Adaptive Multiscale Methods for the Atmosphere and Ocean

Parallelization and Software Concepts for Tsunami Simulation on Dynamically Adaptive Triangular Grids

M. Bader, A. Breuer, O. Meister, K. Rahnema, M. Schreiber

Cambridge, Aug 24, 2012
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)

(Alexander Breuer)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)
Motivation: Adaptive Mesh Refinement

Scenario for Parallel Adaptive Mesh Refinement

- dynamically adaptive refinement along wave fronts
- adaptive refinement at coast lines (capture small scales) especially during inundation
- re-meshing in every time step (capture shock front)

(Alexander Breuer)
Parallel AMR of Structured Grids

Challenges for Grid Refinement and Coarsening:

- strong, dynamically adaptive refinement (to capture geometry details or numerical shocks, etc.)
- frequent re-meshing of large parts of the grid
- substantial change of problem size during simulation
Challenges for Memory and Cache Efficiency:

- “memory wall”: growth of CPU speed (59% p.a.) vs. memory bandwidth (≈ 23%) and latency (≈ 5%)
- minimise memory requirements and footprint
- retain locality properties (cache efficiency) despite frequent re-meshing
Parallel AMR of Structured Grids

Challenges for Parallelisation:

- trend to multi-/manycore; multiple layers of parallelism
- dynamic load balancing due to remeshing and varying computational load (planned: local time-stepping)
- goal: retain locality properties (partitioning)
Parallel AMR of Structured Grids

Challenges for Software:

- paradigms for data structures, processing of grid & unknowns, etc. has major influence on all previous aspects
- usability, maintainability, flexibility, etc.
- further aspects: material/bathymetry data for adaptive refinement; parallelisation of ensemble runs
Parallel AMR of Structured Grids

Our Approach: Structured Adaptive Triangular Grids

- recursively structured triangular grids (newest vertex bisection)
- element orders defined by Sierpinski space-filling curves
- exploit locality properties for cache and parallel efficiency
- study classical vs. novel parallelisation approaches
Part I

Sierpinski-Order Traversals for AMR
Triangular Meshes Generated by Bisection

- start with an initial triangle
- recursive bisection
Triangular Meshes Generated by Bisection

- start with an initial triangle
- recursive bisection
- described by a corresponding refinement tree
Triangular Meshes Generated by Bisection

- start with an initial triangle
- recursive bisection
- described by a corresponding refinement tree
Triangular Meshes Generated by Bisection

- start with an initial triangle
- recursive bisection
- described by a corresponding refinement tree
- adaptive refinement possible
Triangular Meshes Generated by Bisection

- start with an initial triangle
- recursive bisection
- described by a corresponding refinement tree
- adaptive refinement possible
**Triangular Meshes Generated by Bisection**

- start with an initial triangle
- recursive bisection
- described by a corresponding refinement tree
- adaptive refinement possible
Triangular Meshes Generated by Bisection

Result:

- fully adaptive grid described by a corresponding refinement tree
- tree and grid cells traversed in Sierpinski order
- **minimum memory requirements** (1 bit per tree node?!), but only if all processing is based on traversals
To solve for conforming grids:

- **conforming grids** require synchronised refinement of neighbour cells (multiple traversals to propagate information)
- forward and backward propagation of refinement cascades
- **unknowns on nodes and vertices** are updated within several elements (accumulate contributions!)
The Stack Principle – For Data on Edges
Stream- and Stack-based Processing

- Input stream
- System of stacks
- Element-wise processing
- Output stream
- Add cells for refinement
- Delete cells for coarsening
Part II

Towards Simulation Software

(Oliver Meister)
Layer Concept for Sierpinski Approach

Grid Layer:
- grid generation and modification
- traversal implementation

Hook Layer:
- provide unknowns for local element operators
- implement specific traversal operations for kernel layer

Kernel Layer:
- call volume, skeleton, and boundary kernels
- inspired by DUNE (Bastian et al.)

```python
foreach elem in grid:
  read local data from streams/stacks
  call First_Touch_Hook(elem)
  elem_data = struct( cell , edges, nodes)
  call Element_Hook(elem_data)
  call Last_Touch_Hook(elem)
  write local data from streams/stacks
end for

Element_Hook(elem, data) {
  call Volume_Op(elem,elem_data)

  foreach edge in element edges
    call Skeleton_Op(edge,data,nbdata)
    call Boundary_Op(edge,data)
  end for

  call Post_Volume_Op(elem,elem_data)
}
```
Layer Concept for Sierpinski Approach (2)

Lifetime of Variables:

- unknowns are **persistent**: stored on stacks and streams
- residuals, fluxes, etc. preferably **non-persistent**:
  - created by First Touch Hook
  - destroyed by Last Touch Hook
  → after correction/time step/etc. has been performed
- only stored on colour stacks

Role of Kernels is Critical:

- volume kernel: matrix-free implementation, element operations hard-coded or based on template operators
- edge/skeleton kernels requires additional persistent variables:
  - duplicate unknowns of neighbour elements
  - edge-based unknowns to store fluxes
  (or reduced representation of unknowns on edges)
Part III

Distributed-Memory Parallelisation

(Kaveh Rahnema, Oliver Meister: \textit{sam(oa)}^2)
Space-Filling Curves on Refinement Trees:

- uniform load balancing via Sierpinski curve
- also leads to compact partitions (Hölder continuity)
- robust towards dynamical changes of the grid
Communication via Stacks

During Traversals:

1. On partition boundaries: introduce communication stacks (replace colour stacks)
2. stack property: unknowns of two neighbour partitions stored in opposite order
Communication via Stacks

Between Traversals:

1. Exchange and accumulate unknowns (refinement info, flux component, ...) on communication stacks
2. Call edge/skeleton for partition boundary edges
   (⇝ “dynamic” ghost cell approach, if unknowns are transferred)
Test Scenario: Simple Radial Dam Break

- basic shallow water model
- Finite Volume discretisation, Lax-Friedrich flux → lowest-possible computational load
- dynamically adaptive grid with more than 30 Mio cells (max.)
Test Scenario: Parallel Speedup

- Nehalem Cluster with Infiniband interconnect
- 2× quadcore Intel Nehalem-EP per node
Parallel Speedup of Components

- **Computation**
- **Conformity**
- **Adaptivity**
- **Load-Balance**
Part IV

Shared-Memory Parallelisation

(Martin Schreiber: sierpi)
Subtree-Oriented Parallelisation

Level-oriented breadth-first/depth-first traversal:

- leads to simpler, master-slave structure: parent can split off child subtrees
- considers work load on coarse-grid elements
- stack principle and locality properties still provided by space-filling curve
Split & Join Approach

Load-Balancing Using Split & Join Approach:

- generate much more subtree partitions than cores are available
- large partition will split off right subtree (**split**)
- small sibling partitions may be recombined (**join**)

→ **tolerate imbalances** in computational load per element
→ **tolerate variable number of cores** used for computation
  (consider “writer cores” to write output files, e.g.)
Split & Join Approach

Pros and Cons:

- easier communication structure (synchronisation, e.g.)
- easier load migration
- more difficult to achieve uniform load balance
- requires NUMA-aware scheduling of partitions, e.g.
Shared-Memory: Performance Test
(M. Schreiber, HiPC 2012)

- strong speed-up (approx. 19 mio elements)
on 4 Intel Xeon 10-core CPUs (E7-4850@2.00GHz):
  - slightly better results for TBB (compared to OpenMP)
  - close-to-linear speed-up up to 20 cores
Parallelisation Scenario: Ensemble Run
(M. Schreiber, Multicore Challenge 2012)

- 4 identical simulations started on 40 cores with slight delays
- simulations “compete” over available cores
- scheduling component: IPMO
- problem size varies ⇒ number of cores assigned to each simulation changes dynamically
Part V

Current and Future Work
Adaptive Simulation of Benchmark Problems

- using models and solvers provided by GeoClaw (LeVeque et al.)
- piecewise constant on elements, augmented Riemann solvers
- lots of modelling details (friction, etc.) still missing
- primary goal: obtain realistic scenarios for “ensemble runs” on variable number of cores
Adaptive Simulation of Benchmark Problems

- using models and solvers provided by GeoClaw (LeVeque et al.)
- piecewise constant on elements, augmented Riemann solvers
- lots of modelling details (friction, etc.) still missing
- primary goal: obtain realistic scenarios for “ensemble runs” on variable number of cores
Parallel Server for Bathymetry (and similar) Data

**ASAGI** = “parallel server for adaptive geoinformation”
- hold geodata in patches of Cartesian grids (varying/multiple resolution)
- automatic replication (parallel caching) of data across distributed memory (using MPI one-sided communication)
- access from client (i.e., tsunami simulation software) thus purely local
Structured Adaptive Grids and Sierpinski Curves

Allow parallel AMR with dynamical re-meshing:

- (towards) tsunami simulation on fully adaptive meshes with $> 1$ Mio elements per core
- maintain data locality on all scales (stacks & streams, SFC) for cache efficiency and parallelisation
- space-filling curves provide decent partitioning → but can also help for parallelisation?!

Projects & Software:

- **ASCETE**: simulation of coupled tsunami and earthquake events (together with Martin Käser, Jörn Behrens, Luis Dalguer)
- implement augmented-Riemann-solver models by D. George and R. LeVegue
O. Meister, K. Rahnema and M. Bader: 
*A Software Concept for Cache-Efficient Simulation on Dynamically Adaptive Structured Triangular Grids.*

M. Schreiber, H.-J. Bungartz, M. Bader: 
*Shared Memory Parallelization of Fully-Adaptive Simulations Using a Dynamic Tree-Split and -Join Approach.*

M. Bader, H.-J. Bungartz, M. Schreiber: 
*Invasive Computing Paradigms in High Performance Computing* 
Facing the Multicore-Challenge III, 2012, accepted.