Support for Adaptivity in ARMCI Using Migratable Objects

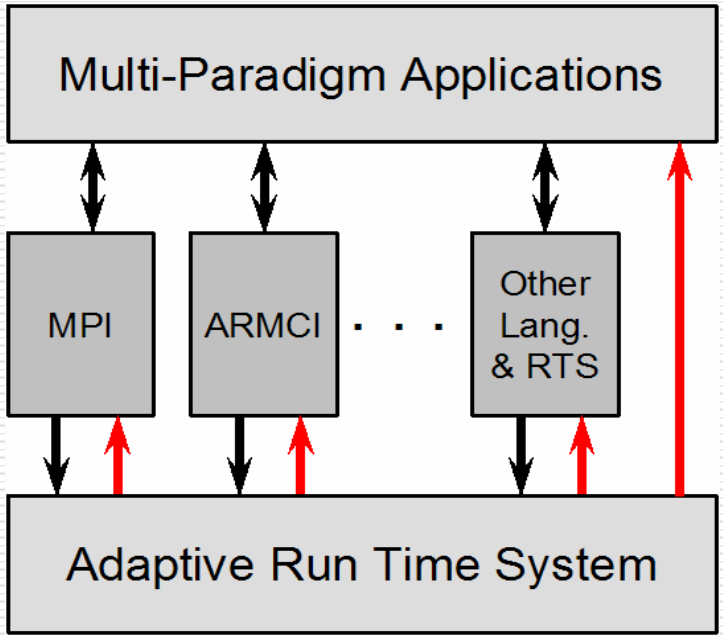
Chao Huang, Chee Wai Lee, Laxmikant Kale
Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

Motivation

- Different programming paradigms fit different algorithms and applications
- Adaptive Run-Time System (ARTS) offers performance benefits
- Goal: to support ARMCI and global address space languages on ARTS

Common RTS

- Motivations for common run-time system
  - Support concurrent composibility
  - Support common functions: load-balancing, checkpoint

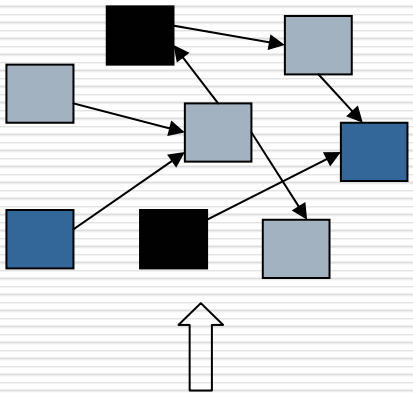


Outline

- Motivation
- Adaptive Run-Time System
- Adaptive ARMCI Implementation
- Preliminary Results
  - Microbenchmarks
  - Checkpoint/Restart
  - Application Performance: LU
- Future Work

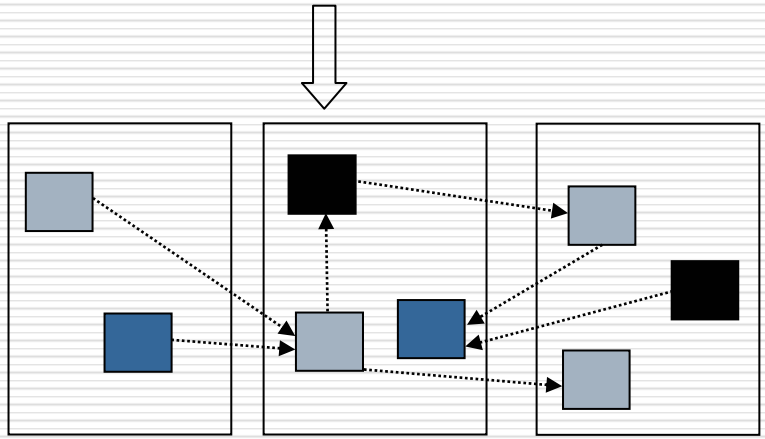
ARTS with Migratable Objects

- Programming model
  - User decomposes work to parallel objects (VPs)
  - RTS maps VPs onto physical processors
  - Typically, number of VPs >> P, to allow for various optimizations



User View

System Implementation

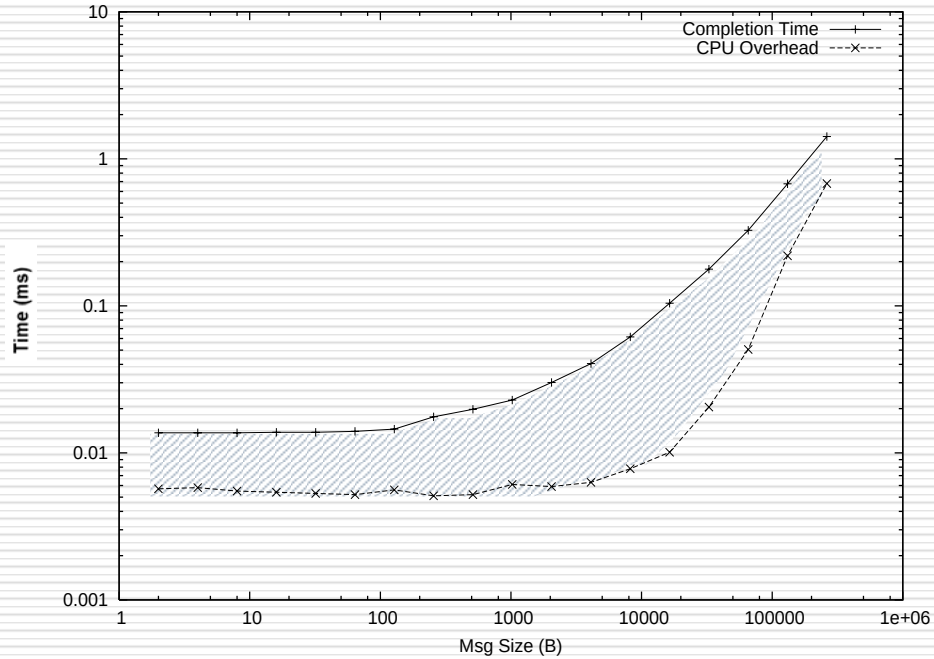


Features and Benefits of ARTS

- Adaptive overlap
- Automatic load balancing
- Automatic checkpoint/restart
- Communication optimizations
- Software engineering benefits

Adaptive Overlap

- Challenge: Gap between completion time and CPU overhead
- Solution: Overlap between communication and computation



Completion time and CPU overhead of 2-way ping-pong communication on Apple G5 Cluster

Automatic Load Balancing

- Challenge
  - Dynamically varying applications
  - Load imbalance impacts overall performance
- Solution
  - Measurement-based load balancing
    - Scientific applications are typically iteration-based
    - The *Principle of Persistence*
    - RTS collects CPU and network usage of VPs
  - Load balancing by migrating threads (VPs)
    - Threads can be packed and shipped as needed
  - Different variations of load balancing strategies
    - Eg. communication-aware, topology-based

Features and Benefits of ARTS

- Adaptive overlap
- Automatic load balancing
- Automatic checkpoint/restart
- Communication optimizations
- Software engineering benefits

Outline

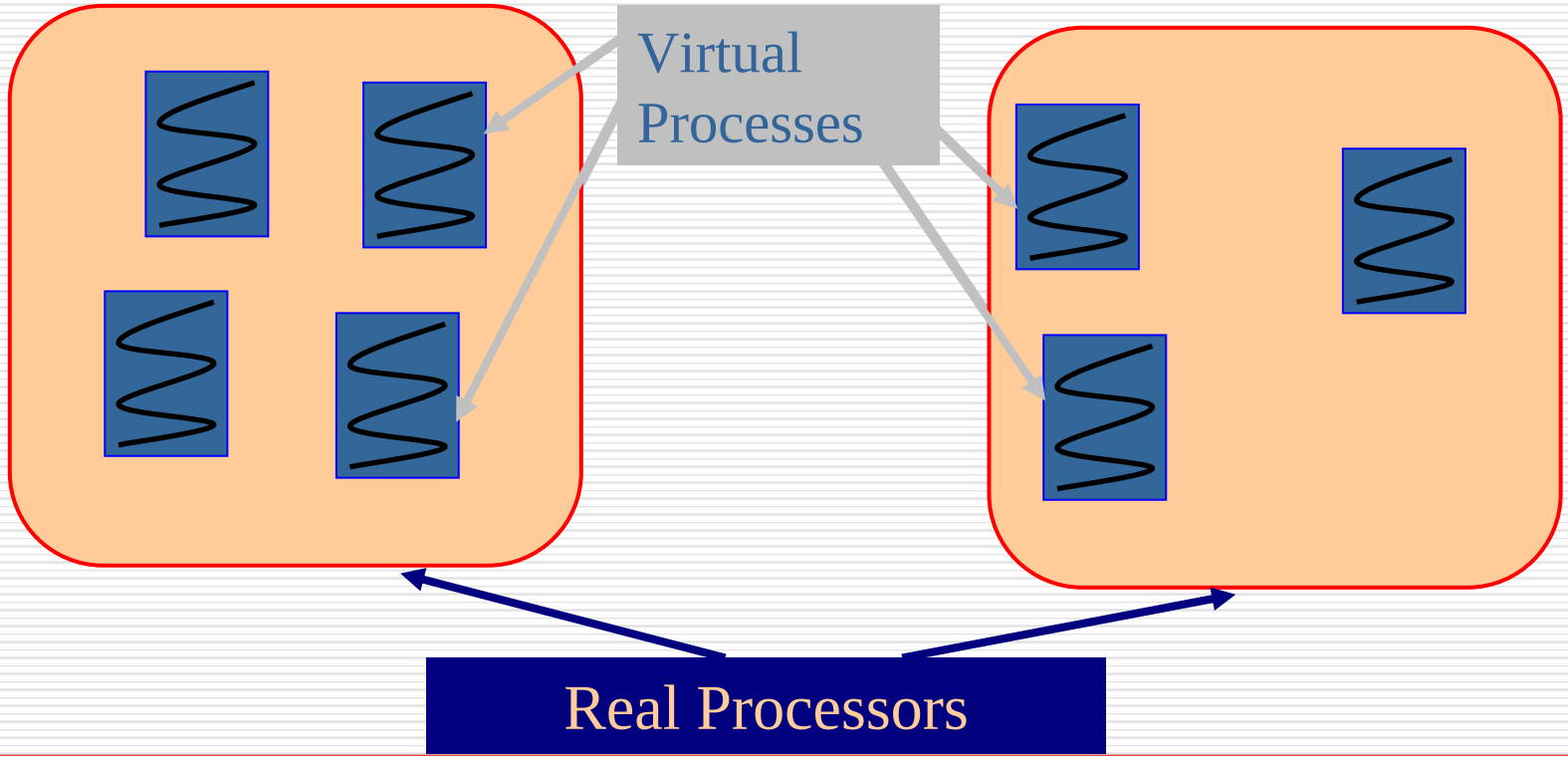
- Motivation
- Adaptive Run-Time System
- Adaptive ARMCI Implementation
- Preliminary Results
  - Microbenchmarks
  - Checkpoint/Restart
  - Application Performance: LU
- Future Work

ARMCI

- Aggregate Remote Memory Copy Interface (ARMCI)
  - Remote memory access (RMA) operations (one-sided communication)
  - Contiguous and noncontiguous (strided, vector); blocking and non-blocking
- Supporting various global-address space models
  - Global Array, Co-Array Fortran compiler, Adlib
- Built on top of MPI or PVM
  - Now on Charm++

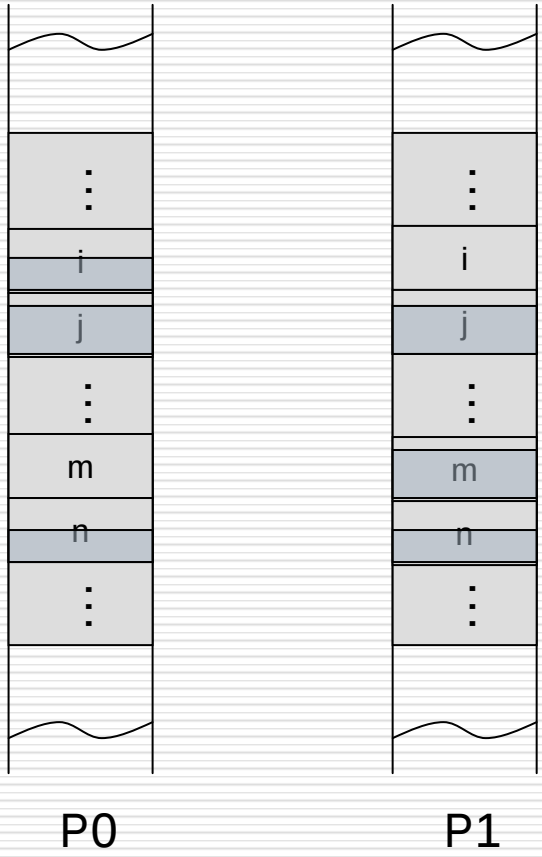
Virtualizing ARMCI Processes

- Each ARMCI virtual process is implemented by a light-weight, user-level *thread* embedded in a migratable *object*

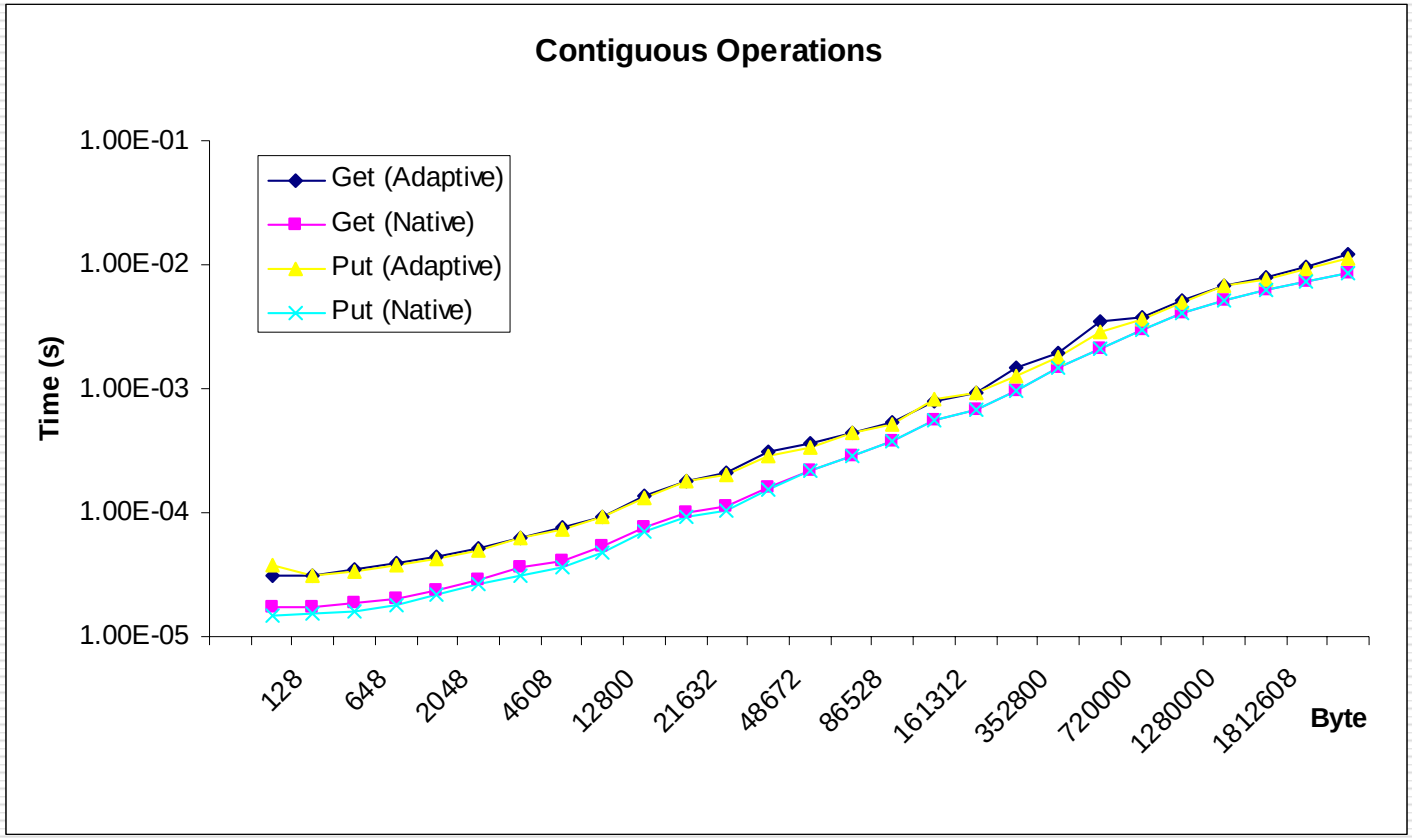


Isomalloc Memory

- Isomalloc approach for migratable threads
  - Same iso-address area in all nodes' virtual address space
  - Separate regions globally reserved for each VP
  - Memory allocated locally
  - Thread data moved, without pointer or address update

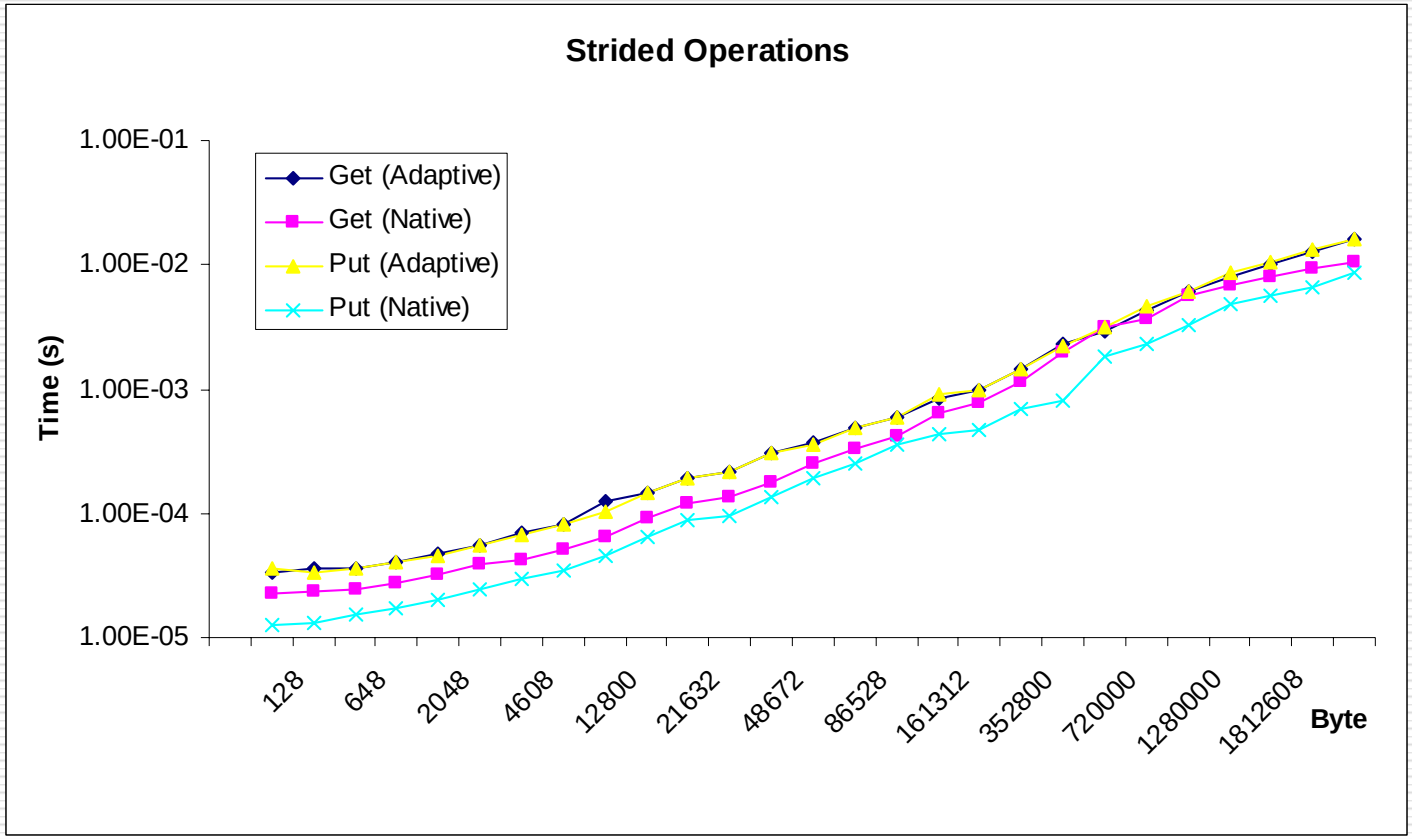


Microbenchmarks



Performance of contiguous operation on IA64 Cluster

Microbenchmarks



Performance of strided operation on IA64 Cluster

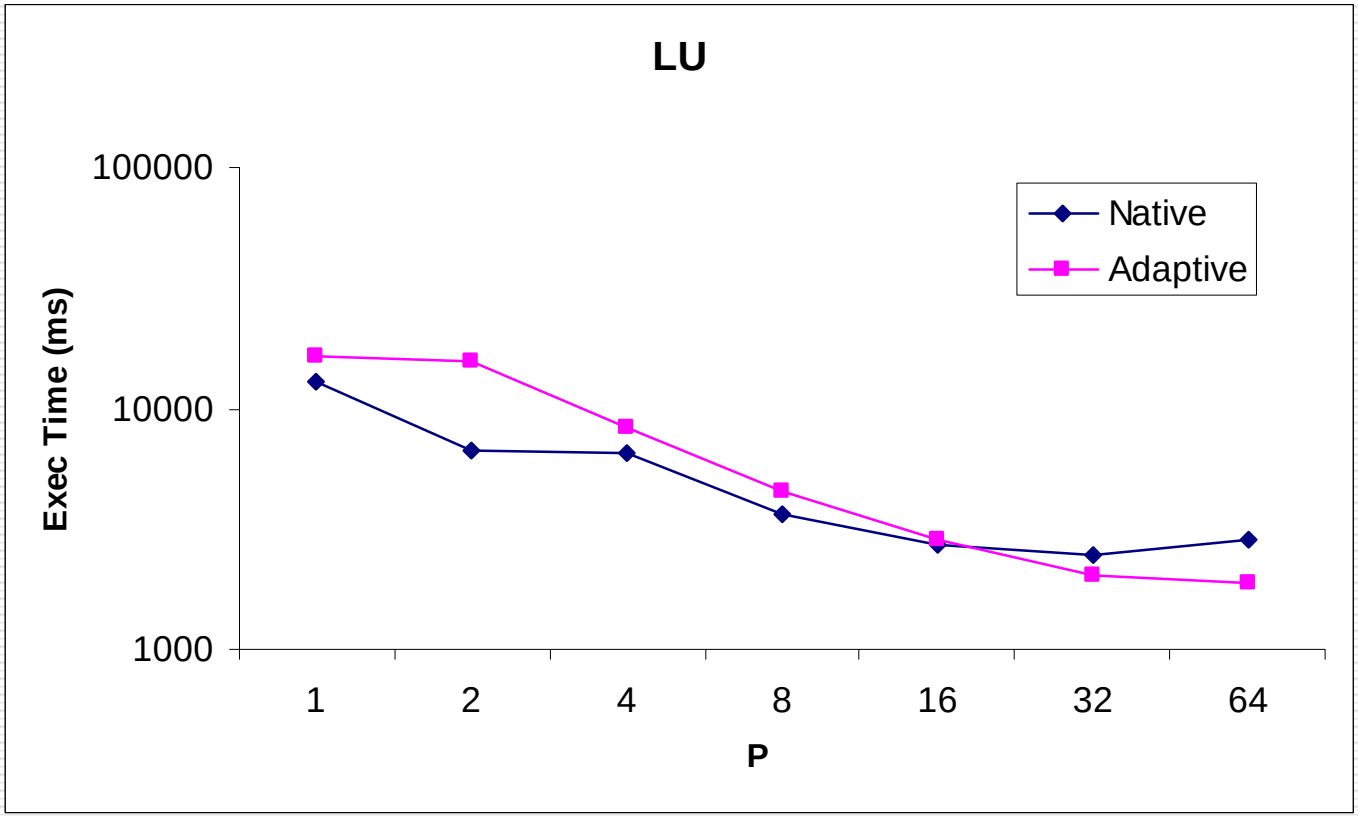
Checkpoint Time

- Checkpoint/restart automated at run-time level
  - User inserts simple function calls
- Possible NFS bottleneck for on-disk scheme
  - Alternative: in-memory scheme

P	Total Data (MB)	Time (ms)	Bandwidth (MB/s)
2	20.05	221	90.8
4	22.29	249	89.7
8	26.5	303	87.6
16	35.43	366	96.9
32	53.27	533	100

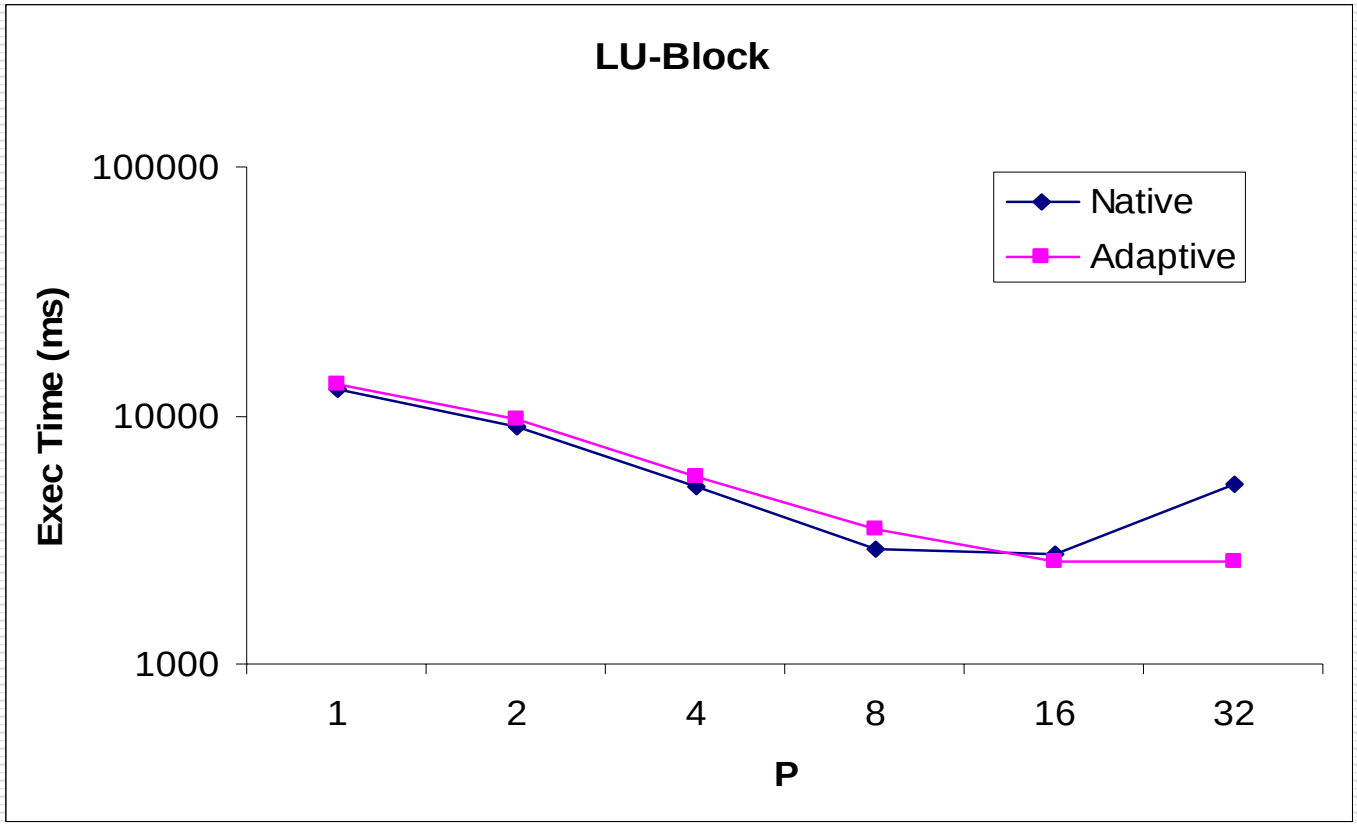
On-disk checkpoint time of LU, on 2 to 32 PEs on IA64 Cluster

Application Performance



Performance of LU application on IA64 Cluster

Application Performance



Performance of LU-Block application on IA64 Cluster

Future Work

- Performance Optimization
  - Reduce overheads
- Performance Tuning
  - Visualization and analysis tools
- Port other GAS languages
  - GA and CAF compiler