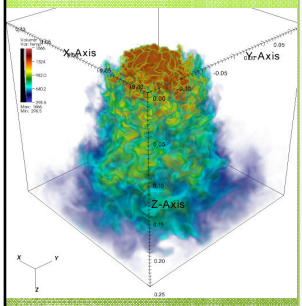
Implementing AMR Grids on a GPU for Massive Parallelization



Lee Allison
(North Georgia College & State University, Dahlonega, GA, 30597)

Dr. Dianna Spence
(North Georgia College & State University, Dahlonega, GA, 30597)

Dr. Bobby Philip
(Oak Ridge National Laboratory, Oak Ridge, TN, 37830)

Image courtesy of http://www.vacet.org/gallery/images_video/AMRVolRend.png

Project Goals

Our main objective was to utilize the resources of the graphical processing unit (GPU) using the CUDA programming language to test whether or not massive parallel processing of AMR grids would improve performance. In order to proceed with this task we first had to gain a basic understanding of the AMR grid system as well as the smoothers/solvers used to find the solutions to elliptic partial differential equations (PDEs)

AMR Grid

Adaptive Mesh Refinement

- Discretizes the domain of a PDE
- Breaks the problem up into a system of linearly independent equations
- Solved using various numerical methods such as Jacobi or Gauss-Seidel

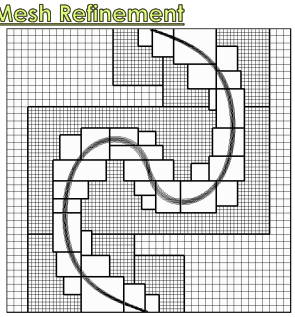
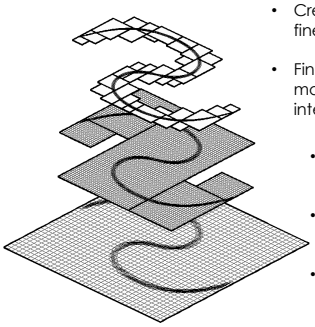


Image courtesy of http://www.amath.washington.edu/~calhoun/research/amr_software/amr_mesh2.png

Mesh Levels



Hierarchy Levels

- Create higher levels for finer refinement
- Finer resolution is much more computation intensive

Patches in Levels

- Can have multiple patches on each level
- Patches can touch but not overlap
- One patch can have multiple parent patches

Image courtesy of http://www.amath.washington.edu/~calhoun/research/amr_software/amr_mesh3.png

Smoothing Methods

Jacobi Method

Matrix representation of System of equations

$$Au = f \Rightarrow \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Decomposition of A into upper, lower, and diagonal matrices

$$A = D - L - U =$$

$$\begin{bmatrix} A_{1,1} & 0 & 0 \\ 0 & A_{2,2} & 0 \\ 0 & 0 & A_{3,3} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ -A_{2,1} & 0 & 0 \\ -A_{3,1} & -A_{3,2} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -A_{1,2} & -A_{1,3} \\ 0 & 0 & -A_{2,3} \\ 0 & 0 & 0 \end{bmatrix}$$

$$Au = f \Rightarrow (D - L - U)u = f$$

Rearrange equation so that an iterative solution may be used

$$u^{k+1} = D^{-1}(L + U)u^k + D^{-1}f$$

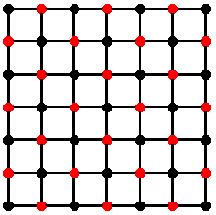
Smoothing Methods

Gauss-Seidel Method

Very similar to Jacobi, only rearrangement is slightly different to effect a faster convergence

$$u^{k+1} = U(D - L)^{-1}u^k + (D - L)^{-1}f$$

Red-Black Gauss-Seidel



- All Red nodes only depend on Black nodes, and vice versa
- Unlike regular Gauss-Seidel (where the (n+1)st node can only be updated after the nth node has been updated) all Red nodes can be updated simultaneously, then all the Black nodes can update using the updated Red values
- Easily parallelizable

Image courtesy of http://www.cs.berkeley.edu/~demmel/cs267/lecture24/RedBlack.gif

Serial vs Parallelized Code

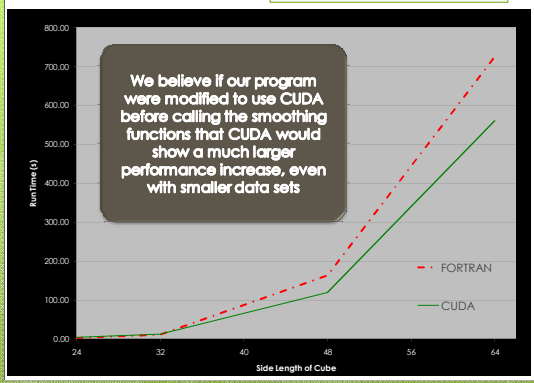
```
for(k=…; k<=…; k++)
   for(j=…; j<=…; j++)
      for(i=…; i<=…; i++)
         for(s=…; s<=…; s++)
         {
            … … …
         }
```

C code
All cells/patches/levels are handled by the CPU one at a time

CUDA code
Each called thread handles one cell in the AMR grid

```
if(Color condition is met)
   for(s=…; s<=…; s++)
   {
      … … …
   }
```

Cube Size vs Time



Cells vs Time