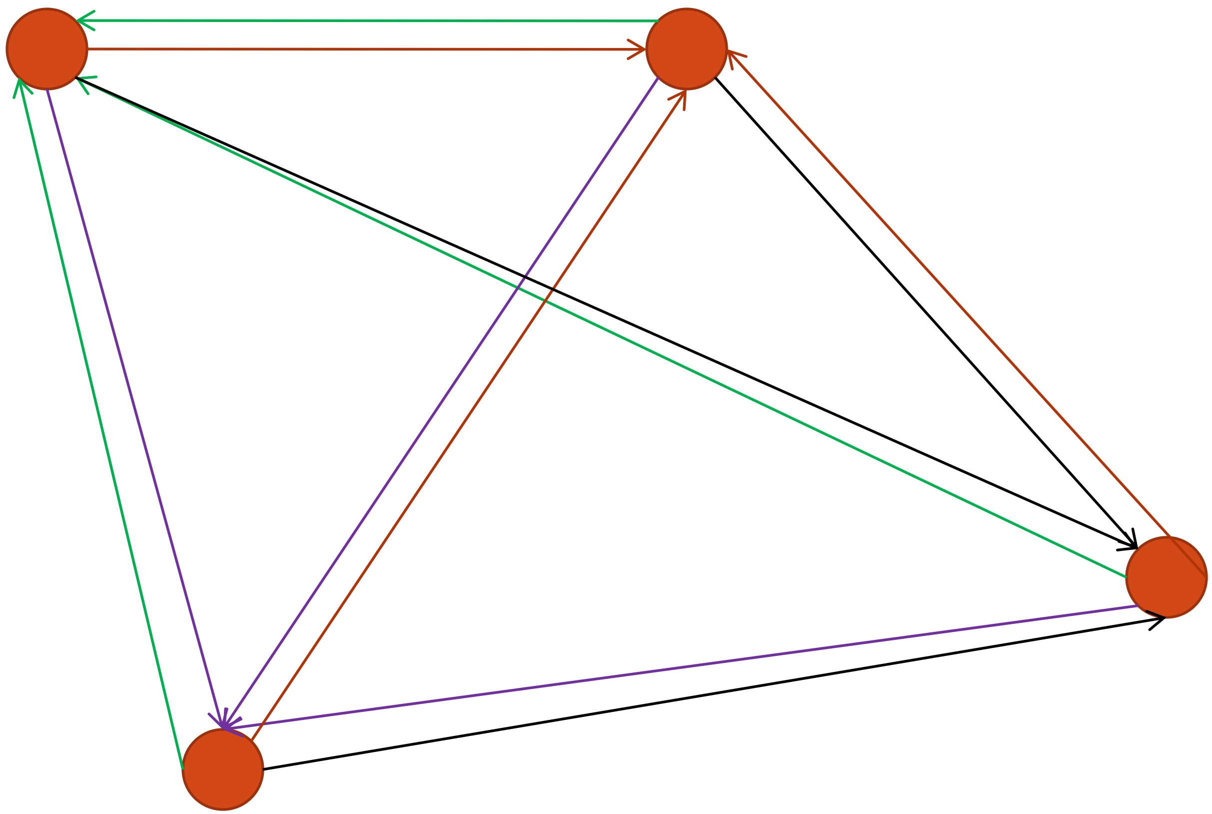
N-Body Simulation using CUDA

CSE 633 Fall 2010
Project by Suraj Alungal Balchand
Advisor: Dr. Russ Miller
State University of New York at Buffalo

Project plan

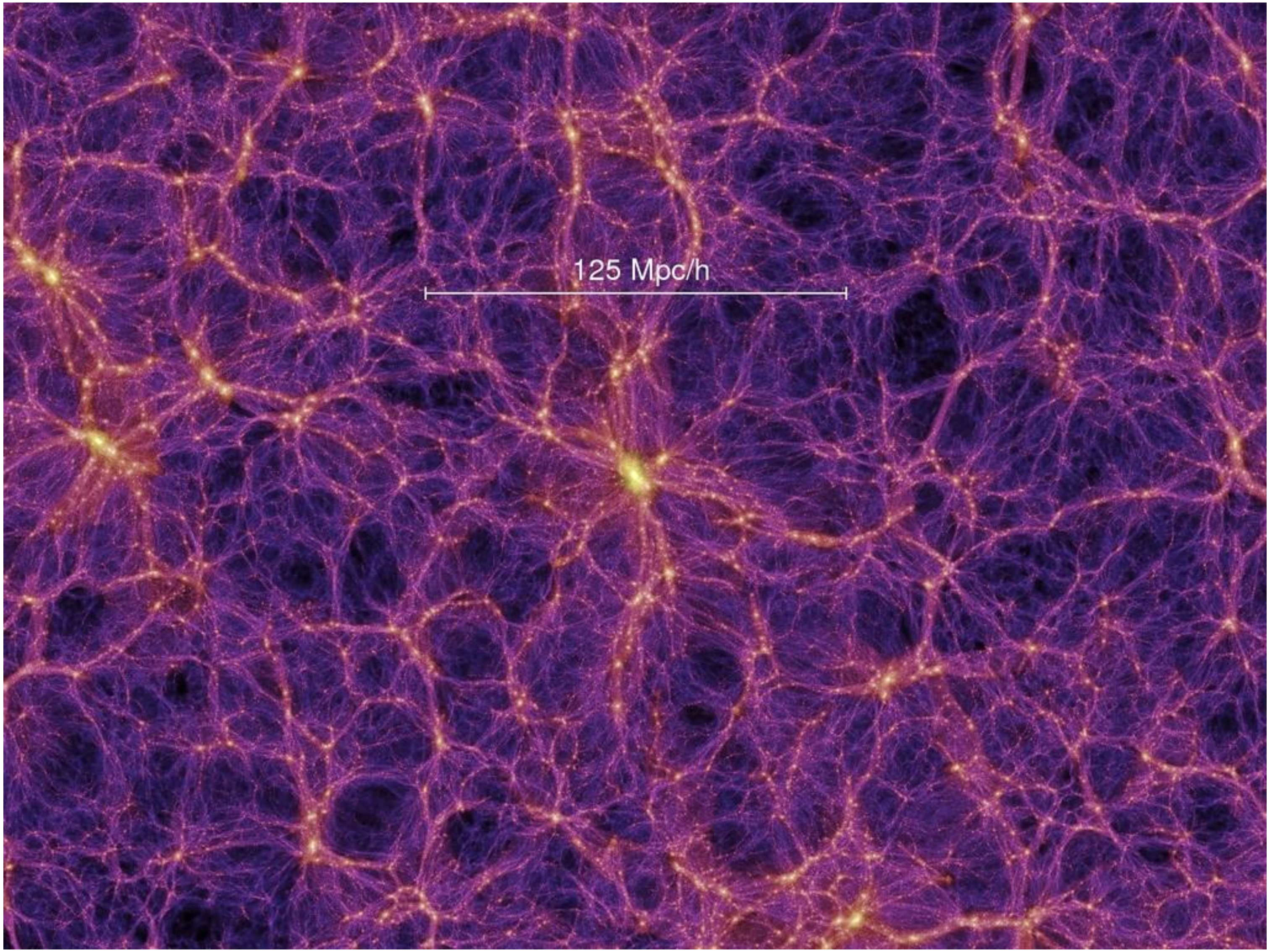
- Develop a program to simulate gravitational forces between *n* bodies in space
- Exploit the massively parallel architecture provided by GPGPUs.
- Compare performance with equivalent openMP and sequential code

Simple *n-body* scenario with n=4 bodies



This can get very complicated…

A *not so simple* N body simulation..



~ 10 billion particles

Millennium Run

The Equation

$$\vec{F}_i = -\sum_{j \neq i} \frac{G m_i m_j (\vec{r}_i - \vec{r}_j)}{(|\vec{r}_i - \vec{r}_j|^2 + \epsilon^2)^{3/2}},$$

$\mathbf{F_i}$ - Force on particle *i*
$\mathbf{m_i}$ – Mass of particle *i*
$\mathbf{m_j}$ – Mass of particle *j*
$\mathbf{r_i}$ – Direction vector for particle *i*
$\mathbf{r_j}$ – Direction vector for particle *j*
$\boldsymbol{\varepsilon}$ - Softening factor

*Assuming that the other fundamental forces of interaction do not influence the system as much as gravity.

Parallelism

- The above equation suggests that the cumulative effect of *n-1* particles on a single particle can be approximated independently for each time step.
- A problem in the parallel computing domain
- nVidia's CUDA allows for massive parallelism.
- Multiple CUDA-enabled devices could also be used for extremely large simulations (E.g.)

Advantages of CUDA

- Each GPGPU is effectively a mini-supercomputer
- For cards that support Compute Capability > 1.2:
  - Each Streaming Multiprocessor (SM) allows for 1024 resident threads (employs latency hiding techniques).
  - Each C1060 GPGPU (on Magic cluster) has 30 SMs.
- Shared Memory architecture built into each SM allows for significant performance gain by reducing the global (device) memory access.
- Memory coalescing allows for good data locality improving performance.
- CUDA threads are lightweight compared to CPU threads and easy to schedule.

Algorithm used:

- All-pairs – calculation of all possible combinations (brute force method – $O(n^2)$)

  - Most accurate values
  - Every particle-particle interaction is calculated
  - Computationally intensive
  - Not necessary in most cases
  - Variable Time-Step schemes can save some of the computations involved.

- Improvement: Barnes Hut Algorithm

CUDA Implementation

- In all-pairs algorithm, force on each body is the sum of acceleration caused by every other particle multiplied by the mass of that body.

- Forces on a single body is independently calculated by a single thread.
  - (Concurrent memory access allows for information of every other body to accessed by every thread)

- The sum of all accelerations are calculated, and further used to calculate the velocity and new position of each particle.

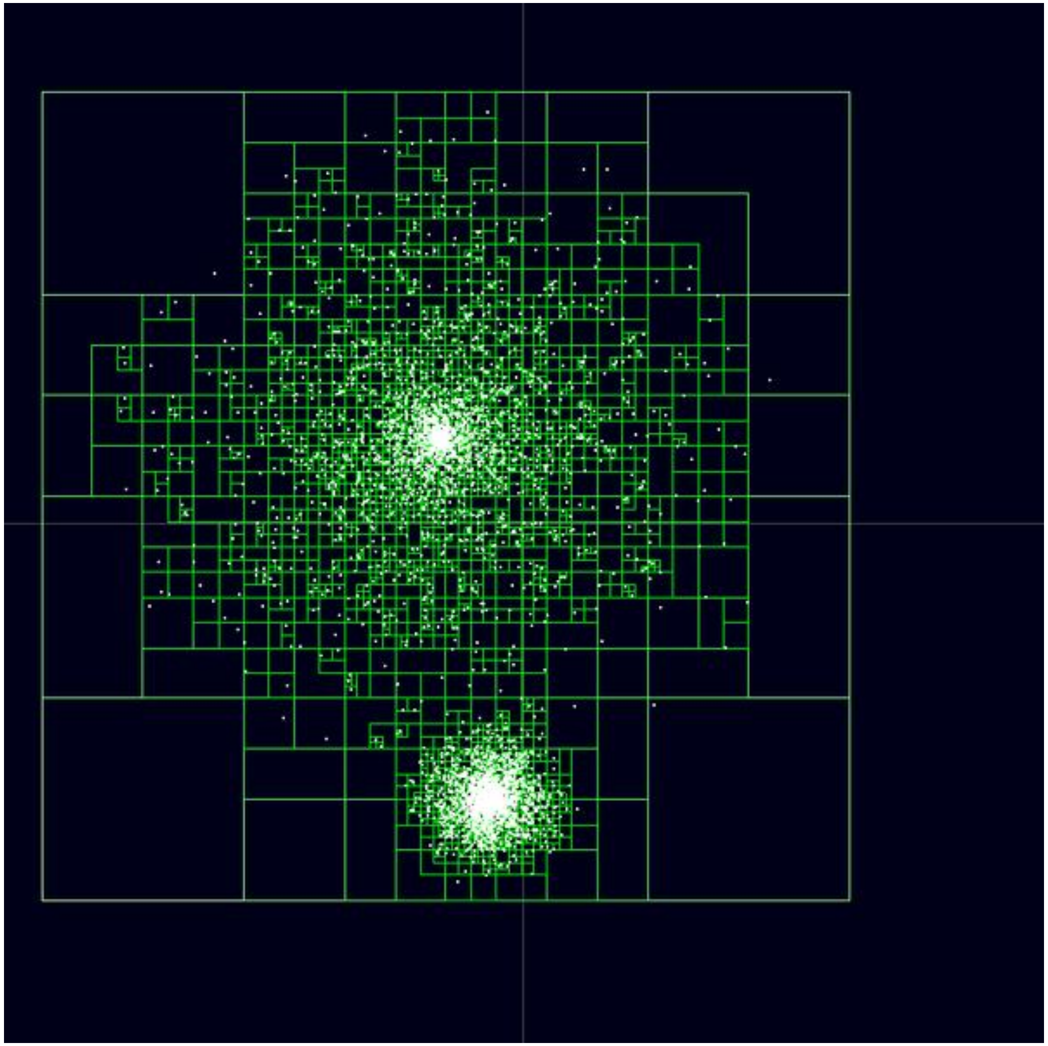
CUDA implementation

- Position and velocity of each particle is updated per time-step.
- Coalesced memory used to store location, mass & velocity of body.
- Shared memory structure used to optimize calculations by having each thread in a block copy one value from the device memory into the shared memory, reducing the total number of device memory accesses.

Barnes Hut Tree code algorithm

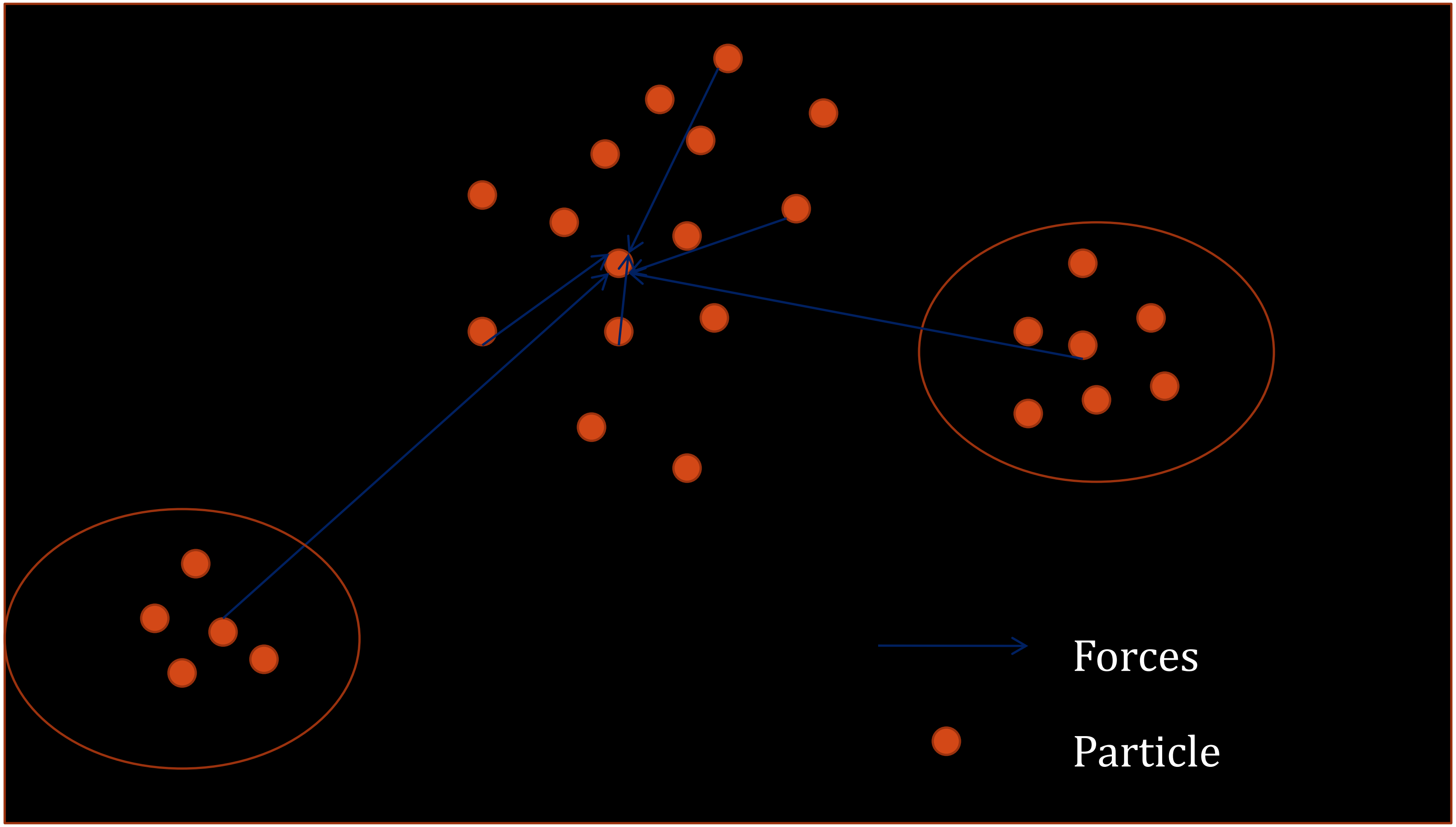
- To be implemented as the next phase of this project.
- Runtime – O($nlogn$)
- Maps the data on to a quad- or oct- tree structure which divides computational region into smaller and smaller regions
- Calculation of forces on a particle is carried out by traversing tree elements close, in detail. The particles farther away are explored only in coarse detail.
- Space devoid of particles is not simulated, which is an additional saving.

How the algorithm partitions data



Courtesy: [Wiki/Barnes-Hut_Simulation]()

Simple Scenario – End effect



Objects, relatively, far away are considered as a single entity to reduce calculations.

Current Status

- Developed an all-pairs program to simulate gravitational forces between *n* bodies in space

- Compared performance of algorithm on:
  - CUDA flavors:
    - Geforce 240M – 48 cores, compute 1.2, 1GB device memory
    - Tesla C1060 – 240 cores, compute 1.3, 3GB device memory

  - Sequential code (CPU, Intel Core2Duo P8700 ~2.53Ghz, 4GB RAM)

  - Open MP code - Edge Cluster – 1 node, 8 processors

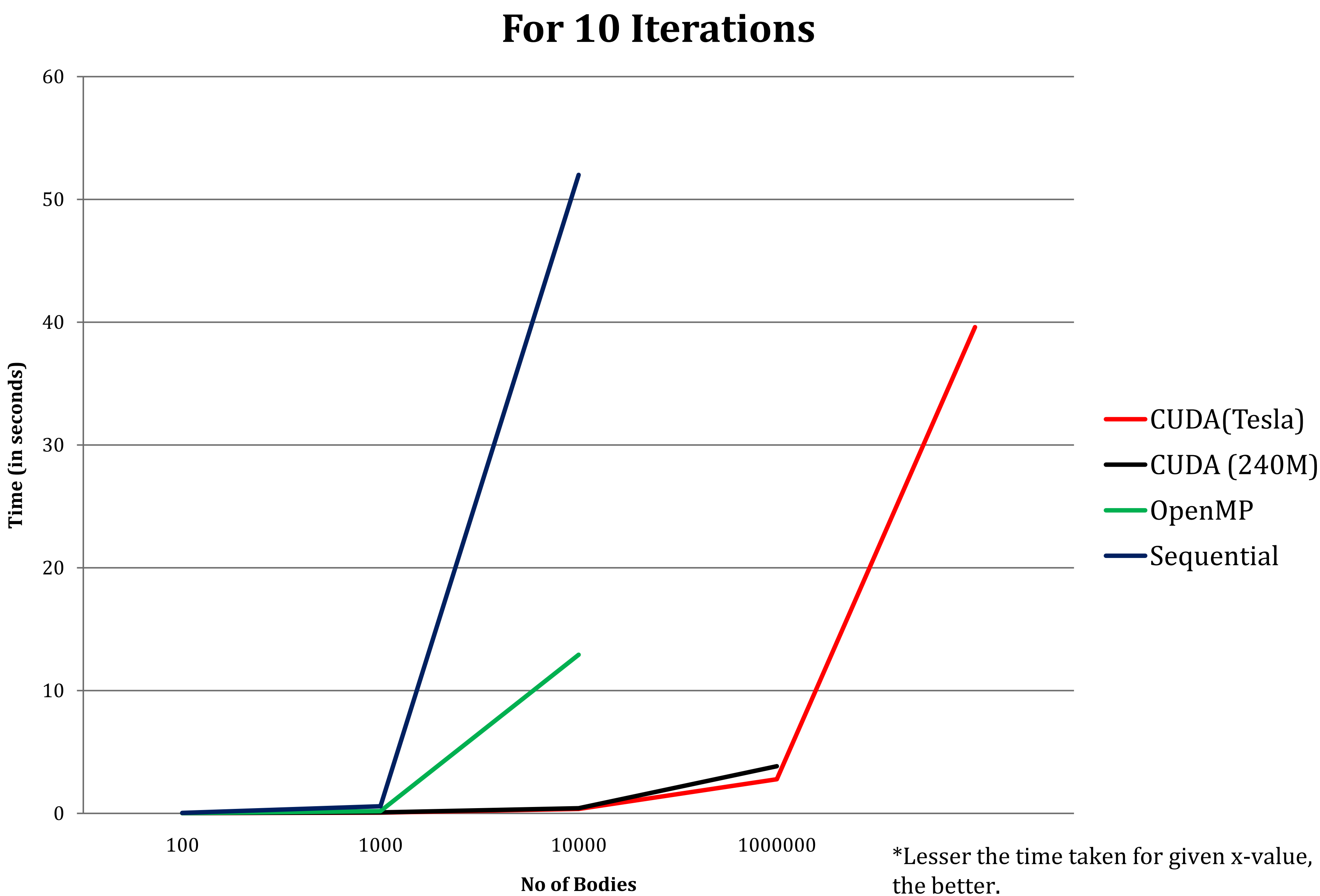
Problems faced

- CUDA / Visual Studio integration was tricky
- Data structure manipulation (Arrays) on device (global) memory is not as easily accomplished as on Host RAM.
- Cannot be absolutely sure of the accuracy output as debugging toolkit has not been installed yet.
  - Magic compute cluster runs linux, and nVidia's new tool – nSight is not available for the platform.
  - Visualization not added yet.

Verifying the output:

- Manually calculated values for 3 bodies on paper
  - Matches GPU output
  - Extending the result to all cases!
- For all the work done, the output is indeed impressive…

A comparison:



*Lesser the time taken for given x-value, the better.

The rest of the data…

		Iterations			
Platform	**Body Count**	**5**	**10**	**20**	**50**
CUDA (Tesla)	480		0.015	0.028	0.067
CUDA (Tesla)	1920		0.06	0.15	0.39
CUDA (Tesla)	7680		0.35	0.73	1.86
CUDA (Tesla)	**30720**		2.779	6.43	**16.119**
CUDA (Tesla)	**122880**	19.85	**39.6**		
CUDA (240M)	384		0.016	0.03	0.81
CUDA (240M)	1536		0.063	0.123	0.303
CUDA (240M)	6144		0.42	0.81	2.057
CUDA (240M)	**24576**		3.84	7.67	**19.07**
OpenMP (Edge)8 cores	100		0.022	0.025	0.036
OpenMP (Edge)8 cores	1000		0.176	0.337	0.784
OpenMP (Edge)8 cores	10000		12.91	25.87	64.688
OpenMP (Edge)8 cores	**100000**	**643.53**			
Sequential	100		0.025	0.029	0.085
Sequential	1000		0.578	1.143	4.85
Sequential	**10000**		**52.01**	104.23	616.93

Future Work: Next Semester

- Implement solutions for the computationally efficient Barnes Hut Tree code algorithm
  - Implementing tree structure is complicated.
    - Load Balancing the tree across processors
- Create visualization using a graphics engine
- If possible, implement on the new M2050 GPGPU cluster being installed at CCR .
- Also, the cloud-compute option available with Amazon.
- Scaling issues – have to get hardware information at runtime to ensure proper scaling from my 240M graphics card to Tesla C1060 card.
  - Currently using a header file to manually tweak block and grid size for each GPU

Conclusions:

- CUDA provides an impressive hardware layer to execute extremely parallel applications.
  - CUDA enabled GPUs really perform when pushed to the limits (upwards of 10000 threads per GPU). It also depends on leveraging the compute-specifications
    - Correct Block size
    - Shared memory tiles
    - Grid design

- CUDA is still a developing technology, but given the cost to power ratio, it is already ahead of the previous parallel architectures in use.

- Can be difficult to use at first as it gives programmers all the flexibility in scheduling the threads, handling memory.
  - This can be a boon and a bane.

References

- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html
- http://en.wikipedia.org/wiki/CUDA
- http://www.ifa.hawaii.edu/~barnes/treecode/treeguide.html
- http://www.scholarpedia.org/article/N-body_simulations
- http://www.sns.ias.edu/~piet/act/comp/algorithms/starter/index.html
- http://www.amara.com/papers/nbody.html
- http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation

Questions?