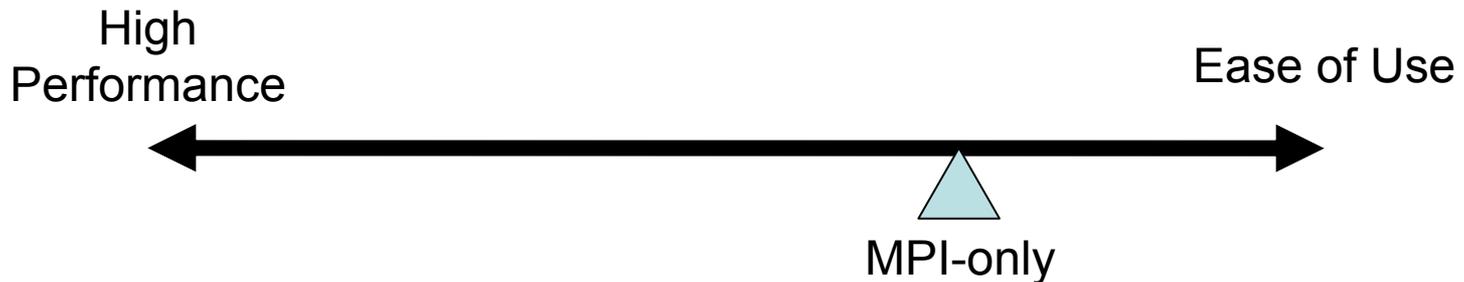


# Panel on Non-MPI Applications

Edmond Chow  
D. E. Shaw Research

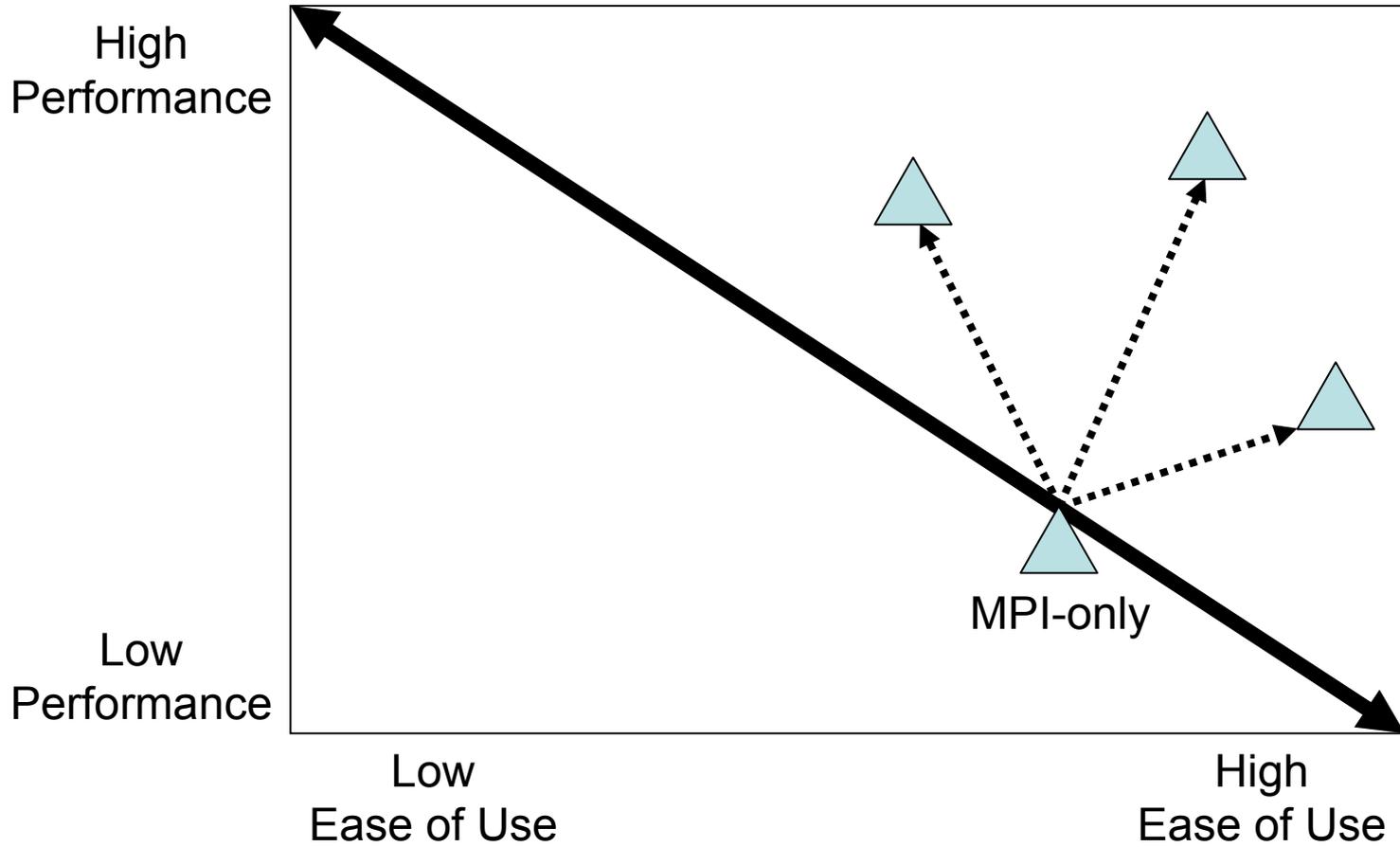
Sandia Scalable Applications Workshop  
June 3-5, 2008

# “When will MPI-only performance be inadequate” (on commodity clusters)?



- We have one main application
- Performance is valued above ease of use, cost, and portability
- We are focused on strong (fixed-size) scaling
- We (mostly) don't use MPI

# Performance vs. Ease of Use

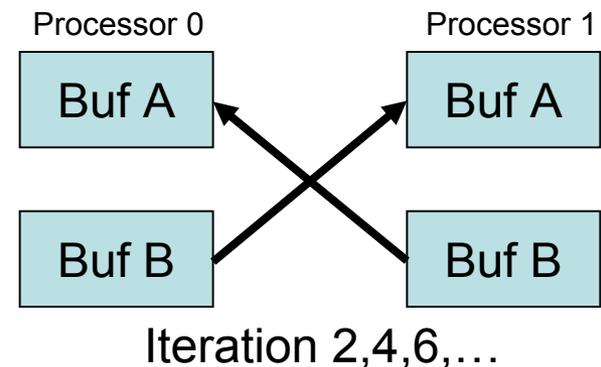
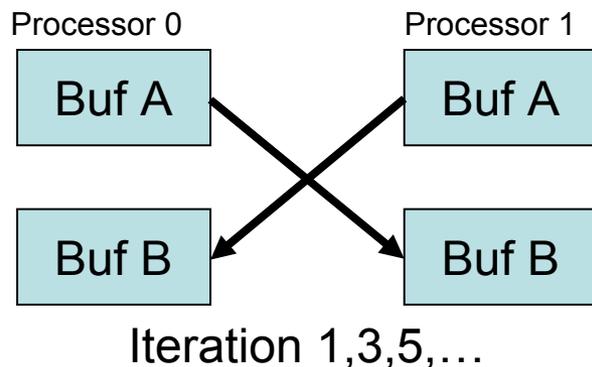


# Why we (mostly) don't use MPI

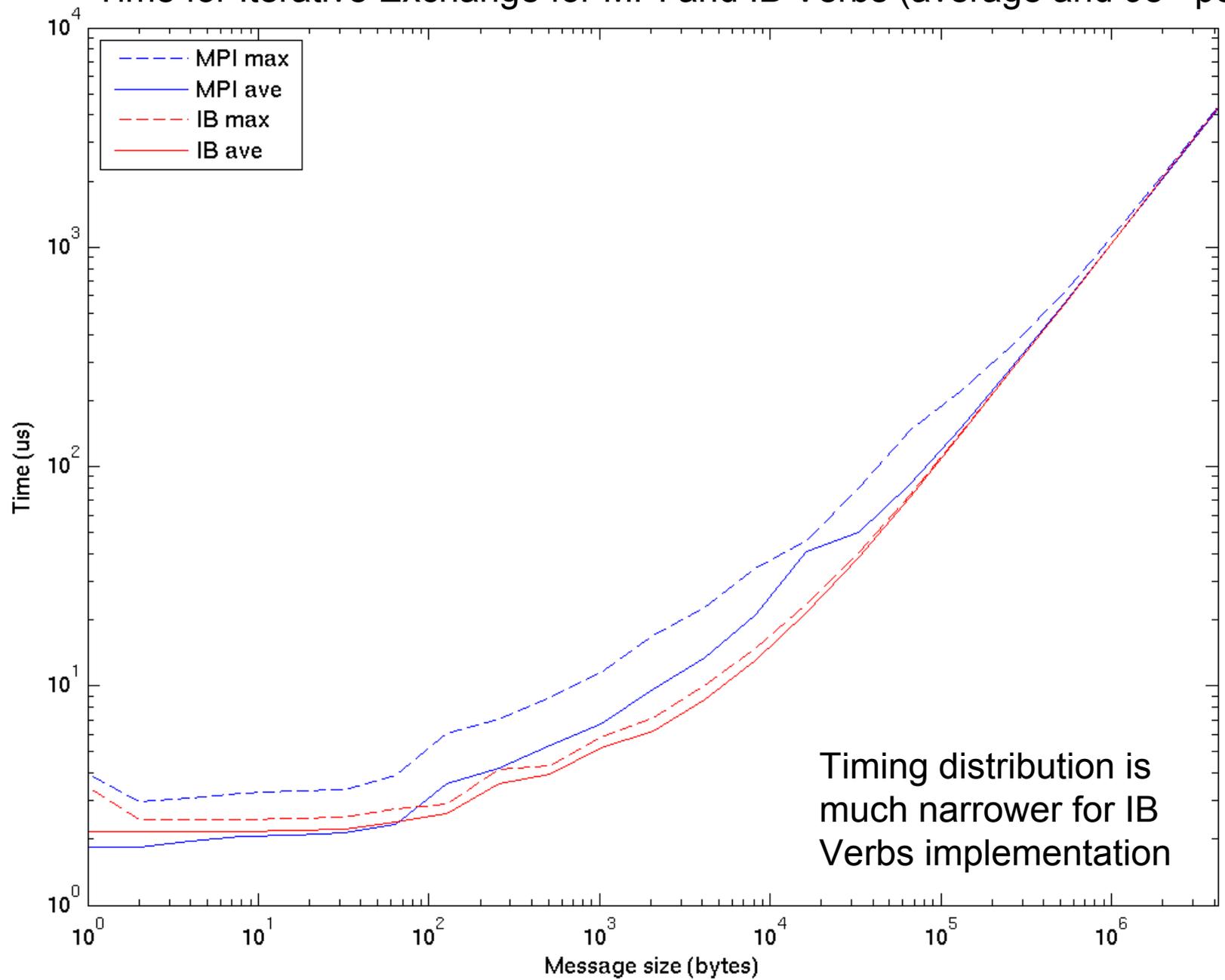
- MPI is general and easy to use but its semantics can limit both the library developer and user
- Use communication primitives that better suit the application and the hardware
- Considerations for multicore if you do use MPI

# Message passing for iterative exchange communication patterns

- always use RDMA eager sends with no memory copies
- no synchronization to make sure a receive buffer is available
- no memory registrations at send-time
- light-weight (transparent) implementation using IB Verbs and mmap (cannot implement with MPI ready-send or MPI-2 one-sided communication)



Time for Iterative Exchange for MPI and IB Verbs (average and 98<sup>th</sup> percentile)

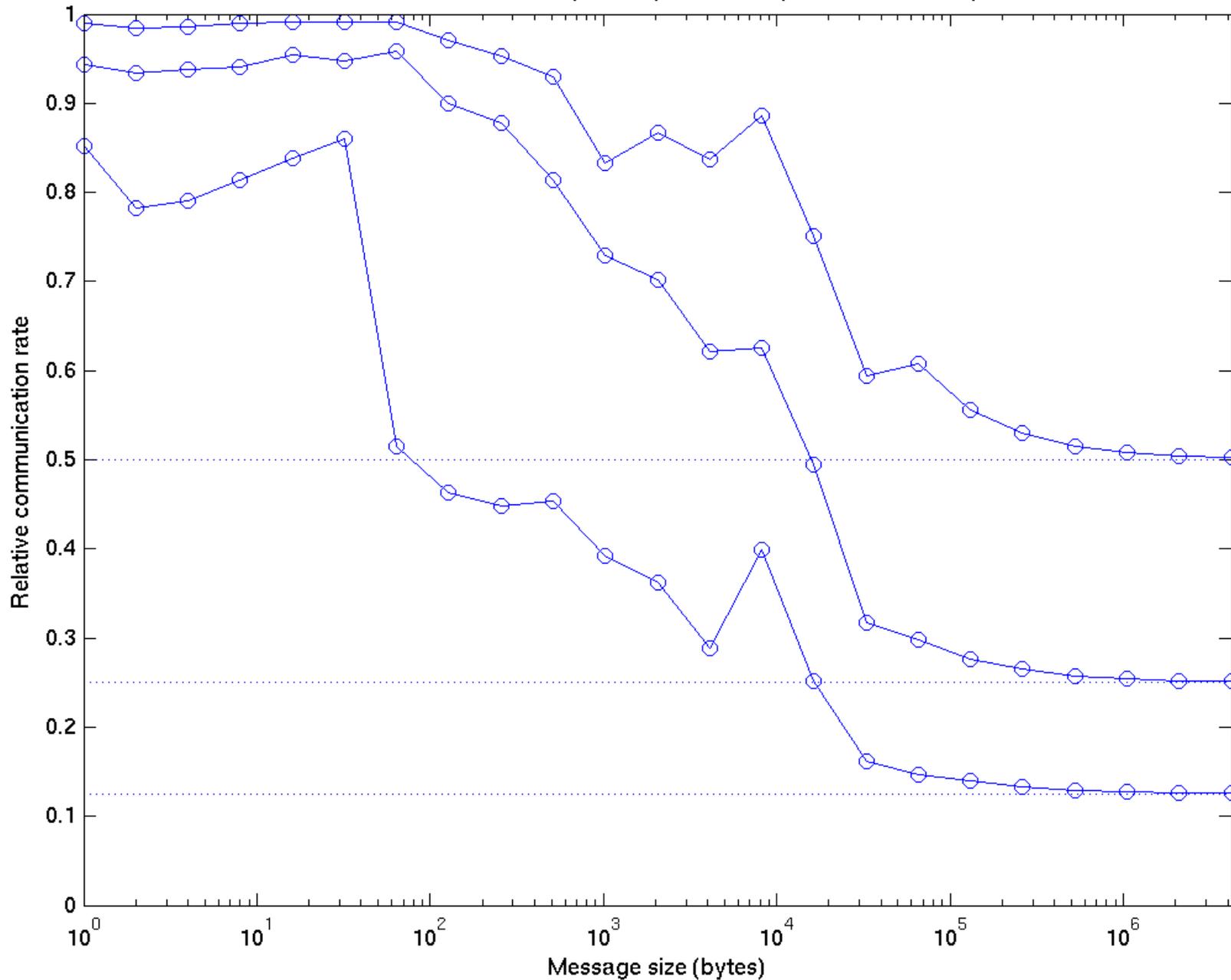


Timing distribution is much narrower for IB Verbs implementation

# How to adapt to multicore architectures?

- How do cores efficiently share cache?
  - App on a single multicore node (many options here)
- More cores per node mean more cores share a single network interface
  - App on a multicore distributed system
  - Some communication-bound applications can expect a slowdown on new hardware

Communication rate for 2 pairs, 4 pairs, and 8 pairs, relative to 1 pair

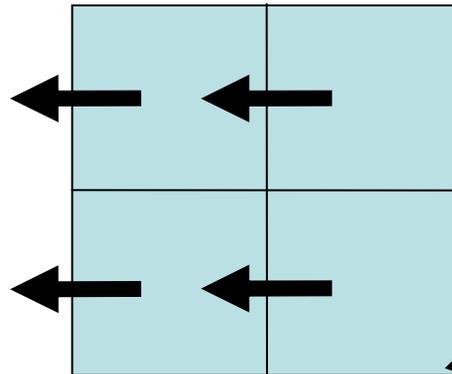


# Multicore: Avoid having all cores communicate off-node simultaneously

- Hybrid MPI-Threads
  - Many non-transparent issues that limit speedups, including serialization of communication
  - Must be aware of which cores share sockets and which cores share cache
    - lock threads to cores (`sched_setaffinity`)
    - assign related threads to cores that share cache (Linux: `/sys/devices/system/cpu`)
    - 30-50% improvement in application, but MPI-only still gives better or comparable performance
  - Profitable when there is an algorithmic advantage to using these two levels of parallelism
  - Is any better performance possible from new hybrid programming models? Cores cooperating on data in cache in pipelined fashion?

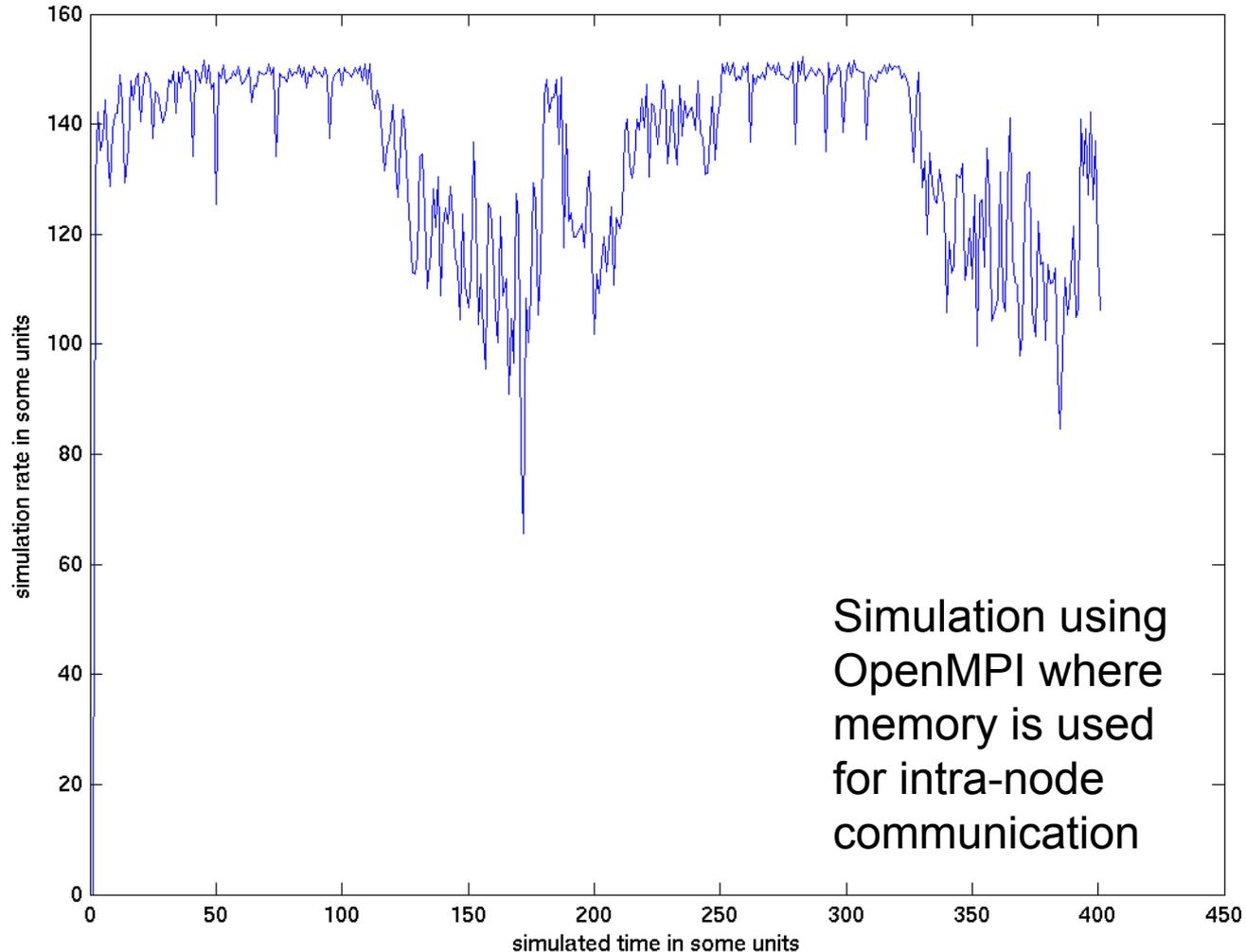
# Multicore: Avoid having all cores communicate off-node simultaneously

- Develop new or rediscover algorithms that can stage their communications so that not all cores on a node need to communicate simultaneously
  - one code runs two similar simulations simultaneously
- Organize communication so that off-node communication is overlapped with on-node communication



# More transparent systems and tools will help users optimize performance

(for users willing to work for it)



# Future Non-MPI Apps?

- On commodity clusters
- On specialized hardware (incl. Cell, GPUs)
  - Specialized hardware for molecular dynamics
    - Heterogeneous (many types of cores on a chip)
    - Four main communication mechanisms between different types of cores and memory
    - Main mechanism is a DMA engine between SRAM on a Tensilica core and all other computational cores on all nodes
  - General purpose hardware with no caches but fast local store managed by the user/compiler; thin/few interfaces across the network stack

# Summary

- MPI vs. Specialized communication libraries
  - specialization gives up to 25% overall improvement in our application
  - we still use MPI collectives
  - MPI implementations are always getting better
- In multicore case, cores must efficiently share the network interface (and efficiently share cache)
  - need algorithms where not all cores communicate simultaneously
  - overlap off-node and on-node communication
- Transparent systems and tools?