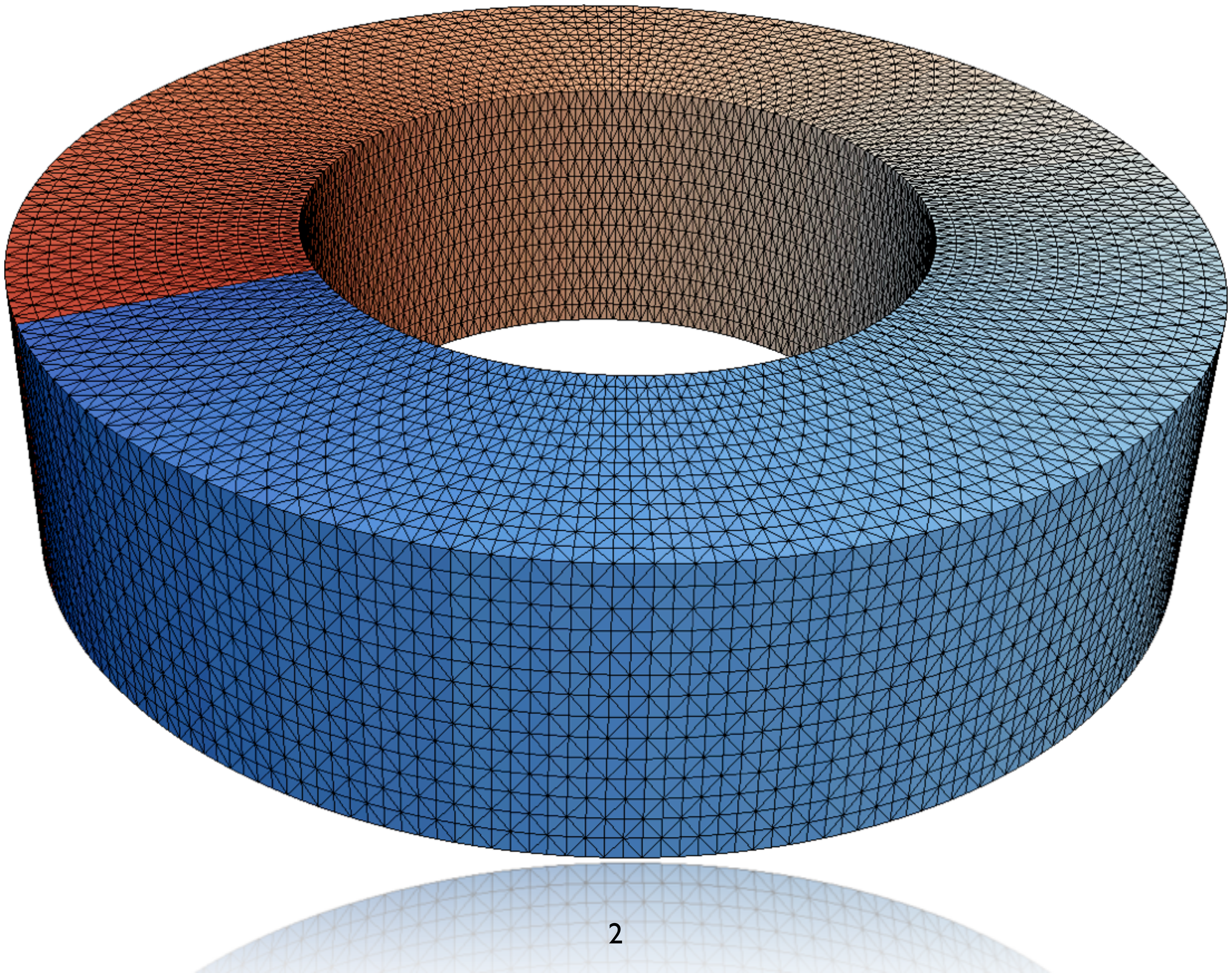
Application Experience with the GPU: Explicit Finite Elements

Isaac Dooley

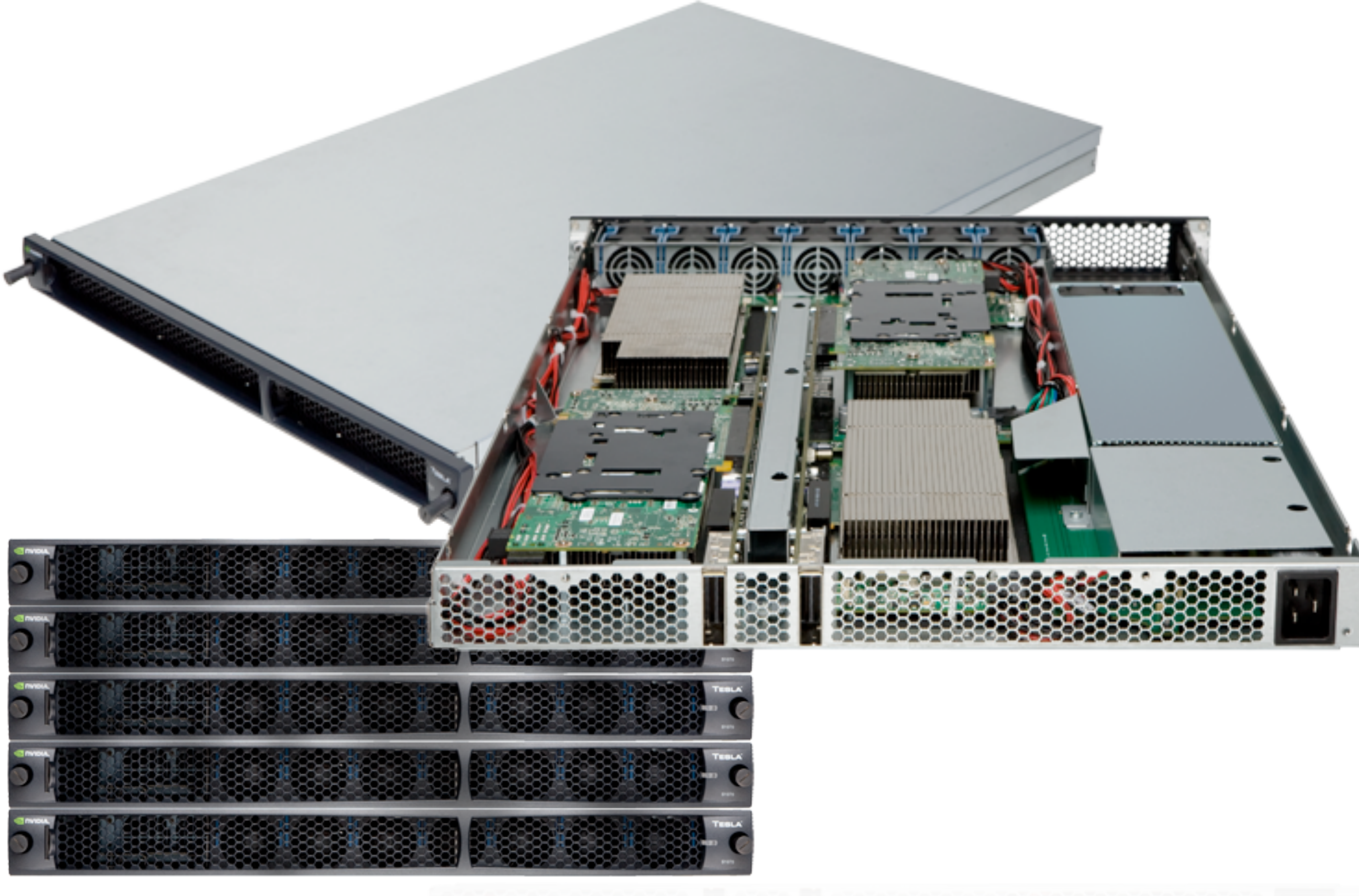
7th Annual Workshop on Charm++ and its Applications

Thursday, April 16th, 2009

Application: 3D finite elements for non-homogeneous materials.

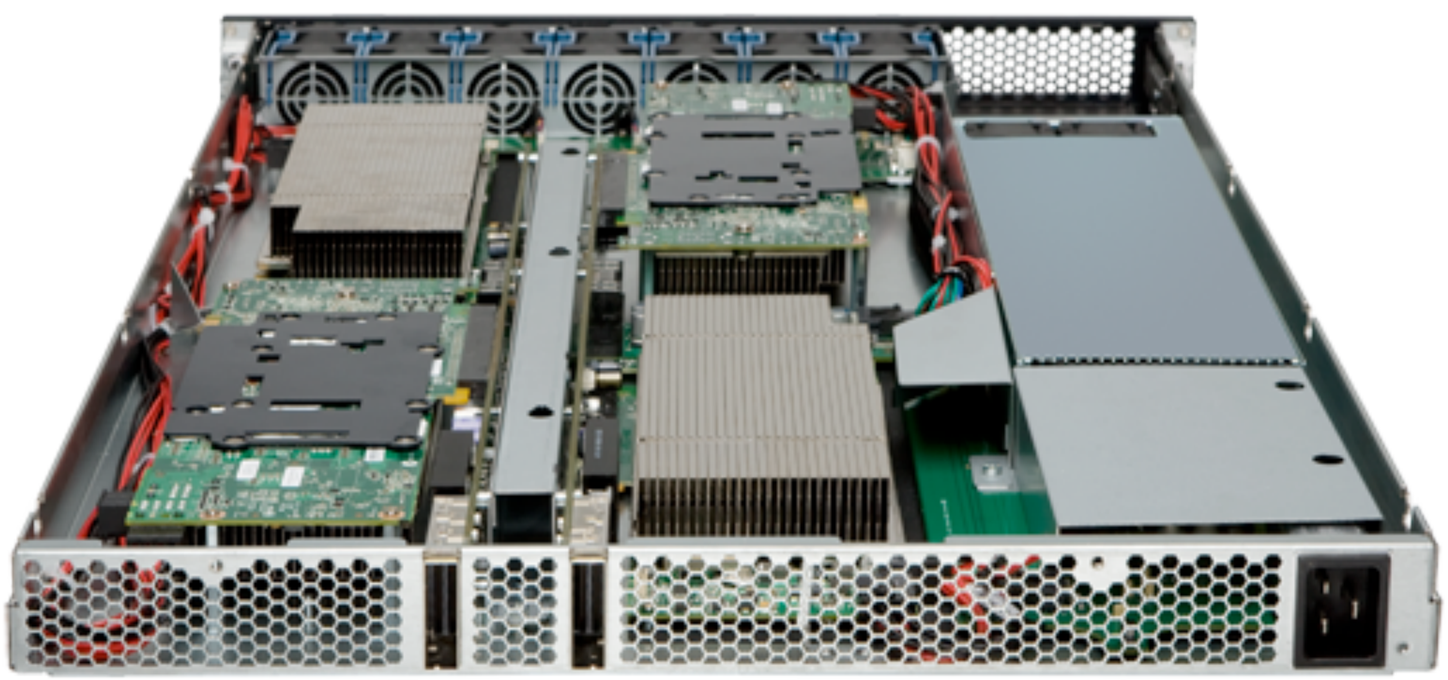


NVIDIA Tesla S1070



NVIDIA Tesla S1070 Specifications:

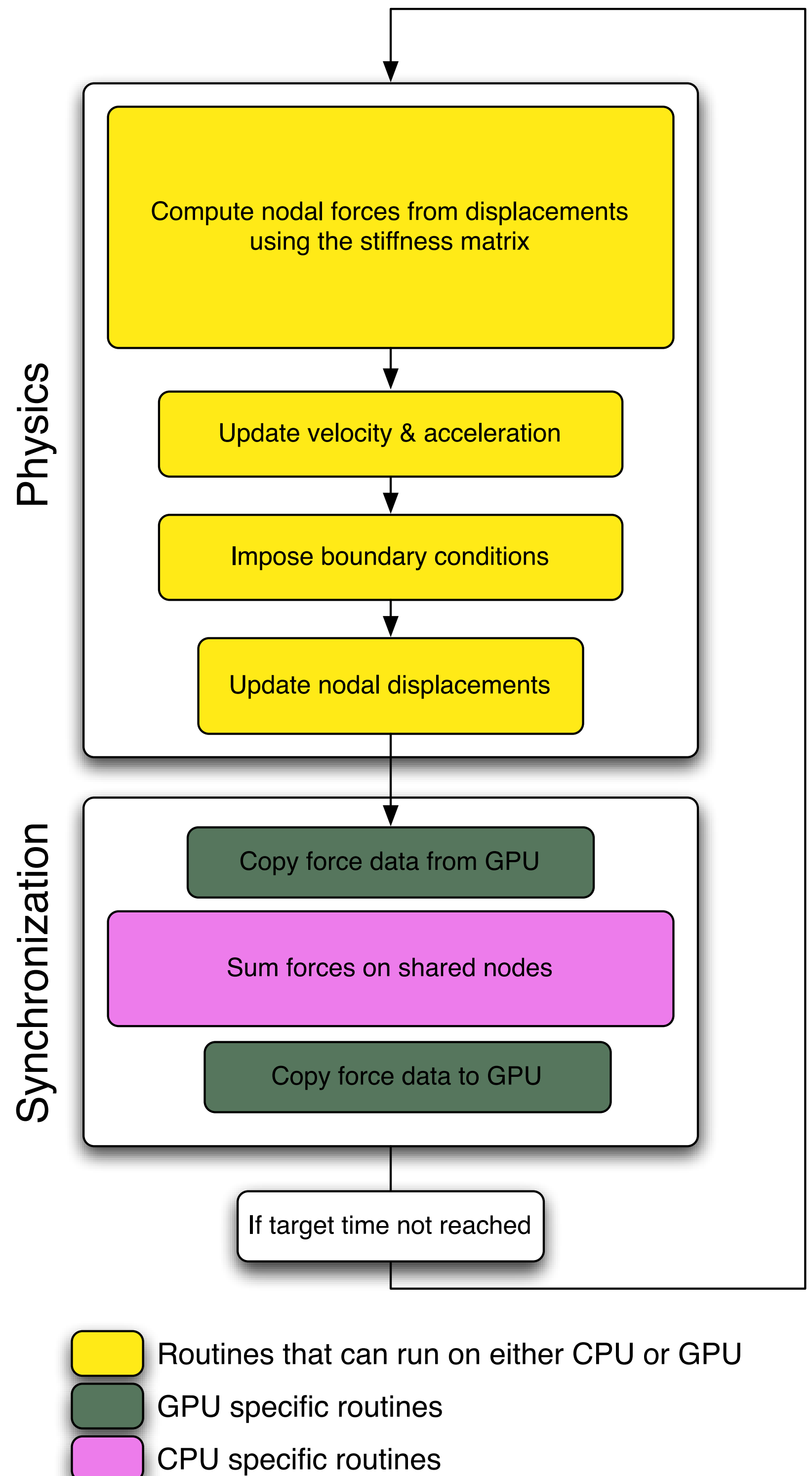
- 960 Cores (240 per processor)
- 4.14 TFlops Single Precision
- 345 GFlops Double Precision
- 16GB RAM



192 NCSA Lincoln Cluster Nodes:

- Two Intel Harpertown quad core E5410 CPUs
- Half of a Tesla Unit (2 GPUs)
- InfiniBand interconnect fabric

Application: 3D finite elements for non-homogeneous materials.



Application: Implementation

CPU Version

```
nodeIterator itr;
for(nodeItr_Begin(itr);  nodeItr_IsValid(itr); nodeItr_Next(itr)){
   n_data=node_GetData(itr);
   for(inti=0;i<dof;++i){
      constFP_TYPE a_old=n_data->a[i];
      n_data->a[i] = -n_data->F[i]/n_data->mass;
      n_data->v[i] += 0.5*dt*(n_data->a[i]+a_old);
   }
}
```

GPU CUDA Version

```
n_data=node_GPU_GetData(my_node);
for(inti=0;i<dof;++i){
   constFP_TYPE a_old = n_data->a[i];
   n_data->a[i] = -n_data->F[i]/n_data->mass;
   n_data->v[i] += 0.5*dt*(n_data->a[i]+a_old);
}
```

Uses an Iterator Interface on ParFUM

Kernel code

Application: Implementation

Mesh over-decomposed into many pieces.

Pieces can execute either on CPU or GPU.

Balance number of pieces between CPU and GPU Manager processors.

Domain specific framework: ParFUM + Iterator Interface

Iterator Interface was customized for CPU & CUDA.

Application Specific Characteristic

Larger than usual data per element/node

Double Precision

	Bytes
Element	1184
Node	912

Single Precision

	Bytes
Element	592
Node	460

Goals / Unknowns

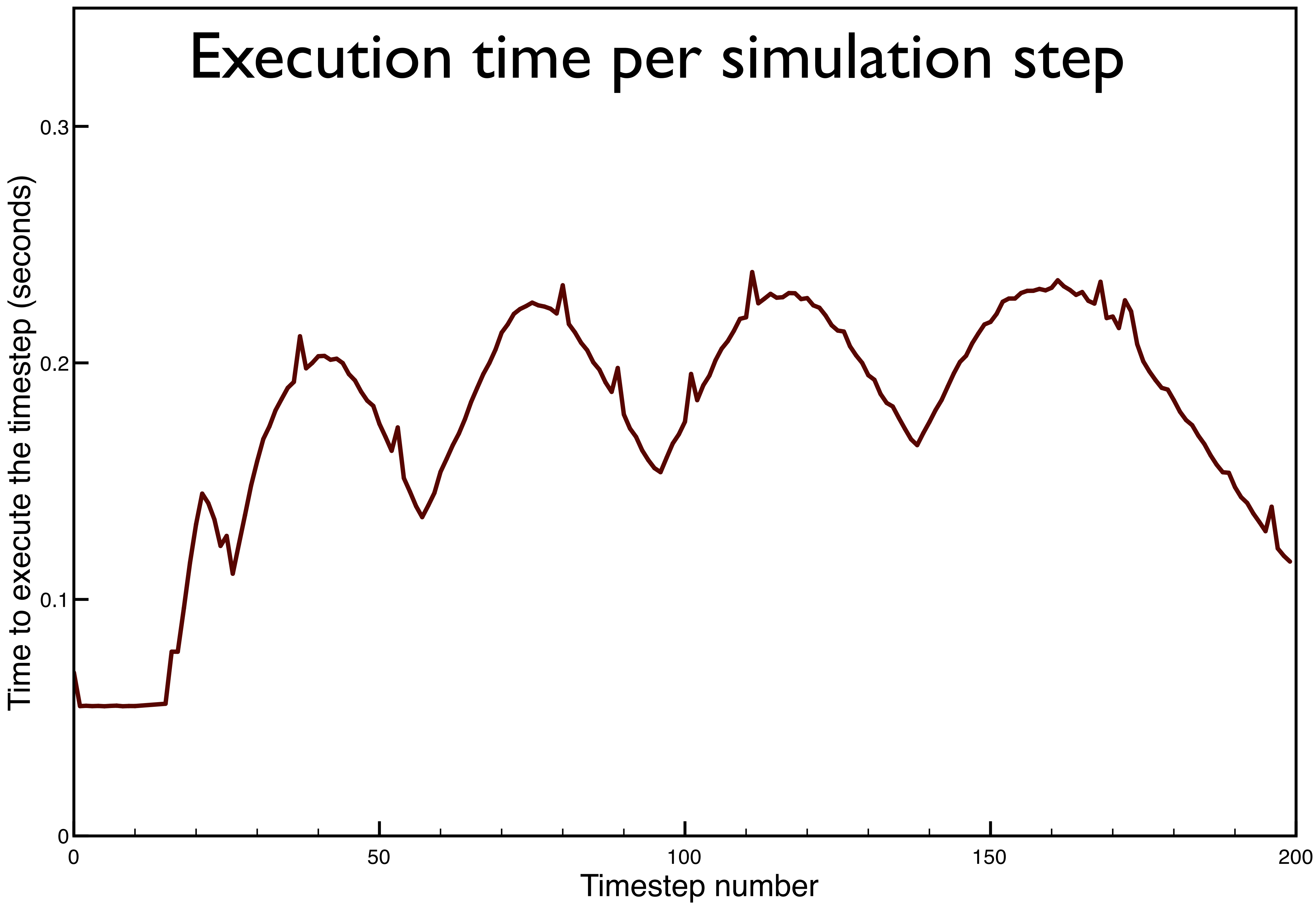
Can this program use the GPUs well?
Most similar published work achieves speedup of 7 on 1 GPU.

What changes will have to be made to adapt it to CUDA?

Can our existing methodology apply to heterogeneous clusters?

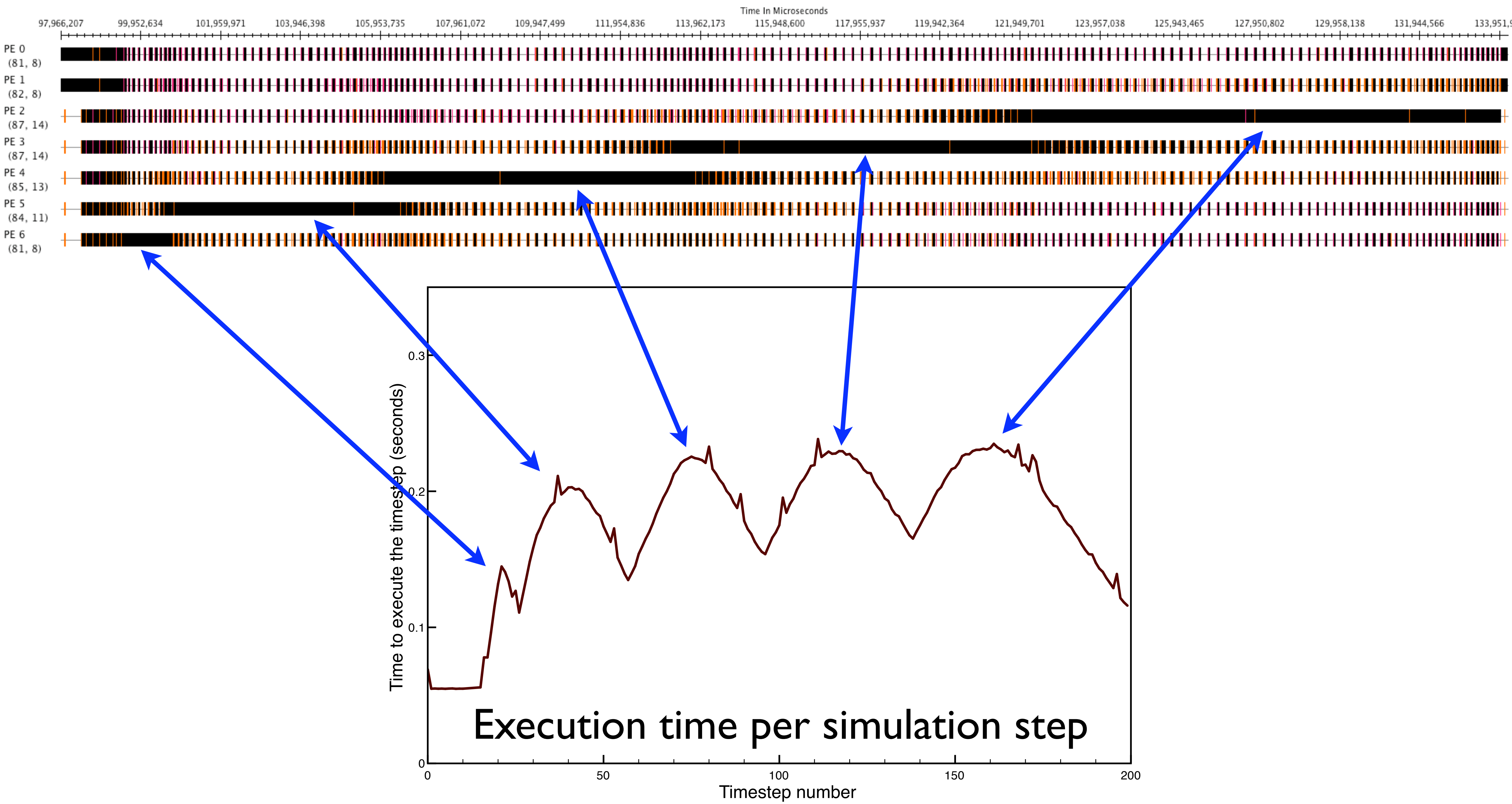
How well will this program scale?

Early Performance Problem with CPU Version



Early Performance Problem with CPU Version

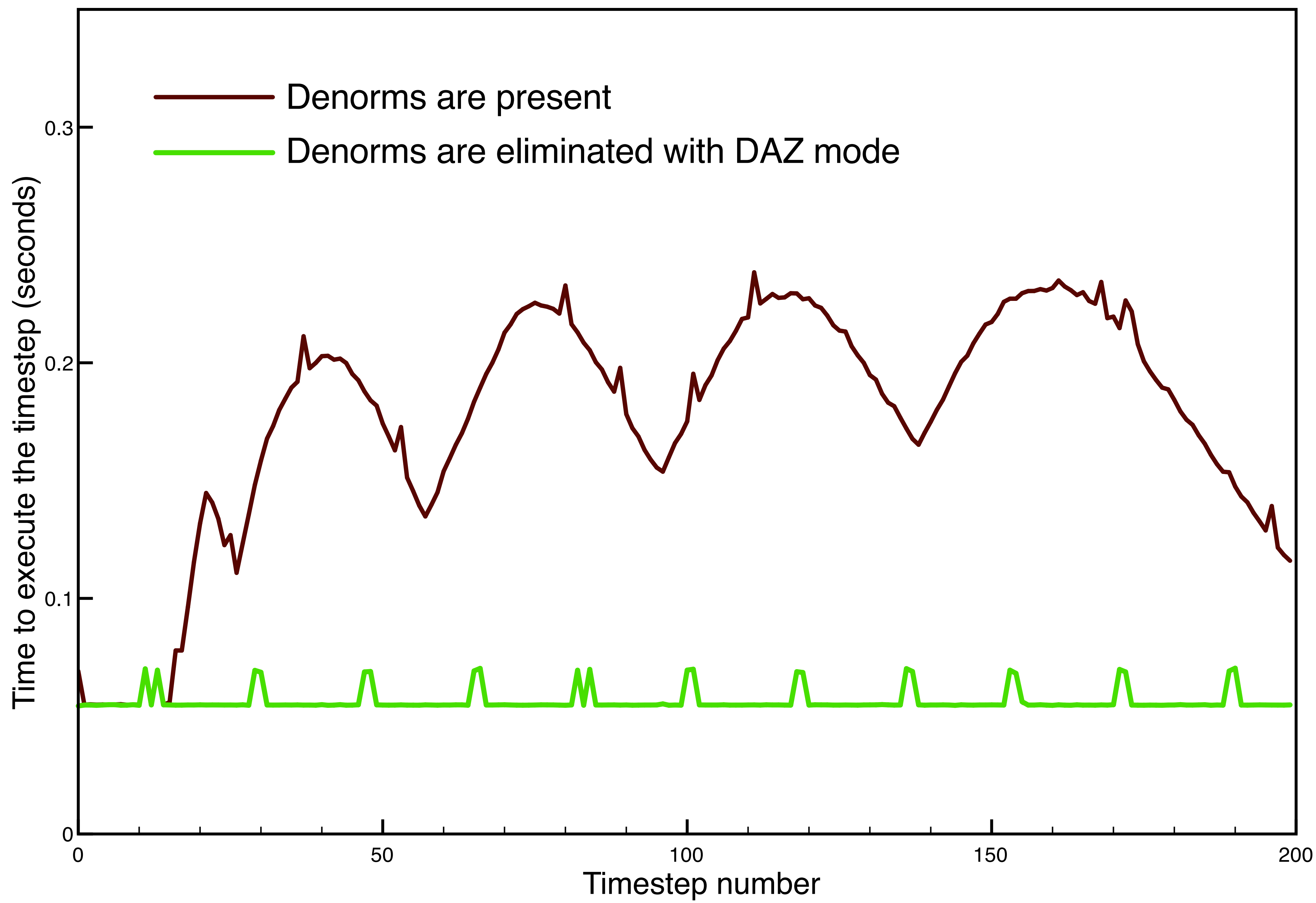
Timeline view:



Execution time per simulation step

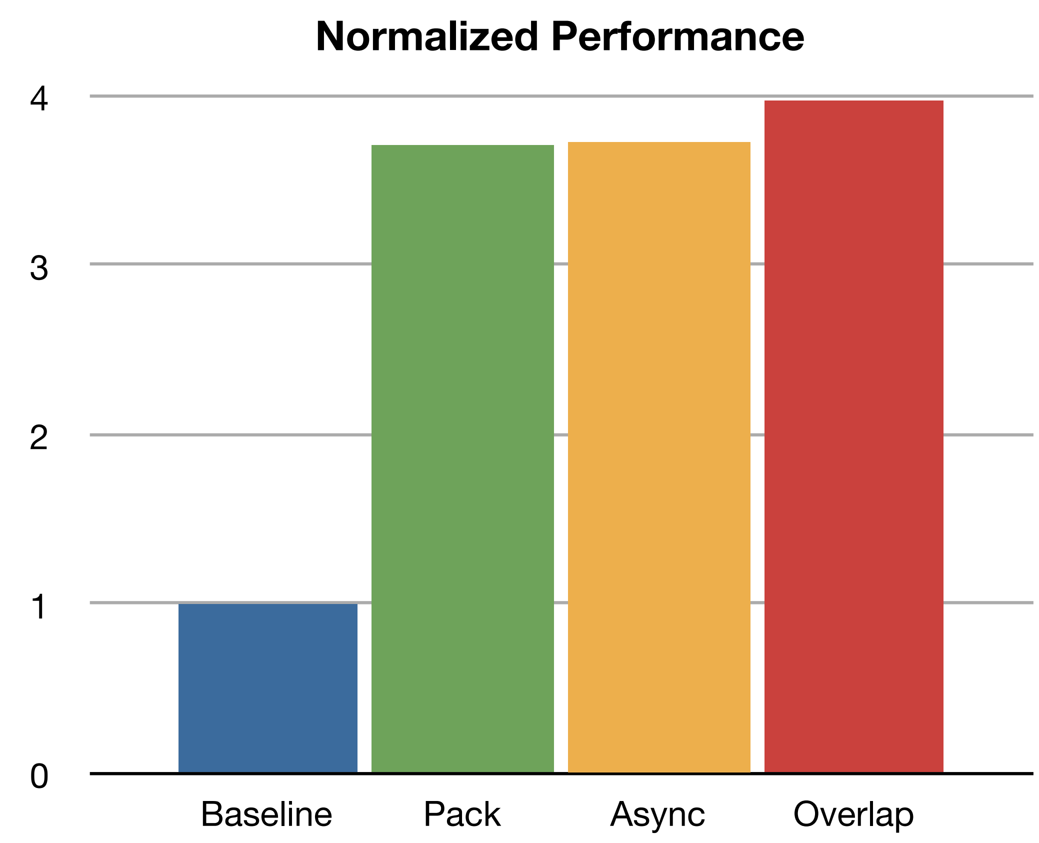
Early Performance Problem with CPU Version

Execution time per simulation step:



Application: Main GPU Optimizations

- Minimized data copied to/from main memory
- Asynchronously executed kernels, memory transfers
- Overlapping CPU work with Asynchronous GPU kernels



Application: Load Balancing CPU/GPU

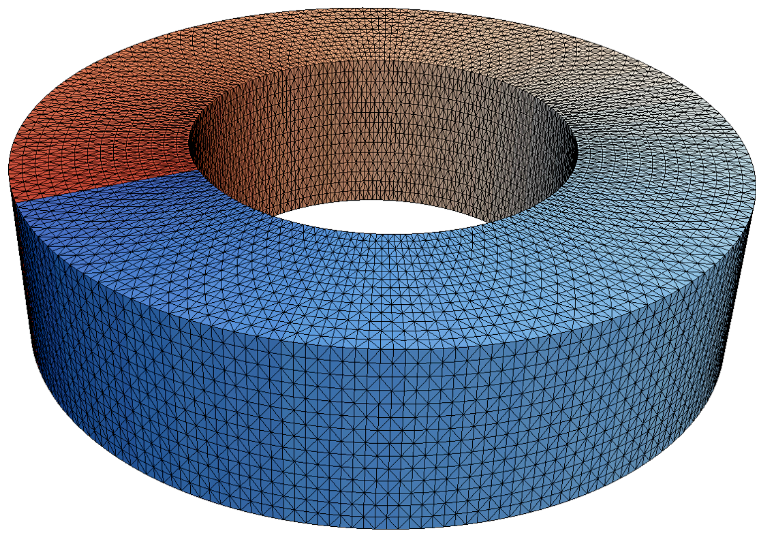
- Current approach: manually tune number of mesh pieces on each GPU/CPU
- Future approach: Develop a heterogeneous load balancer that can automatically map mesh pieces to CPU/GPUs ! How?
- Future approach: Use control point framework to autotune the ratio.

Approach to Understanding Performance

Examine timeline visualizations
Measure time/step
Measure memory overhead

Scaling

235K tetrahedra

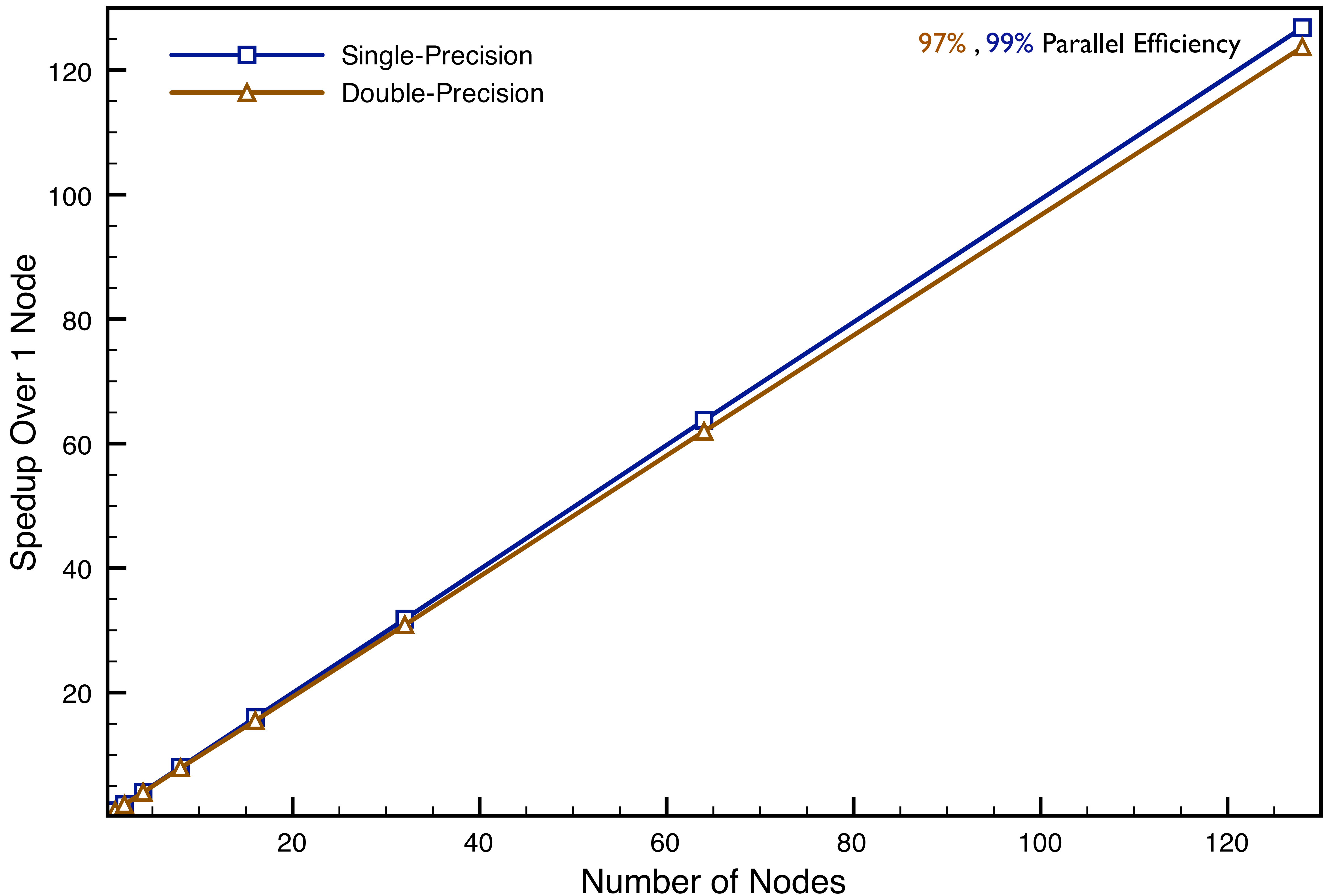


Strong scaling (Same mesh size):
1 core to 1 node

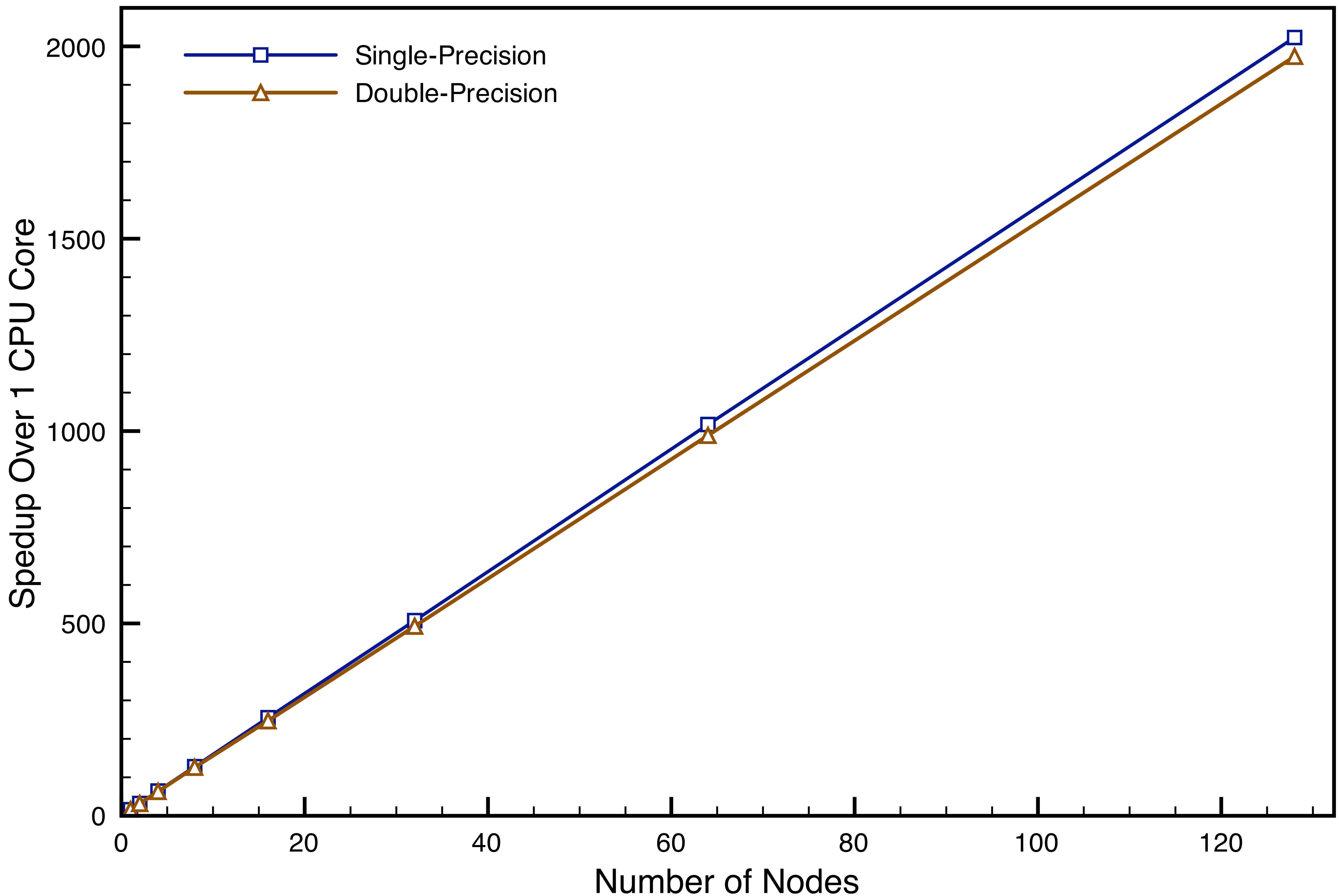
Weak scaling (Scale up mesh sizes):
1 node to 128 nodes

30M tetrahedra

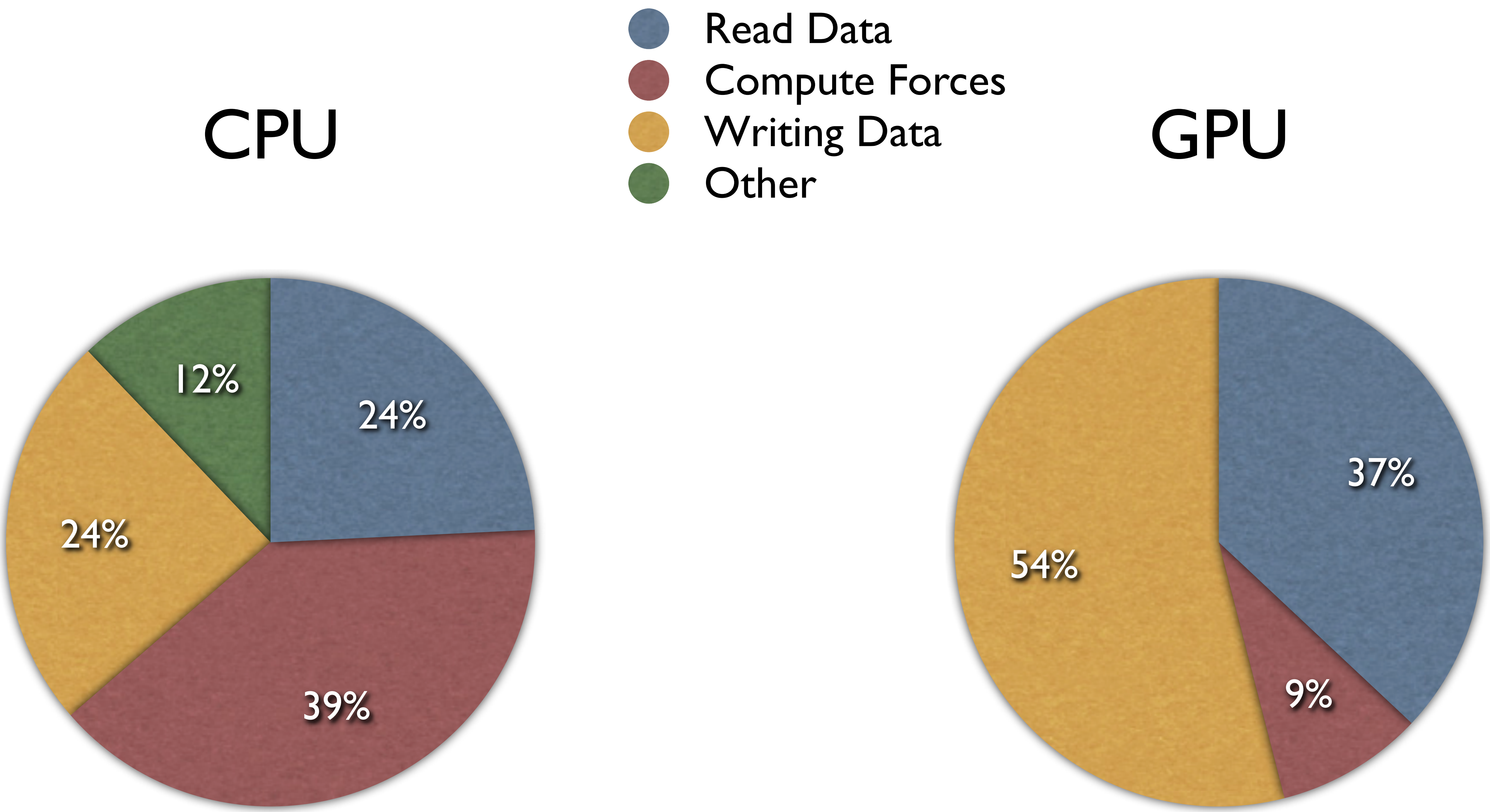
Resulting Application Performance



Resulting Application Performance



Application Performance



Measured by eliminating portions of code & timing

Application: Lessons Learned

- Memory accesses take majority of time on GPU. Potential future improvements?

- Manual load balancing works, but we would like runtime support for instrumented heterogeneous load balancing.

- Finite element simulations with large amount of data per element achieve modest speedups on GPU.

The End

Questions?

Suggestions?

Isaac Dooley
idooley2@uiuc.edu