Adaptive MPI

Chao Huang, Orion Lawlor, L. V. Kalé

Parallel Programming Lab
Department of Computer Science
University of Illinois at Urbana-Champaign

Motivation

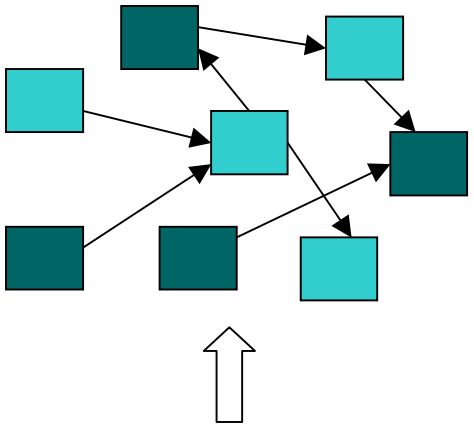
- Challenges
  - New generation parallel applications are:
    - Dynamically varying: load shifting, adaptive refinement
  - Typical MPI implementations are:
    - Not naturally suitable for dynamic applications
  - Set of available processors:
    - May not match those required by the algorithm
- Adaptive MPI
  - Virtual MPI Processor (VP)
  - Solutions and optimizations

Outline

- Motivation
- Implementations
- Features and Experiments
- Current Status
- Future Work

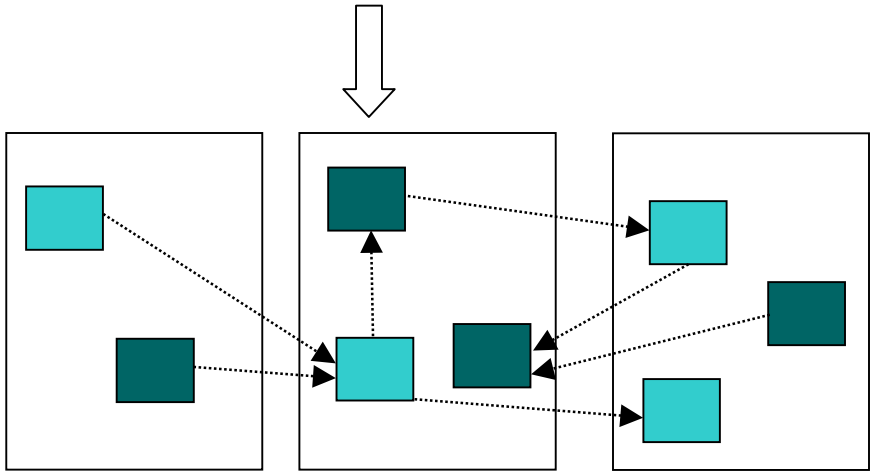
Processor Virtualization

- Basic idea of processor virtualization
  - User specifies interaction between objects (VP's)
  - RTS maps VP's onto physical processors
  - Typically, # virtual processors > # processors



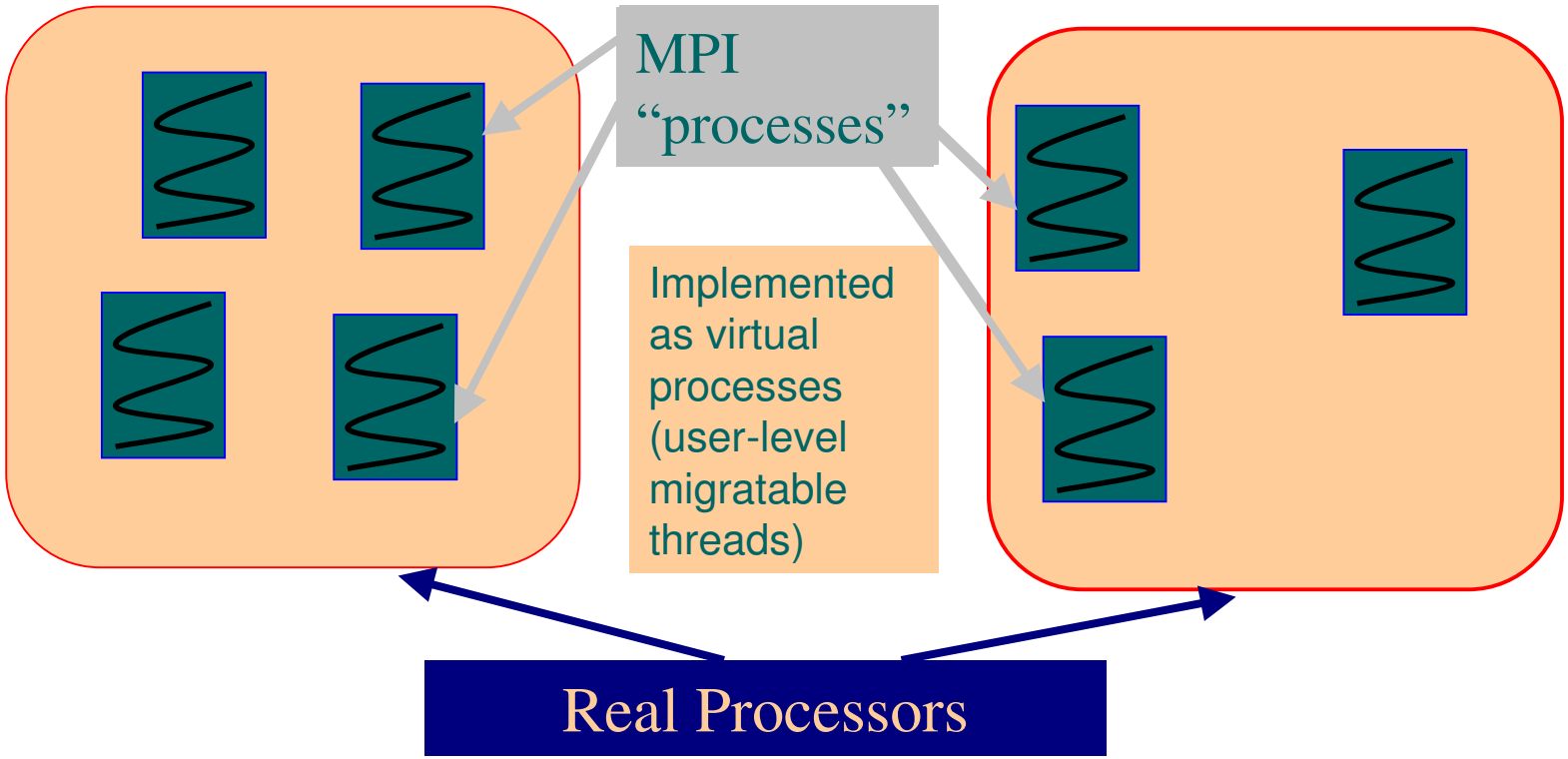
User View

System implementation



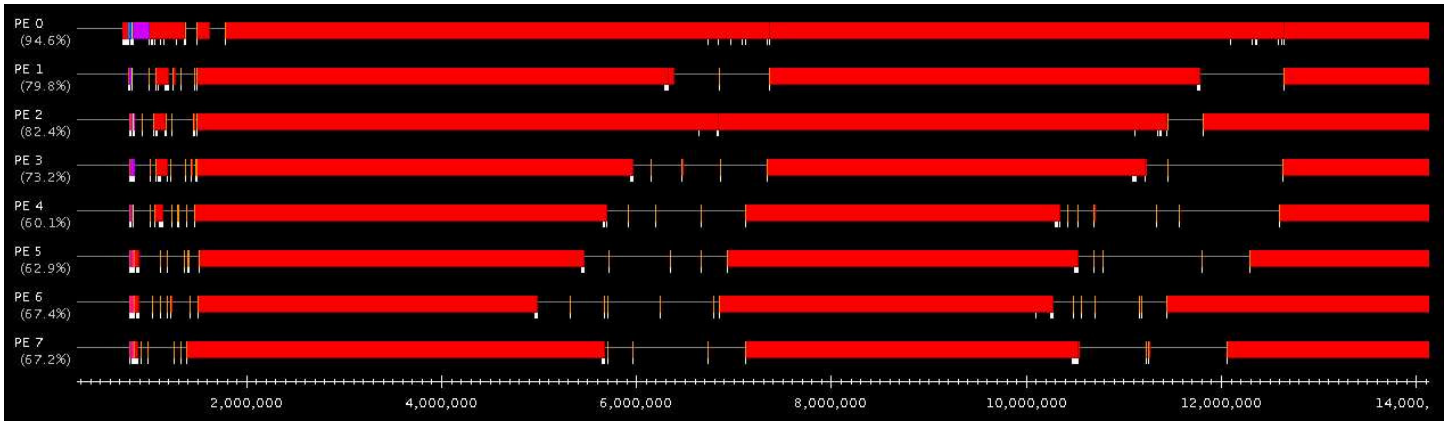
AMPI: MPI with Virtualization

- Each virtual process implemented as a user-level *thread* embedded in a Charm++ *object*

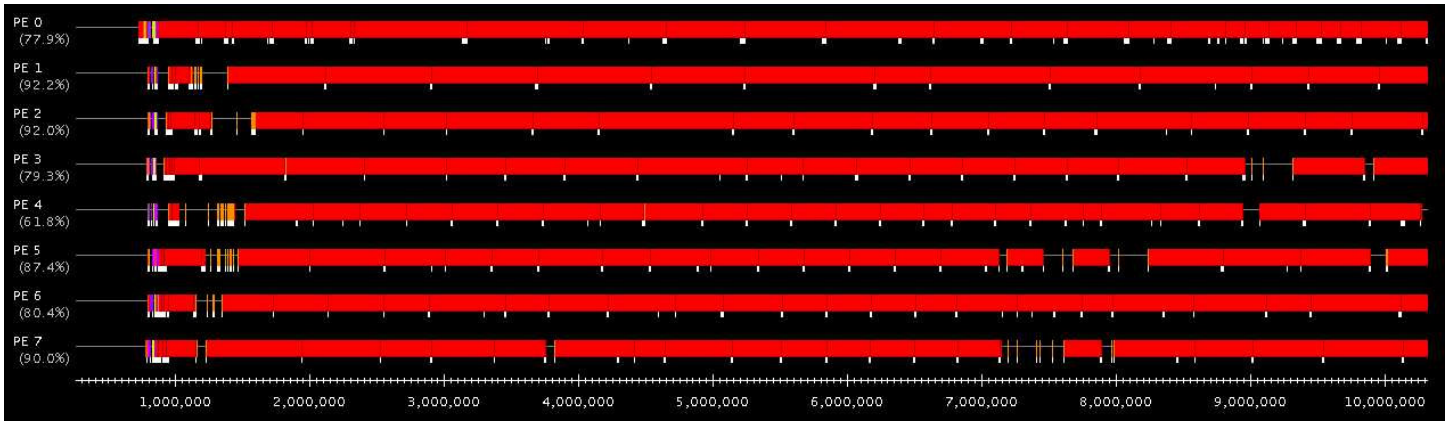


Adaptive Overlap

p = 8
vp = 8

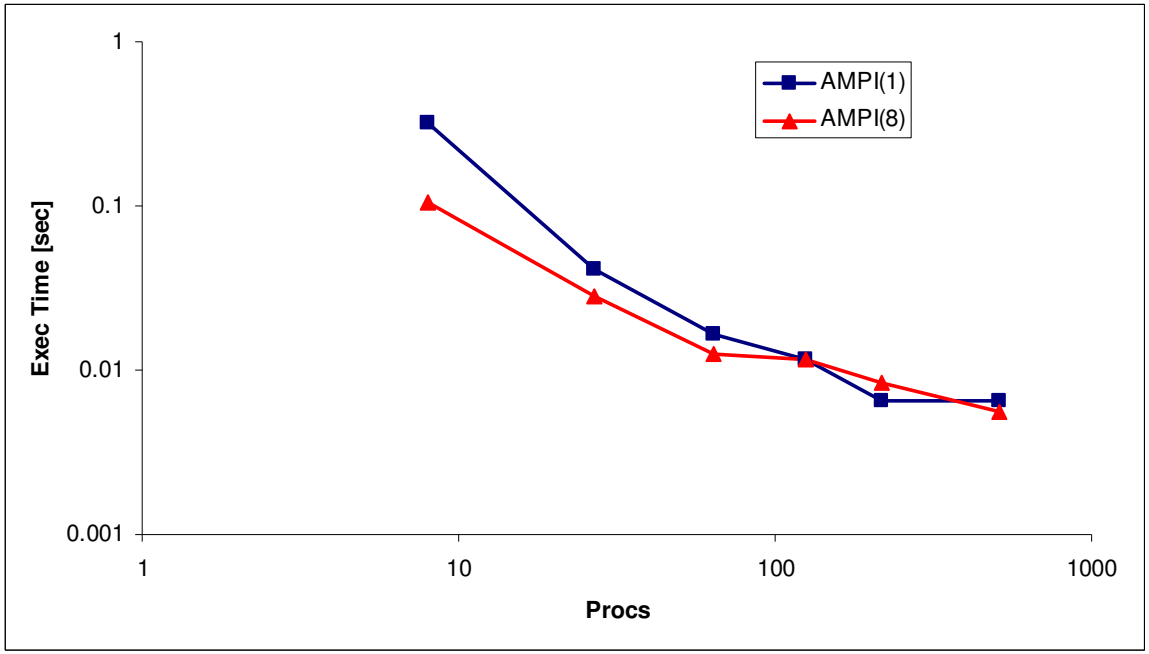


p = 8
vp = 64



Problem setup: 3D stencil calculation of size 240^3 run on Lemieux.
Run with virtualization ratio 1 and 8.

Benefit of Adaptive Overlap

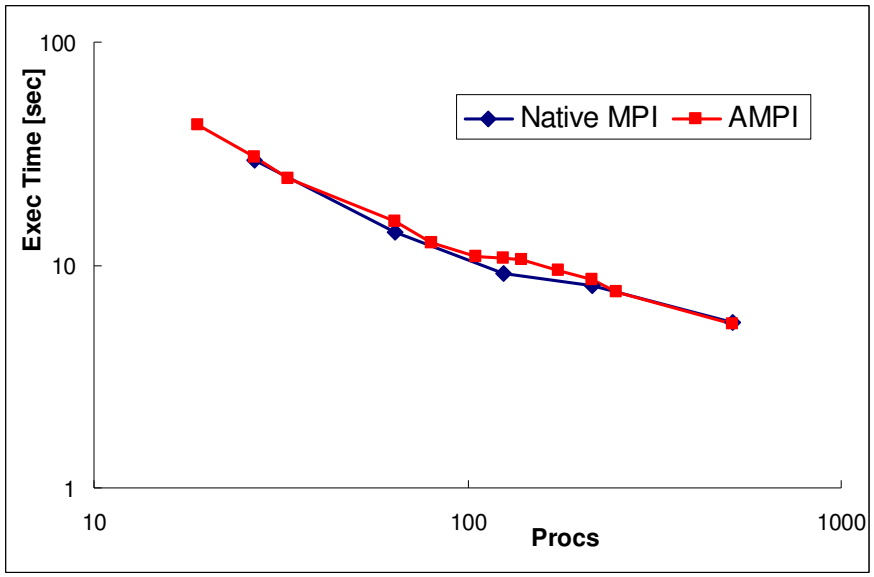


Problem setup: 3D stencil calculation of size 240^3 run on Lemieux.

Shows AMPI with virtualization ratio of 1 and 8.

Comparison with Native MPI

- Performance
  - Slightly worse w/o optimization
  - Being improved
- Flexibility
  - Big runs on a few processors
  - Fits the nature of algorithms



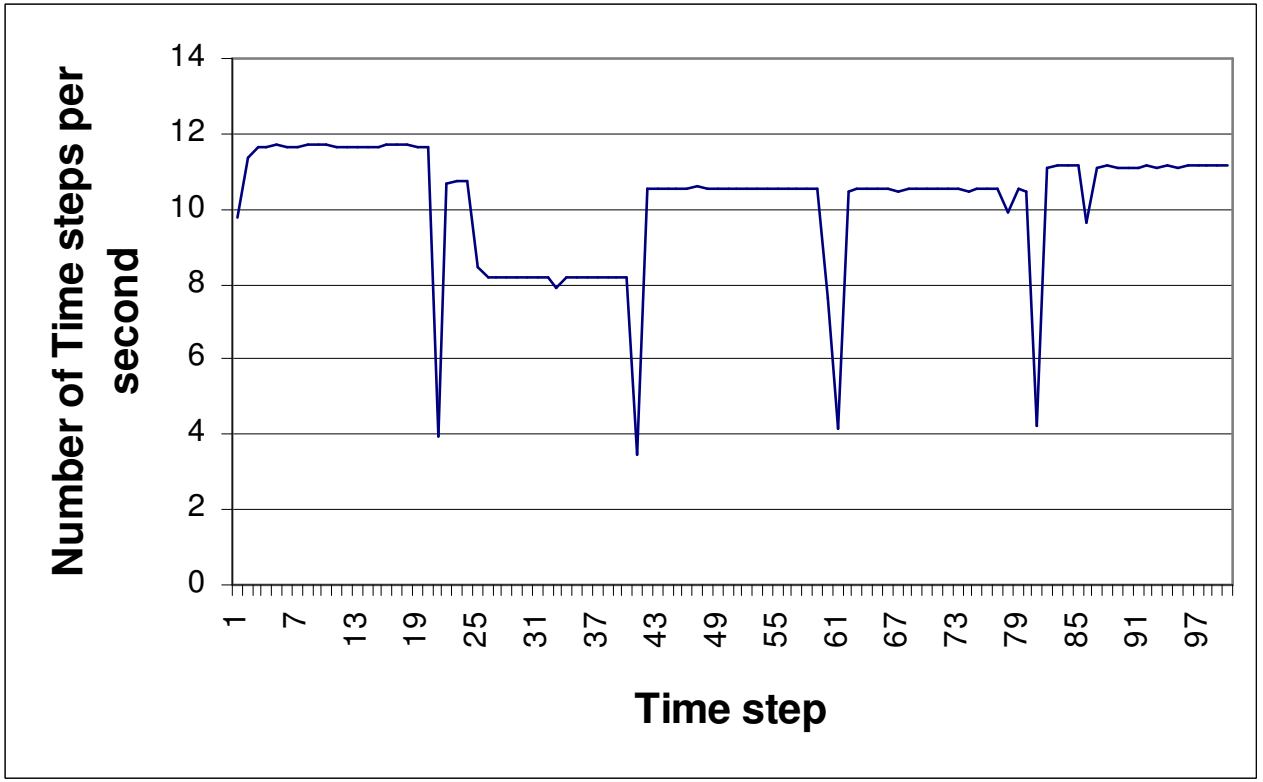
Problem setup: 3D stencil calculation of size 240^3 run on Lemieux.

AMPI runs on any # of PE's (eg 19, 33, 105). Native MPI needs cube #.

Automatic Load Balancing

- Problems
  - Dynamically varying applications
  - Load imbalance impacts overall performance
  - Difficult to move jobs between processors
- Implementation
  - Load balancing by migrating objects (VP's)
    - RTS collects CPU and network usage of VP's
    - New mapping based on collected information
    - Threads are packed up and shipped as needed
  - Different variations of strategies available

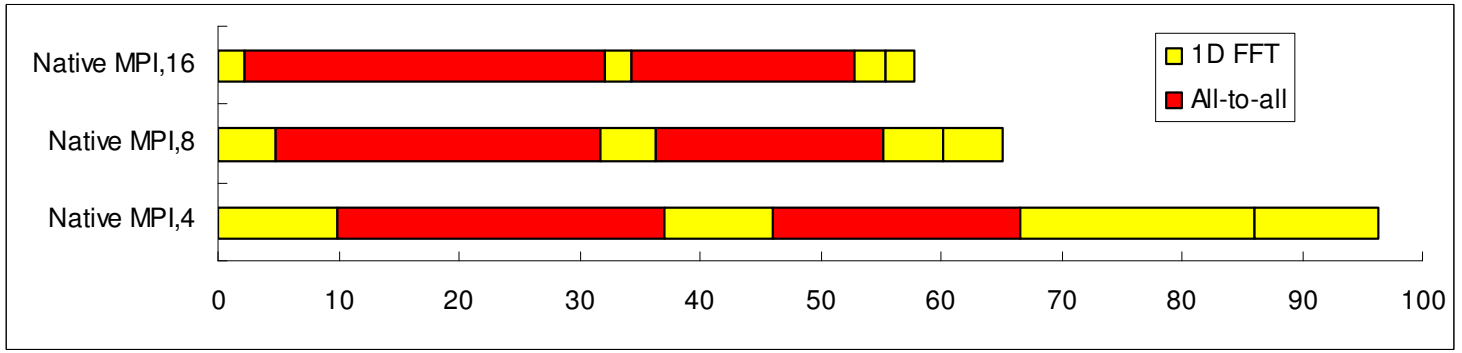
Load Balancing Example



AMR application
Refinement happens at step 25
Load balancer is activated at time steps 20, 40, 60, and 80.

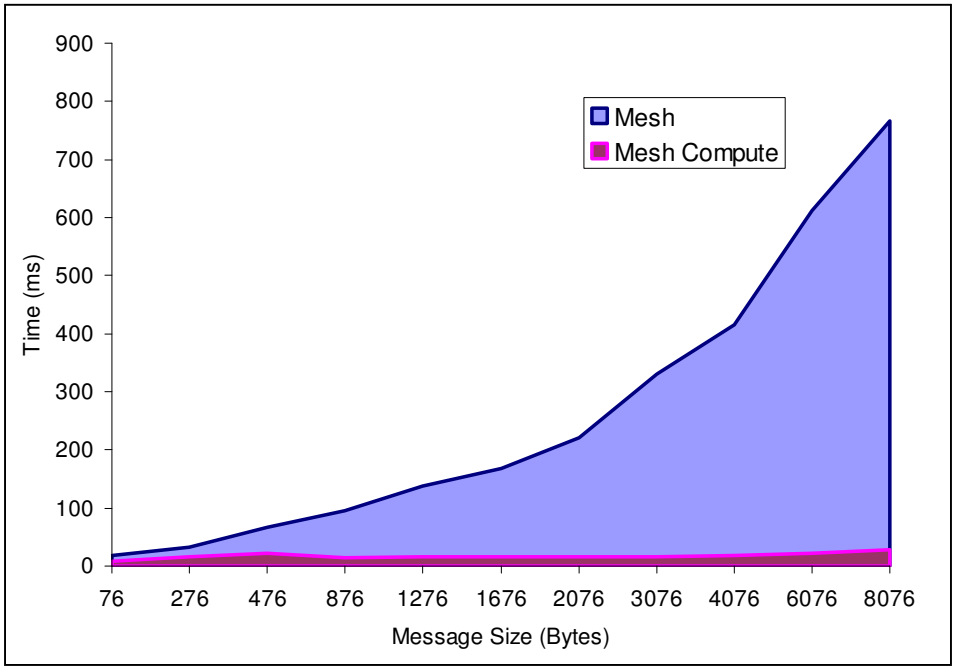
Collective Operations

- Problem with collective operations
  - Complex: involving many processors
  - Time consuming: designed as blocking calls in MPI



Time breakdown of 2D FFT benchmark [ms]

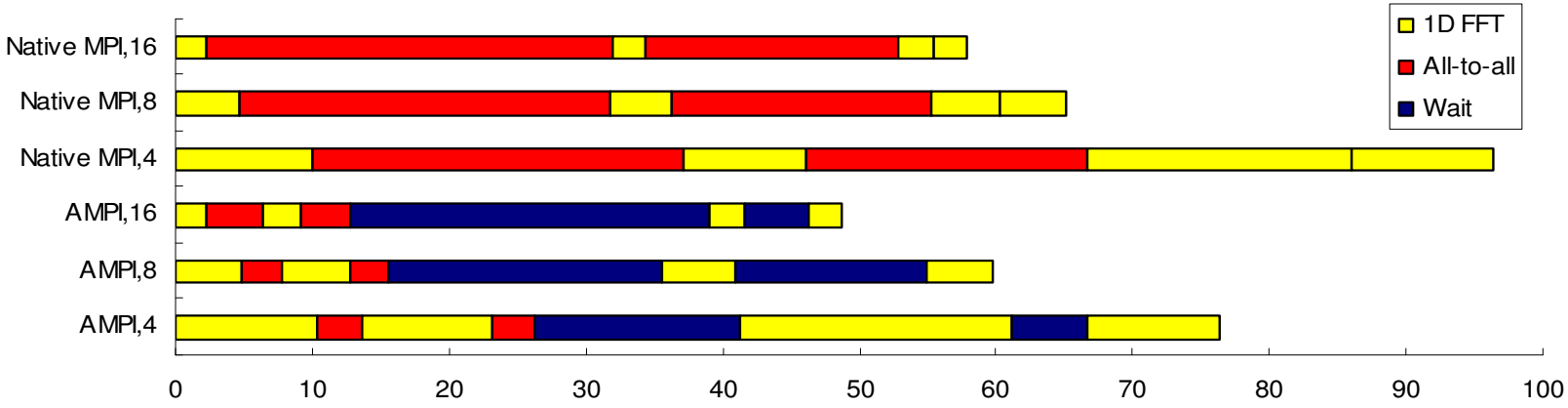
Collective Communication Optimization



Time breakdown of an all-to-all operation
using Mesh library

- Computation is only a small proportion of the elapsed time
- A number of optimization techniques are developed to improve collective communication performance

Asynchronous Collectives

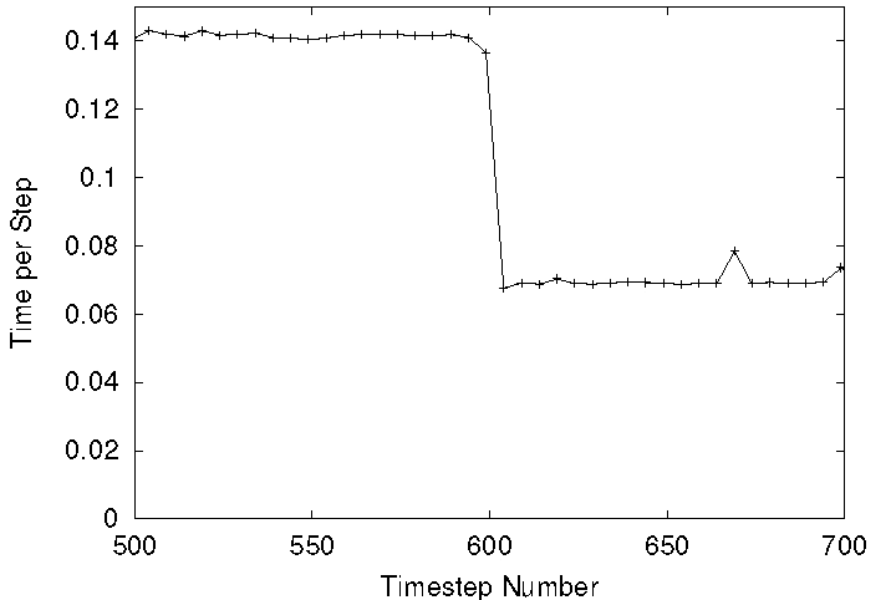


Time breakdown of 2D FFT benchmark [ms]

- VP's implemented as threads
- Overlapping computation with waiting time of collective operations
- Total completion time reduced

Shrink/Expand

- Problem: Availability of computing platform may change
- Fitting applications on the platform by object migration

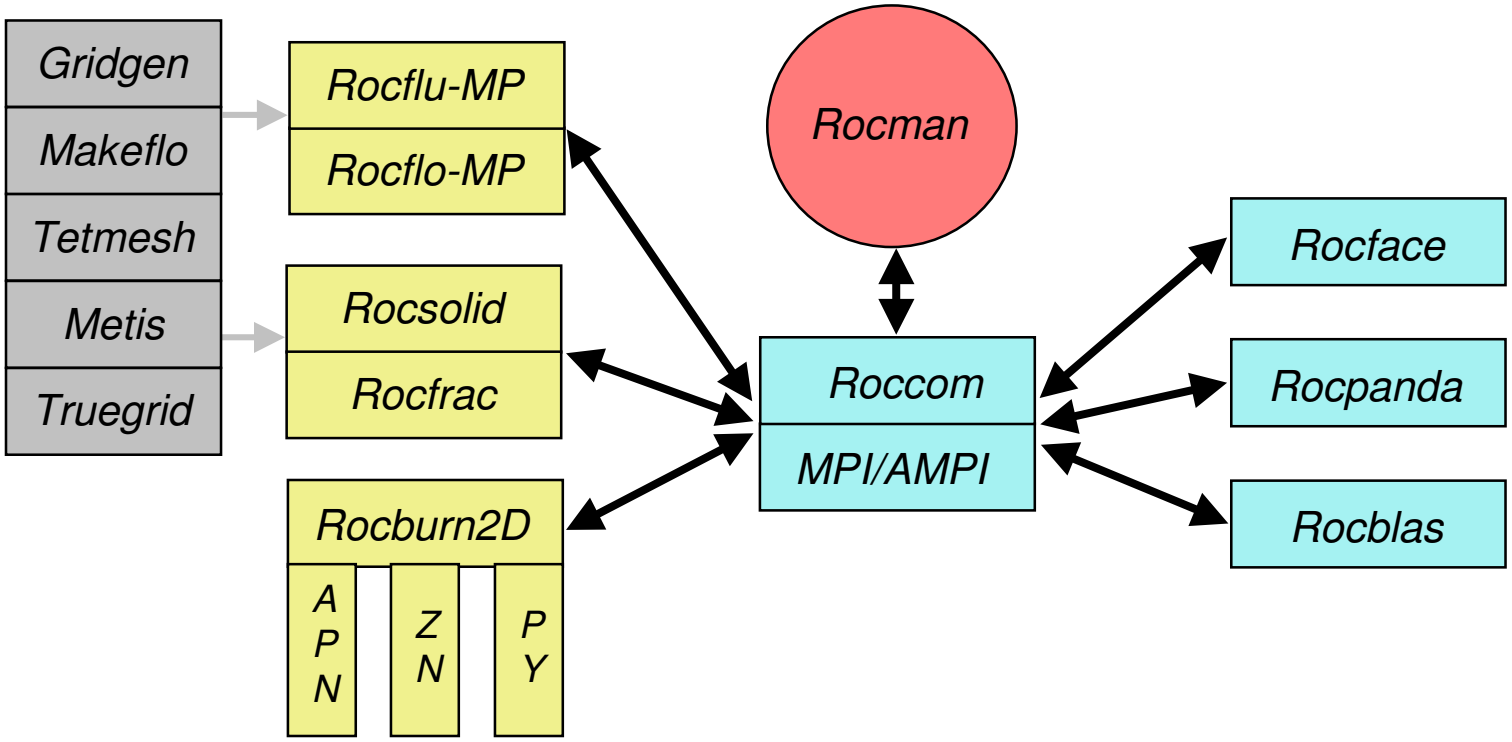


Time per step for the million-row CG solver on a 16-node cluster
Additional 16 nodes available at step 600

Summary of Advantages

- Run MPI programs on any # of PE's
- Overlap computation with communication
- Automatic load balancing
- Asynchronous collective operations
- Shrink/expand
- Automatic checkpoint/restart mechanism
- Cross communicators
- Interoperability

Application Example: GEN2



Future Work

- Performance prediction via direct simulation
  - Performance tuning w/o continuous access to large machine
- Support for visualization
  - Facilitating debugging and performance tuning
- Support for MPI-2 standard
  - ROMIO as parallel I/O library
  - One-sided communications

Thank You!

Free download of AMPI is available at:
http://charm.cs.uiuc.edu/

Parallel Programming Lab
at University of Illinois