

IESE
Institute of
Earth Science
and Engineering
Aotearoa

eResearch NZ 2013

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):

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

$$\mu \frac{\partial \mathbf{H}}{\partial t} = -(\nabla \times \mathbf{E}) - \sigma_m \mathbf{H}$$

Electromagnetic Simulation

Numerical Methods

Finite-Difference vs Finite Element

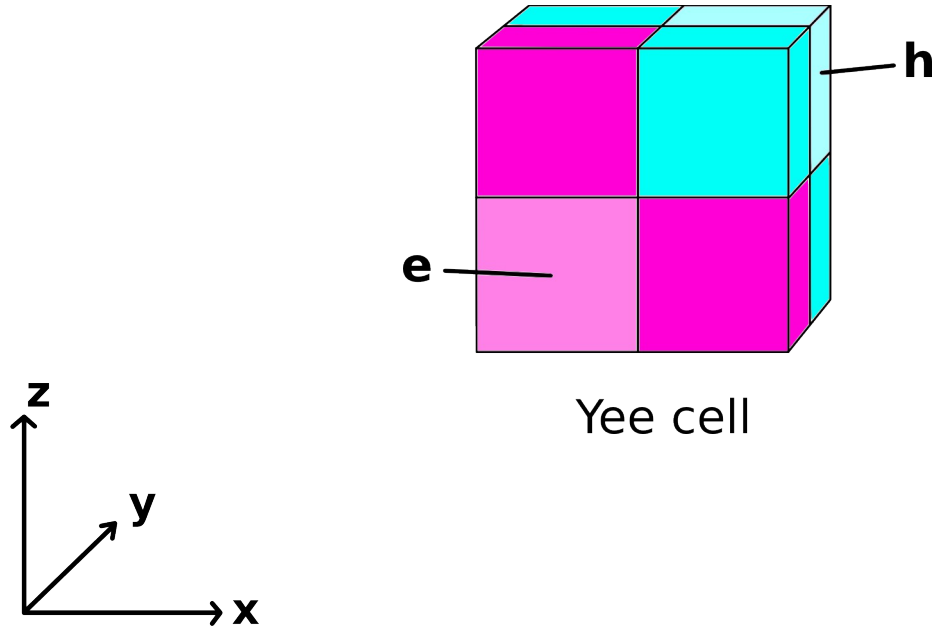
The Finite-Difference Time-Domain method

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

“Yee” cell contains one each of all values

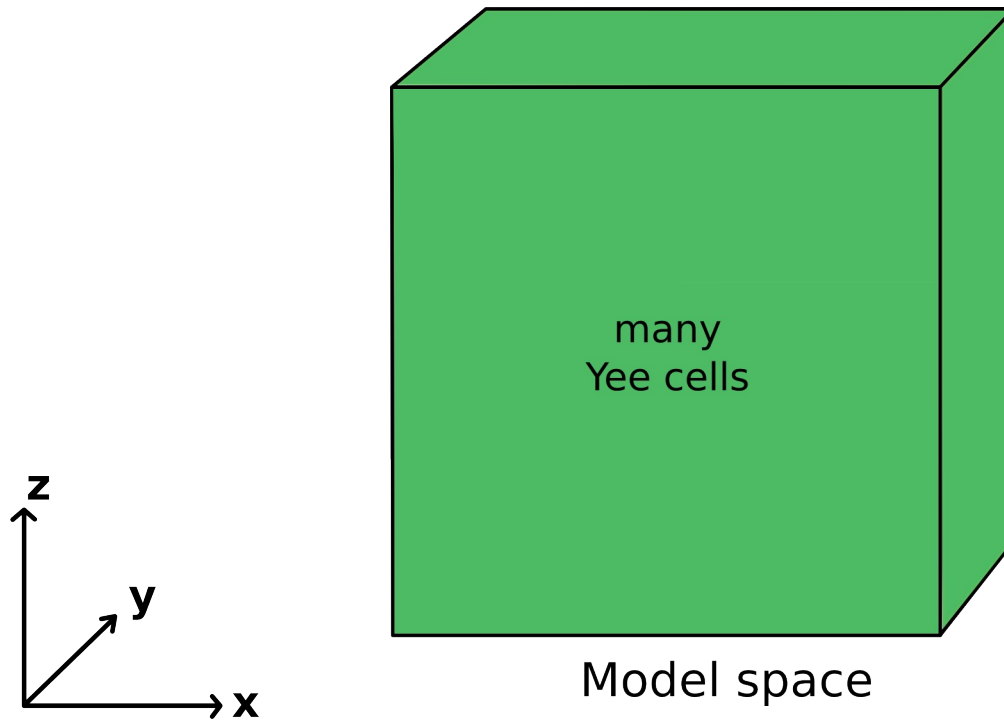
Electromagnetic Simulation

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 * sX + k * sXY
```

```
c[n].ex = c[n].cexe * c[n].ex  
          + c[n].cexh * ((c[n].hz - c[n - sX].hz) - (c[n].hy - c[n - sXY].hy))
```

```
c[n].ey = c[n].ceye * c[n].ey  
          + c[n].ceyh * ((c[n].hx - c[n - sXY].hx) - (c[n].hz - c[n - 1].hz))
```

```
c[n].ez = c[n].ceze * c[n].ez  
          + c[n].cezh * ((c[n].hy - c[n - 1].hy) - (c[n].hx - c[n - sX].hx))
```

```
c[n].hx = c[n].chxh * c[n].hx  
          + c[n].chxe * ((c[n + sXY].ey - c[n].ey) - (c[n + sX].ez - c[n].ez))
```

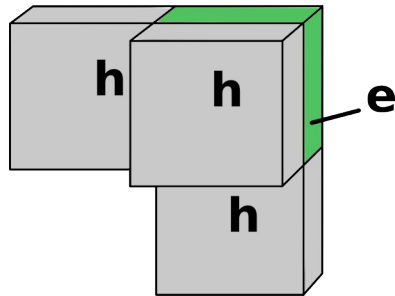
```
c[n].hy = c[n].chyh * c[n].hy  
          + c[n].chye * ((c[n + 1].ez - c[n].ez) - (c[n + sXY].ex - c[n].ex))
```

```
c[n].hz = c[n].chzh * c[n].hz  
          + c[n].chze * ((c[n + sX].ex - c[n].ex) - (c[n + 1].ey - c[n].ey))
```

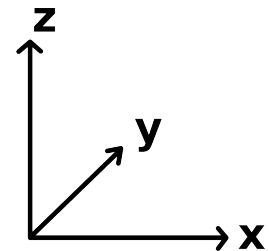
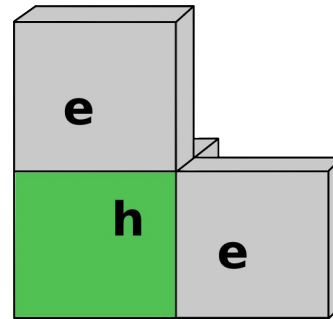
Electromagnetic Simulation

**The field update
boundary problem:**

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

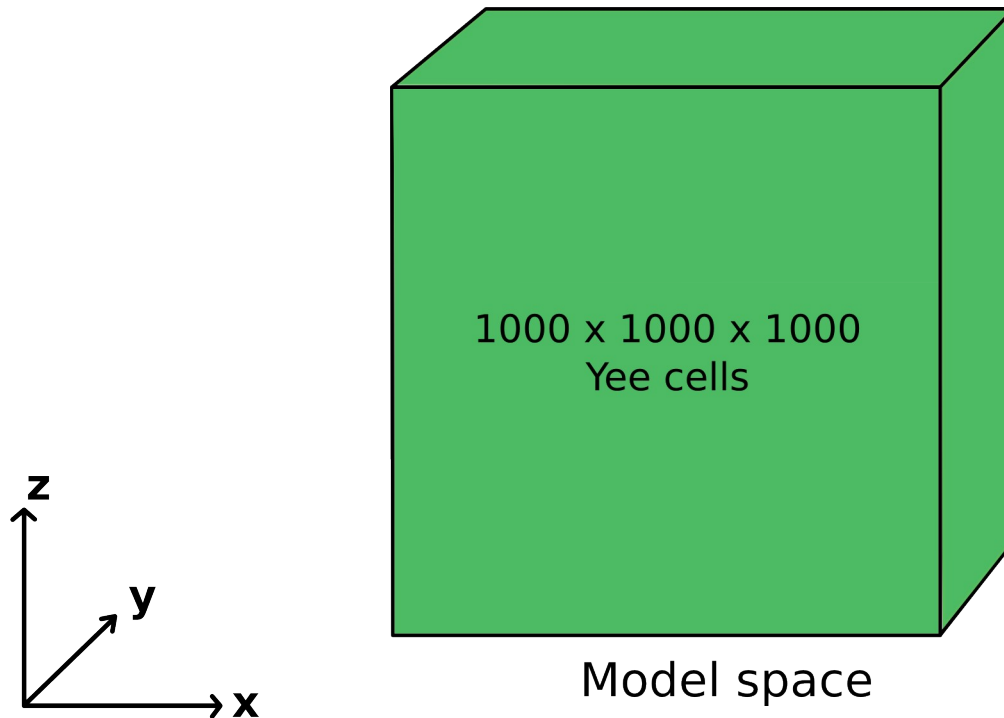


Lowest e-field update needs h-field
from outside the model space.



Electromagnetic Simulation

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



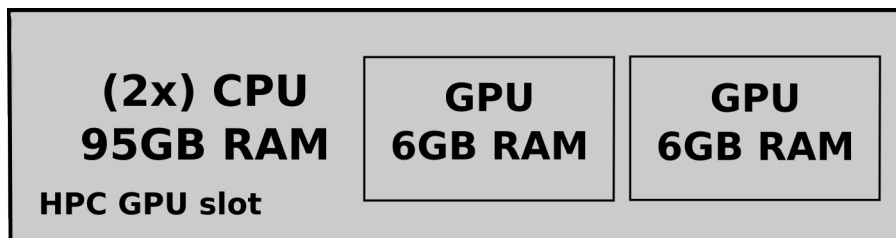
Memory requirement (double precision): $18 \times 8 \times 1\text{GB} = 144\text{GB}$

UoA HPC - hardware

Many nodes with (12x) CPU cores each:



(12x) GPU nodes available:

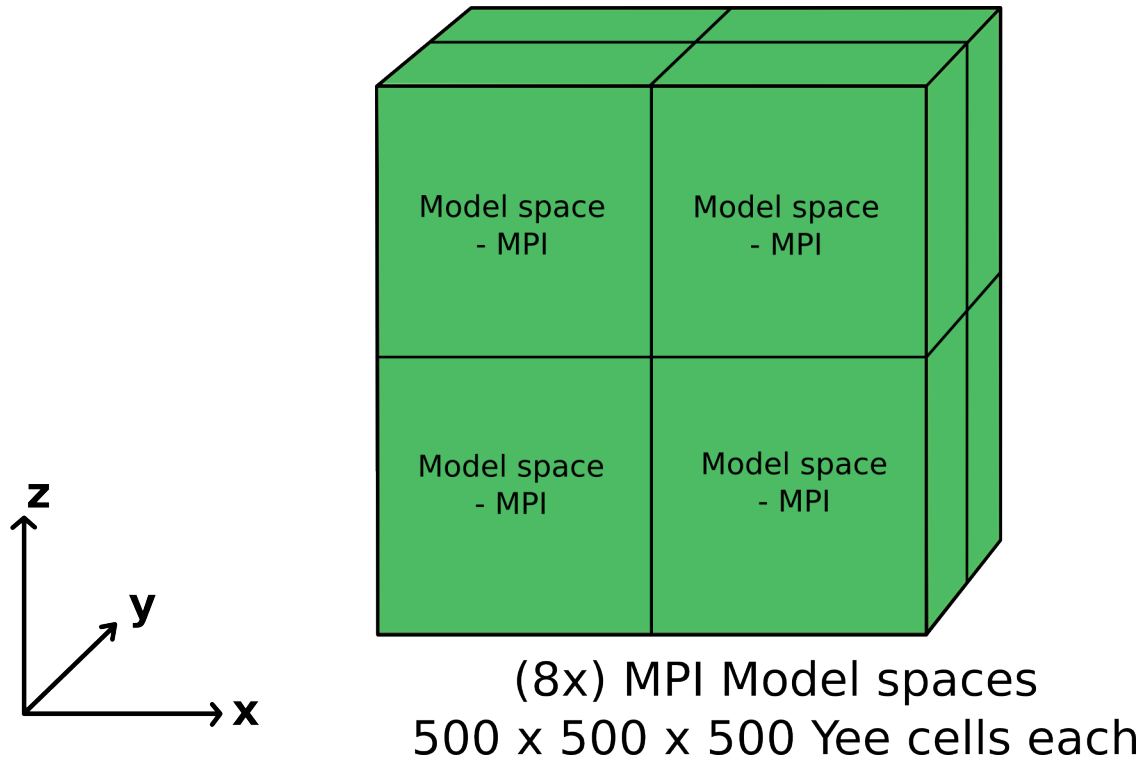


We mostly want the GPU's and memory...

UoA HPC - implementation

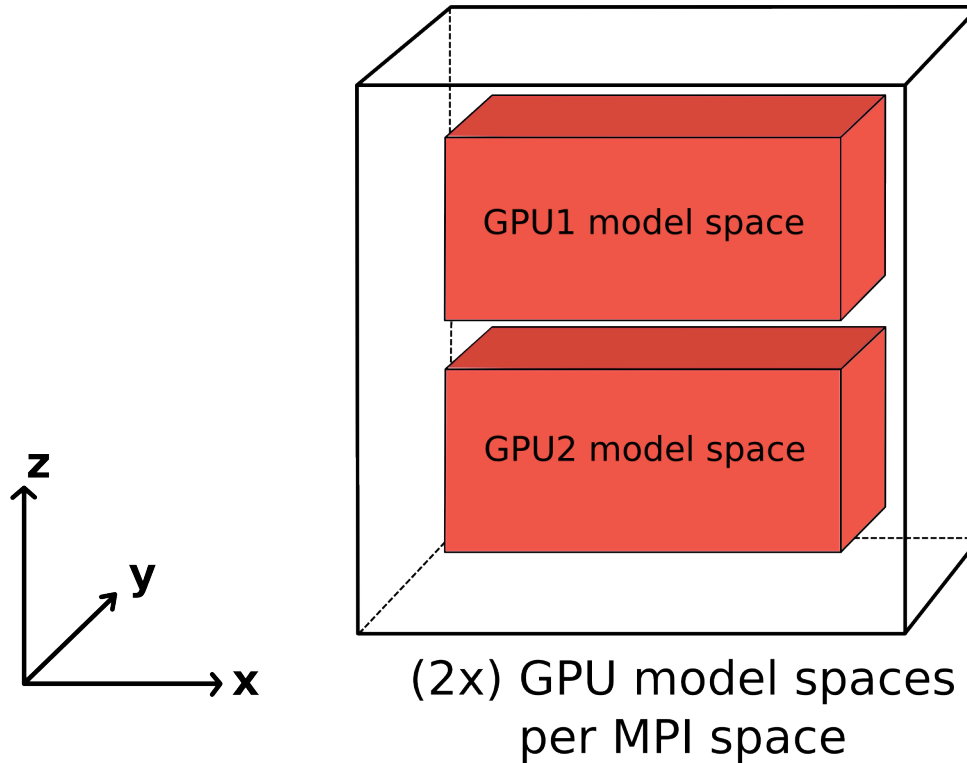
Top-down approach: Use (8x) GPU nodes.

OpenMPI for inter-node communication.



UoA HPC - implementation

Top-down approach: (2x) GPU's in each of the (8x) MPI spaces.
NVIDIA CUDA for GPU coding and communication.



UoA HPC - implementation

- 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)

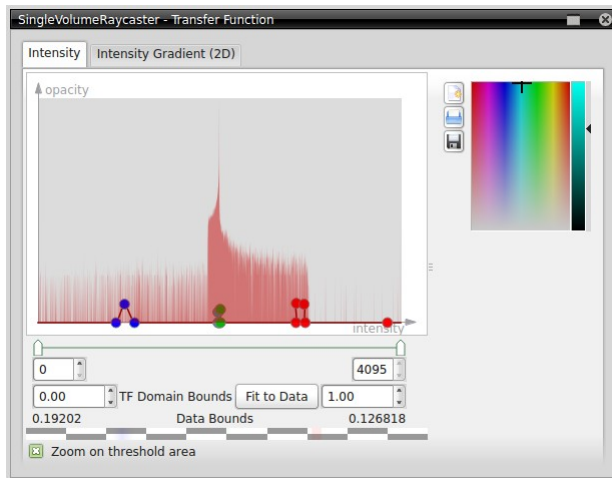
Hardware Platforms

- **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

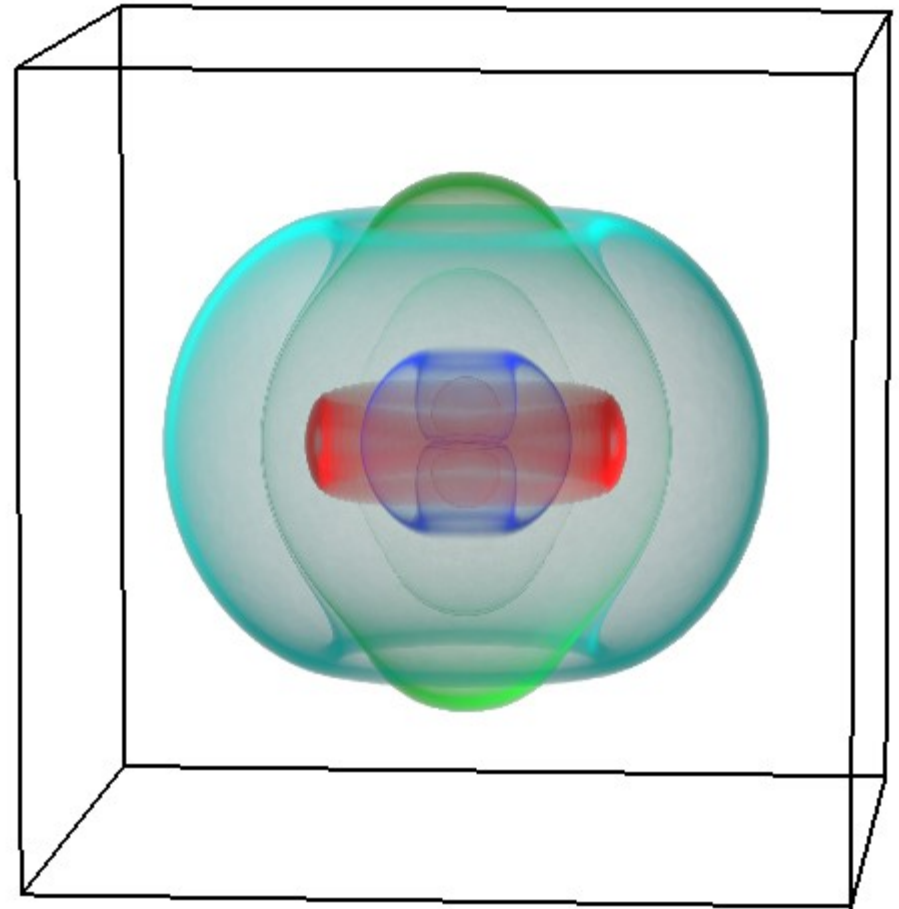
Start with working reference code.

Test and Debug

Data visualization.



Color and transparency settings

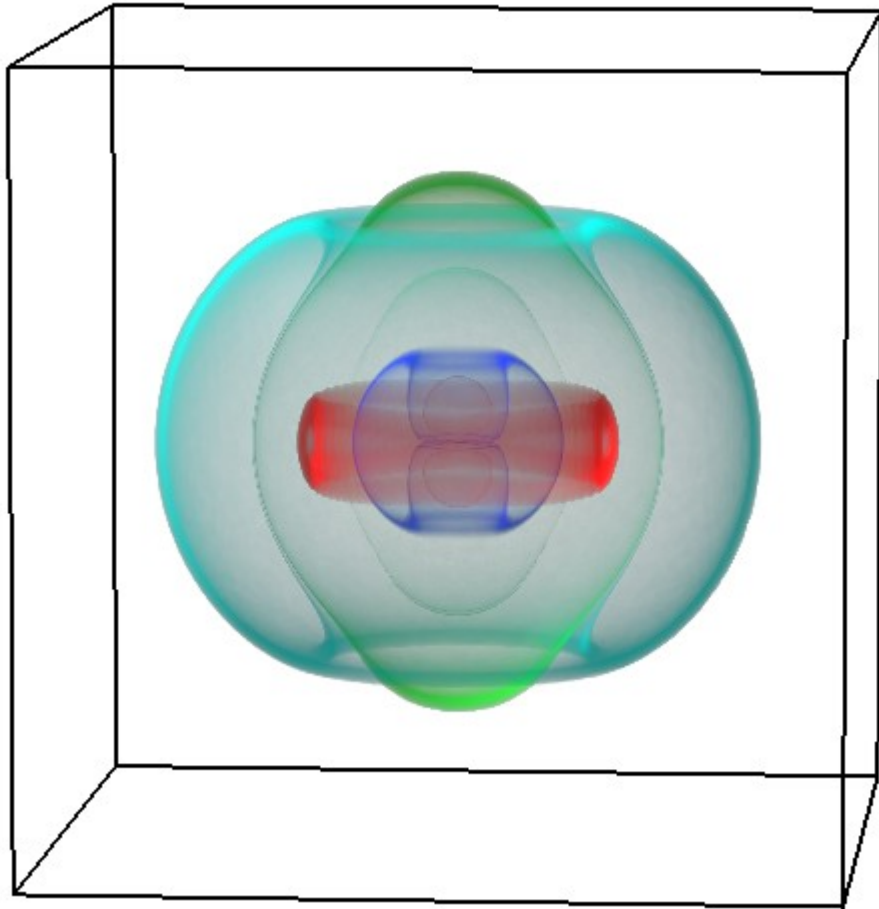


fdtd_100_120_ez_1x

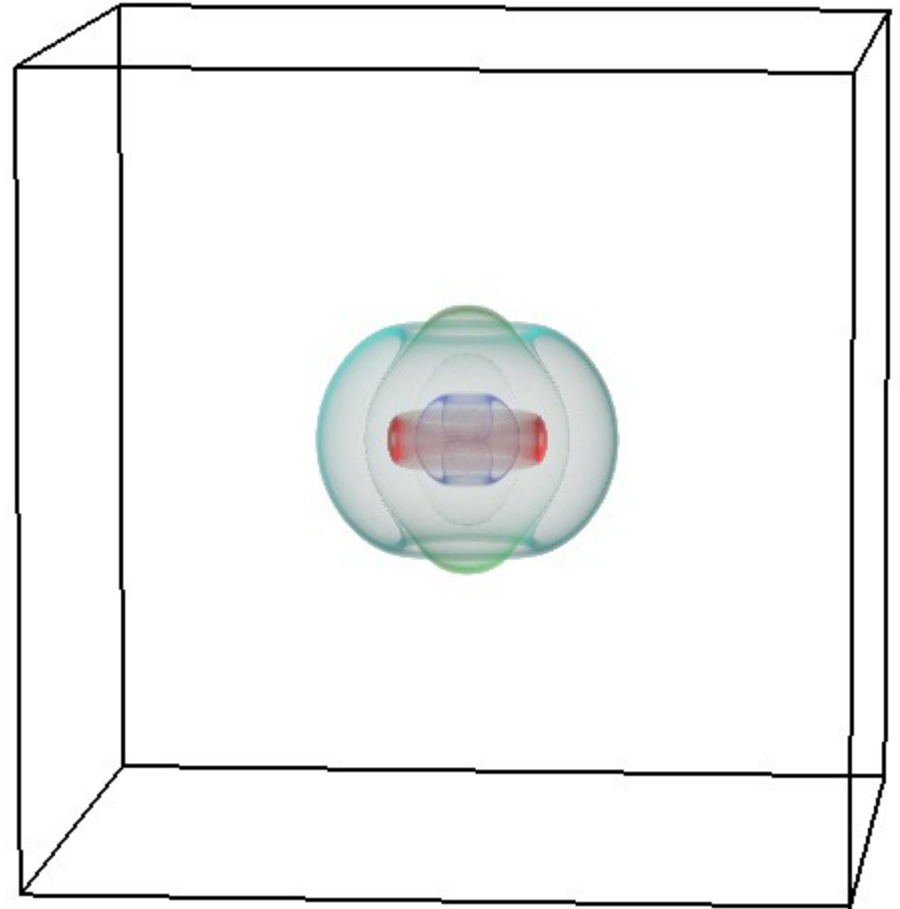
Start with non-MPI reference output.

Test and Debug

Scale-up model space.



fdtd_100_120_ez_1x



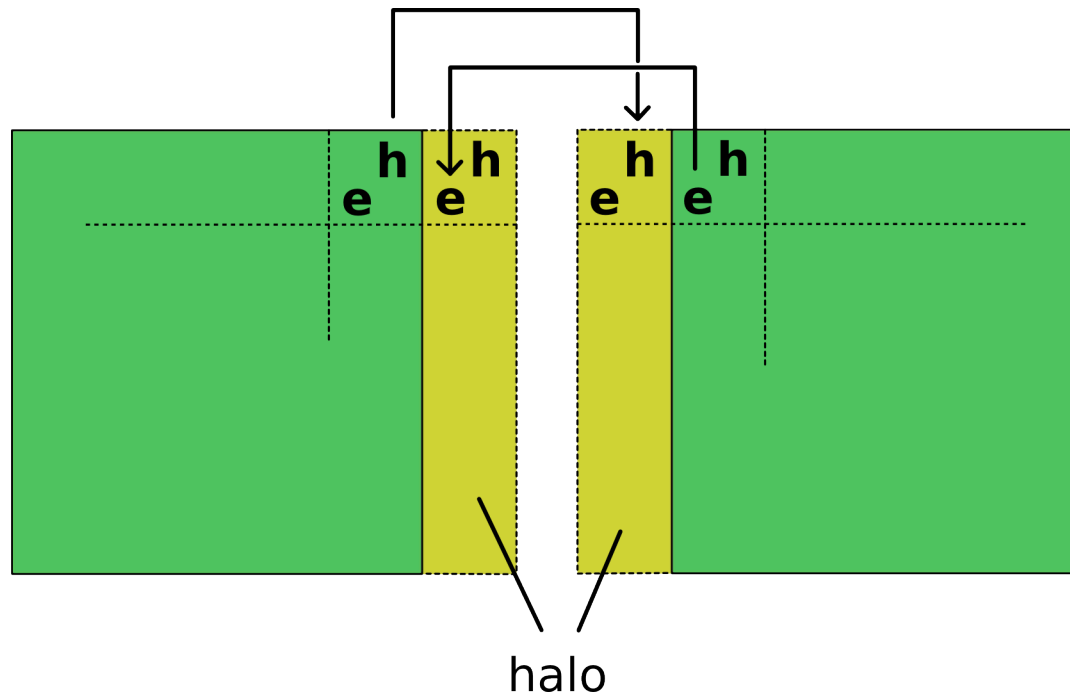
fdtd_200_120_ez_1x

Code Design - MPI

Example design consideration:

Q. How to handle shared boundary with adjacent spaces?

A. Surround space with “halo” and do face swapping.



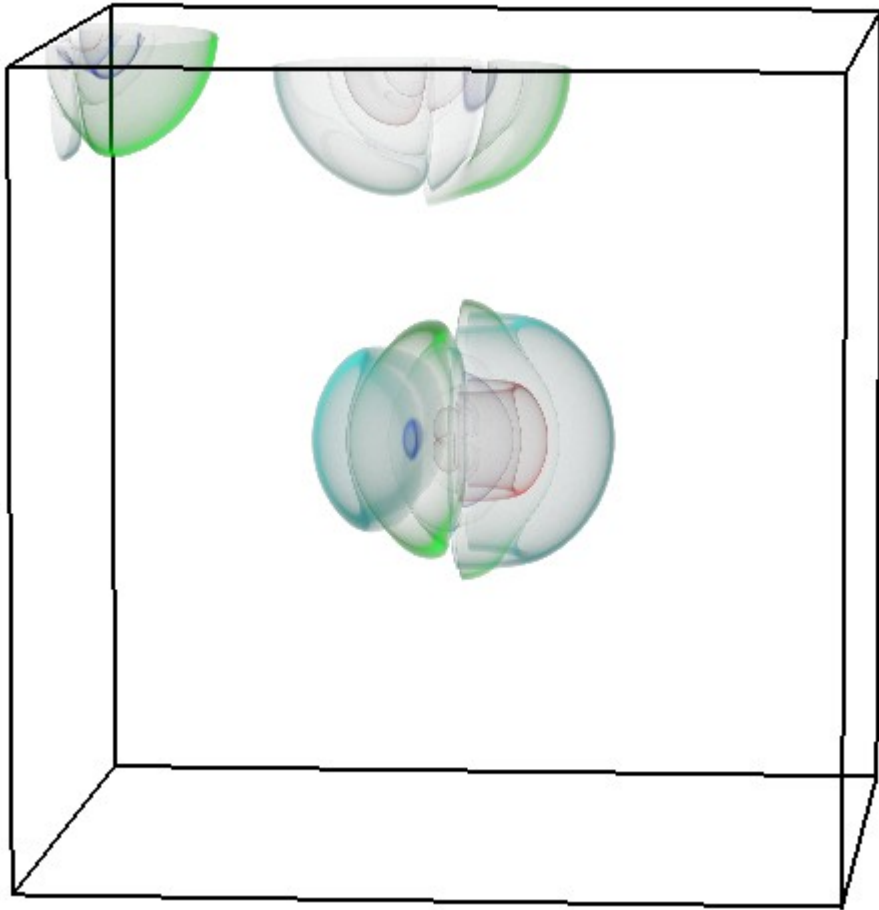
Code Design - MPI

Example: MPI – swap shared faces.

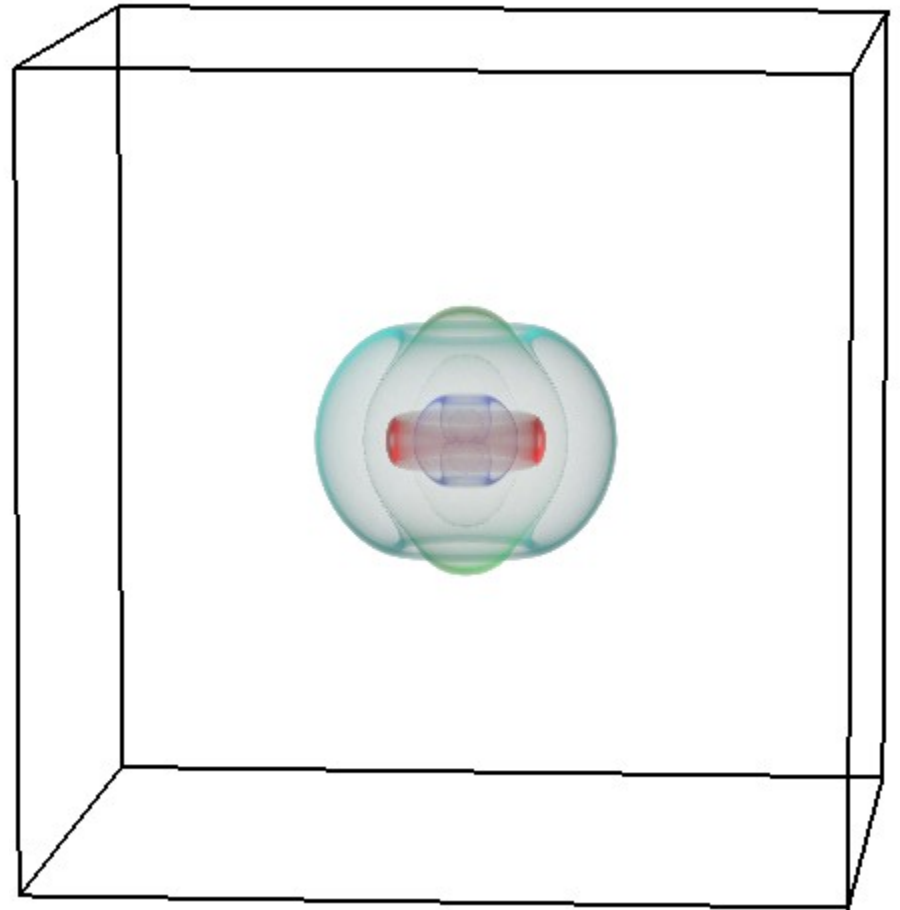
```
// exchange shared face data between nodes
// notes:
// 1) non-blocking used to "bypass buffering"
// 2) implements two exchange types: low-to-high nodes & high-to-low nodes
void exchange_faces(CModel3D *model, bool lo2hi){
    MPI_Status mpi_status;
    MPI_Request mpi_request[3] = {NULL, NULL, 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.



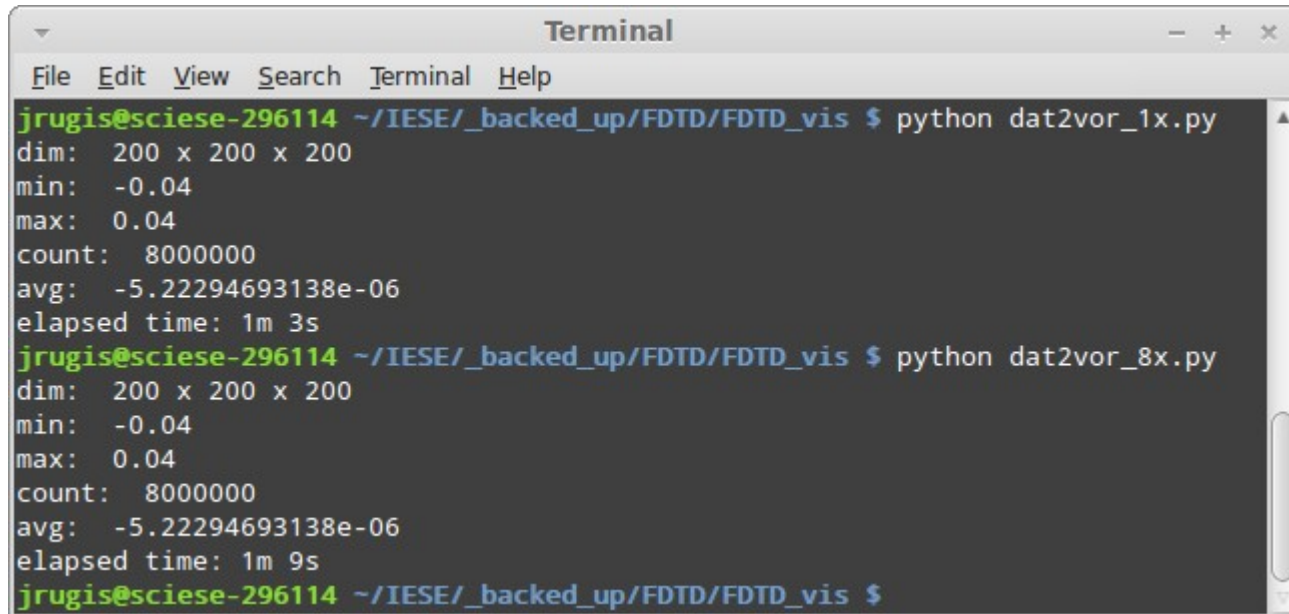
fdtd_200_120_ez_8x_bug



fdtd_200_120_ez_8x

Test and Debug - MPI

Data conversion: non-MPI vs MPI.



```
Terminal
File Edit View Search Terminal Help
jrugis@sciese-296114 ~/IESE/_backed_up/FDTD/FDTD_vis $ python dat2vor_1x.py
dim: 200 x 200 x 200
min: -0.04
max: 0.04
count: 8000000
avg: -5.22294693138e-06
elapsed time: 1m 3s
jrugis@sciese-296114 ~/IESE/_backed_up/FDTD/FDTD_vis $ python dat2vor_8x.py
dim: 200 x 200 x 200
min: -0.04
max: 0.04
count: 8000000
avg: -5.22294693138e-06
elapsed time: 1m 9s
jrugis@sciese-296114 ~/IESE/_backed_up/FDTD/FDTD_vis $
```

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.

Code Design - CUDA

CUDA capabilities

```
jrug001@build-gpu-p:~/CUDA/nsight/deviceQuery/Debug
File Edit View Search Terminal Help
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”.

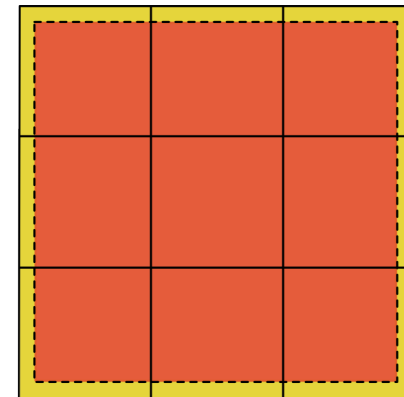
```
void CCudaEH3d::update_e()  
{  
    const dim3 threads(CUDA_TX, CUDA_TY, CUDA_TZ);  
    const dim3 blocks(CUDA_BX, CUDA_BY, CUDA_BZ);  
  
    cudaSetDevice(0);  
    update_e_Kernel<<<blocks, threads>>>  
        (d0[ex], d0[ey], d0[ez], d0[hx], d0[hy], d0[hz],  
         d0[cexe], d0[ceye], d0[ceze], d0[cexh], d0[ceyh], 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

Code Design - CUDA

Example: CUDA kernel – update e-field.

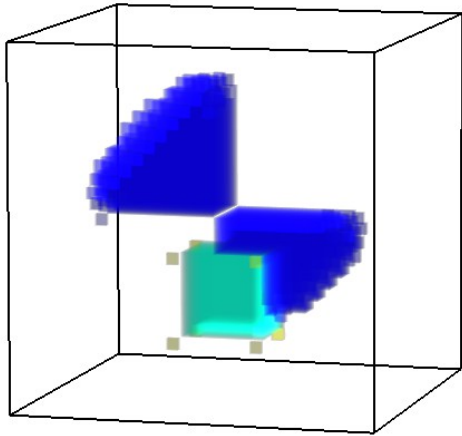
```
__global__ void update_e_Kernel(
    F_TYPE *exG, F_TYPE *eyG, F_TYPE *ezG,
    const F_TYPE *hxG, const F_TYPE *hyG, const F_TYPE *hzG,
    const F_TYPE *cexeG, const F_TYPE *ceyeG, const F_TYPE *cezeG,
    const F_TYPE *cexhG, const F_TYPE *ceyhG, const F_TYPE *cezhG,
    const size_t sX, const 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;

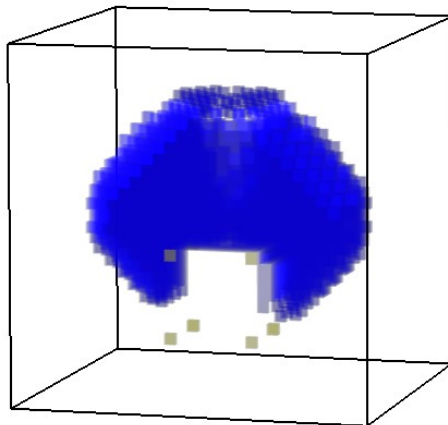
    exG[n] = cexeG[n] * exG[n]
        + cexhG[n] * ((hzG[n] - hzG[n - sX]) - (hyG[n] - hyG[n - sXY]));
    eyG[n] = ceyeG[n] * eyG[n]
        + ceyhG[n] * ((hxG[n] - hxG[n - sXY]) - (hzG[n] - hzG[n - 1]));
    ezG[n] = cezeG[n] * ezG[n]
        + cezhG[n] * ((hyG[n] - hyG[n - 1]) - (hxG[n] - hxG[n - sX]));
```

Test and Debug - CUDA

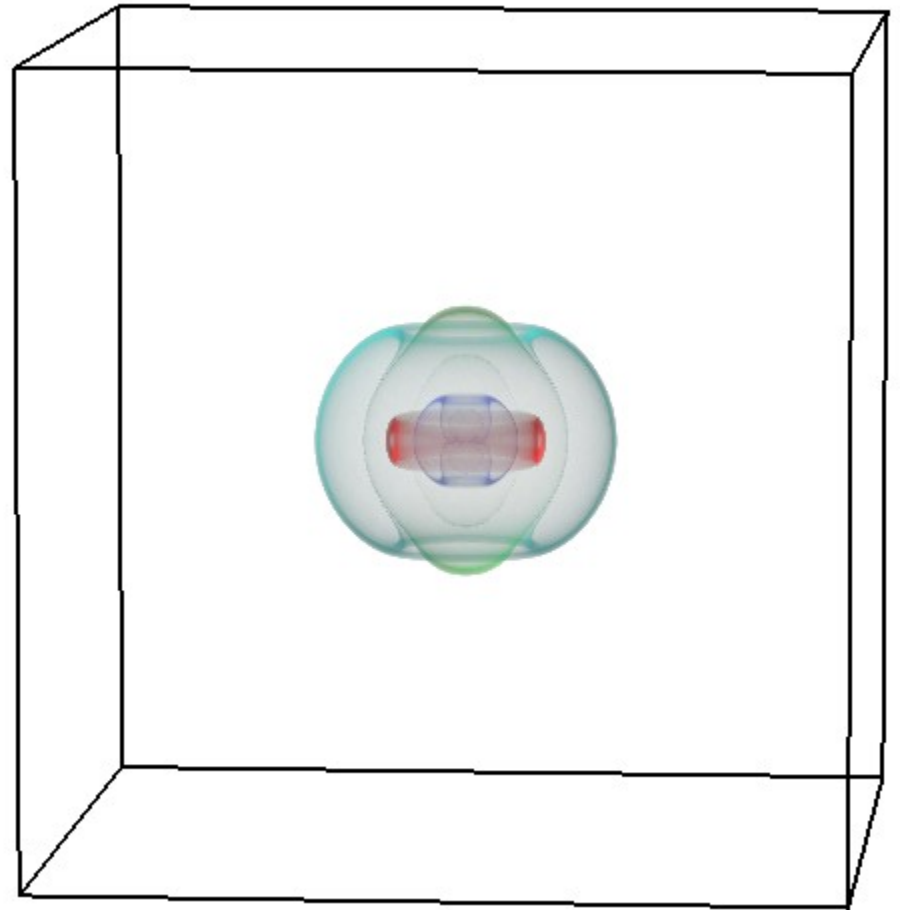
CUDA testing and final.



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:~/CUDA/nsight/FDTD_02
File Edit View Search Terminal Help
[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

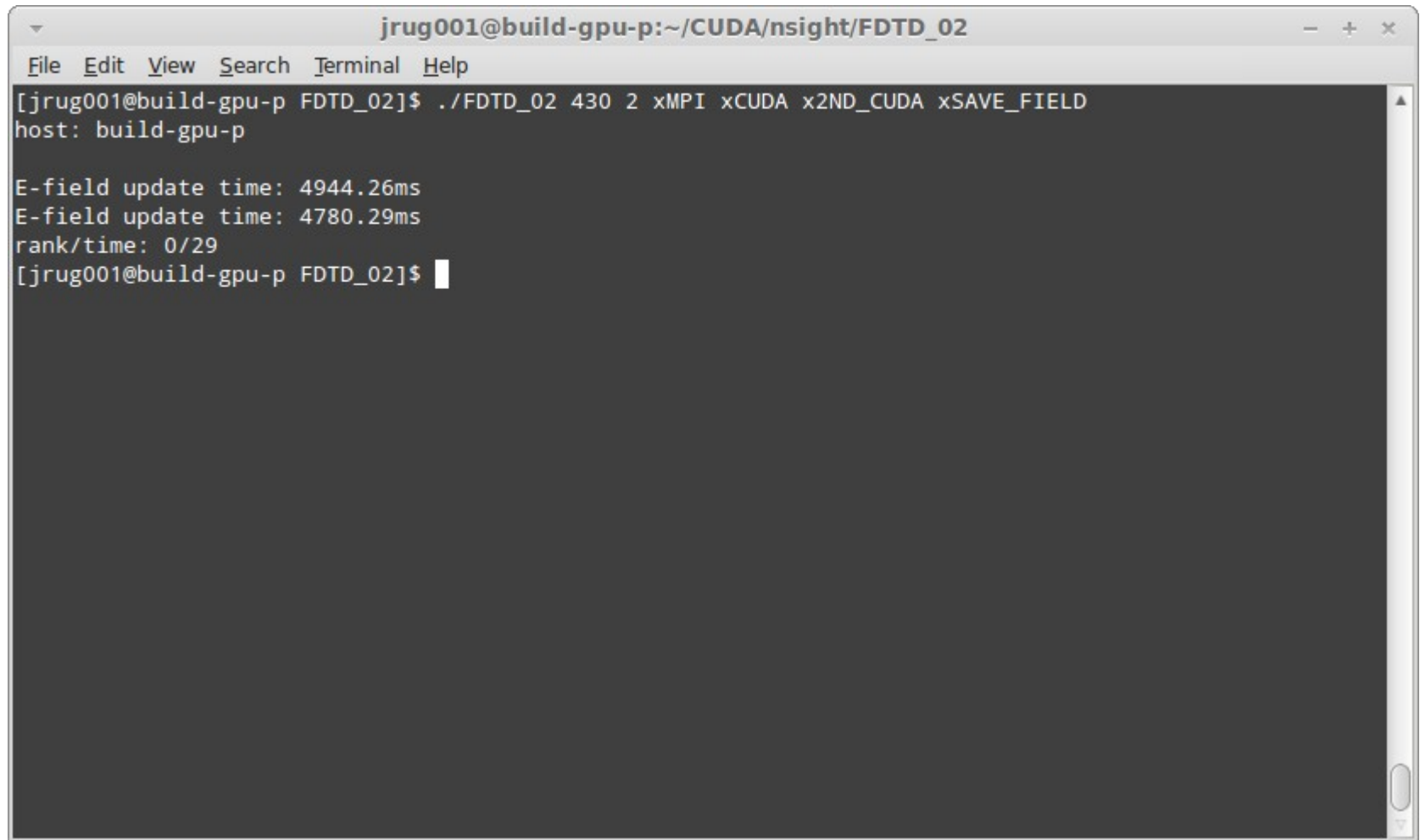
CUDA device count: 2
    device: Tesla M2090
    capability: 2.0
    driver: 5.0
    runtime: 5.0
    pci bus/location: 20/0
    global memory: 5636554752
    shared memory/block: 49152
    integrated host memory: No
    map host memory: Yes
    unified addressing: Yes
    multiprocessor count: 16
    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

```
jrug001@build-gpu-p:~/CUDA/nsight/FDTD_02
File Edit View Search Terminal Help

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:
Time(%)   Time      Calls      Avg      Min      Max      Name
70.07    893.78ms     30    29.79ms  350.52us  49.37ms  [CUDA memcpy HtoD]
14.62    186.50ms      2    93.25ms  92.81ms  93.69ms  update_h_Kernel(double*, double*, double*,
14.47    184.56ms      2    92.28ms  91.38ms  93.19ms  update_e_Kernel(double*, double*, double*,
0.40      5.06ms     12   422.00us 316.50us  632.81us  [CUDA memcpy DtoH]
0.09      1.11ms      2   557.23us 553.57us  560.88us  h_in_Kernel_yz(double const *, double*, dou
0.09      1.10ms      2   552.24us 549.86us  554.62us  e_in_Kernel_yz(double const *, double*, dou
0.07     848.03us      2   424.01us 423.38us  424.65us  h_out_Kernel_yz(double*, double const *, do
0.07     840.57us      2   420.29us 420.02us  420.55us  e_out_Kernel_yz(double*, double const *, do
0.03     344.68us      2   172.34us 171.85us  172.83us  e_out_Kernel_xy(double*, double const *, do
0.03     343.45us      2   171.72us 171.62us  171.83us  h_out_Kernel_xy(double*, double const *, do
0.02     204.40us      2   102.20us 100.99us  103.41us  e_in_Kernel_xy(double const *, double*, dou
0.02     204.36us      2   102.18us  99.72us  104.63us  h_in_Kernel_xy(double const *, double*, dou
0.01     176.36us      2    88.18us  88.13us   88.23us  e_out_Kernel_xz(double*, double const *, do
0.01     174.90us      2    87.45us  87.21us   87.69us  h_out_Kernel_xz(double*, double const *, do
0.01     120.87us      2    60.43us  60.18us   60.69us  h_in_Kernel_xz(double const *, double*, dou
0.01     115.47us      2    57.73us  56.96us   58.51us  e_in_Kernel_xz(double const *, double*, dou
[jrug001@build-gpu-p FDTD_02]$ nvprof ./FDTD_02 430 2 xMPI CUDA x2ND_CUDA xSAVE_FIELD >> profile_07.t
```

Profiling - CUDA



A terminal window titled "jrug001@build-gpu-p:~/CUDA/nsight/FDTD_02" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the execution of the command `./FDTD_02 430 2 xMPI xCUDA x2ND_CUDA xSAVE_FIELD`. The output includes the host name "build-gpu-p", two "E-field update time" measurements (4944.26ms and 4780.29ms), and the rank/time "0/29". The prompt `[jrug001@build-gpu-p FDTD_02]$` is followed by a cursor.

```
jrug001@build-gpu-p:~/CUDA/nsight/FDTD_02
File Edit View Search Terminal Help
[jrug001@build-gpu-p 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

```
jrug001@build-gpu-p:~/CUDA/nsight/FDTD_02
File Edit View Search Terminal Help
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: $4.8 / 0.117 = \mathbf{41}$

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

$(\text{GPU space}) / (\text{model space}) =$

$(2 \times 418 \times 418 \times 208) / (430 \times 430 \times 430) = \mathbf{91\%}$

And this assumes two GPU's running concurrently...

Code Design - CUDA

Concurrency and object-oriented design.

```
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
}
```

GPU1

GPU2

CPU

time →

Concurrent execution

Test and Debug - CUDA

Create and use timer object.

Without CUDA.

```
7      time(sec) step: 9.19e+00 A(4.47e+00) MPI(1.71e-01) B(4.40e+00) MPI(1.43e-01)
5      time(sec) step: 9.20e+00 A(4.48e+00) MPI(1.92e-01) B(4.40e+00) MPI(1.23e-01)
0      time(sec) step: 9.21e+00 A(4.49e+00) MPI(1.72e-01) B(4.34e+00) MPI(2.06e-01)
2      time(sec) step: 9.21e+00 A(4.65e+00) MPI(3.16e-02) B(4.52e+00) MPI(2.81e-03)
4      time(sec) step: 9.21e+00 A(4.64e+00) MPI(3.56e-02) B(4.51e+00) MPI(2.60e-02)
3      time(sec) step: 9.21e+00 A(4.39e+00) MPI(2.83e-01) B(4.35e+00) MPI(1.86e-01)
1      time(sec) step: 9.21e+00 A(4.39e+00) MPI(1.39e-01) B(4.35e+00) MPI(3.30e-01)
6      time(sec) step: 9.20e+00 A(4.52e+00) MPI(1.55e-01) B(4.33e+00) MPI(1.91e-01)
```

With CUDA.

```
7      time(sec) step: 8.81e-01 A(3.93e-01) MPI(2.31e-02) B(4.31e-01) MPI(3.44e-02)
5      time(sec) step: 8.80e-01 A(3.78e-01) MPI(6.00e-02) B(4.00e-01) MPI(4.17e-02)
0      time(sec) step: 8.79e-01 A(4.24e-01) MPI(7.63e-02) B(3.36e-01) MPI(4.37e-02)
2      time(sec) step: 8.81e-01 A(4.53e-01) MPI(4.56e-02) B(3.66e-01) MPI(1.64e-02)
3      time(sec) step: 8.80e-01 A(3.72e-01) MPI(8.39e-02) B(3.90e-01) MPI(3.47e-02)
4      time(sec) step: 8.79e-01 A(4.19e-01) MPI(1.22e-02) B(3.40e-01) MPI(1.08e-01)
1      time(sec) step: 8.79e-01 A(3.75e-01) MPI(2.34e-02) B(3.85e-01) MPI(9.66e-02)
6      time(sec) step: 8.81e-01 A(4.22e-01) MPI(6.53e-02) B(3.53e-01) MPI(4.08e-02)
```

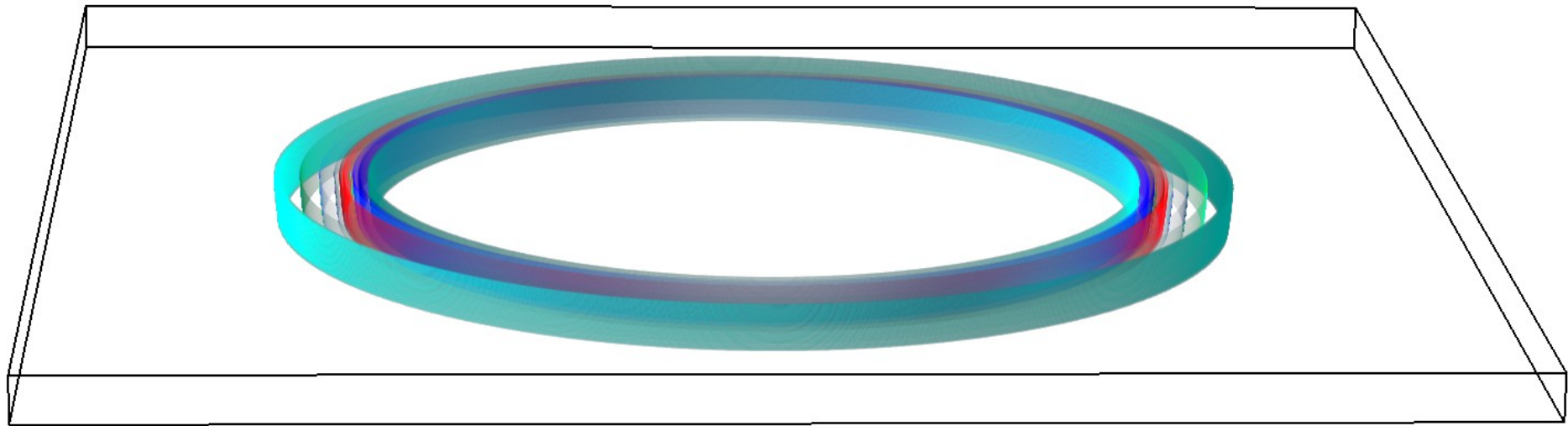
Actual speedup: $9.2 / 0.88 = 10.5$

Test and Debug

Extracted slice from large model simulation.

Runtime: 550 steps at approximately 1 second per step.

Single frame output file size: $848^3 \times 6 \times 8 = \mathbf{29.3GB}$



fdtd_848_550_ez_8x

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?*