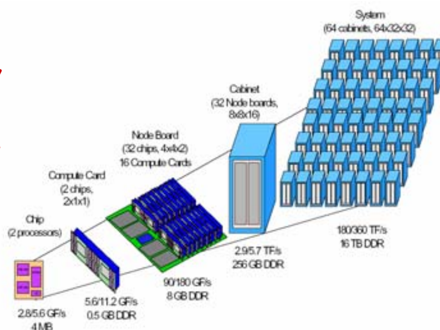
Fault Tolerance in Message Passing and in Action

Jack Dongarra,
Innovative Computing Laboratory
University of Tennessee
and
Computer Science and Mathematics Division
Oak Ridge National Laboratory

Fault Tolerance in the Computation

- The next generation systems are being designed with 100K processors (IBM Blue Gene L)
- 10^6 hours for component MTTF
  - sounds like a lot until you divide by 10^5!
  - Failures for such a system is likely to be just a few minutes / hours away.
- Application checkpoint /restart is today's typical fault tolerance method.
- Problem with MPI, no recovery from faults in the standard



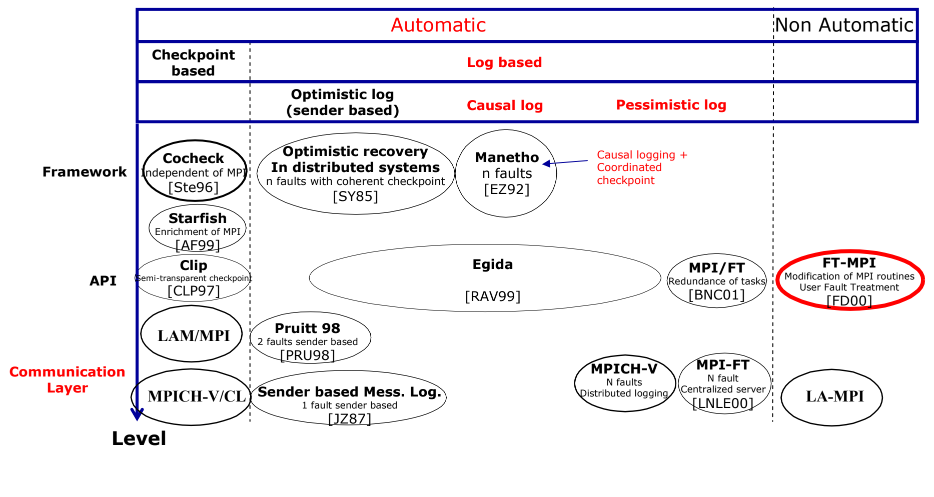
Motivation

- Trends in HPC:
  - High end systems with thousand of processors
  - Grid Computing
- Increased probability of a node failure
  - Most systems nowadays are robust
- Node and communication failure in distributed environments
- MPI widely accepted in scientific computing

Mismatch between hardware and (non fault-tolerant) programming paradigm of MPI.

Related work

A classification of fault tolerant message passing environments considering
A) level in the software stack where fault tolerance is managed and
B) fault tolerance techniques.



Fault Tolerance - Diskless Checkpointing - Built into Software

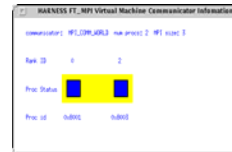
- Maintain a system checkpoint in memory
  - All processors may be rolled back if necessary
  - Use m extra processors to encode checkpoints so that if up to m processors fail, their checkpoints may be restored
  - No reliance on disk
- Other scheme Checksum and Reverse computation
  - Checkpoint less frequently
  - Option of reversing the computation of the non-failed processors to get back to previous checkpoint
- Idea to build into library routines
  - System or user can dial it up
  - Working prototype for MM, LU, LL^T, QR, sparse solvers using PVM
  - *Fault Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing*, J. Plank, Y. Kim, and J. Dongarra, JPDC, 1997.

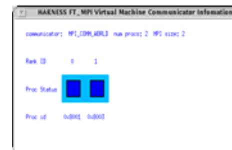
FT-MPI http://icl.cs.utk.edu/ft-mpi/

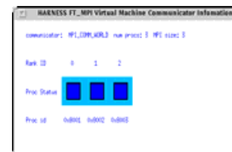
- Define the behavior of MPI in case an error occurs
- FT-MPI based on MPI 1.3 (plus some MPI 2 features) with a fault tolerant model similar to what was done in PVM.
- Gives the application the possibility to recover from a node-failure
- A regular, non fault-tolerant MPI program will run using FT-MPI
- What FT-MPI does not do:
  - Recover user data (e.g. automatic check-pointing)
  - Provide transparent fault-tolerance

FT-MPI Failure Modes

- ABORT: just do as other MPI implementations
- BLANK: leave hole
- SHRINK: re-order processes to make a contiguous communicator
  - Some ranks change
- REBUILD: re-spawn lost processes and add them to MPI_COMM_WORLD



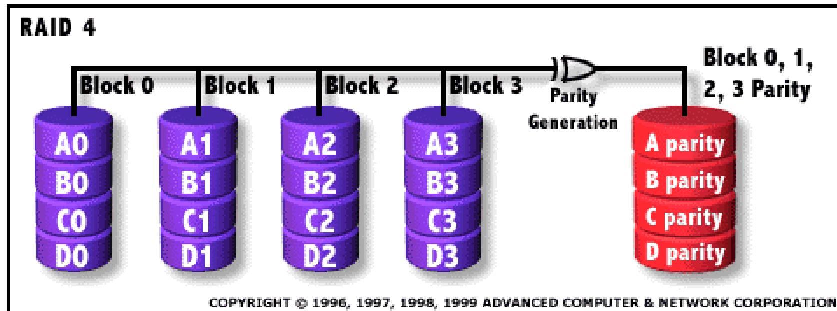




Algorithm Based Fault Tolerance Using Diskless Check Pointing

- Not transparent, has to be built into the algorithm
- N processors will be executing the computation.
  - Each processor maintains their own checkpoint locally (additional memory)
- M (M << N) extra processors maintain coding information so that if 1 or more processors die, they can be replaced
- The example looks at M = 1 (parity processor), can sustain addition failures with Reed-Solomon coding techniques.

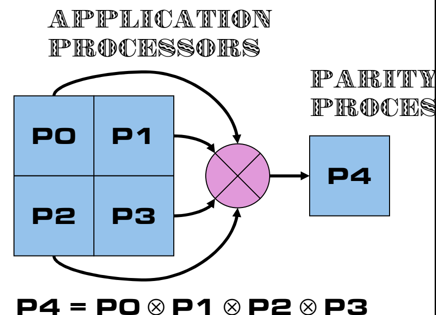
How Diskless Check Pointing Works



- If X = A XOR B then this is true:
  X XOR B = A
  A XOR X = B

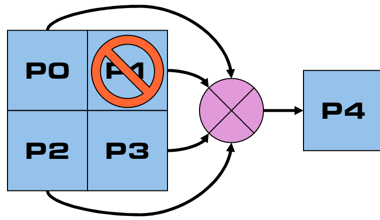
Diskless Checkpointing

- The N application processors (4 in this case) each maintain their own checkpoints locally.
- M extra processors maintain coding information so that if 1 or more processors die, they can be replaced.
- Will describe for m=1 (parity)
- If a single processor fails, then its state may be restored from the remaining live processors

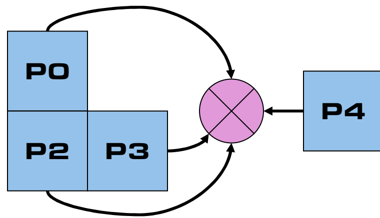


$P4 = P0 \otimes P1 \otimes P2 \otimes P3$

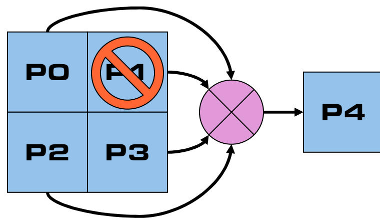
Diskless Checkpointing



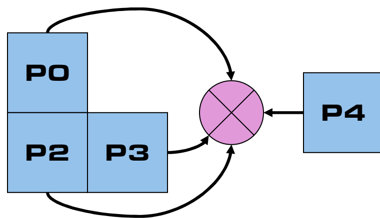
P1 = P0 ⊗ P2 ⊗ P3 ⊗ P4

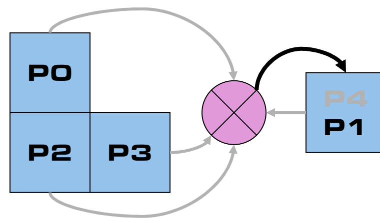


Diskless Checkpointing



P4 takes on the identity of P1 and the computation continues





Algorithm Based

- Built into the algorithm
  - Not transparent
  - Allows for heterogeneity
- Developing prototype examples for ScaLAPACK and iterative methods for Ax=b
- Not with XOR of the data, just accumulate sum of the data.
  - Clearly there can be problem with loss of precision
- Could use XOR as long as the recovery didn't involve roundoff errors

A Fault-Tolerant Parallel CG Solver

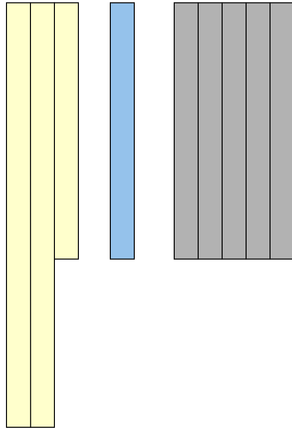
- Tightly coupled computation
- Do a "backup" (checkpoint) every *k* iterations
- Requires each process to keep copy of iteration changing data from checkpoint
- First example can survive the failure of a single process
- Dedicate an additional process for holding data, which can be used during the recovery operation

- For surviving *m* process failures ($m < np$) you need *m* additional processes (second example)

CG Data Storage

Think of the data like this

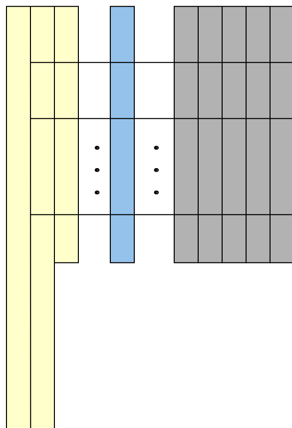
A b 5 vectors



Parallel version

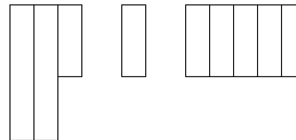
Think of the data like this

A b 5 vectors



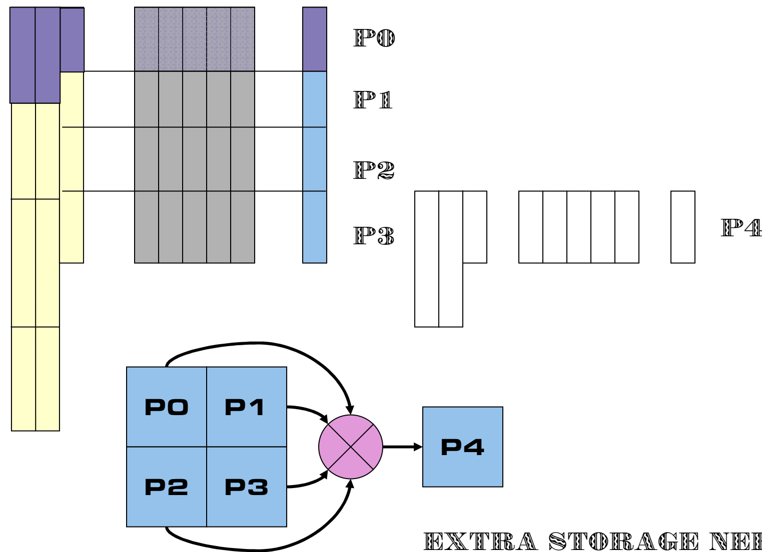
Think of the data like this on each processor

A b 5 vectors



No need to checkpoint each iteration, say every *k* iterations.

Diskless version



Preconditioned Conjugate Grad Performance

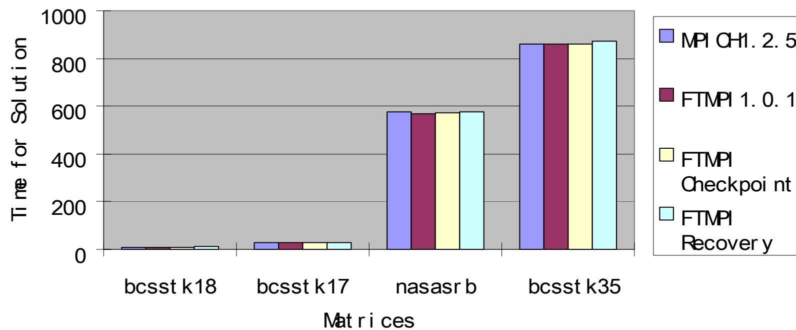


Table 1: PCG performance on 25 nodes of a dual Pentium 4 (2.4 GHz).
24 nodes are used for computation. 1 node is used for checkpoint
Checkpoint every 100 iterations (diagonal preconditioning)

Matrix (Size)	Mpich1.2.5 (sec)
bcsstk18.rsa (11948)	9.81
bcsstk17.rsa (10974)	27.5
nasasrb.rsa (54870)	577.
bcsstk35.rsa (30237)	860.

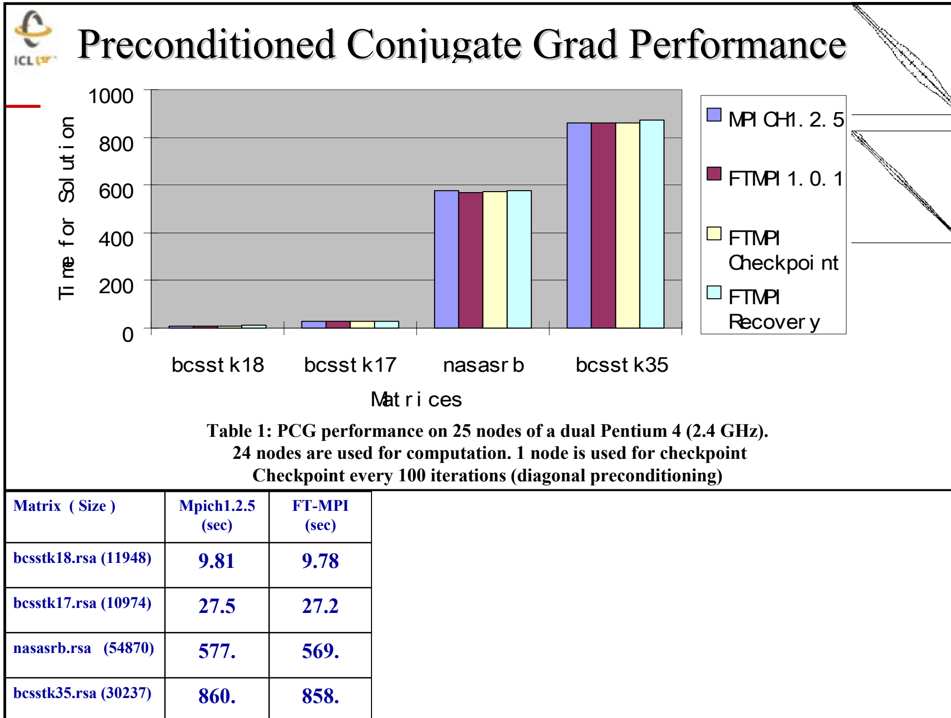


Table 1: PCG performance on 25 nodes of a dual Pentium 4 (2.4 GHz). 24 nodes are used for computation. 1 node is used for checkpoint Checkpoint every 100 iterations (diagonal preconditioning)

Matrix (Size)	Mpich1.2.5 (sec)	FT-MPI (sec)
bcsstk18.rsa (11948)	9.81	9.78
bcsstk17.rsa (10974)	27.5	27.2
nasasrb.rsa (54870)	577.	569.
bcsstk35.rsa (30237)	860.	858.

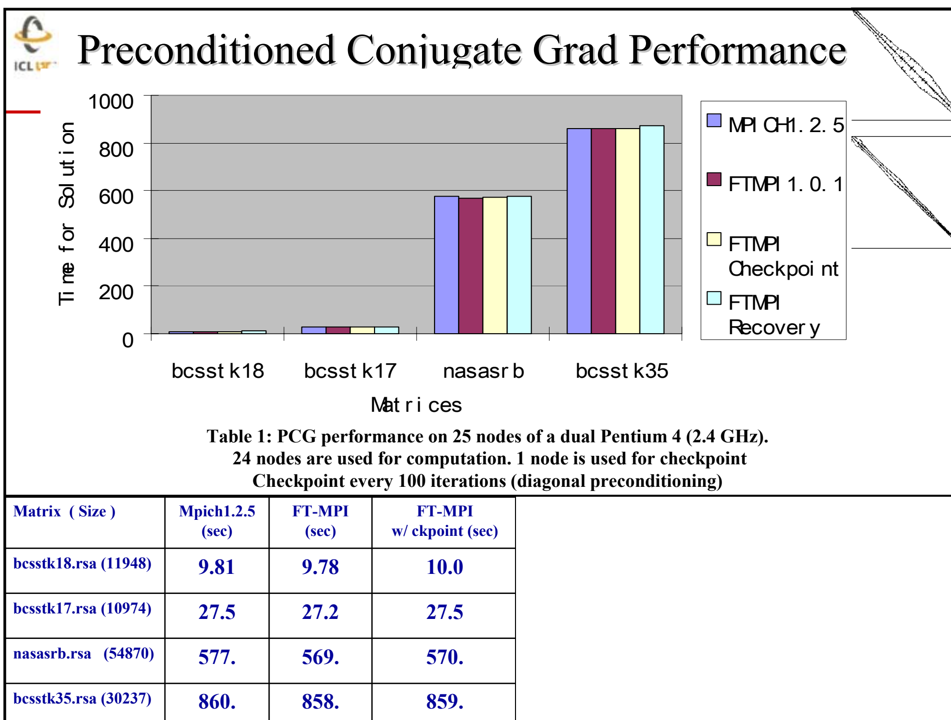


Table 1: PCG performance on 25 nodes of a dual Pentium 4 (2.4 GHz). 24 nodes are used for computation. 1 node is used for checkpoint Checkpoint every 100 iterations (diagonal preconditioning)

Matrix (Size)	Mpich1.2.5 (sec)	FT-MPI (sec)	FT-MPI w/ ckpoint (sec)
bcsstk18.rsa (11948)	9.81	9.78	10.0
bcsstk17.rsa (10974)	27.5	27.2	27.5
nasasrb.rsa (54870)	577.	569.	570.
bcsstk35.rsa (30237)	860.	858.	859.

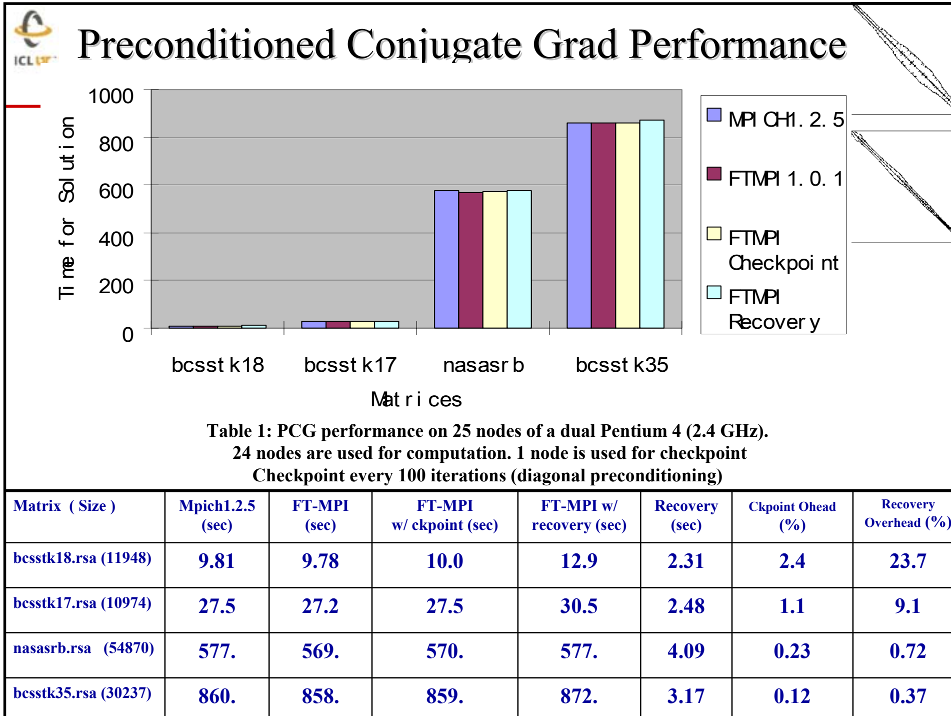


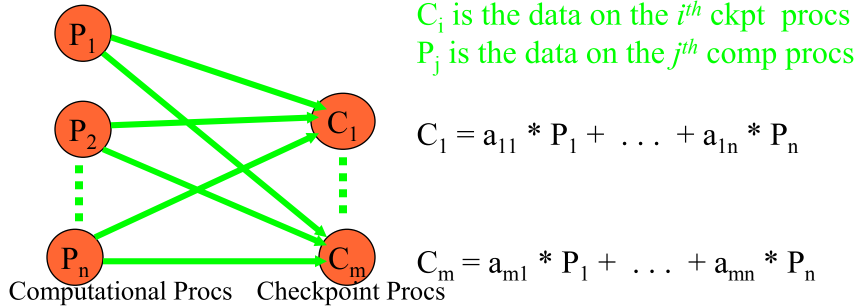
Table 1: PCG performance on 25 nodes of a dual Pentium 4 (2.4 GHz). 24 nodes are used for computation. 1 node is used for checkpoint Checkpoint every 100 iterations (diagonal preconditioning)

Matrix (Size)	Mpich1.2.5 (sec)	FT-MPI (sec)	FT-MPI w/ ckpoint (sec)	FT-MPI w/ recovery (sec)	Recovery (sec)	Ckpoint Ohead (%)	Recovery Overhead (%)
bcsstk18.rsa (11948)	9.81	9.78	10.0	12.9	2.31	2.4	23.7
bcsstk17.rsa (10974)	27.5	27.2	27.5	30.5	2.48	1.1	9.1
nasasrb.rsa (54870)	577.	569.	570.	577.	4.09	0.23	0.72
bcsstk35.rsa (30237)	860.	858.	859.	872.	3.17	0.12	0.37

Protecting for More Than One Failure: Reed-Solomon (Checkpoint Encoding Matrices)

- In order to be able to recover from any k (<= # of checkpoint processes) failures, need a checkpoint encoding matrix
- Say p processes each with P_i data
- Need a function A such that
- $C=A*P$ where $P=(P_1,P_2,\ldots P_p)^T$;
  - C: Checkpoint data $C = (C_1,C_2,\ldots C_k)^T$ (C_i and P_i same size)
  - A: Checkpoint-Encoding matrix A is k x p (k << p)
  - $C_i = a_{i1}P_1 + a_{i2}P_2 + \ldots + a_{ip} P_p$
    - Each checkpoint process get one of these
- The checkpoint matrix A has to satisfy:
  - Any square sub-matrix of A is non-singular
- How to find such an A? Vandermonde matrix, Cauchy matrix, . . ., random?
- When h failures occur, recover the data by taking the h x h submatrix of A, call it A', corresponding to the failed processes and solving A'P' = C'
  - A' is the h x h submatrix
  - C' is made up of the surviving h checkpoints

Checkpoint Encoding to Tolerate *m* Failures Reed Solomon Encoding



C_i is the data on the ith ckpt procs

P_j is the data on the jth comp procs

$$C_1 = a_{11} * P_1 + \ldots + a_{1n} * P_n$$

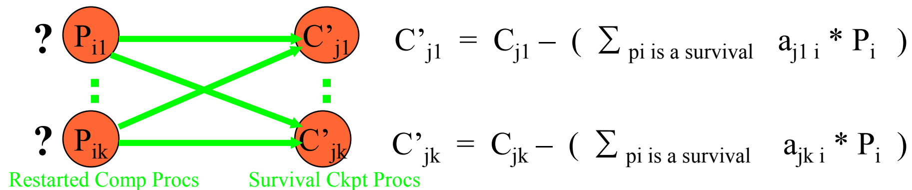
$$C_m = a_{m1} * P_1 + \ldots + a_{mn} * P_n$$

The checkpoint are done in *m* steps: In each step *j*, every computational processor first prepares its own data (computational processor *i* calculates $a_{ji}*P_i$), and then sends its data out. The checkpoint processor *j* receives the sum of these data. This is implemented through MPI_Reduce with a MPI_SUM operation.

Suppose the data size in each computational processor is ***x*** floating point numbers, then

- Computation Overhead: ***mx*** multiplications on each processor. Please note MPI_Reduce involve additions. The number of additions on each processor is dependent on the implementation of the reduce
- Communication Overhead: ***m*** MPI_Reduce with ***x*** numbers in each message.
- Memory Overhead: depend on doing local ckpt or reverse comp. The same as tolerate one failure.

Recovery Decoding



$$C'_{j1} = C_{j1} - \left(\sum_{\text{pi is a survival}} a_{j1\,i} * P_i \right)$$

$$C'_{jk} = C_{jk} - \left(\sum_{\text{pi is a survival}} a_{jk\,i} * P_i \right)$$

How to find out the lost data P' on restarted processes? Solve a linear system: $\mathbf{A'_{kxk} * P'_{kx1} = C'_{kx1}}$

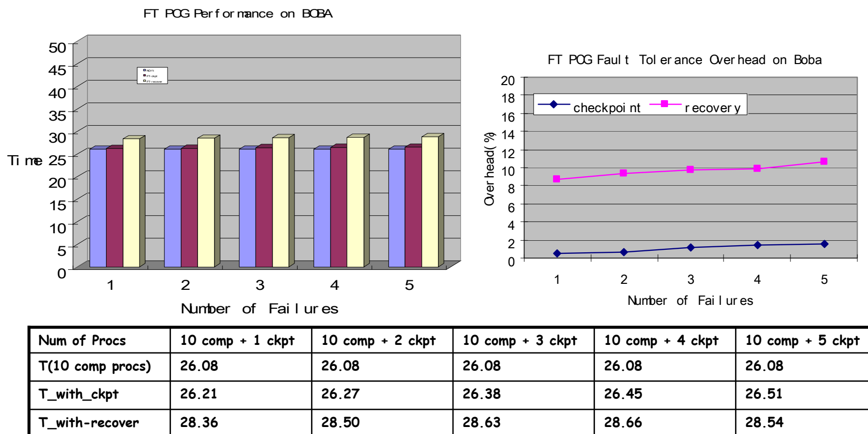
Suppose there are *k* comp procs and *h* ckpt procs dies ($k + h \le m$). Assume the data size in each computational processor is *x* floating point numbers :

(1). Find any *k* survival ckpt procs, modify their encodings from C_j to C'_j (see above formula)
- Computation Overhead: ***kx*** floating point calculations on each processor.
- Communication Overhead: ***k*** MPI_Reduce with ***x*** floating point numbers in each message

(2). The *k* restarted comp procs and the *k* survival ckpt procs calculate the lost data $(P_{i1}, P_{i2}, \ldots, P_{ik})'$ on the *k* restarted comp procs by solving $A'_{kxk} * P'_{kx1} = C'_{kx1}$
- Computation Overhead: $O(k^3+\boldsymbol{kx})$ floating point calculations on each processor.
- Communication Overhead: ***k*** MPI_Reduce with ***x*** floating point numbers in each message

(3). Re-encoding the checkpoint data to the *h* restarted ckpt procs.
- Computation Overhead: ***hx*** floating point calculations on each processor.
- Communication Overhead: ***h*** MPI_Reduce with ***x*** floating point numbers in each message

FT PCG Performance



Num of Procs	10 comp + 1 ckpt	10 comp + 2 ckpt	10 comp + 3 ckpt	10 comp + 4 ckpt	10 comp + 5 ckpt
T(10 comp procs)	26.08	26.08	26.08	26.08	26.08
T_with_ckpt	26.21	26.27	26.38	26.45	26.51
T_with-recover	28.36	28.50	28.63	28.66	28.54

The test matrix is besstk17.rsa(10974x10974). The pcg uses diagonal as preconditioner, and does checkpoint every 100 iterations (about every 1 seconds).

A fault tolerant pcg which can tolerate multiple failures has been developed. In the fault tolerant scheme, each processor maintains a copy of the local checkpoint data in memory. At the same time, multiple encodings of these local checkpoint data are maintained on *m* dedicated checkpoint processors. A Cauchy matrix is used as the checkpoint matrix. If failures happen, the lost data can be recovered through solving a linear system.

Futures

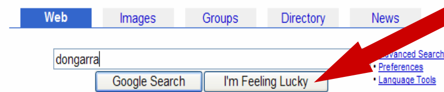
Investigate ideas for 10K to 100K processors:

- Determine the checkpointing interval from MTMF for the machine
- Local checkpoint and restart algorithm.
  - Coordination of local checkpoints.
  - Processors hold backups of neighbors.
- For some algorithms, unwind the computation to get back to the checkpoint, eg LU, QR, LL^T (clearly more expensive)
- Development of super-scalable fault-tolerant MPI implementation with localized recovery.

Collaborators / Support

- FT-MPI
  - Graham Fagg, UTK
  - Edgar Gabriel, UTK
  - Thara Angskun, UTK
  - George Bosilca, UTK
  - Jelena Pjesivac-Grbovic, UTK
- FT Algorithms
  - Jeffery Chen, UTK

- For more information:



Advertise with Us - Business Solutions - Services & Tools - Jobs, Press, & Help

Make Google Your Homepage!

©2003 Google - Searching 3,083,324,652 web pages