Scalable in-memory checkpoint for hard and soft error protection with automatic restart on failure

Xiang Ni

Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

May, 2013

Outline

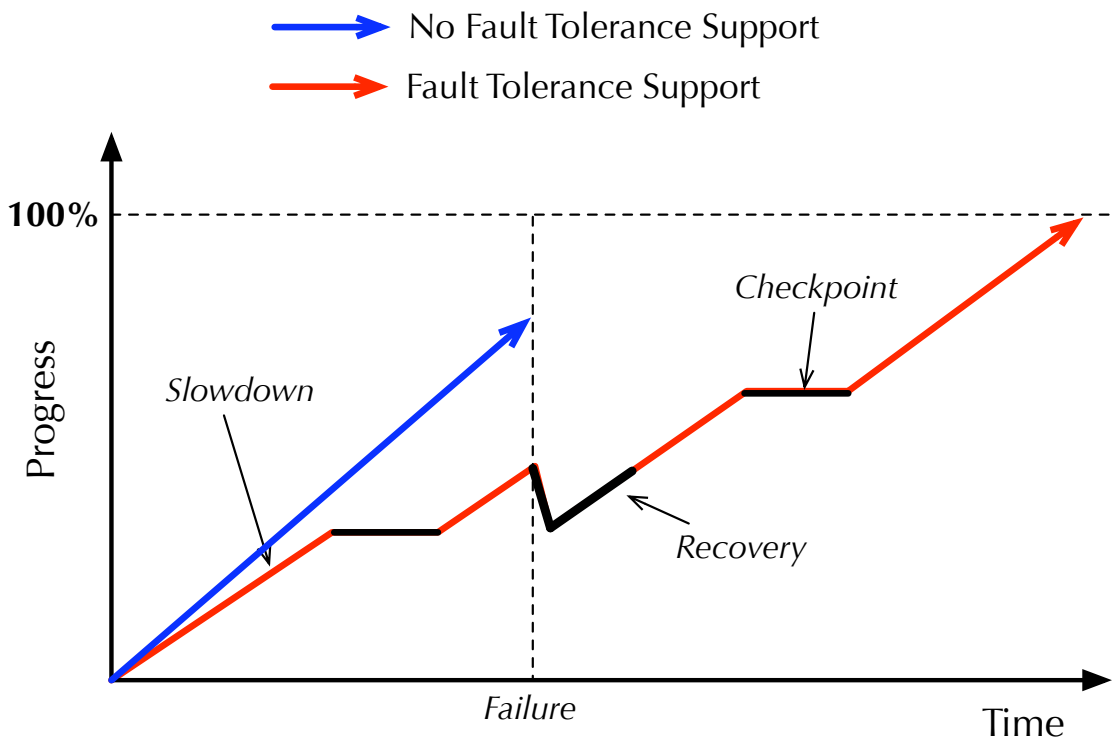
1 Fault Tolerance Philosophy in Charm++

2 Asynchronous Checkpoint/Restart

3 Replication enhanced Checkpoint Restart

Our Philosophy

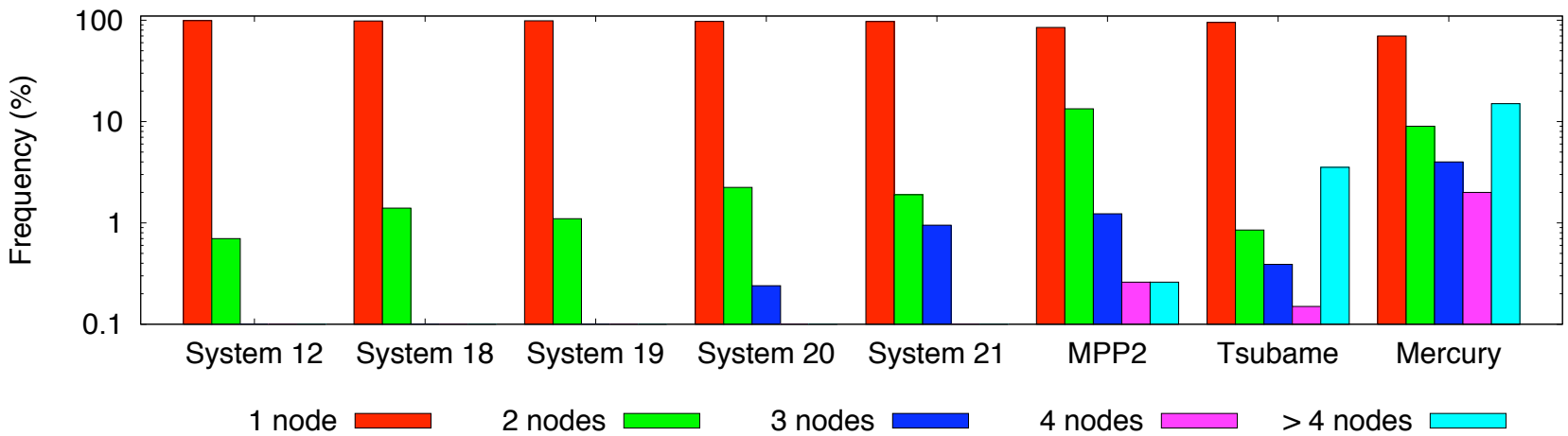
Keep progress rate despite failures



- Optimize for the common case
- Minimize performance overhead

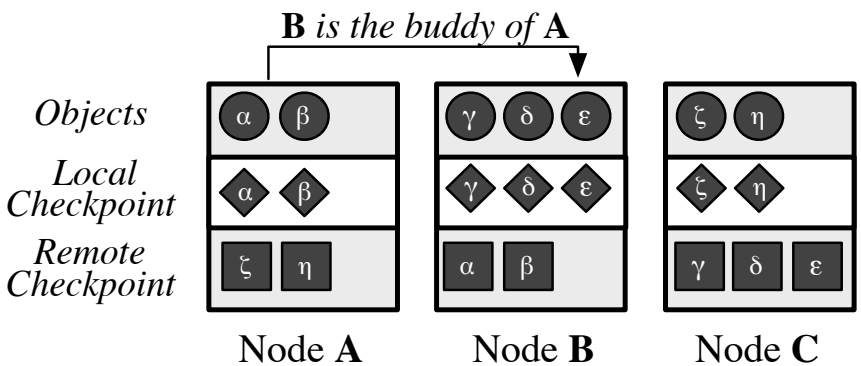
Optimize for the common case

- Failures rarely bring down more than one node at a time
- In Jaguar (now Titan, top 1 supercomputer), 92.27% of failures are individual node crashes
- So, our strategies are geared to handle all single-node failures and most multi-node failures



Minimize performance overhead

- Automatic restart:
  - Failure detection in runtime system
  - Immediate rollback-recovery
- Parallel recovery
- Faster checkpoint
  - Double in-memory checkpoint/restart



Minimize performance overhead

- Automatic restart:
  - Failure detection in runtime system
  - Immediate rollback-recovery
- Parallel recovery
- Faster checkpoint
  - Double in-memory checkpoint/restart
  - Semi-blocking checkpointing: asynchronously store the checkpoint remotely

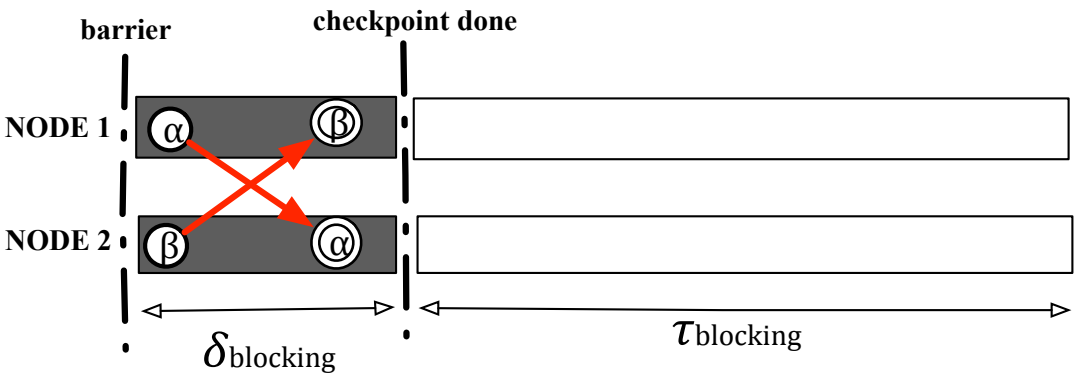
Outline

1 Fault Tolerance Philosophy in Charm++

2 Asynchronous Checkpoint/Restart

3 Replication enhanced Checkpoint Restart

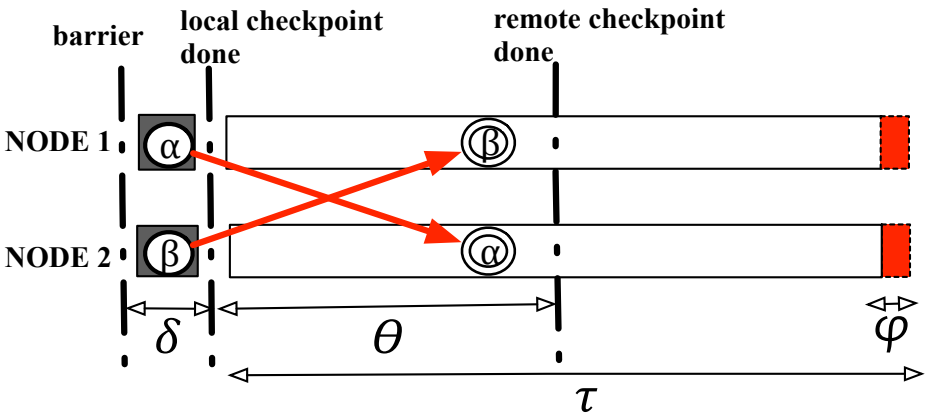
Blocking Checkpoint



$\tau_{blocking}$	checkpoint interval
$\delta_{blocking}$	checkpoint overhead

- Each node has a buddy node to store the checkpoint.
- Resume computation after all the nodes have successfully saved the checkpoints in their buddy nodes.

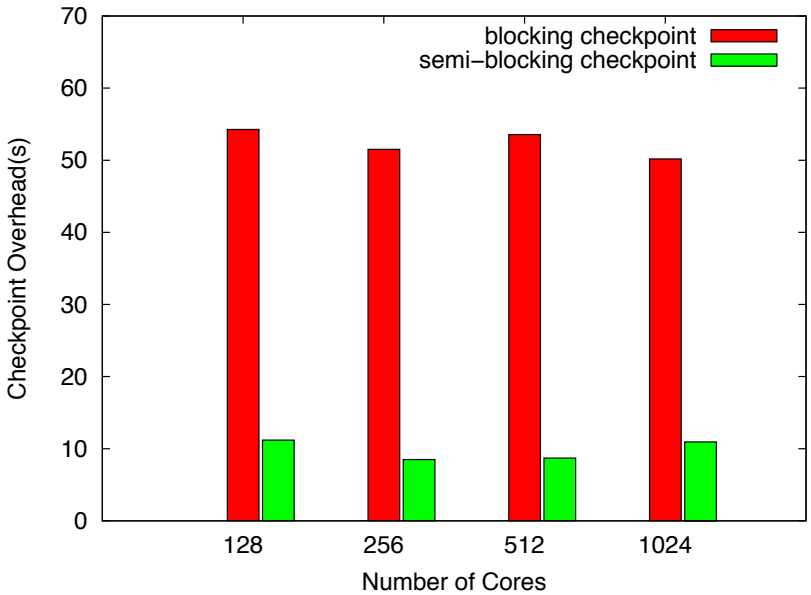
Semi-blocking Checkpointing



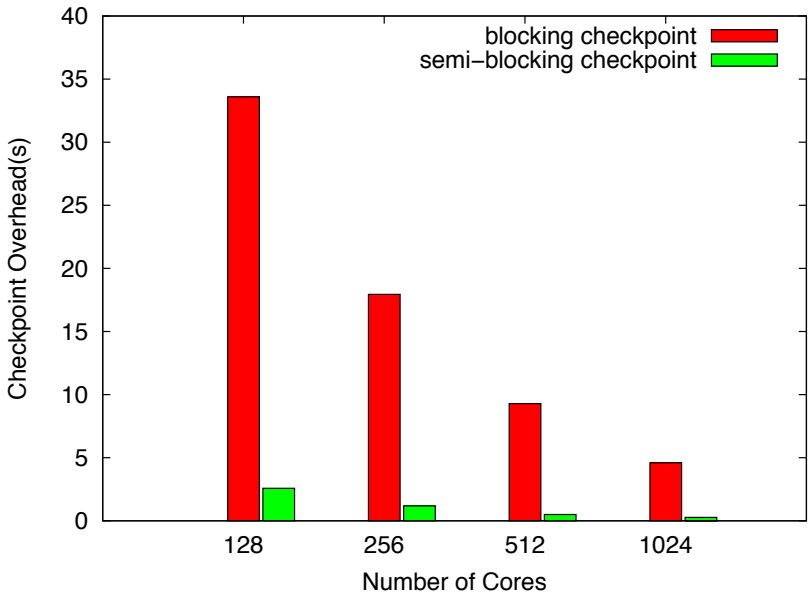
τ	checkpoint interval
δ	local checkpoint overhead
θ	overlap period
φ	remote checkpoint interference

- Resume computation as soon as each node stores its own checkpoint (local checkpoint).
- Interleave the transmission of the checkpoint to buddy with application execution (remote checkpoint).

Single Checkpoint Overhead



Wave2D Weak Scaling



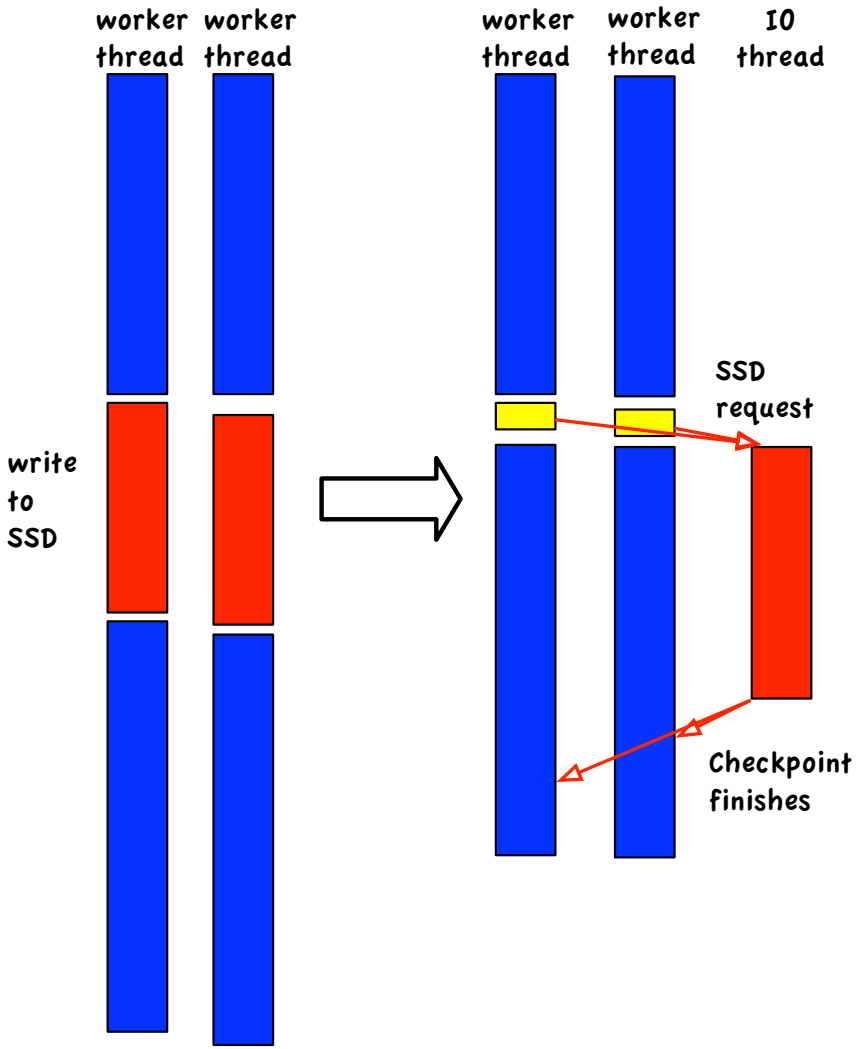
ChaNGa Strong Scaling

- Semi-Blocking checkpoint reduces checkpoint overhead significantly.

Leveraging Solid State Drives

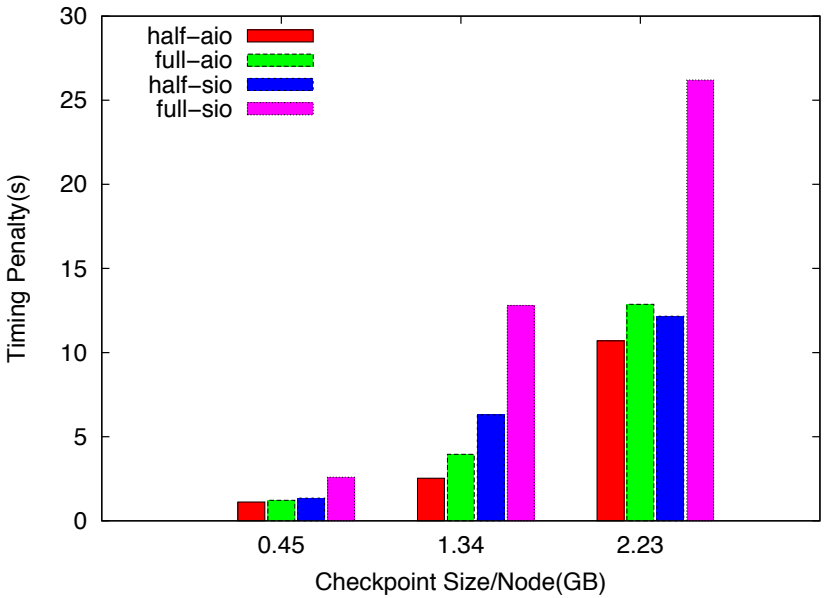
- Solid State Drive: becoming increasingly available on individual nodes
- Full SSD strategy
- Half SSD strategy
  - Only store remote checkpoint in SSD
  - Faster checkpoint and restart

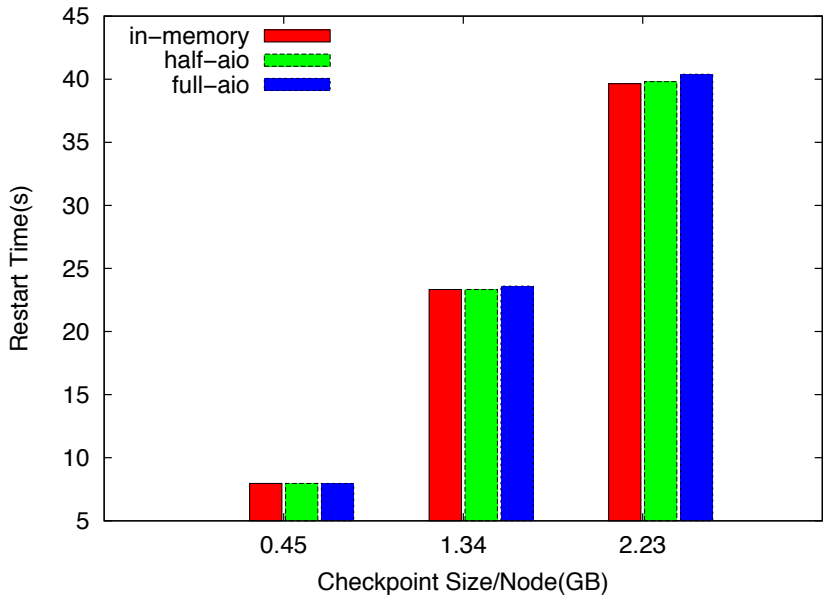
Asynchronous Checkpointing to SSD with IO thread



- IO threads
  - *Write checkpoint to/Read checkpoint from* SSD
    When receive request from worker thread.
  - Notify worker thread
    When SSD is done with certain request.

Checkpoint/Restart on SSD





- Half SSD strategy with asynchronous IO reduces the timing penalty for checkpointing to SSD
- Restart from SSD does not incur extra overhead

aio	asynchronous IO
sio	synchronous IO

Outline

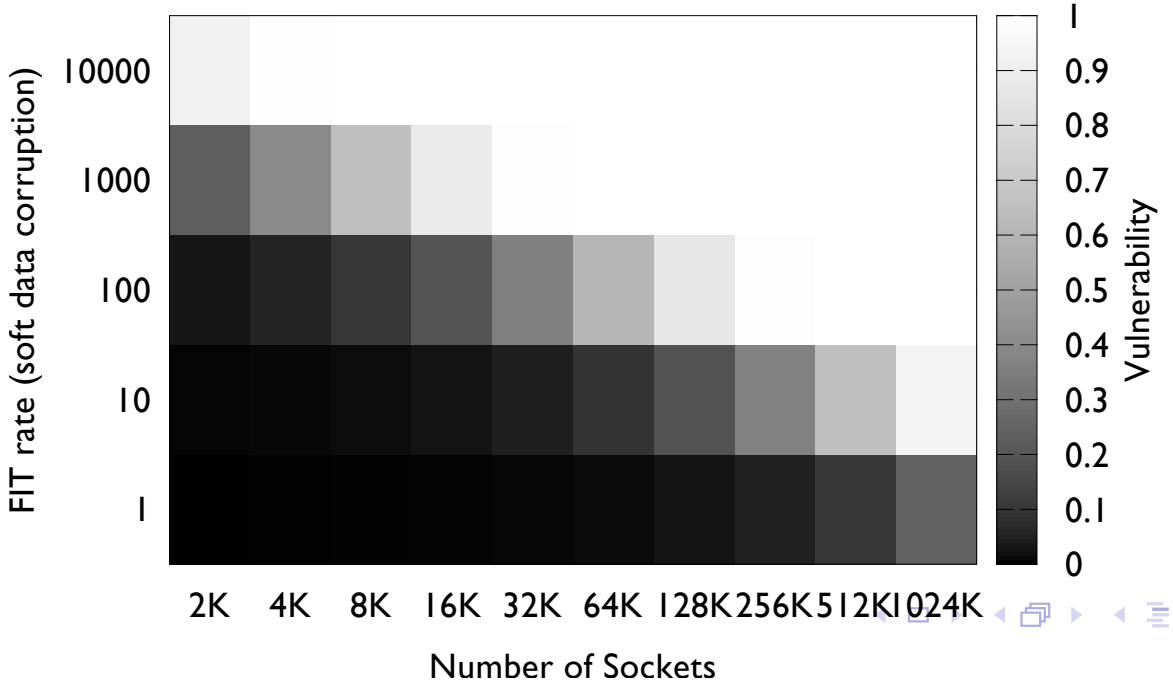
1 Fault Tolerance Philosophy in Charm++

2 Asynchronous Checkpoint/Restart

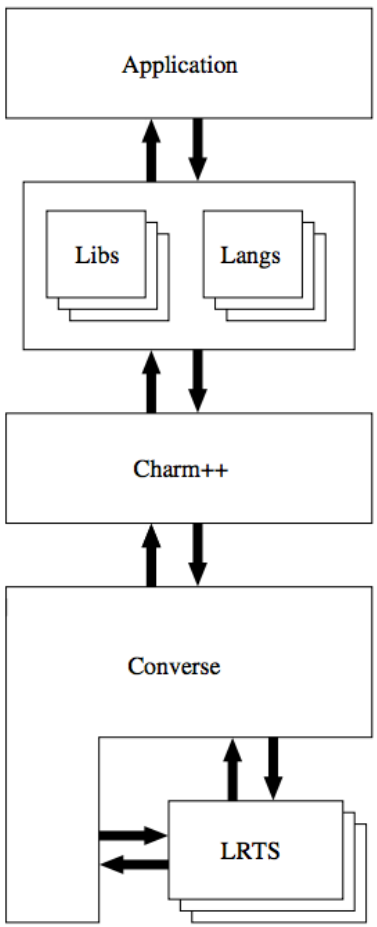
3 Replication enhanced Checkpoint Restart

New challenge: soft error

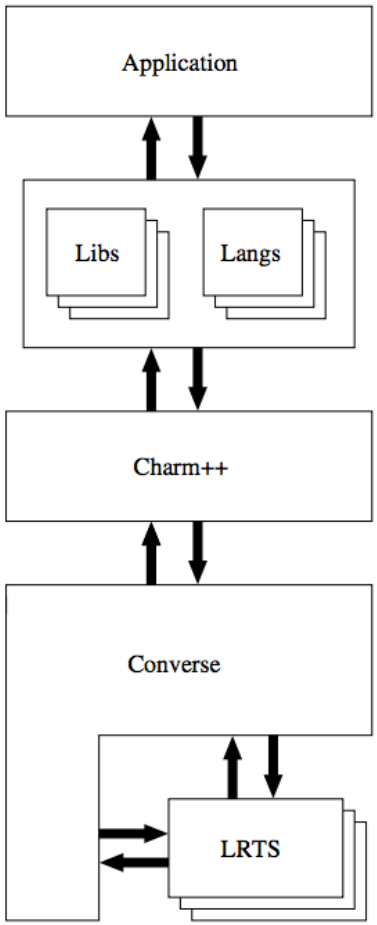
- Not just from cosmic rays
- Computer electronic's sensitivity to radiation increases as their dimensions and operating voltage decreases because of the requirements for high performance and low power.
- What may happen if soft failure rate keeps increasing?



Partition framework in Charm++

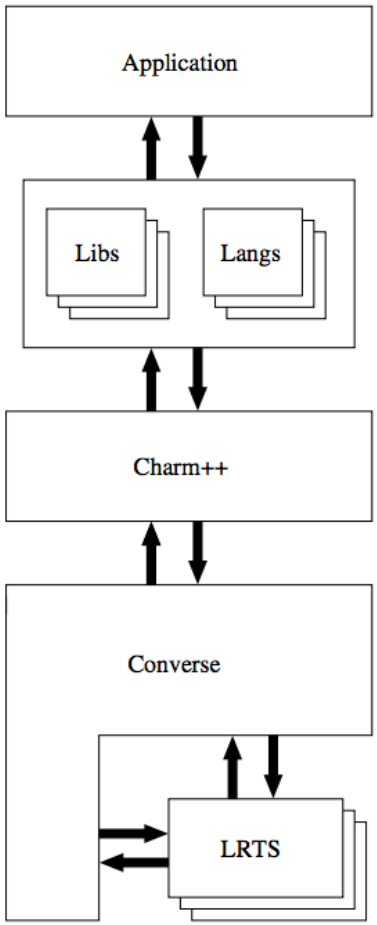


Partition framework in Charm++



- Ranking
  - Local rank
  - Global rank

Partition framework in Charm++



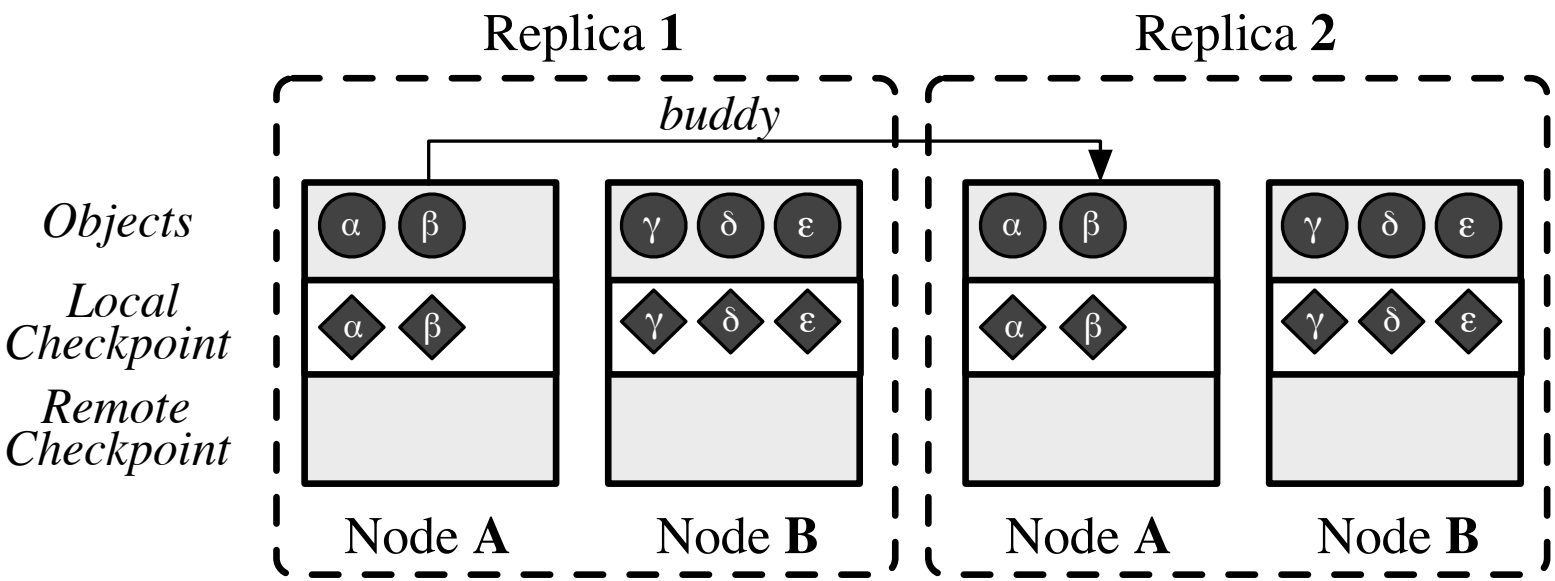
- Ranking
  - Local rank
  - Global rank
- Inter-partition communication
  - CmiInterSyncSend(local_rank, partition, size, message)
  - CmiInterSyncSendAndFree(local_rank, partition, size, message)

Replication enhanced Fault Tolerance Overview

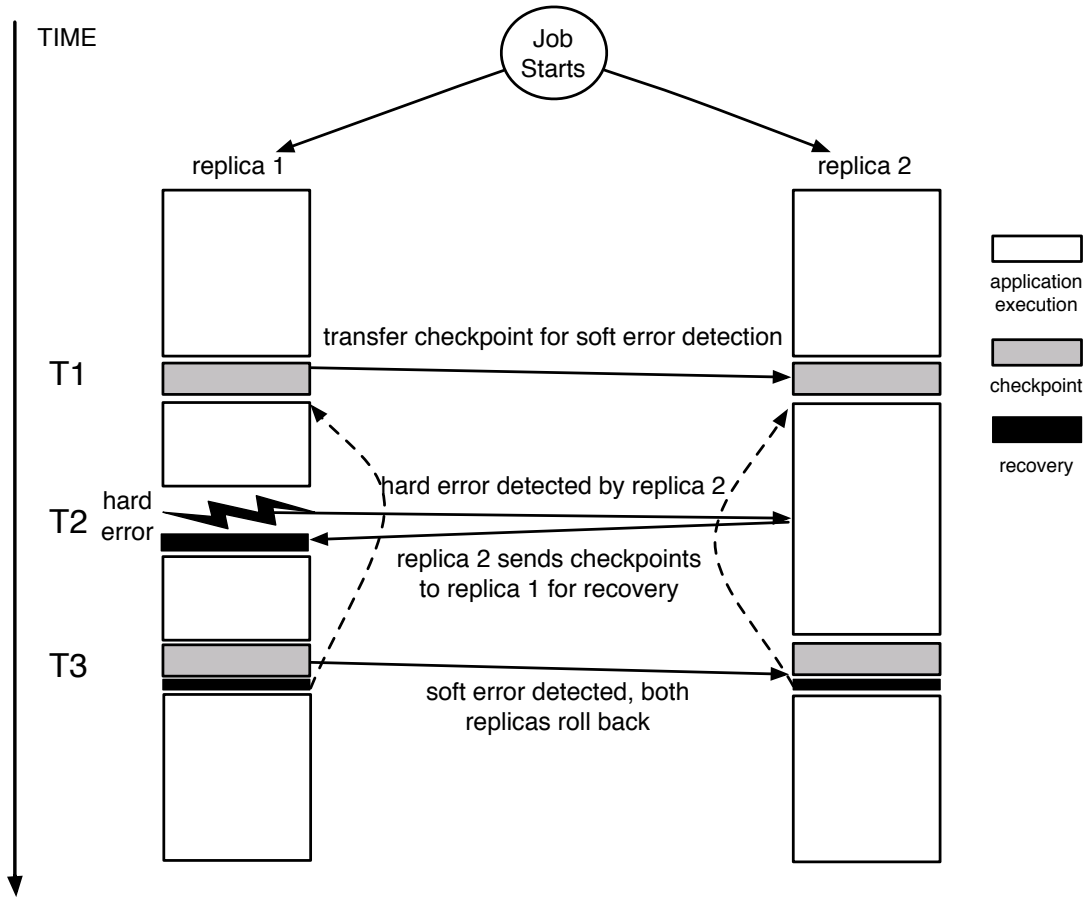
- Periodic soft data corruption detection
- Automatically correct soft error from checkpoint
- Yes, there are benefits for hard failure!
  - No need for remote checkpointing

Replication enhanced Fault Tolerance Overview

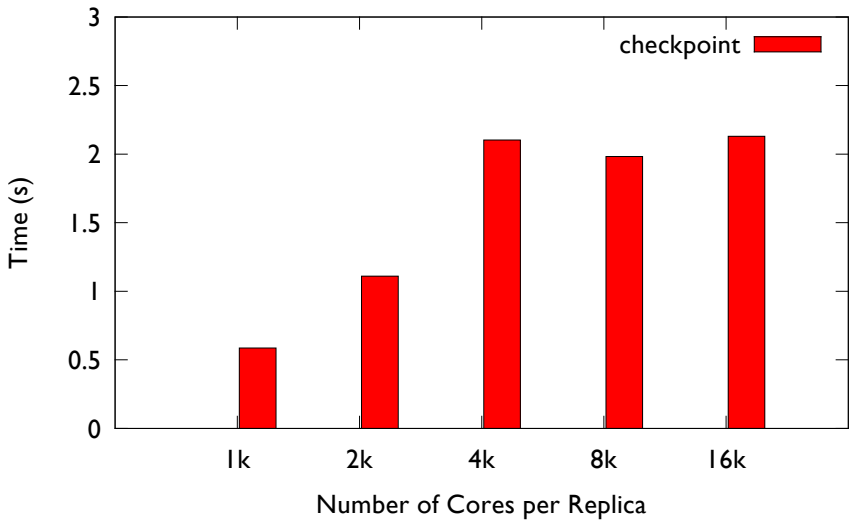
- Extension from the double in-memory checkpointing



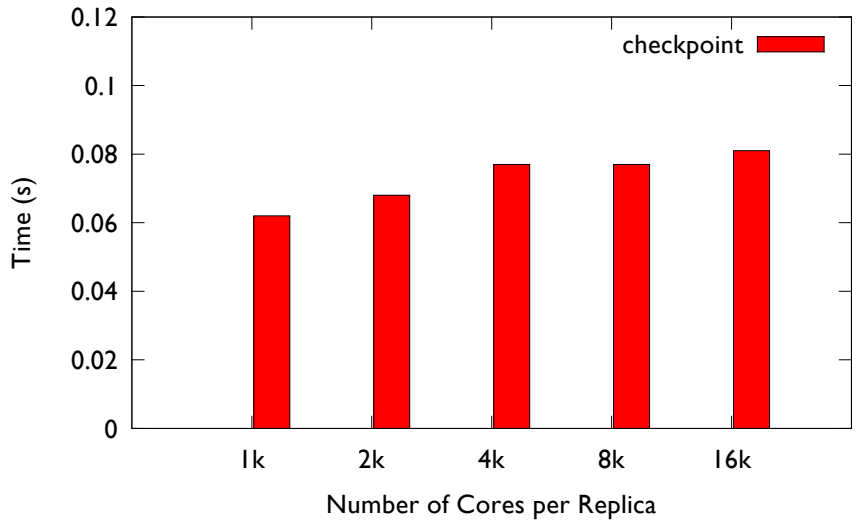
Replication enhanced Fault Tolerance Overview



Initial Result: soft error detection overhead



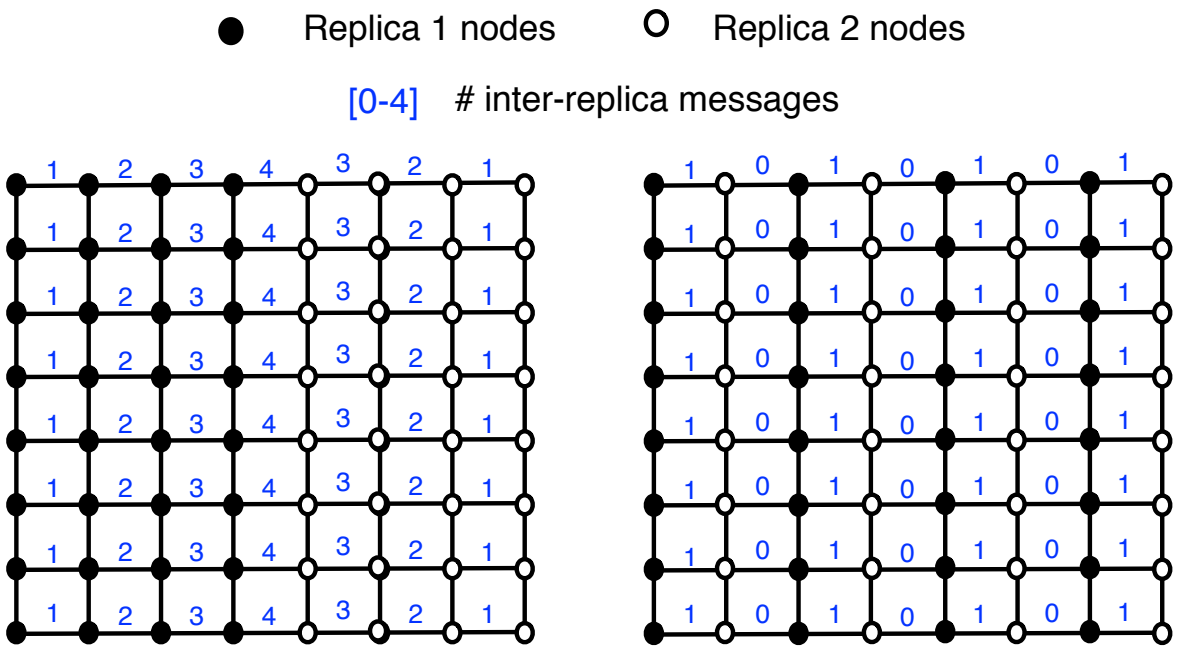
Jacobi3D AMPI



LeanMD

Optimization

- Topology aware mapping



(a) Default-mapping

(b) Optimal-mapping

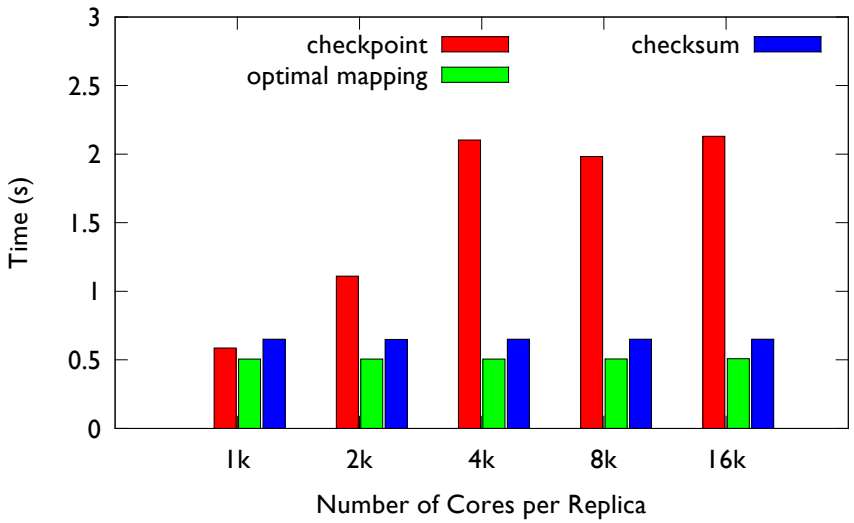
Optimization

- Checksum

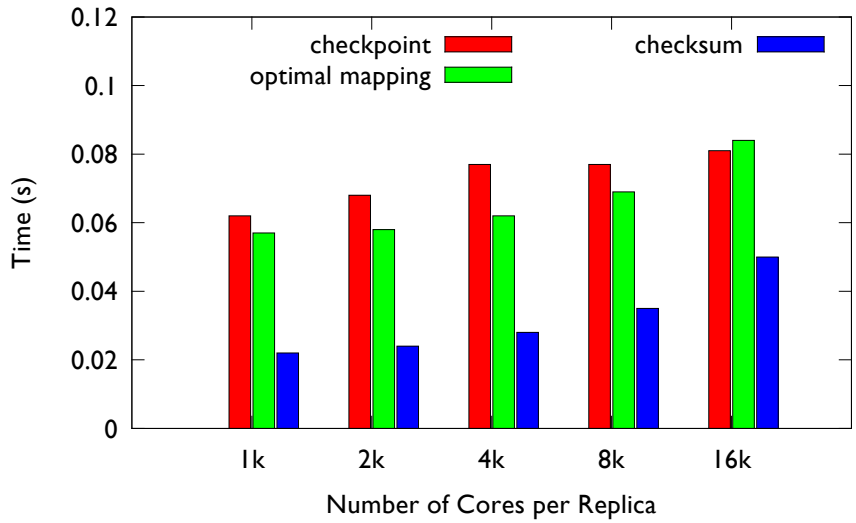
Optimization

- Checksum
- Issue with checksum
  - How to handle floating point round off error?

Result: after optimization

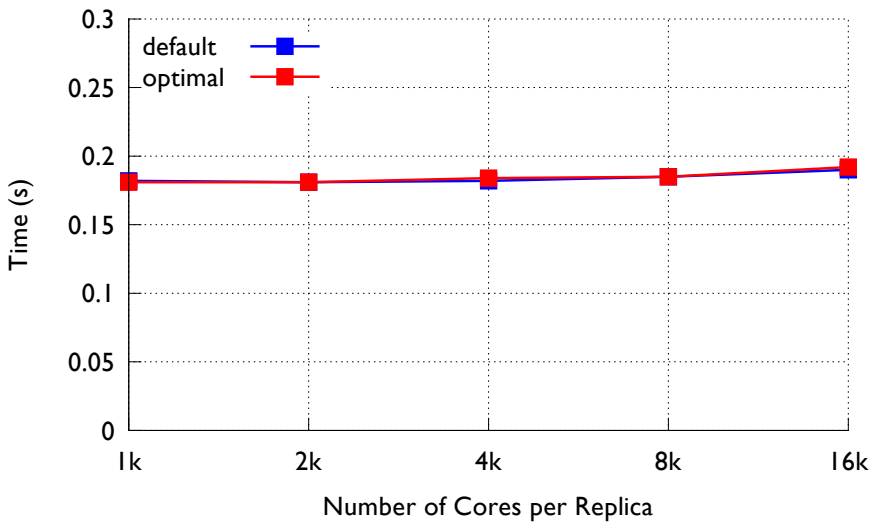


Jacobi3D AMPI

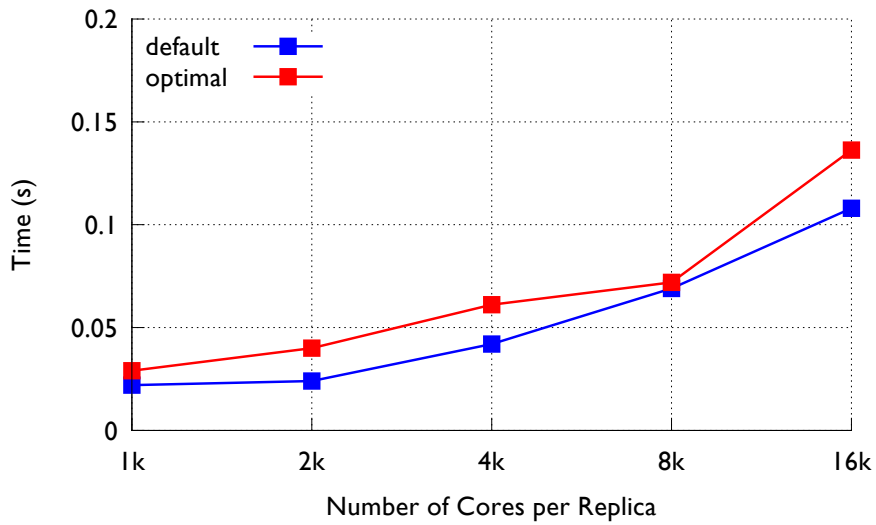


LeanMD

Result: recovery from hard failures



Jacobi3D AMPI



LeanMD

Thanks

- Thanks!
- Questions?