Grid Computing With Charm++ And Adaptive MPI

Gregory A. Koenig (koenig@cs.uiuc.edu)
Department of Computer Science
University of Illinois at Urbana-Champaign

5th Annual Workshop on Charm++ and its Applications
April 18, 2007

Introduction

- *Metacomputer* — A network of heterogeneous, computational resources linked by software in such a way that they can be used as easily as a single computer

  [Smarr, Catlett - *CACM*, June 1992]

- This idea was further developed as "Grid Computing" by Foster & Kesselman (and many others) in the mid-1990's and later

  [Foster, Kesselman - *The Grid: Blueprint for a New Computing Infrastructure*, 1998 (1st Edition), 2004 (2nd Edition)]


NSF TeraGrid
Extensible Terascale Facility
Argonne
Purdue
PSC
NCSA
IU
ORNL
Caltech
SDSC
TACC

Example Grid Computing Applications

- NEKTAR (George Karniadakis, Brown University)
  - Simulation of blood flow in the human arterial tree (fluid dynamics)
- SPICE, Simulated Pore Interactive Computing Environment (Peter Coveney, University of London)
  - Translocation of nucleic acids across membrane channel pores in biological cells
- VORTONICS (Bruce Boghosian, Tufts University)
  - Vortex dynamics (3D Navier-Stokes computations)

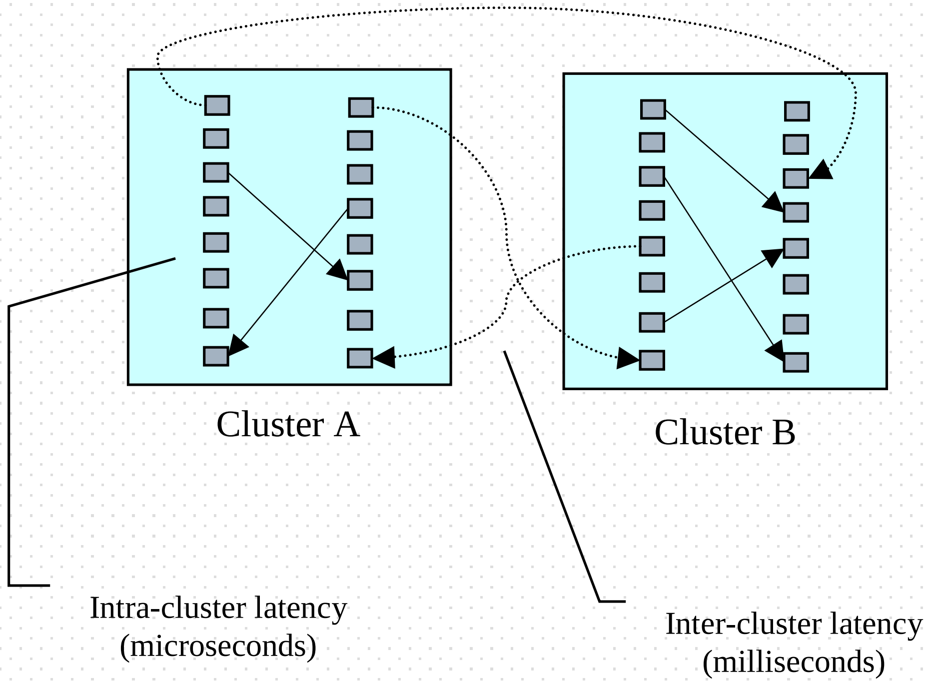
Goals of this Project

- Good performance when executing **tightly-coupled** parallel applications in Grid metacomputing environments

- Require minimal or no changes to the parallel applications themselves
  - This implies that techniques must be developed at the runtime system (middleware) layer

Challenges

- Need for efficient mapping of work to resources
- Grids are a dynamic environment
- Grids involve pervasive heterogeneity
- Cost of cross-site communication (i.e., cross-site latency)



Charm++ and Adaptive MPI

- Charm++ is a parallel implementation of the C++ programming language complemented by an adaptive runtime system
- A programmer decomposes a program into parallel message-driven objects (called *chares)*
- The adaptive runtime system maps (and re-maps) objects onto physical processors; a message-driven scheduler on each processor drives the execution of the objects mapped to the same physical processor; each processor typically holds many (tens or hundreds) of objects
- Adaptive MPI (AMPI) brings the features of the Charm++ runtime system to more traditional MPI applications

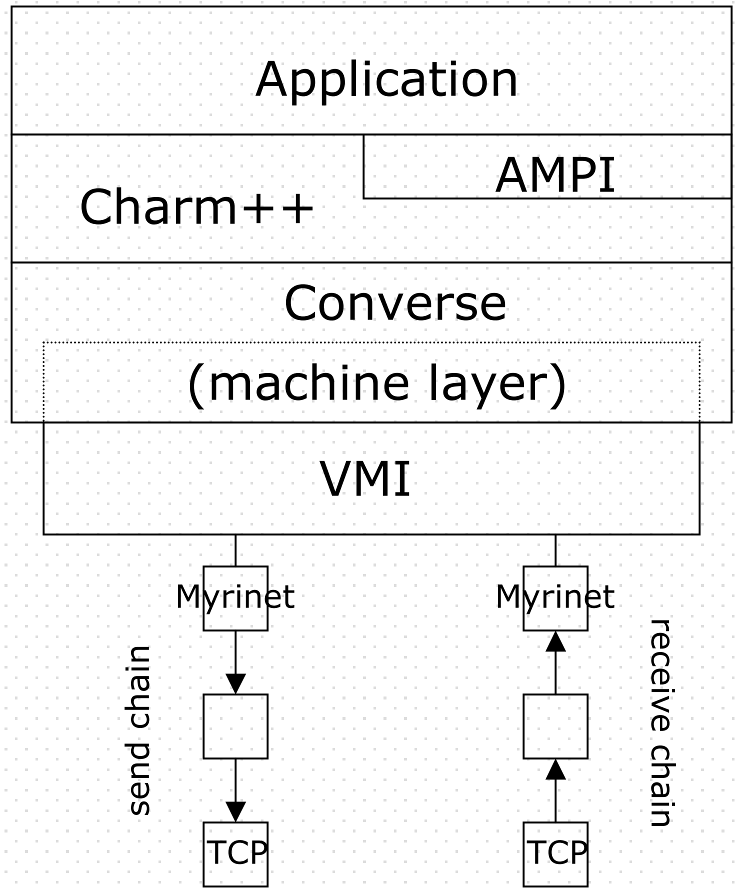
Virtual Machine Interface (VMI)

- VMI is an event-driven messaging layer that provides an abstraction above lower-level layers such as Myrinet, InfiniBand, or Ethernet

- VMI Goals
  - Application portability across interconnects
  - Data striping and automatic failover
  - Support for Grid-computing applications
  - Dynamic monitoring and management

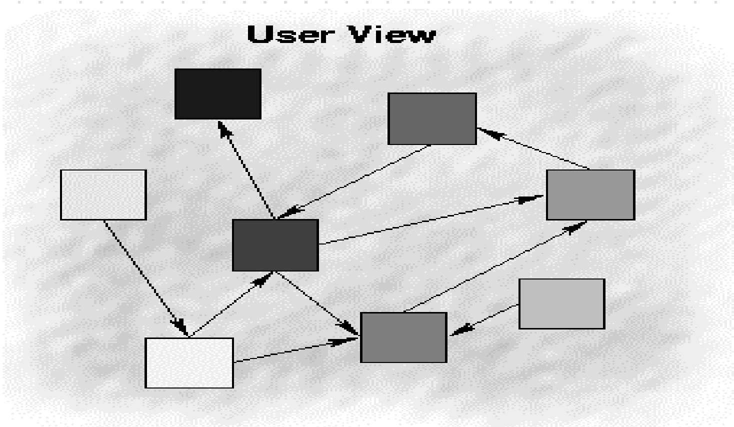
Implementation of Charm++ on Virtual Machine Interface (VMI)

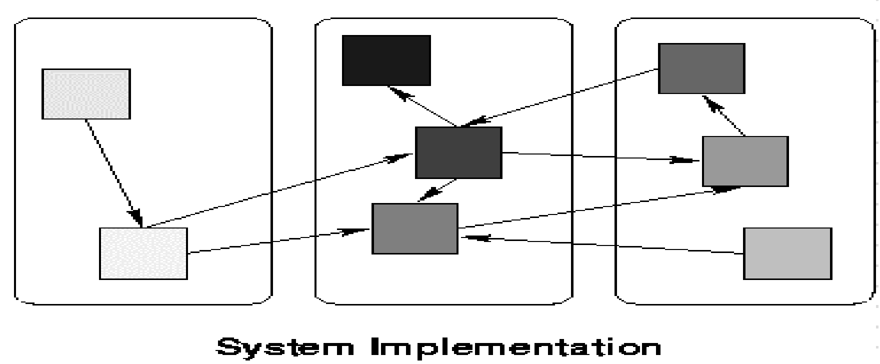
- Message data are passed along VMI "send chain" and "receive chain"
- Devices on each chain may deliver data directly, manipulate data, or pass data to next device



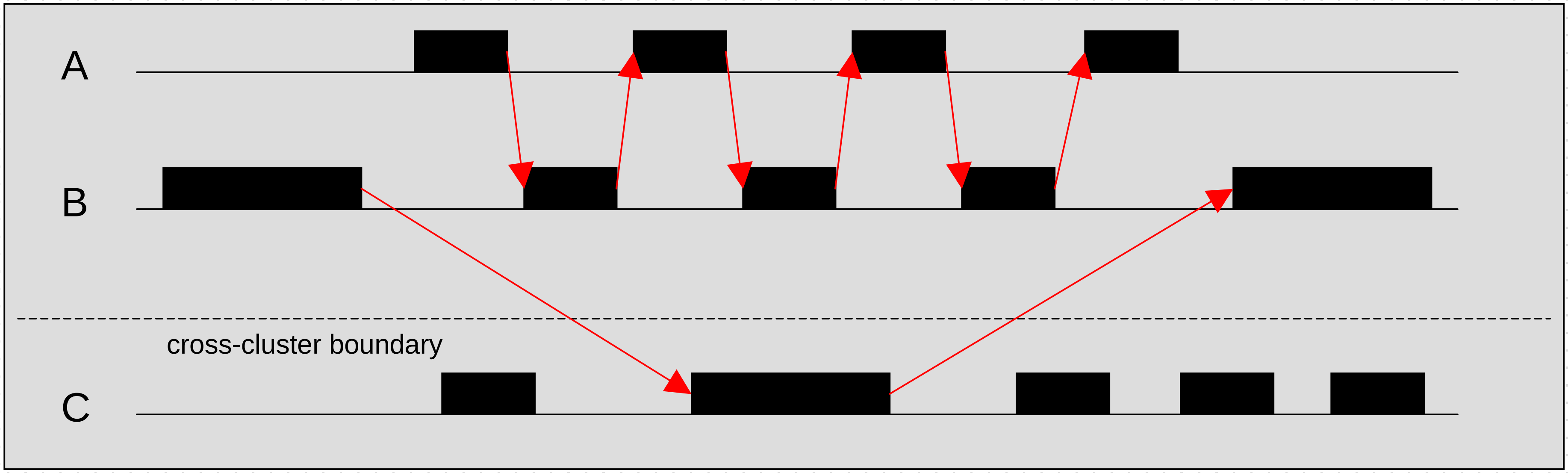
The Charm++ Approach to Grid Computing

- Leverage the use of message-driven objects in the Charm++ runtime system to mask latency
- Each processor holds a small number of remotely-driven objects and a much larger number of locally-driven objects; overlap the latency of remote communication with locally-driven work





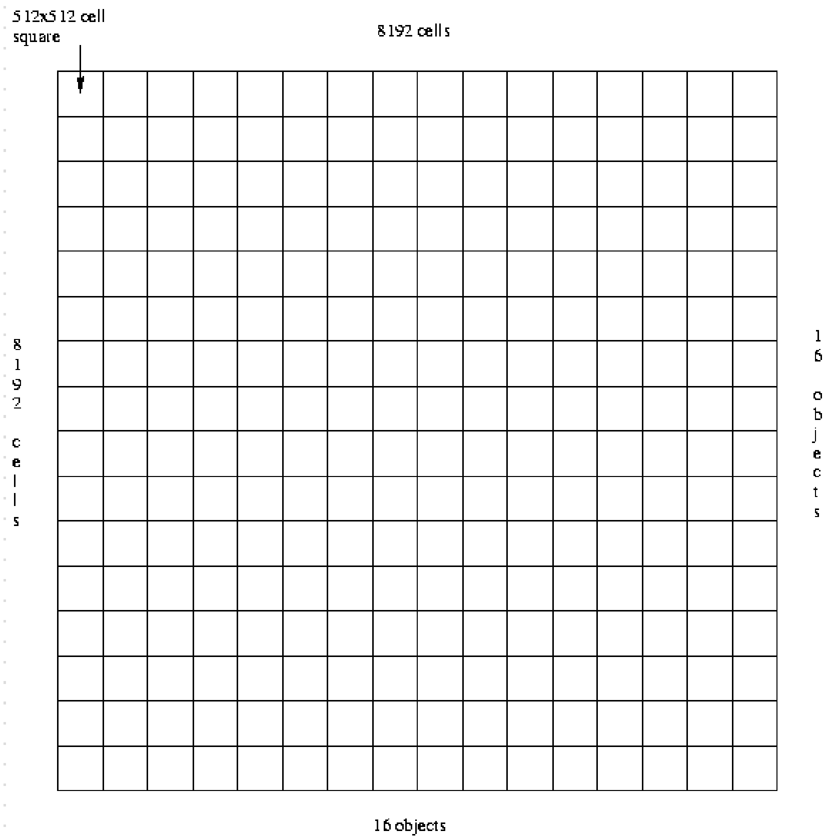
Hypothetical Timeline View of a Multi-Cluster Computation



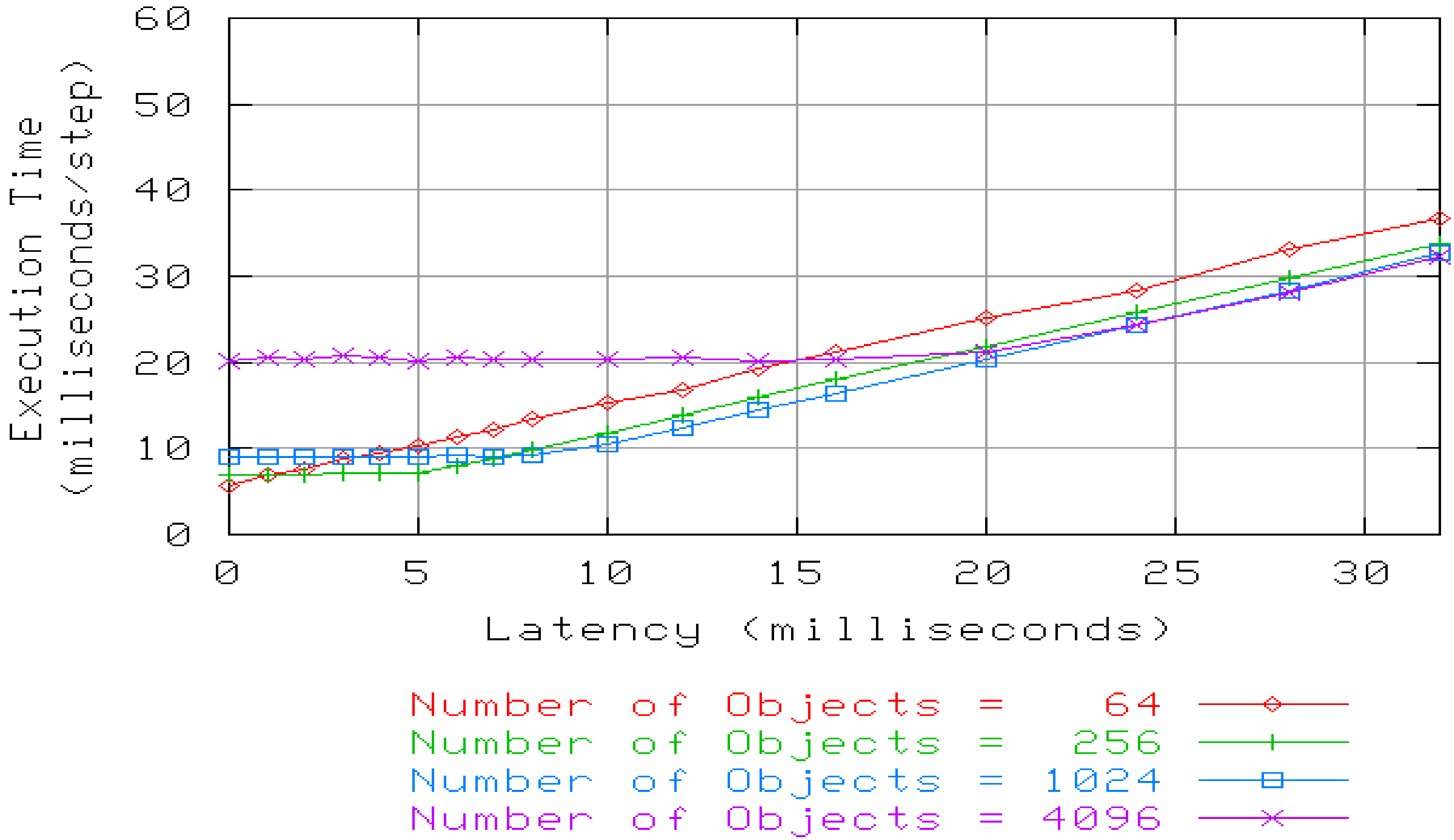
- Processors A and B are on one cluster, Processor C on a second cluster
- Communication between clusters via high-latency WAN
- Work driven by “local objects” allows latency masking

Five-Point Stencil (Jacobi2D)

- Simple finite difference method considering neighbors above, below, left, right
- **Problem size is fixed** (2048x2048 or 8192x8192)
- **Problem is evenly divided between two clusters** (e.g., 32 processors means 16 processors in Cluster A and 16 processors in Cluster B)
- **Number of objects used to decompose problem varies** (allowing the effects of varying the number of objects to be studied)



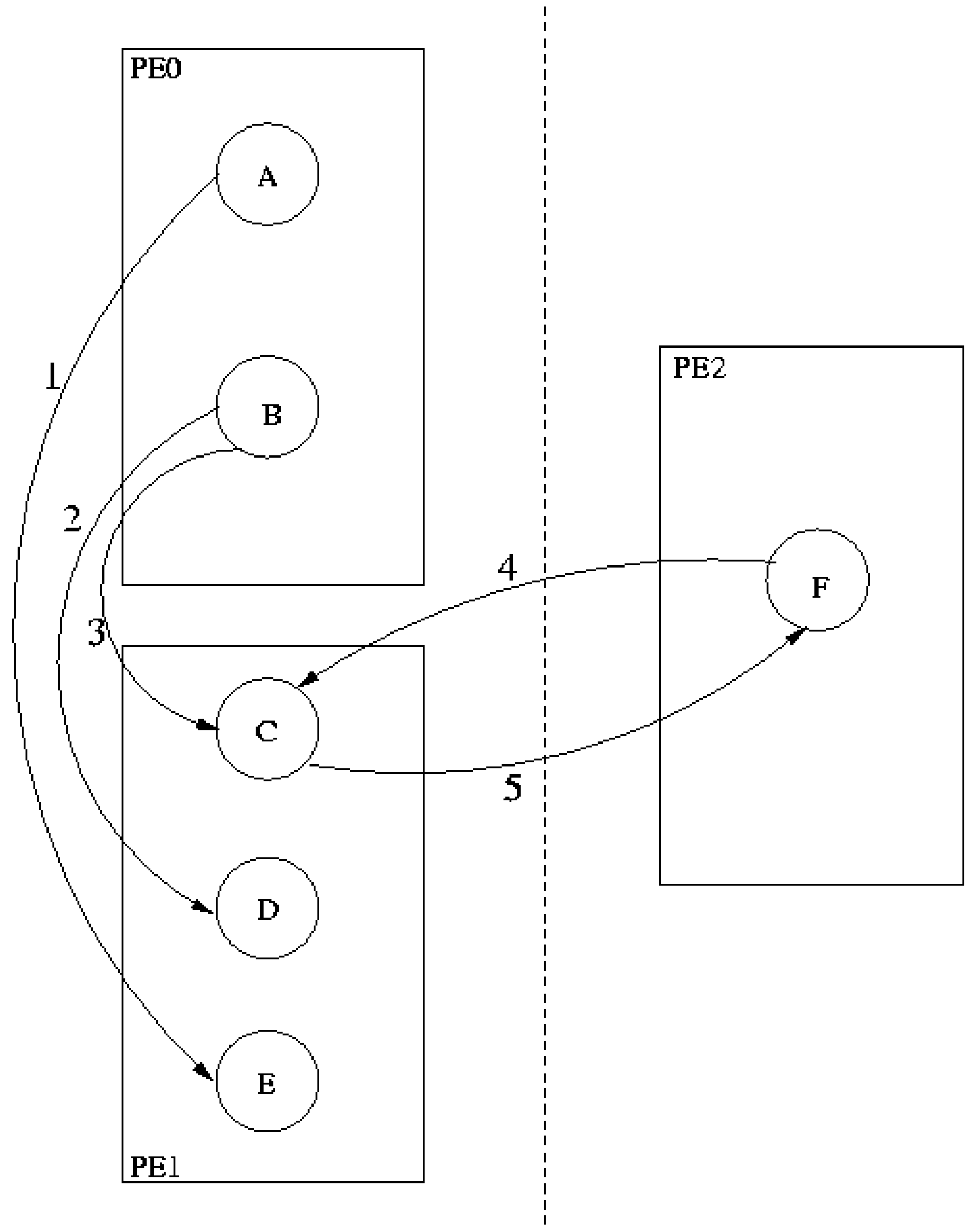
Five-Point Stencil Performance (2048x2048 mesh, 32 Processors)



Object Prioritization

- Latency masking via message-driven objects works by overlapping the communication in border objects with work in local-only objects
- Optimization — Prioritize the border objects to give maximum chance for overlapping cross-site communication with locally-driven work
- Implementation
  - Any time an object sends a message that crosses a cluster boundary, record that object's ID in a table of border objects on the processor
  - Any incoming messages to the processor are checked to determine the destination object ID
    - Destined for local-only object, place in Scheduler Queue
    - Destined for border object, place in high-priority Grid Queue

- Prioritization Example
  - 2 Clusters
  - 3 Processors
  - 6 Objects
- On PE1, Object C is a border object, Objects D and E are local-only objects
- Incoming Messages 1, 2, and 3 to PE1 are examined
- Messages 1 and 2, destined for local-only objects are placed in Scheduler Queue
- Message 3, destined for Object C is placed in high-priority Grid Queue



Grid Topology-Aware Load Balancing

- Charm++ Load Balancing Framework measures characteristics of objects in a running application (e.g., CPU load, number of messages sent)
- Load balancing can greatly improve performance of traditional parallel applications because many applications are dynamic (change as they run)
- In a Grid metacomputing environment, *characteristics of the environment can change too*
- **Couple measured application characteristics with knowledge of the Grid environment to make better object mapping decisions**

Basic Communication Load Balancing (GridCommLB)

- Strategy — Use a greedy algorithm to evenly distribute the border objects over the processors in each cluster
- Does not consider relationship between objects (communication volume internal to each cluster can increase)
- Objects never migrate across cluster boundary (i.e., they stay inside the cluster in which they were originally mapped)
- Must also take into consideration the measured CPU load of each object to avoid overloading processors

Graph Partitioning Load Balancing (GridMetisLB)

- Strategy — Partition the object communication graph (using Metis [Karypis,Kumar - 1995]) to attempt to reduce the amount of cross-cluster communication
- Objects that communicate frequently with each other are mapped to be "close" to each other (same cluster or same processor)
- Two-phase algorithm
  - Phase 1 — Partition objects onto clusters by using Metis to find a "good" cut across cluster boundaries
  - Phase 2 — In each cluster, partition objects onto processors by using Metis to find a "good" partition that balances CPU load and reduces inter-processor communication volume

Case Studies

- Applications
  - Molecular dynamics (LeanMD)
  - Finite element analysis (Fractography3D)

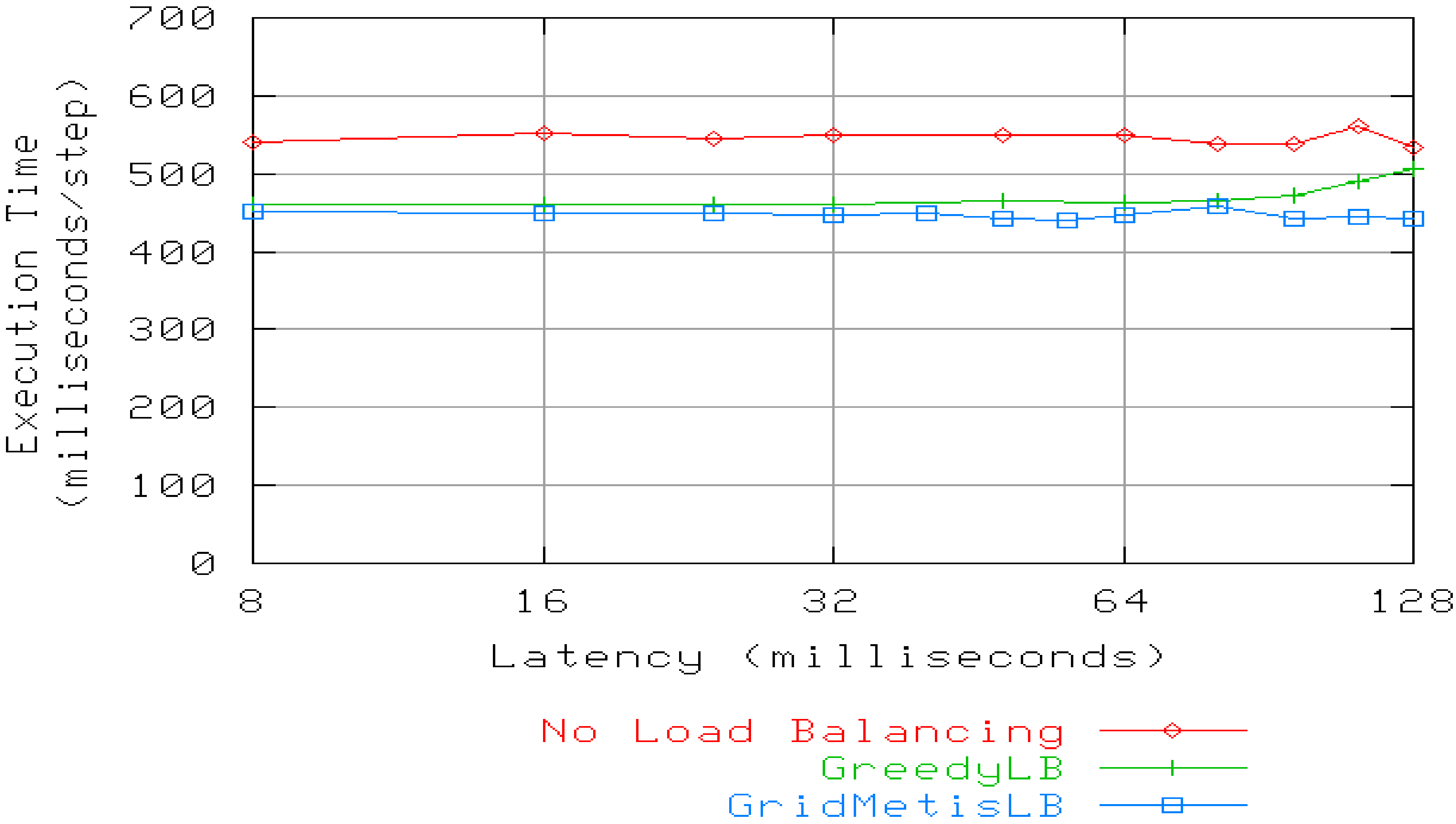
- Grid environments
  - Artificial latency environment — VMI "delay device" adds a pre-defined latency between arbitrary pairs of nodes
  - TeraGrid environment — Experiments run between NCSA and Argonne National Laboratory machines (1.7 milliseconds latency) and between NCSA and SDSC machines (30.1 milliseconds latency)

Molecular Dynamics (LeanMD)

- *Simulation box* made up of *cells*, responsible for all atoms within a given boundary; $K \times K \times K$ regions of cells are organized into *patches*
- The fundamental unit of decomposition is a cell-pair object
- 216 cells and 3024 cell pairs in the molecular system examined here

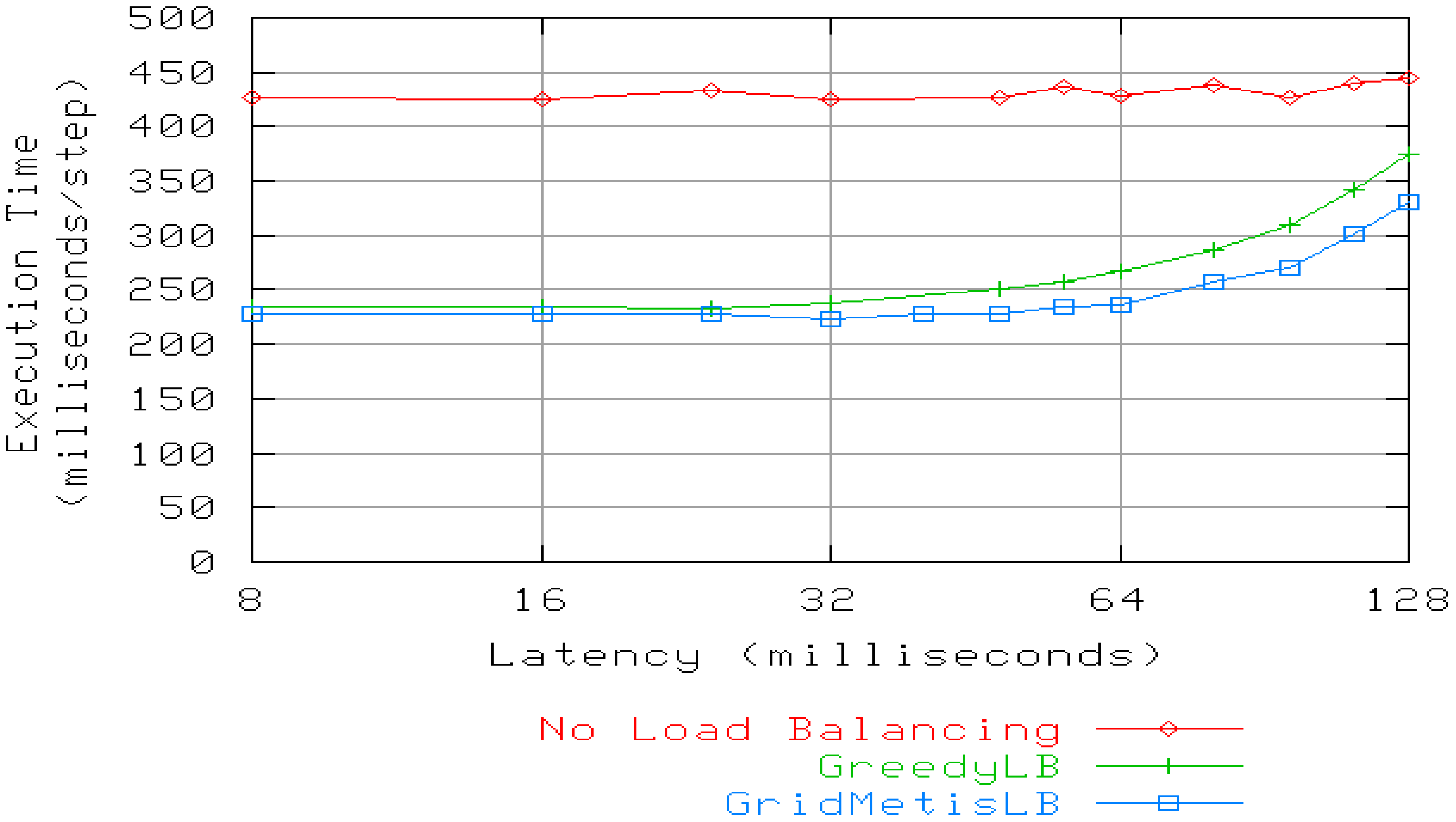
LeanMD Performance

32 Processors (16 Processors + 16 Processors)



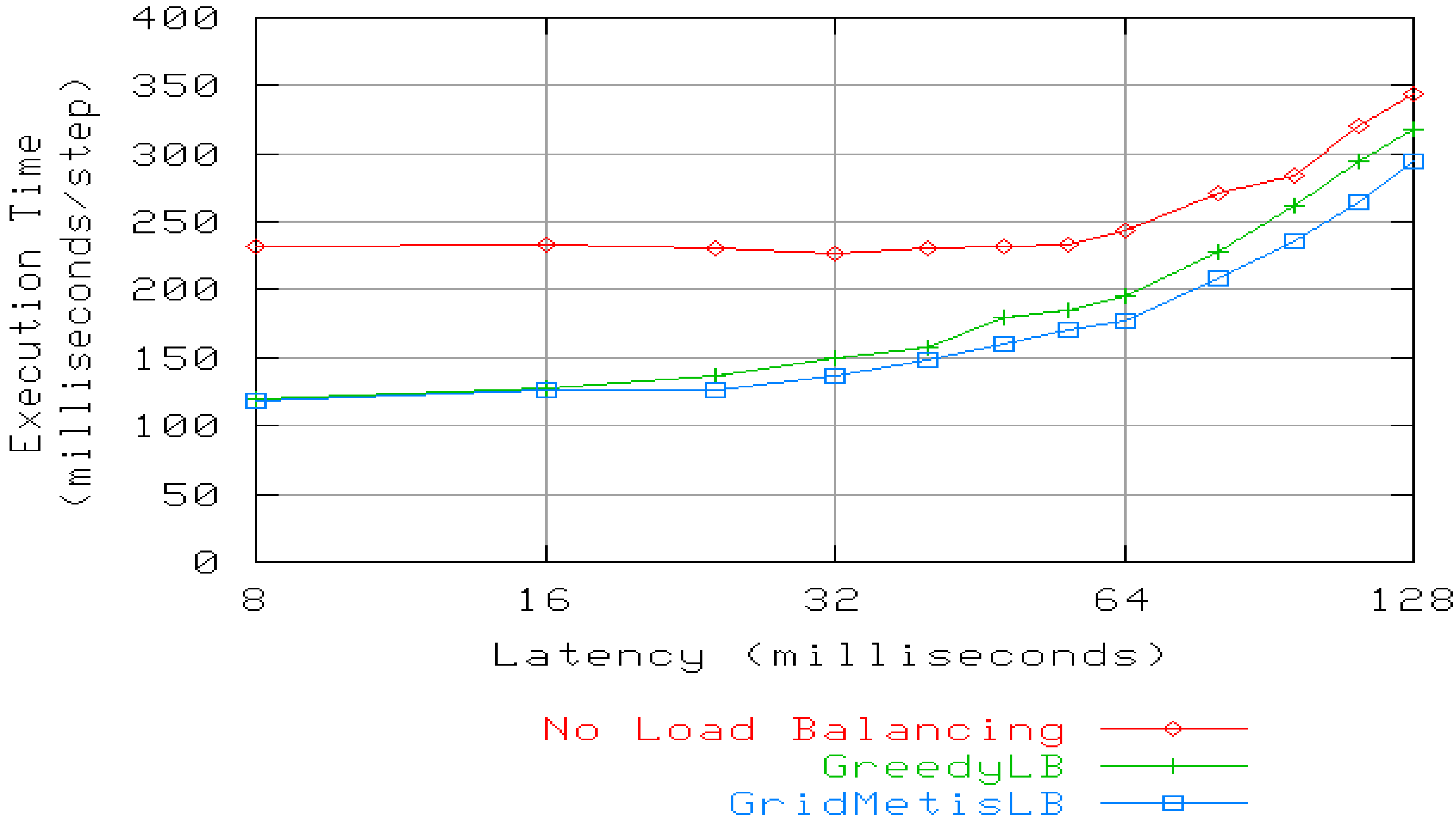
LeanMD Performance

64 Processors (32 Processors + 32 Processors)



LeanMD Performance

128 Processors (64 Processors + 64 Processors)



Conclusion

- Techniques developed at the runtime system (middleware) level **can** enable tightly-coupled applications to run efficiently in Grid metacomputing environments with few or no changes necessary to the application software
  - Latency masking with message-driven objects
  - Border object prioritization
  - Grid topology-aware load balancing

- Case studies
  - Molecular dynamics (LeanMD)
  - Finite element analysis (Fractography3D)