Towards achieving GPU-native adaptive mesh refinement

Ania Brown

Prof Takayuki Aoki
Why AMR?
AMR is not GPU friendly

• Complicated, time varying data structures

• Can you use AMR and keep GPU performance?

My conclusion: yes, but it’s messy
Contents

• Introduction to the algorithm + data structures
• The challenges
• Optimisation possibilities
• The RSE perspective — some lessons learnt
Problem domain

- Stencil calculations on a square structured mesh
- Cell centre values
1) Block structured AMR
1) Block structured AMR
1) Block structured AMR
2) Tree based AMR

Simulation mesh

Refinement representation
2) Tree based AMR

Simulation mesh

Refinement representation
2) Tree based AMR

Simulation mesh

Refinement representation
2) Tree based AMR

Simulation mesh

Refinement representation
2) Tree based AMR

Simulation mesh

Refinement representation
3) Patches + Tree AMR
<table>
<thead>
<tr>
<th>CPU</th>
<th></th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block-structured</td>
<td>Enzo</td>
<td>Octree + patches</td>
</tr>
<tr>
<td></td>
<td>CHOMBO</td>
<td>GAMER (2011)</td>
</tr>
<tr>
<td>Octree</td>
<td></td>
<td>Daino (2016)</td>
</tr>
<tr>
<td>RAMSES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Octree + patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH, using PARAMESH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIRVANA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The AMR algorithm

Initialize data structures

Main loop:

Create halo regions

Update patch values

Refine/coarsen patches

Update neighbour relations

Output values for visualisation
Initialize data structures

Main loop:

Create halo regions

Refine/coarsen patches

Update neighbour relations

Output values for visualisation

Update patch values
Update step

1 CUDA block
Update step

- Tune block size for coalesced access
- Zmarching
Initialize data structures

Main loop:

Create halo regions

Refine/coarsen patches

Update neighbour relations

Output values for visualisation

Update patch values
Initialize data structures

Main loop:

**CPU**
- Initialize data structures
- Update neighbour relations
- Output values for visualisation

**GPU**
- Create halo regions
- Update patch values
- Refine/coarsen patches
Ordering leaves
Hilbert curve

At each step:

• Divide space into 4

• Replace each quadrant with rotated or reflected versions of the original curve

• Connect such that that start and end points remain the same
Rules for refinement

\[ H \leftarrow A \uparrow H \rightarrow H \downarrow B \]
\[ A \leftarrow H \rightarrow A \uparrow A \leftarrow C \]
\[ B \leftarrow C \leftarrow B \downarrow B \rightarrow H \]
\[ C \leftarrow B \downarrow C \leftarrow C \uparrow A \]
Neighbour relations

**Leaf nodes:**
Neighbour indices in each direction
Parent index

**Parent nodes:**
Child indices
Create halo regions

Find correct neighbour node
Create halo regions
Find correct neighbour node
Create halo regions

Find correct neighbour node
Create halo regions

Copy halo values
Interpolating halo values
Interpolating halo values
Interpolating halo values
Interpolating halo values
Interpolating halo values
Reducing halo values
Reducing halo values
Coarsen/refine step

Main loop:

**CPU**
- Update neighbour relations

**GPU**
- Find patches to coarsen/refine
- Refine/coarsen patch values
- Defragment value array
Defragment value array
refine node
Defragment value array

refine node
Defragment value array
refine node
Defragment value array
coarsen node
Defragment value array
coarsen node
Defragment value array

coarsen node
Calculate new defragmented position

- **Input:** for all nodes, flag whether that node is to be refined, coarsened or unchanged

- **Refined:** +3  
  **Coarsened:** -3  
  **Unchanged:** 0

- For each element, sum all preceding elements in the array

- For n nodes, requires n/2 threads and $O(\log_2(n))$ serial steps
Multi-GPU
Load balancing
Boundaries between subdomains
Boundaries between subdomains
Boundaries between subdomains
How to distribute tree
How to distribute tree
Software design

• Code framework — allow user to edit/add functions for initialisation, resolution criterion, stencil calc

• Code generation — annotated regular data structures

• How much to offer? — cell/node centre, interpolation level, stencil type
Software development process

- Unit testing
- Verification
- Profiling
Phase field model of dendritic solidification in a binary alloy

7 refinement levels in quad-tree
Regular mesh

Adaptive mesh
Performance testing for dendritic solidification model

$L = 1.5 \times 10^{-3} m$

$R = 4.5 \times 10^{-4} m$

$\Delta x_{min} = 6 \times 10^{-6} m$

$\Delta x_{max} = 1.2 \times 10^{-5} m$

256 x 256
Performance testing for dendritic solidification model

$L = 1.5 \times 10^{-3} m$

$R = 4.5 \times 10^{-4} m$

$\Delta x_{min} = 1.9 \times 10^{-7} m$

$\Delta x_{max} = 1.2 \times 10^{-5} m$

8192 x 8192
In summary

- Patch based tree-AMR
- For quick gains, offload update step to GPU
- GPU-native version possible — values on GPU, neighbour relations on CPU
- Likely won’t be a one size fits all fix
Governing PDEs for phase field model

\[
\frac{\partial \phi}{\partial t} = -M_\phi \left[ \nabla \cdot (a^2 \nabla \phi) + \frac{\partial}{\partial x} \left( a \frac{\partial a}{\partial \phi_x} |
abla \phi|^2 \right) + \frac{\partial}{\partial y} \left( a \frac{\partial a}{\partial \phi_y} |
abla \phi|^2 \right) \right]
\]

\[
+ \frac{\partial}{\partial z} \left( a \frac{\partial a}{\partial \phi_z} |
abla \phi|^2 \right) - S \Delta T \frac{dp(\phi)}{d\phi} - W \frac{dq(\phi)}{d\phi}
\]

\[
\frac{\partial c}{\partial t} = \nabla \cdot \left[ D_S \phi \nabla c_S + D_L (1 - \phi) \nabla c_L \right]
\]

\( \phi(x, y, t) \): phase

\( c(x, y, t) = (1 - \phi)c_L + \phi c_S \)

\( c_L \): liquid concentration

\( c_S \): solid concentration

\( M_\phi \): mobility

\( a \): interface anisotropy

\( p(\phi) \): interpolating function

\( q(\phi) \): double well function

\( D_S, D_L \): diffusion in solid, liquid

\( S \): entropy of fusion

\( W \): height of double well potential

\( T \): temperature