A Semi-automatic System for Application-level Checkpoint-Recovery

Keshav Pingali
Cornell University

Joint work with Greg Bronevetsky, Rohit Fernandes,

Daniel Marques, Martin Schulz(LLNL), Paul Stodghill,

Why CPR?

- Program runtimes are exceeding MTBF of hardware
  - Protein-folding on Blue Gene may take one year per protein
  - Fault-tolerance is critical
- Fault tolerance comes in different flavors
  - Mission-critical systems: no down-time, fail-over, redundancy
  - Computational science applications : restart after failure, minimizing lost work
- Fault models
  - Fail-stop:
    - Failed process dies silently w/o corrupting data
  - Byzantine:
    - Arbitrary misbehavior is allowed
- Our focus:
  - Computational science applications
  - Fail-stop faults
  - One solution: checkpoint/restart (CPR)

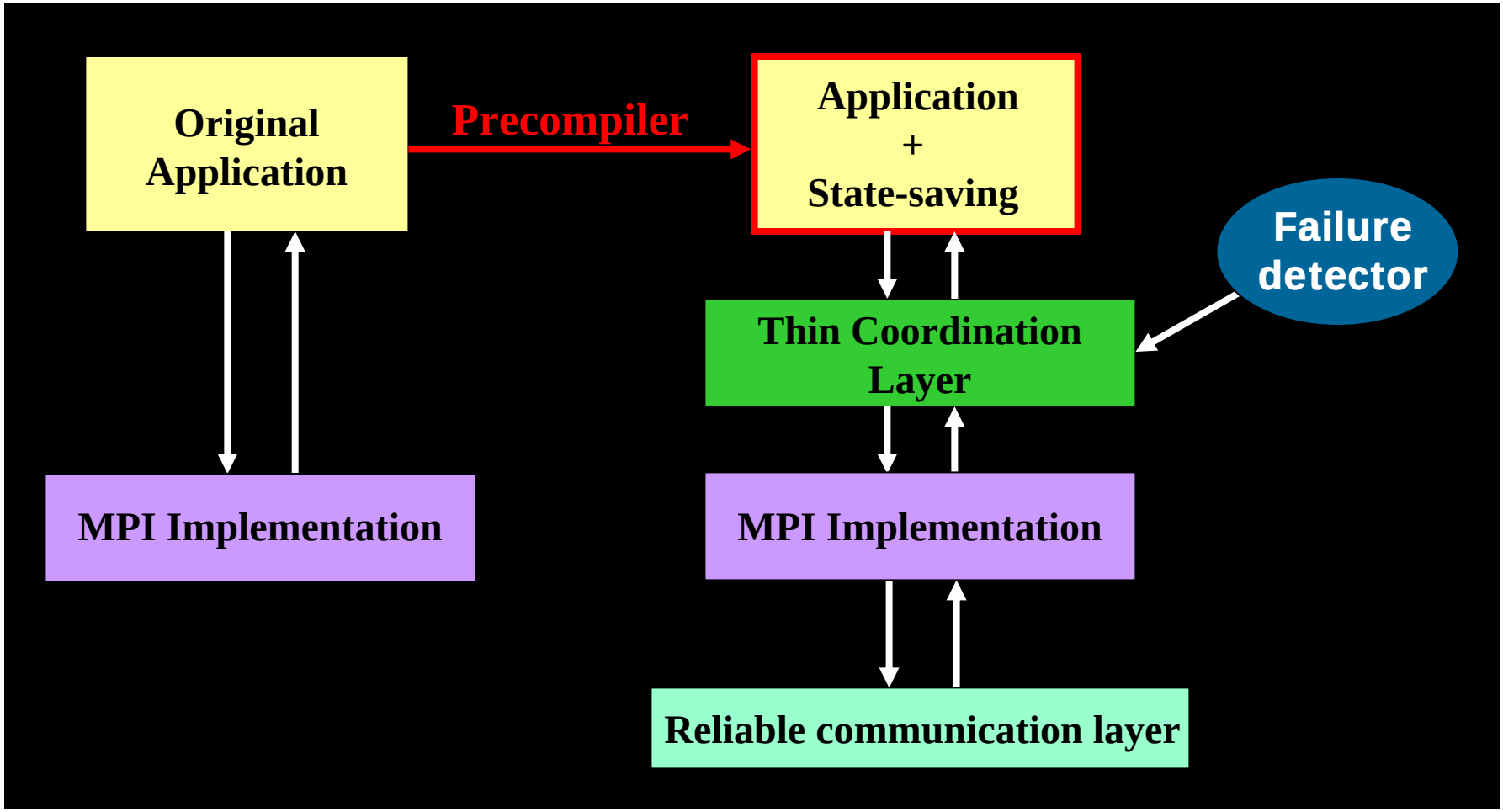
Why CPR? (contd.)

- Grid computing: utility computing
  - Programs execute wherever there are computational resources
  - Program are mobile to take advantage of changing resource availability
  - Key mechanism: checkpoint/restart
  - Identical platforms at different sites on grid
    - Platform-dependent checkpoints (cf. Condor)
  - Different platforms at different sites on grid
    - Platform-independent (portable) checkpoints

Two approaches to CPR

- System-level checkpointing (SLC) (eg) Condor
  - core-dump style snapshots of computations
  - mechanisms very architecture and OS dependent
  - checkpoints are not portable
- Application-level checkpointing (ALC)
  - programs are self-checkpointing and self-restarting
    - (eg) n-body codes save and restore positions and velocities of particles
  - amount of state saved can be much smaller than SLC
    - IBM's BlueGene protein folding : megabytes vs terabytes
- Disadvantage of current application-level check-pointing
  - manual implementation
  - requires global barriers in programs

Our approach



Cornell Checkpointing Compiler (C^3) Project

- Automate application-level check-pointing of C programs
  - minimize programmer effort
- MPI programs: [PPoPP 2003,ICS2003,SC2004]
  - coordination of single-process states into a global snapshot
  - non-blocking protocol: no barriers needed in program
- OpenMP programs:[EWOMP 2004, ASPLOS 2004]
  - blocking protocol
- Portable MPI checkpointing: [SC 2005]
  - Requires type information for each object created at runtime
  - Pre-compiler analyzes C programs and flags potential portability problems
  - Successfully restarted and completed 64 processor PSC Lemieux checkpoints on Cornell Windows cluster
- Ongoing work
  - program analysis to reduce amount of saved state
  - Other languages: C++

Outline

- Pre-compiler:
  - saving state at application level
- Check-pointing MPI programs
  - Non-blocking protocol
- Check-pointing OpenMP programs
  - Blocking protocol
- Portable check-pointing
  - Restart on different platform
- Ongoing work

Precompiler

- Where to checkpoint
  - At calls to *potentialCheckpoint()* function
    - Mandatory calls in main process (initiator)
    - Other calls are optional
      - Process checks if global checkpoint has been requested, and if so, joins in protocol to save state
  - Inserted by programmer or automated tool
    - Currently inserted by programmer
- Transformed program can save its state only at calls to *potentialCheckpoint()*

Application-level checkpointing: saving position in program

```
main()
{
   int a;
   if(restart)
      load LS;
      copy LS to LS.old
      jump dequeue(LS.old)
   // …
   LS.push(label1);
label1:
   function();
   LS.pop();
   // …
}
```

- Recovery structure LS (Location Stack) keeps track of function calls that could lead to potentialCheckpoint
- Code for updating LS is inserted by pre-compiler
- On recovery, function calls on LS are repeated to rebuild the stack frames
- Portable way of saving the PC

Saving Application State

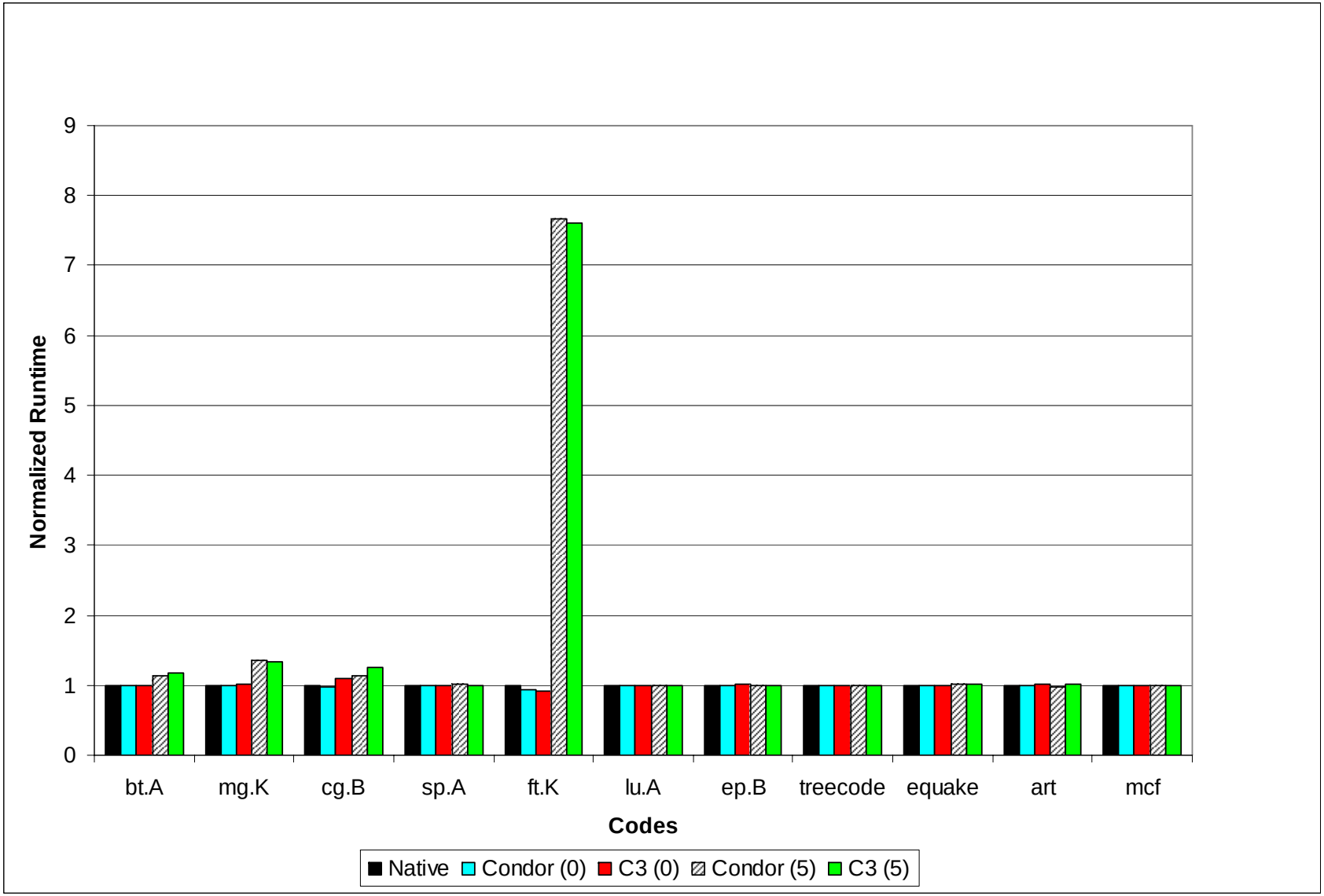
- Stack
  - Location stack (LS): track which function invocations led to place where checkpoint was taken
  - Variable Description Stack (VDS): records local variables in these function invocations that must be saved
  - On recovery
    - LS is used to re-execute sequence of function invocations and re-create stack frames
    - VDS is used to restore variables into stack
- Heap
  - special malloc that tracks memory that is allocated and freed
- Globals
  - precompiler inserts statements to save them

Sequential Experiments (vs Condor)

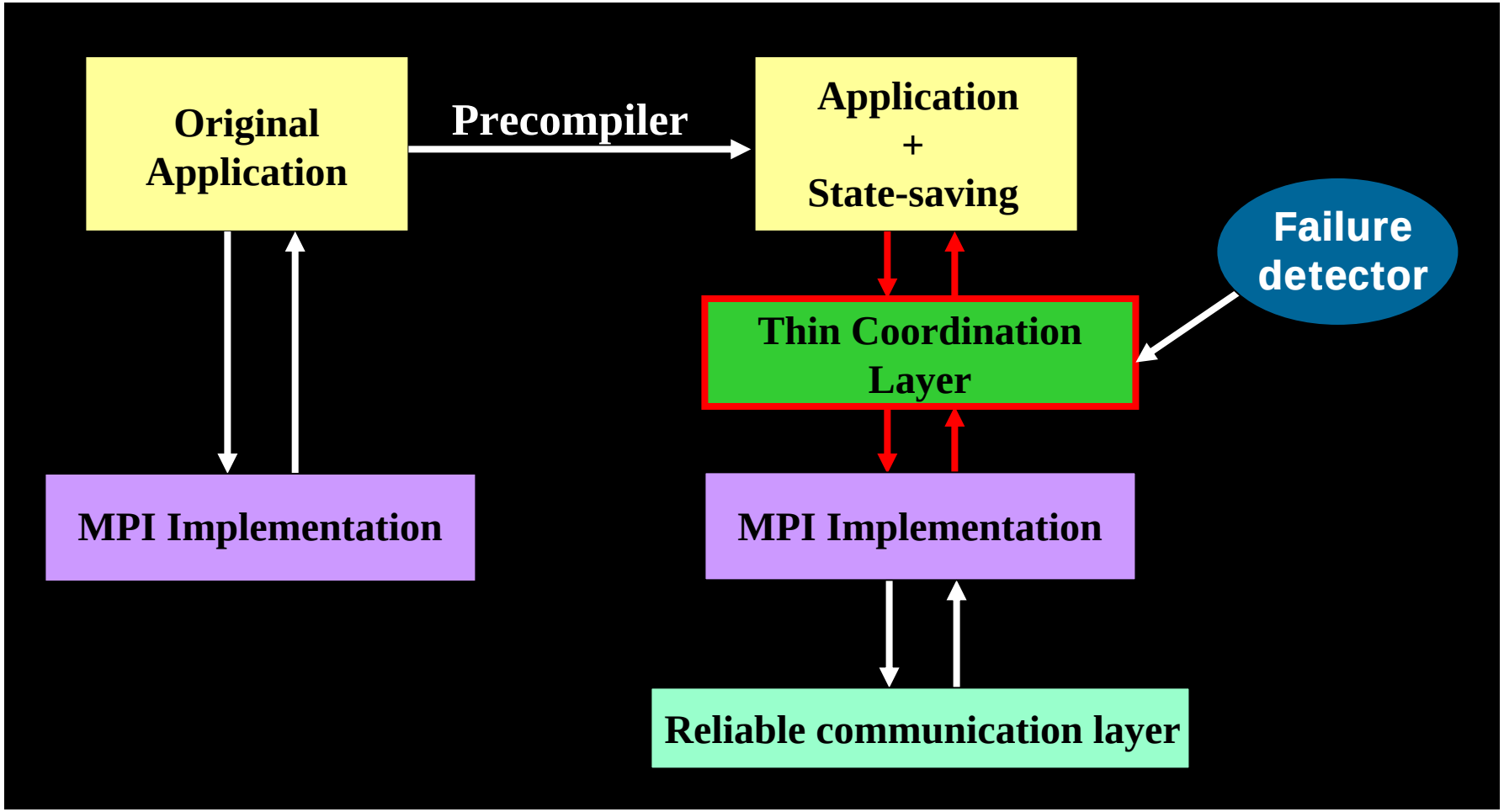
Platform	Bench	Class	Size in megabytes Condor	C^3	Reduction
Solaris	BT	A	308.85	306.39	0.80%
	CG	B	429.89	427.44	0.57%
	EP	A	3.46	1.00	71.07%
	FT	A	421.28	418.69	0.61%
	IS	A	100.45	96.00	4.43%
	LU	A	46.99	44.54	5.21%
	MG	B	436.99	435.48	0.34%
	SP	A	82.09	79.63	2.99%
Linux	BT	A	307.13	306.39	0.24%
	CG	B	428.17	427.44	0.17%
	EP	A	1.74	1.00	42.29%
	FT	A	419.43	418.69	0.17%
	IS	A	96.74	96.00	0.76%
	LU	A	45.27	44.54	1.61%
	MG	B	435.24	435.55	-0.07%
	SP	A	80.36	79.63	0.91%

- Checkpoint sizes are comparable.

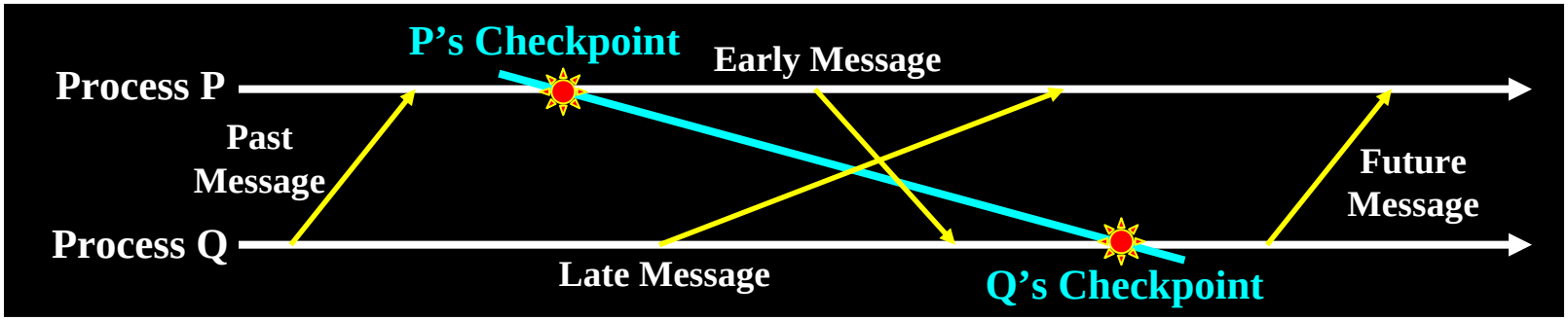
Runtime Overheads (Linux)



Check-pointing MPI programs

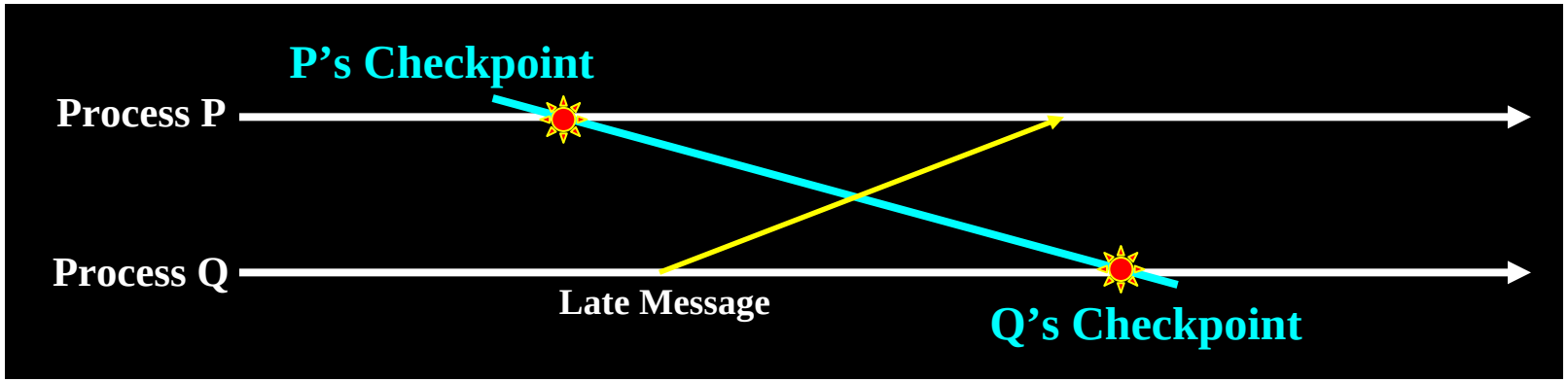


Need for Coordination



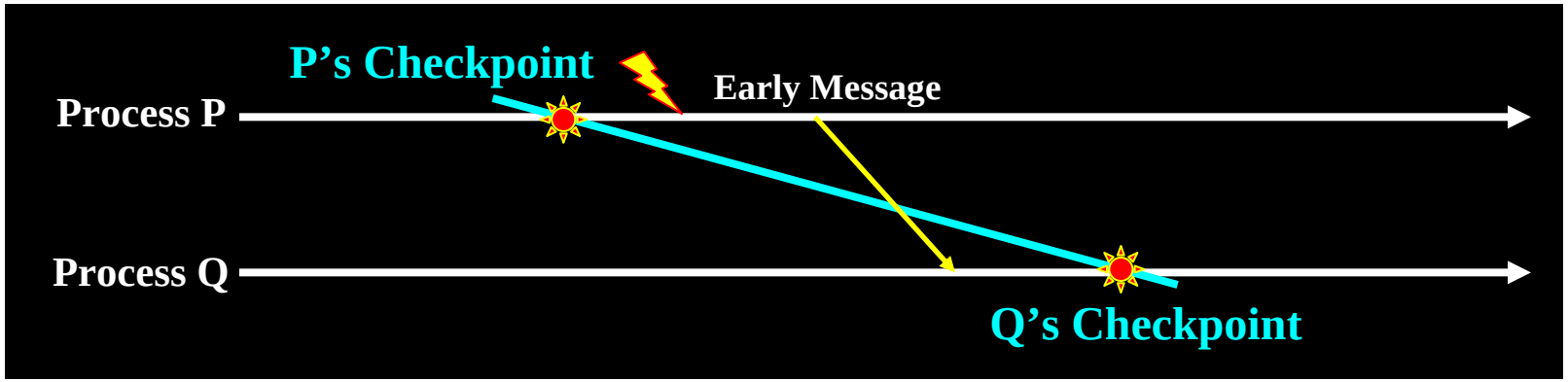
- Horizontal Lines – events in each process
- Recovery Line
  - line connecting checkpoints on each processor
  - represents global system state on recovery
- Problem with Communication
  - messages may cross recovery line

Late Messages



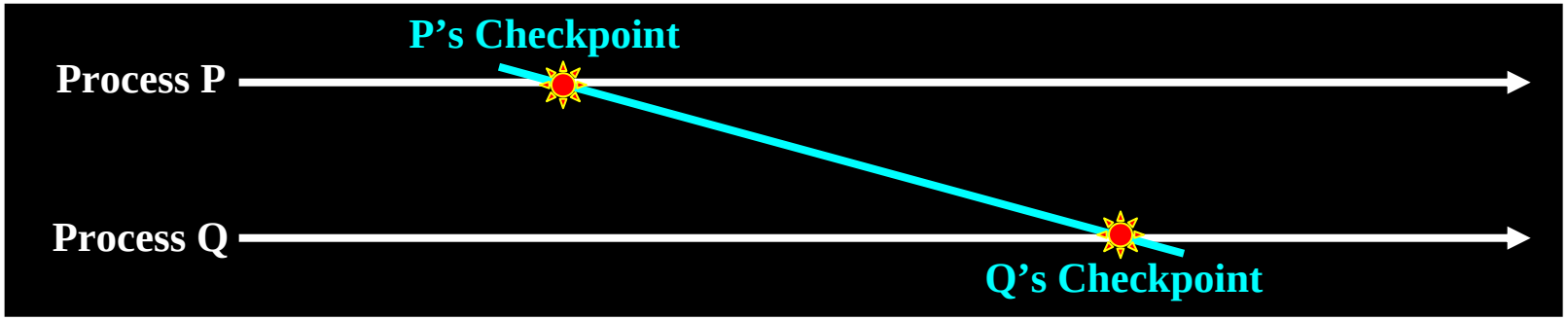
- Record message data at receiver as part of checkpoint
- On recovery, re-read recorded message data

Early Messages



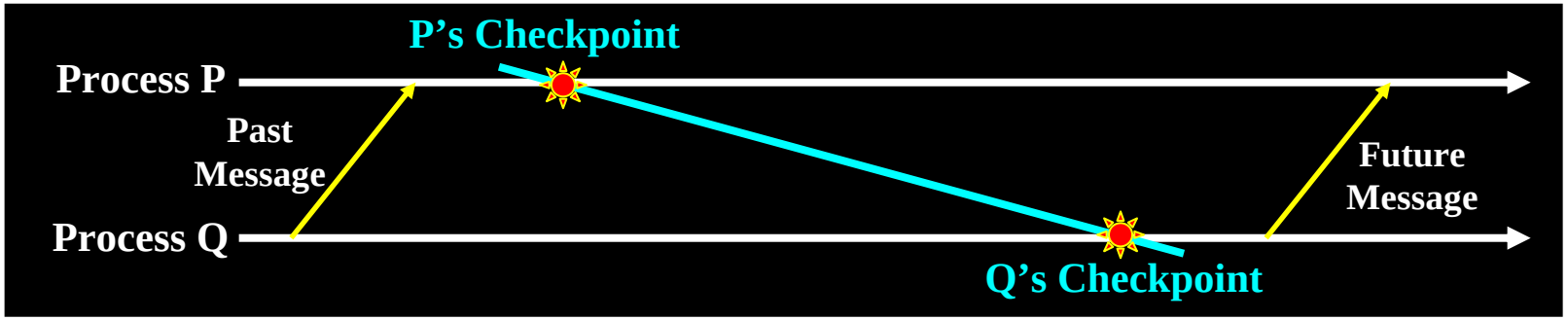
- Must suppress the resending of message on recovery
- What about non-deterministic events before the send?
  - Must ensure the application generates the same early message on recovery
  - Record and replay all non-deterministic events between checkpoint and send
  - In our applications, non-determinism arises from wild-card MPI receives

Difficulty of Coordination



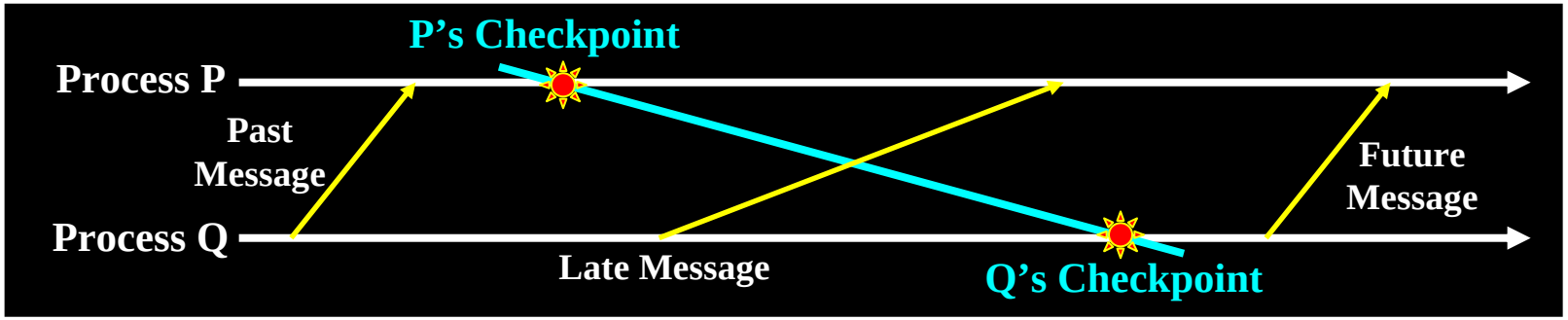
- No communication → no coordination necessary

Difficulty of Coordination



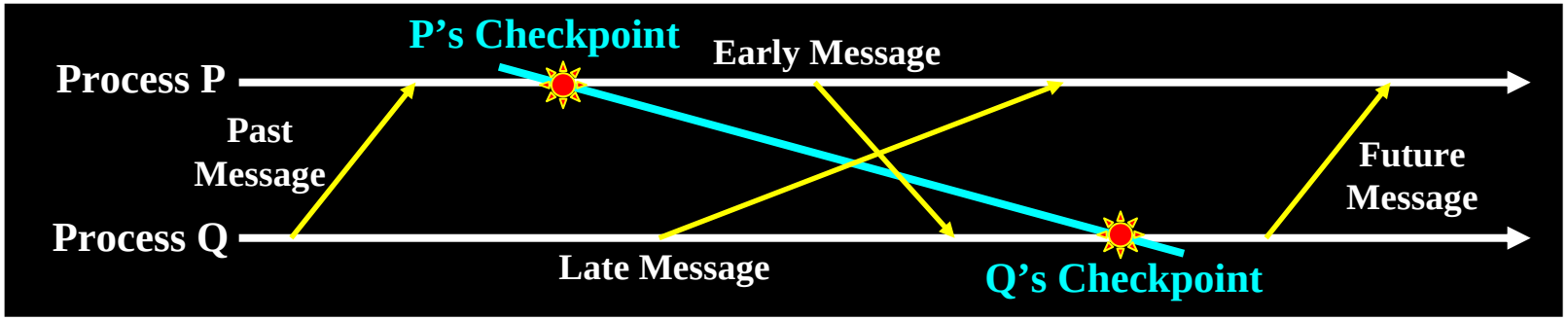
- No communication → no coordination necessary
- BSP Style programs → checkpoint at barrier

Difficulty of Coordination



- No communication → no coordination necessary
- BSP Style programs → checkpoint at barrier
- General MIMD programs
  - System-level checkpointing (eg. Chandy-Lamport)
    - Forces checkpoints to avoid early messages
    - Only consistent cuts

Difficulty of Coordination



- No communication → no coordination necessary
- BSP Style programs → checkpoint at barrier
- General MIMD programs
  - System-level checkpointing (eg. Chandy-Lamport)
    - Only late messages: consistent cuts
  - Application-level checkpointing
    - Checkpoint locations fixed, may not force
    - Late and early messages: inconsistent cuts
    - **Requires new protocol**

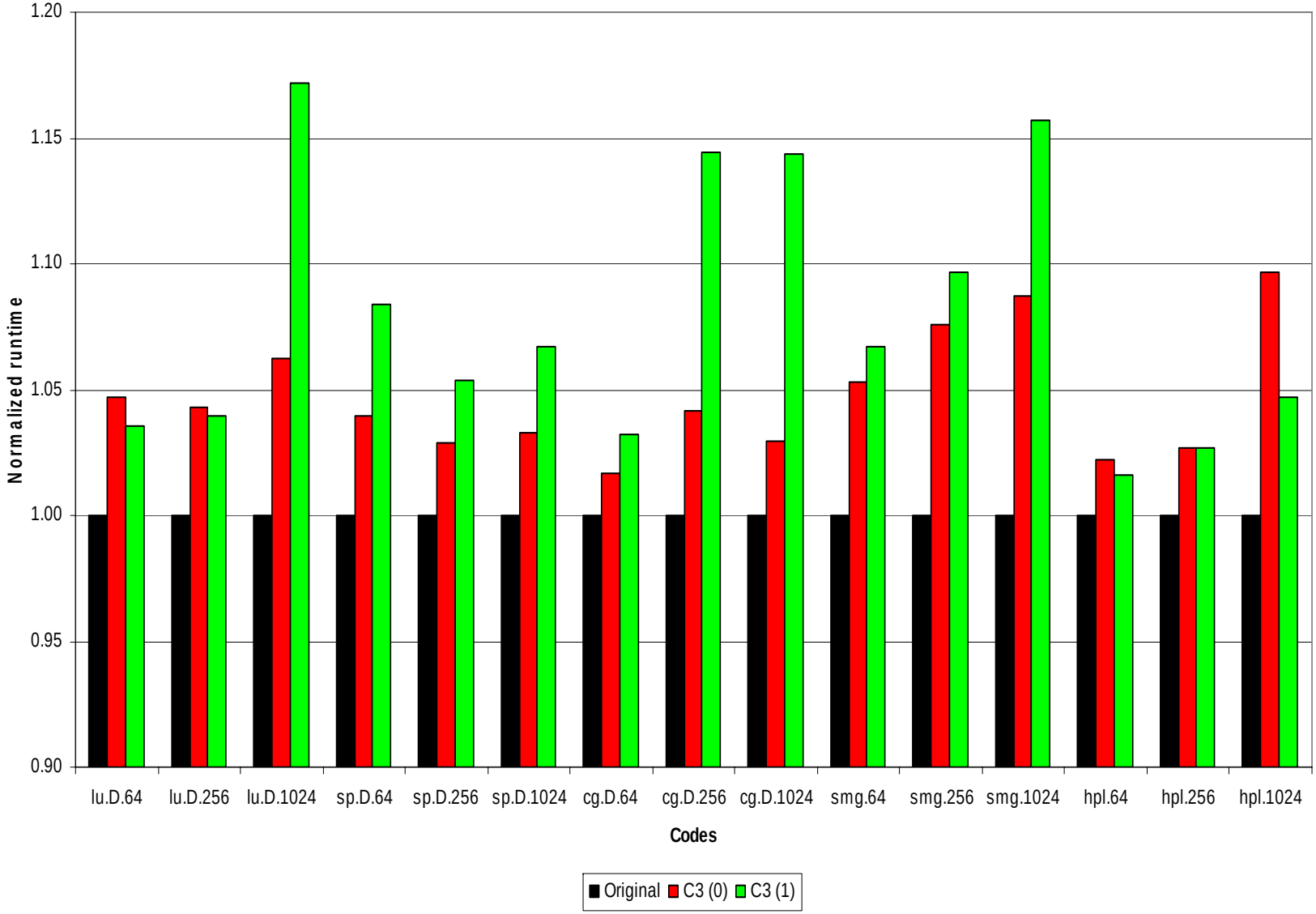
MPI-specific issues

- Non-FIFO communication – tags
- Non-blocking communication
- Collective communication
  - MPI_Reduce(), MPI_AllGather(), MPI_Bcast()…
- Internal MPI library state
  - Visible:
    - non-blocking request objects, datatypes, communicators, attributes
  - Invisible:
    - buffers, address mappings, etc.

Implementation

- Two large parallel platforms
  - Lemieux: Pittsburgh Supercomputing Center
    - 750 Compaq Alphaserver ES45 nodes
    - Node: four 1-GHz Alpha processors, 4 GB memory, 38GB local disk
    - Tru64 Unix operating system.
    - Quadrics interconnection network.
  - Velocity 2: Cornell Theory Center
    - 128 dual-processor Windows cluster
- Benchmarks:
  - NAS suite: CG, LU, SP
  - SMG2000, HPL

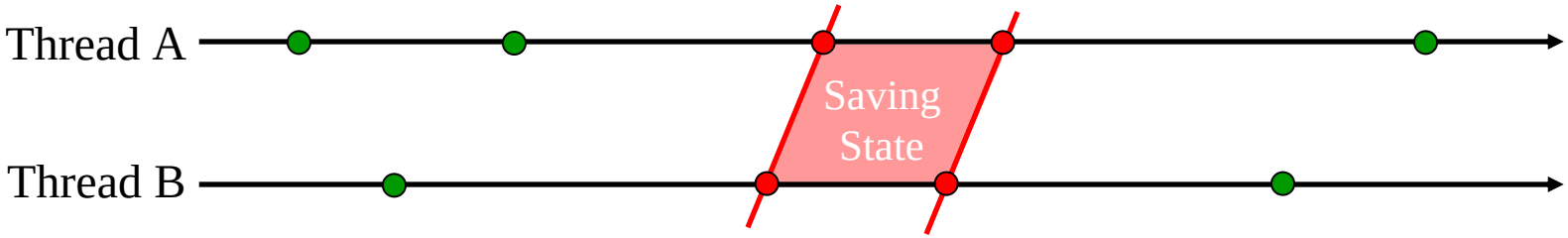
Overheads on Lemieux



Outline

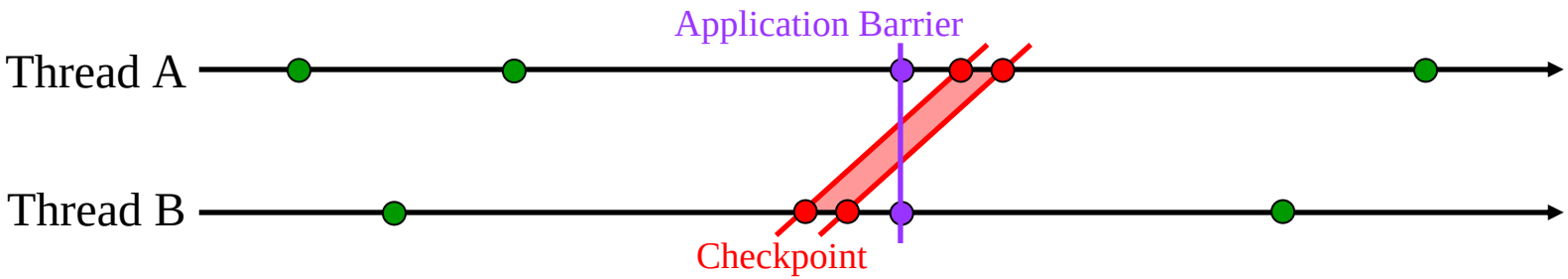
- Pre-compiler:
  - saving state at application level
- Check-pointing MPI programs
  - Non-blocking protocol
- Check-pointing OpenMP programs
  - Blocking protocol
- Portable check-pointing
  - Restart on different platform
- Ongoing work

Blocking Protocol



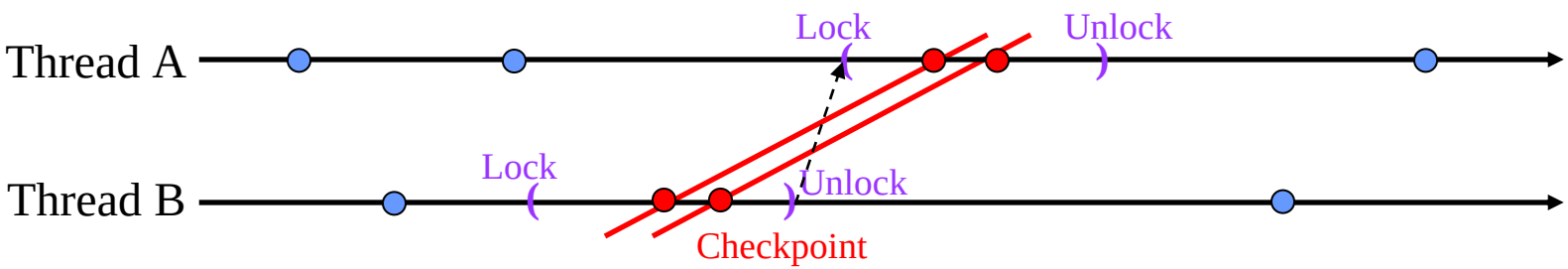
- Protocol:
  - Barrier
  - Each thread records own state
  - Thread 0 records shared state
  - Barrier

Problems Due to Barriers



- Protocol introduces new barriers into program
- May cause errors or deadlock
- If checkpoint crosses barrier:
  - Thread A reaches barrier, waits for thread B
  - Thread B reaches checkpoint, calls first barrier
  - Both threads advance
    - Thread B recording checkpoint
    - Thread A computing, possibly corrupting checkpoint

Deadlock due to locks

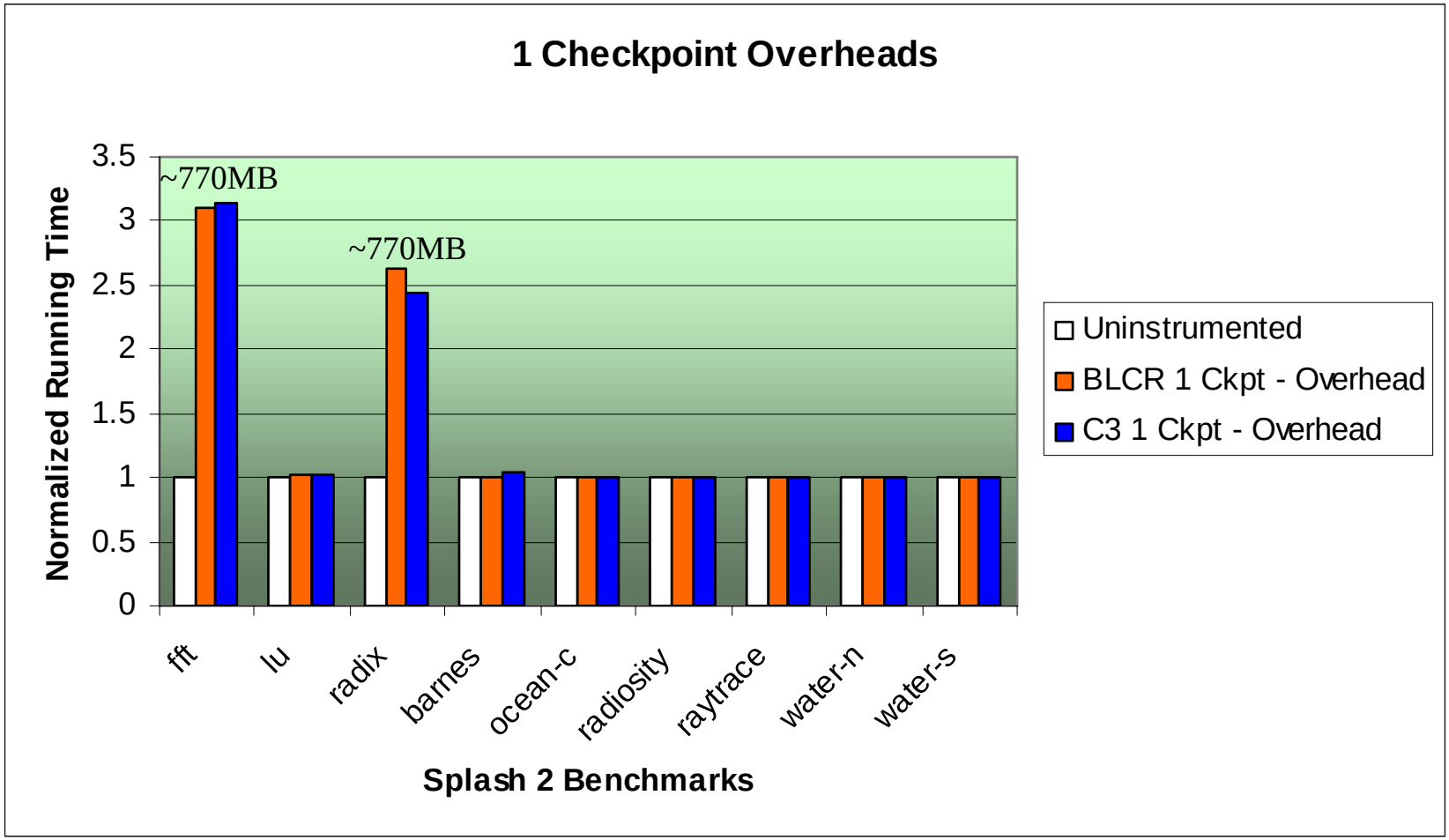


- Suppose checkpoint crosses dependence
  - Thread B grabs lock, will not release until after checkpoint
  - Thread A won't checkpoint before it acquires lock
- Since checkpoint is barrier: deadlock

Experimental Setup

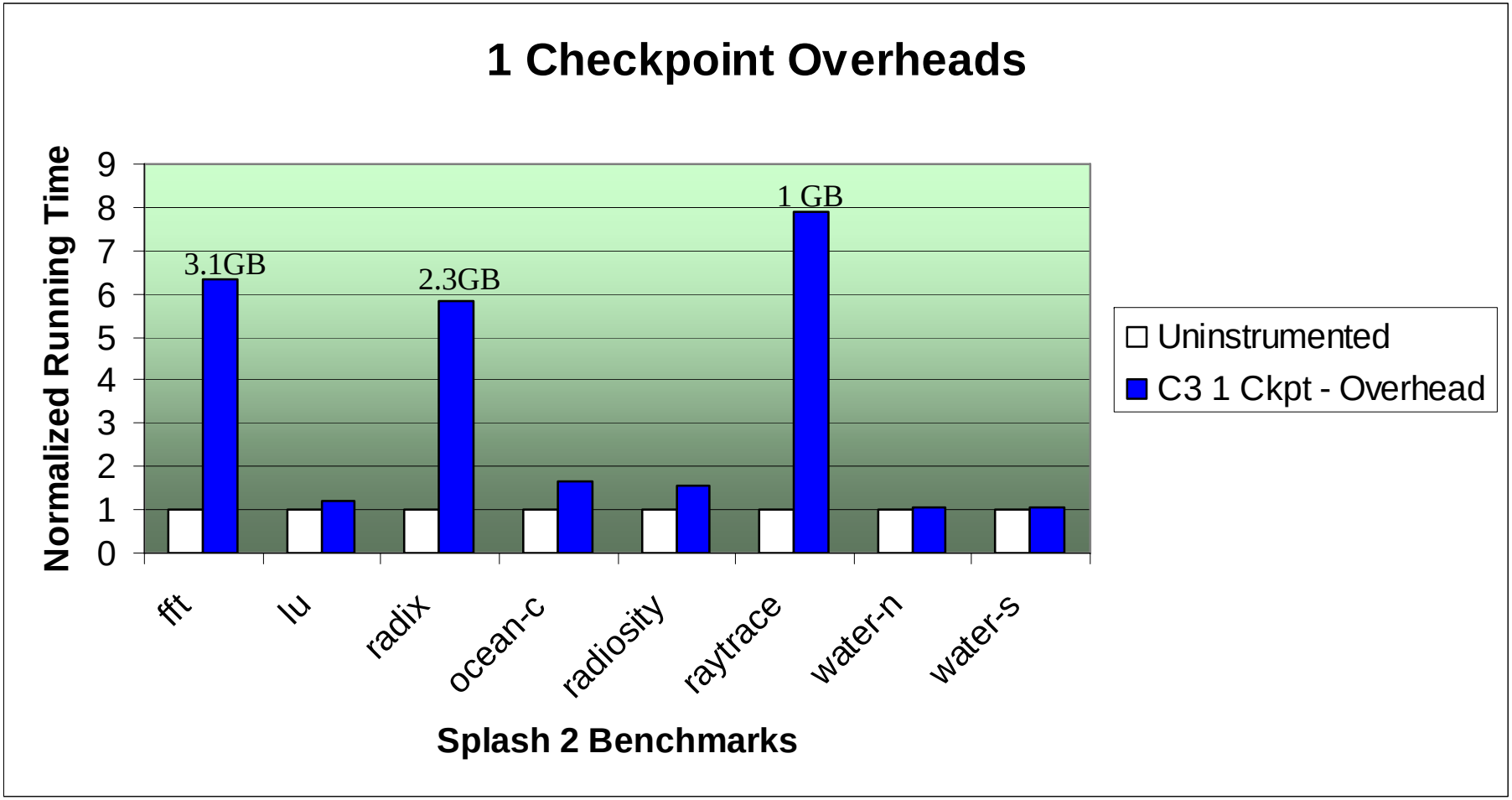
- SPLASH-2 benchmarks
  - Generic shared memory benchmarks
  - Can be specialized to any shared memory API
- Test platforms:
  - 2-way Athlon / Linux
    - Compared to BLCR
  - 4-way Alpha EV68 / Tru64
  - Windows/x86

Linux/86



Checkpoints saved to local disk

Tru64/Alpha



Checkpoints saved to network file system.

Portable Checkpointing Problem

- Migrate checkpoint to a different platform and restart there
- Dimensions of heterogeneity
  - Architecture (Pentium, Alpha, Sparc, Itanium, …)
  - OS (Windows, Linux, Solaris, TRU64 Unix, …)
  - Compilers (GNU, SUNWSPRO, Intel, HP, Visual C)
  - MPI implementations (MPIPro, MPICH, LAM)
- Approach:
  - Each data object in checkpoint must have an associated type
  - Migrating checkpoint requires translating between representations of types

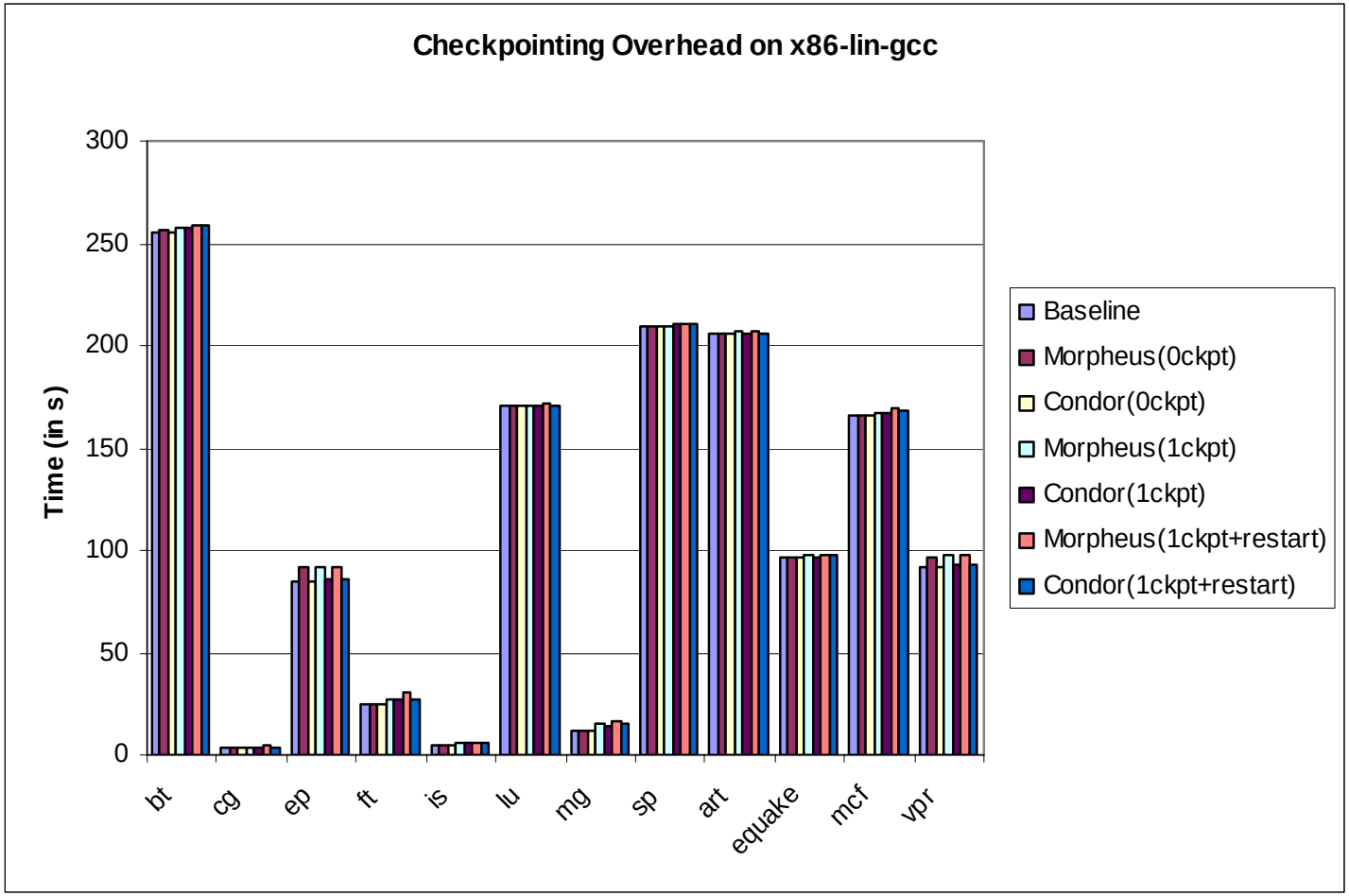
Ensuring Correctness

- C is a fairly low-level language with weak typing guarantees
- Portable checkpointing requires strong type information
- Program analysis
  - Identify constructs that could lead to errors in state translation
  - Flag warnings that allow programmers to rectify portability problems with programs

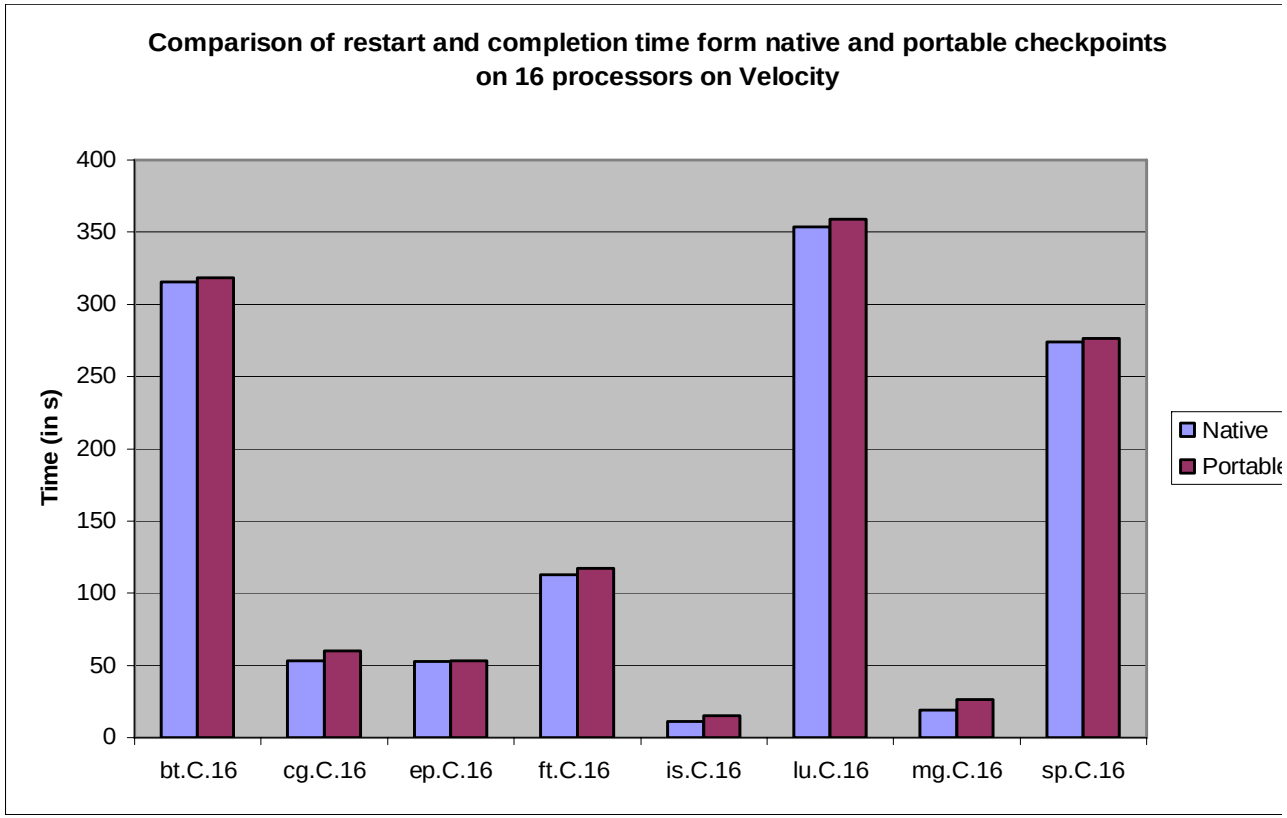
Performance Results

- Sequential
  - 3.6 Ghz Dell Dimension Desktop (Windows, Enterprise Linux)
  - Sun UltraSPARC IIIi 1060MHz(Solaris 9)
- MPI Blocking Checkpointing
  - Velocity Cluster at Cornell Theory Center
  - Lemieux Cluster at Pittsburgh Supercomputing Center

Comparing Performance with Condor on Linux



Portable vs Non-portable Restart



Ongoing work

- Integration of MPI and OpenMP sub-systems
- Integration with Pittsburgh Supercomputing Center (PSC) system for saving checkpoint data
- Compiler analysis to reduce the amount of saved state (with Radu Rugina)
  - Identify live data
  - Incremental checkpointing
  - Recomputation vs state-saving
- Other languages
  - C++

Summary

- System for CPR of MPI and OpenMP apps
  - Application-level checkpointing for C programs
  - Programs become self-checkpointing and self-restarting
- Precompiler-based single-process checkpointer
  - Minimal programmer annotations
- Novel protocols
  - Work with any single-process checkpointer
  - Portable across MPI and OpenMP implementations
- Components orthogonal
  - Can be used/applied independently
- Portable checkpointing
  - Need type information for each object created at runtime
- Overhead is low
- For more information: http://iss.cs.cornell.edu