

Dynamic Load Balancing of the Adaptive Fast Multipole Method in Heterogeneous Systems

Robert Overman, Jan Prins, Laura
Miller, Michael Minion

What is this talk about?

- Time-dependent simulation of physical systems
 - Solve N-body problem
 - Fast Multipole Method (FMM) - $O(N)$

Key points:

- Adaptive decomposition of space
- Targets heterogeneous, shared memory node
 - Multi-core CPUs, Multiple Discrete GPUs
- Dynamic Load Balancing

Problem

N-body Problem

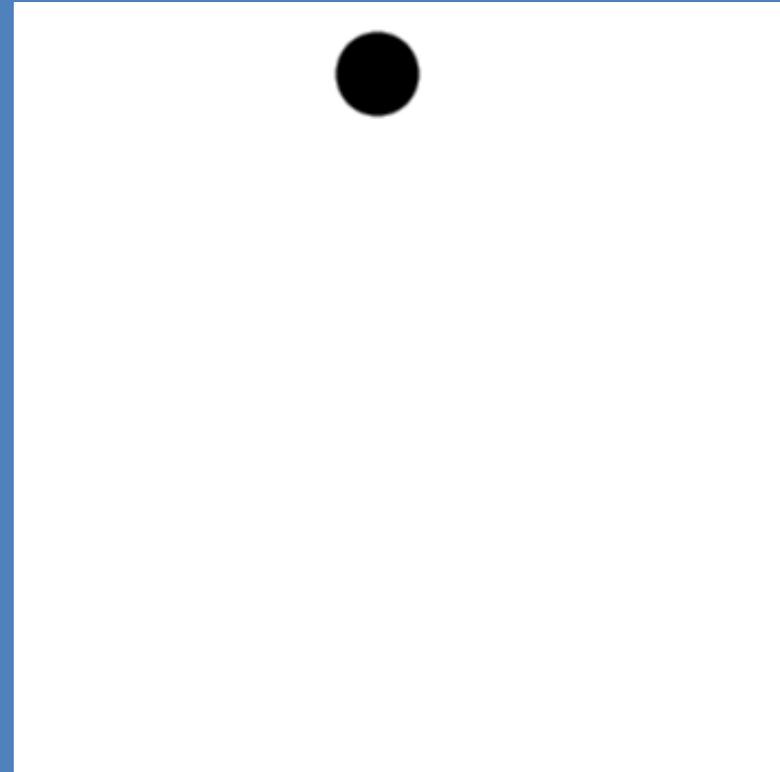
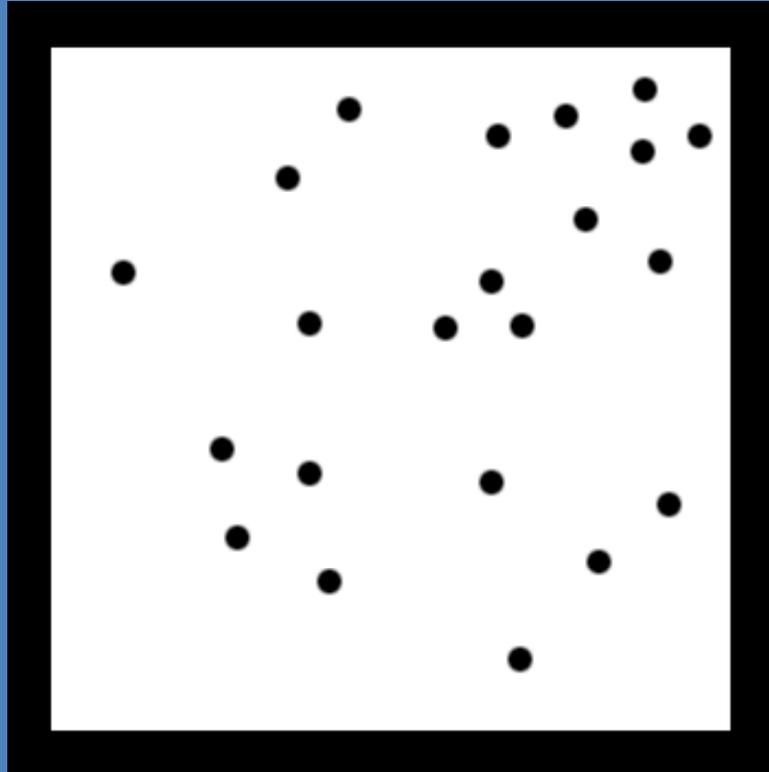
For each body x_i , $i \in [1 \dots N]$ compute $F(x_i)$

$$F(x_i) = \sum_{j=1}^N f(x_i, x_j)$$

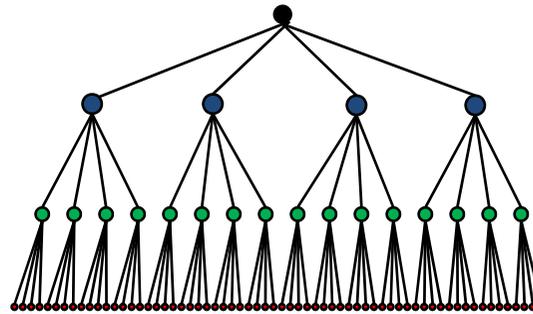
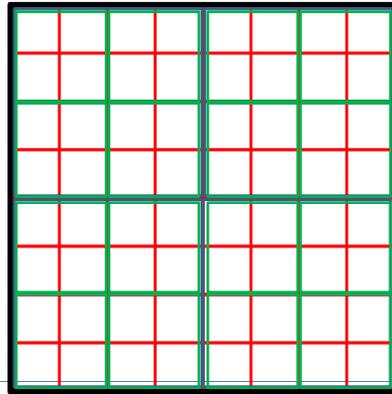
Gravitational Potential

$$f(x_i, x_j) = G \frac{m_i m_j}{r_{ij}^2}$$

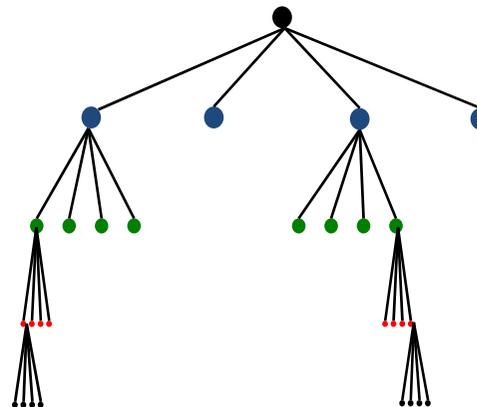
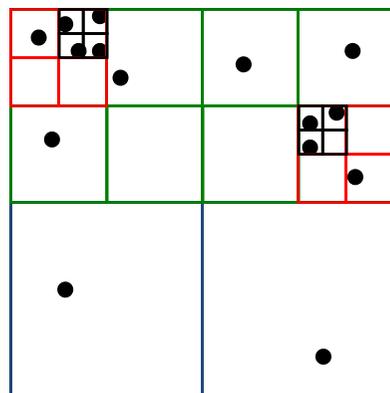
Spatial Decomposition



Spatial Decompositions



Uniform



Adaptive

Simulation Video

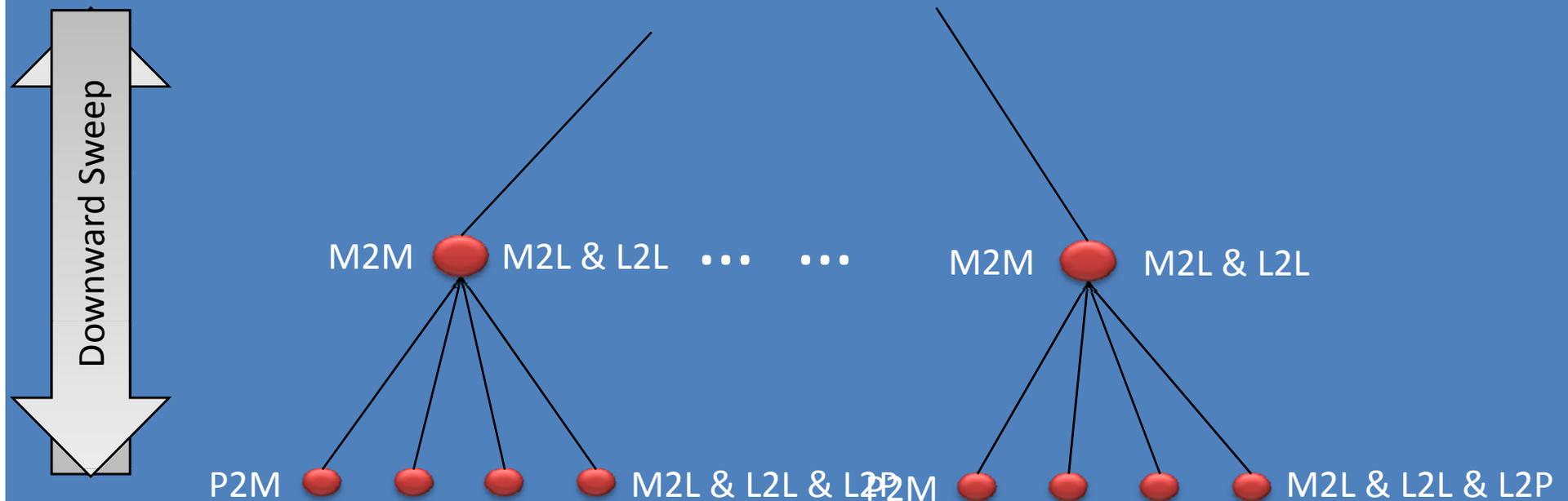
FMM Overview

Hierarchical Method

- Far-field
- Near-field

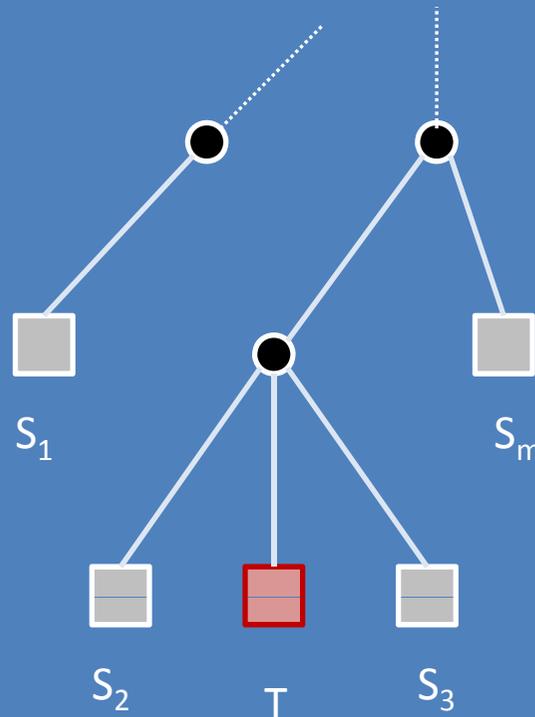
FMM Overview

Multiple expansions of a box approximates the total force due to all the detail in boxes at a distance in that box



FMM Overview

- Nodes insufficiently separated from T form the near-field of T
- Near-field interactions for a target node T are defined in terms of a list of sources nodes



Parallelization

Heterogeneous division of work

- CPU handles far-field (P2M, M2M, M2L, L2L, L2P)
- GPU performs near-field (P2P)

Time Integrator and body movement on CPU

Need three types of load balance

- Far-field over multi-core CPUs
- Near-field over multiple GPUs
- CPU-GPU

CPU Load Balance

- OpenMP tasking facilities
- Recursively traverse octree data structure
- Spawn tasks on recursion
- Spawning specifics and therein load balancing determined by OpenMP runtime

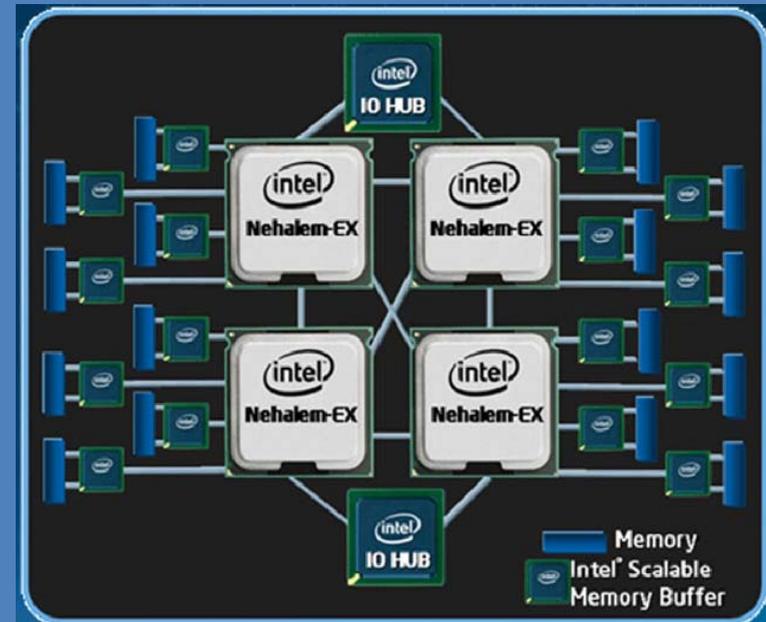
CPU Scaling

Test system

- Intel X7560 Nehalem-EX
- 32 cores over 4 sockets
- Each socket has 18MB L3 on chip

Results:

- Superlinear speedup through 16 cores
- Max of 28.6x with 32 cores



GPU computation of direct interaction

All computations for a single target particle are performed by a single CUDA thread.

Say there are three threads per CUDA block → three blocks to compute for the eight target particles.

Threads in a block are SIMD, compute in lock step. Step serially through the source particles

- Computing
- Completed

S_2

S_4

S_{167}

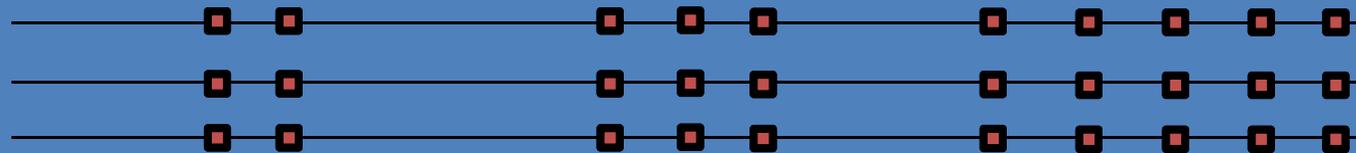
$S_9 S_{10}$

$S_{21} S_{22} S_{23}$

$S_{340} S_{341} S_{342} S_{343} S_{344}$

$Thread_{00}$
 $Thread_{01}$
 $Thread_{02}$

t_1
 t_2
 t_3



t_4
 t_5
 t_6
 t_7
 t_8

Other blocks compute the same way. There may be unused threads. Source particles are loaded in parallel into shared memory, one per thread.

Load Balance Over GPUs

Total work:

$$\sum_{t \in L} \sum_{s \in \text{NF}(t)} |t||s|$$

Where L is the set of leaves for the octree

NF(t) is the near field of node t

- Total work is equally divided between GPUs such that all computations for a given target are performed on the same GPU

GPU Scaling

- Test System
 - Two Intel® Xeon® Processor X5670
 - 2.93 GHz, 6 cores each
 - Four Tesla C2050's
 - 3Gb GDDR5
 - 448 Cuda Cores (14 SMs)
 - Single Precision: 1.03 Tflop
 - ECC

Plummer distribution of 10 Million particles

No. GPUs	Speedup
2	1.99
3	2.96
4	3.95

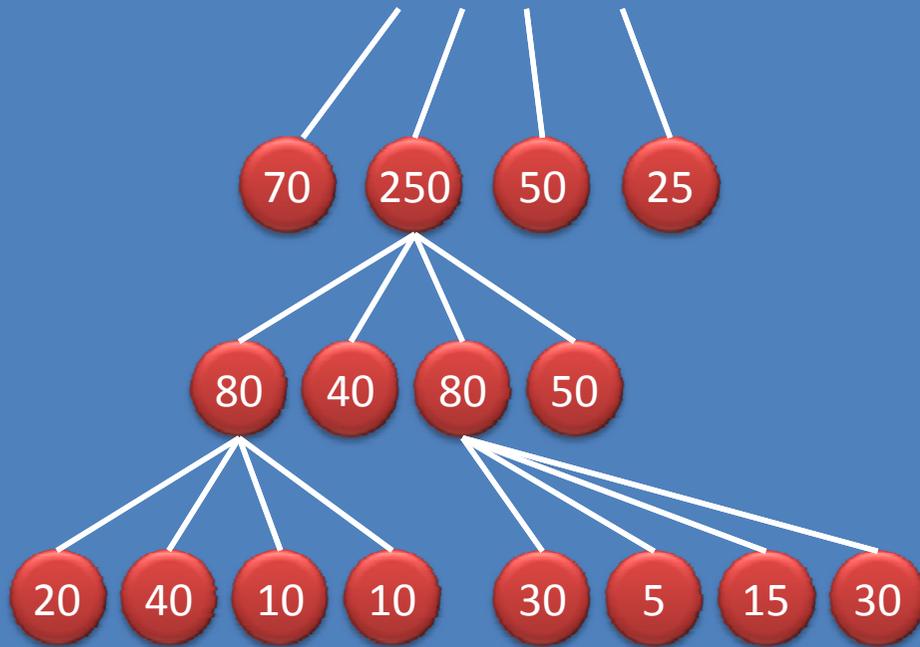
CPU-GPU Load Balance

Factors determining far-field and near-field work:

- N – problem size
- P – Terms in Multipole Expansion
- S – max number of bodies per node
- Distribution – e.g. Plummer/Uniform

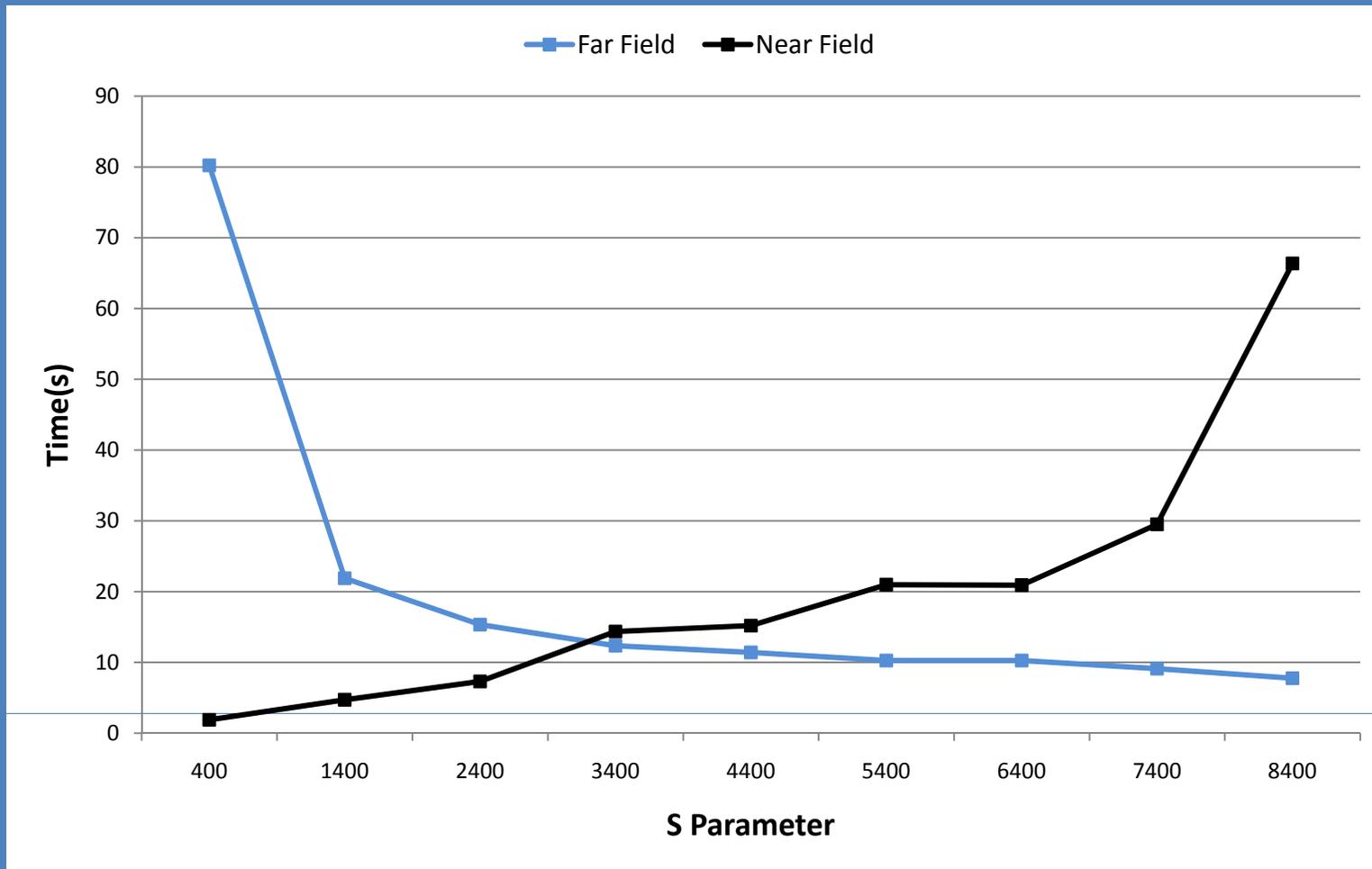
Solution: Can convert Far-field \leftrightarrow Near-field work by varying S parameter

Effect of S

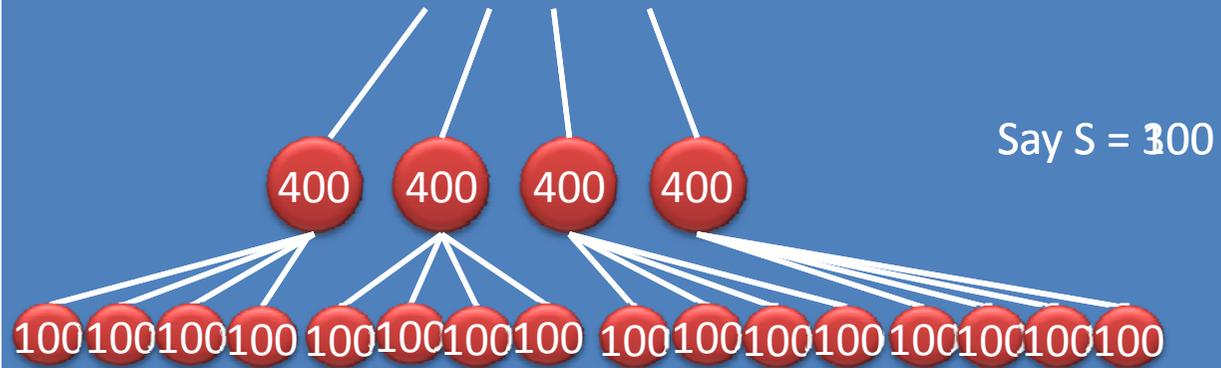


Say $S = 100$

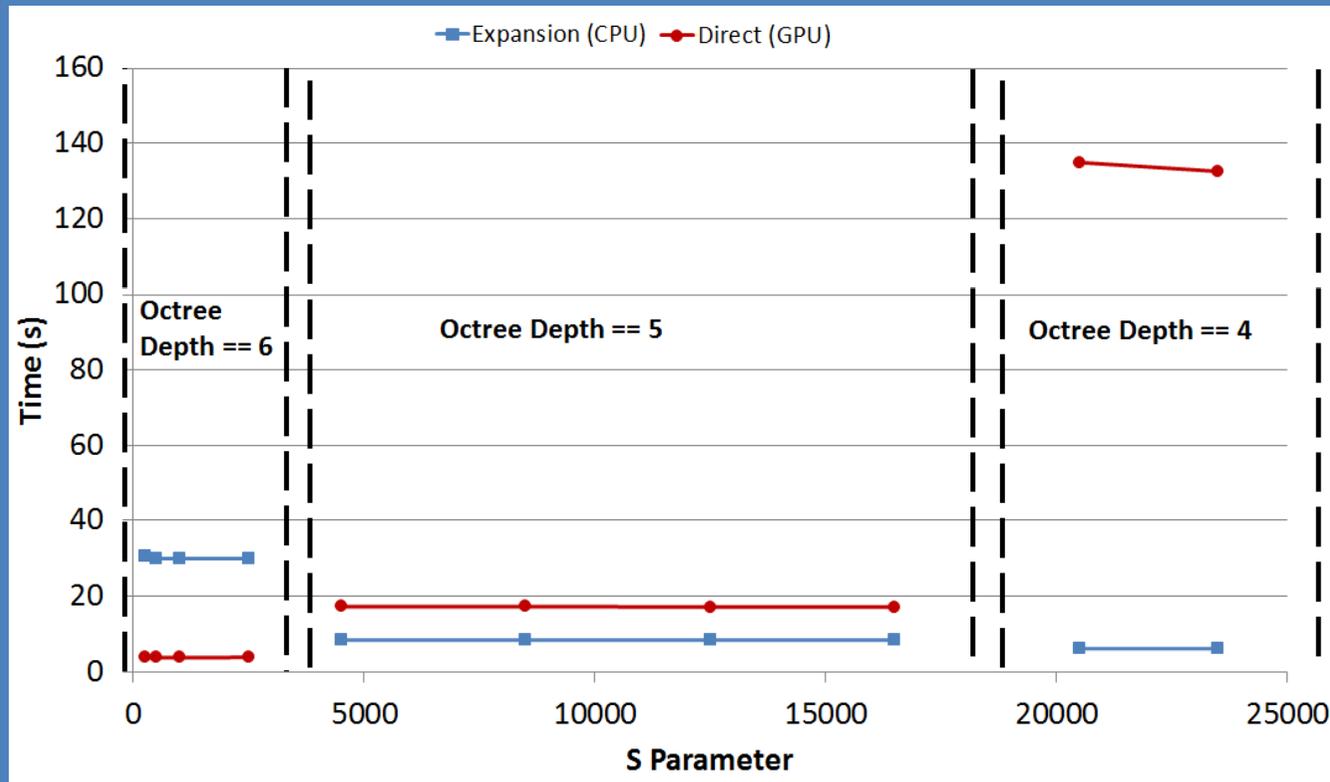
Expected Trend



Effect of S



Cost Tradeoff in Uniform Distributions



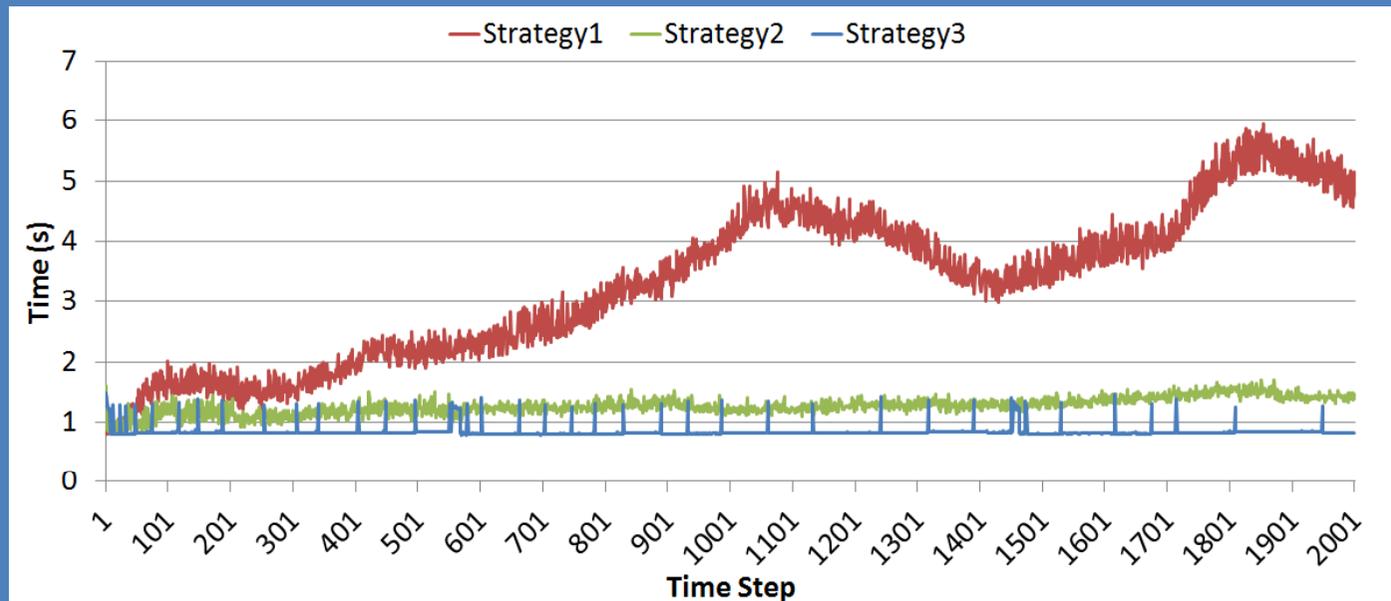
Can occur in uniform FMM with any distribution or uniform distribution in Adaptive FMM

Adaptive structure of the implementation combined with our local modifications can bridge the performance gap between levels

Vary Distribution of Work

Experimental comparison of 3 strategies

- Pick a good S at start keep octree fixed (Strategy1)
- Fixed S varying octree decomposition (Strategy2)
- Varying S and varying octree (Strategy3)
 - Fast tree rebuild



Finding Initial Settings

Make one global S modification per time step and react to results

- Binary Search
- Incremental

Design primarily driven by GPU efficiency

- Too little work in total (no latency hiding) or
- Too little work per block
- Don't want to predict with bad view of GPU efficiency

Slow change of workload from time step to time step

Adding Local Modifications

- Operate on very local regions of tree
- Ignore global S value
- Two simple operations
 - Collapse
 - PushDown

Local Tree Modification Operations

Collapse Operation



Local Tree Modification Operations

PushDown Operation



Adding Time Prediction

Time prediction used when applying fine grained operations

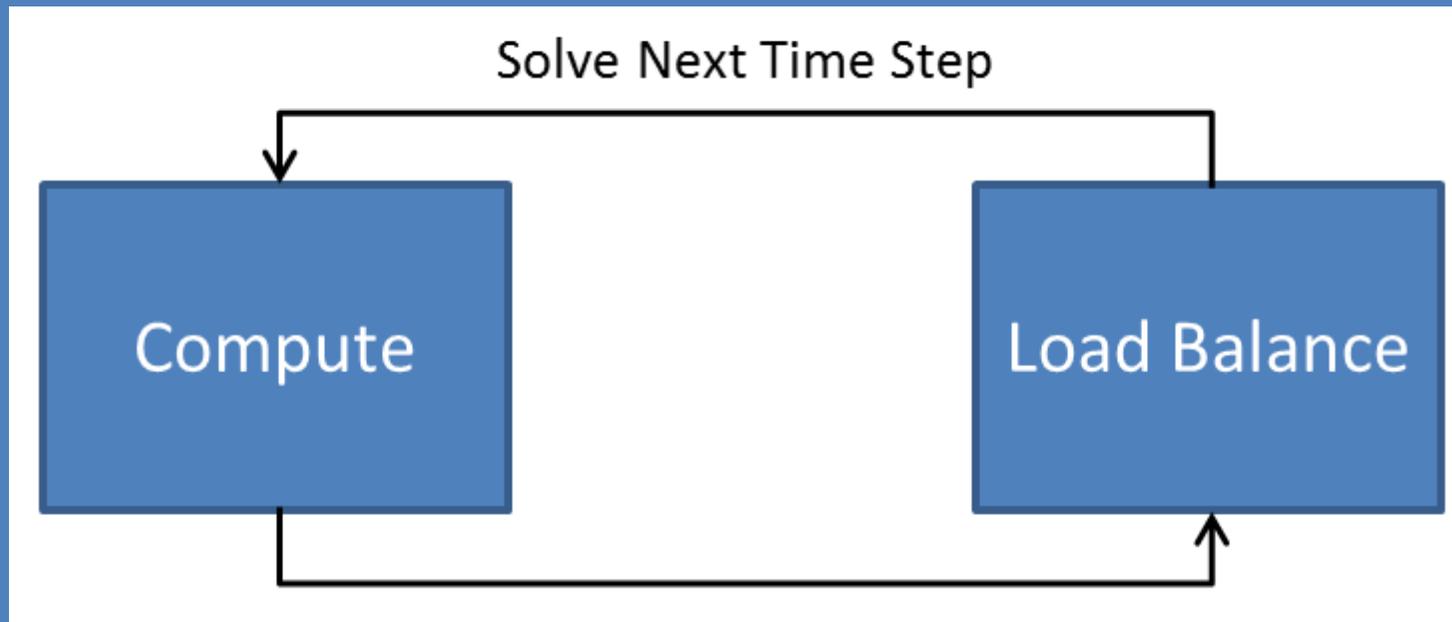
$$\text{CPU Time Prediction} = \sum_{Op \in F} C(Op) * M(Op)$$

where $F = \{P2M, M2M, M2L, L2L, L2P\}$

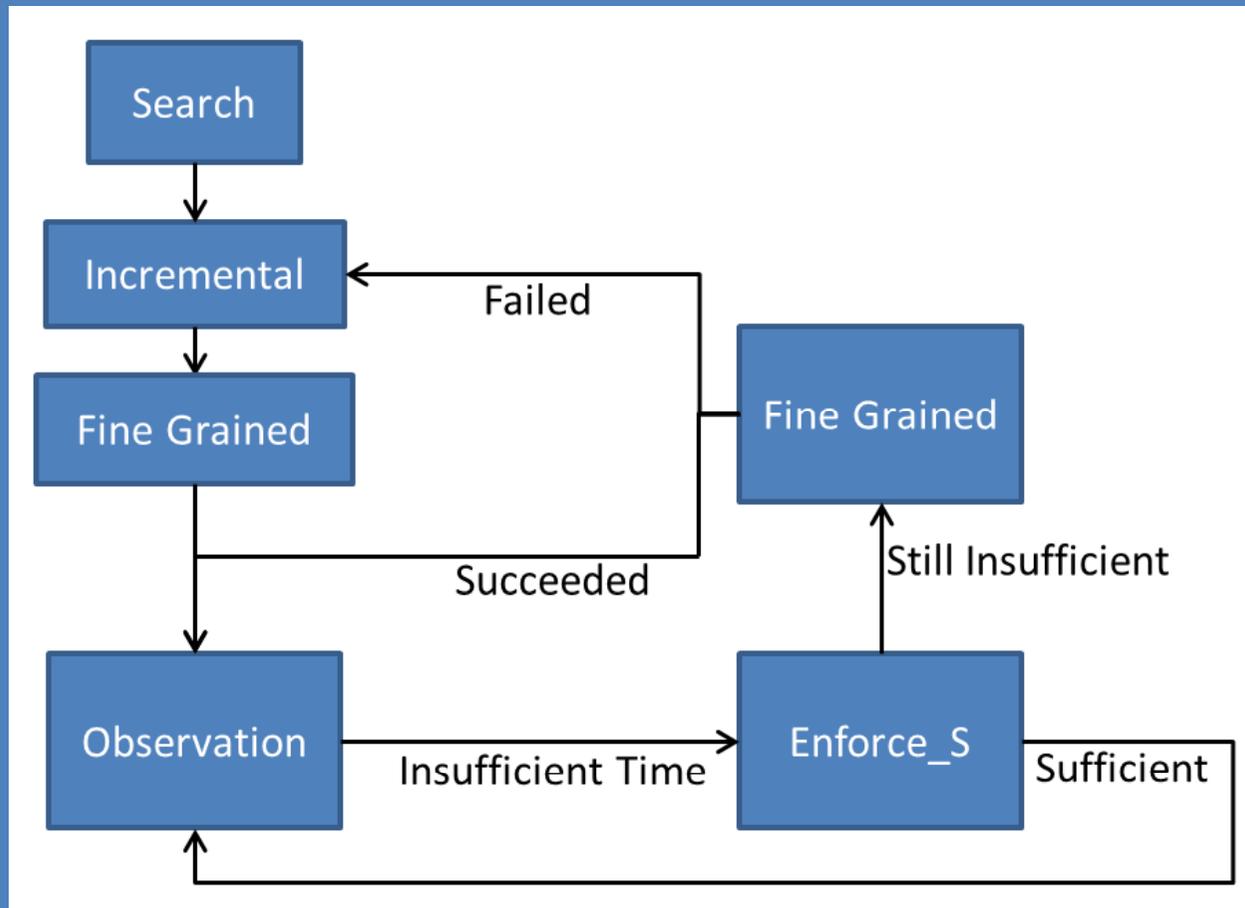
$$\text{GPU Time Prediction} = C(P2P) * M(P2P)$$

- $C(Op)$ derived by observation
- After initial search phases we are confident enough to predict – good GPU coefficient

Time Dependent Modifications – State Machine



State Machine

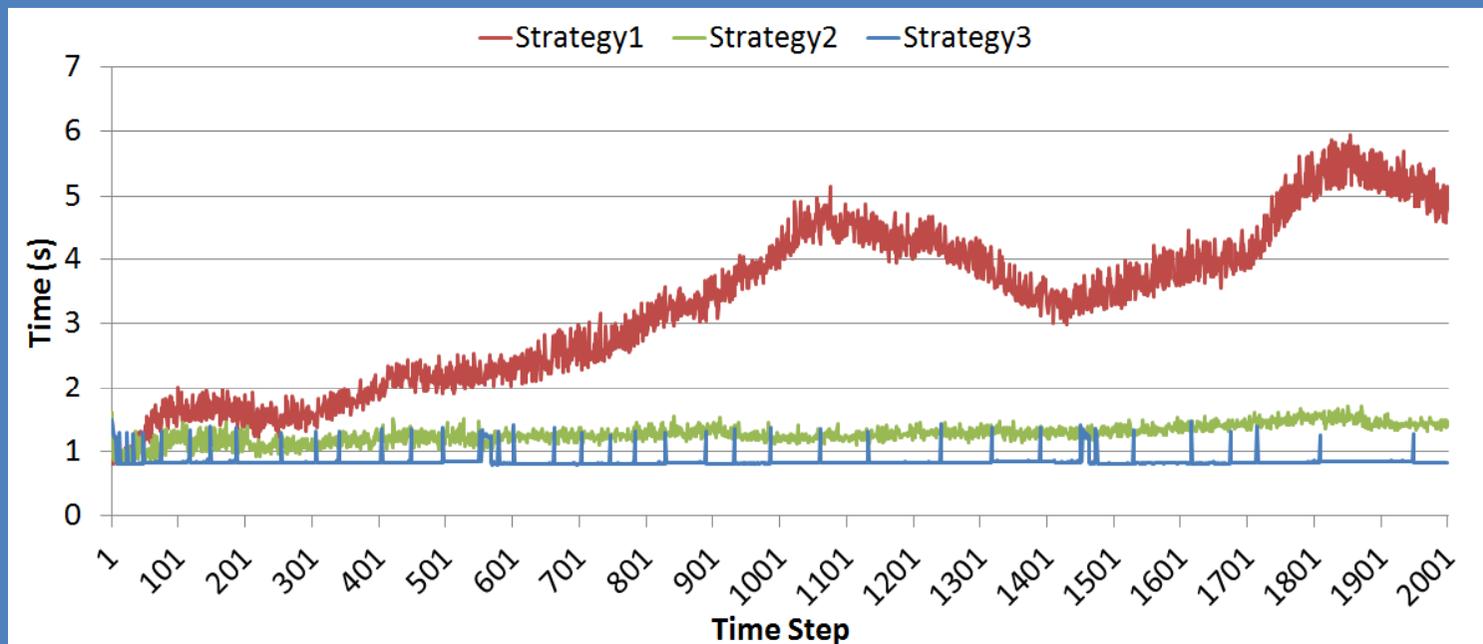


*Insufficient implies outside of 5% deviation from best time

Load Balancing Summary

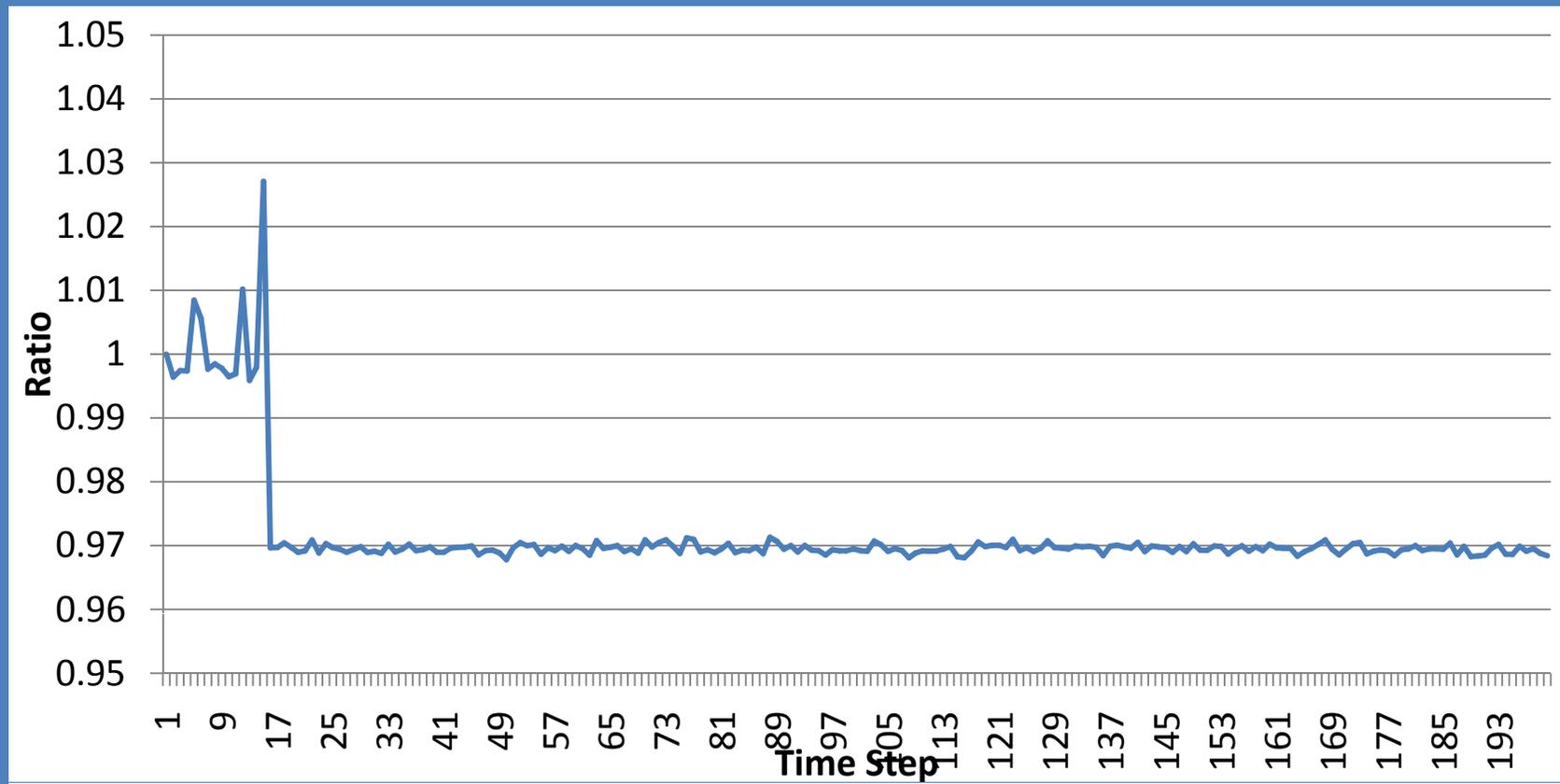
Strategy	Total Compute	Total LB	LB as % of Compute	Relative cost per time step
1	6576.17s	1.32s	0.02%	3.91
2	2544.79s	2.78s	0.11%	1.51
3	1651.57s	30.98s	1.88%	1.00

Plummer distribution of 1Million particles



Uniform Distribution Result

Shown is ratio of time observed when using local modification to time observed when not using



Conclusions

- Time-dependent simulations N-body simulations using FMM on heterogeneous platforms
 - Accommodate arbitrary distribution of bodies
 - Continuous load balancing throughout simulation which minimizes cost of time step
 - 100x speedup on one million Plummer (10CPU, 4GPU) over best serial
- Extension to larger simulations
 - Parallel in time methods such as PFASST*

*R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, P. Gibbon, "A massively space-time parallel N-body solver" Supercomputing 2012