

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

Xiaorong Xiang

Gregory Madey

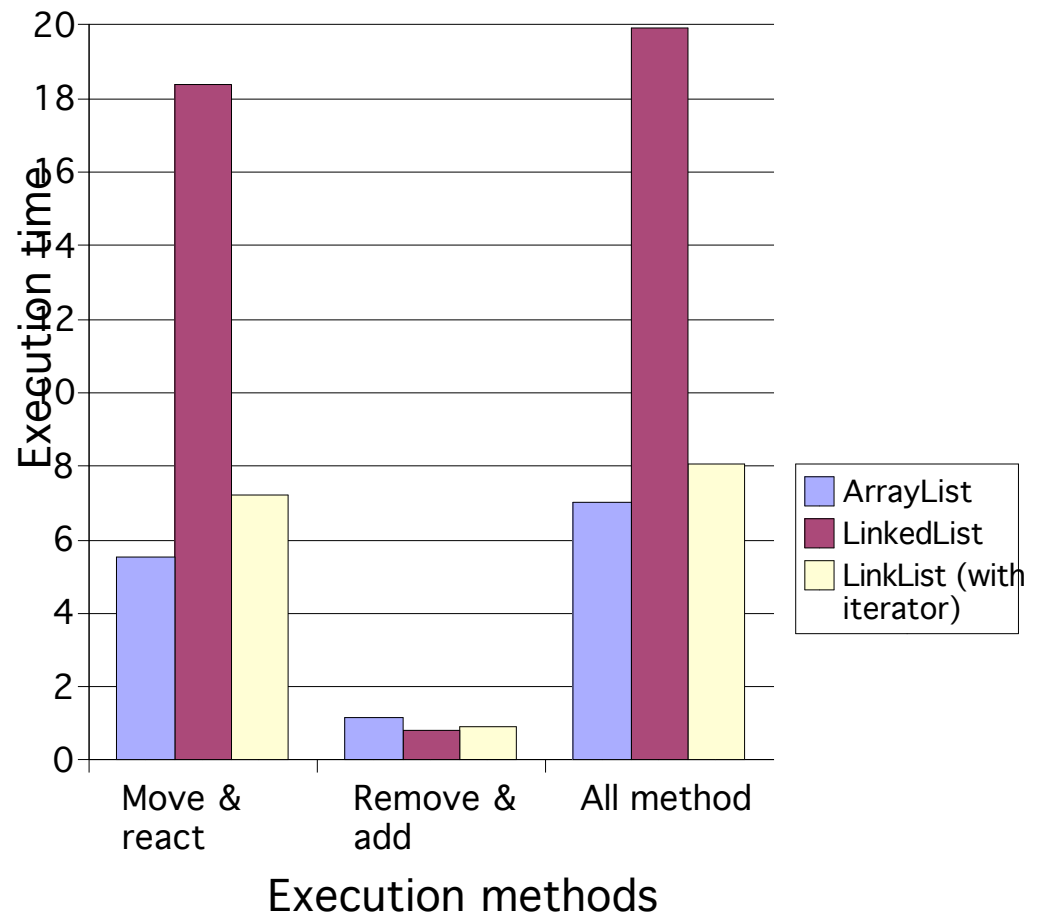
University of Notre Dame

# 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

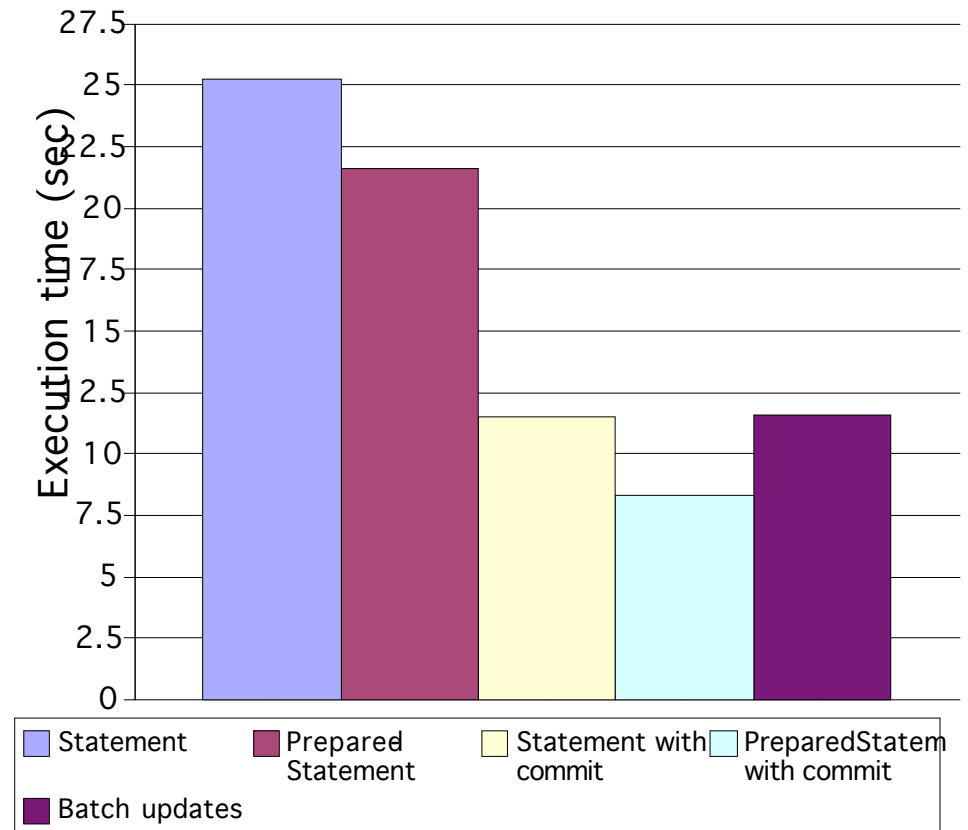


# Objects reuse

- \_ 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)

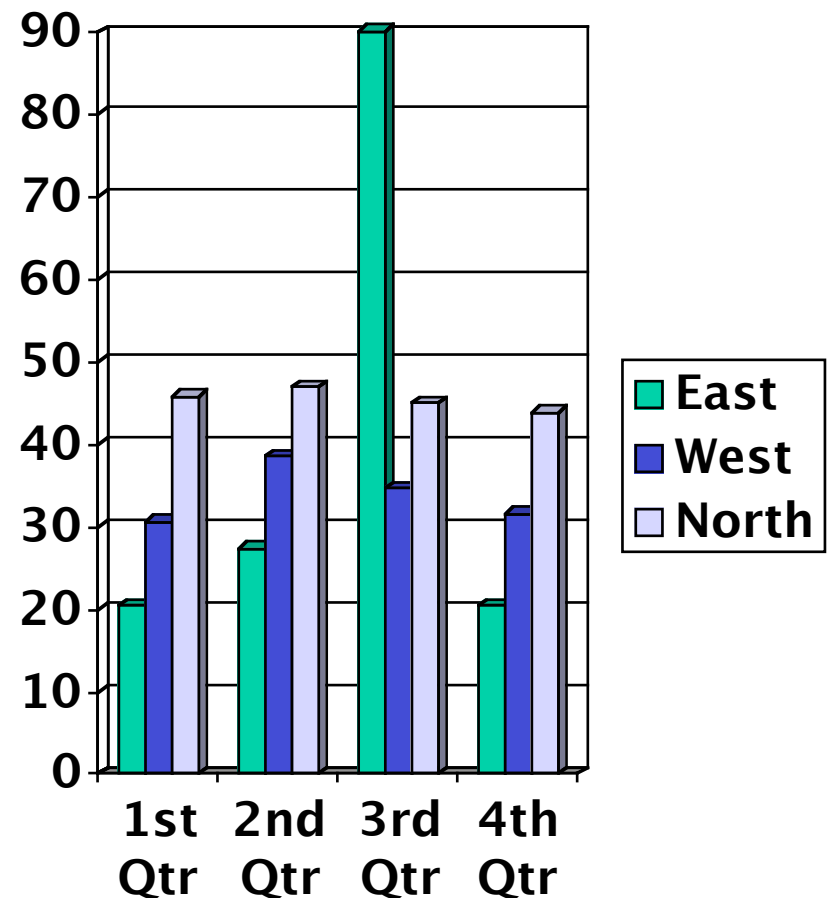
# JDBC with Data Insertion

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



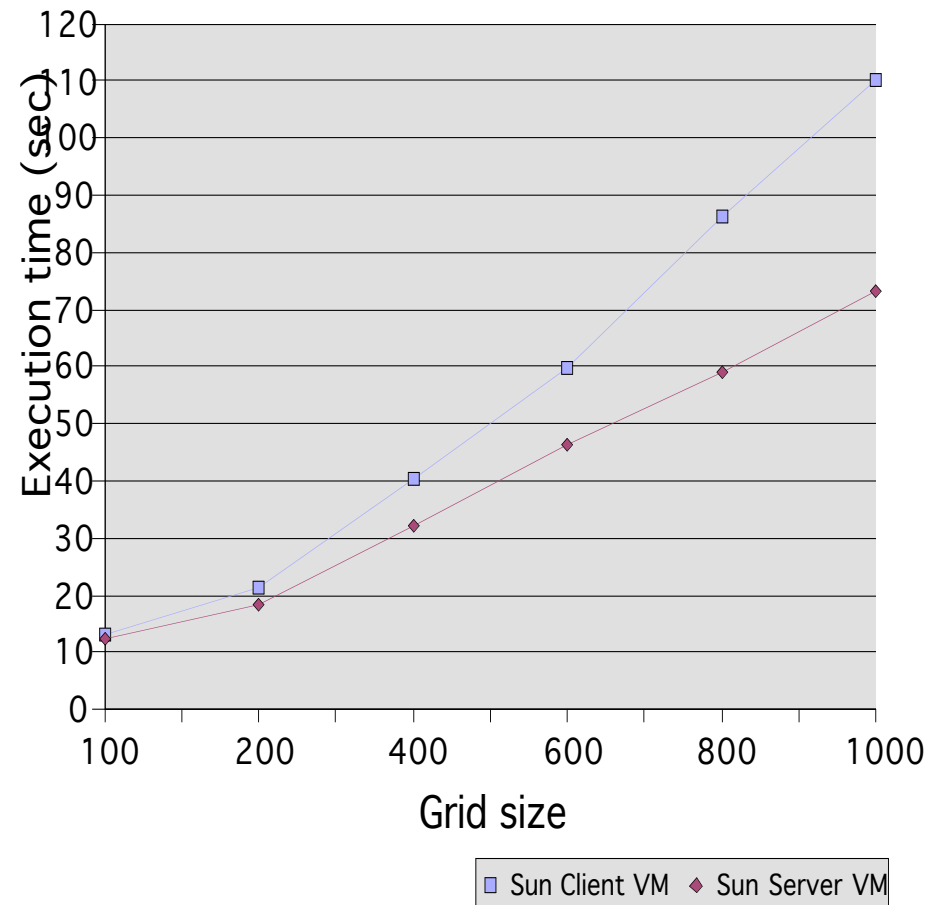
# Data output using multi-threading

- 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

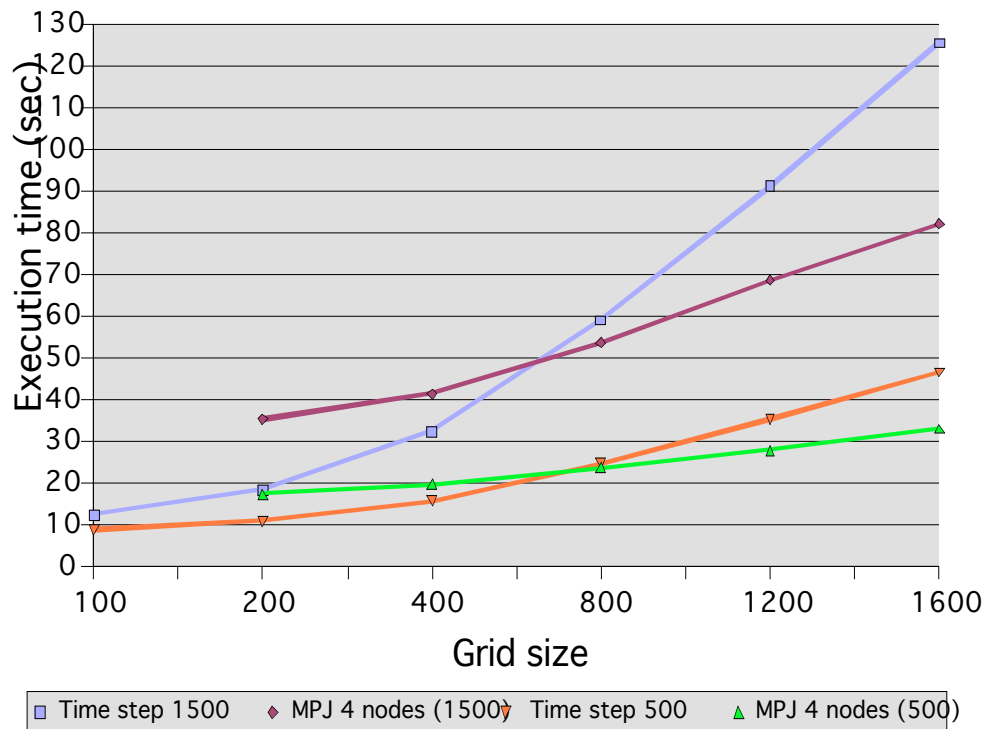


# Scalability

- \_ 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.

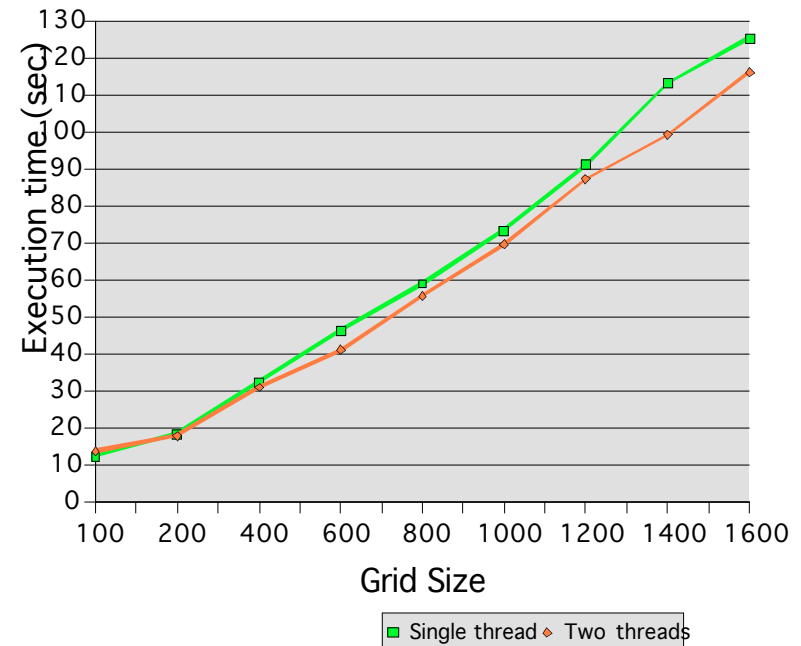
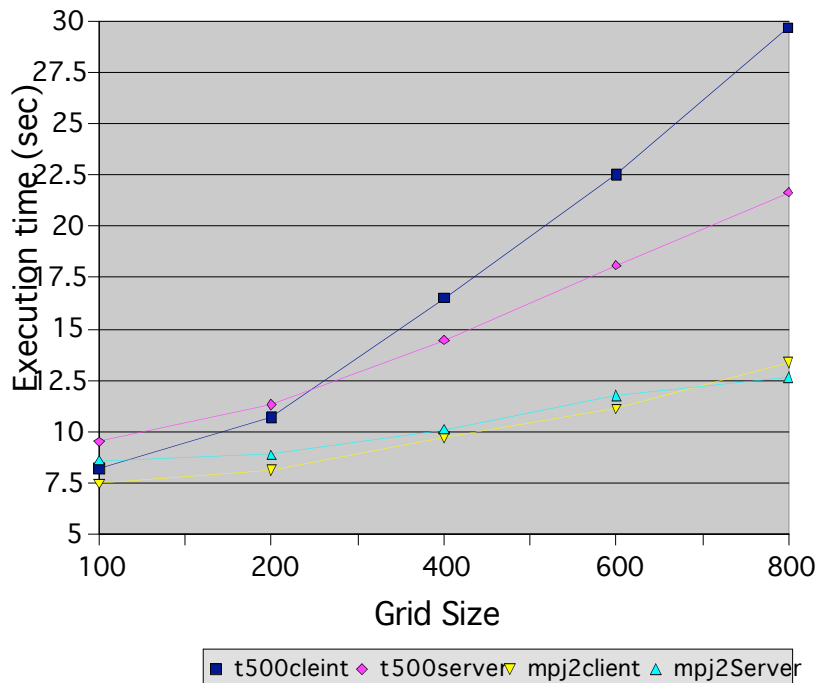
# Experiment Results

- \_ 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