Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver

Mohsin Ali and Peter Strazdins*,
Computer Systems Group,
Research School of Computer Science,
The Australian National University
(with James Southern and Brendan Harding)

(slides available from http://cs.anu.edu.au/~Peter.Strazdins/seminars)

The 15th Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2014),
Phoenix, Arizona, 23 May 2014

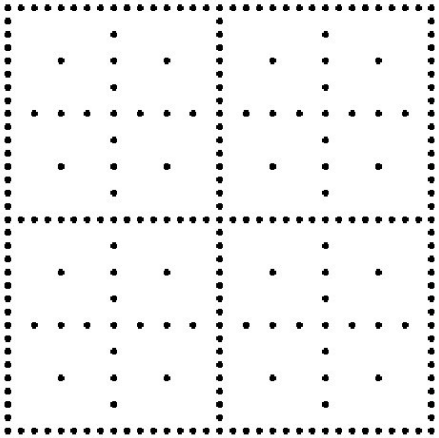
THE AUSTRALIAN NATIONAL UNIVERSITY

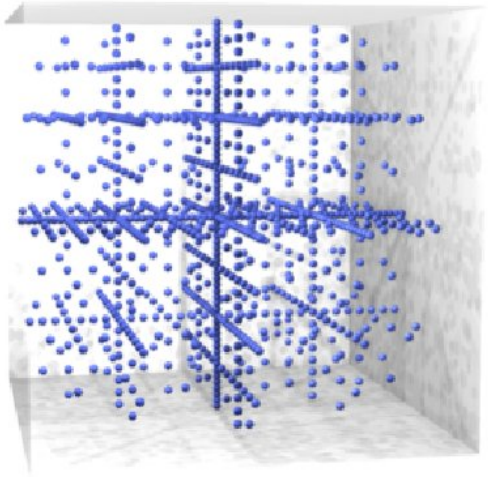
1 Talk Overview

- background: solving PDEs via sparse grids with combination technique
- recovery methods: replication/re-sampling and alternate combination technique
- application recovery via User Level Fault Mitigation (ULFM) MPI
  - detecting / identifying failed processes
  - failed process and communicator reconstruction
  - data recovery
- experimental results
  - failure identification and communicator reconstruction time
  - failed grid data recovery overhead
  - approximation error
  - scalability
- conclusions and future work

2 Background: Sparse Grids

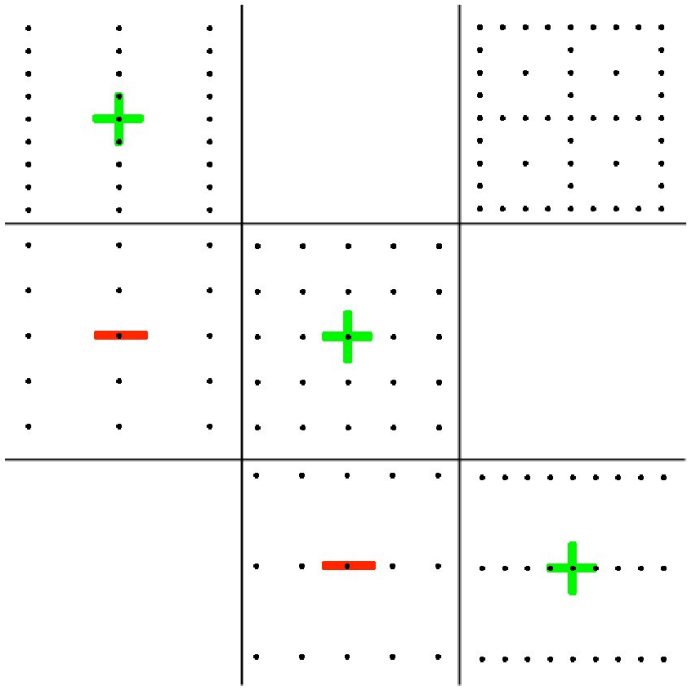
- introduced by Zenger (1991)
- for (regular) grids of dimension d having uniform resolution n in all dimensions, the number of grid points is n^d
  - known as the *curse of dimensionality*
- a sparse grid provides fine-scale resolution
- can be constructed from regular sub-grids that are fine-scale in some dimensions and coarse in others
- has been proved successful for a variety of different problems:
  - good accuracy for given effort (over single higher resolution grid)
  - various options for fault-tolerance!

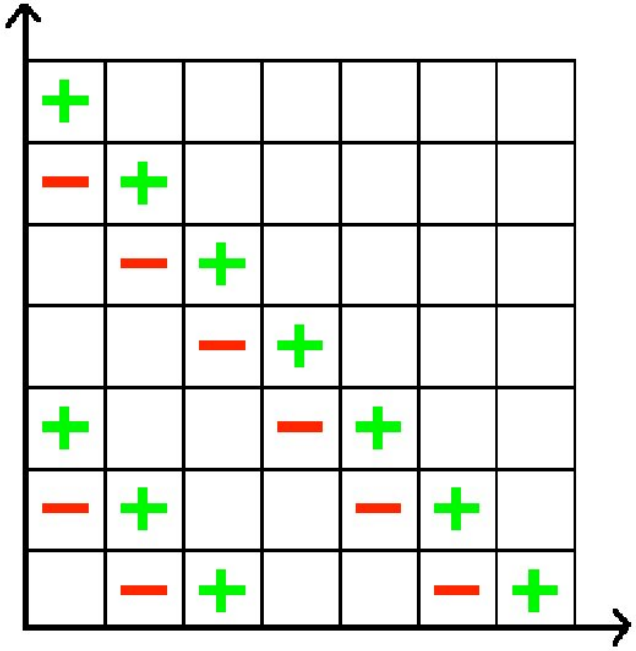




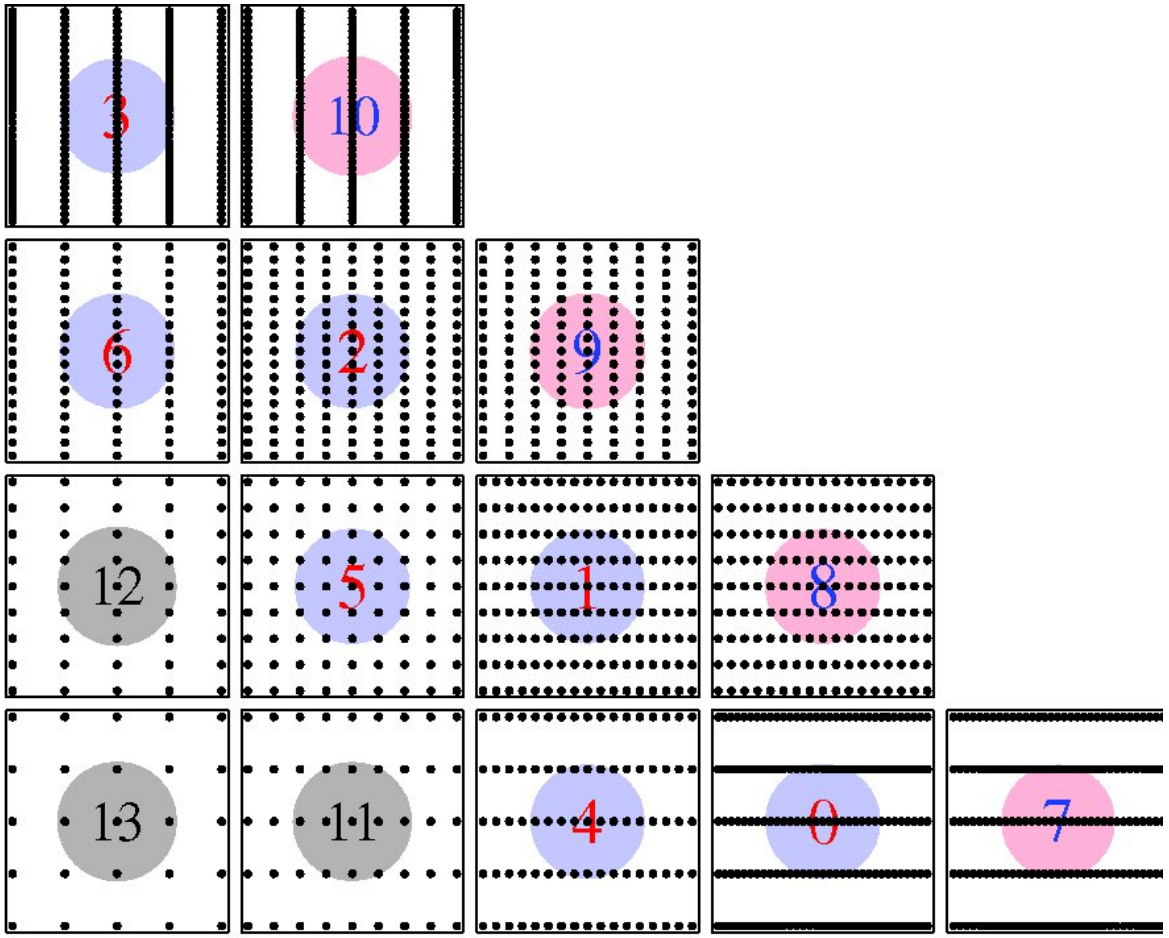
3 Background: Combination Technique for Sparse Grids

- computations over sparse grids may be approximated by being solved over the corresponding set of regular sub-grids
  - overall solution is from 'combining' sub-solutions via an inclusion-exclusion principle (complexity is still $O(n \lg(n)^{d-1})$)
- for 2D at 'level' $m = 4$, combine grids $(3,1)$, $(2,2)$ $(1,3)$ minus $(2,1)$, $(1,2)$





4 Two-dimension PDE Solver: Recovery Methods



- replication/re-sampling:

  recover grids 0–3 from duplicate grids 7–10;

  recover grids 4–6 via resampling from grid 0–3

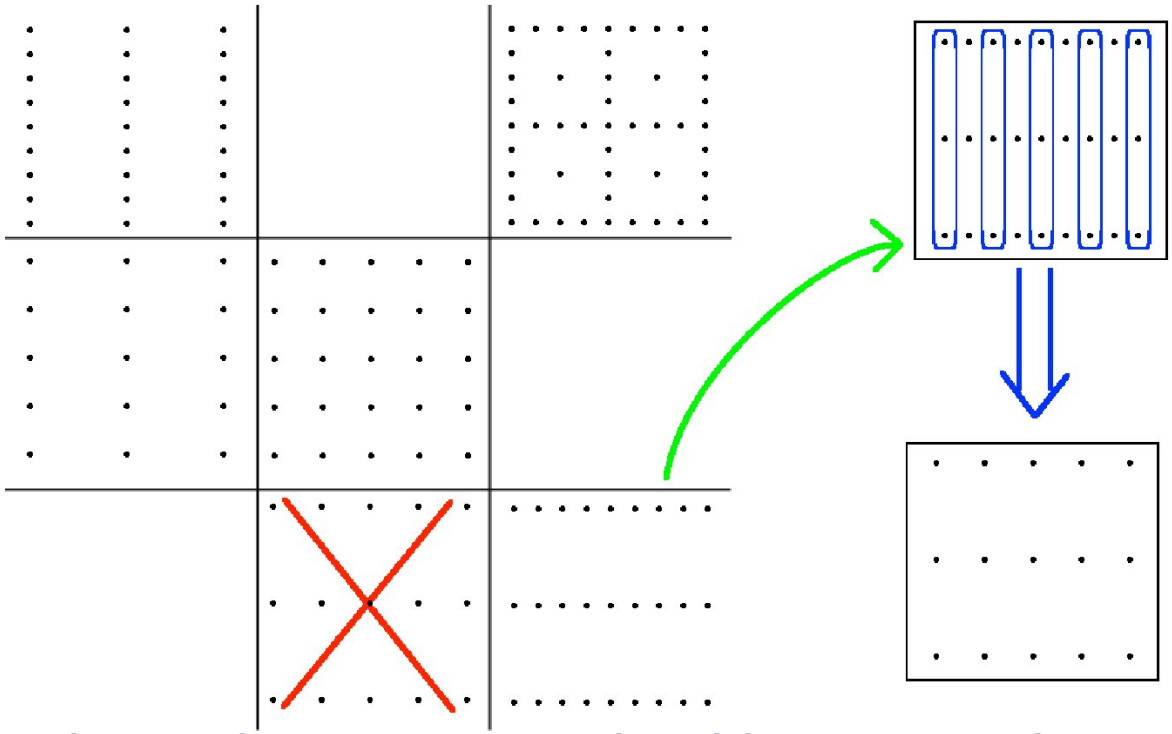
- alternate combination:

  lost grid $g \in \{0..6\}$ is ignored; final result (sparse grid) is constructed via a subset of $\{0..6, 11..13\} - \{g\}$

5 Two-dimension PDE Solver: Details

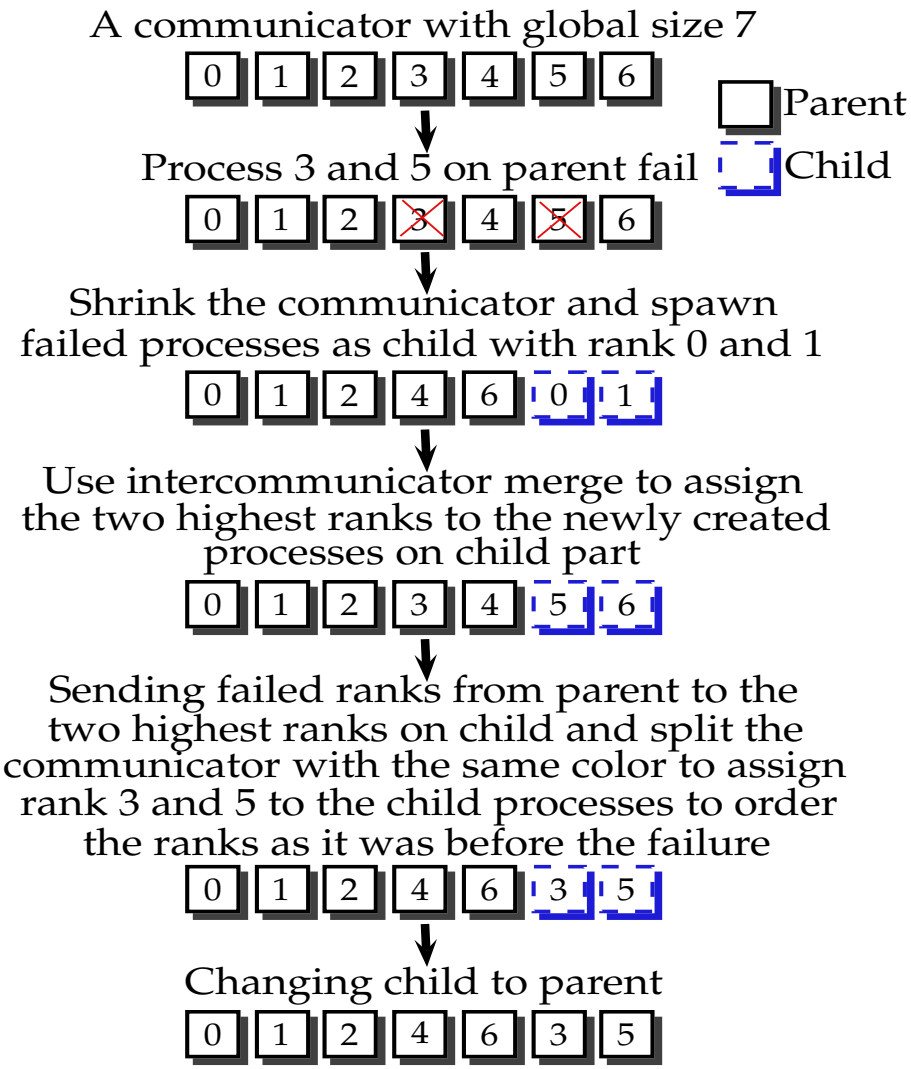
Resampling from a coarser grid:

- always possible
- highly efficient
- possibly a small loss of accuracy



- PDE (2D advection) evolved over time T over each grids at same time on separate sets of processes (same Δt); apply combination technique
  - no. of processes halves going down to next diagonal (load balance)
  - 1–2 faults injected during this time
  - whole grid of the failed process is discarded

6 Fault Recovery Procedure: Detect Failed Process



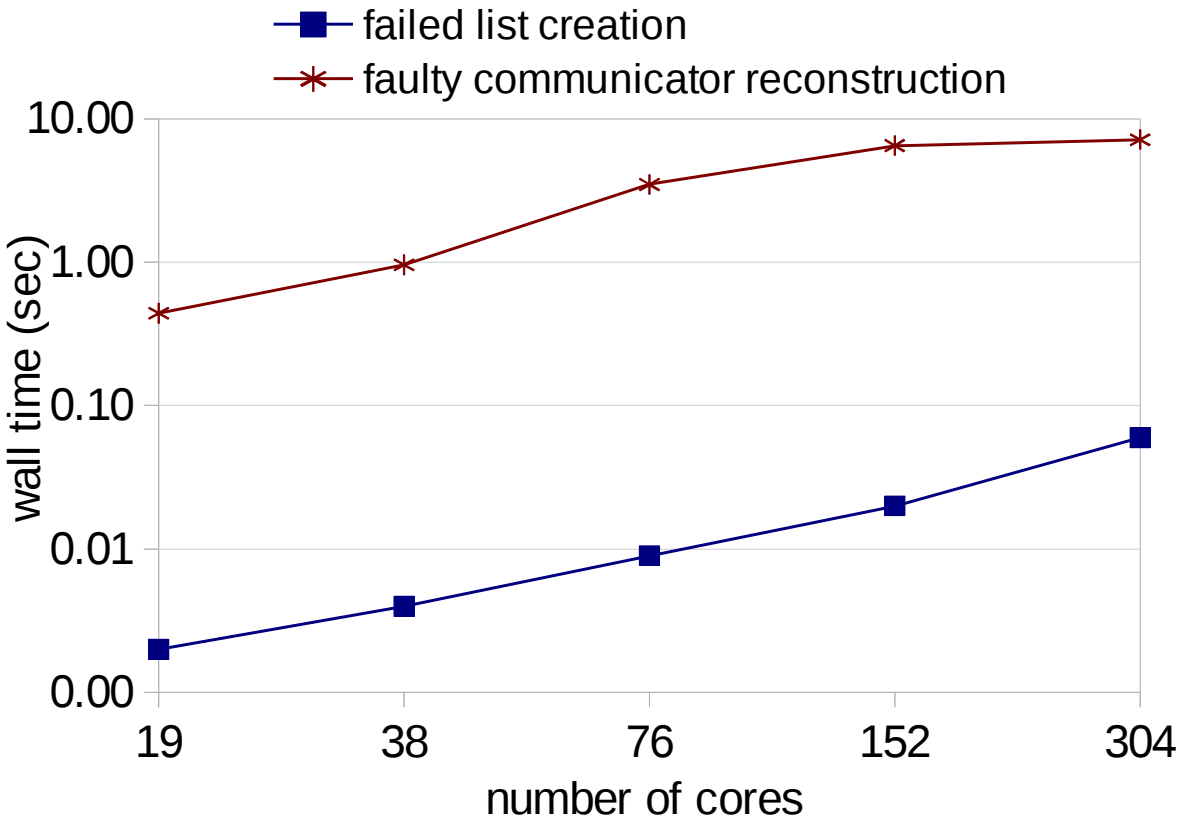
- can detect failed processes as follows:
  - attach an error handler ensuring failures get acknowledged on (original) communicator `comm`
  - call `MPI_Barrier(comm)`; if fails:
  - revoke it via `MPI_Comm_revoke(comm)` and create shrunken communicator via `OMPI_Comm_shrink(comm, &scomm)`
  - use `MPI_Group_difference(..., &fg)` to make a globally consistent list of failed processes

7 Fault Recovery Procedure: Process and Data Recovery

- process recovery:
  - use `MPI_Group_translate_ranks(fg, ..., comm, ...)` to re-rank remaining processes
  - spawn required number of failed processes via `MPI_Comm_spawn_multiple()`
    - these are called *child processes* and have own communicator
  - use `MPI_Intercomm_merge()` to merge child's comm. with parent's with `MPI_Comm_split()` to order the ranks
  - finally, `OMP_Comm_agree()` used to synchronize child and parent processes
- data recovery: must be done on whole of grid where a process has failed (data on non-failed process will be out-of-date)
  - checkpointing: use optimal interval $C = \sqrt{T/T_{I/O}}$ where
    $T_{I/O}$ is the single checkpoint disk write time
    T = MTBF (= 1/2 application run time)
  - alternate combination: recovery only involves identifying lost grids

8 Results: Process Failure Detection / Recovery

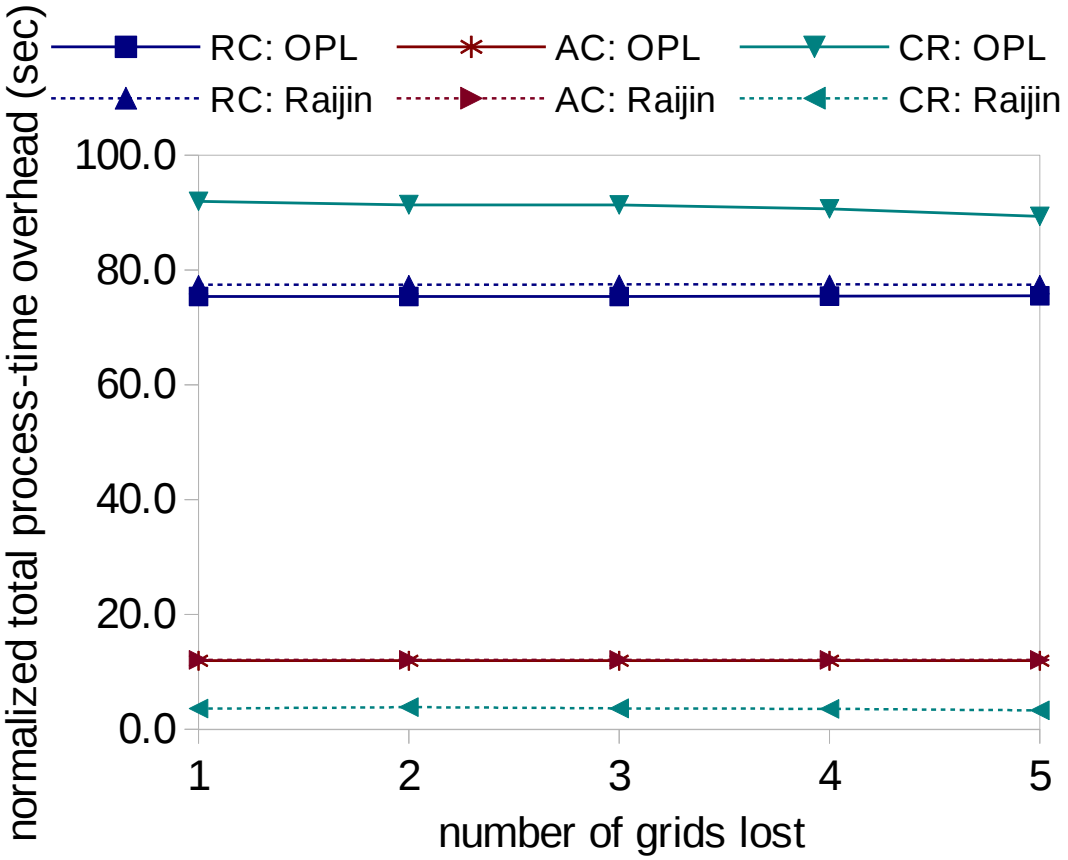
- results on OPL cluster (FLE): 2×6 Xeon X5670 nodes, QDR InfiniBand



  - significant reconstruction time in `OMPI_Comm_agree()`
- results overleaf from Raijin (NCI): 2×8 Sandy Bridge, FDR Infiniband

9 Results: Data Recovery

- results use a grids with a maximum resolution of 2^{13}
- take into account dilation of the process-time product of application



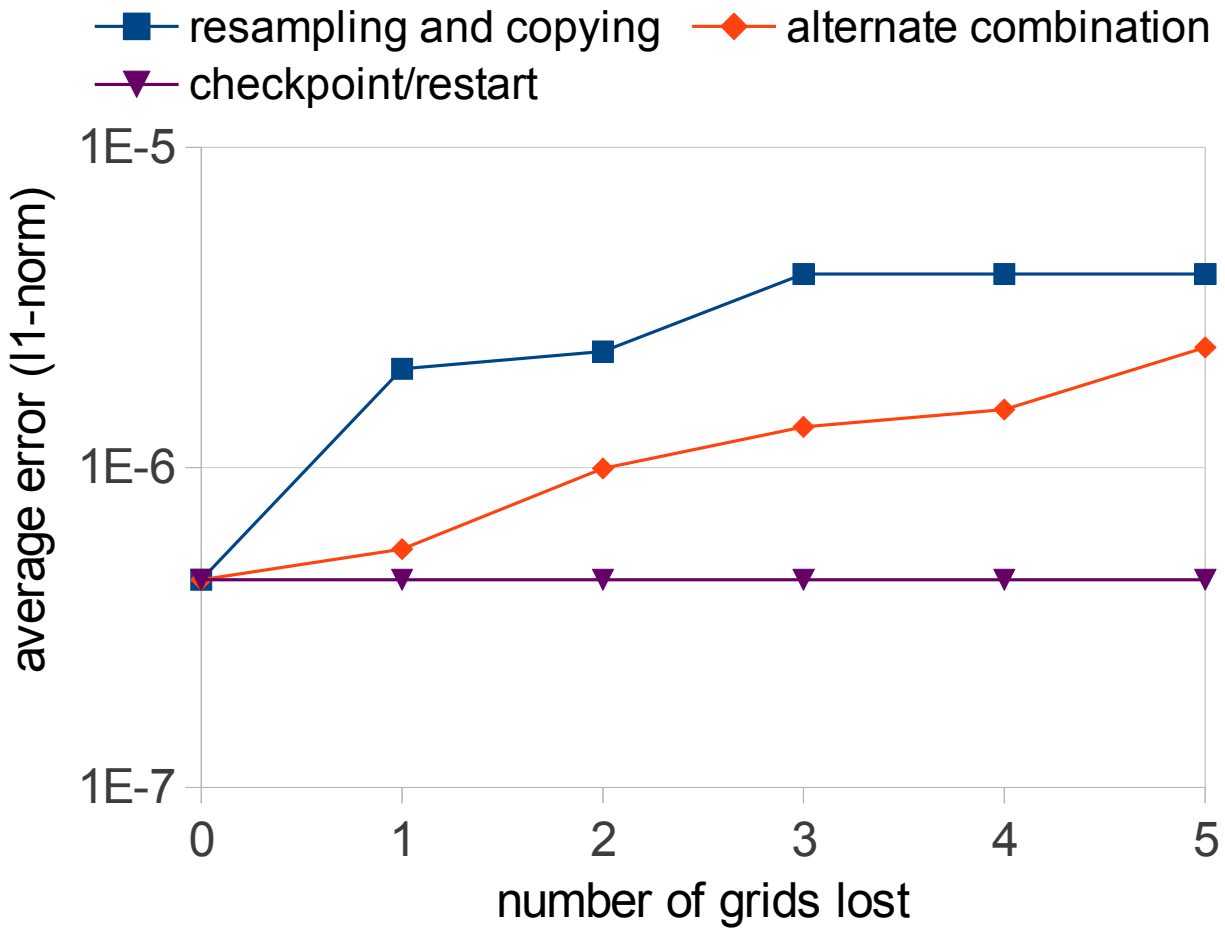
- RC = replication/resampling, AC = Alternate Combination, CR = Checkpoint/Restart

$$T_{CR} = T_{r,CR} + C \times T_{I/O}$$

- $T_{RC} = T_{r,RC} + T_{a,RC}\frac{P_{RC}-P_{CR}}{P_{CR}}$
  $T_{AC} = T_{r,AC} + T_{a,AC}\frac{P_{AC}-P_{CR}}{P_{CR}}$
- almost independent of number of failures (simulated if > 2)

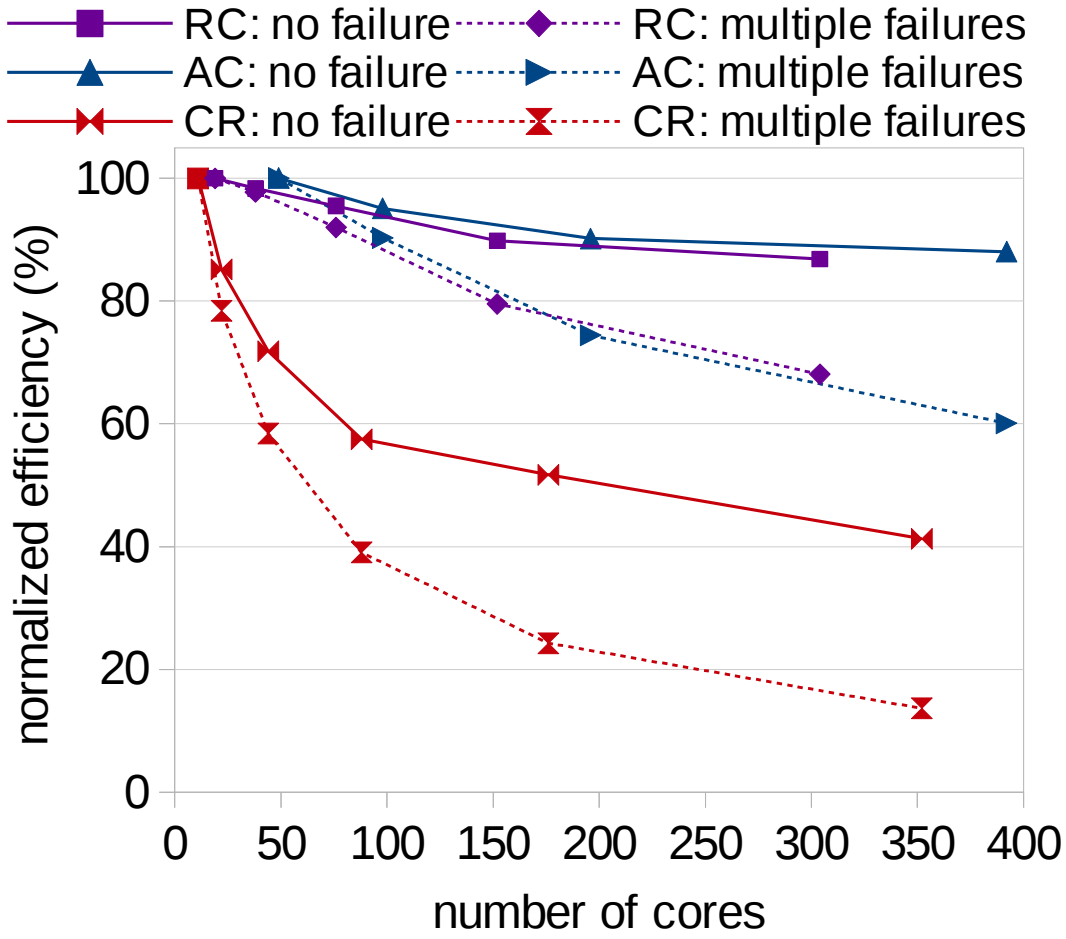
10 Results: Approximation Error in Combined Grid

- error computed via 1-norm of exact analytical solution



- as expected, errors grow with number of failed grids for RC and AC
  - but near-exact RC is always greater!

11 Results: Scalability



- results on OPL cluster, max. resolution of 2^{13}
- in terms of absolute time, CR is always more longer (however, uses fewer processes)
- RC and AC also show best scalability

12 Conclusions

- possible to implement process failure recovery in an application using ULFM MPI
  - actual code for recovery is surprisingly complex!
  - overhead of process recovery OK, communicator reconstruction too high!
- three recovery techniques were explored
  - necessary to take into account number of extra processes for fair comparison
  - on a cluster with typical disk write latency, checkpoint-restart has most overhead
    and alternate combination technique an order of magnitude lower
  - on cluster with ultra-low write latency, checkpoint has a clear ascendancy
- error of all three techniques acceptable ($< 10\times$ for up to 5 failures)
  - alternate combination technique is surprisingly better

13 Future Work

- investigate node failure, restart failed processes on spare nodes
- use higher efficiency combination techniques (via hierarchical basis functions)
- investigate using in existing, complex applications

Thank You!!

. . . Questions???