Lecture 4b

Design of SAMR Systems, Advanced Parallelization, Usage

Course *Block-structured Adaptive Mesh Refinement Methods for Conservation Laws*
Theory, Implementation and Application

Ralf Deiterding
Computer Science and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008 MS6367, Oak Ridge, TN 37831, USA

E-mail: deiterdingr@ornl.gov

Outline

Outline

Outline

Outline

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

- 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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

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

Simplified structured designs

Distributed memory parallelization fully supported if not otherwise states.

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

Systems that support general SAMR

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
  - https://computation.llnl.gov/casc/SAMRAI

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
  - https://computation.llnl.gov/casc/SAMRAI
- 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
  - https://ccse.lbl.gov/Software (no codes available)

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
  - https://computation.llnl.gov/casc/SAMRAI
- 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
  - https://ccse.lbl.gov/Software (no codes available)
- Chombo
  - Redesign and extension of BoxLib by P. Colella et al.
  - Both multigrid and explicit algorithms demonstrated
  - Some embedded boundary support
  - https://seesar.lbl.gov/anag/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
  - https://computation.llnl.gov/casc/Overture

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
  - https://computation.llnl.gov/casc/Overture
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://www.clawpack.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
  - https://computation.llnl.gov/casc/Overture
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://www.clawpack.org
- Amrita by J. Quirk
  - Only 2D explicit finite volume methods supported
  - Embedded boundary algorithm
  - http://www.amrita-cfd.org

Further SAMR software

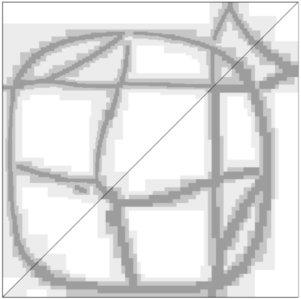
- Overture (Object-oriented tools for solving PDEs in complex geometries)
  - Overlapping meshes for complex geometries by W. Henshaw et al. [Brown et al., 1997]
  - Explicit and implicit algorithms supported
  - https://computation.llnl.gov/casc/Overture
- AMRClaw within Clawpack [Berger and LeVeque, 1998]
  - Serial 2D Fortran 77 code for the explicit Wave Propagation method with own memory management
  - http://www.clawpack.org
- 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++"
- $\sim 46,000$ LOC for C++ SAMR kernel, $\sim 140,000$ total C++, C, Fortran-77
- uses parallel hierarchical data structures that have evolved from DAGH

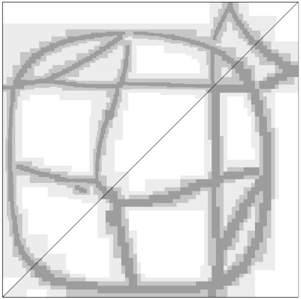
AMROC

- "Adaptive Mesh Refinement in Object-oriented C++"
- $\sim 46,000$ LOC for C++ SAMR kernel, $\sim 140,000$ 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++"
- $\sim 46,000$ LOC for C++ SAMR kernel, $\sim 140,000$ 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



	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

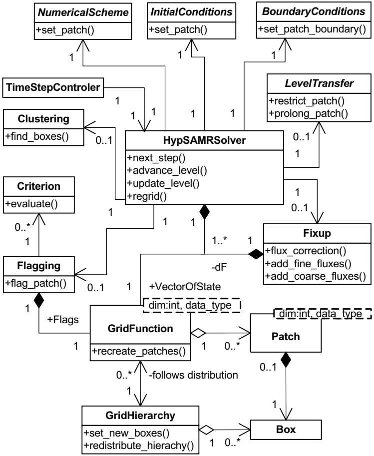
Comparison of number of cells and grids in DAGH and AMROC

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]

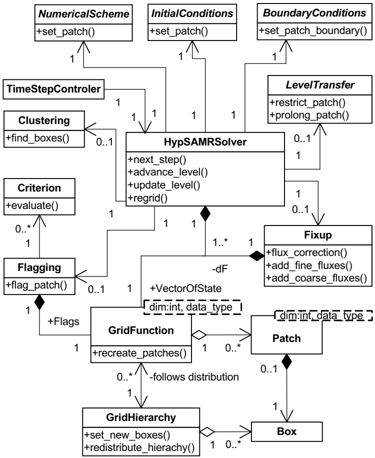
UML design of AMROC

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



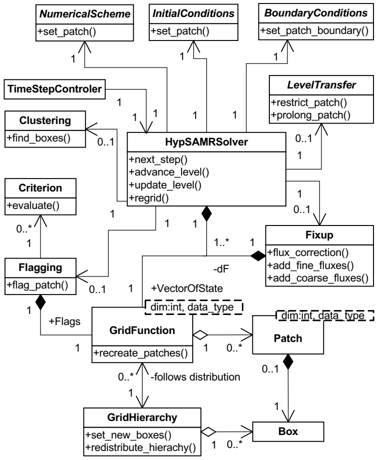
UML design of AMROC

- 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



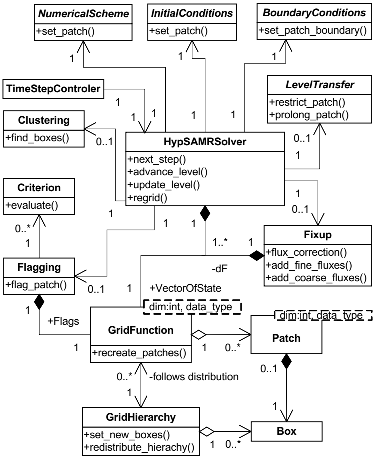
UML design of AMROC

- 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



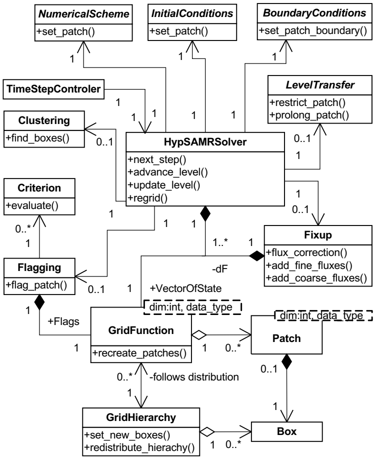
UML design of AMROC

- 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



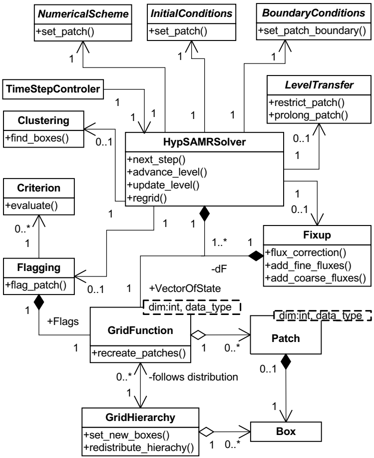
UML design of AMROC

- 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 $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin the hierarchy

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{max}} r_\kappa$ to uniquely identify a cell witin the hierarchy
- `Box<dim>`, `BoxList<dim>` class that define rectangular regions $G_{m,l}$ by `lowerleft`, `upperright`, `stepsize` and specify topological operations $\cap$, $\cup$, $\setminus$

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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

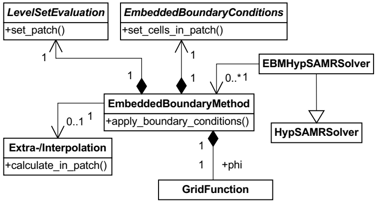
- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{max}} r_\kappa$ to uniquely identify a cell witin the hierarchy
- `Box<dim>`, `BoxList<dim>` class that define rectangular regions $G_{m,l}$ by `lowerleft`, `upperright`, `stepsize` and specify topological operations $\cap$, $\cup$, $\setminus$
- `Patch<dim,type>` class that assigns data to a rectangular grid $G_{m,l}$
- A class, here `GridFunction<dim,type>`, that defines topogical relations between lists of Patch objects to implement sychronization, restriction, prolongation, re-distribution

Commonalities in software design

- Index coordinate system based on $\Delta x_{n,l} \cong \prod_{\kappa=l+1}^{l_{\max}} r_\kappa$ to uniquely identify a cell witin 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 topogical relations between lists of Patch objects to implement sychronization, restriction, prolongation, re-distribution
- Hierarchical parallel data structures are typically C++, routines on patches often Fortran

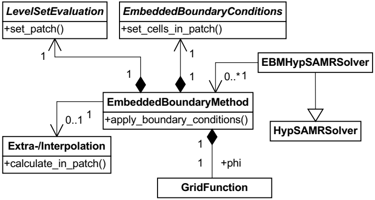
Embedded boundary method / FSI coupling

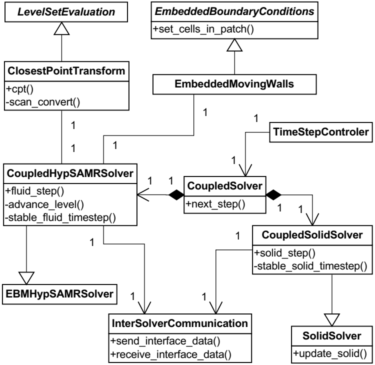
- Multiple independent EmbeddedBoundaryMethod objects possible
- Specialization of GFM boundary conditions, level set description in scheme-specific F77 interface classes



Embedded boundary method / FSI coupling

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





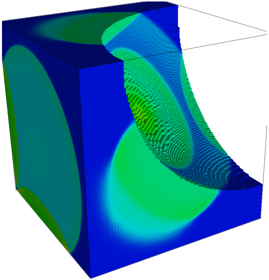
- Coupling algorithm implemented in further derived HypSAMRSolver class
- Level set evaluation always with CPT algorithm
- Parallel communication through efficient non-blocking communication module
- Time step selection for both solvers through CoupledSolver class

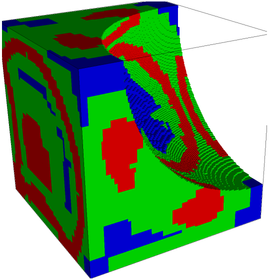
Performance assessment

- 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

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

Grid and cells used on 16 CPUs





Cost of SAMR (AMROC V2.0)

- Flux correction is negligible

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
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%

Cost of SAMR (AMROC V2.0)

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

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
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%

Cost of SAMR (AMROC V2.0)

- 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%

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
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%

CPUs	16	32	64
Time per step	43.97s	25.24s	16.21s
Uniform	69.09s	35.94s	18.24s
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%

Cost of SAMR (AMROC V2.0)

- 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(l)` operation and ghost cell synchronization

CPUs	16	32	64
Time per step	32.44s	18.63s	11.87s
Uniform	59.65s	29.70s	15.15s
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%

CPUs	16	32	64
Time per step	43.97s	25.24s	16.21s
Uniform	69.09s	35.94s	18.24s
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 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

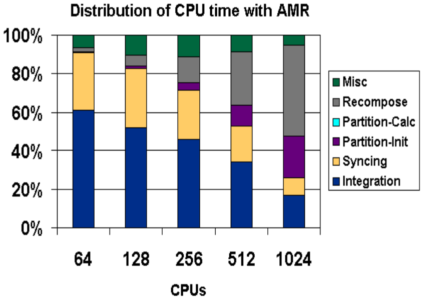
Level	Grids	Cells
0	606	32,768
1	575	135,312
2	910	3,639,040

Strong scalability test

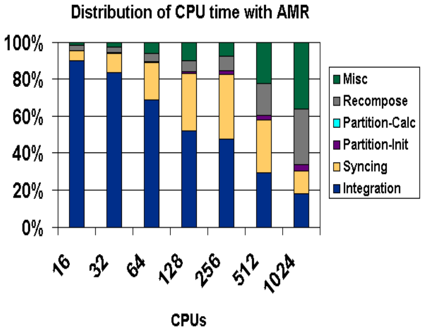
- $64 \times 32 \times 32$ base grid, uniform $512 \times 256 \times 256 = 33.6 \cdot 10^6$ cells

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

Scalability tests AMROC V2.0



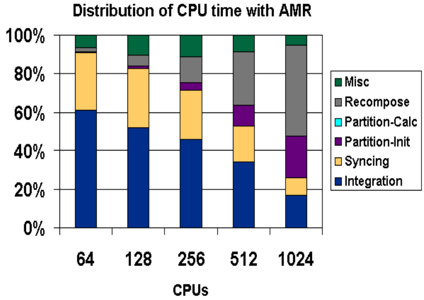
weak scalability test



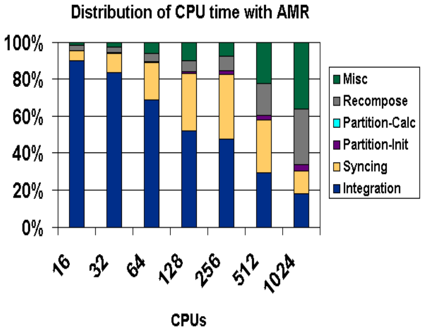
strong scalability test

- Syncing: Parallel communication portion of boundary setting
- Recompose: topological list operations, construction of boundary info, redistribution of data blocks
- Partition-Init, Partition-Calc: construction of space filling curve

Scalability tests AMROC V2.0



weak scalability test



strong scalability test

- Syncing: Parallel communication portion of boundary setting
- Recompose: topological list operations, construction of boundary info, redistribution of data blocks
- Partition-Init, Partition-Calc: construction of space filling curve
- Costs for Partition-Init and Recompose increase dramatically for large problem size

Topological list operations

Weak scalability problems are due to non-parallel sub-algorithms!

Topological list operations

Weak scalability problems are due to non-parallel sub-algorithms!

- Operations $\cap$, $\setminus$ on two box lists have complexity $O(NM)$
- Costs of operations in `Recompose(l)`, that use global box lists G_l, increase quadratically, cf. [Wissink et al., 2003]

Topological list operations

Weak scalability problems are due to non-parallel sub-algorithms!

- Operations $\cap$, $\setminus$ on two box lists have complexity $O(NM)$
- Costs of operations in `Recompose(l)`, that use global box lists G_l, increase quadratically, cf. [Wissink et al., 2003]
- Solution
  - Clip G_l with properly chosen quadratic bounding box around G_l^p before using $\cap$, $\setminus$

Topological list operations

Weak scalability problems are due to non-parallel sub-algorithms!

- Operations $\cap$, $\setminus$ on two box lists have complexity $O(NM)$
- Costs of operations in `Recompose(l)`, that use global box lists G_l, increase quadratically, cf. [Wissink et al., 2003]
- Solution
  - Clip G_l with properly chosen quadratic bounding box around G_l^p before using $\cap$, $\setminus$
- All topological operations in `Recompose(l)` involving global lists can be reduced to local ones

Topological list operations

Weak scalability problems are due to non-parallel sub-algorithms!

- Operations $\cap$, $\setminus$ on two box lists have complexity $O(NM)$
- Costs of operations in `Recompose(l)`, that use global box lists G_l, increase quadratically, cf. [Wissink et al., 2003]
- Solution
  - Clip G_l with properly chosen quadratic bounding box around G_l^p before using $\cap$, $\setminus$
- All topological operations in `Recompose(l)` involving global lists can be reduced to local ones
- Present code V2.1β 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

Construction of space-filling curve

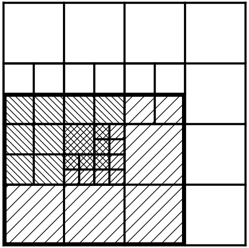
Computation of space filling curve

- Partition-Init

Construction of space-filling curve

Computation of space filling curve

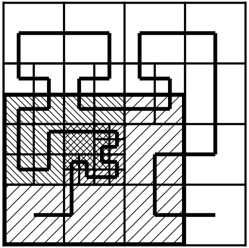
- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l and project result onto level 0



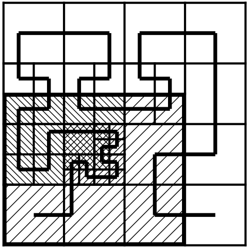
Construction of space-filling curve

Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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



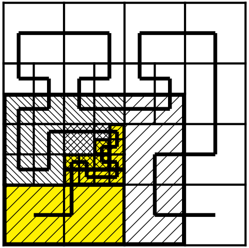
Construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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

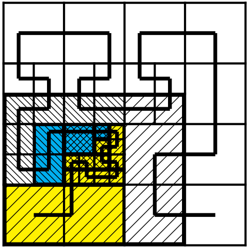
Construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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

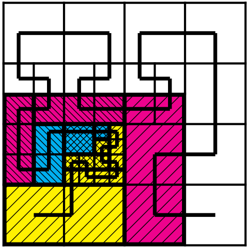
Construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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

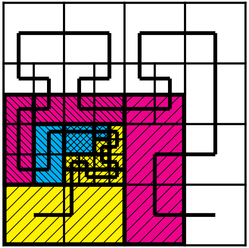
Construction of space-filling curve



Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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

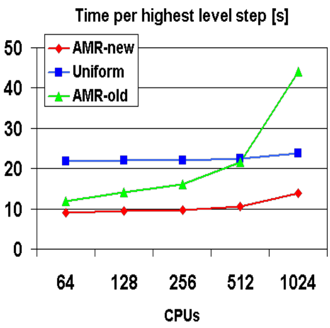
Construction of space-filling curve

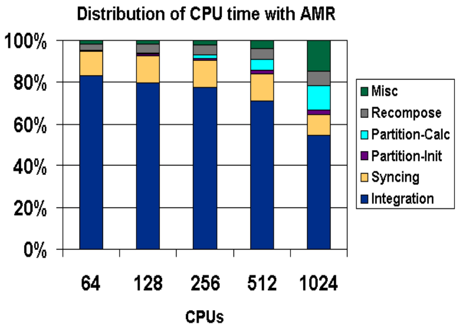


Computation of space filling curve

- Partition-Init
  1. Compute aggregated workload for new grid hierarchy G_l 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

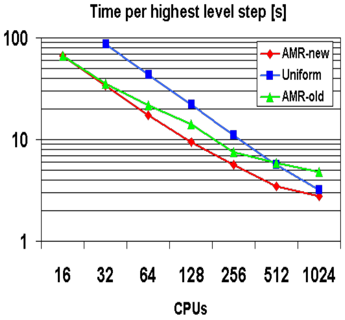
Weak scalability test V2.1β

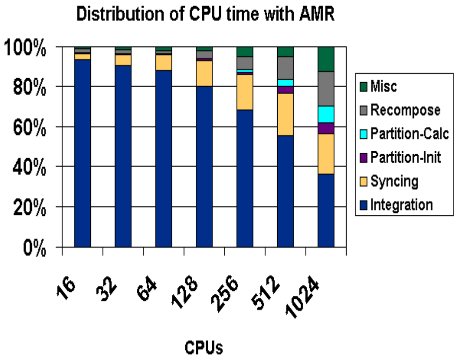




- Overall performance improvement for 1024 CPUs by $\sim 69\,\%$
- Absolute costs for Syncing are almost constant
- 1024 required usage of -DUAL option due to usage of global lists data structures in Partition-Calc and Recompose

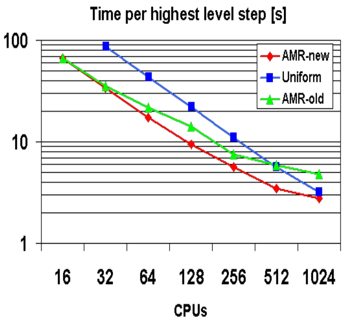
Strong scalability test V2.1β

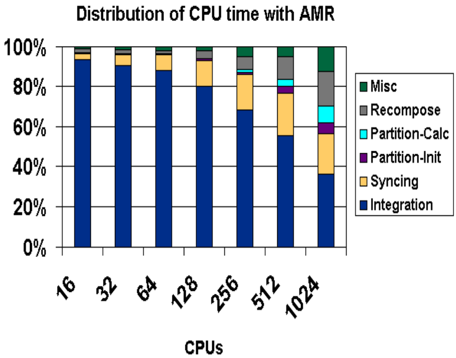




- Overall performance improvement for 1024 CPUs by 43 %

Strong scalability test V2.1β





- 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

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

Next step

- Eliminate aggregation of global box list data (that currently uses simply `MPI_allgather()`)

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

Next step

- Eliminate aggregation of global box list data (that currently uses simply `MPI_allgather()`)
  - Partition-Calc: assignment of SFC-ordered sequence and refinement could be executed sequentially on each node

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

Next step

- Eliminate aggregation of global box list data (that currently uses simply `MPI_allgather()`)
  - Partition-Calc: assignment of SFC-ordered sequence and refinement could be executed sequentially on each node
  - 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

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

Next step

- Eliminate aggregation of global box list data (that currently uses simply `MPI_allgather()`)
  - Partition-Calc: assignment of SFC-ordered sequence and refinement could be executed sequentially on each node
  - 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

Future work

- Large-scale hierarchical I/O

To-Do

Comments

- Scalability of V2.1β for finite volume methods is comparable to results reported from Chombo and SAMRAI
- Significantly better scalability has so far only been reported for shallow hierarchies [Greenough et al., 2005] and/or tailored parameters choices

Next step

- Eliminate aggregation of global box list data (that currently uses simply `MPI_allgather()`)
  - Partition-Calc: assignment of SFC-ordered sequence and refinement could be executed sequentially on each node
  - 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

Future work

- Large-scale hierarchical I/O
- Hybrid parallelization (considering accelerators), cf. [Schive et al., 2010], [Jourdon, 2005]

Quick start with AMROC V2.0 I

Standard Linux development system assumed! See install_vtf.pdf for details.

1. Unpack hdf4_src.tgz into home directory: cd; tar -xvzf hdf4_src.tgz
2. cd asc; ./build_hdf4.sh
3. If last step successful libdf.a, libjpeg.a, libmfhdf.a, libsz.a, libz.a are in $HOME/asc/hdf4/lib.
4. Unpack AMROC-Clawpack-1.0.tgz into $HOME/asc:
   cd $HOME/asc; tar -xvzf AMROC-Clawpack-1.0.tgz
5. With mpicc, mpicxx commands
   5.1 cd vtf
   5.2 ./configure -C --enable-opt=yes --enable-mpi=yes HDF4_DIR=$HOME/asc/hdf4
       If autoconf, automake are available add --enable-maintainer-mode to last line
   5.3 cd gnu-opt-mpi

Quick start with AMROC V2.0 II

5.4 `make`

5.5 `source ../ac/paths.sh`

5.6 Optional unit test

  5.6.1 `../amroc/testrun.sh -m make -r 4 -s`

  5.6.2 `../amroc/testrun.sh -c`

6. Without MPI

6.1 `cd vtf`

6.2 `./configure -C --enable-opt=yes --enable-mpi=no HDF4_DIR=$HOME/asc/hdf4`
If `autoconf`, `automake` are available add
`--enable-maintainer-mode` to last line

6.3 `cd gnu-opt`

6.4 `make`

6.5 `source ../ac/paths.sh`

6.6 Optional unit test

  6.6.1 `../amroc/testrun.sh -m make -r 0 -s`

  6.6.2 `../amroc/testrun.sh -c`

Quick start with AMROC V2.0 III

7. Realistic example
   - 7.1 Change to compilation directory `gnu-opt/gnu-opt-mpi`: `cd amroc/clawpack/applications/euler/2d/SphereLiftOff`
   - 7.2 `make`
   - 7.3 Change to corresponding directory with solver.in: `cd vtf/amroc/clawpack/applications/euler/2d/SphereLiftOff`
   - 7.4 `./run.py` or `./run.py 2` (if you have compiled with MPI on a dual-core system)
   - 7.5 `gnuplot Density.gnu` shows density evolution of lower boundary
   - 7.6 Create binary VTK files for Paraview or VisIt: `hdf2tab.sh "-f display_file_visit.in"`
   - 7.7 Execute VisIt or Paraview and load the VTK files for visualization.

Quick start with AMROC V2.0 IV

8. FSI example (requires MPI)
   - 8.1 `cd $HOME/asc/gnu-opt-mpi/vtf/fsi/beam-amroc/VibratingBeam`
   - 8.2 `make`
   - 8.3 `cd $HOME/asc/vtf/fsi/beam-amroc/VibratingBeam`
   - 8.4 `./run.py 4`
     Change `LastNode` entry in `solver.in` for different processor number.
   - 8.5 `hdf2tab.sh`
   - 8.6 Execute VisIt or Paraview and load the VTK files for visualization.
9. For further documentation see http://www.cacr.caltech.edu/asc

References I

[Berger and LeVeque, 1998] Berger, M. and LeVeque, R. (1998). Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM J. Numer. Anal.*, 35(6):2298–2316.

[Brown et al., 1997] Brown, D. L., Henshaw, W. D., and Quinlan, D. J. (1997). Overture: An object oriented framework for solving partial differential equations. In *Proc. ISCOPE 1997, appeared in Scientific Computing in Object-Oriented Parallel Environments*, number 1343 in Springer Lecture Notes in Computer Science.

[Deiterding et al., 2007] Deiterding, R., Cirak, F., Mauch, S. P., and Meiron, D. I. (2007). A virtual test facility for simulating detonation- and shock-induced deformation and fracture of thin flexible shells. *Int. J. Multiscale Computational Engineering*, 5(1):47–63.

[Deiterding et al., 2006] Deiterding, R., Radovitzky, R., Mauch, S. P., Noels, L., Cummings, J. C., and Meiron, D. I. (2006). A virtual test facility for the efficient simulation of solid materials under high energy shock-wave loading. *Engineering with Computers*, 22(3-4):325–347.

References II

[Gittings et al., 2008] Gittings, M., Weaver, R., Clover, M., Betlach, T., Byrne, N., Coker, R., Dendy, E., Hueckstaedt, R., New, K., Oakes, R., Rantal, D., and Stefan, R. (2008). The RAGE radiation-hydrodynamics code. *Comput. Sci. Disc.*, 1. doi:10.1088/1749-4699/1/1/015005.

[Greenough et al., 2005] Greenough, J. A., de Supinski, B. R., Yates, R. K., Rendleman, C. A., Skinner, D., Beckner, V., Lijewski, M., Bell, J., and Sexton, J. C. (2005). Performance of a block structured, hierarchical adaptive mesh refinement code on the 64k node IBM BlueGene/L computer. Technical Report LBNL-57500, Ernest Orlando Lawrence Berkeley NationalLaboratory, Berkeley.

[Gunney et al., 2007] Gunney, B. T., Wissink, A. M., and Hysoma, D. A. (2007). Parallel clustering algorithms for structured AMR. *J. Parallel and Distributed Computing*, 66(11):1419–1430.

[Hornung et al., 2006] Hornung, R. D., Wissink, A. M., and Kohn, S. H. (2006). Managing complex data and geometry in parallel structured AMR applications. *Engineering with Computers*, 22:181–195.

References III

[Jourdon, 2005] Jourdon, H. (2005). HERA: A hydrodynamic AMR platform for multi-physics simulation. In Plewa, T., Linde, T., and Weirs, V. G., editors, *Adaptive Mesh Refinement - Theory and Applications*, volume 41 of *Lecture Notes in Computational Science and Engineering*, pages 283–294. Springer.

[MacNeice et al., 2000] MacNeice, P., Olson, K. M., Mobarry, C., deFainchtein, R., and Packer, C. (2000). PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126:330–354.

[Parashar and Browne, 1997] Parashar, M. and Browne, J. C. (1997). System engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. In *Structured Adaptive Mesh Refinement Grid Methods*, IMA Volumes in Mathematics and its Applications. Springer.

[Rendleman et al., 2000] Rendleman, C. A., Beckner, V. E., Lijewski, M., Crutchfield, W., and Bell, J. B. (2000). Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Computing and Visualization in Science*, 3:147–157.

References IV

[Schive et al., 2010] Schive, H.-Y., Tsai, Y.-C., and Chiueh, T. (2010). GAMER: a GPU-accelerated adaptive mesh refinement code for astrophysics. *Astrophysical J. Supplement Series*, 186:457–484.

[Wissink et al., 2003] Wissink, A., Hysom, D., and Hornung, R. (2003). Enhancing scalability of parallel structured amr calculations. In *Proc. 17th Int. Conf. Supercomputing*, pages 336–347.