Load Balancing in Parallel Molecular Dynamics

Laxmikant V. Kalé,

Milind Bhandarkar, and Robert Brunner

Department of Computer Science

University of Illinois

`kale@cs.uiuc.edu`

`http://charm.cs.uiuc.edu`

Molecular Dynamics

- Simulate the motions of collections of atoms
- Forces due to bonds and non-bonded (Coulomb and Lennard-Jones) interactions
- Cutoff radius for non-bonded forces
- Sparse, but not very sparse, force matrix
- Configuration changes due to atom movement

Existing Methods

- Computation cost is O(N/P) for cutoff simulations
- Replicated Data: non scalable
- Atom decomposition:
  - Communication: O(N)
- Force decomposition
  - Communication: O($N/\sqrt{P}$)
- "Irregular" force decomposition

Spatial Decomposition

- Fixed size boxes vs. one box per processor
- Scalability of spatial decomposition
  - Computation: O(N/P)
  - Communication: O(N/P)

Difficulties with Spatial Decomposition

- Load imbalance, especially for non-periodic configurations
- Parallelism limited to the number of Boxes

Hybrid Decomposition

- Combines advantages of spatial and force decomposition
- Retains spatial decomposition in boxes
- One force-object for each pair of neighboring boxes
- Load balancer may map each force object to any processor!
- Flexible tradeoff between communication overhead and load imbalance

Specifics of Load Balancing Strategy

Top level structure:

- Initial Balancer
- Learn from previous timesteps: explot temporal locality of performance characteristics
- Measurement based mapping
- Migratable and non-migratable objects

Greedy Algorithm Variant

- Use "Greedy" heuristic, modified to take communication into account

Multi-Paradigm Parallel Programming

- Irregular applications often need Multiparadigm parallel programming

Advantages of multi-paradigm programming

- One can use appropriate language for each module, separately
- Reuse existing libraries, irrespective of the language

Multilingual Parallel Programming

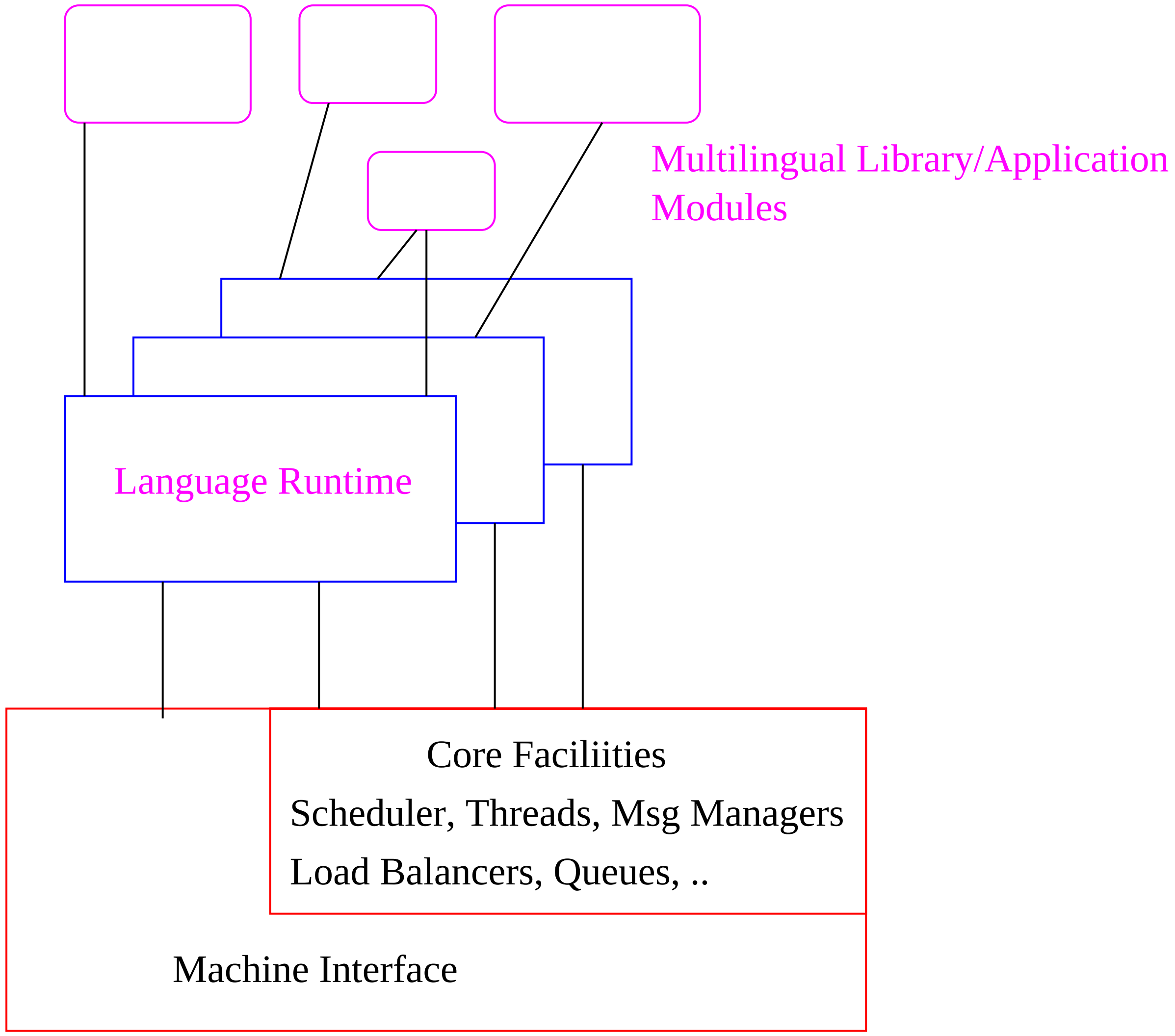
Challenges:

- Interoperating is difficult in face of concurrent languages (such as multi-threaded and object-based languages)
- Languages impose a processor scheduling policy
- Implicit vs. Explicit transfer of control
- Solution: exposed common scheduler (across languages)

The Parallel Programming Framework

- Converse: multilingual interoperability
- Languages: Charm++, Charm, PC++, tSM, tPVM, MPI, threaded MPI, DP (HPF) Import (Simulation language), Agents
- Libraries:
- Applications: NAMD

Using Converse


Multilingual Library/Application Modules
Language Runtime
Core Faciliities
Scheduler, Threads, Msg Managers
Load Balancers, Queues, ..
Machine Interface

Charm++ Overview

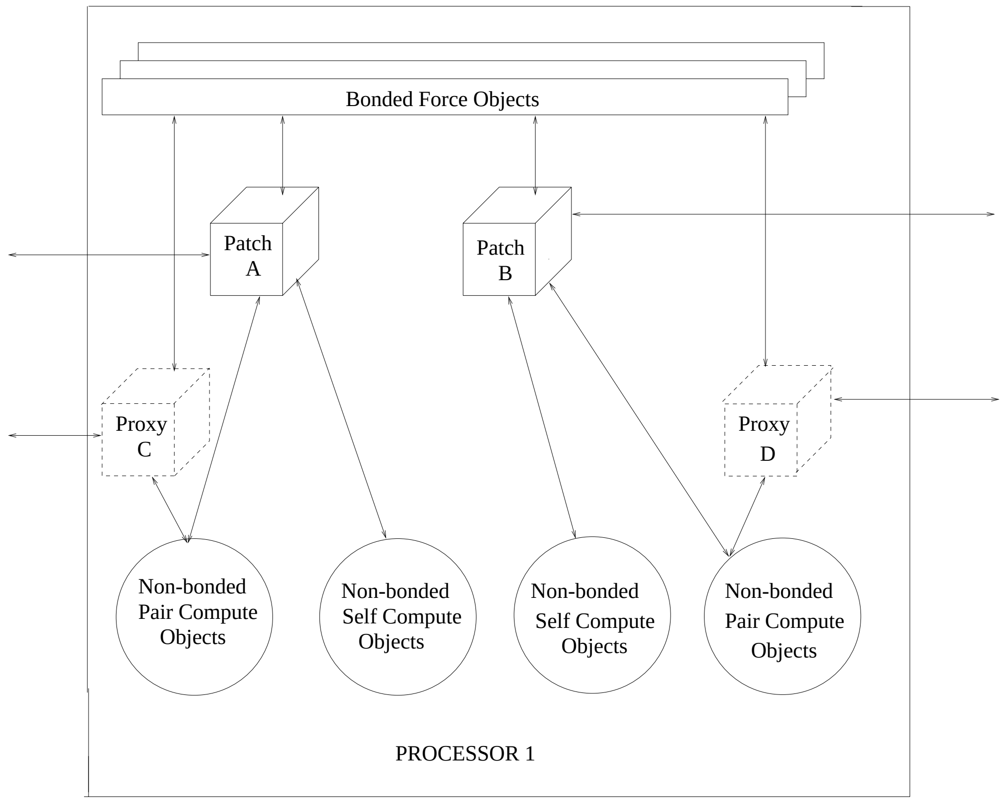
- Separation of sequential and parallel objects
- *Message driven objects*: dynamically load balanced
- *Asynchronous method invocation*: Message driven execution
- *Object groups*: distributed object with a branch on every processor
- *Object Arrays*: remappable
- No globally shared memory, but *globally shared object space.*

More info: http://charm.cs.uiuc.edu

Object Groups

- When an object group is created, one object instance is created on each processor.
- You may invoke a method on any member of the group
- *Broadcast* invocation: when processor number is omitted
- Invoking a method in the local branch may be a synchronous function call

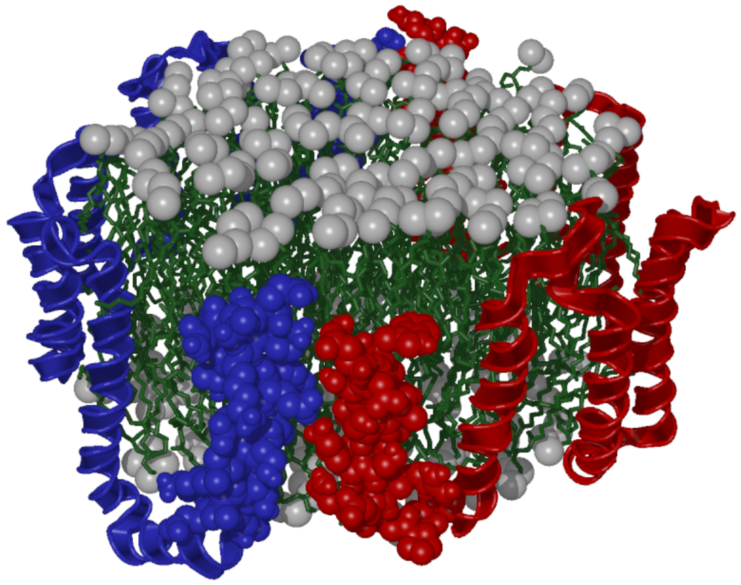
NAMD Multilingual Modules


Bonded Force Objects
Patch A
Patch B
Proxy C
Proxy D
Non-bonded Pair Compute Objects
Non-bonded Self Compute Objects
Non-bonded Self Compute Objects
Non-bonded Pair Compute Objects
PROCESSOR 1

NAMD Information

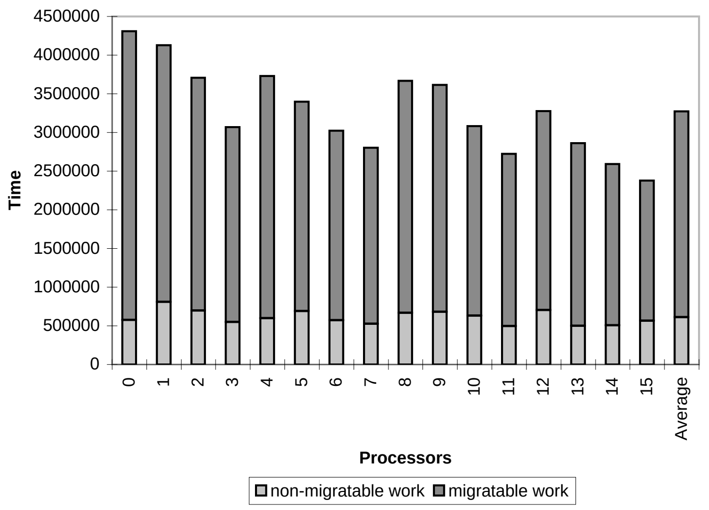
- NAMD is a production-quality program.
  - NAMD 2 contains over 23,000 lines of code
  - DPMTA is an additional 8,000 lines
  - SM is used in modules containing 4,900 lines
- Supports features required by application scientists

Simulations using NAMD



A simulation of Apolipoprotein A-I is currently being done with NAMD. The ApoA-I simulation contains over 90,000 atoms.

Load Balancing



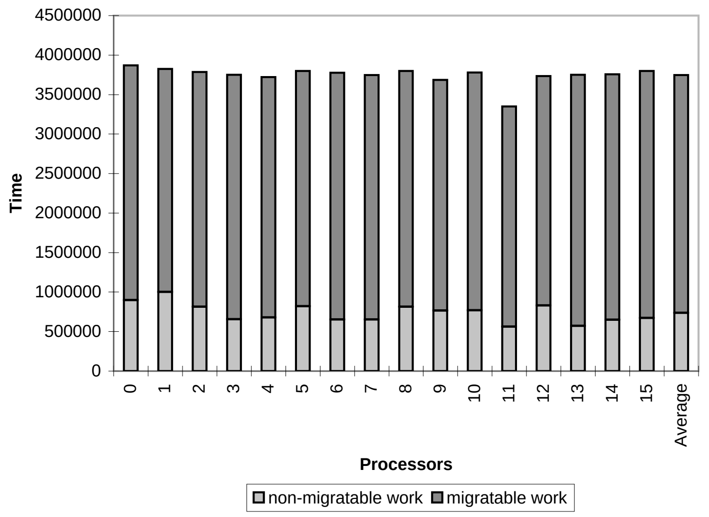


Figure 1: Load distribution for a 16 processor simulation, showing the load before (left) and after (right) running the load balancer.

Performance

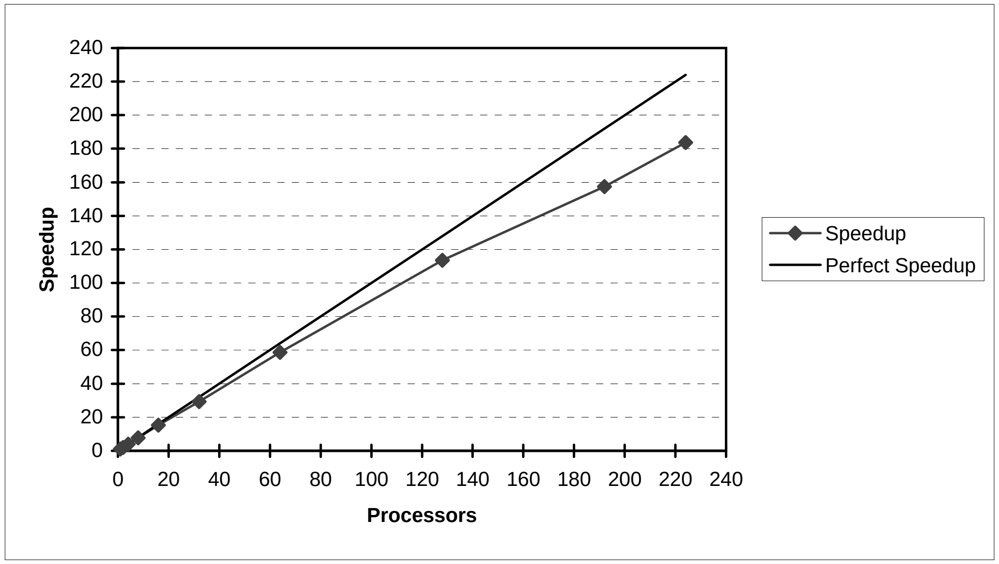


Figure 2: The speed-up for ApoA-I (92,224 atoms, 12Å cutoff) on the ASCI-RED.

Performance

Simulation	Processors								
# of atoms	1	2	4	8	16	32	64	128	160
bR	1.138	0.578	0.315	0.158	0.086	0.048			
(3,762)	1.0	1.97	3.61	7.20	13.2	23.7			
ER-ERE		6.115	3.099	1.598	0.810	0.397	0.212	0.123	0.098
(36,573)		(1.97)	3.89	7.54	14.9	30.3	56.8	97.9	123
ApoA-I			10.760	5.464	2.850	1.470	0.729	0.382	0.321
(92,224)			(3.88)	7.64	14.7	28.4	57.3	109	130

Table 1: Execution time (seconds) per timestep and speedups for several simulations on the CRAY T3E. All the simulations were run using a 12Å cutoff. Some simulations could not run on small numbers of processors due to lack of memory, so speed-up numbers in parantheses are estimates.

Performance

	Processors								
	1	2	8	16	32	64	128	160	192
T3E		6.12	1.60	0.810	0.397	0.212	0.123	0.098	
		(1.97)	7.54	14.9	30.3	56.8	97.9	123	
Origin2000	10.7	5.43	1.37	0.723	0.514	0.987			
	1.0	1.96	7.75	14.7	20.7	10.8			
ASCI-RED	28.0	13.9	3.76	1.91	1.01	0.500	0.279	0.227	0.196
	1.0	2.01	7.45	14.7	27.9	56.0	100	123	143
NOWs	24.1	12.4	3.69						
HP735/125	1.0	1.94	6.54						

Table 2: Execution time (seconds) per timestep and speedups for ER-ERE (36,573 atoms, 12Å cutoff) on several parallel machines.