Scaling Hierarchical *N*-Body Simulations on GPU Clusters

Pritish Jetley Lukasz Wesolowski Filippo Gioachin
Laxmikant V. Kalé Thomas R. Quinn

April 29, 2010

Motivation

- Clusters of GPUs provide immense computational power

Motivation

- Clusters of GPUs provide immense computational power
- Suitable for well-structured data parallel operations

Motivation

- Clusters of GPUs provide immense computational power
- Suitable for well-structured data parallel operations
- Algorithms with high flop intensity do well

Motivation

- Clusters of GPUs provide immense computational power
- Suitable for well-structured data parallel operations
- Algorithms with high flop intensity do well
- What about complex, asynchronous applications with medium grain size?

Motivation

- Clusters of GPUs provide immense computational power
- Suitable for well-structured data parallel operations
- Algorithms with high flop intensity do well
- What about complex, asynchronous applications with medium grain size?
  - How can we optimize kernel performance?

Motivation

- Clusters of GPUs provide immense computational power
- Suitable for well-structured data parallel operations
- Algorithms with high flop intensity do well
- What about complex, asynchronous applications with medium grain size?
    - How can we optimize kernel performance?
    - What are the obstacles to scaling on clusters of GPUs?

ChaNGa

- Barnes-Hut simulator

ChaNGa

- Barnes-Hut simulator
  - Tree traversal
  - Force computation

ChaNGa

- Barnes-Hut simulator
  - Tree traversal
  - Force computation
- Several production-quality techniques
  - Ewald summation
  - SPH
  - Gravitational softening
  - Quadrupole moments

ChaNGa

- Barnes-Hut simulator
  - Tree traversal
  - Force computation
- Several production-quality techniques
  - Ewald summation
  - SPH
  - Gravitational softening
  - Quadrupole moments
- Multiple timestepping

ChaNGa

- Barnes-Hut simulator
    - Tree traversal
    - Force computation
- Several production-quality techniques
    - Ewald summation
    - SPH
    - Gravitational softening
    - Quadrupole moments
- Multiple timestepping
- Optimized for parallel performance
    - Particle cache
    - Prefetching
    - Overlap of fetch latency with useful work
    - Scales up to 32K cores

GPU Manager

- Work-request (WR) abstraction

GPU Manager

- Work-request (WR) abstraction
- Asynchronous invocation-callback paradigm

GPU Manager

- Work-request (WR) abstraction
- Asynchronous invocation-callback paradigm
- Asynchronous memory transfer and kernel invocation

Adapting ChaNGa to the GPU

- Work accrual framework

Adapting ChaNGa to the GPU

- Work accrual framework
- Kernel structure

Adapting ChaNGa to the GPU

- Work accrual framework
- Kernel structure
- Balance CPU and GPU work

Adapting ChaNGa to the GPU

- Work accrual framework
- Kernel structure
- Balance CPU and GPU work
- Overlap tasks

Adapting ChaNGa to the GPU

- Work accrual framework
- Kernel structure
- Balance CPU and GPU work
- Overlap tasks
- Reduce serial overheads

Force Kernel Organization

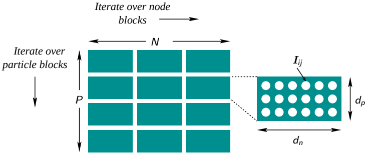


Figure: Organization of force computation kernels.

- Threads per block

Force Kernel Organization

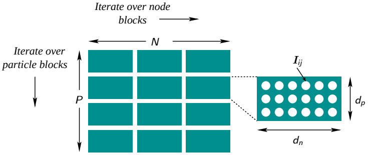


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency

Force Kernel Organization

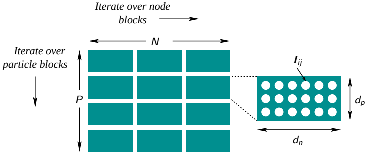


Figure: Organization of force computation kernels.

- Threads per block
    - More threads ⇒ more concurrency
    - More threads ⇒ fewer blocks/SM

Force Kernel Organization

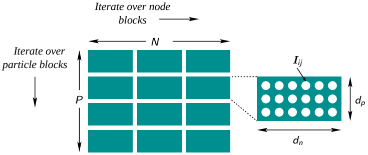


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency
  - More threads ⇒ fewer blocks/SM
  - Optimal value?

Force Kernel Organization

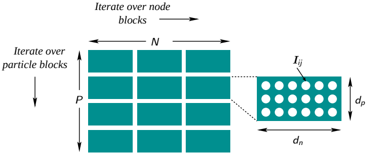


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency
  - More threads ⇒ fewer blocks/SM
  - Optimal value?
- Block shape

Force Kernel Organization

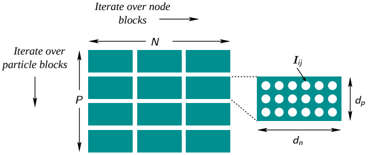


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency
  - More threads ⇒ fewer blocks/SM
  - Optimal value?
- Block shape
  - More particles ⇒ fewer loads

Force Kernel Organization

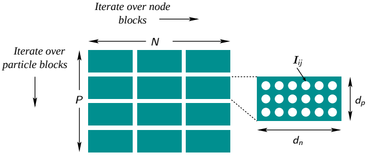


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency
  - More threads ⇒ fewer blocks/SM
  - Optimal value?
- Block shape
  - More particles ⇒ fewer loads
  - More particles ⇒ more shared memory/block

Force Kernel Organization

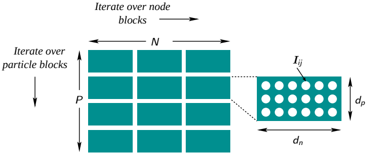
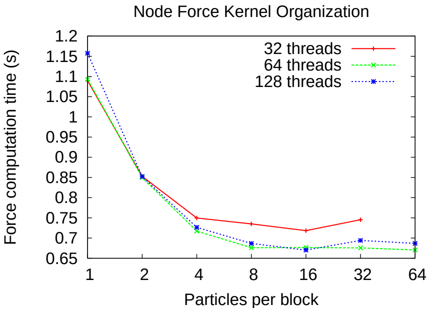
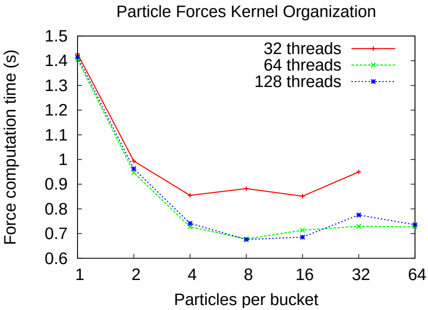


Figure: Organization of force computation kernels.

- Threads per block
  - More threads ⇒ more concurrency
  - More threads ⇒ fewer blocks/SM
  - Optimal value?
- Block shape
  - More particles ⇒ fewer loads
  - More particles ⇒ more shared memory/block
  - Optimal shape?

Experimental Results





- Works best with $T = 128$, 16 particles, 8 nodes per block

Ewald Computation

- Structured as two (real and Fourier space) kernels
  - Fewer registers per thread
  - More blocks per SM
- Constant memory used in Fourier-space
- Speedup of about 20 over CPU

Balancing Tree Traversal and Computation

- GPU is hungry for work

Balancing Tree Traversal and Computation

- GPU is hungry for work
  - CPU shouldn't hold back GPU

Balancing Tree Traversal and Computation

- GPU is hungry for work
  - CPU shouldn't hold back GPU
  - Spend less time traversing tree, more time computing

Balancing Tree Traversal and Computation

- GPU is hungry for work
  - CPU shouldn't hold back GPU
  - Spend less time traversing tree, more time computing
- Increase average bucket size

Balancing Tree Traversal and Computation

- GPU is hungry for work
  - CPU shouldn't hold back GPU
  - Spend less time traversing tree, more time computing
- Increase average bucket size
  - Tree is shallower: less traversal time on CPU

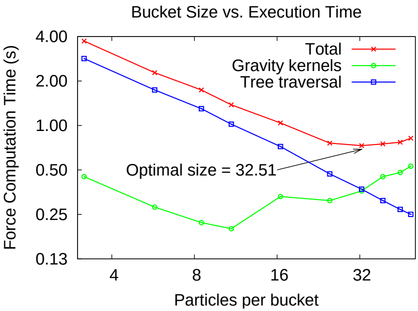
Balancing Tree Traversal and Computation

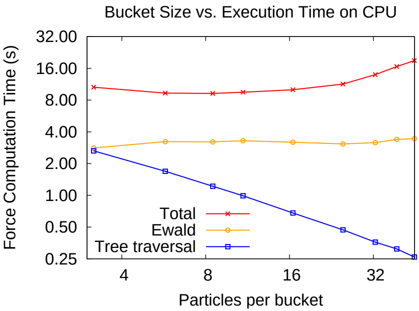
- GPU is hungry for work
  - CPU shouldn't hold back GPU
  - Spend less time traversing tree, more time computing
- Increase average bucket size
  - Tree is shallower: less traversal time on CPU
  - Generates more computation: GPU kept busy

Balancing Tree Traversal and Computation

- GPU is hungry for work
  - CPU shouldn't hold back GPU
  - Spend less time traversing tree, more time computing
- Increase average bucket size
  - Tree is shallower: less traversal time on CPU
  - Generates more computation: GPU kept busy
  - Too much computation work hinders performance
  - Optimal bucket size?

Experimental Results





Overlapping CPU and GPU Work

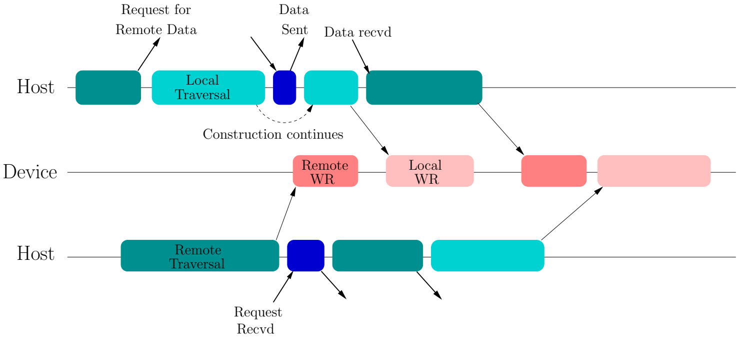


Figure: Traversals construct interaction lists on host. These are sent to the device as Work Requests (WRs) for computation. Overlap is possible between these activities.

Obtaining Optimal Overlap

- More WRs ⇒ more overlap

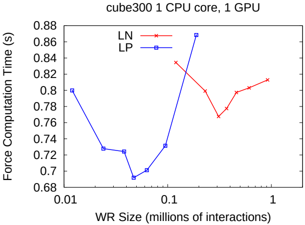
Obtaining Optimal Overlap

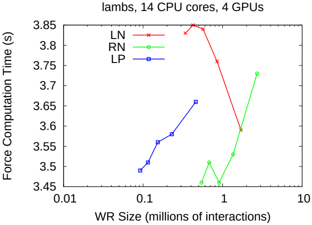
- More WRs ⇒ more overlap
- More WRs ⇒ more offload overhead

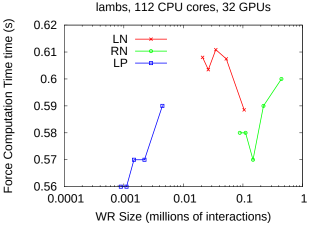
Obtaining Optimal Overlap

- More WRs ⇒ more overlap
- More WRs ⇒ more offload overhead
- Optimal overlap?

Experimental Results







A Lower Bound on Execution Time

- How well can we do?

A Lower Bound on Execution Time

- How well can we do?
- $T_{gpu} = max(T^{l}_{cpu}, T^{f}_{gpu}) + T^{ovhd}_{cpu}$

A Lower Bound on Execution Time

- How well can we do?
- $T_{gpu} = max(T^{l}_{cpu}, T^{f}_{gpu}) + T^{ovhd}_{cpu}$
- Perfect overlap $\Rightarrow max(T^{l}_{cpu}, T^{f}_{gpu}) = T^{l}_{cpu}$

A Lower Bound on Execution Time

- How well can we do?
- $T_{gpu} = max(T^{l}_{cpu}, T^{f}_{gpu}) + T^{ovhd}_{cpu}$
- Perfect overlap $\Rightarrow max(T^{l}_{cpu}, T^{f}_{gpu}) = T^{l}_{cpu}$
- Full efficiency $\Rightarrow T^{ovhd}_{cpu} = 0$

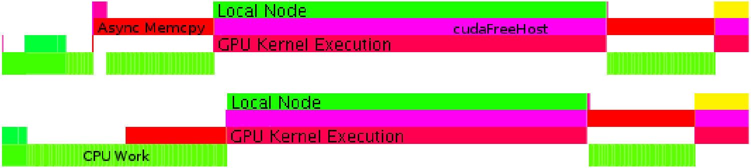
A Lower Bound on Execution Time

- How well can we do?
- $T_{gpu} = max(T^{l}_{cpu}, T^{f}_{gpu}) + T^{ovhd}_{cpu}$
- Perfect overlap $\Rightarrow$ $max(T^{l}_{cpu}, T^{f}_{gpu}) = T^{l}_{cpu}$
- Full efficiency $\Rightarrow$ $T^{ovhd}_{cpu} = 0$
- Therefore, $T^{*}_{gpu} = T^{l}_{cpu}$

A Lower Bound on Execution Time

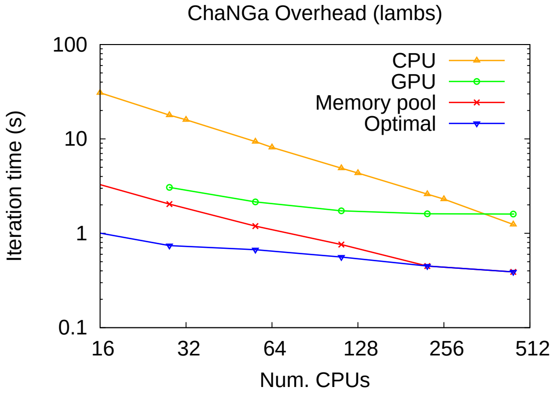
- How well can we do?
- $T_{gpu} = max(T^{l}_{cpu}, T^{f}_{gpu}) + T^{ovhd}_{cpu}$
- Perfect overlap $\Rightarrow max(T^{l}_{cpu}, T^{f}_{gpu}) = T^{l}_{cpu}$
- Full efficiency $\Rightarrow T^{ovhd}_{cpu} = 0$
- Therefore, $T^{*}_{gpu} = T^{l}_{cpu}$
- And, $T^{ovhd}_{cpu} = T_{gpu} - T^{*}_{gpu} = T_{gpu} - T^{l}_{cpu}$

Unexpected Serial Overhead

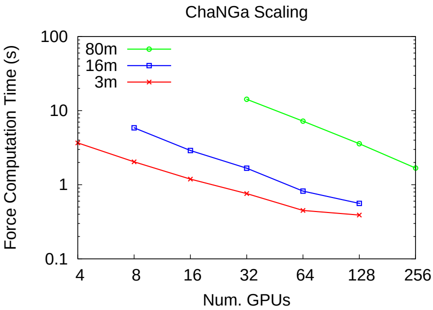


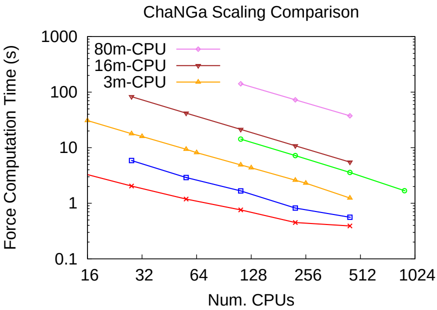
- CUDA memory allocation/free calls block CPU
- Repeated memory pinning costs

Experimental Results



Scaling Performance on *Lincoln*





Comparison of CPU-only and CPU-GPU versions

Procs.	GPUs	3m		16m		80m	
		S_n	*GFLOPS*	S_n	*GFLOPS*	S_n	*GFLOPS*
14	4	9.5	57.17				
28	8	8.75	102.84	14.14	176.43		
56	16	7.87	176.31	14.43	357.11		
112	32	6.45	276.06	12.78	620.14	9.92	450.32
224	64	5.78	466.23	13.21	1262.96	10.07	888.79
448	128	3.18	537.96	9.82	1849.34	10.47	1794.06
896	256					-	3819.69

Table: Speedups and computation rates with various data sets.

Future Work

- Larger data sets, full machine runs
- Multistepped execution performance
- Load balancing issues with highly-clustered data sets
- (Single precision) hexadecapole moments
- Port SPH computation to GPU
- Fast multipole methods
- Pipelined tree traversal on the GPU
- Compare with other heterogeneous systems