Exploring performance improvement of Java-based scientific applications that use the Swarm toolkit

Xiaorong Xiang Gregory Madey

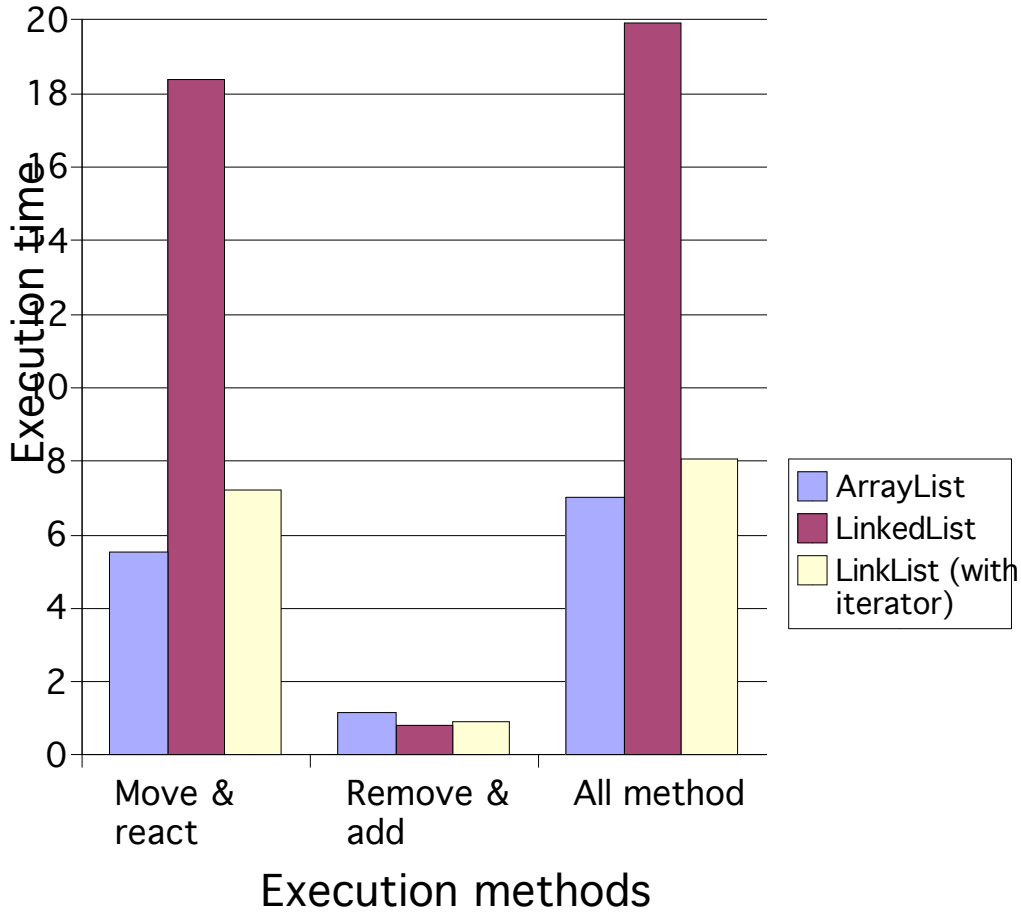
University of Notre Dame

The NOM simulation model

- NOM (Natural Organic Matter) , a mixture of molecular compounds with heterogeneous properties
- NOM, micro-organisms, and their environment form a complex system
- Transformations: transport, adsorption, desorption and other chemical reactions
- A distributed stochastic model using agent-based modeling approach

Data structure

- Molecule object management
- LinkedList or ArrayList ?
- Position access get() in move & react methods
- Shuffle algorithm
- Add & remove operations
- ArrayList is choice

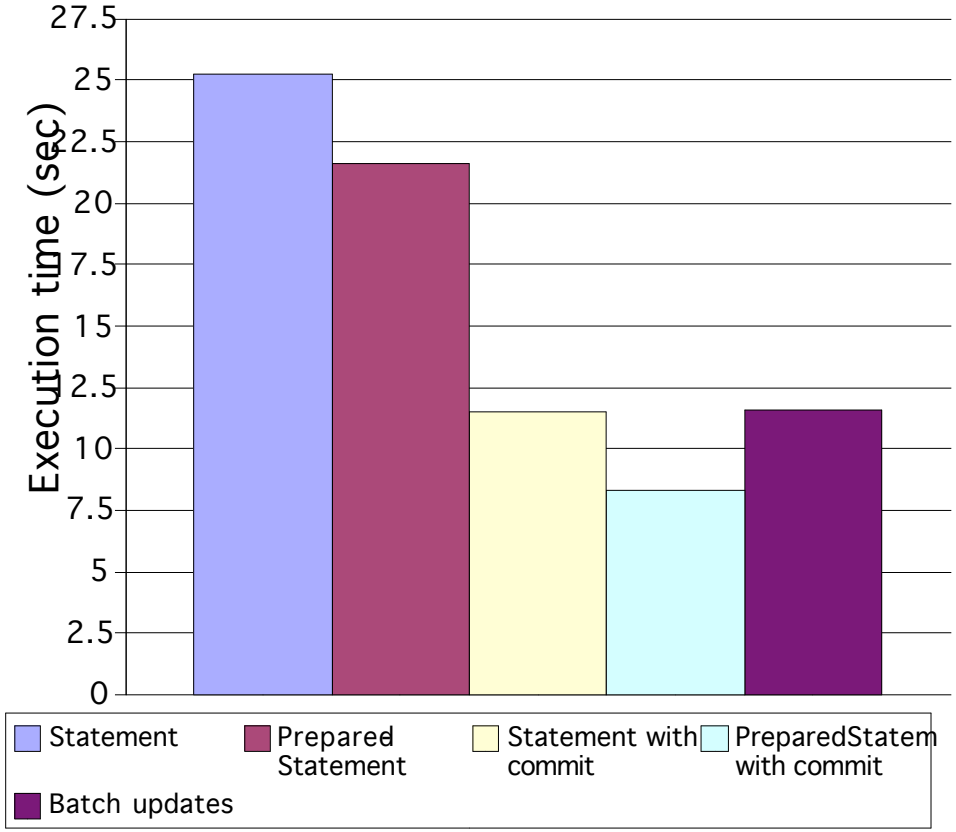


Objects reuse

- Reduce the overhead of object creation
- Reduce the CPU cycle for garbage collection
- Reduce the probability of the potential memory leak
- Steps for objects reuse
  - Isolating objects that need to be created and destroyed frequently
  - Optimizing objects size
  - Objects reinitialize
  - Object pool management (data structure, pool size)

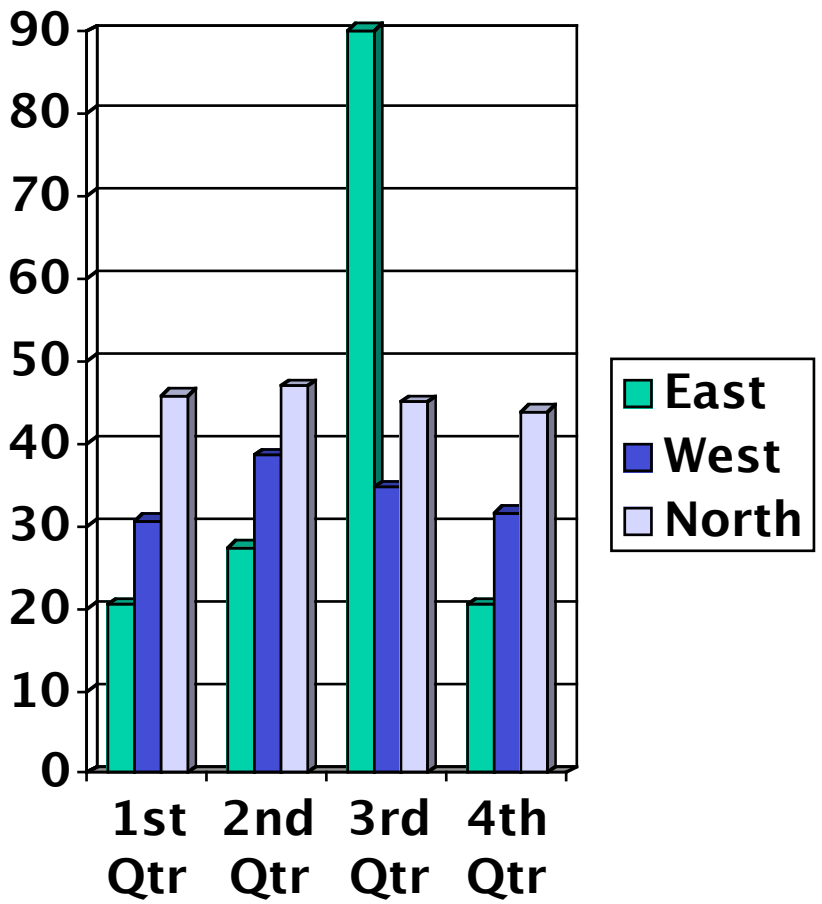
JDBC with Data Insertion

- Connection pooling
- Prepared statement vs. Statement
- Batch updates
- Explicit transaction commit
- PreparedStatement with explicit transaction commit has best performance



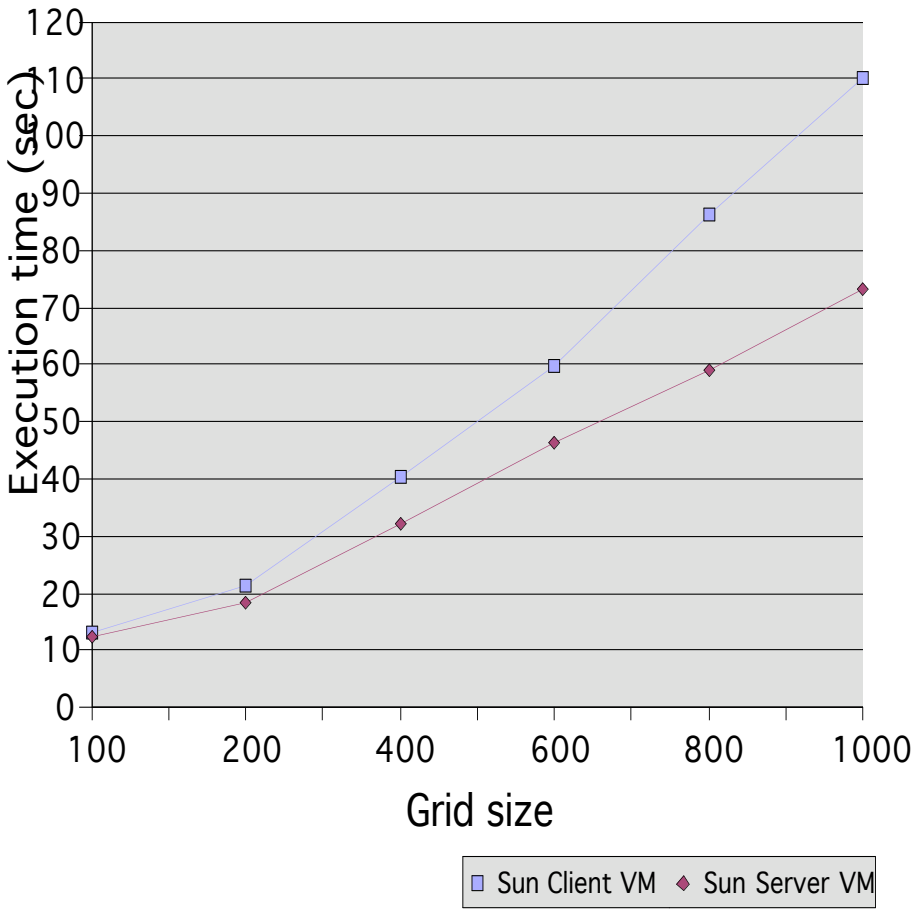
Data output using multi-threading

- Overlap the computation and I/O
- Take advantage of idle CPU time
- About 30%-40% speed up



Runtime environment

- Sun HotSpot Client VM with faster start up
- Sun HotSpot Server VM with advanced dynamic optimizing compiler
- As the problem size increases, larger performance gain over client
- IBM JVM is another choice

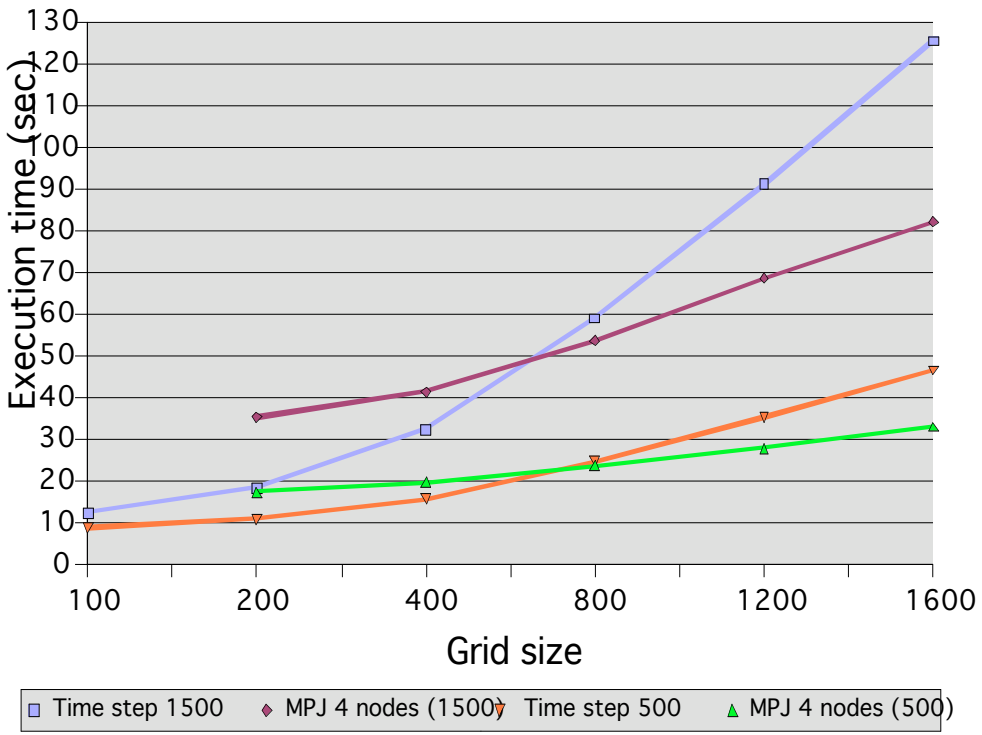


Scalability

- Two aspects of scalability: large grid size and time steps
- Equally separate the grid to 2 or 4 parts
- Exchange the molecules that cross the boundary at each time step
- Two Java threads are used to take advantage of dual CPU
- MPJ (mpiJava) with LAM MPI are used to distribute the job between 2 or 4 nodes.

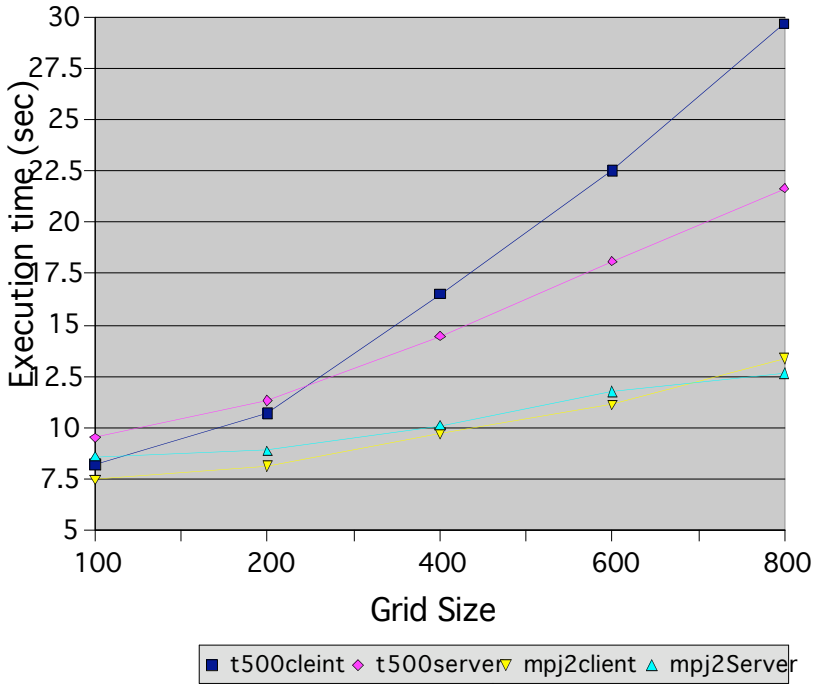
Experiment Results

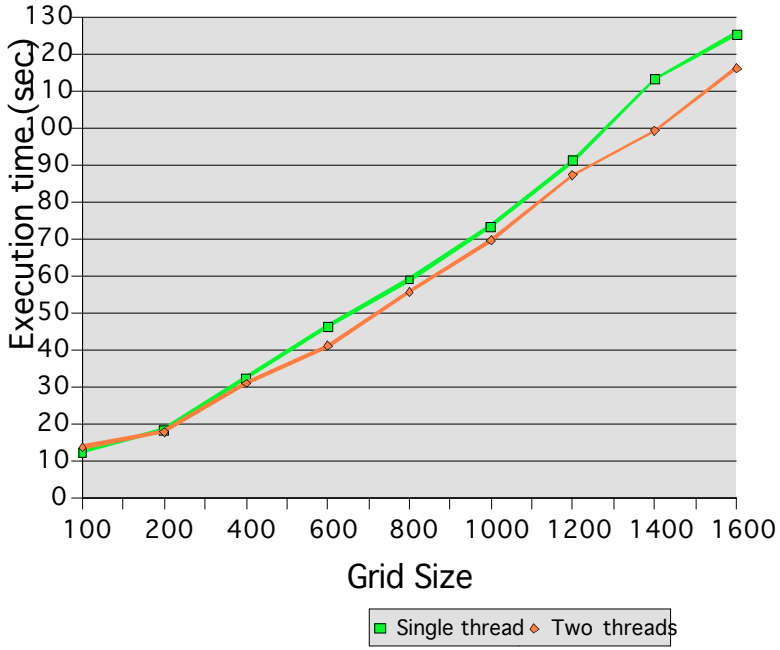
- Simulations were run on a Linux cluster, 4 PC with 650 MHz dual CPU, RedHat Linux 8.0
- 500 time steps and 1500 time steps with comm



Experience Results (cont.)

- Left figure: 500 time step, 2 nodes with Server VM and Client VM, no communication
- Right figure: two threads with communication, Server VM





Conclusion

- Multi-threading on the dual processor PC with Linux OS does not speed up
- MPJ, speed up offset by the communication and the maintenance of the list and the grids
- When the time step increase, the speed up increase
- GCJ compiler to native code
- Code clean up, cache the result