An MPI + GPU Implementation Case Study

Finite-Difference Time-Domain Electromagnetic Field Simulation

John Rugis
Agenda

• Electromagnetic Simulation: Background
• Implementation on UoA HPC
  – Architect the solution
  – Tool selection
  – Code design
  – Test
  – Profile
• Opportunities and Future Work
• Conclusions and Discussion
Electromagnetic Simulation

Maxwell's equations (curl):

\[ \varepsilon \frac{\partial E}{\partial t} = (\nabla \times H) - \sigma E \]

\[ \mu \frac{\partial H}{\partial t} = - (\nabla \times E) - \sigma_m H \]
Electromagnetic Simulation

Numerical Methods

Finite-Difference vs Finite Element

The Finite-Difference Time-Domain method

• 6 field values (3 electric + 3 magnetic)
• 12 material values
• 18 total values

“Yee” cell contains one each of all values
Electromagnetic Simulation

A single Yee cell (interlocked electric and magnetic fields):
Electromagnetic Simulation

Fill model space with indexed Yee cells.
Electromagnetic Simulation

The field update code:

\[ n = i + j \times sX + k \times sXY \]

\[
\begin{align*}
c[n].ex &= c[n].cexe \times c[n].ex \\
          &\quad + c[n].cexh \times ((c[n].hz - c[n - sX].hz) - (c[n].hy - c[n - sXY].hy)) \\
c[n].ey &= c[n].ceye \times c[n].ey \\
          &\quad + c[n].ceyh \times ((c[n].hx - c[n - sXY].hx) - (c[n].hz - c[n - 1].hz)) \\
c[n].ez &= c[n].ceze \times c[n].ez \\
          &\quad + c[n].cezh \times ((c[n].hy - c[n - 1].hy) - (c[n].hx - c[n - sX].hx)) \\
c[n].hx &= c[n].chxh \times c[n].hx \\
          &\quad + c[n].chxe \times ((c[n + sXY].ey - c[n].ey) - (c[n + sX].ez - c[n].ez)) \\
c[n].hy &= c[n].chyh \times c[n].hy \\
          &\quad + c[n].chye \times ((c[n + 1].ez - c[n].ez) - (c[n + sXY].ex - c[n].ex)) \\
c[n].hz &= c[n].chzh \times c[n].hz \\
          &\quad + c[n].chze \times ((c[n + sX].ex - c[n].ex) - (c[n + 1].ey - c[n].ey))
\end{align*}
\]
Electromagnetic Simulation

The field update boundary problem:

Highest h-field update needs e-field from outside the model space.

Lowest e-field update needs h-field from outside the model space.
Electromagnetic Simulation

The Goal: 1,000,000,000 Yee cells.

Memory requirement (double precision): $18 \times 8 \times 1\text{GB} = 144\text{GB}$
UoA HPC - hardware

Many nodes with (12x) CPU cores each:

(2x) CPU
95GB RAM
HPC slot

(12x) GPU nodes available:

(2x) CPU
95GB RAM
HPC GPU slot

| GPU 6GB RAM | GPU 6GB RAM |

We mostly want the GPU's and memory...
UoA HPC - implementation

Top-down approach: Use (8x) GPU nodes.

*OpenMPI for inter-node communication.*

(8x) MPI Model spaces
500 x 500 x 500 Yee cells each
UoA HPC - implementation

Top-down approach: (2x) GPU's in each of the (8x) MPI spaces.

*NVIDIA CUDA for GPU coding and communication.*
1) GPU's can only access their own memory for computation.

2) Memory requirement for each MPI space:
   \[ 18 \times 8 \times 500 \times 500 \times 500 = 18\text{GB} \]

But we only have (2x) 6GB = 12GB available GPU memory.

The GPU model space can't fill the target MPI model space.

(Consider decreasing the model size goal?)
Software Tool Selection

• Build code
  – gcc (c++)
  – mpicxx
  – nvcc (CUDA)
  – make

• IDE
  – nsight (edit, debug, profile)

• Test and Debug
  – python (data conversion)
  – voreen (data visualization)
  – nvprof (CUDA profiling)

Hardware Platforms

• UoA HPC: job queue
  – large models
  – more than (2x) GPU's
  – not interactive

• UoA HPC: gpu build node
  – interactive edit, debug, profile
  – (2x) GPU's

• Local workstation
  – interactive edit, debug, profile
  – (1x) GPU
  – data analysis
Code Design

Start with working reference code.
Test and Debug

Data visualization.

Color and transparency settings

Start with non-MPI reference output.
Test and Debug

Scale-up model space.

fdtd_100_120_ez_1x

fdtd_200_120_ez_1x
Example design consideration:

Q. How to handle shared boundary with adjacent spaces?

A. Surround space with “halo” and do face swapping.
Code Design - MPI

Example: MPI – swap shared faces.

```c
// exchange shared face data between nodes
// notes:
// 1) non-blocking used to "bypass buffering"
// 2) implements two exchange types: low-to-high nodes & high-to-low nodes
void exchange_faces(CModel3D *model, bool lo2hi){
    MPI_Status mpi_status;
    MPI_Request mpi_request[3] = {NULL, NULL, NULL};
    for(int face = 0; face < 3; face++) { // receive up to three
        if(lo2hi && (commRank < mesh[commRank][face])) continue; // lo2hi (don't receive from higher)
        if(!lo2hi && (commRank > mesh[commRank][face])) continue; // hi2lo (don't receive from lower)
        MPI_CHECK(MPI_irecv(
            model->share_in_face[face], model->size_face, MPI_F_TYPE,
            mesh[commRank][face], 0, MPI_COMM_WORLD, &mpi_request[face]));
    }
    for(int face = 0; face < 3; face++) { // send up to three
        if(lo2hi && (commRank > mesh[commRank][face])) continue; // lo2hi (don't send to lower)
        if(!lo2hi && (commRank < mesh[commRank][face])) continue; // hi2lo (don't send to higher)
        MPI_CHECK(MPI_Send(
            model->share_out_face[face], model->size_face, MPI_F_TYPE,
            mesh[commRank][face], 0, MPI_COMM_WORLD));
    }
    for(int face = 0; face < 3; face++) { // wait until all received
        if(mpi_request[face] != NULL) MPI_CHECK(MPI_Wait(&mpi_request[face], &mpi_status));
    }
}
```
Test and Debug - MPI

MPI bug and fixed.
Test and Debug - MPI

Data conversion: non-MPI vs MPI.
Code Design - CUDA

• CUDA challenges
  – Efficient data transfer between host (CPU) and device (GPU) only in large contiguous blocks to device global memory.
  – Device global memory is the only large memory space on device.
  – CUDA scheduler “behind curtain”.

• CUDA paradigm features
  – Very small granularity of parallel kernels.
  – Informative profiling.
CUDA capabilities

Device 1: "Tesla M2090"
CUDA Driver Version / Runtime Version: 5.0 / 5.0
CUDA Capability Major/Minor version number: 2.0
Total amount of global memory: 5375 MBytes (5636554752 bytes)
(16) Multiprocessors X (32) CUDA Cores/MP: 512 CUDA Cores
GPU Clock rate: 1301 MHz (1.30 GHz)
Memory Clock rate: 1848 Mhz
Memory Bus Width: 384-bit
L2 Cache Size: 786432 bytes
Max Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers 1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 32768
Warp size: 32
Maximum number of threads per multiprocessor: 1536
Maximum number of threads per block: 1024
Maximum sizes of each dimension of a block: 1024 x 1024 x 64
Maximum sizes of each dimension of a grid: 65535 x 65535 x 65535
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Concurrent copy and kernel execution: Yes with 2 copy engine(s)
Run time limit on kernels: No
Integrated GPU sharing Host Memory: No
Support host page-locked memory mapping: Yes
Alignment requirement for Surfaces: Yes
Device has ECC support: Enabled
Device supports Unified Addressing (UVA): Yes
Device PCI Bus ID / PCI location ID: 21 / 0
Compute Mode: < Exclusive Process (many threads in one process is able to use ::cudaSetDevice() with this device) >
Code Design - CUDA

CUDA “threads” and “blocks”.

```
void CCudaEH3d::update_e()
{
    const dim3 threads(CUDA_TX, CUDA_TY, CUDA_TZ);
    const dim3 blocks(CUDA_BX, CUDA_BY, CUDA_BZ);

    cudaSetDevice(0);

    update_e_Kernel<<<blocks, threads>>>(
        (d0[ex], d0[ey], d0[ez], d0[hx], d0[hy], d0[hz],
        d0[cexe], d0[ceye], d0[ceze], d0[cehx], d0[cehy], d0[cezh], sX, sXY);
}
```

Example settings:

- threads = 10 x 10 x 10
- blocks = 42 x 42 x 21
- GPU space = 418 x 418 x 208

GPU blocks and halo
Example: CUDA kernel – update e-field.

```c
void update_e_Kernel(
    F_TYPE *exG, F_TYPE *eyG, F_TYPE *ezG,
    const F_TYPE *hxG, const F_TYPE *hyG, const F_TYPE *hzG,
    const F_TYPE *cexeG, const F_TYPE *ceyeG, const F_TYPE *cezeG,
    const F_TYPE *cexhG, const F_TYPE *ceyhG, const F_TYPE *cezhG,
    const size_t sX, const size_t sXY)
{
    const size_t n =
        (((blockIdx.z * blockDim.z) + threadIdx.z) * gridDim.x * blockDim.x * gridDim.y * blockDim.y)
        + (((blockIdx.y * blockDim.y) + threadIdx.y) * gridDim.x * blockDim.x)
        + (blockIdx.x * blockDim.x) + threadIdx.x;

    // skip halo
    if (blockIdx.x == 0 and threadIdx.x == 0) or
        (blockIdx.y == 0 and threadIdx.y == 0) or
        (blockIdx.z == 0 and threadIdx.z == 0) or
        (blockIdx.x == (gridDim.x - 1) and threadIdx.x == (blockDim.x - 1)) or
        (blockIdx.y == (gridDim.y - 1) and threadIdx.y == (blockDim.y - 1)) or
        (blockIdx.z == (gridDim.z - 1) and threadIdx.z == (blockDim.z - 1))) return;

    exG[n] = cexxeG[n] * exG[n]
        + cexhG[n] * ((hzG[n] - hzG[n - sX]) - (hyG[n] - hyG[n - sXY]));
    eyG[n] = ceyeE[n] * eyG[n]
        + ceyhG[n] * ((hxG[n] - hxG[n - sXY]) - (hzG[n] - hzG[n - 1]));
    ezG[n] = cezeE[n] * ezG[n]
```
Test and Debug - CUDA

CUDA testing and final.

fdtd_30_16_ex_CUDA_TEST_1x

fdtd_30_16_ez_CUDA_TEST_1x

fdtd_200_120 ez 8x CUDA
Profiling - CUDA

```
[jrug001@build-gpu-p FDTD_02]$ nvprof ./FDTD_02 430 2 xMPI CUDA x2ND_CUDA xSAVE_FIELD
======== NVPROF is profiling FDTD_02...
======== Command: FDTD_02 430 2 xMPI CUDA x2ND_CUDA xSAVE_FIELD
host: build-gpu-p

CUDA device count: 2
    device: Tesla M2090
    capability: 2.0
    driver: 5.0
    runtime: 5.0
    pci bus/location: 20/0
    global memory: 5636554752
    shared memory/block: 49152
    integrated host memory: No
    map host memory: Yes
    unified addressing: Yes
    multiprocessor count: 16
    clock rate (kHz): 1301000
        device: Tesla M2090
    capability: 2.0
    driver: 5.0
    runtime: 5.0
    pci bus/location: 21/0
    global memory: 5636554752
    shared memory/block: 49152
    integrated host memory: No
```
Profiling - CUDA

```
integrated host memory: No
map host memory: Yes
unified addressing: Yes
multiprocessor count: 16
clock rate (kHz): 1301000
Memory device/free/total: 0/218824704/5636554752
rank/time: 0/29
====== Profiling result:
<table>
<thead>
<tr>
<th>Time</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>70.07</td>
<td>29.79</td>
<td>350.52</td>
<td>49.37</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>14.62</td>
<td>93.25</td>
<td>92.81</td>
<td>93.69</td>
<td>update_h_Kernel(double*, double*, double*,</td>
</tr>
<tr>
<td>14.47</td>
<td>92.28</td>
<td>91.38</td>
<td>93.19</td>
<td>update_e_Kernel(double*, double*, double*,</td>
</tr>
<tr>
<td>0.40</td>
<td>422.00</td>
<td>316.50</td>
<td>632.81</td>
<td></td>
</tr>
<tr>
<td>0.09</td>
<td>557.23</td>
<td>553.57</td>
<td>560.88</td>
<td></td>
</tr>
<tr>
<td>0.09</td>
<td>552.24</td>
<td>549.86</td>
<td>554.62</td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td>424.01</td>
<td>423.38</td>
<td>424.65</td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td>420.29</td>
<td>420.02</td>
<td>420.55</td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>172.34</td>
<td>171.85</td>
<td>172.83</td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>171.72</td>
<td>171.62</td>
<td>171.83</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>102.20</td>
<td>100.99</td>
<td>103.41</td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>102.18</td>
<td>99.72</td>
<td>104.63</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>88.18</td>
<td>88.13</td>
<td>88.23</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>87.45</td>
<td>87.21</td>
<td>87.69</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>60.43</td>
<td>60.18</td>
<td>60.69</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>57.73</td>
<td>56.96</td>
<td>58.51</td>
<td></td>
</tr>
</tbody>
</table>
```

Profiling - CUDA

```
jrug001@build-gpu-p:~/CUDA/nsight/FDTD_02$ ./FDTD_02 430 2 xMPI xCUDA x2ND_CUDA xSAVE_FIELD
host: build-gpu-p

E-field update time: 4944.26ms
E-field update time: 4780.29ms
rank/time: 0/29

[jrug001@build-gpu-p FDTD_02]$ 
```
Profiling - CUDA

CUDA device count: 2

device: Tesla M2090
capability: 2.0
driver: 5.0
runtime: 5.0
pci bus/location: 20/0
global memory: 5636554752
shared memory/block: 49152
integrated host memory: No
map host memory: Yes
unified addressing: Yes
multiprocessor count: 16
clock rate (kHz): 1301000
device: Tesla M2090
capability: 2.0
driver: 5.0
runtime: 5.0
pci bus/location: 21/0
global memory: 5636554752
shared memory/block: 49152
integrated host memory: No
map host memory: Yes
unified addressing: Yes
multiprocessor count: 16
clock rate (kHz): 1301000

Memory device/free/total: 0/219471872/5636554752
E-field update time: 117.805ms
E-field update time: 117.463ms
rank/time: 0/29
[jrug001@build-gpu-p FDTD_02]$
Profiling - CUDA

Raw speedup ratio: 4.8 / 0.117 = 41

But the GPU space didn't quite fill the model space:

\[
\frac{\text{GPU space}}{\text{model space}} = \frac{2 \times 418 \times 418 \times 208}{430 \times 430 \times 430} = 91\%
\]

And this assumes two GPU's running concurrently...
Code Design - CUDA

Concurrency and object-oriented design.

```c
F_TYPE *pf;
if(use_cuda0) cuda3d0->update_e(); // update e-field in cuda
if(use_cuda1) cuda3d1->update_e(); // update e-field in cuda

if(not use_cuda0 and not use_cuda1) {
    update_e_block(1, sx - 1, 1, sy - 1, 1, sz - 1);
}
else {
    update_e_block(1, cuda_min_x, cuda_min_y, cuda_max_y + 1, 1, sz - 1); // 0
    update_e_block(cuda_max_x + 1, sx - 1, cuda_min_y, cuda_max_y + 1, 1, sz - 1); // 1
    update_e_block(cuda_min_x, cuda_max_x + 1, cuda_min_y, cuda_max_y + 1, 1, cuda_min_z0); // 2
    update_e_block(cuda_min_x, cuda_max_x + 1, cuda_min_y, cuda_max_y + 1, cuda_max_z1 + 1, sz - 1); // 3
    update_e_block(1, sx - 1, 1, cuda_min_y, 1, sz - 1); // 4
    update_e_block(1, sx - 1, cuda_max_y + 1, sy - 1, 1, sz - 1); // 5
```
Test and Debug - CUDA

Create and use timer object.

Without CUDA.

<table>
<thead>
<tr>
<th>Step</th>
<th>Time (sec)</th>
<th>A</th>
<th>B</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>9.19e+00</td>
<td>0.44e+00</td>
<td>4.40e+00</td>
<td>1.43e-01</td>
</tr>
<tr>
<td>5</td>
<td>9.20e+00</td>
<td>0.44e+00</td>
<td>4.40e+00</td>
<td>1.23e-01</td>
</tr>
<tr>
<td>0</td>
<td>9.21e+00</td>
<td>0.44e+00</td>
<td>4.34e+00</td>
<td>2.06e-01</td>
</tr>
<tr>
<td>2</td>
<td>9.21e+00</td>
<td>0.45e+00</td>
<td>4.52e+00</td>
<td>2.81e-03</td>
</tr>
<tr>
<td>4</td>
<td>9.21e+00</td>
<td>0.45e+00</td>
<td>4.51e+00</td>
<td>2.60e-02</td>
</tr>
<tr>
<td>3</td>
<td>9.21e+00</td>
<td>0.45e+00</td>
<td>4.35e+00</td>
<td>1.86e-01</td>
</tr>
<tr>
<td>1</td>
<td>9.21e+00</td>
<td>0.43e+00</td>
<td>4.35e+00</td>
<td>3.30e-01</td>
</tr>
<tr>
<td>6</td>
<td>9.20e+00</td>
<td>0.45e+00</td>
<td>4.33e+00</td>
<td>1.91e-01</td>
</tr>
</tbody>
</table>

With CUDA.

<table>
<thead>
<tr>
<th>Step</th>
<th>Time (sec)</th>
<th>A</th>
<th>B</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8.81e-01</td>
<td>3.93e-01</td>
<td>4.31e-01</td>
<td>3.44e-02</td>
</tr>
<tr>
<td>5</td>
<td>8.80e-01</td>
<td>3.78e-01</td>
<td>4.00e-01</td>
<td>4.17e-02</td>
</tr>
<tr>
<td>0</td>
<td>8.79e-01</td>
<td>4.24e-01</td>
<td>3.36e-01</td>
<td>4.37e-02</td>
</tr>
<tr>
<td>2</td>
<td>8.81e-01</td>
<td>4.53e-01</td>
<td>3.66e-01</td>
<td>1.64e-02</td>
</tr>
<tr>
<td>3</td>
<td>8.80e-01</td>
<td>3.72e-01</td>
<td>3.90e-01</td>
<td>3.47e-02</td>
</tr>
<tr>
<td>4</td>
<td>8.79e-01</td>
<td>4.19e-01</td>
<td>3.40e-01</td>
<td>1.08e-01</td>
</tr>
<tr>
<td>1</td>
<td>8.79e-01</td>
<td>3.75e-01</td>
<td>3.85e-01</td>
<td>9.66e-02</td>
</tr>
<tr>
<td>6</td>
<td>8.81e-01</td>
<td>4.22e-01</td>
<td>3.53e-01</td>
<td>4.08e-02</td>
</tr>
</tbody>
</table>

Actual speedup: \( \frac{9.2}{0.88} = 10.5 \)
Test and Debug

Extracted slice from large model simulation.
Runtime: 550 steps at approximately 1 second per step.
Single frame output file size: $848^3 \times 6 \times 8 = 29.3\text{GB}$
Opportunities and Future Work

• **Additional Performance tuning**
  - Shared device memory?
  - Loop optimization?
  - Concurrent MPI?
  - CPU threads?

• **Simulation features**
  - Load / save material & field values
  - Boundary conditions
  - Special material properties
  - ...
Conclusions and Discussion

- MPI + GPU code functional on UoA HPC.
- ~10x speedup with GPU's.
- Implementation: non-blocking MPI.
- Implementation: CUDA usage
  - Blocks and threads in cubes to minimize surface area.
  - Device global memory (built-in L1 cache), not device shared.
  - Concurrency using asynchronous calls and streams.
- Object oriented code design.
- Visual evaluation of output data to debug code.
- How might this experience apply to other applications?