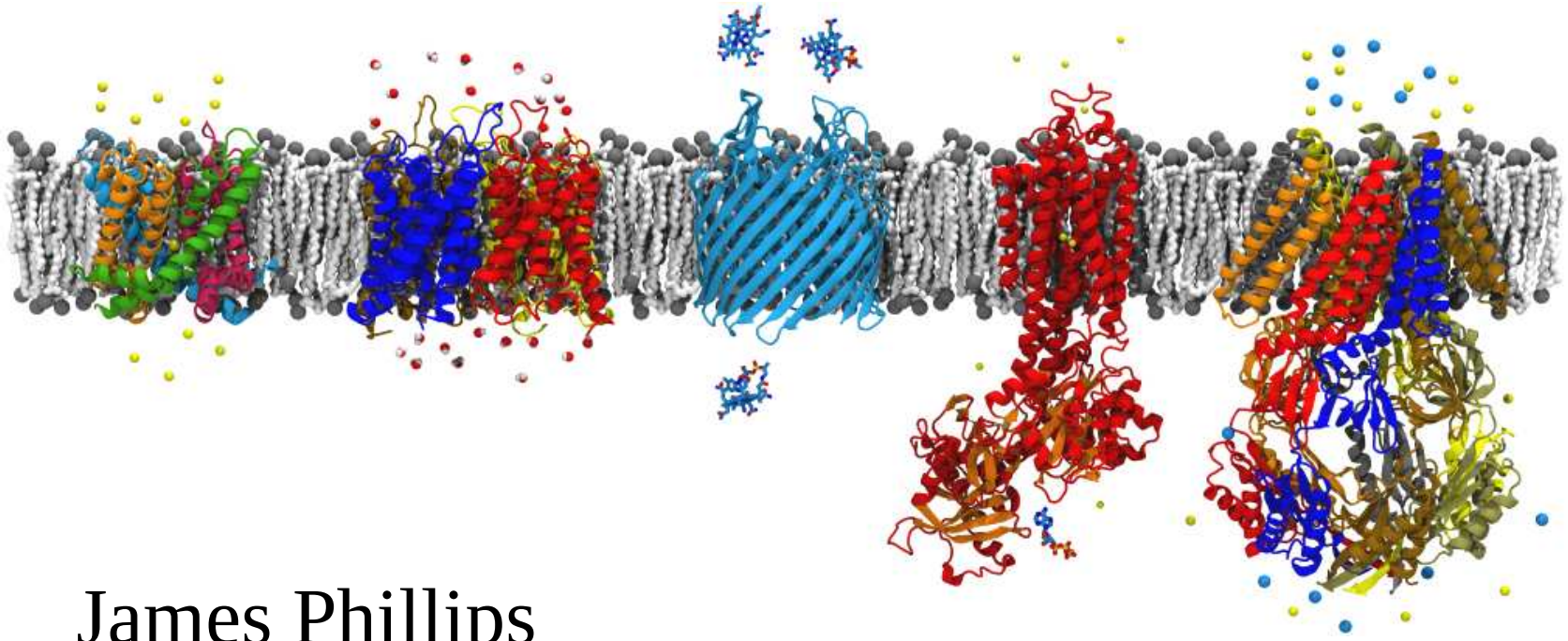
# Refactoring NAMD for Petascale Machines and Graphics Processors



James Phillips

http://www.ks.uiuc.edu/Research/namd/

# NAMD Design

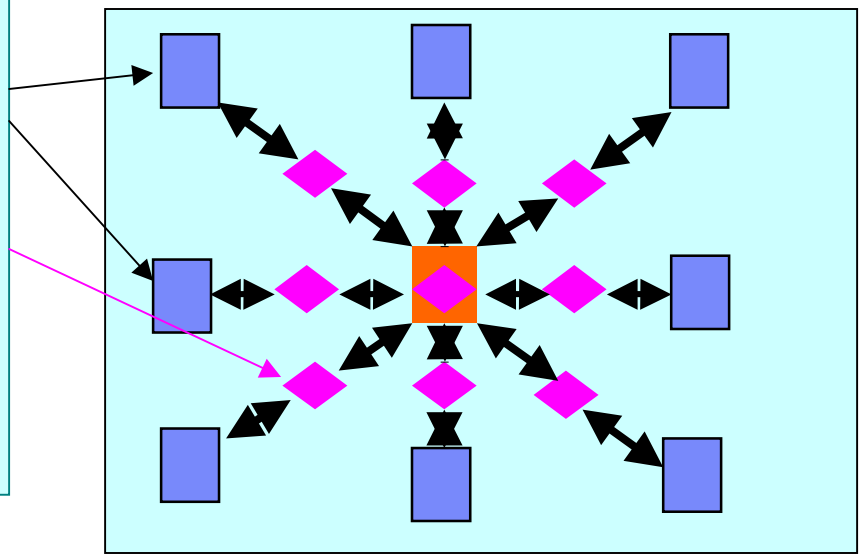
- Designed from the beginning as a parallel program
- Uses the Charm++ idea:
  - Decompose the computation into a large number of objects
  - Have an Intelligent Run-time system (of Charm++) assign objects to processors for dynamic load balancing

Hybrid of spatial and force decomposition:

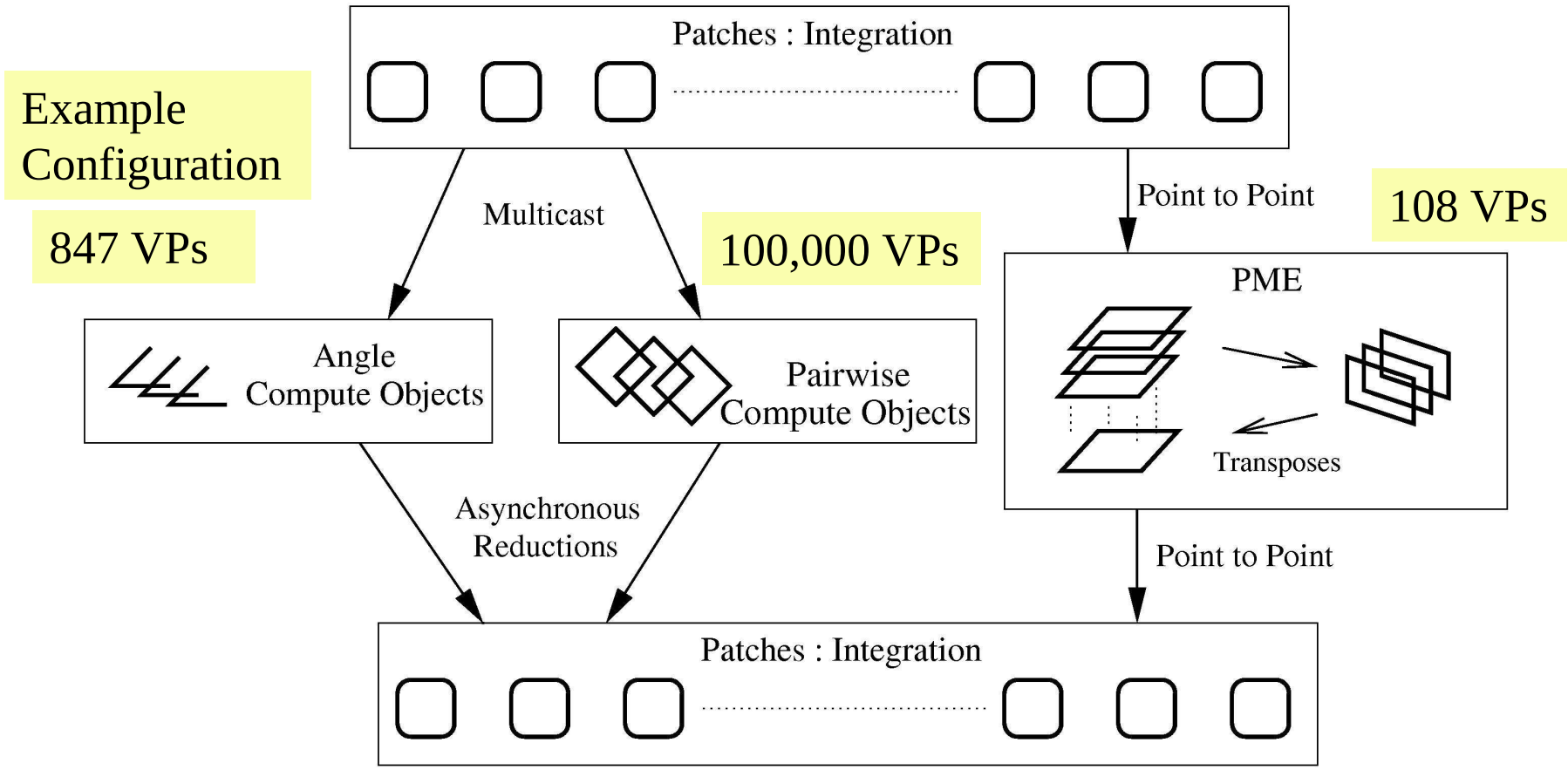
•Spatial decomposition of atoms into cubes (called patches)

•For every pair of interacting patches, create one object for calculating electrostatic interactions

•Recent: Blue Matter, Desmond, etc. use this idea in some form

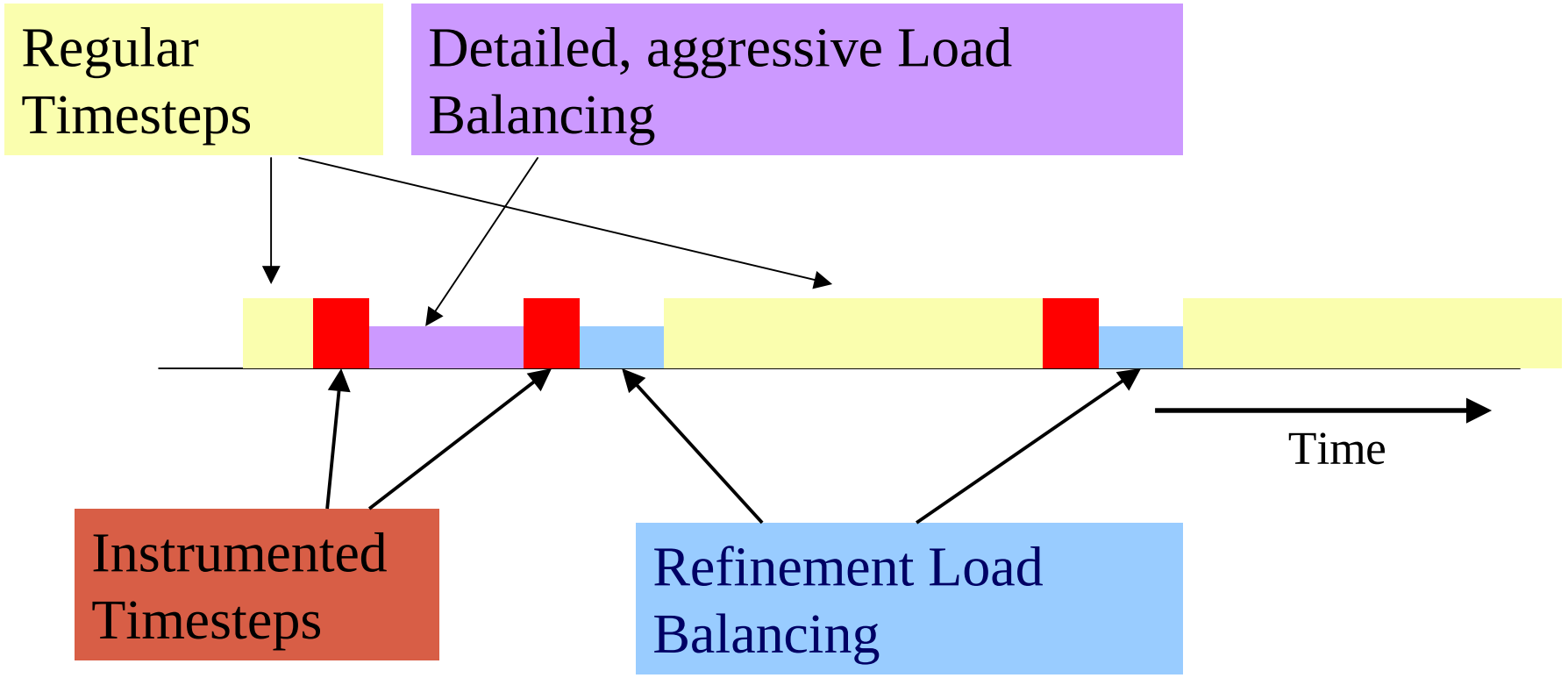


# NAMD Parallelization using Charm++




These 100,000 Objects (virtual processors, or VPs) are assigned to real processors by the Charm++ runtime system

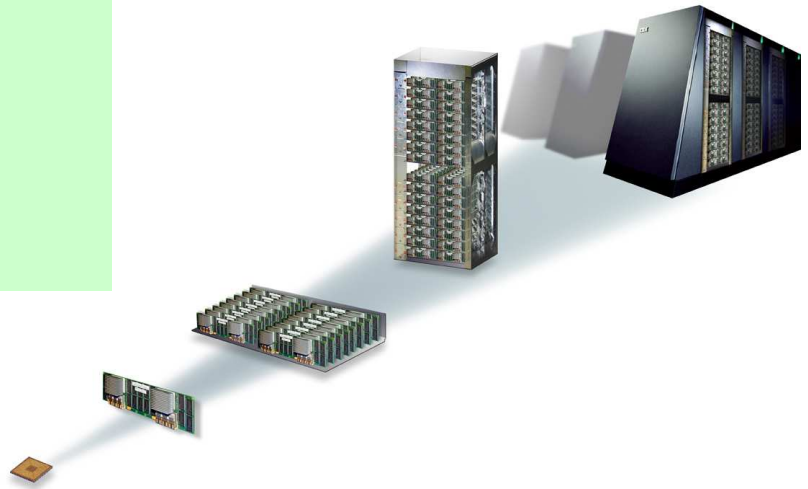
# Load Balancing Steps



# Parallelization on BlueGene/L

- Sequential Optimizations
- Messaging Layer Optimizations
- NAMD parallel tuning
- Illustrates porting effort

| Optimization | Performance |
|---|---|
| NAMD v2.5 | 40 ms |
| NAMD v2.6 Blocking | 25.2 |
| Fine Grained | 24.3 |
| Congestion Control | 20.5 |
| Topology Loadbalancer | 14 |
| Chessboard Dynamic FIFO Mapping | 13.5 |
| Fast Memcpy | 13.3 |
| Non Blocking | 11.9 |
| 2AwayXY + Spanning tree | 8.6 (10 ns/day) |

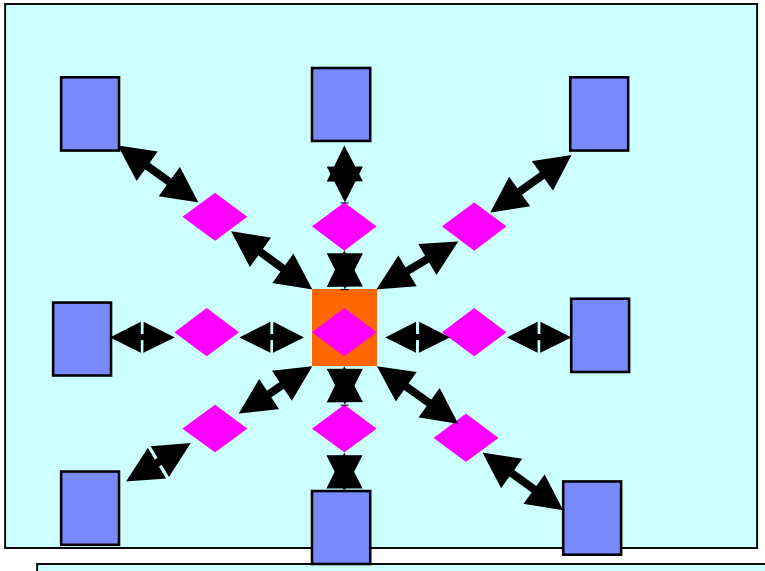


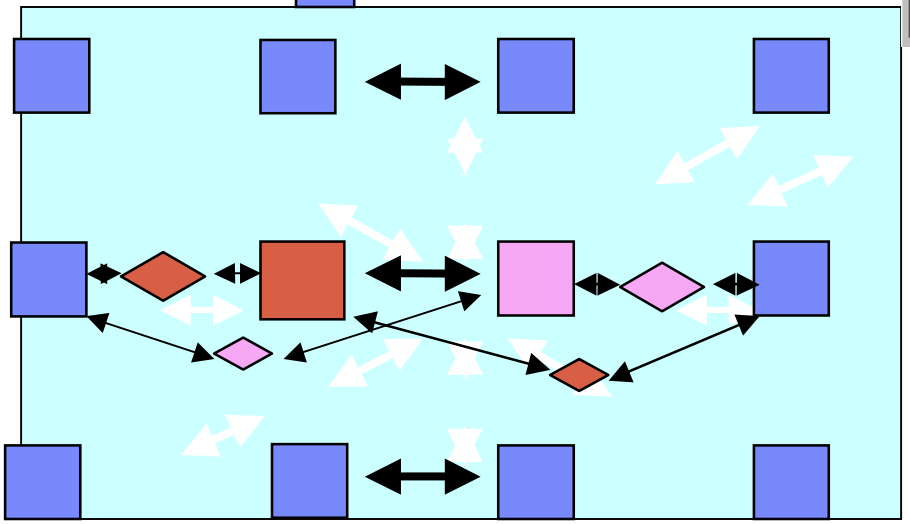
"Inside" help by:

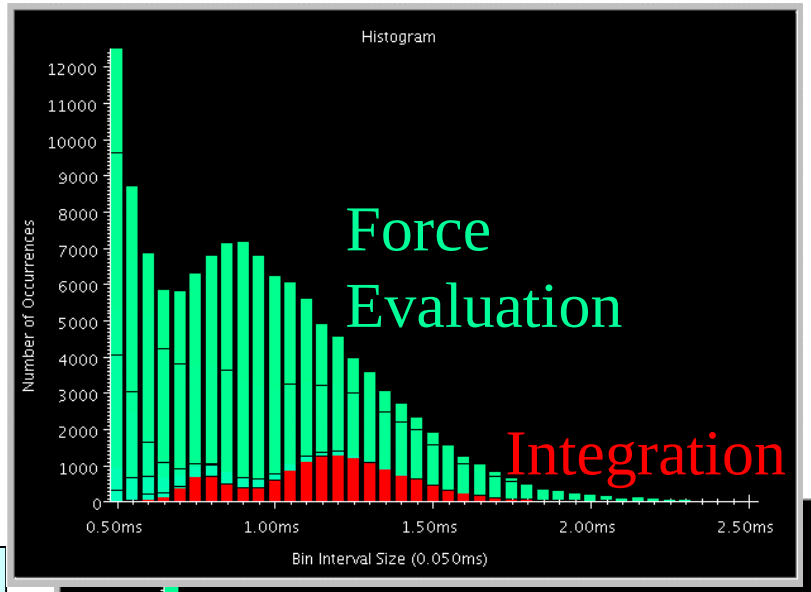
**Sameer Kumar**, former CS/TCB student, now at IBM BlueGene group, tasked by IBM to support NAMD

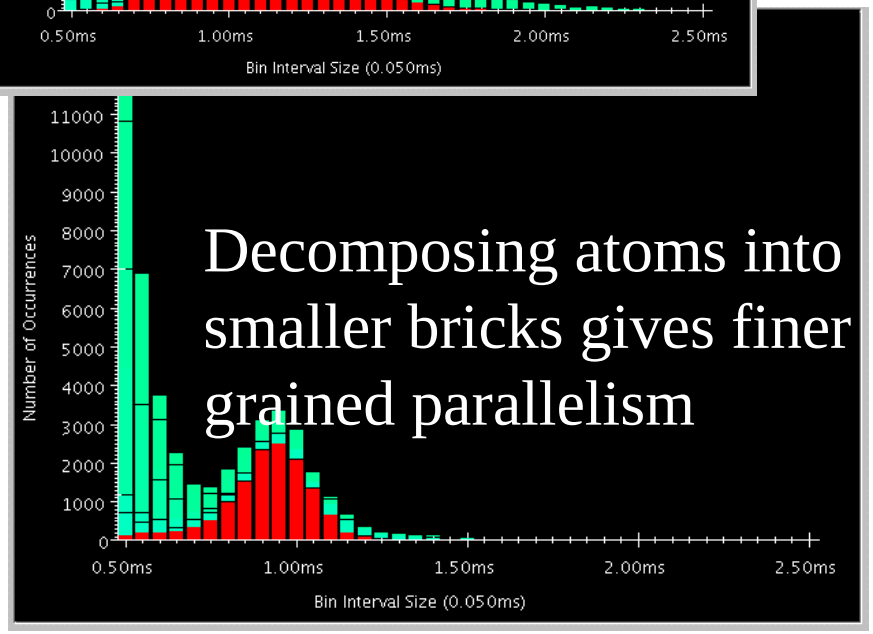
**Chao Huang**, spent summer at IBM on messaging layer

# Fine Grained Decomposition on BlueGene







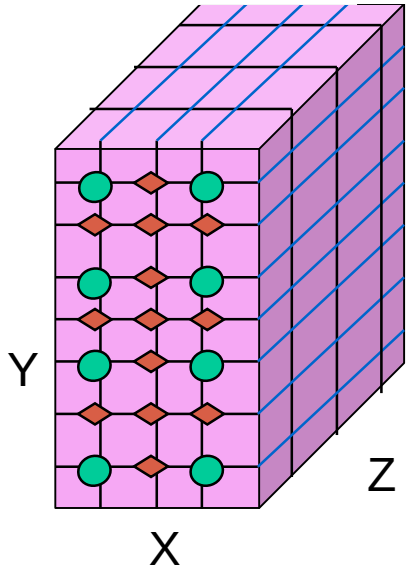



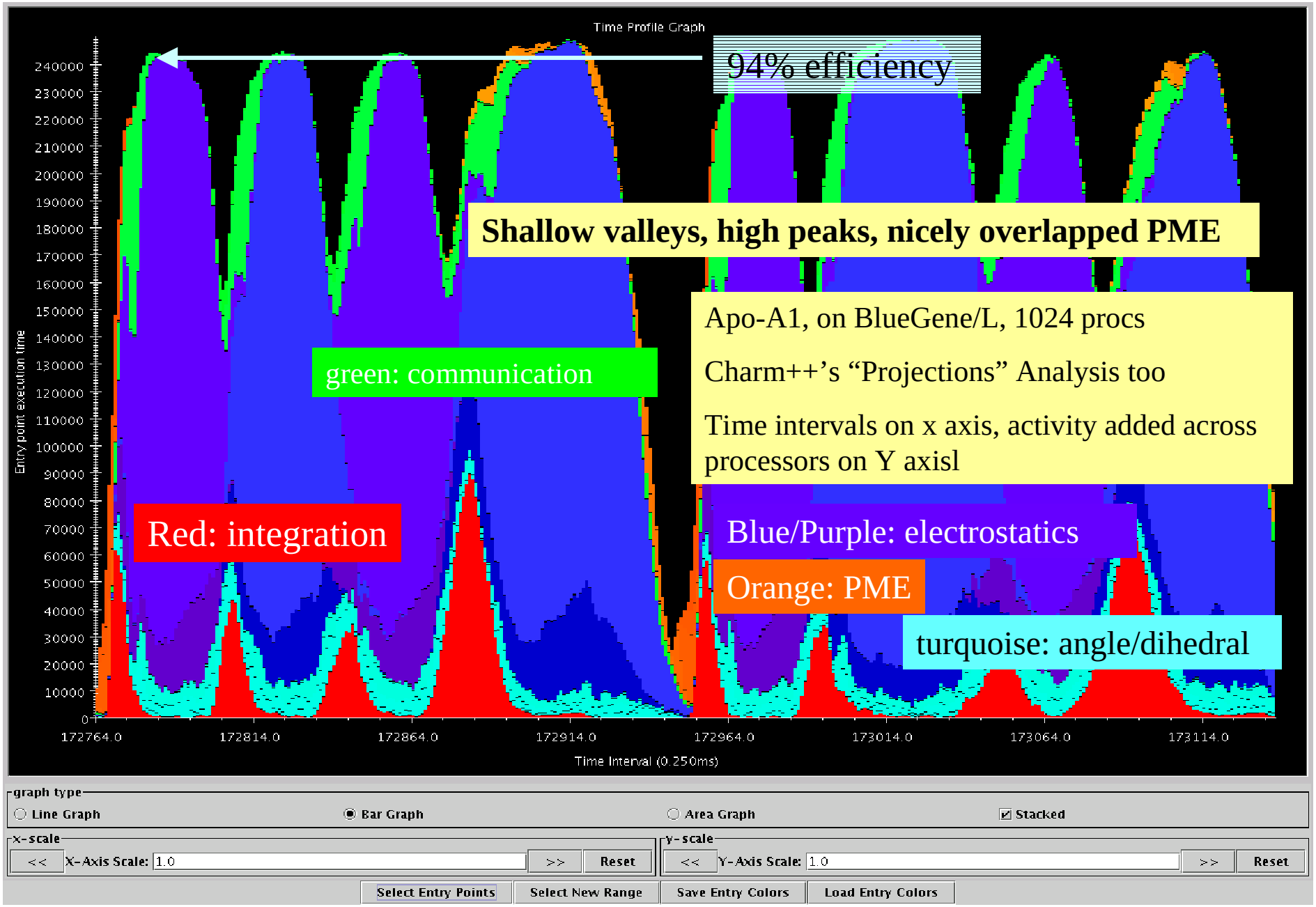
# Recent Large-Scale Parallelization

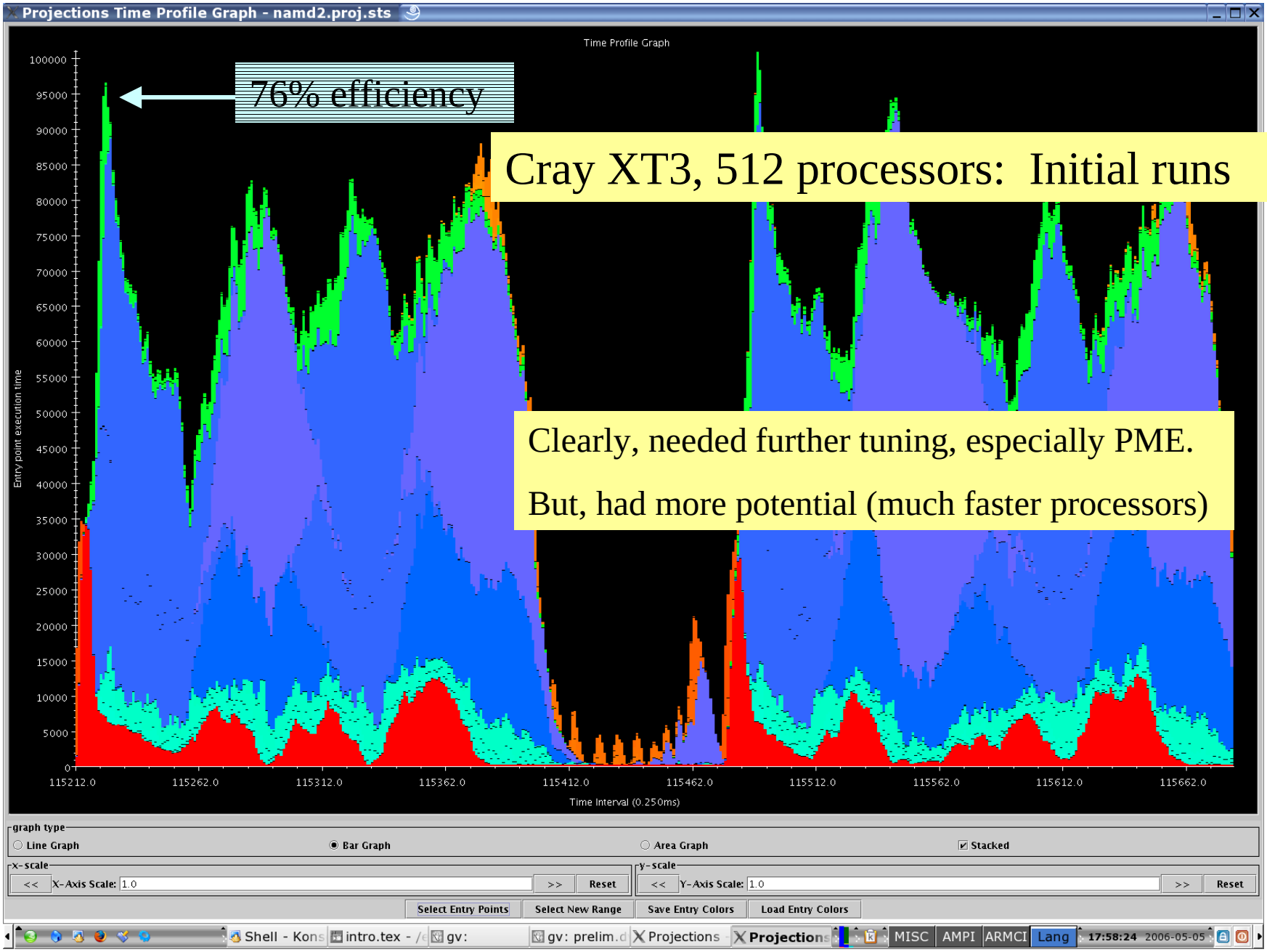
- Since the proposal was submitted
- PME parallelization: needs to be fine grained
  - We recently did a 2-D (Pencil-based) parallelization:
    - will be tuned further
  - Efficient data-exchange between atoms and grid
- Memory issues:
  - New machines will stress memory/node
    - 256MB per processor on BlueGene/L
    - NSF's selection of NAMD, and BAR domain benchmark
  - Plan: partition all static data,
    - Preliminary work done:
    - We can now simulate ribosome on BlueGene/L
    - Much larger systems on Cray XT3:
- Interconnection topology:
  - Is becoming a strong factor: bandwidth
  - Topology-aware load balancers in Charm++, some specialized to NAMD

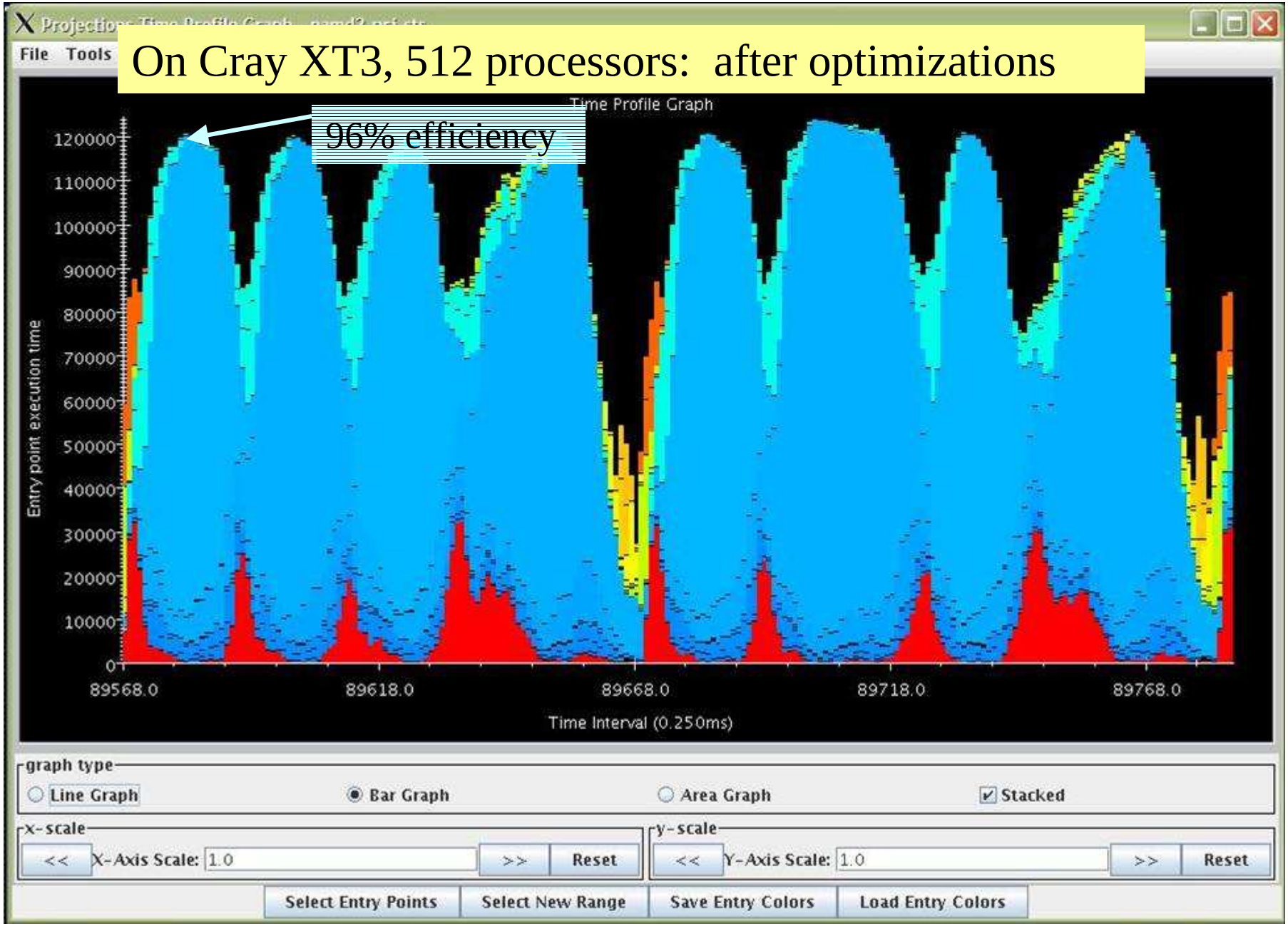
Improvement with pencil: 0.65 ns per day to 1.2 ns/day.

Fibrinogen system: 1 million atoms running on 1024 processors at PSC XT3

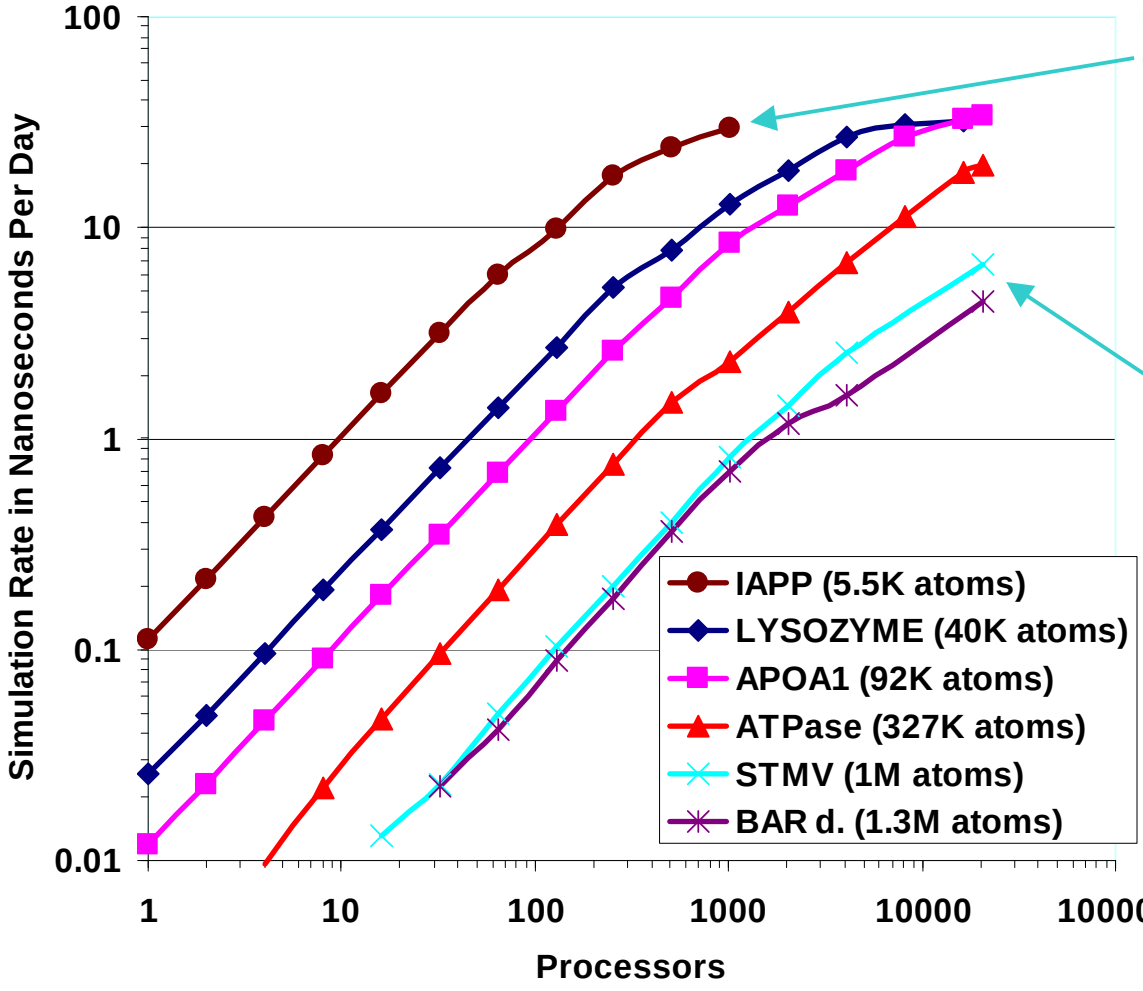


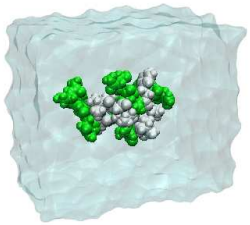

Time Profile Graph
94% efficiency
Shallow valleys, high peaks, nicely overlapped PME
Apo-A1, on BlueGene/L, 1024 procs
Charm++'s "Projections" Analysis too
Time intervals on x axis, activity added across processors on Y axisl
green: communication
Red: integration
Blue/Purple: electrostatics
Orange: PME
turquoise: angle/dihedral
Entry point execution time
Time Interval (0.250ms)
240000
230000
220000
210000
200000
190000
180000
170000
160000
150000
140000
130000
120000
110000
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
172764.0
172814.0
172864.0
172914.0
172964.0
173014.0
173064.0
173114.0
graph type
Line Graph
Bar Graph
Area Graph
Stacked
x-scale
<<
X-Axis Scale: 1.0
>>
Reset
y-scale
<<
Y-Axis Scale: 1.0
>>
Reset
Select Entry Points
Select New Range
Save Entry Colors
Load Entry Colors


Projections Time Profile Graph - namd2.proj.sts
Time Profile Graph
76% efficiency
Cray XT3, 512 processors: Initial runs
Clearly, needed further tuning, especially PME.
But, had more potential (much faster processors)
Entry point execution time
Time Interval (0.250ms)
graph type
Line Graph
Bar Graph
Area Graph
Stacked
x-scale
y-scale
X-Axis Scale: 1.0
Y-Axis Scale: 1.0
Reset
Select Entry Points
Select New Range
Save Entry Colors
Load Entry Colors
MISC
AMPI
ARMCI
Lang
17:58:24
2006-05-05

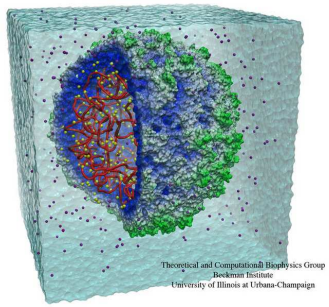

On Cray XT3, 512 processors: after optimizations
96% efficiency
Time Profile Graph
Entry point execution time
120000
110000
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
89568.0
89618.0
89668.0
89718.0
89768.0
Time Interval (0.250ms)
File
Tools
graph type
Line Graph
Bar Graph
Area Graph
Stacked
x-scale
<<
X-Axis Scale: 1.0
>>
Reset
y-scale
<<
Y-Axis Scale: 1.0
>>
Reset
Select Entry Points
Select New Range
Save Entry Colors
Load Entry Colors

# Performance on BlueGene/L



IAPP simulation
(Rivera, Straub, BU)
at 20 ns per day
on 256 processors
**1 us in 50 days**




STMV simulation
at 6.65 ns per day
on 20,000 processors

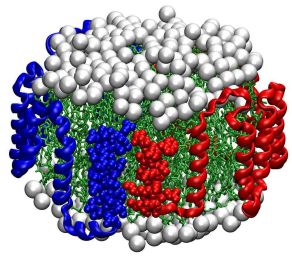
# Comparison with Blue Matter

ApoLipoprotein-A1 (92K atoms)

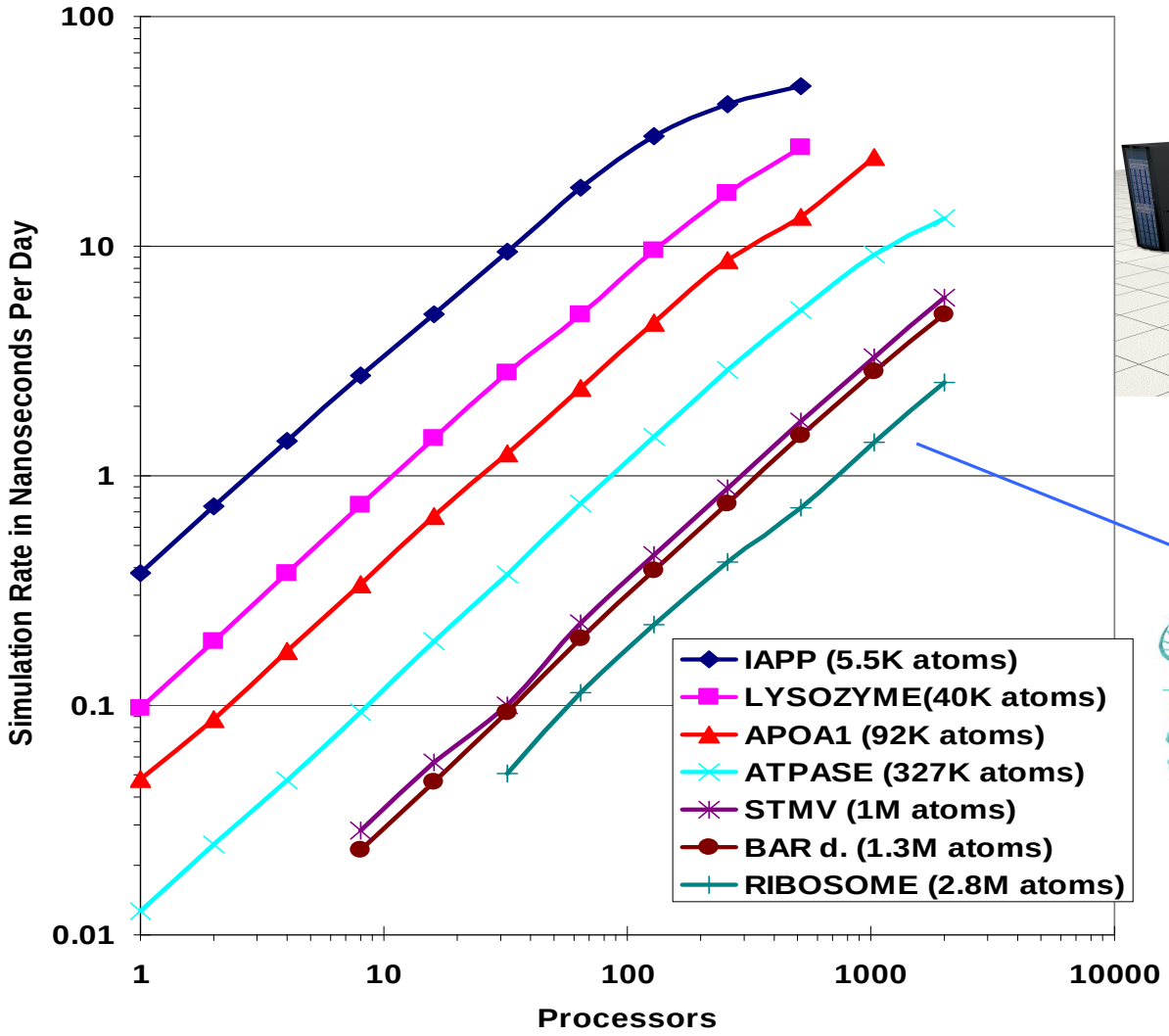
| Nodes | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | |
|---|---|---|---|---|---|---|---|
| Blue Matter (SC'06) | 38.42 | 18.95 | 9.97 | 5.39 | 3.14 | 2.09 | ms/step |
| NAMD | 18.6 | 10.5 | 6.85 | 4.67 | 3.2 | 2.33 | ms/step |
| NAMD (Virtual Node) | 11.3 | 7.6 | 5.1 | 3.7 | 3.0 | 2.33 (CP) | ms/step |

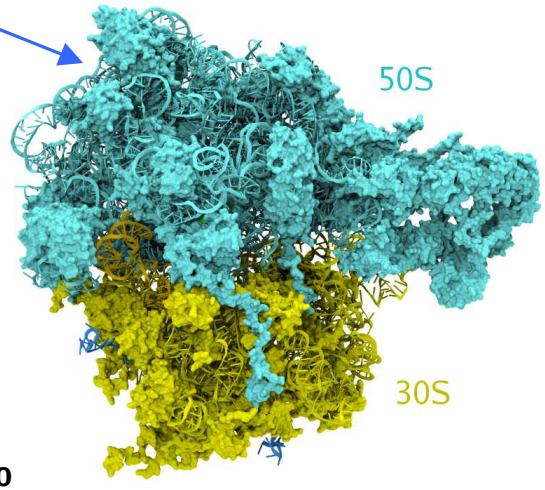
NAMD is about 1.8 times faster than Blue Matter on 1024 processors (and 3.4 times faster with VN mode, where NAMD can use both processors on a node effectively).

However: Note that NAMD does PME every 4 steps.



# Performance on Cray XT3




National Center for Research Resources

# NAMD: Practical Supercomputing

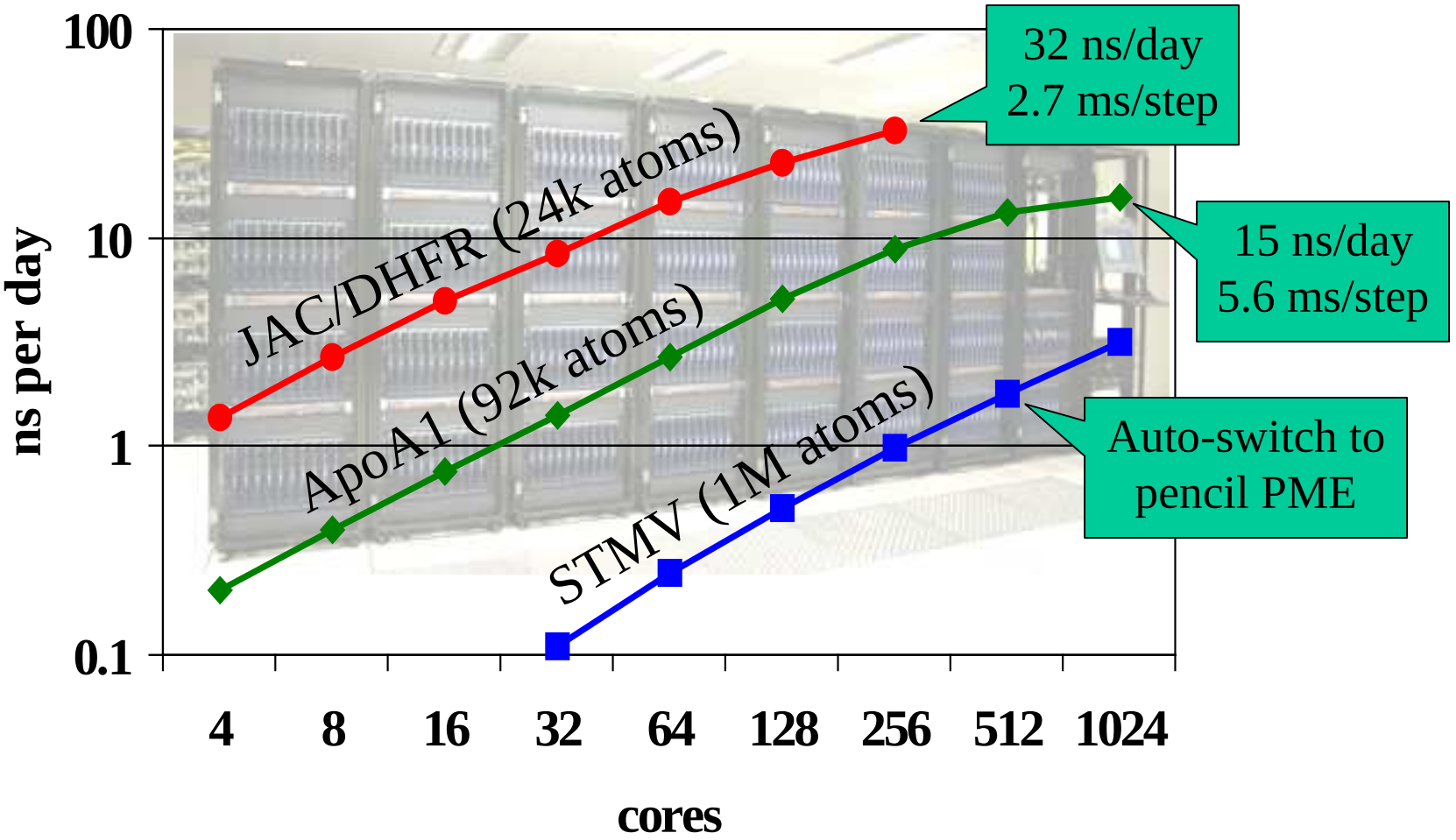
- 20,000 users can't all be computer experts.
  - 18% are NIH-funded; many in other countries.
  - 4200 have downloaded more than one version.

- User experience is the same on all platforms.
  - No change in input, output, or configuration files.
  - Run any simulation on **any number of processors**.
  - Automatically split patches and enable pencil PME.
  - Precompiled binaries available when possible.
- Desktops and laptops – setup and testing
  - x86 and x86-64 Windows, PowerPC and x86 Macintosh
  - Allow both shared-memory and network-based parallelism.

- Linux clusters – affordable workhorses
  - x86, x86-64, and Itanium processors
  - Gigabit ethernet, Myrinet, InfiniBand, Quadrics, Altix, etc

# NAMD Shines on InfiniBand

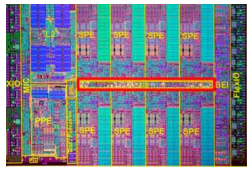
*TACC Lonestar is based on Dell servers and InfiniBand.*
*Commodity cluster with 5200 cores!  (Everything's bigger in Texas.)*



# Hardware Acceleration for NAMD

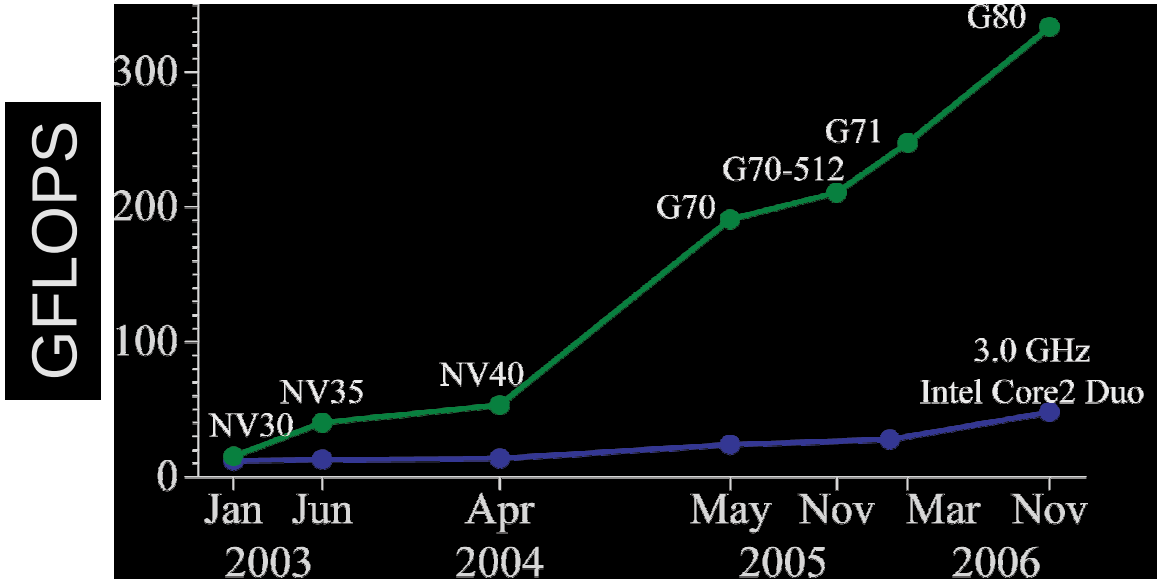
*Can NAMD offload work to a special-purpose processor?*

- Resource studied all the options in 2005-2006:
  - FPGA reconfigurable computing (with NCSA)
    - Difficult to program, slow floating point, expensive
  - Cell processor (NCSA hardware)
    - Relatively easy to program, expensive
  - ClearSpeed (direct contact with company)
    - Limited memory and memory bandwidth, expensive
  - MDGRAPE
    - Inflexible and expensive
  - Graphics processor (GPU)
    - Program must be expressed as graphics operations



# GPU Performance Far Exceeds CPU

- A quiet revolution – in games world so far
  - Calculation: 450 GFLOPS vs. 32 GFLOPS
  - Memory Bandwidth: 80 GB/s vs. 8.4 GB/s




G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800

# CUDA: Practical Performance

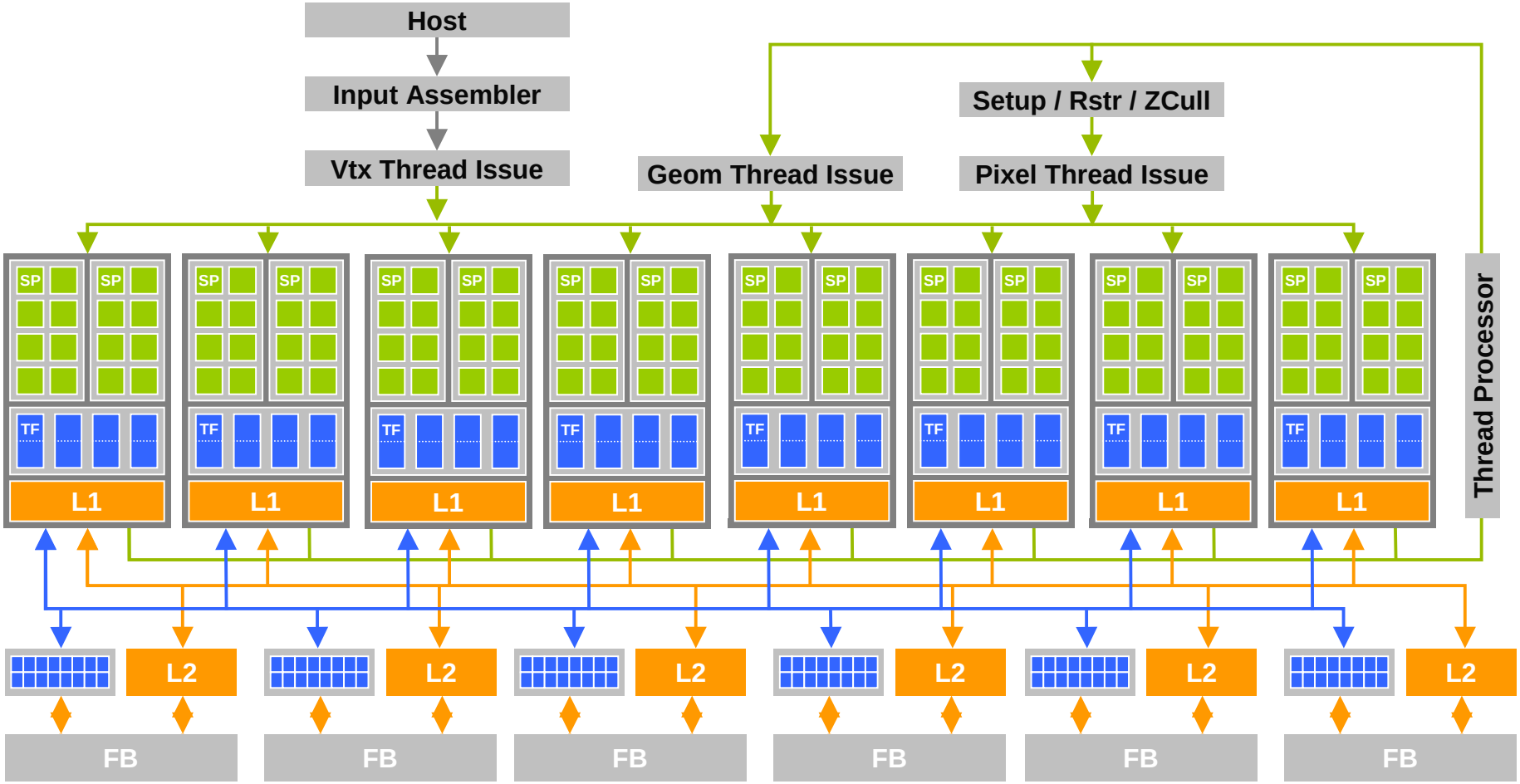
*November 2006: NVIDIA announces CUDA for G80 GPU.*

- CUDA makes GPU acceleration usable:
  - Developed and supported by NVIDIA.
  - No masquerading as graphics rendering.
  - New shared memory and synchronization.
  - No OpenGL or display device hassles.
  - Multiple processes per card (or vice versa).
- Resource and collaborators make it useful:
  - Experience from VMD development
  - David Kirk (Chief Scientist, NVIDIA)
  - Wen-mei Hwu (ECE Professor, UIUC)


Fun to program (and drive)

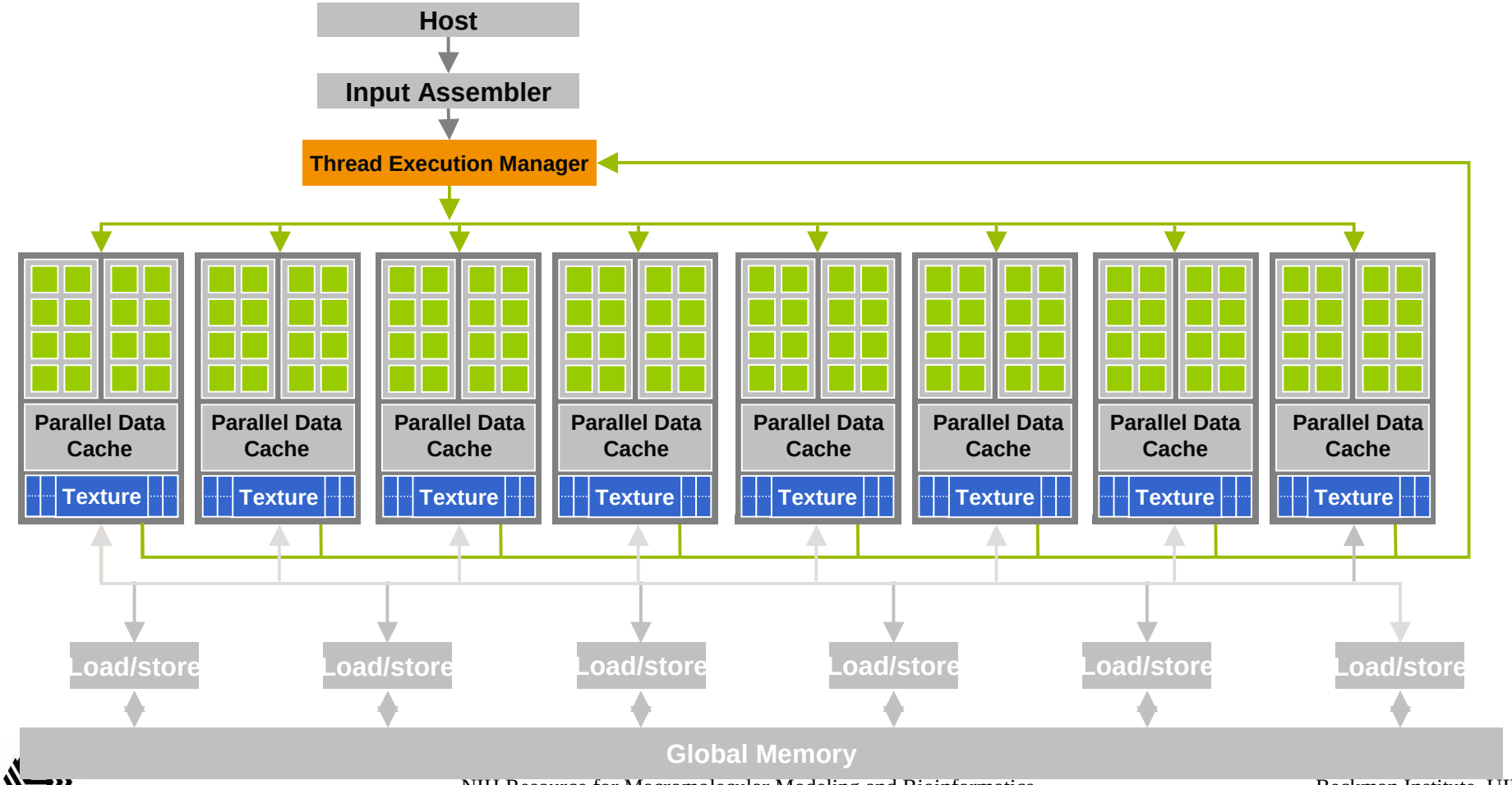
# GeForce 8800 Graphics Mode

- New GPUs are built around threaded cores

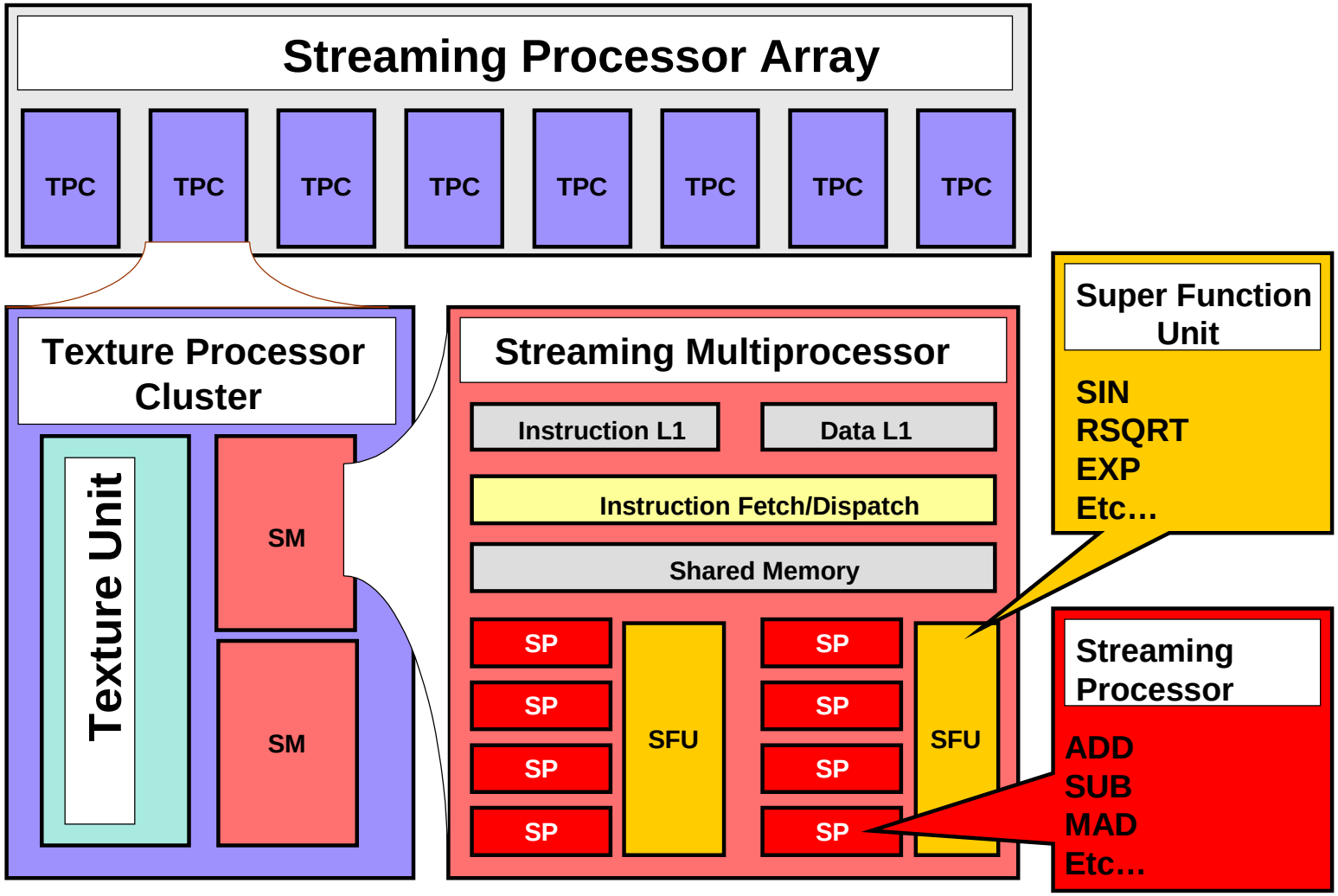


# GeForce80 General Computing

## Up to 65,535 threads, 128 cores, 450 GFLOPS, 768 MB DRAM, 4GB/S BW to CPU

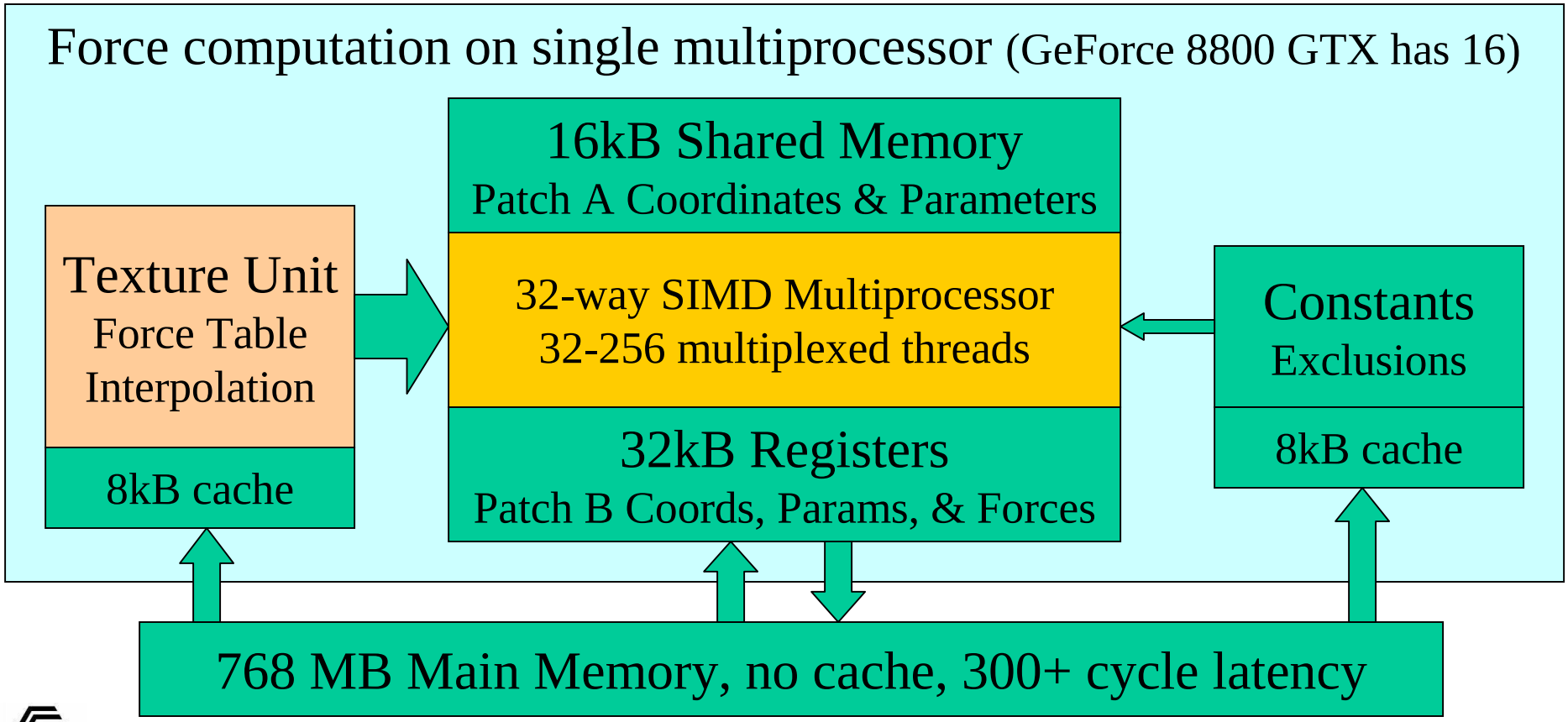


# NVIDIA G80 GPU Hardware



# Nonbonded Forces on CUDA GPU

- Start with most expensive calculation: direct nonbonded interactions.
- Decompose work into pairs of patches, identical to NAMD structure.
- GPU hardware assigns patch-pairs to multiprocessors dynamically.

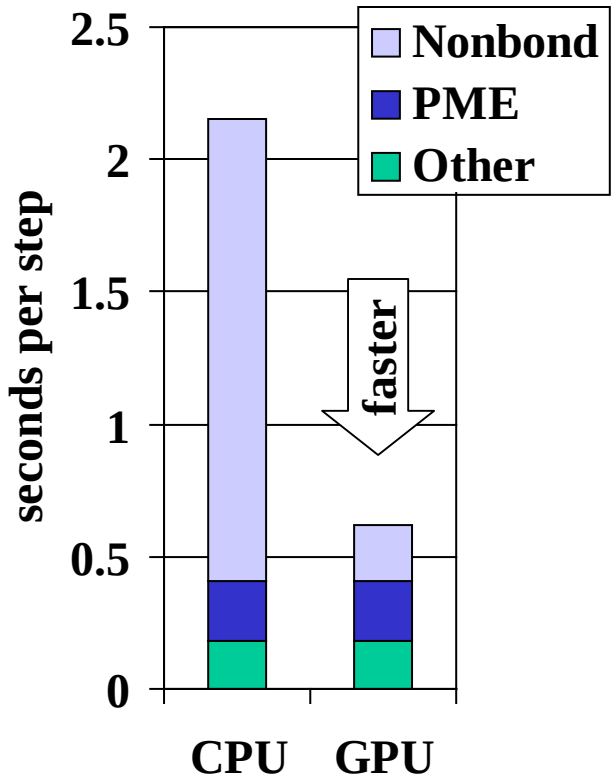


```
texture<float4> force_table;
__constant__ unsigned int exclusions[];
__shared__ atom jatom[];
atom iatom;      // per-thread atom, stored in registers
float4 iforce;   // per-thread force, stored in registers
for ( int j = 0; j < jatom_count; ++j ) {
  float dx = jatom[j].x - iatom.x;   float dy = jatom[j].y - iatom.y;  float dz = jatom[j].z - iatom.z;
  float r2 = dx*dx + dy*dy + dz*dz;
  if ( r2 < cutoff2 ) {
    float4 ft = texfetch(force_table, 1.f/sqrt(r2));
    bool excluded = false;
    int indexdiff = iatom.index - jatom[j].index;
    if ( abs(indexdiff) <= (int) jatom[j].excl_maxdiff ) {
      indexdiff += jatom[j].excl_index;
      excluded = ((exclusions[indexdiff>>5] & (1<<(indexdiff&31))) != 0);
    }
    float f = iatom.half_sigma + jatom[j].half_sigma;  // sigma
    f *= f*f;  // sigma^3
    f *= f;  // sigma^6
    f *= ( f * ft.x + ft.y );  // sigma^12 * fi.x - sigma^6 * fi.y
    f *= iatom.sqrt_epsilon * jatom[j].sqrt_epsilon;
    float qq = iatom.charge * jatom[j].charge;
    if ( excluded ) { f = qq * ft.w; }  // PME correction
    else { f += qq * ft.z; }  // Coulomb
    iforce.x += dx * f;   iforce.y += dy * f;    iforce.z += dz * f;
    iforce.w += 1.f;  // interaction count or energy
  }
}
```

# Initial GPU Performance

- Full NAMD, not test harness
- Useful performance boost
  - 8x speedup for nonbonded
  - 5x speedup overall w/o PME
  - 3.5x speedup overall w/ PME
  - GPU = quad-core CPU
- Plans for better performance
  - Overlap GPU and CPU work.
  - Tune or port remaining work.
    - PME, bonded, integration, etc.

ApoA1 Performance

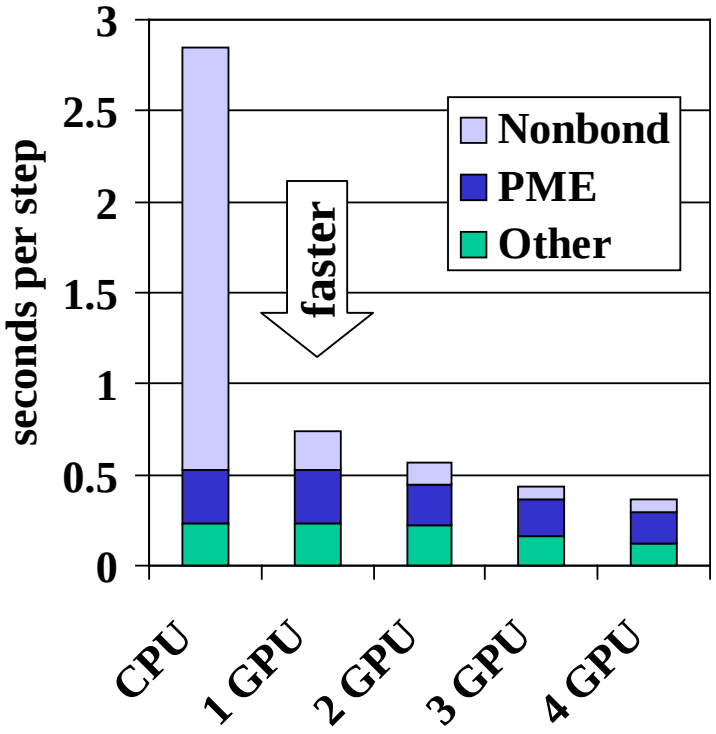



2.67 GHz Core 2 Quad Extreme + GeForce 8800 GTX

# Initial GPU Cluster Performance

- Poor scaling unsurprising
  - 2x speedup on 4 GPUs
  - Gigabit ethernet
  - Load balancer disabled
- Plans for better scaling
  - InfiniBand network
  - Tune parallel overhead
  - Load balancer changes
    - Balance GPU load.
    - Minimize communication.

ApoA1 Performance

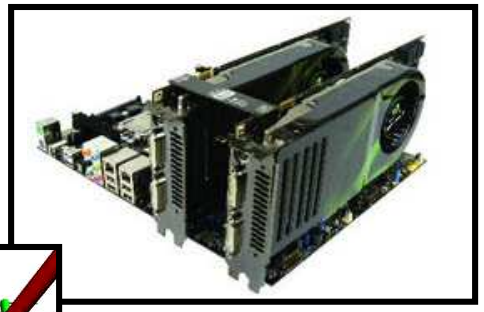



2.2 GHz Opteron + GeForce 8800 GTX

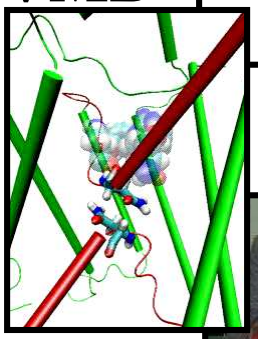
# Next Goal: Interactive MD on GPU

- Definite need for faster serial IMD
  - Useful method for tweaking structures.
  - 10x performance yields 100x sensitivity.
  - Needed on-demand clusters are rare.
- AutoIMD available in VMD already
  - Isolates a small subsystem.
  - Specify molten and fixed atoms.
  - Fixed atoms reduce GPU work.
  - Pairlist-based algorithms start to win.
- Limited variety of simulations
  - Few users have multiple GPUs.
  - Move entire MD algorithm to GPU.

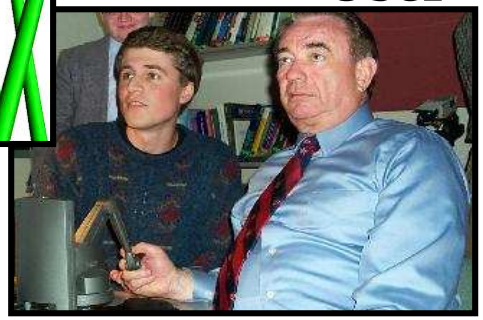
**NAMD**



**VMD**



**User**



(Former HHS Secretary Thompson)