelized construction of space-filling curve

Computation of space filling curve

▶ Partition-Init

1. Compute aggregated workload for new grid hierarchy and project result onto level 0
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
Parallelized construction of space-filling curve

Computation of space filling curve

► Partition-Init

1. Compute aggregated workload for new grid hierarchy and project result onto level 0
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 (can be a bottleneck for weak scalability)
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

[Laurence et al., 2007]
First performance assessment

- 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 $64^3$, $r_{1,2} = 2$, 89 time steps on coarsest level
- With embedded boundary method: 96 time steps on coarsest level
- Redistribute in parallel every 2nd base level step
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells

<table>
<thead>
<tr>
<th>Level</th>
<th>Grids</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>115</td>
<td>262,144</td>
</tr>
<tr>
<td>1</td>
<td>373</td>
<td>1,589,808</td>
</tr>
<tr>
<td>2</td>
<td>2282</td>
<td>5,907,064</td>
</tr>
</tbody>
</table>

Grid and cells used on 16 CPUs
Cost of SAMR and ghost-fluid method

- Flux correction is negligible
- Clustering is negligible (already local approach). For the complexities of a scalable global clustering algorithm see [Gunney et al., 2007]

<table>
<thead>
<tr>
<th>CPUs</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per step</td>
<td>32.44s</td>
<td>18.63s</td>
<td>11.87s</td>
</tr>
<tr>
<td>Uniform</td>
<td>59.65s</td>
<td>29.70s</td>
<td>15.15s</td>
</tr>
<tr>
<td>Integration</td>
<td>73.46%</td>
<td>64.69%</td>
<td>50.44%</td>
</tr>
<tr>
<td>Flux Correction</td>
<td>1.30%</td>
<td>1.49%</td>
<td>2.03%</td>
</tr>
<tr>
<td>Boundary Setting</td>
<td>13.72%</td>
<td>16.60%</td>
<td>20.44%</td>
</tr>
<tr>
<td>Regridding</td>
<td>10.43%</td>
<td>15.68%</td>
<td>24.25%</td>
</tr>
<tr>
<td>Clustering</td>
<td>0.34%</td>
<td>0.32%</td>
<td>0.26%</td>
</tr>
<tr>
<td>Output</td>
<td>0.29%</td>
<td>0.53%</td>
<td>0.92%</td>
</tr>
<tr>
<td>Misc.</td>
<td>0.46%</td>
<td>0.44%</td>
<td>0.47%</td>
</tr>
</tbody>
</table>
Cost of SAMR and ghost-fluid method

- Flux correction is negligible
- Clustering is negligible (already local approach). For the complexities of a scalable global clustering algorithm see [Gunney et al., 2007]
- Costs for GFM constant around $\sim$ 36%
- Main costs: Regrid(1) operation and ghost cell synchronization
AMROC scalability tests

Basic test configuration

- Spherical blast wave, Euler equations, 3D wave propagation method
- AMR base grid $32^3$ with $r_{1,2} = 2, 4$. 5 time steps on coarsest level
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells, 19 time steps
- Flux correction deactivated
- No volume I/O operations
- Tests run IBM BG/P (mode VN)
AMROC scalability tests

Basic test configuration

- Spherical blast wave, Euler equations, 3D wave propagation method
- AMR base grid $32^3$ with $r_{1,2} = 2, 4$. 5 time steps on coarsest level
- Uniform grid $256^3 = 16.8 \cdot 10^6$ cells, 19 time steps
- Flux correction deactivated
- No volume I/O operations
- Tests run IBM BG/P (mode VN)

Weak scalability test

- Reproduction of configuration each 64 CPUs
- On 1024 CPUs: $128 \times 64 \times 64$ base grid, $> 33,500$ Grids, $\sim 61 \cdot 10^6$ cells, uniform $1024 \times 512 \times 512 = 268 \cdot 10^6$ cells

<table>
<thead>
<tr>
<th>Level</th>
<th>Grids</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>606</td>
<td>32,768</td>
</tr>
<tr>
<td>1</td>
<td>575</td>
<td>135,312</td>
</tr>
<tr>
<td>2</td>
<td>910</td>
<td>3,639,040</td>
</tr>
</tbody>
</table>

Strong scalability test

- $64 \times 32 \times 32$ base grid, uniform $512 \times 256 \times 256 = 33.6 \cdot 10^6$ cells

<table>
<thead>
<tr>
<th>Level</th>
<th>Grids</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1709</td>
<td>65,536</td>
</tr>
<tr>
<td>1</td>
<td>1735</td>
<td>271,048</td>
</tr>
<tr>
<td>2</td>
<td>2210</td>
<td>7,190,208</td>
</tr>
</tbody>
</table>
Weak scalability test

![Graph showing time per highest level step with SAMR and Uniform]

- **Time per highest level step**
  - **X-axis:** CPUs (64, 128, 256, 512, 1024)
  - **Y-axis:** sec (0 to 40)
  - **Legend:**
    - SAMR
    - Uniform

![Bar chart showing breakdown of time per step with SAMR]

- **Breakdown of time per step with SAMR**
  - **X-axis:** 64, 128, 256, 512, 1024
  - **Y-axis:** sec (0 to 14)
  - **Legend:**
    - Integration
    - Syncing
    - Partition
    - Recompose
    - Misc

The AMROC software system
Weak scalability test

Costs for Syncing basically constant
Weak scalability test

- Costs for Syncing basically constant
- Partitioning, Recompose, Misc (origin not clear) increase
- 1024 required usage of -DUAL option due to usage of global lists data structures in Partition-Calc and Recompose
Strong scalability test

Time per highest level step

- SAMR
- Uniform

Breakdown of time per step with SAMR

- Integration
- Syncing
- Partition
- Recompose
- Misc

The AMROC software system
Strong scalability test

Uniform code has basically linear scalability (explicit method)

SAMR visibly loses efficiency for > 512 CPUs, or 15,000 finite volume cells per CPU
Strong scalability test - II

Scaling of main operations

Breakdown of time per step with SAMR

The AMROC software system
Strong scalability test - II

Scaling of main operations

- Integration
- Syncing
- Partition
- Recompose
- Misc

Breakdown of time per step with SAMR

Perfect scaling of Integration, reasonable scaling of Syncing
Strong scalability test - II

Scaling of main operations

- Perfect scaling of Integration, reasonable scaling of Syncing
- Strong scalability of Partition needs to be addressed (eliminate global lists)
Available SAMR software

AMROC

Massively parallel SAMR

References

Performance data from AMROC

Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at $t = 0.526408$ s.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).
Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at $t = 0.526408\,\text{s}$.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Re-gridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells ($l_3$: 10.2M, $l_2$: 15.3M, $l_1$: 21.5M, $l_0$: 4.8M) vs. 19.66 billion (uniform).

![Graph showing time per coarse level step vs. CPUs]

<table>
<thead>
<tr>
<th>Cores</th>
<th>48</th>
<th>96</th>
<th>192</th>
<th>288</th>
<th>384</th>
<th>576</th>
<th>768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per step</td>
<td>132.43s</td>
<td>69.79s</td>
<td>37.47s</td>
<td>27.12s</td>
<td>21.91s</td>
<td>17.45s</td>
<td>15.15s</td>
</tr>
<tr>
<td>Par. Efficiency</td>
<td>100.0</td>
<td>94.88</td>
<td>88.36</td>
<td>81.40</td>
<td>75.56</td>
<td>63.24</td>
<td>54.63</td>
</tr>
<tr>
<td>LBM Update</td>
<td>5.91</td>
<td>5.61</td>
<td>5.38</td>
<td>4.92</td>
<td>4.50</td>
<td>3.73</td>
<td>3.19</td>
</tr>
<tr>
<td>Re-gridding</td>
<td>15.44</td>
<td>12.02</td>
<td>11.38</td>
<td>10.92</td>
<td>10.02</td>
<td>8.94</td>
<td>8.24</td>
</tr>
<tr>
<td>Partitioning</td>
<td>4.16</td>
<td>2.43</td>
<td>1.16</td>
<td>1.02</td>
<td>1.04</td>
<td>1.16</td>
<td>1.34</td>
</tr>
<tr>
<td>Interpolation</td>
<td>3.76</td>
<td>3.53</td>
<td>3.33</td>
<td>3.05</td>
<td>2.83</td>
<td>2.37</td>
<td>2.06</td>
</tr>
<tr>
<td>Sync Boundaries</td>
<td>54.71</td>
<td>59.35</td>
<td>59.73</td>
<td>56.95</td>
<td>54.54</td>
<td>52.01</td>
<td>51.19</td>
</tr>
<tr>
<td>Level set</td>
<td>0.78</td>
<td>0.93</td>
<td>1.21</td>
<td>1.37</td>
<td>1.45</td>
<td>1.48</td>
<td>1.47</td>
</tr>
<tr>
<td>Interp./Extrap.</td>
<td>3.87</td>
<td>3.81</td>
<td>3.76</td>
<td>3.49</td>
<td>3.26</td>
<td>2.75</td>
<td>2.39</td>
</tr>
<tr>
<td>Misc</td>
<td>2.27</td>
<td>1.91</td>
<td>1.79</td>
<td>1.67</td>
<td>1.58</td>
<td>1.38</td>
<td>1.25</td>
</tr>
</tbody>
</table>
Strong scalability test - Train side wind computation

- Computation is restarted from disk checkpoint at \( t = 0.526408 \) s.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (\( l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M \)) vs. 19.66 billion (uniform).
- Portions for parallel communication quite considerable (4 ghost cells still used).

![Time per coarse level step graph](image)

**Time in % spent in main operations**

<table>
<thead>
<tr>
<th>Cores</th>
<th>48</th>
<th>96</th>
<th>192</th>
<th>288</th>
<th>384</th>
<th>576</th>
<th>768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per step</td>
<td>132.43s</td>
<td>69.79s</td>
<td>37.47s</td>
<td>27.12s</td>
<td>21.91s</td>
<td>17.45s</td>
<td>15.15s</td>
</tr>
<tr>
<td>Par. Efficiency</td>
<td>100.0</td>
<td>94.88</td>
<td>88.36</td>
<td>81.40</td>
<td>75.56</td>
<td>63.24</td>
<td>54.63</td>
</tr>
<tr>
<td>LBM Update</td>
<td>5.91</td>
<td>5.61</td>
<td>5.38</td>
<td>4.92</td>
<td>4.50</td>
<td>3.73</td>
<td>3.19</td>
</tr>
<tr>
<td>Regridding</td>
<td>15.44</td>
<td>12.02</td>
<td>11.38</td>
<td>10.92</td>
<td>10.02</td>
<td>8.94</td>
<td>8.24</td>
</tr>
<tr>
<td>Partitioning</td>
<td>4.16</td>
<td>2.43</td>
<td>1.16</td>
<td>1.02</td>
<td>1.04</td>
<td>1.16</td>
<td>1.34</td>
</tr>
<tr>
<td>Interpolation</td>
<td>3.76</td>
<td>3.53</td>
<td>3.33</td>
<td>3.05</td>
<td>2.83</td>
<td>2.37</td>
<td>2.06</td>
</tr>
<tr>
<td>Sync Boundaries</td>
<td>54.71</td>
<td>59.35</td>
<td>59.73</td>
<td>56.95</td>
<td>54.54</td>
<td>52.01</td>
<td>51.19</td>
</tr>
<tr>
<td>Level set</td>
<td>0.78</td>
<td>0.93</td>
<td>1.21</td>
<td>1.37</td>
<td>1.45</td>
<td>1.48</td>
<td>1.47</td>
</tr>
<tr>
<td>Interp./Extrap.</td>
<td>3.87</td>
<td>3.81</td>
<td>3.76</td>
<td>3.49</td>
<td>3.26</td>
<td>2.75</td>
<td>2.39</td>
</tr>
<tr>
<td>Misc</td>
<td>2.27</td>
<td>1.91</td>
<td>1.79</td>
<td>1.67</td>
<td>1.58</td>
<td>1.38</td>
<td>1.25</td>
</tr>
</tbody>
</table>
References


References II


