ISS

CORNELL UNIVERSITY
COMPUTER SCIENCE

Fault Tolerance for OpenMP Programs

Greg Bronevetsky, Rohit Fernandes, Maya Haridasan, Daniel Marques, Keshav Pingali, Martin Schulz, Paul Stodghill, Peter Szwed, S. Shafat Zaman

http://iss.cs.cornell.edu

The Problem

- Shared memory machines becoming larger
  - As many as 1000 processors (NASA Ames)
- Very large computers built out of shared memory machines
  - Earth Simulator, PSC: 1000's of processors
  - Support OpenMP+MPI hybrid programs
- Program runtimes increasing
  - Weeks, months, even a year
- Program runtimes greatly exceed mean time to failure
  - ASCI, Blue Gene, Illinois Rocket Center

Our Approach

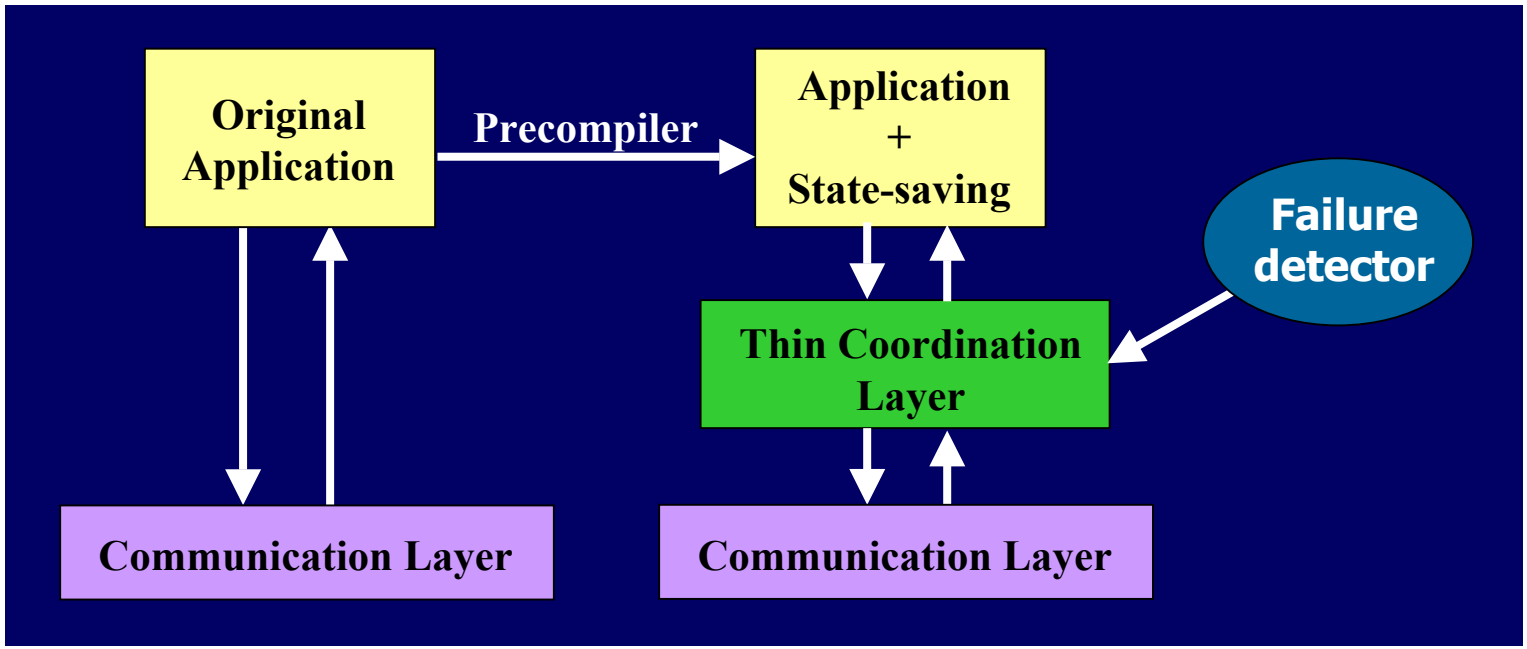
- Checkpointing
  - Regularly save state of all processes
  - When one process fails, roll everybody back to a prior state, continue computing
  - Also useful for restarting from scheduled downtime
- Application-level Checkpointing
  - System state can be very large
    - Ex: 1000 processors, 1GB RAM each = 1TB
  - Instead of saving entire system state, save just the problem state
    - Ex: Alegra application state = 5% of system state
  - Checkpoint locations placed statically in program source code

Cornell Checkpointing Compiler (C^3)
Goals

- Takes programs as input
  - Either sequential, MPI or OpenMP
- Produces equivalent fault tolerant programs

- Key Components
  - Saving the state of each process
  - Coordinating these states across the network
- Compiler techniques used to keep overhead low by minimizing the amount of state saved

Cornell Checkpointing Compiler (C^3)



Sequential and MPI solutions already developed:

Bronevetsky, Marques, Pingali, Stodghill

- PPoPP '03: "Automated Application-level Checkpointing for MPI Programs"
- ICS '03: "Collective Operations in an Application-level Fault Tolerant System"

Current Work

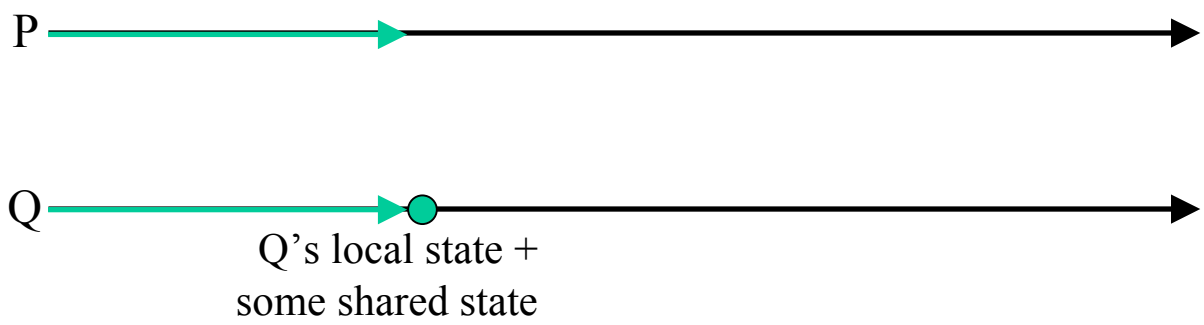
- Sequential
  - Developing compiler analyses to minimize checkpoint size
- MPI
  - Release implementation to be finished by end of year
  - Will deal with all features of MPI: collectives, non-blocking, datatypes, etc.
- Shared Memory….

Shared Memory Checkpointing

- Developing checkpointers for shared-memory programs

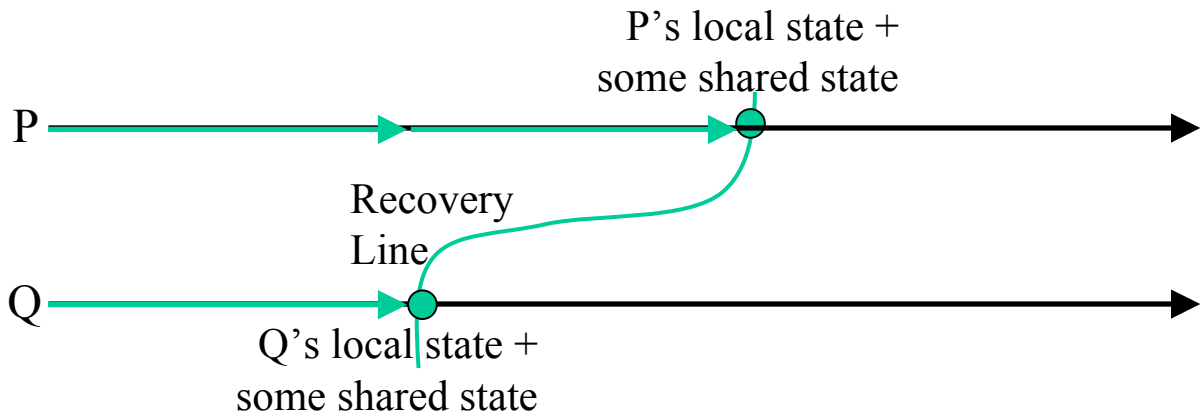
- Components
  - Program transformation to have it save Local State, Shared State, Execution Context
  - Coordination protocol to turn individual thread records into a recoverable global snapshot

Checkpointing Shared Memory



- Q reaches Checkpoint location

Checkpointing Shared Memory



- Q reaches Checkpoint location
- P reaches Checkpoint location

Saving Local State

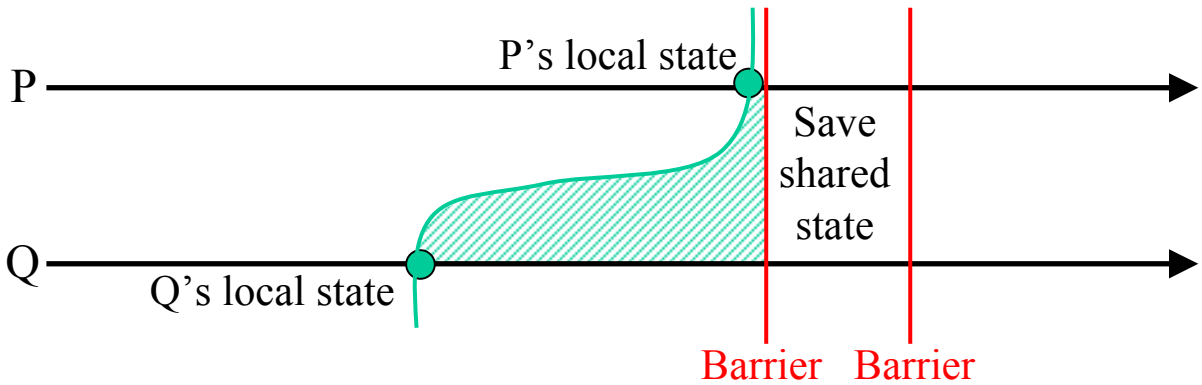
- Globals
  - Preprocessor knows the globals
  - Inserts statements to explicitly save them
- Call Stack, Locals and Program Counter
  - Maintain a separate stack which records all functions that got called and the local vars inside them
  - On recovery, call the functions again in same order
  - Entry into parallel regions done the same way

Saving Shared State

- Stack data
  - Preprocessor knows which stack variables are shared (from omp directives)
  - Inserts statements to explicitly save them
- Heap data
  - Use special malloc() that keeps track of currently allocated memory

Coordination

- A blocking protocol



- Get snapshot of shared and local states as they were at some point in time
- Application locks and barrier pose problems
  - Efficient solutions available

Moving into the Future

- Implementation of checkpointer with blocking protocol almost complete
  - Preliminary results show 1.4%-4.5% overhead
- Near future:
  - Extend to more OpenMP constructs
    - Currently only support parallel, barriers, locks, flush
  - Blocking protocols force threads to wait. Can we recover if we had states from different points in time?
    - A large space of possible protocols that can do this
    - Which can be practically implemented with OpenMP? A question for the compunity.

Conclusion

- Have developed fault tolerance solutions for sequential and MPI programs
  - Working on optimizations and release implementation
- Working on Shared Memory checkpointer
  - Blocking protocol almost complete
  - Nonblocking protocols in development

Any comments/suggestions appreciated!