

Fault Tolerance at Extreme Scales

Dorian Arnold

University of New Mexico

(funded by and in collaboration w/ Sandia National Lab)

MY ICL Tenure

- ▶ January 1999 – August, 2001
- ▶ NetSolve Project
- ▶ ICL Collaborators:
 - Henri, Sudesh, Sathish, Jakob, Thara, Michelle, Dieter, Keith S., Tsinghua, Victor, Susan, Shirley, Keith M., Ganapathy, Nathan, David and, of course, Jack

So ... How's the Family?



So ... How's the Family?



So ... How's the Family?



... And What've You Been Up To?

- ▶ PhD Student, U. of Wisconsin, '01 - '08
 - Scalable, reliable communication infrastructure
 - Scalable, lightweight tools and applications
- ▶ Asst. Professor, U. of New Mexico, since '09
 - Carry-over from dissertation work (naturally)
 - **Fault Tolerance**
 - Autonomous Infrastructure
 - Virtualization for HPC
 - Thin Computing

Today, I'll Talk About ...

- ▶ Current and future agenda for
 - Fault-tolerance
 - Compression for improved checkpoint/restart efficiency

Fault Tolerance @ Scale

- ▶ You've heard the story:
 - **Systems are growing**: millions of cores? billion-way concurrency
 - System **MTBF will decrease** significantly (O(minutes)?)
 - Checkpoint/restart overheads will **reduce application efficiency** to intolerable levels
 - We must reduce the impact of MTBF on **JMTTI** while maintaining **sufficient application efficiency**

We need better fault-tolerance mechanisms
with lower overheads!

Current Fault-tolerance Studies

- ▶ Checkpoint compression
 - Viability demonstration
 - Impact on application time to solution
- ▶ Incremental Checkpointing
- ▶ Hybrid approach: replication + checkpoint/restart
 - What's the pay-off point?
 - Is partial replication worth doing?

Checkpoint/Restart

- ▶ Checkpoint/restart combines space & time redundancy
 - Periodically **save snapshots** of application state
 - **Rollback** to recent checkpoint upon application failure
- ▶ Checkpoints must be committed to **stable storage**
 - Storage persistent relative to the given fault-model

Checkpoint/Restart Optimizations

- ▶ Hide/reduce (perceived) commit latency
 - Reduce the time to commit checkpoint data
 - E.g., concurrent, diskless, hierarchical, remote checkpointing
 - Checkpointing file systems
- ▶ Reduce checkpoint sizes
 - Reduce the information in a checkpoint
 - E.g., memory exclusion, incremental checkpointing
 - Reduce the data in a checkpoint
 - E.g., hash-based incremental checkpointing

Related Checkpoint Compression Work

▶ Checkpoint compression

- To facilitate “main-memory checkpointing” [Plank and Li ‘94]
 - parallel environment with secondary storage contentions
 - demonstrated viability with compression-factor > 19.3%
- Compiler-based checkpoint compression [Li and Fuchs, ‘90]
 - not viable if compression-factor < 100% (impossible)
- Hardware-based [Moshovos and Kostopoulos, 2004]

▶ Differential compression

- Compressing incremental “differences” [Plank et al, 1995]

▶ Indirectly related

- Compression for **data migration** [Lee et al, 2002]
- Frameworks for **in situ compression**
 - FUSECompress, stackable FS, ADIOS, functional partitioning

Methodology

Empirical approach based on

- Checkpoint compression **viability model**
- and
- Representative applications
 - from the **Mantevo project**
 - **LAMMPS** molecular dynamics code
 - Representative compression algorithms
 - popular **Linux compression utilities**
 - Representative checkpoint generator
 - **Berkeley Lab Checkpoint/Restart (BLCR)**

Compression Factor (Data Savings)

$$1 - \frac{|compressed_checkpoint|}{|checkpoint|}$$

E.g., 100 MB file that compresses to 20 MB → 0.8 compression factor

Checkpoint Compression Viability Model

$$t_{comp} + 2t_{cc} + t_{dcomp} < 2t_{uc}$$

$$\frac{c}{r_{comp}} + 2\frac{(1-\alpha)c}{r_{commit}} + \frac{c}{r_{dcomp}} < 2\frac{c}{r_{commit}}$$

$$\frac{2\alpha \times r_{comp} \times r_{dcomp}}{r_{comp} + r_{dcomp}} < r_{commit}$$

t_{comp} : compression latency

t_{dcomp} : decompression latency

t_{cc} : time to read/write compressed file

t_{uc} : time to read/write uncompressed file

c : original checkpoint size

α : compression factor

r_{comp} : compression speed

r_{dcomp} : decompression speed

r_{commit} : checkpoint read/write rate


The Mantevo Project

- ▶ Proxies for larger, representative scientific applications
 - For HPC application analysis, prediction and improvements
 - Open source
 - <https://software.sandia.gov/mantevo/index.html>
- ▶ Mini-applications (mini-apps)
 - Small, self-contained programs that embody essential performance characteristics of key applications
 - Implicit, unstructured PDEs, explicit dynamics, molecular dynamics, circuit simulation

The Mini apps

- ▶ **HPCCG v0.5**
 - conjugate gradient code for 3D chimney domain
- ▶ **pHPCCG v0.4**
 - similar to HPCCG w/ arbitrary scalar and integer types and different sparse matrix data structures
- ▶ **miniFE v1.0**
 - finite element generation assembly and solution
 - unstructured grid
- ▶ **phdMesh v0.1**
 - parallel, heterogeneous, dynamic unstructured mesh library
- ▶ All except phdMesh are implicit finite element codes
- ▶ phdMesh is explicit finite element code

LAMMPS

- ▶ Large-scale Atomic/Molecular Massively Parallel Simulator
 - ▶ Molecular dynamics code
 - ▶ Built-in checkpointing support
- 

Compression Tools

Compression Tool	Brief Description
7zip	Lempel-Zif-Markov chain algorithm, dictionary-based compression
zip	LZ77 algorithm and Huffman coding
bzip2	Burrows-Wheeler Transform, block-sorting for better compression
pbzip2	multithreaded bzip2
rzip	large buffers for long distance redundancy, bzip2 backend

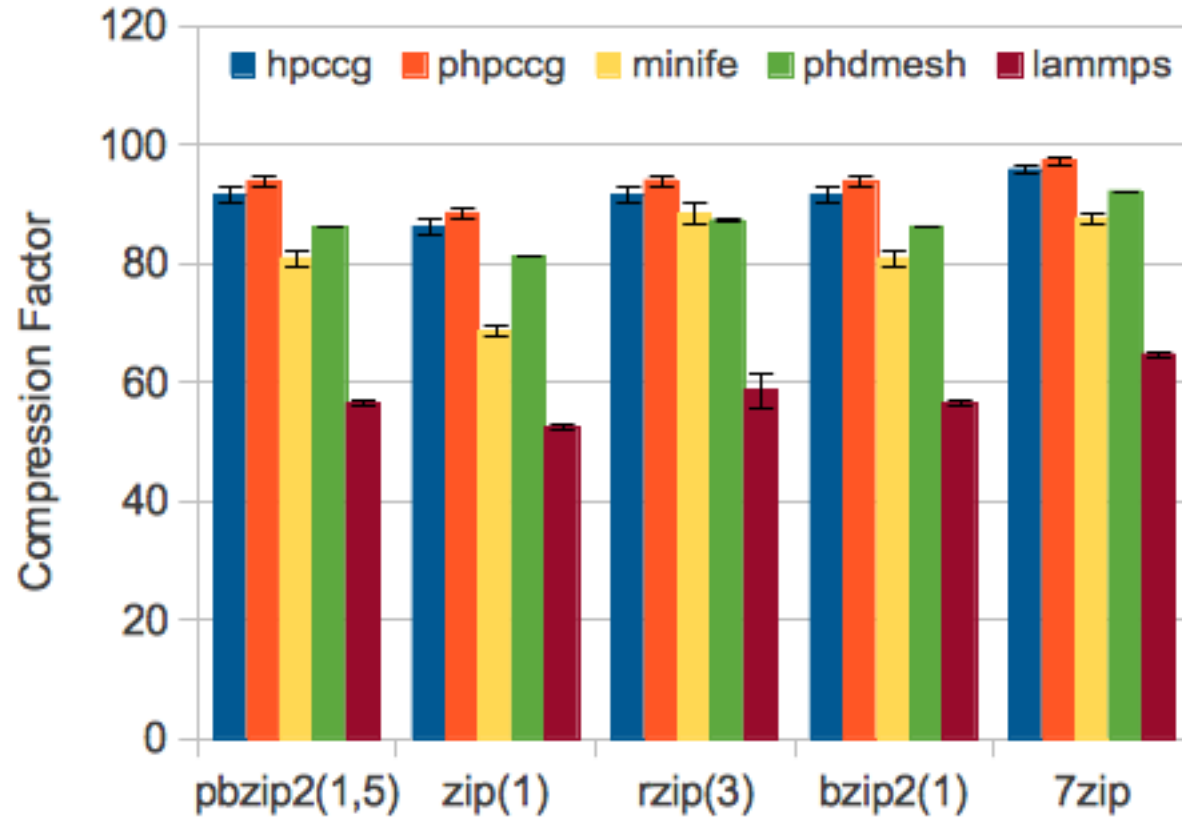
BLCR Checkpoint Library

- ▶ Berkeley Lab Checkpoint/Restart
- ▶ System-level checkpoint library
- ▶ (Per process) checkpoints stored to files
- ▶ OpenMPI framework can leverage BLCR

Testing Environment

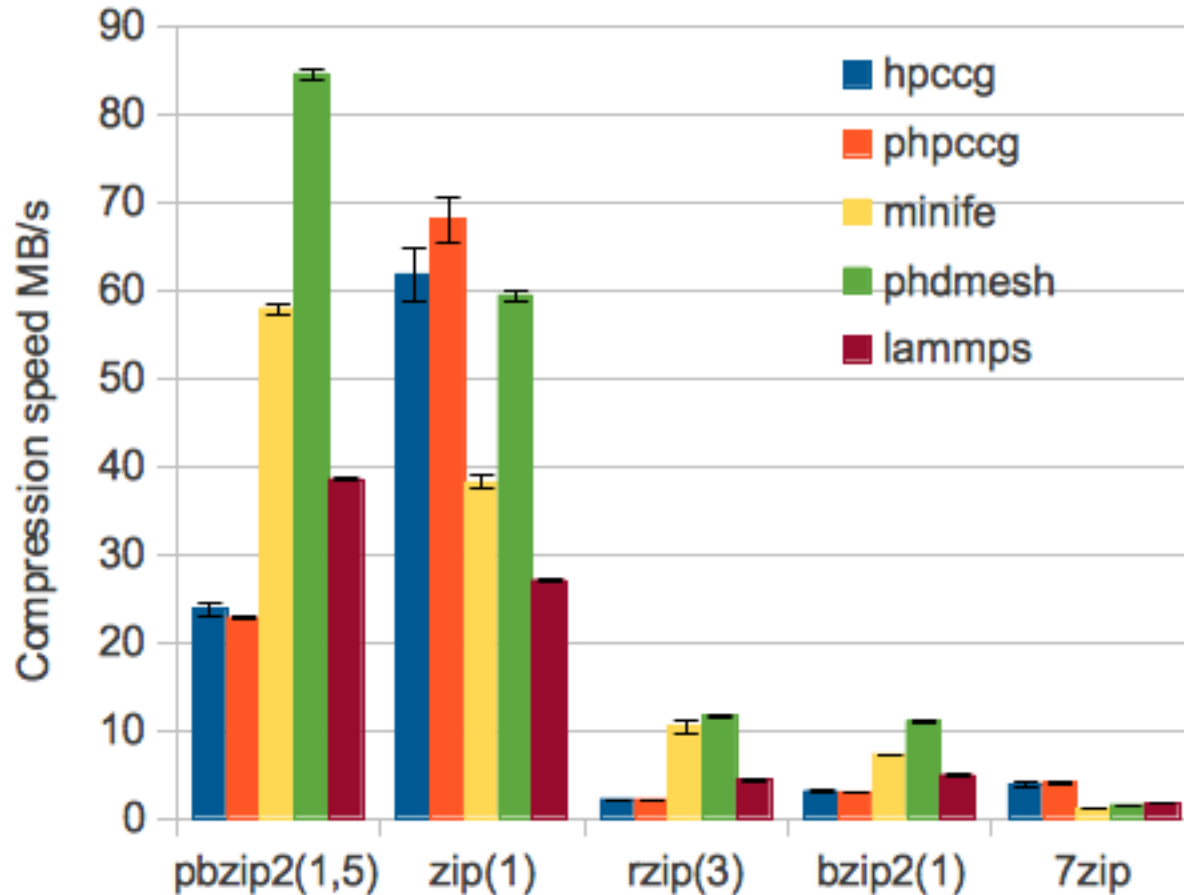
- ▶ 64-bit 4 core 2.33 GHz Intel Xeon processor
- ▶ 2 GB of memory
- ▶ Linux 2.6
- ▶ BLCR 0.8.2
- ▶ Compression tools: zip 3.0, rzip 2.1, bzip2 1.0.5, pbzip2 1.0.5, p7zip
- ▶ 5 checkpoints uniformly over each run

Results: Compression Factor

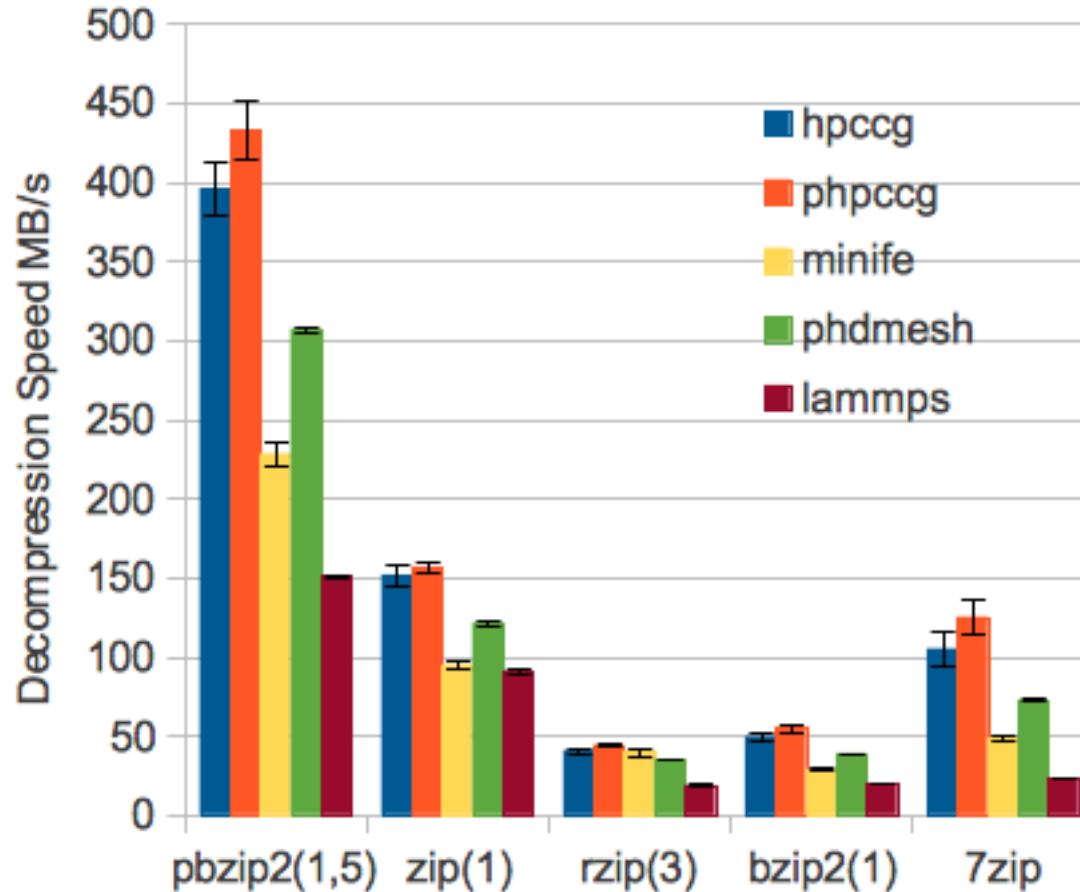


(checkpoint size varies from ~300-400MB)

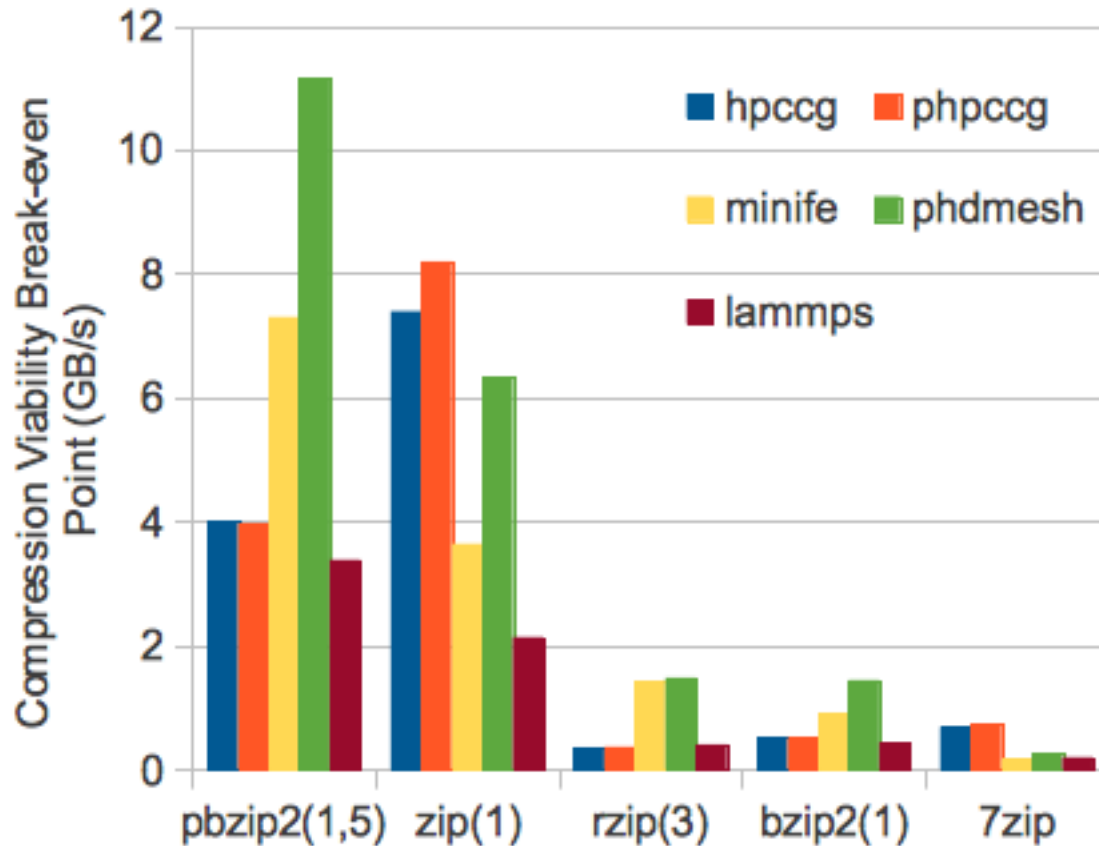
Results: Compression Speed



Results: Decompression Speed



Checkpoint Compression Viability



Discussion of I/O Bandwidths

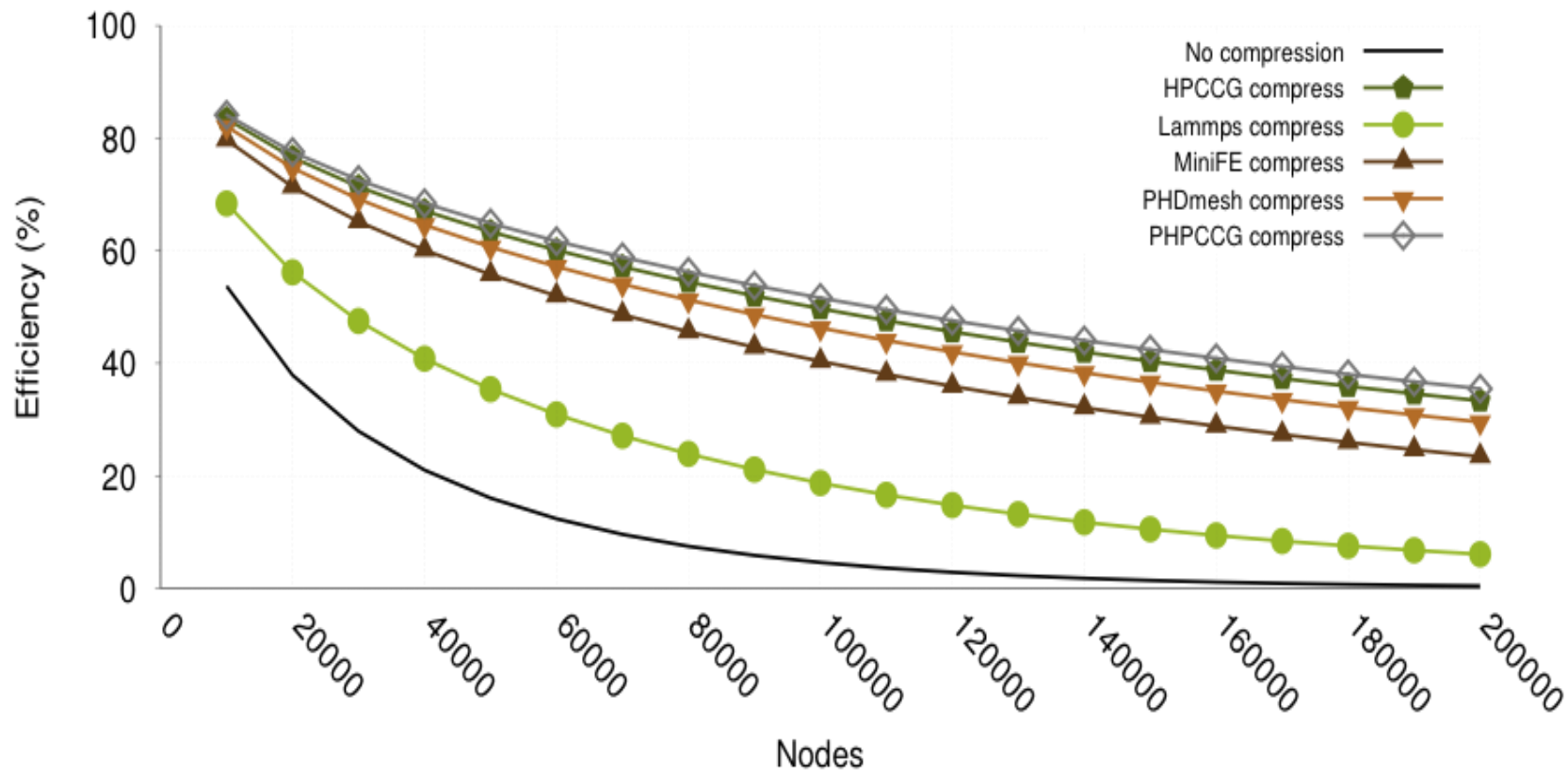
- ▶ Best observable (that we've found)
 - 1 MB/s/per process (Argonne's BG/P)
 - Scaled to 32K then drops off significantly
- ▶ ORNL Jaguar's peak
 - 5.3 MB/s per node
- ▶ LLNL Dawn's peak
 - 2 MB/s per node
- ▶ Worst case viability break-even: ~3 GB/s/process

Checkpoint Compression and Application Efficiency

Performance model for application time to solution

- Based on Daly's high-order model
 - Nodes fail independently and exponentially
 - Input MTBF, checkpoint commit time, recovery time, number of nodes and failure-free time to solution
- Modified to integrate compression/decompression
- Assumptions:
 - 5 year node MTBF
 - 1 process/node
 - 1 MB/s per node (process) I/O bandwidth
 - 2GB per node (1/3 of that checkpointed)

The Impact of Checkpoint Compression



Paths forward for Compression

- ▶ More real applications
 - How representative are the mini-apps for memory footprints?
- ▶ Improving compression speeds
 - E.g., gpu acceleration or multicore parallelism
 - Compress on region-by-region basis
- ▶ Understand what **memory footprint features** impact compression performance

Paths forward for Fault-Tolerance

- ▶ Numerous other related efforts
 - Replication/checkpoint/restart (Ferreira et al)
 - Hash-based incremental checkpointing (Ferreira et al)
 - Partial replication impact (Stearley et al)
 - Fault-tolerance advisory toolkit (in the works)
- ▶ Wouldn't it be nice to have a robust fault-tolerance framework to accelerate such studies?
 - And to use in practice!

Questions?

Future Agenda

- ▶ Fault-tolerance Advisory Toolkit:
 - What checkpoint interval should I use?
 - Should I use compression? Which algorithm?
 - Should I also use replication? What level?
 - Given a problem, a set of resources, what is the optimal fault-tolerance configuration to get me a particular time to solution?
 - Given a problem and a desired time to solution, how much resources do I need to allocate?
 - How do our methods map to uncoordinated checkpointing and message logging?