Introduction to Parallel Computing Issues

Laxmikant Kale

http://charm.cs.uiuc.edu

Parallel Programming Laboratory

Dept. of Computer Science

And Theoretical Biophysics Group

Beckman Institute

University of Illinois at Urbana Champaign

Outline

- Parallel Computing:
  - Challenges and Opportunities
  - Survey of CPU speeds trends
  - Trends: parallel machines
  - Trends: Clusters
- Challenges:
  - Communication costs
  - Memory Performance
  - Complex algorithms
  - Parallel Performance issues
  - Virtualization
  - Principle of persistence,
  - Measurementt load balancing
- Case Studies:
- NAMD parallelization
  - Scalability
  - Analysis of other approaches
  - NAMD approach:
    - Hybrid decomposition
    - Virtual processors
  - Performance Optimizations
- Car-Parinello ab Initio MD
  - Algorithm
  - Parallelization strategy
  - Initial results

Overview and Objective

- What is parallel computing
- What opportunities and challenges are presented by parallel computing technology
- Focus on basic understanding of issues in parallel computing

CPU speeds continue to increase

- Current speeds: 3 Ghz on PCs
  - I.e. 330 picosecond for each cycle
  - 2 floating point operations each cycle
    - On some processors, it is 4 per cycle
- Implications
  - We can do a lot more computation in a reasonable time period
  - Do we say “that’s enough”? No!
  - It just brings new possibilities within feasibility horizon

Parallel Computing Opportunities

- Parallel Machines now
  - With thousands of powerful processors, at national centers
    - ASCI White, PSC Lemieux
  - Power: 100GF – 5 TF (5×10^{12}) Floating Points Ops/Sec
- Japanese Earth Simulator
  - 30-40 TF!
- Future machines on the anvil
  - IBM Blue Gene / L
  - 128,000 processors!
  - Will be ready in 2004
- Petaflops around the corner

Clusters Everywhere

- Clusters with 32-256 processors commonplace in research labs
- Attraction of clusters
  - Inexpensive
  - Latest processors
  - Easy to put them together: session this afternoon
- Desktops in a cluster
  - “Use wasted CPU power”

Parallel Applications Opportunities

- Unprecedented opportunities for breakthroughs
  - Rational Drug Design
  - Molecular machines and nanotechnology
  - Optimized engineering designs based on simulations
    - Rockets
    - Materials
    - Industrial Processes: Quenching, Dendritic Growth..
  - Understanding of the Universe
    - Computational Cosmology
  - Operations Research
  - Data Mining
  - Artificial Intelligence?

Parallel Computing Challenges

- It is not easy to develop an efficient parallel program
- Some Challenges:
  - Parallel Programming
  - Complex Algorithms
  - Memory issues
  - Communication Costs
  - Load Balancing

Why Don't Applications Scale?

- Algorithmic overhead
  - Some things just take more effort to do in parallel
    - Example: Parallel Prefix (Scan)
- Speculative Loss
  - Do A and B in parallel, but B is ultimately not needed
- Load Imbalance
  - Makes all processor wait for the "slowest" one
  - Dynamic behavior
- Communication overhead
  - Spending increasing proportion of time on communication
- Critical Paths:
  - Dependencies between computations spread across processors
- Bottlenecks:
  - One processor holds things up

Complex Algorithms

- Consider the problem of computing Electrostatic forces on a set of N atoms due to each other
  - Straightforward algorithm:
    - Calculate force for each pair of atoms
    - $O(N^2)$ computations
    - Relatively easy to parallelize
  - Modern algorithms:
    - Particle-Mesh Ewald (PME) or Fast Multipole
    - O(N. log N) operations
    - But much complex parallelization
  - Multiple time-stepping
  - QM/MM

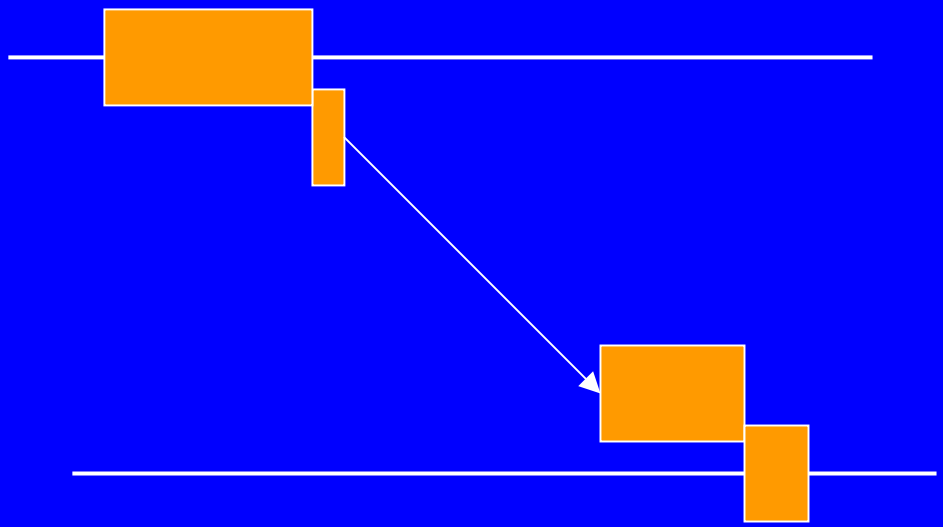
Memory Performance

- DRAM Memory
  - (the kind you buy from Best Buy) is dense, cheap
  - 30-50 nanoseconds to get data from memory into CPU
- CPU can do 2-4 operations every 300 picoseconds
  - 100+ times slower than CPU!
- Solution:
  - Memory Hierarchy:
  - Use small amount fast but expensive memory (Cache)
  - If data fits in Cache, you get Gigaflops performance per proc.
  - Otherwise, can be 10-50 times slower!

Communication Costs

- Normal Ethernet is slow compared with CPU
  - Even 100 Mbit ethernet..
- Some basic concepts:
  - How much data can you get across per unit time: *bandwidth*
  - How much time it needs to send just 1 byte : *Latency*
  - What fraction of this time is spent by the processor?
    - *CPU overhead*
- Problem:
  - CPU overhead is too high for ethernet
  - Because of layers of software and Operating System (OS) involvement

Communication Basics: Point-to-point



Sending processor

Sending Co-processor

Network

Receiving co-processor

Receiving processor

Each component has a per-message cost, and per byte cost

Elan-3 cards on alphaservers (TCS):

Of 2.3 μs "put" time

1.0 : proc/PCI

1.0 : elan card

0.2: switch

0.1 Cable

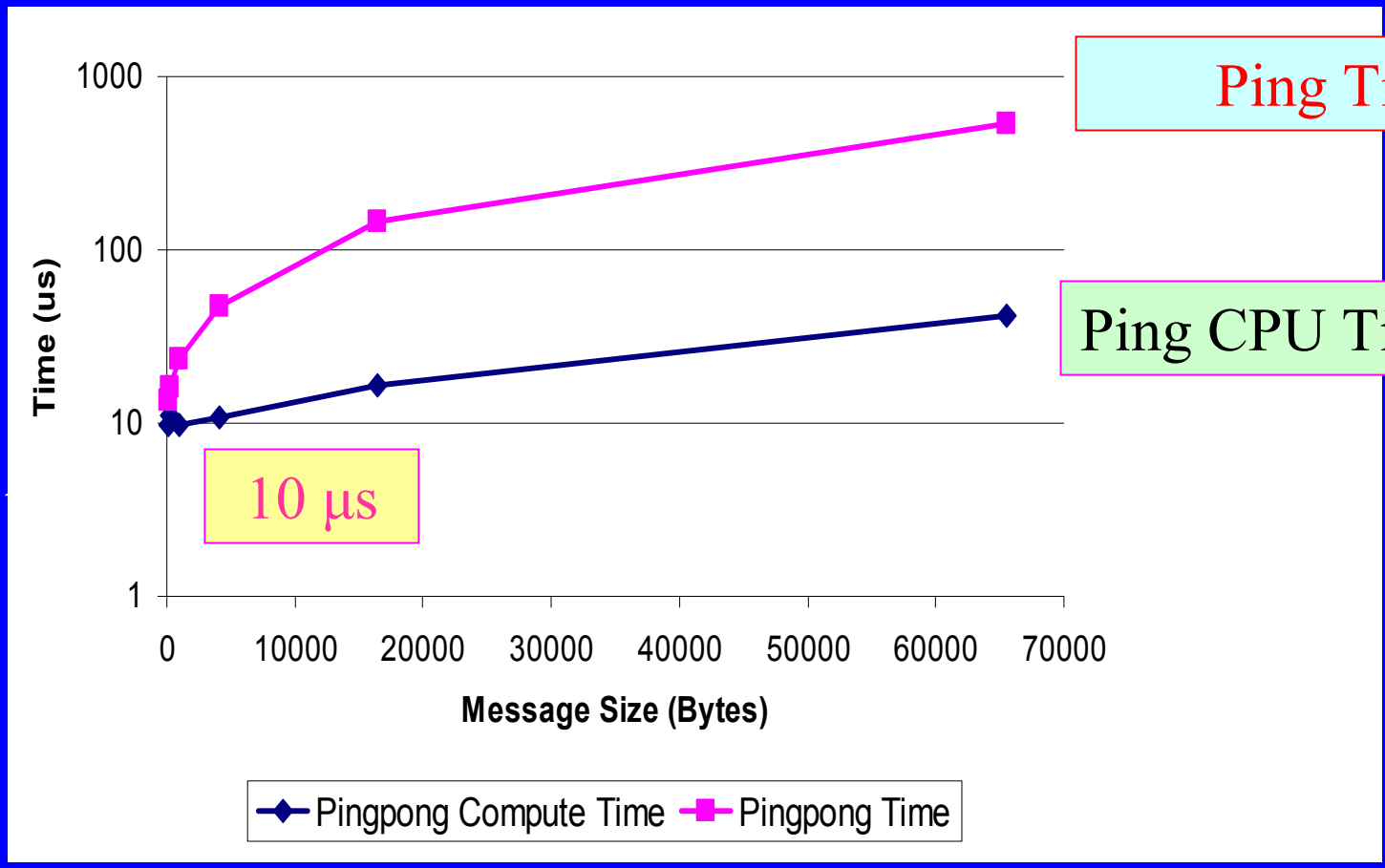
Communication Basics

- Each cost, for an n-byte message
  - $= \alpha + n\,\beta$
- Important metrics:
  - Overhead at Processor, co-processor
  - Network latency
  - Network bandwidth consumed
    - Number of hops traversed
- Elan-3 TCS Quadrics data:
  - MPI send/recv: 4-5 μs
  - Shmem put: 2.5 μs
  - Bandwidth : 325 MB/S (about 3 ns per byte)

Communication network options

	100Mb ethernet	Myrinet	Quadrics
Latency	100 μs	15 μs	3-5 μs
CPU overhead	High	Low	Low
Bandwidth	5-10 MB	100+ MB	325 MB (about 3 ns/byte)
Cost	$	$$$	$$$$

Ping Pong Time on Lemieux

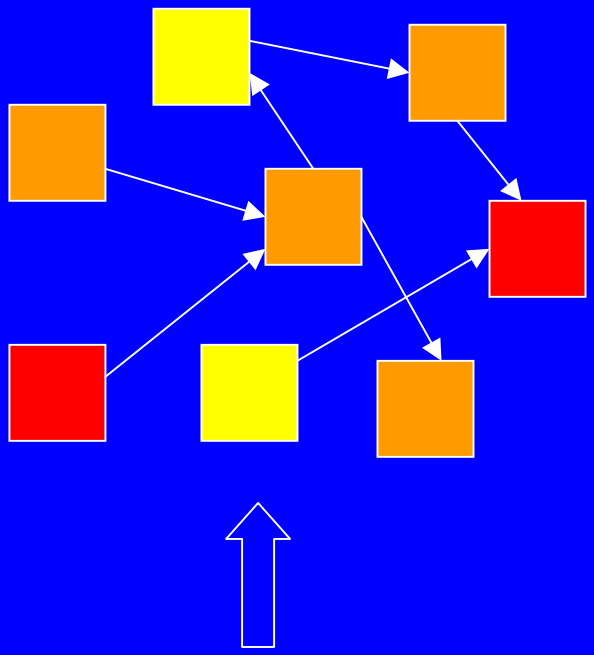


Parallel Programming

- Writing Parallel programs is more difficult than writing sequential programs
  - Coordination
  - Race conditions
  - Performance issues
- Solutions:
  - Automatic parallelization: hasn't worked well
  - MPI: message passing standard in common use
  - Processor virtualization:
    - Programmer decompose the program into parallel parts, but the system assigns them to processors
  - Frameworks: Commonly used parallel patterns are reused

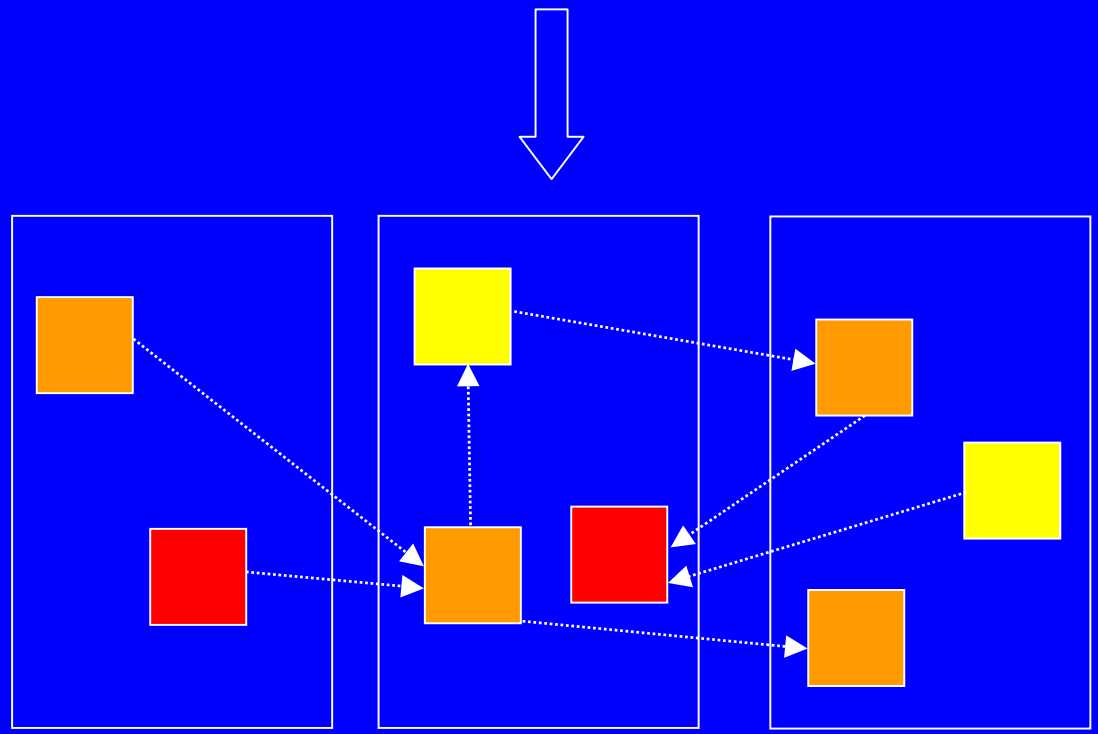
Virtualization: Object-based Parallelization

User is only concerned with interaction between objects

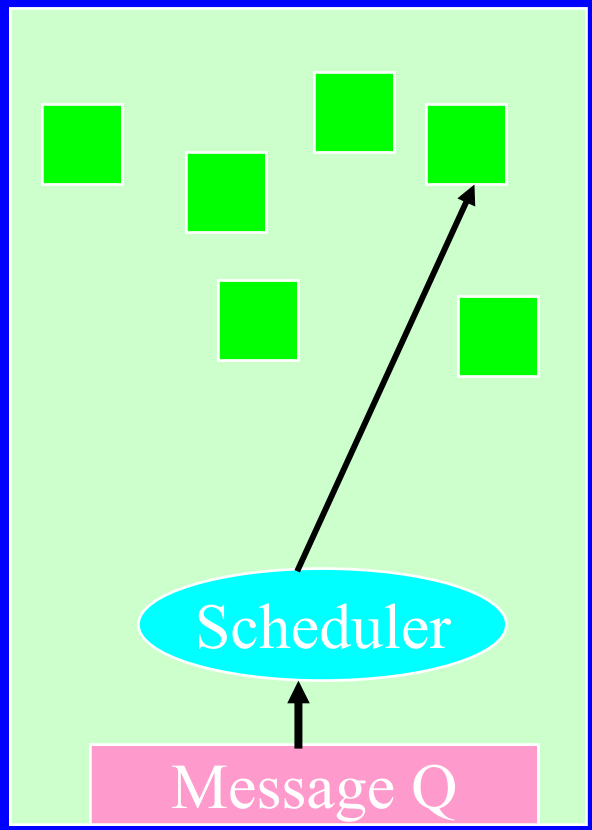


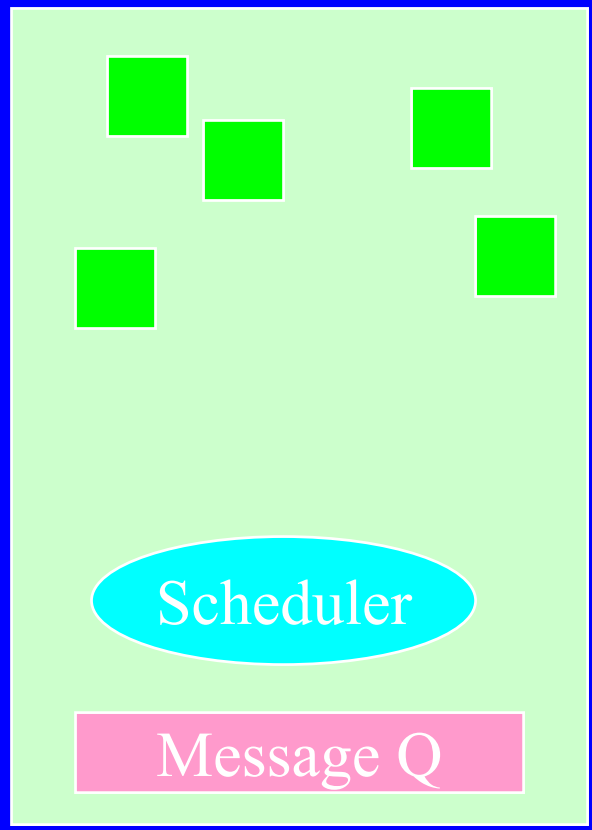
User View

System implementation



Data driven execution





Charm++ and Adaptive MPI Realizations of Virtualization Approach

Charm++

- Parallel C++
  - Asynchronous methods
- In development for over a decade
- Basis of several parallel applications
- Runs on all popular parallel machines and clusters

AMPI

- A migration path for MPI codes
  - Allows them dynamic load balancing capabilities of Charm++
- Minimal modifications to convert existing MPI programs
- Bindings for
  - C, C++, and Fortran90

Both available from http://charm.cs.uiuc.edu

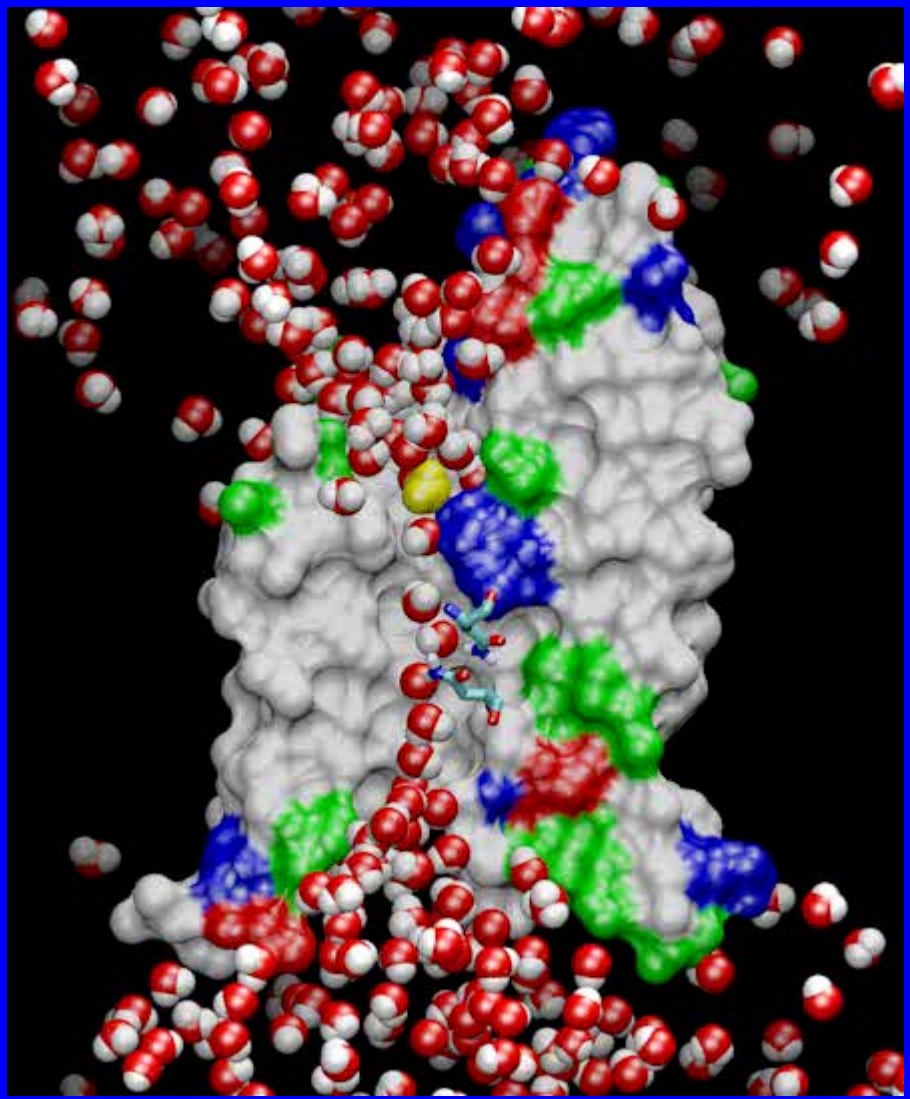
Benefits of Virtualization

- Software Engineering
  - **Number of virtual processors can be independently controlled**
  - Separate VPs for modules
- Message Driven Execution
  - **Adaptive overlap**
  - **Modularity**
  - Predictability:
    - Automatic Out-of-core
- Dynamic mapping
  - Heterogeneous clusters:
    - Vacate, adjust to speed, share
  - Automatic checkpointing
  - Change the set of processors
- Principle of Persistence:
  - Enables Runtime Optimizations
  - **Automatic Dynamic Load Balancing**
  - **Communication Optimizations**
  - Other Runtime Optimizations

More info:

http://charm.cs.uiuc.edu

NAMD: A Production MD program



NAMD

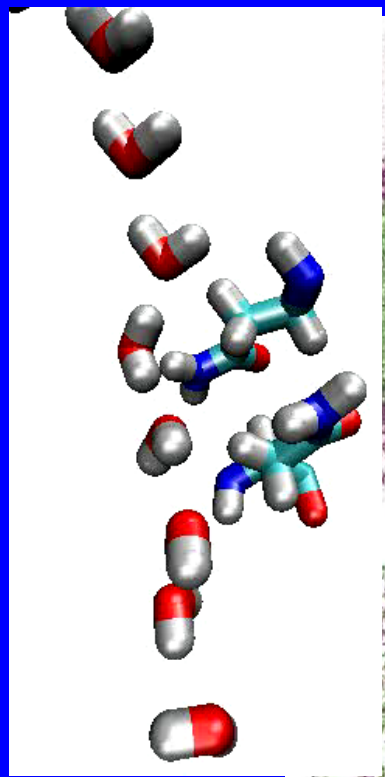
- Fully featured program
- NIH-funded development
- Distributed free of charge (~5000 downloads so far)
- Binaries and source code
- Installed at NSF centers
- User training and support
- Large published simulations (e.g., aquaporin simulation featured in keynote)

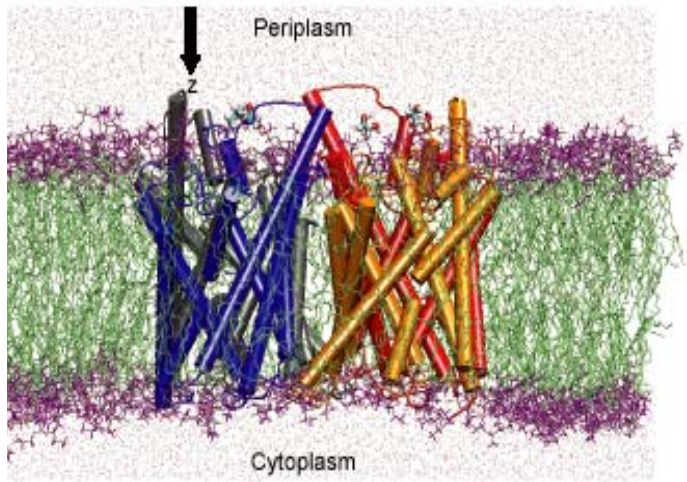
Acquaporin Simulation

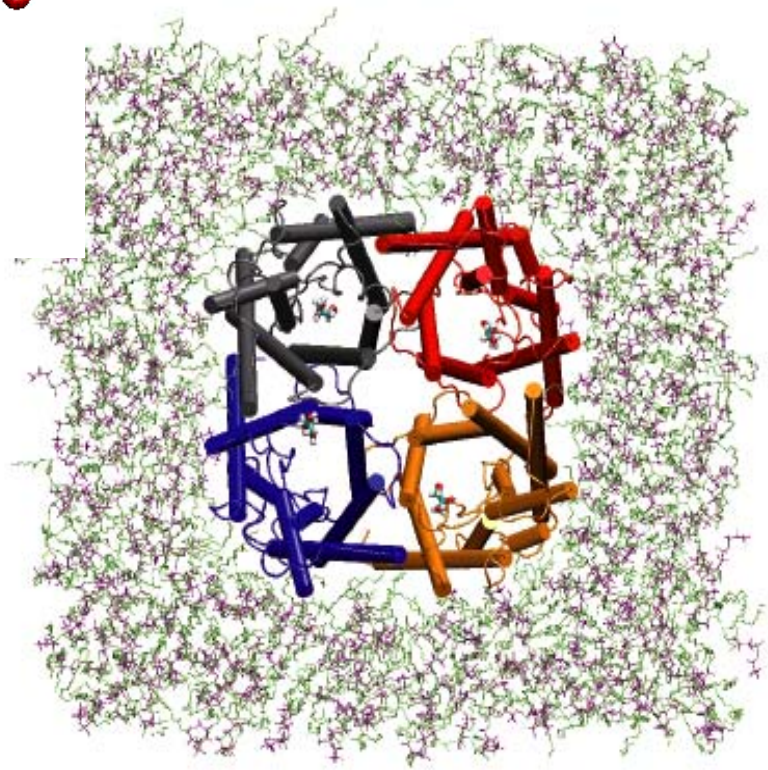
NAMD, CHARMM27, PME
NpT ensemble at 310 or 298 K
1ns equilibration, 4ns production

Protein: ~ 15,000 atoms
Lipids (POPE): ~ 40,000 atoms
Water: ~ 51,000 atoms
Total: ~ 106,000 atoms

3.5 days / ns - 128 O2000 CPUs
11 days / ns - 32 Linux CPUs
.35 days/ns–512 LeMieux CPUs







F. Zhu, E.T., K. Schulten, *FEBS Lett.* **504,** 212 (2001)
M. Jensen, E.T., K. Schulten, *Structure* **9,** 1083 (2001)

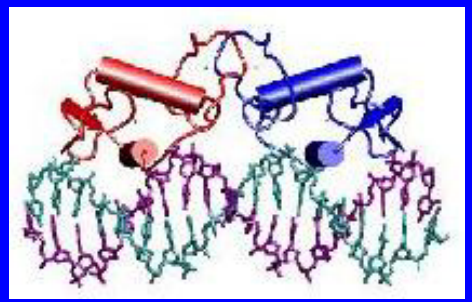
Molecular Dynamics

- Collection of [charged] atoms, with bonds
  - Newtonian mechanics
  - Thousands of atoms (10,000 - 500,000)
- At each time-step
  - Calculate forces on each atom
    - Bonds:
    - Non-bonded: electrostatic and van der Waal's
      - Short-distance: every timestep
      - Long-distance: using PME (3D FFT)
      - Multiple Time Stepping : PME every 4 timesteps
  - Calculate velocities and advance positions
- Challenge: femtosecond time-step, millions needed!

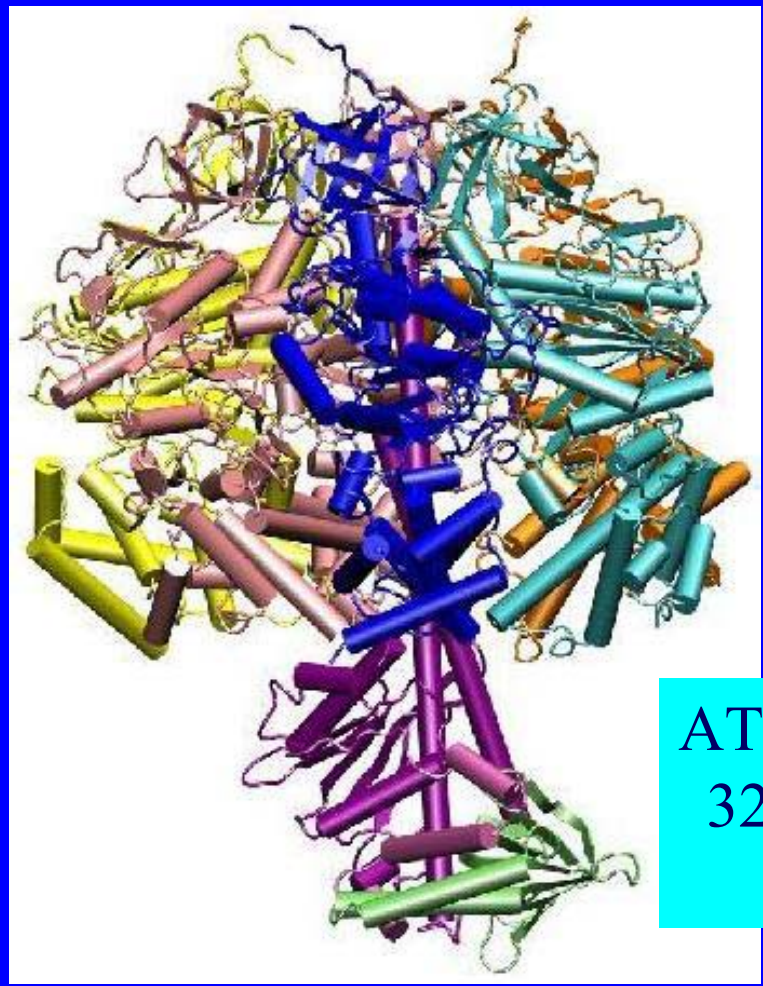
Collaboration with K. Schulten, R. Skeel, and coworkers

Sizes of Simulations Over Time

BPTI
3K atoms



Estrogen Receptor
36K atoms (1996)



ATP Synthase
327K atoms
(2001)

Parallel MD: Easy or Hard?

- Easy
  - Tiny working data
  - Spatial locality
  - Uniform atom density
  - Persistent repetition
  - Multiple timestepping
- Hard
  - Sequential timesteps
  - Short iteration time
  - Full electrostatics
  - Fixed problem size
  - Dynamic variations
  - Multiple timestepping!

Other MD Programs for Biomolecules

- CHARMM
- Amber
- GROMACS
- NWChem
- LAMMPS

Scalability

- The Program should scale up to use a large number of processors.
  - But what does that mean?
- An individual simulation isn't truly scalable
- Better definition of scalability:
  - If I double the number of processors, I should be able to retain parallel efficiency by increasing the problem size

Scalability

- The Program should scale up to use a large number of processors.
  - But what does that mean?
- An individual simulation isn't truly scalable
- Better definition of scalability:
  - If I double the number of processors, I should be able to retain parallel efficiency by increasing the problem size

Isoefficiency

- Quantify scalability
- How much increase in problem size is needed to retain the same efficiency on a larger machine?
- Efficiency : Seq. Time/ (P · Parallel Time)
  - parallel time =
    - computation + communication + idle
- Scalability:
  - If P increases, can I increase N, the problem-size so that the communication/computation ratio remains the same?
- Corollary:
  - If communication /computation ratio of a problem of size N running on P processors increases with P, it can't scale

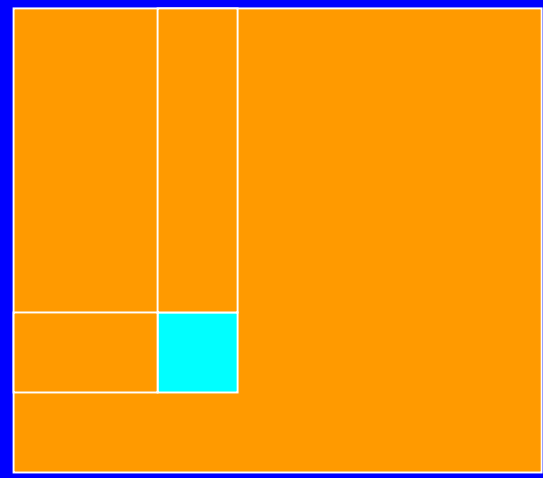
Traditional Approaches

- Replicated Data:
  - All atom coordinates stored on each processor
  - Non-bonded Forces distributed evenly
  - Analysis: Assume N atoms, P processors
    - Computation: O(N/P)
    - Communication: O(N log P)
    - Communication/Computation ratio: P log P
    - Fraction of communication increases with number of processors, independent of problem size!

Atom decomposition

- Partition the Atoms array across processors
  - Nearby atoms may not be on the same processor
  - Communication: O(N) per [illegible]
  - Communication [illegible] O(P)

Force Decomposition



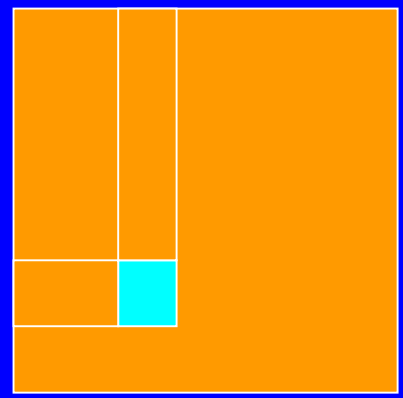
- Distribute force matrix to processors
  - Matrix is sparse, non uniform
  - Each processor has one block
  - Communication: N/sqrt(P)
  - Ratio: sqrt(P)

- Better scalability (use 100+ processors)
  - Hwang, Saltz, et al:
  - 6% on 32 Pes → 36% on 128 processor

Traditional Approaches: non isoefficient

- Replicated Data:
  - All atom coordinates stored on each processor
    - Communication/Computation ratio: P log P
- Partition the Atoms array across processors
  - Nearby atoms may not be on the same processor
  - C/C ratio: O(P)
- Distribute force matrix to processors
  - Matrix is sparse, non uniform,
  - C/C Ratio: sqrt(P)

Not Scalable



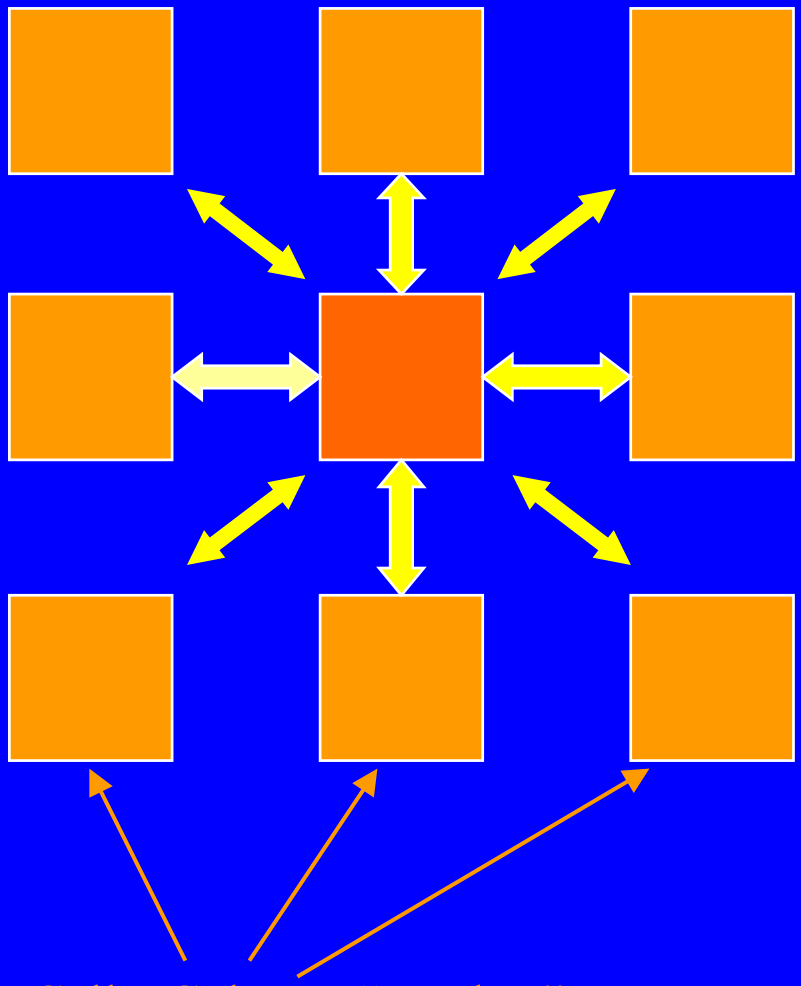
Spatial Decomposition

- Allocate close-by atoms to the same processor
- Three variations possible:
  - Partitioning into P boxes, 1 per processor
    - Good scalability, but hard to implement
  - Partitioning into fixed size boxes, each a little larger than the cutoff disctance
  - Partitioning into smaller boxes
- Communication: O(N/P)

Spatial Decomposition in NAMD

- NAMD 1 used spatial decomposition
- Good theoretical isoefficiency, but for a fixed size system, load balancing problems
- For midsize systems, got good speedups up to 16 processors….
- Use the symmetry of Newton's 3rd law to facilitate load balancing

Spatial Decomposition Via Charm

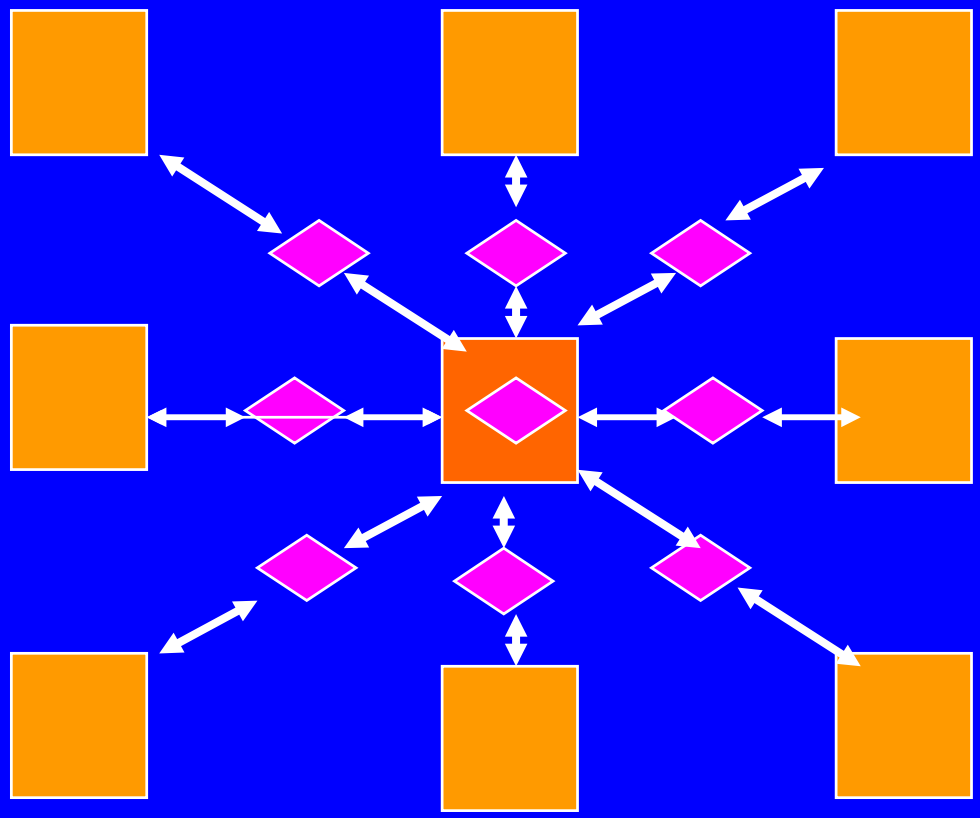


- Atoms distributed to cubes based on their location
- Size of each cube :
  - Just a bit larger than cut-off radius
  - Communicate only with neighbors
  - Work: for each pair of nbr objects
- C/C ratio: O(1)
- ***However:***
  - ***Load Imbalance***
  - ***Limited Parallelism***

Charm++ is useful to handle this

Object Based Parallelization for MD:

Force Decomposition + Spatial Decomposition



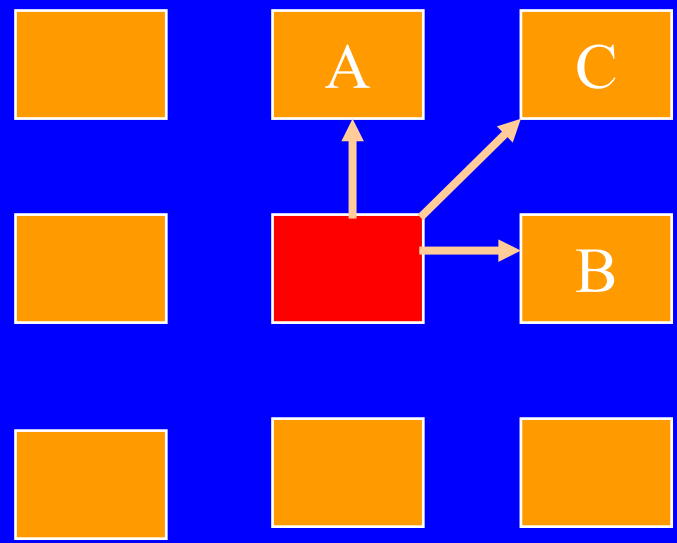
- Now, we have many objects to load balance:
  - –Each diamond can be assigned to any proc.
  - – Number of diamonds (3D):
  - –14·Number of Patches

Bond Forces

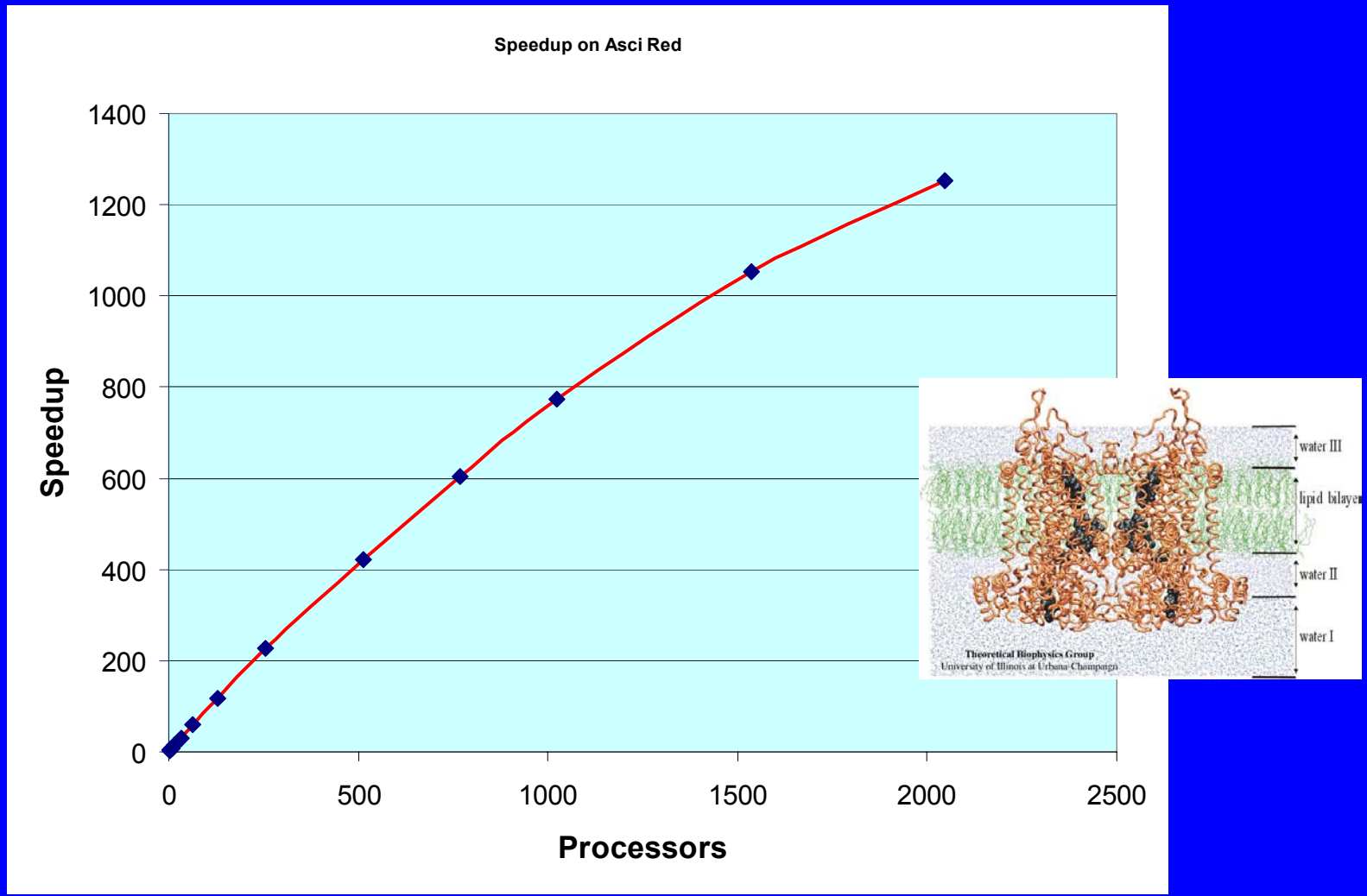
- Multiple types of forces:
  - Bonds(2), Angles(3), Dihedrals (4), ..
  - Luckily, each involves atoms in neighboring patches only
- Straightforward implementation:
  - Send message to all neighbors,
  - receive forces from them
  - 26*2 messages per patch!

Bond Forces

- Multiple types of forces:
  - Bonds(2), Angles(3), Dihedrals (4), ..
  - Luckily, each involves atoms in neighboring patches only
- Straightforward implementation:
  - Send message to all neighbors,
  - receive forces from them
  - 26*2 messages per patch!
- Instead, we do:
  - Send to (7) upstream nbrs
  - Each force calculated at one patch



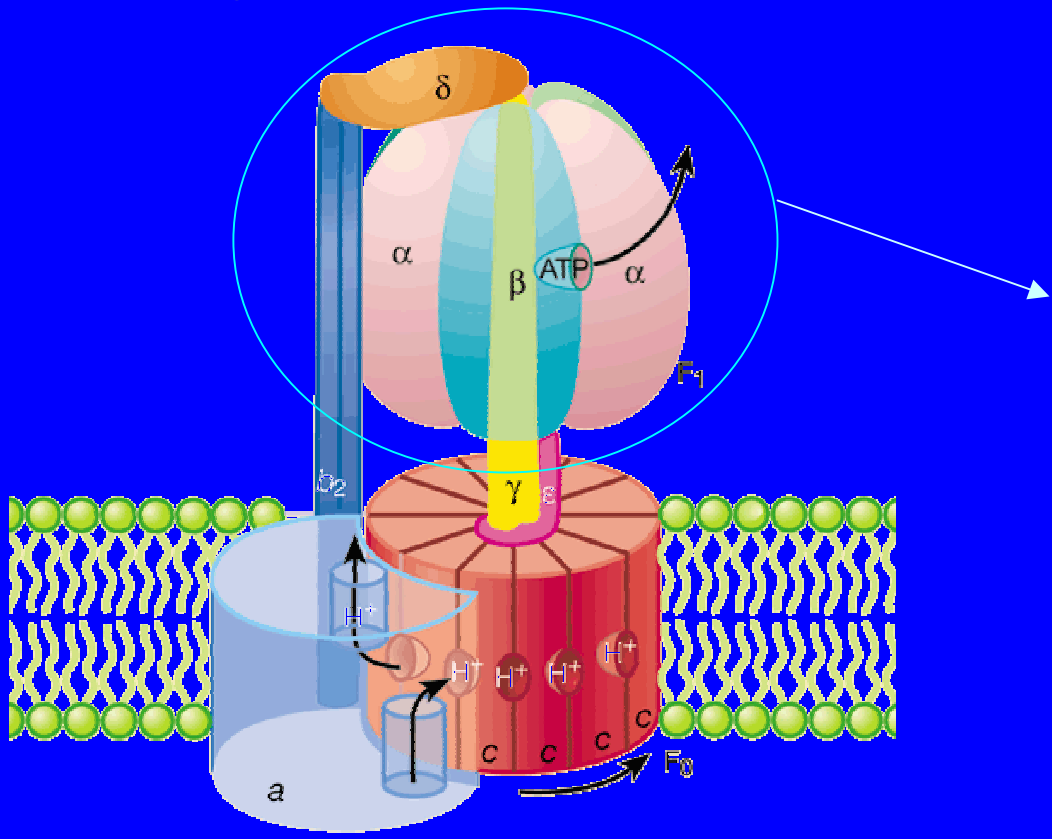
Performance Data: SC2000



New Challenges

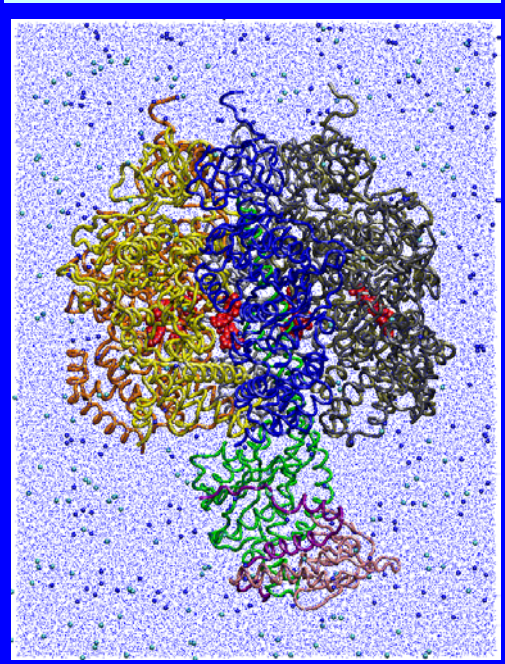
- New parallel machine with faster processors
  - PSC Lemieux
  - 1 processor performance:
    - 57 seconds on ASCI red to 7.08 seconds on Lemieux
  - Makes is harder to parallelize:
    - E.g. larger communication-to-computation ratio
    - Each timestep is few milliseconds on 1000's of processors
- Incorporation of Particle Mesh Ewald (PME)

F_1F_0 ATP-Synthase (ATP-ase)



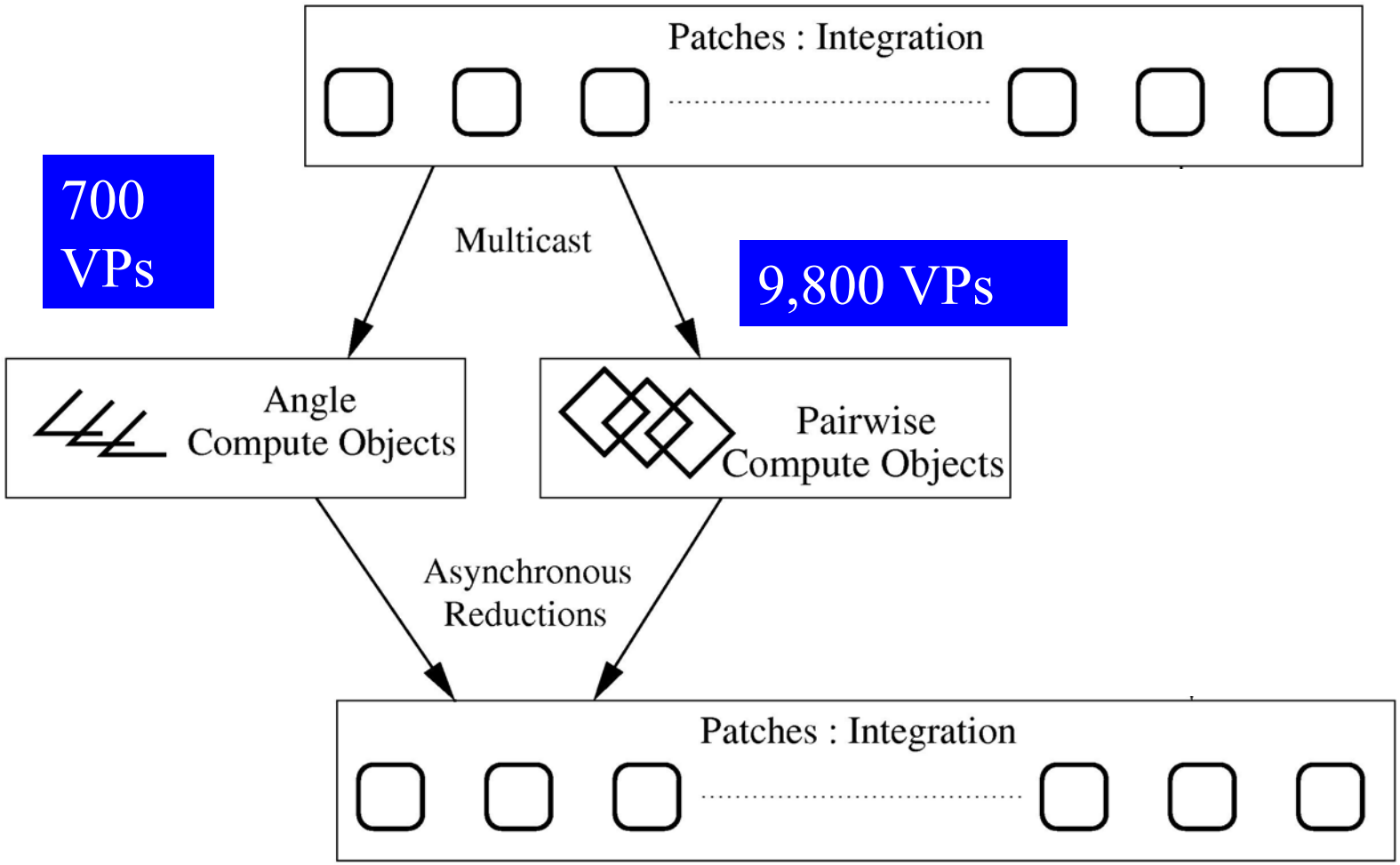
Converts the electrochemical energy of the proton gradient into the mechanical energy of the central stalk rotation, driving ATP synthesis (ΔG = 7.7 kcal/mol).

The Benchmark



327,000 atoms total,
51,000 atoms -- protein and nucletoide
276,000 atoms -- water and ions

NAMD Parallelization using Charm++



These Virtual Processors (VPs) are mapped to real processors by charm runtime system

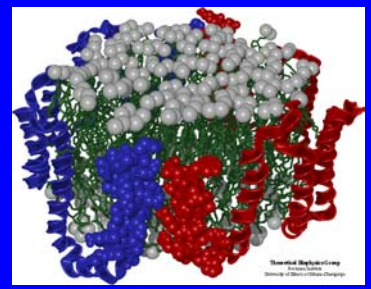
Overview of Performance Optimizations

- Grainsize Optimizations
- Load Balancing Improvements:
  - Explicitly model communication cost
- Using Elan library instead of MPI
- Asynchronous reductions
- Substep dynamic load adjustment

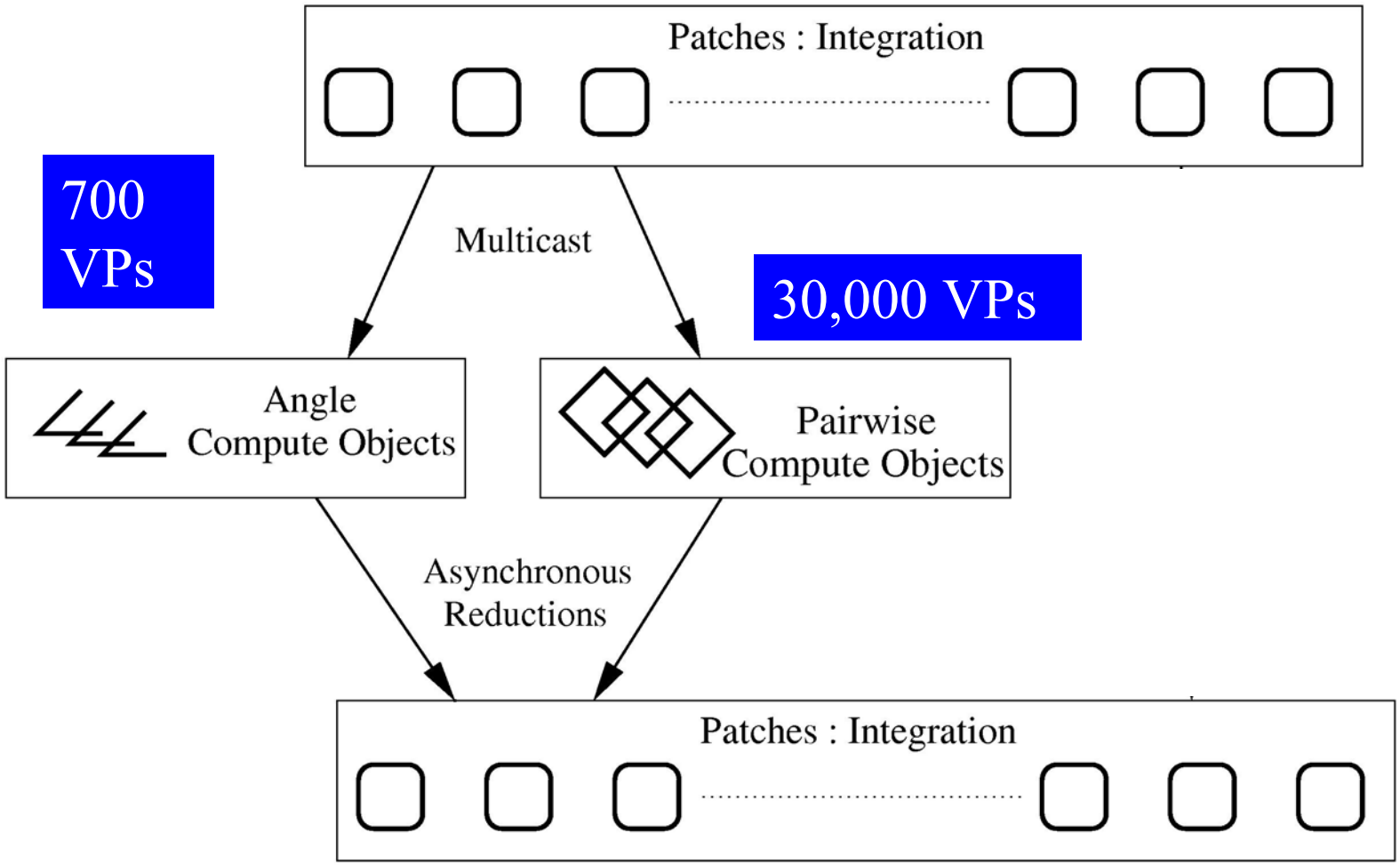
- PME Parallelization

Grainsize and Amdahls's law

- A variant of Amdahl's law, for objects:
  - The fastest time can be no shorter than the time for the biggest single object!
  - Lesson from previous efforts
- Splitting computation objects:
  - 30,000 nonbonded compute objects
  - Instead of approx 10,000

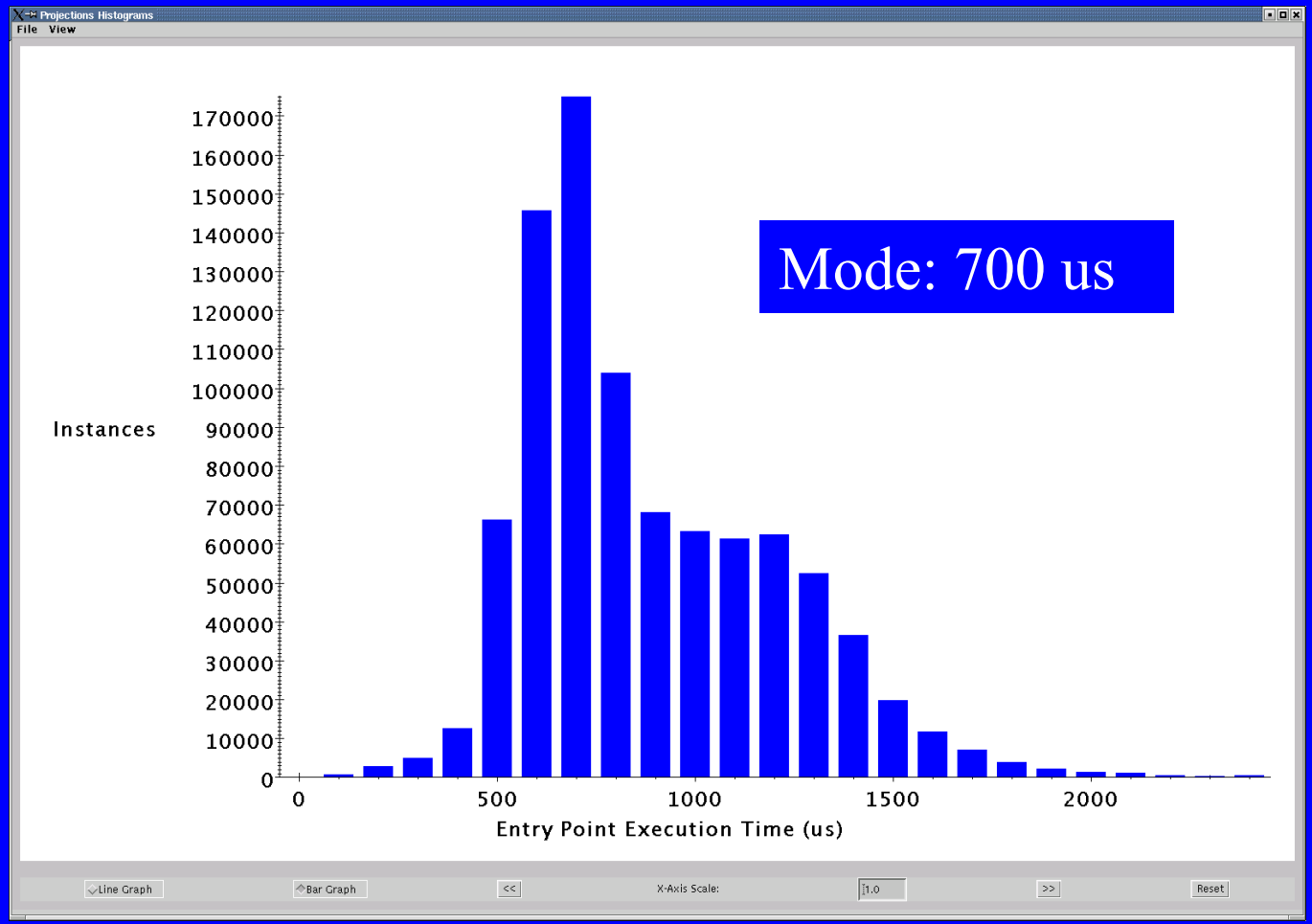


NAMD Parallelization using Charm++



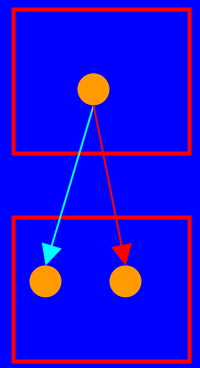
These 30,000+ Virtual Processors (VPs) are mapped to real processors by charm runtime system

Distribution of execution times of non-bonded force computation objects (over 24 steps)



Periodic Load Balancing Strategies

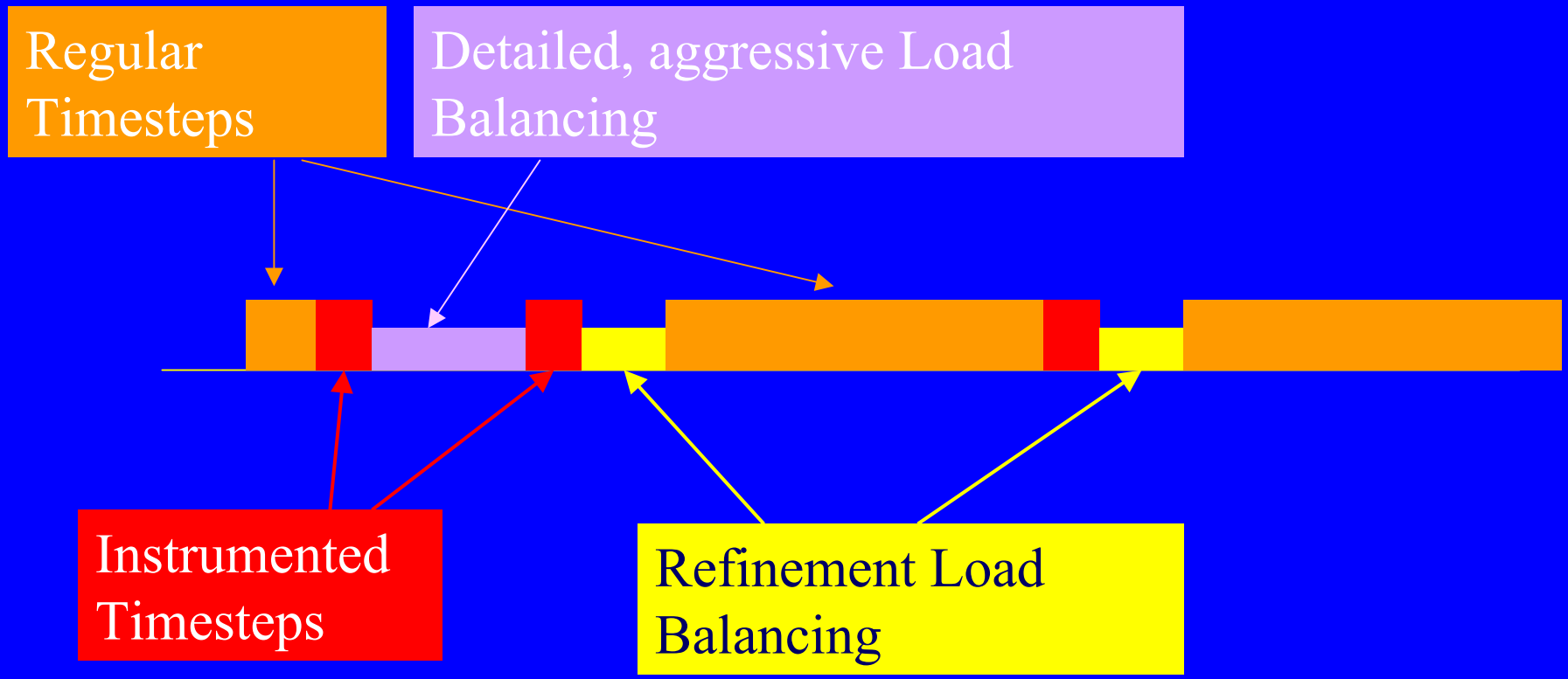
- Centralized strategy:
  - Charm RTS collects data (on one processor) about:
    - Computational Load and Communication for each pair
  - Partition the graph of objects across processors
    - Take communication into account
      - Pt-to-pt, as well as multicast over a subset
      - As you map an object, add to the load on both sending and receiving processor
    - The red communication is free, if it is a multicast.



Measurement Based Load Balancing

- Principle of persistence
  - *Object communication patterns and computational loads tend to persist over time*
  - In spite of dynamic behavior
    - Abrupt but infrequent changes
    - Slow and small changes
- Runtime instrumentation
  - Measures communication volume and computation time
- Measurement based load balancers
  - Use the instrumented data-base periodically to make new decisions

Load Balancing Steps



Another New Challenge

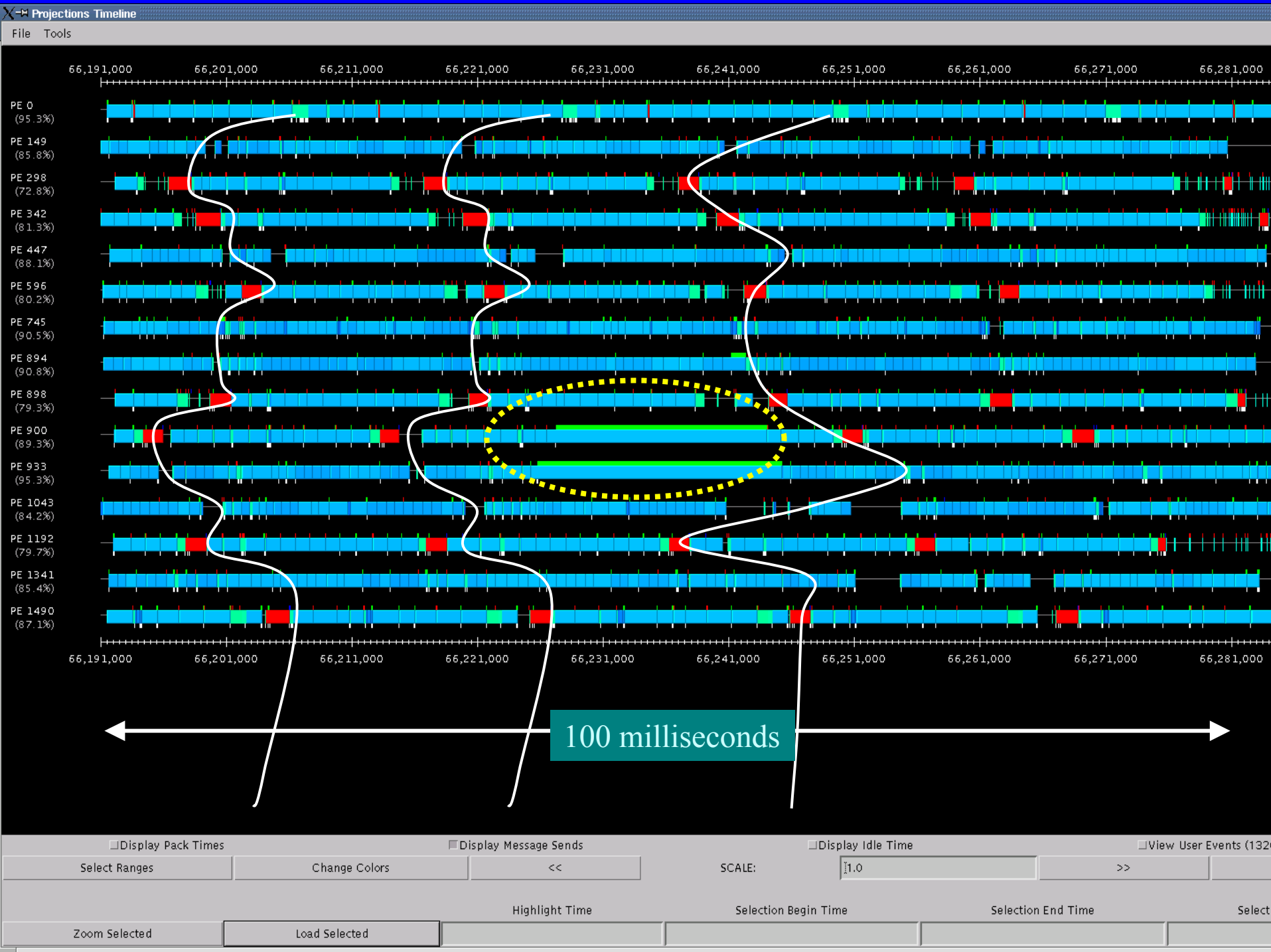
- Jitter due small variations
  - On 2k processors or more
  - Each timestep, ideally, will be about 12-14 msec for ATPase
  - Within that time: each processor sends and receives :
    - Approximately 60-70 messages of 4-6 KB each
  - Communication layer and/or OS has small “hiccups”
    - No problem until 512 processors
    - Small rare hiccups can lead to large performance impact
      - When timestep is small (10-20 msec), AND
      - Large number of processors are used

Benefits of Avoiding Barrier

- Problem with barriers:
  - Not the direct cost of the operation itself as much
  - But it prevents the program from adjusting to small variations
    - E.g. K phases, separated by barriers (or scalar reductions)
    - Load is effectively balanced. But,
      - In each phase, there may be slight non-deterministic load imbalance
      - Let Li,j be the load on I'th processor in j'th phase.

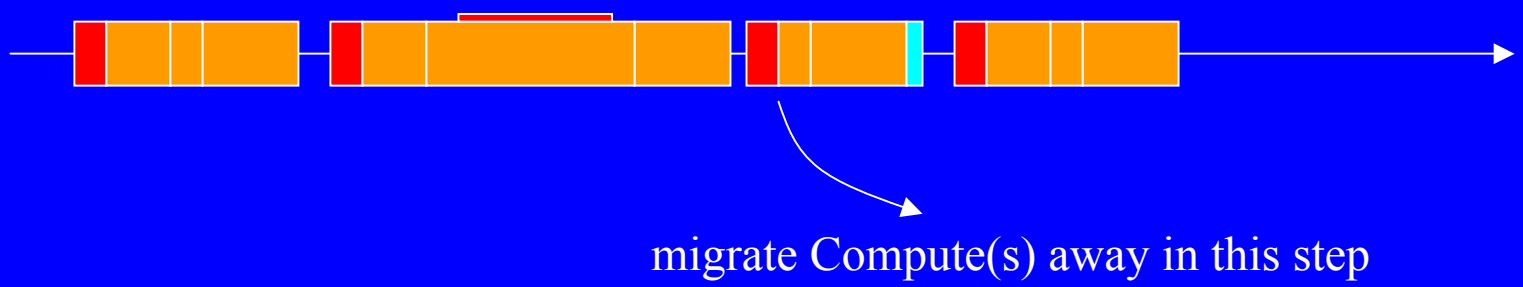
With barrier: $\sum_{j=1}^{k} \max_i \{L_{i,j}\}$ Without: $\max_i \{\sum_{j=1}^{k} L_{i,j}\}$

- In NAMD, using Charm++'s message-driven execution:
  - The energy reductions were made asynchronous
  - No other global barriers are used in cut-off simulations


Projections Timeline
File
Tools
66,191,000
66,201,000
66,211,000
66,221,000
66,231,000
66,241,000
66,251,000
66,261,000
66,271,000
66,281,000
PE 0 (95.3%)
PE 149 (85.8%)
PE 298 (72.8%)
PE 342 (81.3%)
PE 447 (88.1%)
PE 596 (80.2%)
PE 745 (90.5%)
PE 894 (90.8%)
PE 898 (79.3%)
PE 900 (89.3%)
PE 933 (95.3%)
PE 1043 (84.2%)
PE 1192 (79.7%)
PE 1341 (85.4%)
PE 1490 (87.1%)
100 milliseconds
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (132
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Highlight Time
Selection Begin Time
Selection End Time
Select
Zoom Selected
Load Selected

Substep Dynamic Load Adjustments

- Load balancer tells each processor its expected (predicted) load for each timestep
- Each processor monitors its execution time for each timestep
  - after executing each force-computation object
- If it has taken well beyond its allocated time:
  - Infers that it has encountered a "stretch"
  - Sends a fraction of its work in the next 2-3 steps to other processors
    - Randomly selected from among the least loaded processors



NAMD on Lemieux without PME

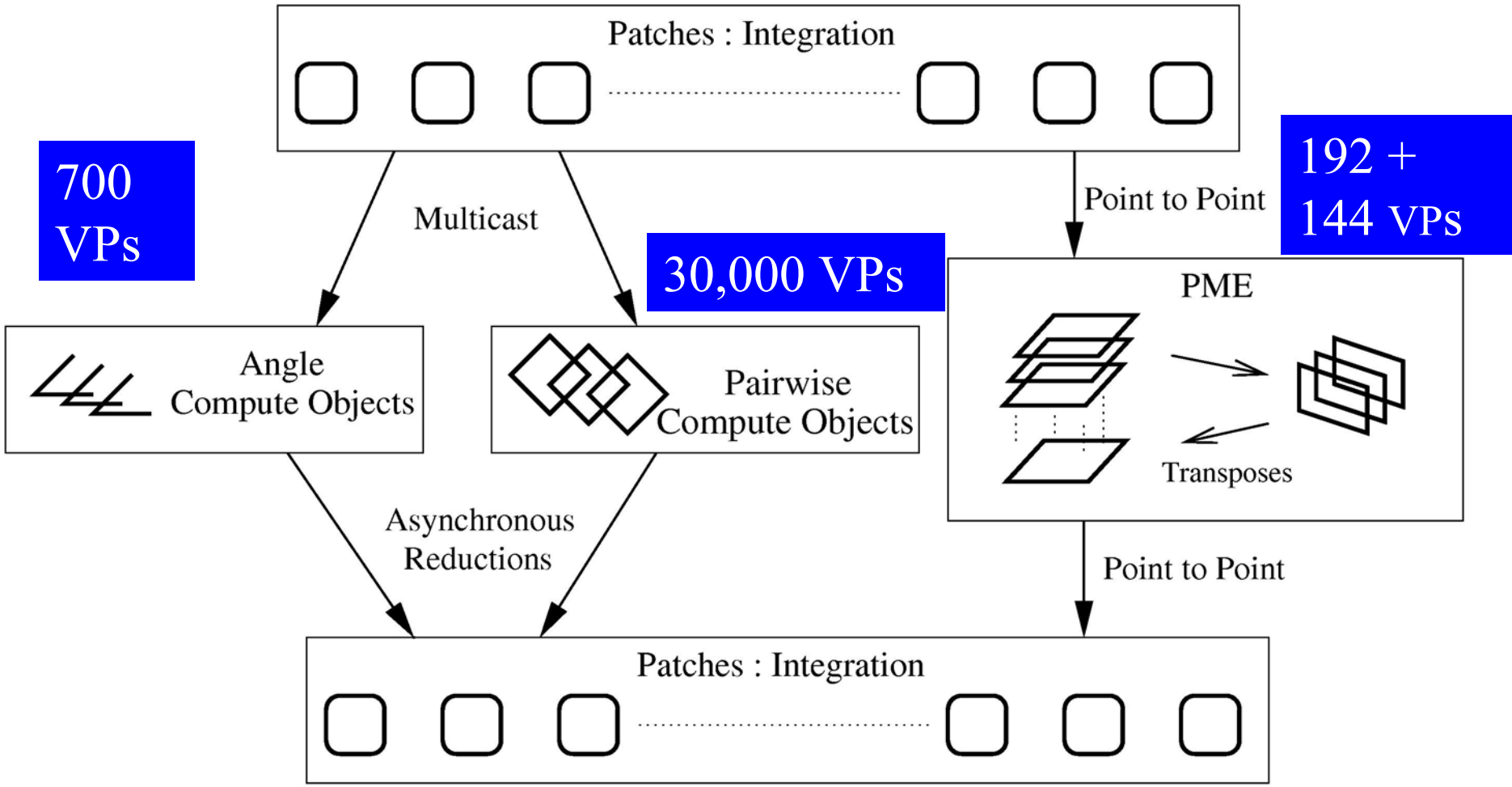
Procs	Per Node	Time (ms)	Speedup	GFLOPS
1	1	24890	1	0.494
128	4	207.4	119	59
256	4	105.5	236	116
512	4	55.4	448	221
510	3	54.8	454	224
1024	4	33.4	745	368
1023	3	29.8	835	412
1536	3	21.2	1175	580
1800	3	18.6	1340	661
2250	3	14.4	1728	850

ATPase: 327,000+ atoms including water

Adding PME

- PME involves:
  - A grid of modest size (e.g. 192x144x144)
  - Need to distribute charge from patches to grids
  - 3D FFT over the grid
- Strategy:
  - Use a smaller subset (non-dedicated) of processors for PME
  - Overlap PME with cutoff computation
  - Use individual processors for both PME and cutoff computations
  - Multiple timestepping

NAMD Parallelization using Charm++ : PME

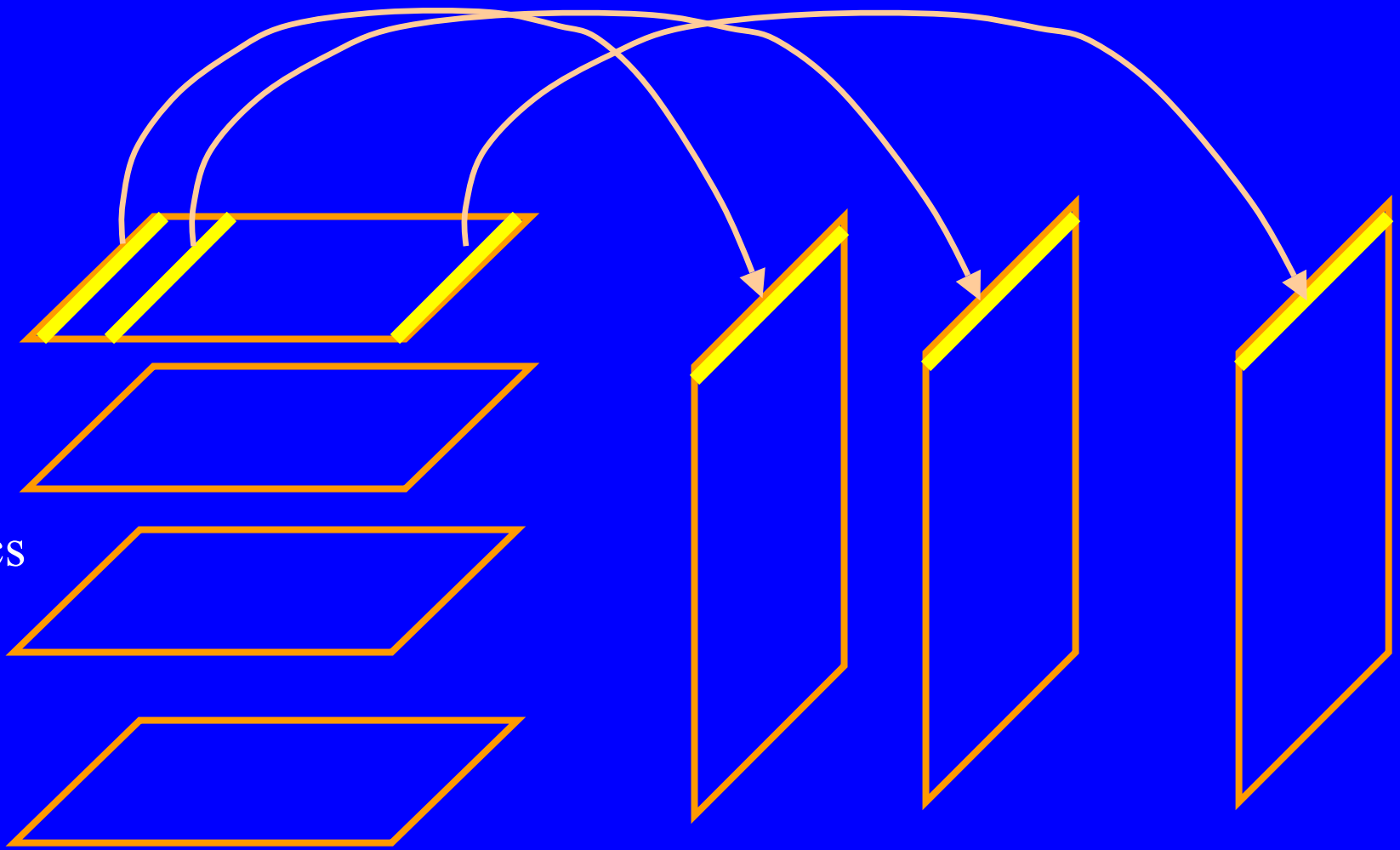


These 30,000+ Virtual Processors (VPs) are mapped to real processors by charm runtime system

Optimizing PME

- Initially, we used FFTW for parallel 3D FFT
  - FFTW is very fast, optimizes by analyzing machine and FFT size, and creates a "plan".
  - However, parallel FFTW was unsuitable for us:
    - FFTW not optimize for "small" FFTs needed here
    - Optimizes for memory, which is unnecessary here.
- Solution:
  - Used FFTW only sequentially (2D and 1D)
  - Charm++ based parallel transpose
  - Allows overlapping with other useful computation

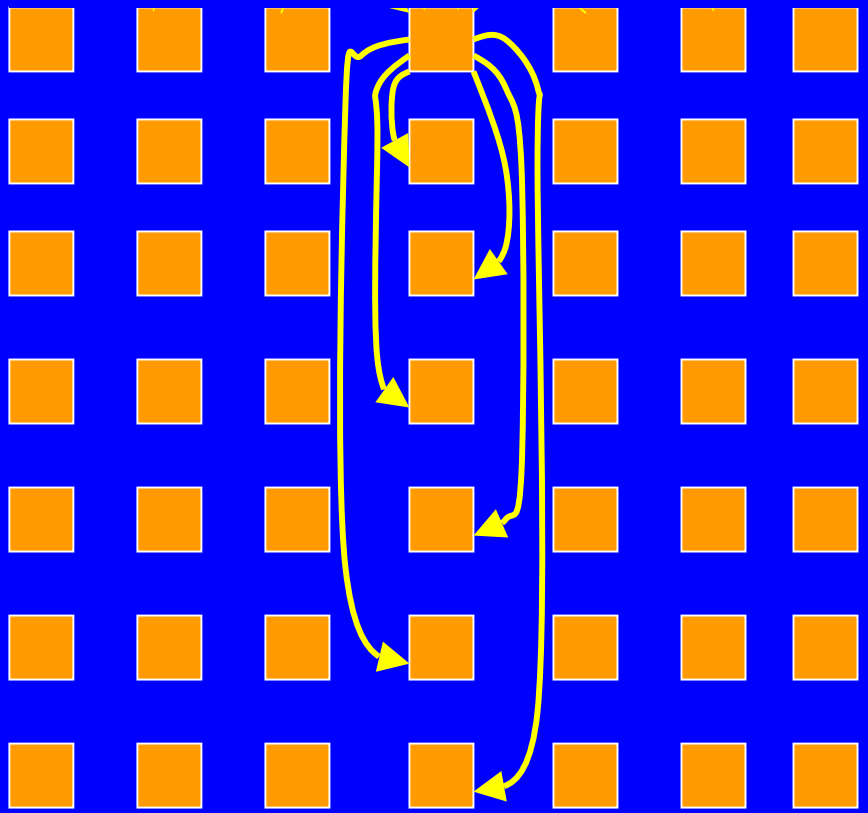
Communication Pattern in PME



Optimizing Transpose

- Transpose can be done using MPI all-to-all
  - But: costly
    - Direct point-to-point messages were faster
  - Per message cost significantly larger compared with total per-byte cost (600-800 byte messages)
- Solution:
  - Mesh-based all-to-all
  - Organized destination processors in a virtual 2D grid
  - Message from (x1,y1) to (x2,y2) goes via (x1,y2)
  - 2.sqrt(P) messages instead of P-1.
  - For us: 28 messages instead of 192.

All to all via Mesh



Organize processors in a 2D (virtual) grid

Phase 1:

*E*ach processor sends $(\sqrt{P}-1)$ messages within its row

Phase 2:

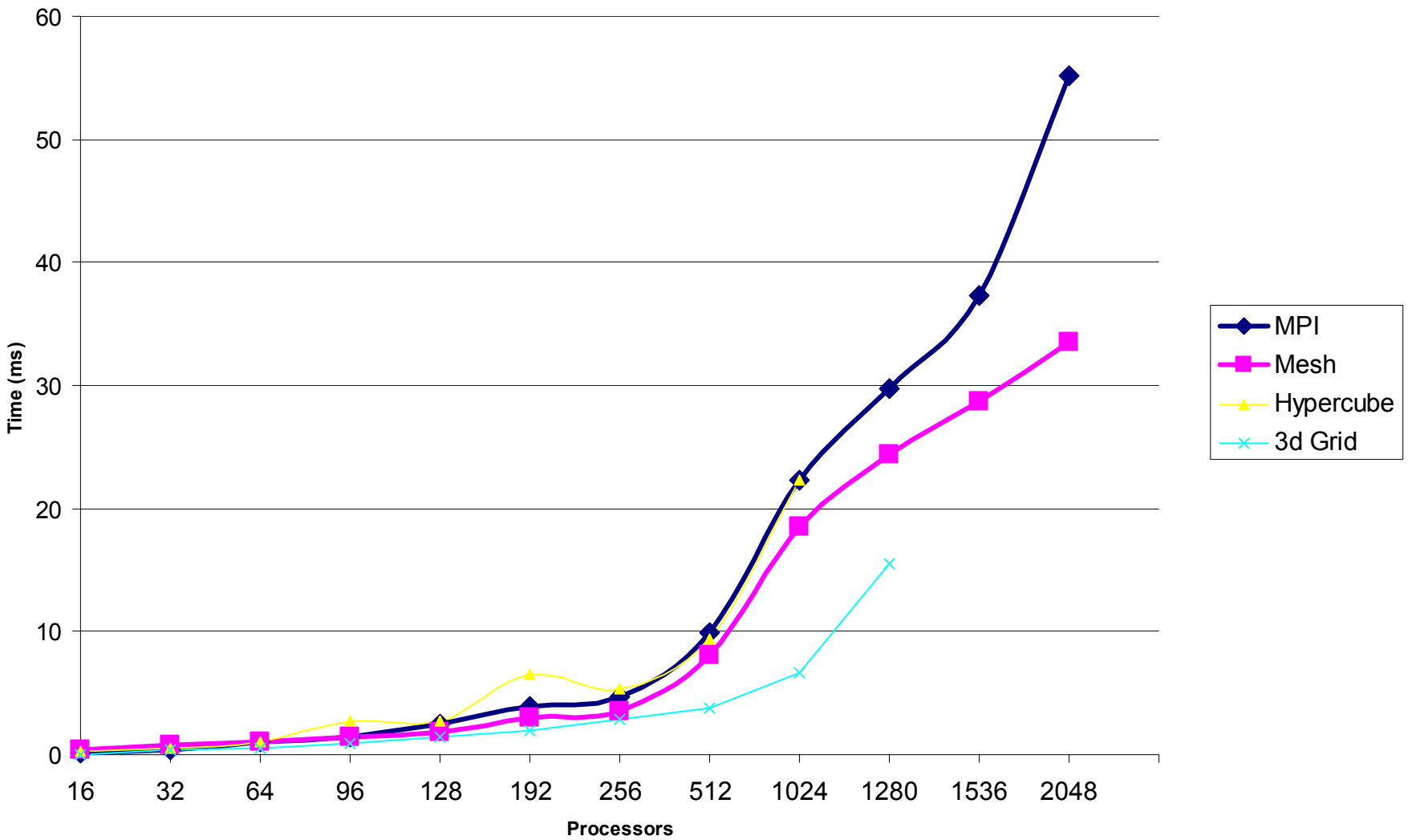
*E*ach processor sends $(\sqrt{P}-1)$ messages within its column

Message from (x1,y1) to (x2,y2) goes via (x1,y2)

2. $(\sqrt{P}-1)$ messages instead of P-1

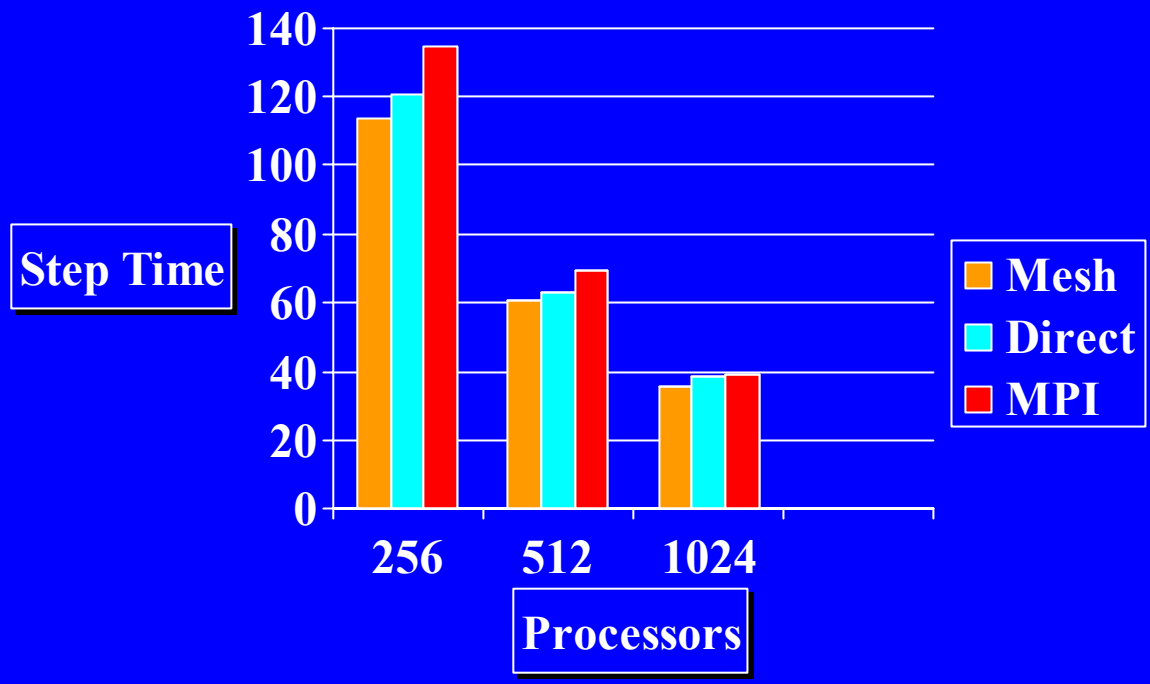
For us: 26 messages instead of 192

All to all on Lemieux for a 76 Byte Message

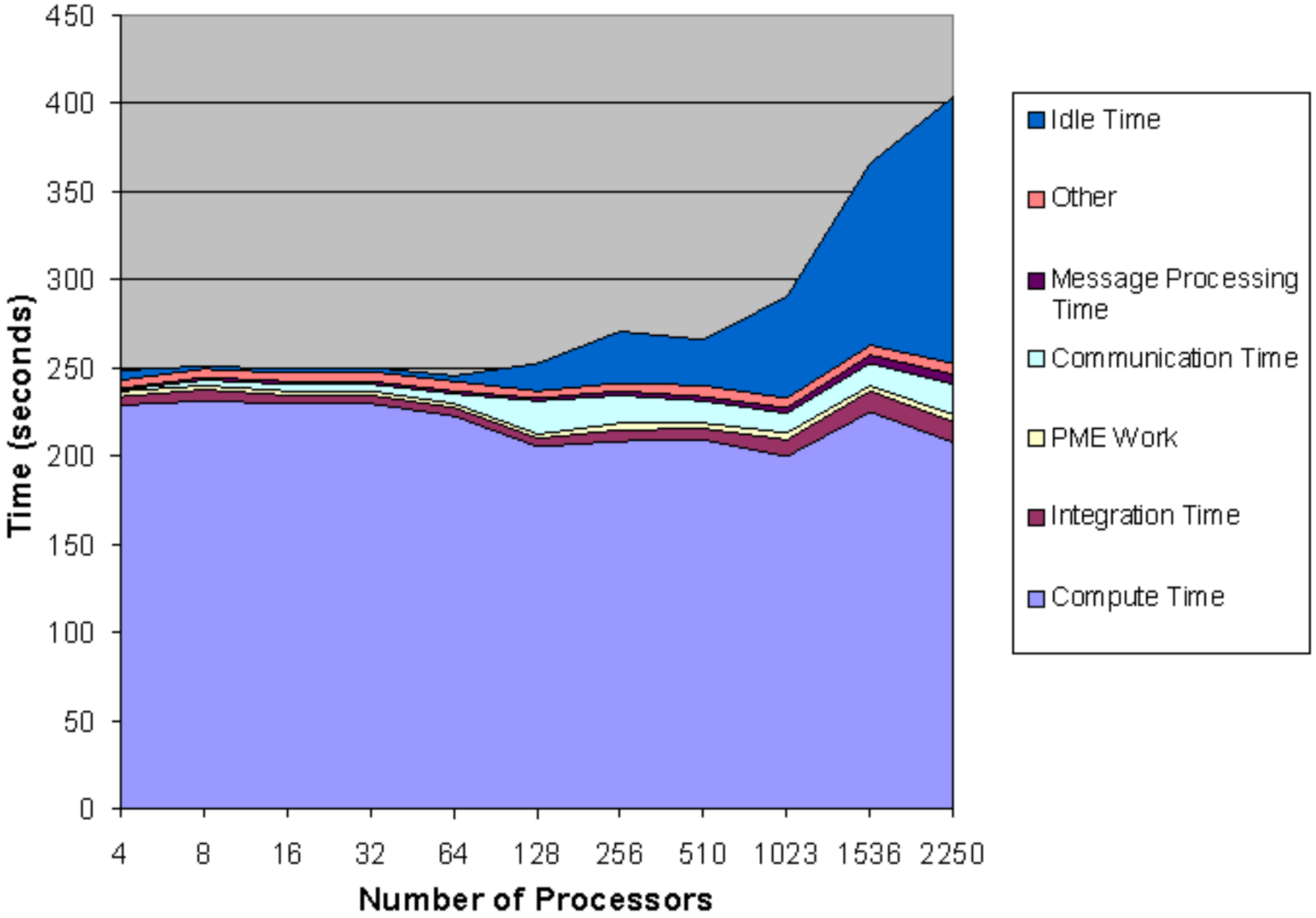


Impact on Namd Performance

Namd Performance on Lemieux, with the transpose step implemented using different all-to-all algorithms



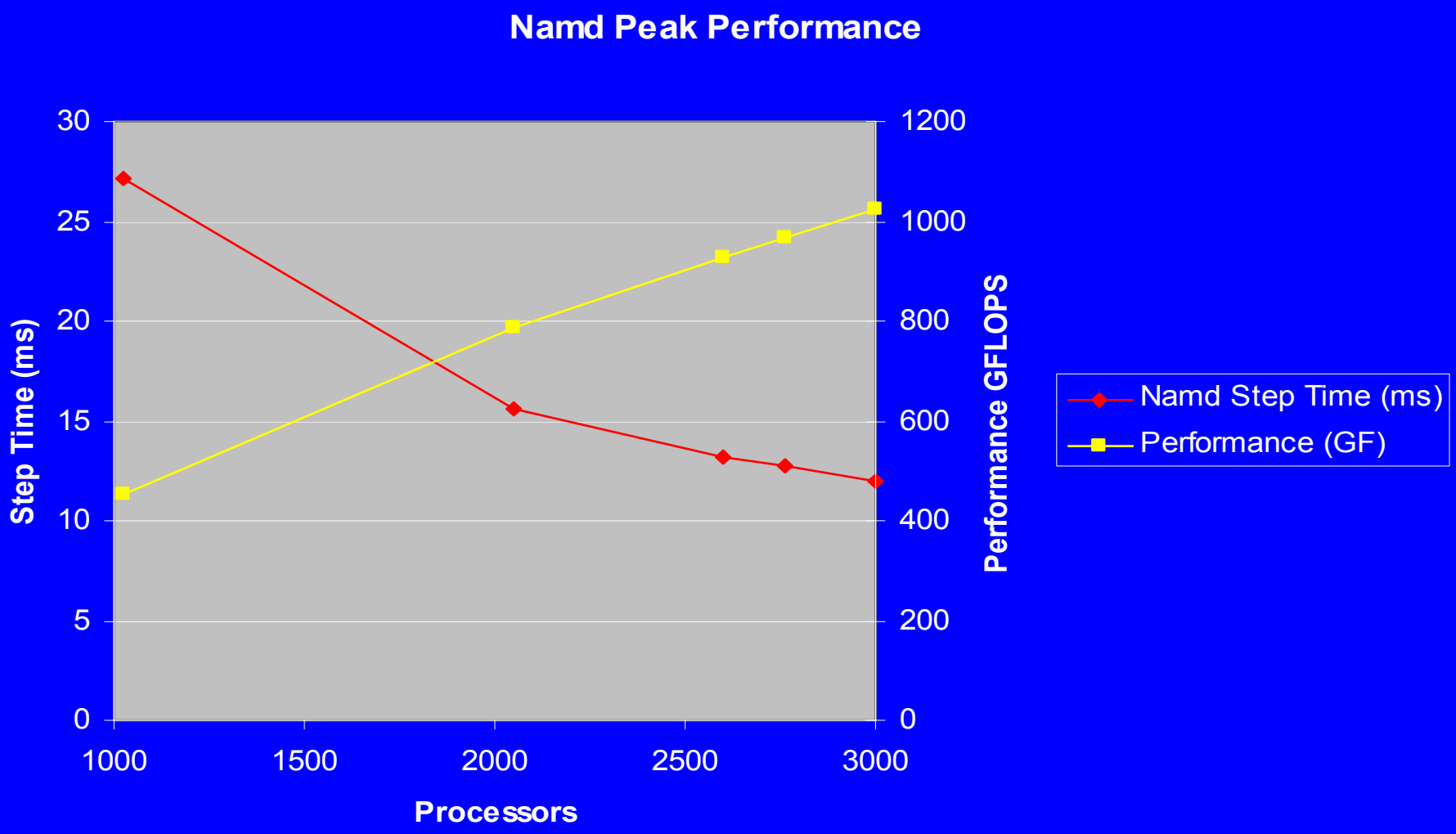
Relative Scaling



Performance: NAMD on Lemieux

		Time (ms)			Speedup			GFLOPS		
Procs	Per Node	Cut	PME	MTS	Cut	PME	MTS	Cut	PME	MTS
1	1	24890	29490	28080	1	1	1	0.494	0.434	0.48
128	4	207.4	249.3	234.6	119	118	119	59	51	57
256	4	105.5	135.5	121.9	236	217	230	116	94	110
512	4	55.4	72.9	63.8	448	404	440	221	175	211
510	3	54.8	69.5	63	454	424	445	224	184	213
1024	4	33.4	45.1	36.1	745	653	778	368	283	373
1023	3	29.8	38.7	33.9	835	762	829	412	331	397
1536	3	21.2	28.2	24.7	1175	1047	1137	580	454	545
1800	3	18.6	25.8	22.3	1340	1141	1261	661	495	605
2250	3	14.4	23.5	17.54	1728	1256	1601	850	545	770

ATPase: 320,000+ atoms including water


Namd Peak Performance
Step Time (ms)
Performance GFLOPS
Processors
30
25
20
15
10
5
0
1200
1000
800
600
400
200
0
1000
1500
2000
2500
3000
Namd Step Time (ms)
Performance (GF)

Conclusion: NAMD case study

- We have been able to effectively parallelize MD,
  - A challenging application
  - On realistic Benchmarks
  - To 2250 processors, 850 GF, and 14.4 msec timestep
  - To 2250 processors, 770 GF, 17.5 msec timestep with PME and multiple timestepping
- These constitute unprecedented performance for MD
  - 20-fold improvement over our results 2 years ago
  - Substantially above other production-quality MD codes for biomolecules
- Using Charm++'s runtime optimizations
    - Automatic load balancing
    - Automatic overlap of communication/computation
      - Even across modules: PME and non-bonded
    - Communication libraries: automatic optimization