COMPUTER SCIENCES

Self Adapting Numerical Software (SANS) – Effort and Fault Tolerance in Linear Algebra Algorithms

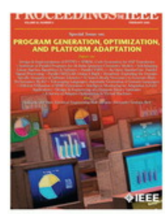
Jack Dongarra
University of Tennessee
and
Oak Ridge National Laboratory

9/16/2006 1

Self Adapting Numerical Software

- The process of arriving at an efficient solution involves many decisions by an expert.
  - Algorithm decisions
  - Data decisions
  - Management of the computing environment
  - Processor specific tuning

Proceedings of the IEEE, V: 93 #: 2 Feb. 2005 Issue on Program Generation, Optimization, and Platform Adaptation



Complex set of interaction between
Users' applications
Algorithm
Programming language
Compiler
Machine instruction
Hardware
Many layers of translation from the application to the hardware. Changing with each generation of hardware.

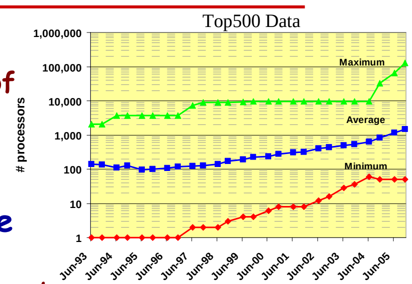
20 2

Self Adapting Numerical Software

- Optimizing software to exploit the features of a given system has historically been an exercise in hand customization.
  - Time consuming and tedious
  - Hard to predict performance from source code
  - Must be redone for every architecture and compiler
    - Software technology often lags hardware/architecture
    - Best algorithm may depend on input, so some tuning may be needed at run-time.

- With good reason scientists expect their computing tools to serve them and not the other way around.
- There is a need for quick/dynamic deployment of optimized routines.
  - ATLAS, PhiPAC, BeBoP, Spiral, FFTW, GCO, …

Fault Tolerance: Failure is an Option



- Trends in HPC:
  - High end systems with thousand of processors
  - Move to multicore nodes

- Increased probability of a node failure
  - However, most systems today are robust

- MPI widely accepted in scientific computing
  - Process faults not tolerated in MPI standard

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

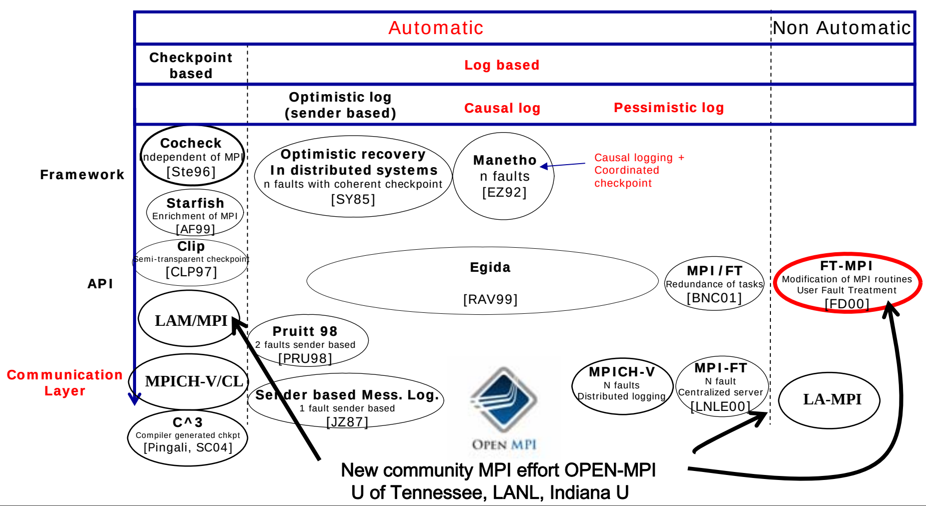
Reliability of Large Systems

(Source: Daniel Reed, UNC)

Machine	# CPU	Reliability
ASCI Q	8,192	MTBI 6.5 hr. 114 unplanned outages/month. HW outage sources: storage, CPU, memory *
ASCI White	8,192	MTBF 5 hr ('01) and 40 hr ('03) HW outage sources: storage, CPU, 3rd party hardware **
NERSC Seaborg	6,656	MTBI 14 days. MTTR 3.3 hr Availability 98.74%. SW is main outage source. ***
PSC Lemieux	3,016	MTBI 9.7 hr Availability 98.33% ****
Google	~150,000	20 reboots/day. 2-3% machines replaced/year. HW outage sources: storage, memory

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.



Open-MPI: Fault Tolerance Models Overview

- Frameworks planned to be supported, depending on the user involvement level
  - Automatic (no user involvement)
    - Check-point/restart (coordinated)
    - Log Based (uncoordinated)
      - Optimistic, Pessimistic, Casual
  - User driven
    - The environment recover depending on the user specifications, then the user recover the algorithmic requirements

20 7

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

- Define the behavior of MPI in case a process failure 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.
  - Complete reimplementation, not based on other implementations.
- A regular, non fault-tolerant MPI program will run using FT-MPI.
- Gives the application the possibility to recover from a process-failure.
- What FT-MPI does not do:
  - Recover user data (e.g. automatic check-pointing)
  - Provide transparent fault-tolerance

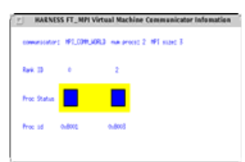
20 Part of the DOE Harness (ORNL, UTK, Emory) Project 8

Assumptions and Basic Ideas

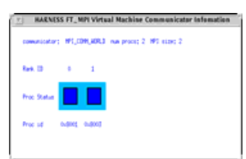
- Assume
  - Only a small number (or percentage) of processes will fail
  - The failed processes stop working (fail-stop model)
  - It is possible to detect such failures with the help of the execution environment (such as PVM, FT-MPI, Open MPI, …)
- The basic idea of our work
  - Keep all surviving processes (DO NOT ABORT)
  - Maintain redundancy locally by coding approaches
  - Eliminate periodical I/O access to stable storage which is the bottle neck for performance and scalability
  - Restart only the failed processes (same number to resume i.e. no data redistribution)
  - Reconstruct the global consistent states from the local redundancy
- Two approaches
  - Scalable in-memory (or local disk) checkpointing
  - Scalable algorithm-based checkpoint-free fault tolerance

FT-MPI Failure Modes

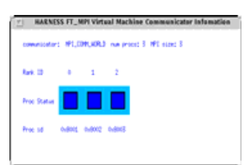
- ABORT: just do as other implementations



- 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



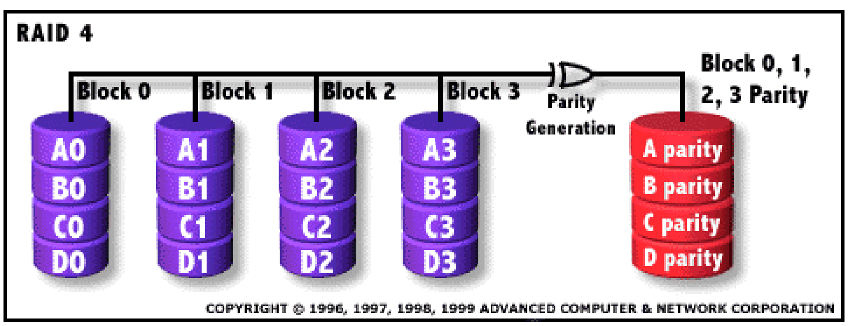
Three Ideas for Fault Tolerant Linear Algebra Algorithms

- **Lossless diskless check-pointing for iterative methods**
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- **Lossy approach for iterative methods**
  - No checkpoint maintained
  - On failure, approximate missing data and carry on
  - Lost data but use approximation to recover
- **Check-pointless methods for dense algorithms**
  - Checksum maintained as part of computation
  - No roll back needed; No lost data

20

Disk-less Checkpointing

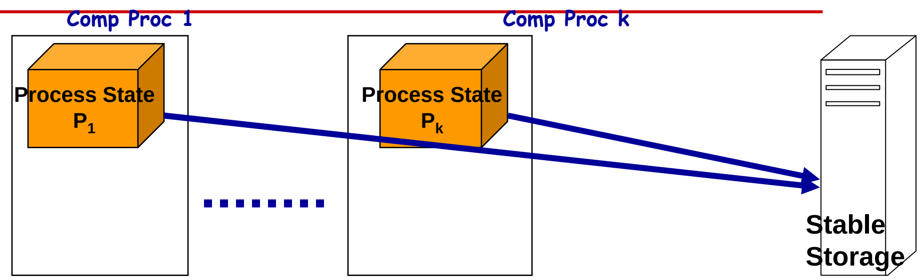
- **Similar to RAID for disks.**



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

20 12

Checkpoint/Restart

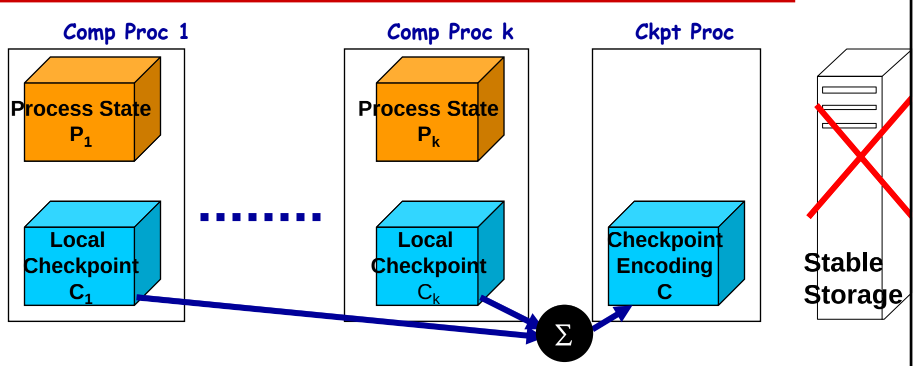


- Checkpoint/restart is today's typical fault tolerance approach in HPC
  - Periodically write process states into *stable-storage*
  - If one process fails, *abort all processes*
  - Good to tolerate the failure of the whole system
  - But the overhead is high : T = # of procs * size_ckpt / bandwidth

20 13

First Approach: Diskless Checkpointing

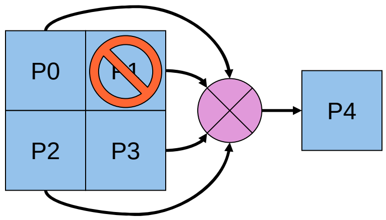
(K. Li & J. Plank, et. el.)



- Each computational processor saves a copy of its state locally in memory
- Dedicate an additional processor to save the encoding of these states
- The checkpoint overhead is (binary tree encoding):

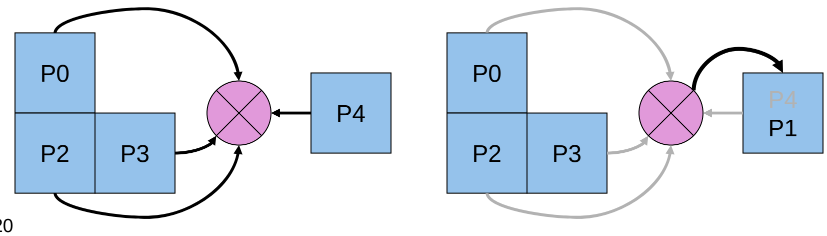
20 T = log (# of procs) *size_ckpt / bandwidth + log (# of procs)*latenc 14

Diskless Checkpointing



- **When failure occurs:**
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

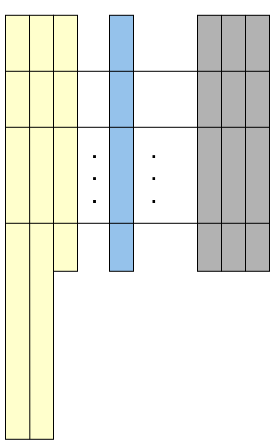
P4 takes on the identity of P1
and the computation continues.



CG Parallel Version

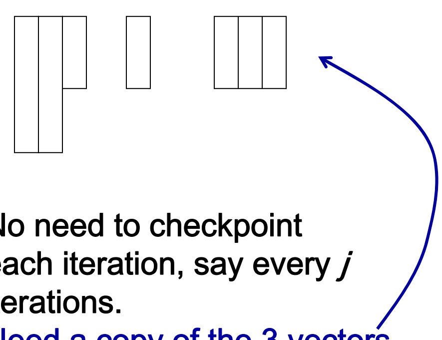
Think of the data like this

A b 3 vectors



Think of the data like this on each processor

A b 3 vectors



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

Need a copy of the 3 vectors from checkpt in each processor to maintain state.

FT PCG Algorithm Analysis

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1} / p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Global Operations

Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

20 Global operation in Checkpoint: encoding the local checkpoint. 17

FT PCG Algorithm Analysis

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1} / p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Checkpoint *x, r,* and *p* every k iterations

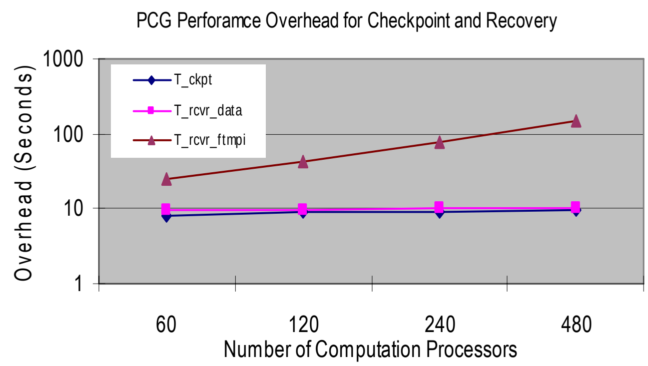
Global Operations

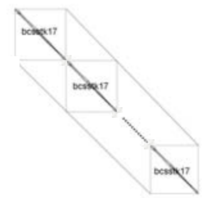
Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

20 Global operation in Checkpoint: encoding the local checkpoint. 18

Global operation in checkpoint can be localized by sub-group.

PCG: Performance





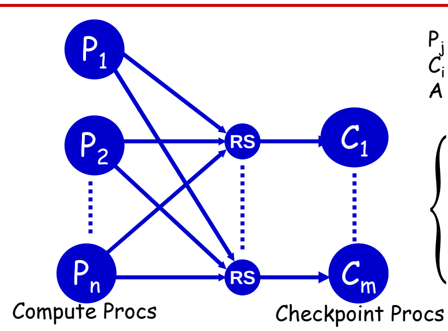
IBM RS/6000 SP w/176 Winterhawk II thin nodes (each with four 375 MHz Power3-II processors)

Run PCG for 5000 iterations and take checkpoint every 1000 iterations
Cause a failure at the 3000-th iteration.
Matrix size scales with the processors used, i.e. 60 procs: n=658,440; 480 procs: n=5.2M

Time (Sec)	Time w/o checkpoint	Checkpoint time	Data Recovery time	System Recovery time	Total time to recover from fault
60 procs	1399.1	8.0	9.8	24.8	1441.7
120 procs	1429.3	9.2	9.9	42.1	1490.5
240 procs	1461.1	9.2	10.0	77.2	1557.5
480 procs	1531.1	9.7	10.1	146.1	1697.0

20 19

Coding to Survive Multiple Failures: Basic Scheme (Reed-Solomon Encoding)



P_j is the checkpoint data on the j^{th} comp procs
C_i is the encoded data on the i^{th} ckpt procs
$A = (a_{ij})_{m \times n}$ is a encoding matrix

$$\begin{cases} C_1 = a_{11} * P_1 + \; ... \; + a_{1n} * P_n \\ \vdots \\ C_m = a_{m1} * P_1 + \; ... \; + a_{mn} * P_n \end{cases}$$

Key idea: establish *m* equalities by *m* encodings

If there are *k* (<= *m*) processes failed, then the *m* equalities become

m equations with *k* unknowns

By appropriately choosing A, the lost data can be recovered by solving the *m* equations.

The checkpoint overhead (assume pipelined encoding):

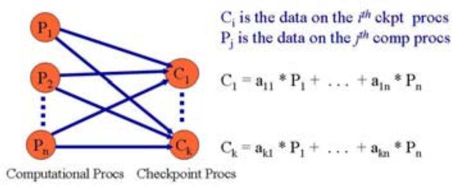
T = m * { (1 + O(1/size_ckpt^0.5)) * size_ckpt / bandwidth + #of procs * latency } 20 20

Reed-Solomon Approach

$A*P = C$, *where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$



Reed-Solomon Approach

$A*P = C$, *where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ \not{P_2} \\ \not{P_3} \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Say 2 processors fail, P_2 and P_3.

Reed-Solomon Approach

$A*P = C$, *where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Say 2 processors fail, P_2 and P_3.

Take a subset of A's (columns 2 and 3) and solve for P_2 and P_3.

$$\begin{pmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

Floating-Point Coding to Tolerate Multiple Failures

$$\begin{cases} C_1 = a_{11} * P_1 + \cdots + a_{1j} * P_j + \cdots + a_{1(j+m)} * P_{j+m} + \cdots + a_{1n} * P_n \\ \vdots \\ C_m = a_{m1} * P_1 + \cdots + a_{mj} * P_j + \cdots + a_{m(j+m)} * P_{j+m} + \cdots + a_{mn} * P_n \end{cases}$$

- **In order to be able to recover from any k (k <= m) failures, the checkpoint encoding matrix A has to satisfy**

  Any square sub-matrix of A is non-singular

- **How to find such an A ?**
  - **Vandermonde matrix, Cauchy matrix, ...**

- To maintain the checksum relationship
  - Floating-point arithmetic **has to be used in calculating encodings**

- **Due to** round-off errors **in floating-point computations**

  Require any square sub-matrix of A is well-conditioned

Condition Numbers of Gaussian Random Matrices: Theory

$$\frac{1}{\sqrt{2\pi}}\left(\frac{0.245\,\frac{n}{|n-m|+1}}{x}\right)^{|n-m|+1} < \Pr\left(\kappa_2(G_{m\times n}) > x\right) < \frac{1}{\sqrt{2\pi}}\left(\frac{6.414\,\frac{n}{|n-m|+1}}{x}\right)^{|n-m|+1}$$

$$E\left(\log_{10}\kappa_2(G_{m\times n})\right) < \log_{10}\frac{n}{|n-m|+1} + 0.981 .$$

In our fault tolerant applications:

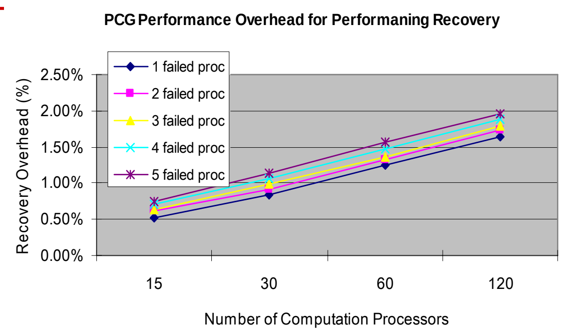
- n: is the number of redundant processor used.
- m: is the number of processor failures actually occurred.
- $G_{m\times n}$: is the coefficient matrix of the recovery equation.

Example: Assume you are running an application on a 100K-processor system, and tolerating 20 concurrent failures. If there are 10 concurrent failures actually occurred, then m=10, n=20

$E(\log_{10} k) < 1.25$: On average, you will loss about 1 digit in recovery
$\Pr(k > 10^2) < 3.1*10^{-11}$: The probability to loss 2 digits is less than 10^{-10}

20 *Condition Numbers of Gaussian Random Matrices,* Z. Chen & J. Dongarra, SIAM Matrix Analysis and Applications, Volume 27, Number 3, pp 603-620, 2005. 25

PCG: Performance Overhead of Recovery



64 dual processor 2.4 GHz Opteron
Nodes are connected with GigE

Run PCG for 20000 iterations and take checkpoint every 2000 iterations
Cause a failure by exiting some processes at the 10000-th iteration

T (ckpt T)	0 failures	1 failures	2 failures	3 failures	4 failures	5 failures
15 comp	517.8	521.7 (2.8)	522.1 (3.2)	522.8 (3.3)	522.9 (3.7)	523.1 (3.9)
30 comp	532.2	537.5 (4.5)	537.7 (4.9)	538.1 (5.3)	538.5 (5.7)	538.6 (6.1)
60 comp	546.5	554.2 (6.9)	554.8 (7.4)	555.2 (7.6)	555.7 (8.2)	556.1 (8.7)
120 comp	622.9	637.1 (10.5)	637.2 (11.1)	637.7 (11.5)	638.0 (12.0)	638.5 (12.5)

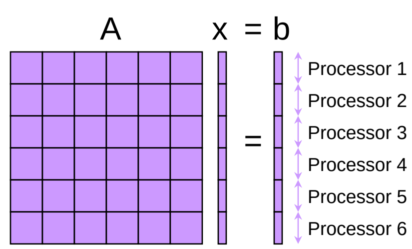
20 26

Second Approach

- Lossy approach for iterative methods
  - Here there is only a checkpoint of the primary data
    - Continuous checkpointing is not done during the iteration.
  - When the failure occurs we will approximate the missing data and continue
    - No guarantee here; may or may not work

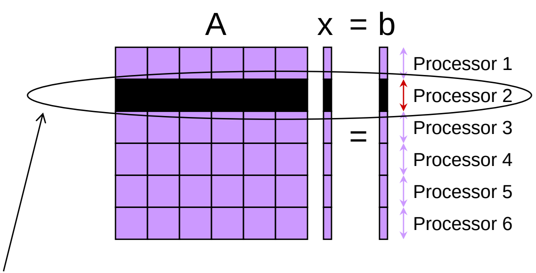
Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows



Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows

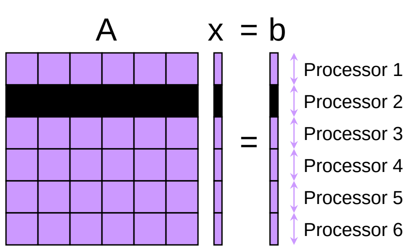


Processor 2 (e.g.) fails, all its data is lost.

How to recover the lost part of x in this case?

Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows

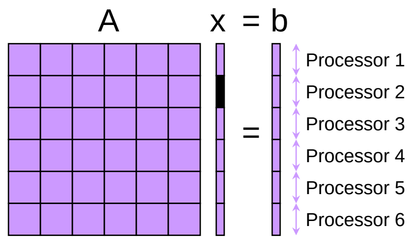


3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows



3 steps

Step 1: revover a processor and a running parallel environement (the job of the FT-MPI library)

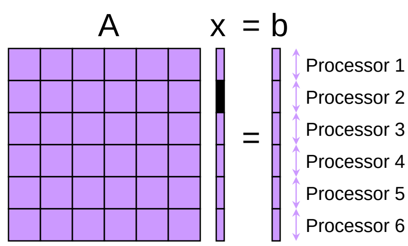
Step 2: recover A_{21} A_{22}, ..., A_{n2} and b_2 (the original data) on the failed processor

20

31

Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows



3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Step 2: recover A_{21} A_{22}, ..., A_{n2} and b_2 (the original data) on the failed processor

Step 3: Notice that

$$A_{21} x_1 + A_{22} x_2 + \dots + A_{2n} x_n = b_2 \Rightarrow$$

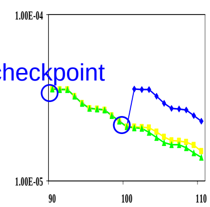
$$x_2 = A_{22}^{-1} (b_2 - \Sigma_{i \neq 2} A_{2i} x_i)$$

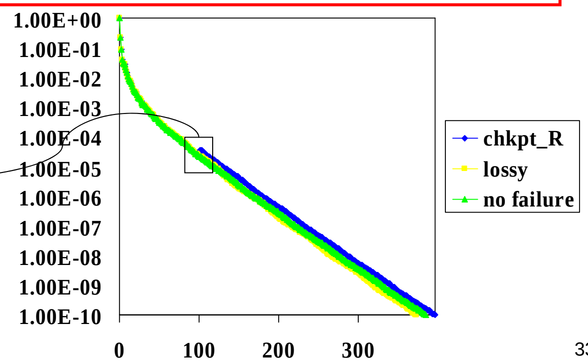
20

32

Using GMRES(30) Non Symetric Matrix

stomach; n=213,360; nnz=3,021,648; tol=10^{-10}; #procs=16; n_f=13,335; nnz=185,541									
recovery	$iter_f$	#iter	T_{Wall}	T_{Chkpt}	T_{Roll}	T_{Recov}	T_I	$T_{II,a,b}$	T_{III}
lossy	no	385	38.89						
$chkpt_D$	no	385	41.04	1.92					
lossy	100	372	42.38		1.56	5.38	1.03	0.33	3.91
$chkpt_R$	100	395	45.49	1.92	2.40	1.68	1.02	0.32	0.20





Time are given in seconds
Intel Xeon at 2.40 GHz with
Myrinet interconnect

20

33

Third Approach: Matrix-Vector Multiplication with Checksum Matrix

$$M^r = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1q} \\ \vdots & \vdots & \cdots & \vdots \\ M_{p1} & M_{p2} & \cdots & M_{pq} \\ \sum_{i=1}^{p} M_{i1} & \sum_{i=1}^{p} M_{i2} & \cdots & \sum_{i=1}^{p} M_{iq} \end{pmatrix} \qquad v = \begin{pmatrix} v_1 \\ \cdots \\ v_q \\ \sum_q v_i \end{pmatrix}$$

Let $$b = M^r v = \begin{pmatrix} \sum_{j=1}^{q} M_{1j} v_j \\ \vdots \\ \sum_{j=1}^{q} M_{pj} v_j \\ \sum_{i=1}^{p} \sum_{j=1}^{q} M_{ij} v_j \end{pmatrix}$$

Then $b_1 + \cdots + b_p = b_{p+1}$

Matrix and vectors stored by rows on processors.

Conclusion: Any singular failure in the result b can be corrected

20 K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," IEEE Transactions on Computers, vol. C-33, June 1984, pp. 518--528.

34

Fault Tolerant Dense Matrix Computations

- Assume the original matrix M is distributed into a *p* by *q* processor grid with a 2D block cyclic distribution. Then from processor point of view, the distributed matrix is

$$M = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \end{pmatrix}$$

, where M_{ij} is the local matrix on processor (*i* , *j*).

- Define the *full distributed checksum* matrix of M as:

$$M^f = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^{q} M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^{q} M_{pj} \\ \sum_{i=1}^{p} M_{i1} & \cdots & \sum_{k=1}^{p} M_{iq} & \sum_{i=1}^{p}\sum_{j=1}^{q} M_{ij} \end{pmatrix}$$

- For p x q processors need extra p + q + 1 processors to maintain the checksum.

20 35

An Example: ScaLAPACK/PBLAS Matrix Multiplication

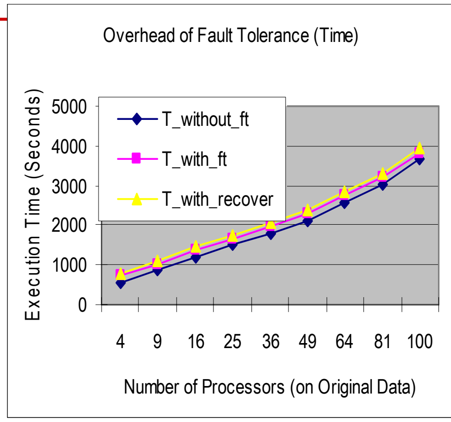
$$\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & \cdots & \vdots \\ A_{p1} & \cdots & A_{pq} \\ \sum_{i=1}^{p} A_{i1} & \cdots & \sum_{i=1}^{p} A_{iq} \end{pmatrix} * \begin{pmatrix} B_{11} & \cdots & B_{1p} & \sum_{j=1}^{p} B_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ B_{q1} & \cdots & B_{qp} & \sum_{j=1}^{p} B_{qj} \end{pmatrix}$$

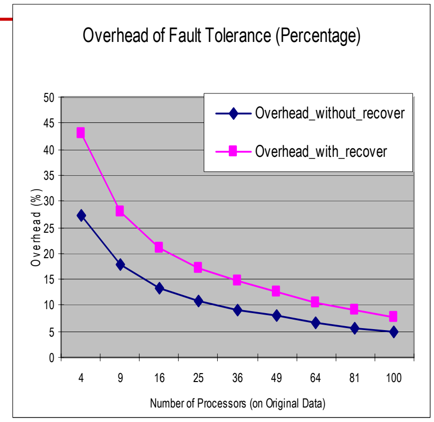
$$= \begin{pmatrix} C_{11} & \cdots & C_{1p} & \sum_{j=1}^{p} C_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ C_{p1} & \cdots & C_{pp} & \sum_{j=1}^{p} C_{pj} \\ \sum_{i=1}^{p} C_{i1} & \cdots & \sum_{k=1}^{p} C_{ip} & \sum_{i=1}^{p}\sum_{j=1}^{p} C_{ij} \end{pmatrix}$$

- Single failure during computation can be recovered from the checksum relationship
- By using a floating-point version Reed-Solomon code, multiple failures can be tolerated

20 36

PDGEMM: the Overhead for Fault Tolerance



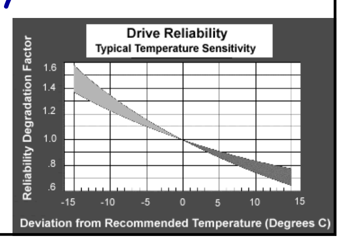


- **Size of local matrices on each process: 6,400 by 6,400**
- **Platform: 128 processors, Intel EM64T, 64bit w/Myrinet**
- **Note that the overhead (%) for fault tolerance is**
  - **O (1 / (p*n)) ⟶ 0, as p ⟶ ∞**

20 37

Predictive Adaptive Fault Tolerence

- **Large-scale fault tolerance**
  - **adaptation: resilience and recovery**
  - **predictive techniques for probability of failure**
    - **resource classes and capabilities**
    - ***coupled to application usage modes***
  - **resilience implementation mechanisms**
    - **adaptive checkpoint frequency**
    - **in memory checkpoints**
- **By monitoring, one can identify**
  - **performance problems**
  - **failure probability**
- **When potential of failure**
  - **Migrate process to another processor**



20

Next Steps

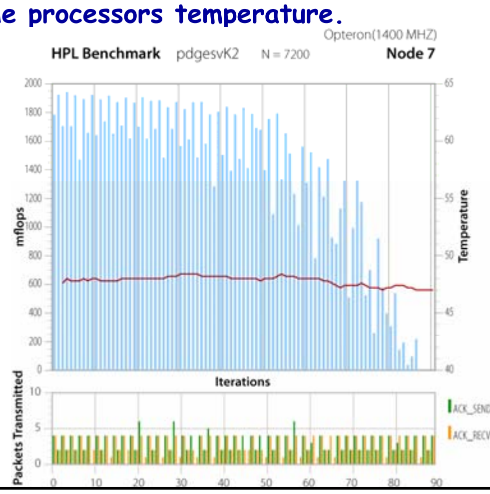
- Software to determine the checkpointing interval and number of checkpoint processors from the machine characteristics.
  - Perhaps use historical information.
  - Monitoring
  - Migration of task if potential problem
- Local checkpoint and restart algorithm.
  - Coordination of local checkpoints.
  - Processors hold backups of neighbors.
- Have the checkpoint processes participate in the computation and do data rearrangement when a failure occurs.
  - Use p processors for the computation and have k of them hold checkpoint.
- Generalize the ideas to provide a library of routines to do the diskless check pointing.

PAPI 4.0

- PAPI is software layer that aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors.
- PAPI has historically targeted on on-processor performance counters
  - Ops, cycles, memory traffic
  - Extending to look at other features of system
    - Communication and power issues
- Substrates available for
  - ACPI (Advanced Configuration and Power Interface)
  - Myrinet MX
- Substrates under development for
  - Infiniband
  - GigE
- PAPI 4.0 Beta release expected Q2, 2006

Temperature Sensor

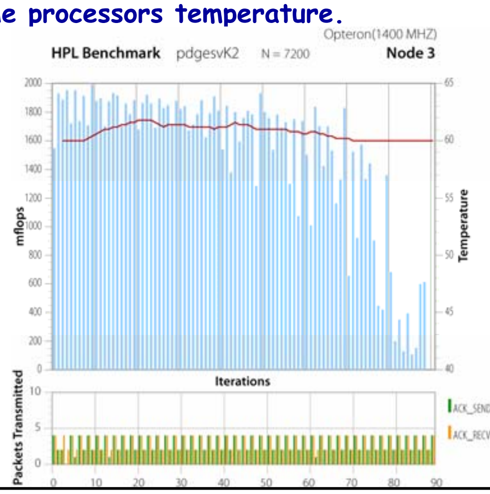
- AMD Opteron provides an on-die thermal diode with anode and cathode brought out to processor pins.
- This diode can be read by an external temperature sensor to determine the processors temperature.



20 41

Temperature Sensor

- AMD Opteron provides an on-die thermal diode with anode and cathode brought out to processor pins.
- This diode can be read by an external temperature sensor to determine the processors temperature.



20 42

Summary of Current Unmet Needs

- Performance / Portability
- Fault tolerance
- Memory bandwidth/Latency
- Adaptability: Some degree of autonomy to self optimize, test, or monitor.
  - Able to change mode of operation: static or dynamic
- Better programming models
  - Global shared address space
  - Visible locality
- Maybe coming soon (incremental, yet offering real benefits):
  - Global Address Space (GAS) languages: UPC, Co-Array Fortran, Titanium, Chapel, X10, Fortress)
    - "Minor" extensions to existing languages
    - More convenient than MPI
    - Have performance transparency via explicit remote memory references
- What's needed is a long-term, balanced investment in hardware, software, algorithms and applications in the HPC Ecosystem.

20 43

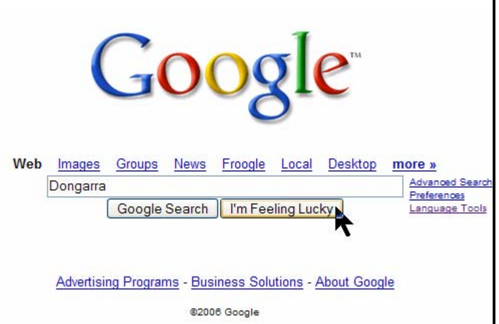
Collaborators / Support

- Top500 Team
  - Erich Strohmaier, NERSC
  - Hans Meuer, Mannheim
  - Horst Simon, NERSC
- Fault Tolerant Work
  - Julien Langou, UTK
  - Jeffery Chen, UTK
- FT-MPI

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

  - Graham Fagg, UTK
  - Edgar Gabriel, UH
  - Thara Angskun, UTK
  - George Bosilca, UTK
  - Jelena Pjesivac-Grbovic, UTK

Office of Science
U.S. DEPARTMENT OF ENERGY



20

20 45

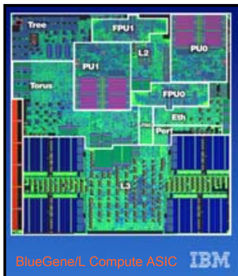
26th List: The TOP10

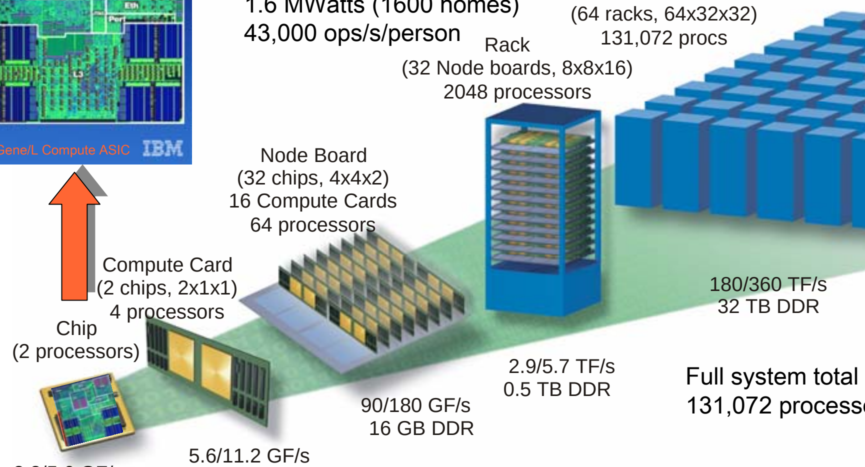
	Manufacturer	Computer	Rmax [TF/s]	Installation Site	Country	Year	#Proc
1	IBM	BlueGene/L eServer Blue Gene	280.6	DOE/NNSA/LLNL	USA	2005 custom	131072
2	IBM	BGW eServer Blue Gene	91.29	IBM Thomas Watson	USA	2005 custom	40960
3	IBM	ASC Purple Power5 p575	63.39	DOE/NNSA/LLNL	USA	2005 custom	10240
4 ~~3~~	SGI	Columbia Altix, Itanium/Infiniband	51.87	NASA Ames	USA	2004 hybrid	10160
5	Dell	Thunderbird Pentium/Infiniband	38.27	Sandia	USA	2005 commod	8000
6 ~~10~~	Cray	Red Storm Cray XT3 AMD	36.19	Sandia	USA	2005 hybrid	10880
7 ~~4~~	NEC	Earth-Simulator SX-6	35.86	Earth Simulator Center	Japan	2002 custom	5120
8 ~~5~~	IBM	MareNostrum PPC 970/Myrinet	27.91	Barcelona Supercomputer Center	Spain	2005 commod	4800
9 ~~6~~	IBM	eServer Blue Gene	27.45	ASTRON University Groningen	Netherlands	2005 custom	12288
10	Cray	Jaguar Cray XT3 AMD	20.53	Oak Ridge National Lab	USA	2005 hybrid	5200

20 46

IBM BlueGene/L #1 131,072 Processors

Total of 18 systems all in the Top100





The compute node ASICs include all networking and processor functionality. Each compute ASIC includes two 32-bit superscalar PowerPC 440 embedded cores (note that L1 cache coherence is not maintained between these cores). (13K sec about 3.6 hours; n=1.8M)

"Fastest Computer"
BG/L 700 MHz 131K proc
64 racks
Peak: 367 Tflop/s
Linpack: 281 Tflop/s
77% of peak

FT-MPI Approach for Dealing with Faults

- **Application checkpointing, MP API+Fault management, automatic.**
  - Application ckpt: application store intermediate results and restart form them
  - MP API+FM: message passing API returns errors to be handled by the programmer
  - Automatic: runtime detects faults and handle recovery

Open-MPI Approach for Dealing with Faults

- **Application checkpointing, MP API+Fault management, automatic.**
  - Application ckpt: application store intermediate results and restart form them
  - MP API+FM: message passing API returns errors to be handled by the programmer
  - Automatic: runtime detects faults and handle recovery
- **Checkpoint coordination: no, coordinated, uncoordinated.**
  - Coordinated: all processes are synchronized, network is flushed before ckpt;
    - all processes rollback from the same snapshot
  - Uncoordinated: each process checkpoint independently of the others
    - each process is restarted independently of the other
- **Message logging: no, pessimistic, optimistic, causal.**
  - Pessimistic: all messages are logged on reliable media and used for replay
  - Optimistic: all messages are logged on non reliable media. If 1 node fails, replay is done according to other nodes logs. If >1 node fail, rollback to last coherent checkpoint
  - Causal: optimistic+Antecedence Graph, reduces the recovery time

FT MM: Perform Computation with Encoded Data

- Assume the original matrix M is distributed into a *p* by *q* processor grid with a 2D block cyclic distribution. Then from processor point of view, the distributed matrix is

$$M = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \end{pmatrix}$$

, where M_{ij} is the local matrix on processor (i , j).

- Define the *row distributed checksum* matrix of M as

$$M^r = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \\ \sum_{i=1}^{p} M_{i1} & \cdots & \sum_{i=1}^{p} M_{iq} \end{pmatrix} = A_r$$

- Define the *column distributed checksum* matrix of M as

$$M^c = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^{q} M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^{q} M_{pj} \end{pmatrix} = B_c$$

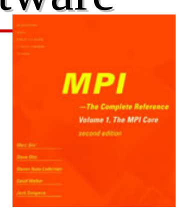
- Define the *full distributed checksum* matrix of M as

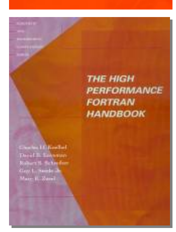
$$M^f = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^{q} M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^{q} M_{pj} \\ \sum_{i=1}^{p} M_{i1} & \cdots & \sum_{i=1}^{p} M_{iq} & \sum_{i=1}^{p}\sum_{j=1}^{q} M_{ij} \end{pmatrix} = C_f$$

Real Crisis With HPC Is With The Software

- Our ability to configure a hardware system capable of 1 PetaFlop (10^{15} ops/s) is without question just a matter of time and $$.
- A supercomputer application and software are usually much more long-lived than a hardware
  - Hardware life typically five years at most…. Apps 20-30 years
  - Fortran and C are the main programming models (still!!)
- The REAL CHALLENGE is Software
  - Programming hasn't changed since the 70's
  - HUGE manpower investment
    - MPI… is that all there is?
  - Often requires HERO programming
  - Investments in the entire software stack is required (OS, libs, etc.)
- Software is a major cost component of modern technologies.
  - The tradition in HPC system procurement is to assume that the software is free… SOFTWARE COSTS (over and over)

"Last Mile" Problem With Software





- Expected to be innovative
  - Proof of concept software generated
- Message Passing Interface (MPI)
  - "assembly language" of parallel computing
  - lowest common denominator
    - portable across architectures and systems
- High-Performance Fortran (HPF)
  - higher level data parallel specification
    - limited to regular data structures
  - we expected too much too soon
    - see Earth System Simulator
- Costs and implications
  - Software productivity is low
  - Next generation of machine will have increased levels of parallelism
  - human productivity
    - low-level programming model
  - software innovation
    - limited development of alternatives

Basic Idea

- Assume
  - we are running a parallel program where $P_i(t)$ denotes the data on the i^{th} processor at time t
  - $P_1(t) + P_2(t) + \ldots + P_n(t) = P_{n+1}(t)$
- If the first processor failed, how can we recover the lost data $P_1(t)$?
  - Answer: $P_1(t) = - P_2(t) - \ldots - P_n(t) + P_{n+1}(t)$
- In this special case, we are lucky enough to be able to recover the lost data without maintaining any checkpoint due to the relationship
  - $P_1(t) + P_2(t) + \ldots + P_n(t) = P_{n+1}(t)$
- Question: can we create this kind of special relationship on purpose ?
  - The answer is YES for many programs doing matrix computations
  - How ?
    - Perform computation with encoded data

Overhead and Scalability Analysis

- Assume a *p* by *p* processor grid and a *n* by *n* local matrix per processor
- Without fault tolerance, the number of calculations on each processor is
  - 2*p*n^3. (because 2*(p*n)^3 calculations by p*p processors)
- With fault tolerance, the number of calculations on each processor is still
  - 2*p*n^3. (the # of calculations per processor does not increase !)
- Overhead for fault tolerance
  - Calculate encoding at the beginning: O (1 / (p*n))
  - Increased communication (due to larger processor grid): O (1 / (p*n))
  - Recover decoding : O (1 / (p*n))
- Note that
  - $O(1/(p*n)) \longrightarrow 0$, as $n, p \longrightarrow \infty$

Example Matrices from Discretizing Boltzmann Equation in the TSI project at ORNL

$$\begin{pmatrix} D_1 & C_1 & & & & \\ B_2 & D_2 & C_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & B_{n-1} & D_{n-1} & C_{n-1} \\ & & & B_n & D_n \end{pmatrix}$$

D_i is dense: m by m.

B_i and C_i are diagonal.

$n = 128 \times 32$	m	n
$G = 12, Q = 4$	386	4,096
$G = 40, Q = 4$	1,282	4,096
$G = 40, Q = 16$	20,482	4,096
$n = 512 \times 512$	m	n
$G = 12, Q = 4$	386	262,144
$G = 40, Q = 4$	1,282	262,144
$G = 40, Q = 16$	20,482	262,144

G, Q, m, and n are parameters used to discretize the problem

55

Prototype Example II: Fault Tolerant Matrix Multiplication (PDGEMM in ScaLAPACK/PBLAS)

- Demonstrate how to survive (adapt to) partial process failures in parallel matrix multiplication
  - Based on FT-MPI library
  - Adapt to failures rather than restart the whole application
  - Can be used in heterogeneous environments
- Use checkpoint-free technique
  - No periodical checkpoint is involved
  - Perform computation with encoded matrices
- Answer four questions
  - what is the overhead of calculating encodings ?
  - what is the overhead of performing computation with encoded matrices?
  - what is the overhead of recovering FT-MPI environment ?
  - what is the overhead of recovering application data ?

20

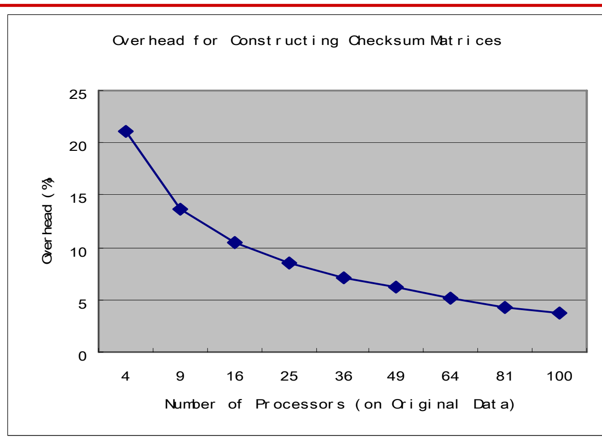
56

PDGEMM: Experiment Configurations

Process grid w/out FT	Process grid w/ FT	Size of the original matrix	Size of the checksum matrix
2 by 2	3 by 3	12,800	19,200
3 by 3	4 by 4	19,200	25,600
4 by 4	5 by 5	25,600	32,000
5 by 5	6 by 6	32,000	38,400
6 by 6	7 by 7	38,400	44,800
7 by 7	8 by 8	44,800	51,200
8 by 8	9 by 9	51,200	57,600
9 by 9	10 by 10	57,600	64,000
10 by 10	11 by 11	64,000	70,400

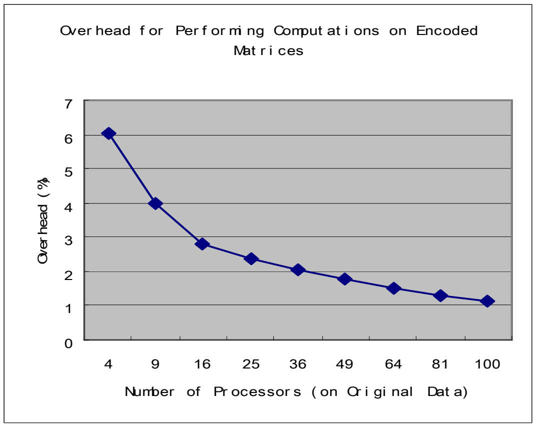
- **Size of local matrices on each process: 6,400 by 6,400**
- **Platform (Grig @ UTK):**
  - 64 nodes, 128 processors, Intel EM64T, 64bit
  - Myrinet
  - FT-MPI + Debian Linux

PDGEMM: The Overhead (%) for Calculating Encodings



- Note that the overhead for encoding is
  - O (1 / (p*n)) → 0, as p → ∞

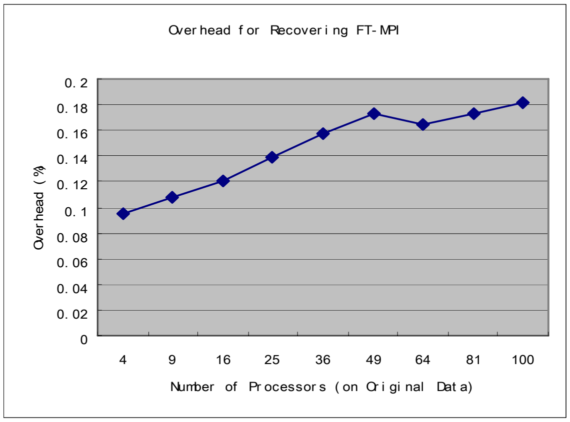
PDGEMM: The Overhead for Performing Computation with Encoded Matrices



- Note that the overhead for performing computation with encoded matrices is
  - O(1 / (p*n) → 0, as p → ∞

20 59

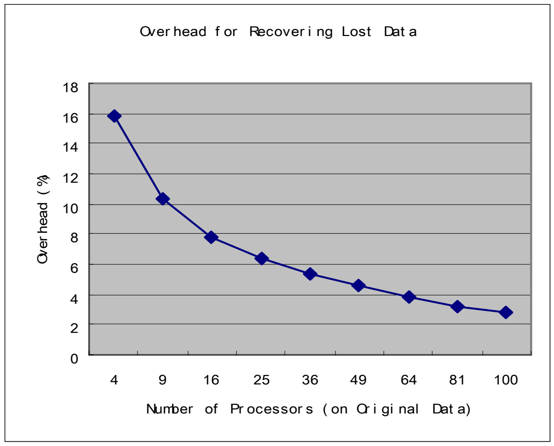
PDGEMM: The Overhead for Recovering FT-MPI Environment



- Note that the time to recover FT-MPI
  - is currently O(p^2)
  - will be improved to O(log p) soon
  - is negligible compared with the time to recover application data

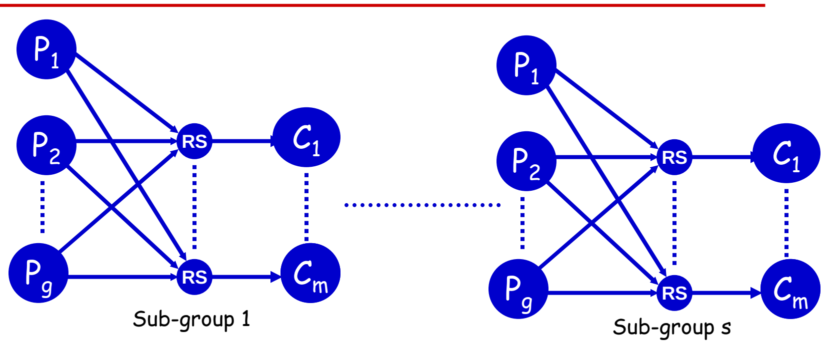
20 60

PDGEMM: The Overhead for Recovering Application Data



- Note that the overhead for recovering the application data is
  - O (1 / (p*n)) → 0, as p → ∞

Coding to Survive Multiple Failures: Subgroup Scheme



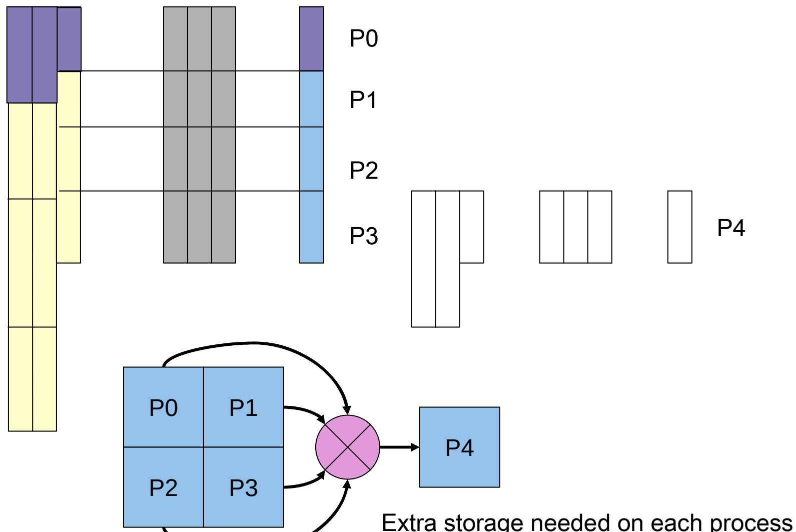
Divide the computational processors into *s* sub-groups (with g procs per group), dedicate *m* checkpoint processors for each sub-group to holding the encodings of the local checkpoint.

The checkpoint overhead (assume pipelined encoding within each sub-group):

T = m * { (1 + O(1/size_ckpt^0.5)) * size_ckpt / bandwidth + g * latency }

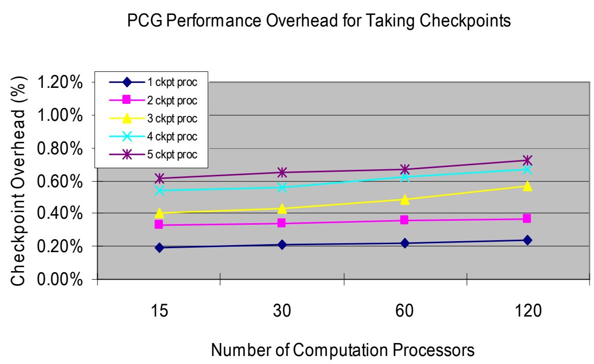
Note that *g* (*g* << # of total procs) is a constant independent of # of total procs, therefore, the checkpoint overhead is independent of # of total procs.

Diskless Version



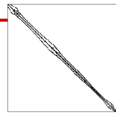
Extra storage needed on each process from the data that is changing.
Actually don't do XOR, add the information.

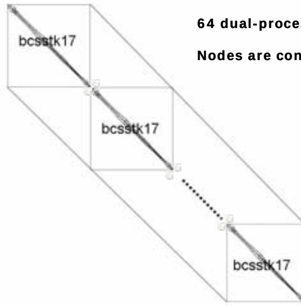
PCG: Performance Overhead of Taking Checkpoints



T (ckpt T)	0 ckpt	1 ckpt	2 ckpt	3 ckpt	4 ckpt	5 ckpt
15 comp	517.8	518.9 (1.0)	519.6 (1.7)	519.8 (2.1)	520.4 (2.8)	521.0 (3.2)
30 comp	532.2	533.3 (1.1)	533.7 (1.8)	534.5 (2.3)	535.1 (3.0)	535.6 (3.5)
60 comp	546.5	547.8 (1.2)	548.0 (2.0)	548.8 (2.7)	549.7 (3.2)	550.1 (3.7)
120 comp	622.9	624.4 (1.5)	625.5 (2.3)	626.7 (3.6)	627.5 (4.2)	628.6 (4.5)

PCG: Performance with Different MPI Implementations





64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

bcsstk17:
- **The size is:**
  10974 x 10974
- **Non-zeros:**
  428650
- **Sparsity:**
  39 non-zeros per row on average
- **Source:**
  Linear equation from elevated pressure vessel

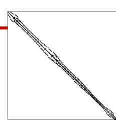
N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI
165K	15	522.5	536.3	517.8
329K	30	532.9	542.9	532.2
658K	60	545.5	553.0	546.5
1317K	120	674.3	624.4	622.9

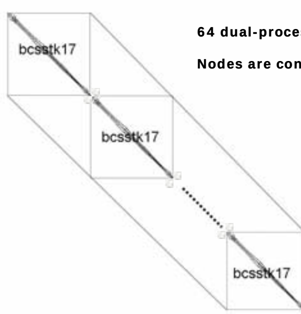
20

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

65

PCG: Performance with Different MPI Implementations





64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

bcsstk17:
- **The size is:**
  10974 x 10974
- **Non-zeros:**
  428650
- **Sparsity:**
  39 non-zeros per row on average
- **Source:**
  Linear equation from elevated pressure vessel

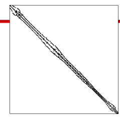
N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI	FT-MPI ckpt /2000 iters
165K	15	522.5	536.3	517.8	518.9
329K	30	532.9	542.9	532.2	533.3
658K	60	545.5	553.0	546.5	547.8
1317K	120	674.3	624.4	622.9	624.4

20

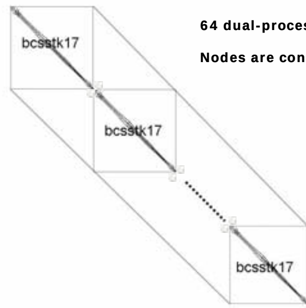
http://icl.cs.utk.edu/ft-mpi/

66

PCG: Performance with Different MPI Implementations



bcsstk17:
- **The size is:**
  - 10974 x 10974
- **Non-zeros:**
  - 428650
- **Sparsity:**
  - 39 non-zeros per row on average
- **Source:**
  - Linear equation from elevated pressure vessel



64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

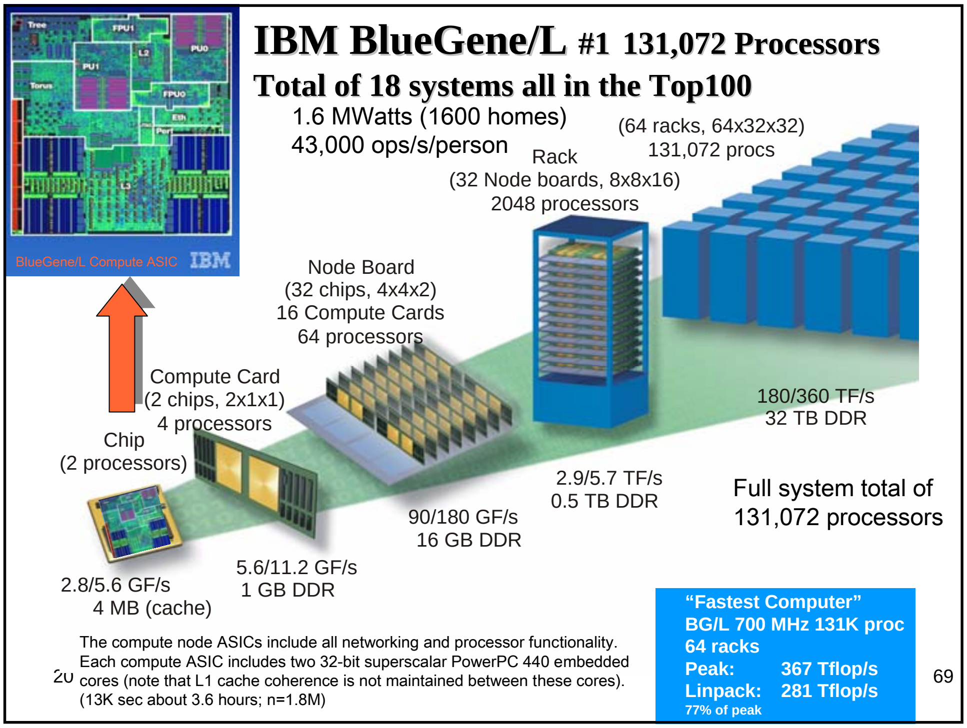
N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI	FT-MPI ckpt /2000 iters	FT-MPI exit 1 proc @10000 iters
165K	15	522.5	536.3	517.8	518.9	521.7
329K	30	532.9	542.9	532.2	533.3	537.5
658K	60	545.5	553.0	546.5	547.8	554.2
1317K	120	674.3	624.4	622.9	624.4	637.1

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

Reliability of Large Systems

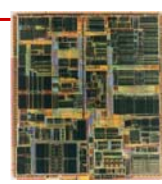
(Source: Daniel Reed, UNC)

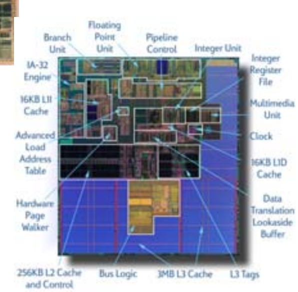
Machine	# CPU	Reliability
ASCI Q	8,192	MTBI 6.5 hr. 114 unplanned outages/month. HW outage sources: storage, CPU, memory *
ASCI White	8,192	MTBF 5 hr ('01) and 40 hr ('03) HW outage sources: storage, CPU, 3rd party hardware **
NERSC Seaborg	6,656	MTBI 14 days. MTTR 3.3 hr Availability 98.74%. SW is main outage source. ***
PSC Lemieux	3,016	MTBI 9.7 hr Availability 98.33% ****
Google	~15,000	20 reboots/day. 2-3% machines replaced/year. HW outage sources: storage, memory

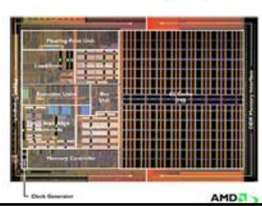


Commodity Processors

- **Intel Pentium Nocona**
  - 3.6 GHz, peak = 7.2 Gflop/s
  - Linpack 100 = 1.8 Gflop/s
  - Linpack 1000 = 4.2 Gflop/s
- **Intel Itanium 2**
  - 1.6 GHz, peak = 6.4 Gflop/s
  - Linpack 100 = 1.7 Gflop/s
  - Linpack 1000 = 5.7 Gflop/s
- **AMD Opteron**
  - 2.6 GHz, peak = 5.2 Gflop/s
  - Linpack 100 = 1.6 Gflop/s
  - Linpack 1000 = 3.9 Gflop/s



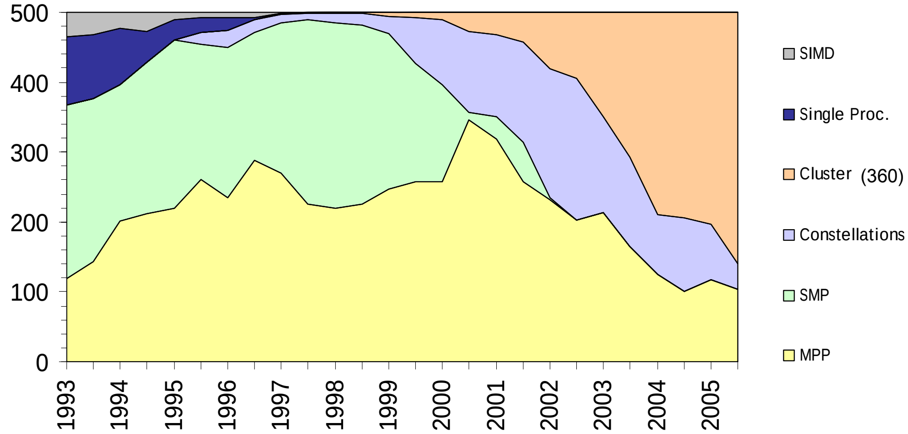




20

70

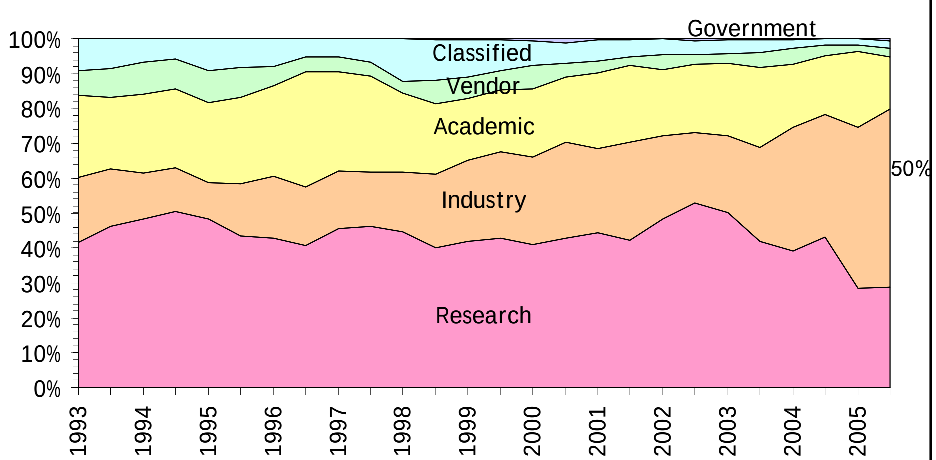
Architectures / Systems



Cluster: Commodity processors & Commodity interconnect

Constellation: # of procs/node ⩾ nodes in the system

Customer Segments / Performance



A PetaFlop Computer by the End of the Decade

- 10 Companies working on a building a Petaflop system by the end of the decade.
  - Cray
  - IBM
  - Sun
  - Dawning
  - Galactic
  - Lenovo
  - Hitachi
  - NEC
  - Fujitsu
  - Bull

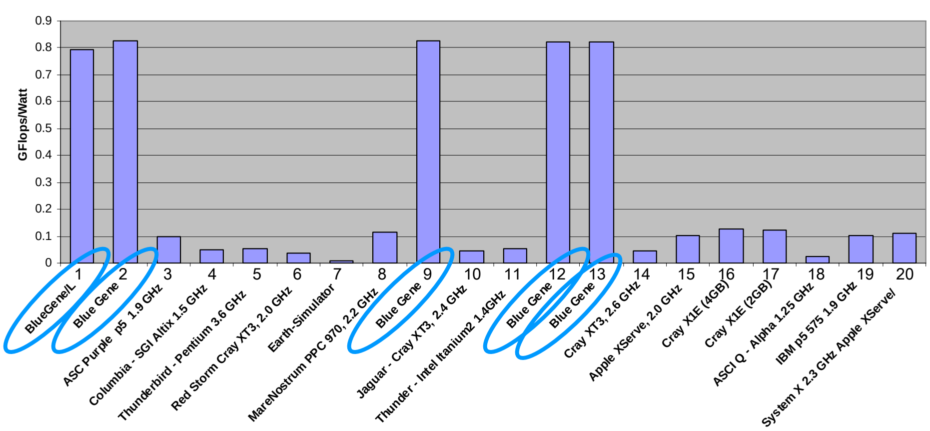
} HPCS (Cray, IBM, Sun)

} Chinese Companies (Dawning, Galactic, Lenovo)

} Japanese "Life Simulator" (10 Pflop/s) (Hitachi, NEC, Fujitsu)

20 73

Fuel Efficiency: GFlops/Watt



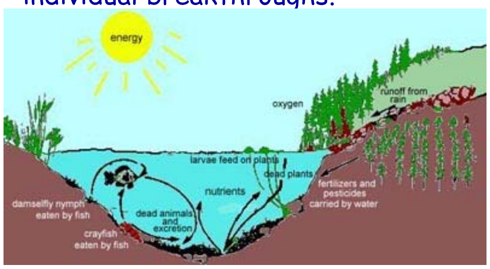
20 Top 20 systems 74

Based on processor power rating only (3,>100,>800)

Future Challenge: Developing the Ecosystem for HPC

From the NRC Report on "The Future of Supercomputing":

- Hardware, software, algorithms, tools, networks, institutions, applications, and people who solve supercomputing applications can be thought of collectively as a multifaceted ecosystem
- Research investment in HPC should be informed by the ecosystem point of view - progress must come on a broad front of interrelated technologies, rather than in the form of individual breakthroughs.



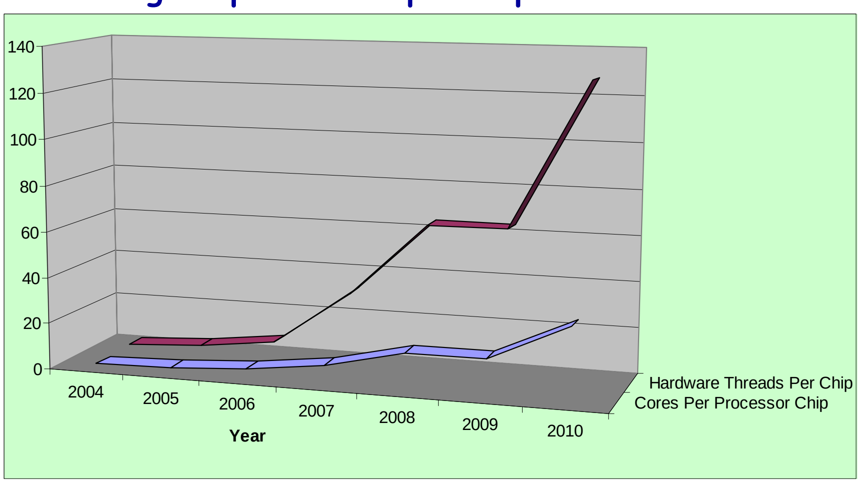
A supercomputer ecosystem is a continuum of computing platforms, system software, algorithms, tools, networks, and the people who know how to exploit them to solve computational science applications.

20

75

CPU Desktop Trends 2004-2010

- **Relative processing power will continue to double every 18 months**
- **256 logical processors per chip in late 2010**



76

Third Approach

- **Checkpointless methods for dense algorithms**
  - We need extra processors to participate in the computation
    - The extra processors carry the active checksum
  - No roll back needed; just compute what's missing and carry on.

FT ScaLAPACK: Perform Computation with Encoded Data

- Assume the original matrix M is distributed into a *p* by *q* processor grid with a 2D block cyclic distribution. Then from processor point of view, the distributed matrix is $M = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \end{pmatrix}$, where M_{ij} is the local matrix on processor (*i,j*).
- Define the *row distributed checksum* matrix of M as

$$M^r = \begin{pmatrix} M_{11} & \cdots & M_{1q} \\ \vdots & \cdots & \vdots \\ M_{p1} & \cdots & M_{pq} \\ \sum_{i=1}^{p} M_{i1} & \cdots & \sum_{i=1}^{p} M_{iq} \end{pmatrix}$$

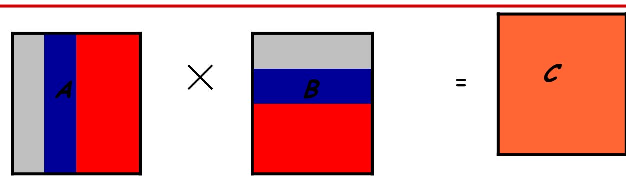
- Define the *column distributed checksum* matrix of M as

$$M^c = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^{q} M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^{q} M_{pj} \end{pmatrix}$$

- Define the *full distributed checksum* matrix of M as

$$M^f = \begin{pmatrix} M_{11} & \cdots & M_{1q} & \sum_{j=1}^{q} M_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ M_{p1} & \cdots & M_{pq} & \sum_{j=1}^{q} M_{pj} \\ \sum_{i=1}^{p} M_{i1} & \cdots & \sum_{k=1}^{p} M_{iq} & \sum_{i=1}^{p}\sum_{j=1}^{q} M_{ij} \end{pmatrix}$$

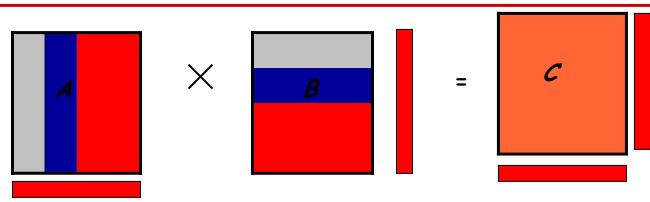
An Example: ScaLAPACK Matrix Multiplication



20

79

An Example: ScaLAPACK Matrix Multiplication



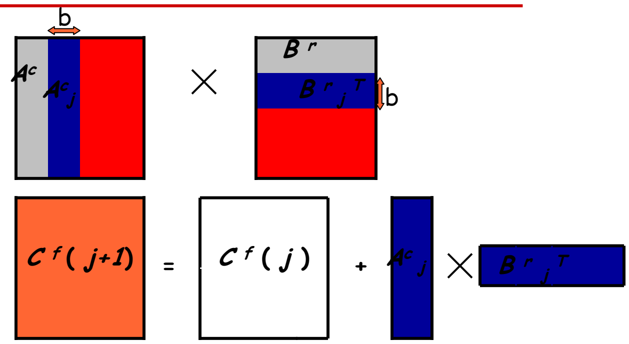
20

80

An Example: ScaLAPACK Matrix Multiplication

FT-PDGEMM operates on A^c, B^r and C^f

At the j^{th} iteration:



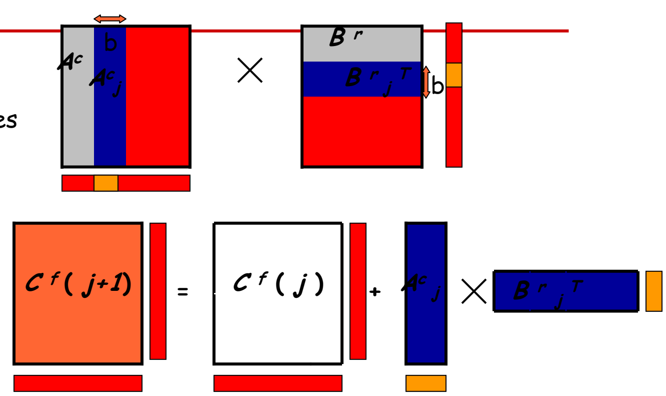
- Theorem:
  At the end of each iteration, the checksum relationship in A^r, B^c, and C^f are still maintained
- Conclusion
  - Single failure during computation can be recovered from the checksum relationship
  - By using a floating-point version Reed-Solomon code, multiple failures can be

20 81

An Example: ScaLAPACK Matrix Multiplication

FT-PDGEMM operates on A^c, B^r and C^f

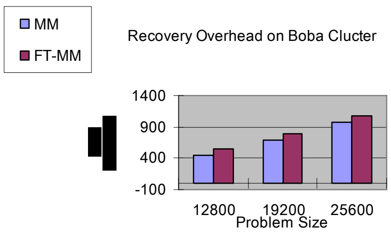
At the j^{th} iteration:

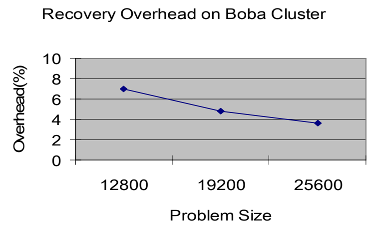


- Theorem:
  At the end of each iteration, the checksum relationship in A^r, B^c, and C^f are still maintained
- Conclusion
  - Single failure during computation can be recovered from the checksum relationship
  - By using a floating-point version Reed-Solomon code, multiple failures can be

20 82

Overhead for Recovery





Size of Matrix	12,800	19,200	25,600
Process Grid w/o FT	2 by 2	3 by 3	4 by 4
Process Grid w/ FT	3 by 3	4 by 4	5 by 5
Execution time w/o FT	453.9	699.1	965.3
Execution time w/ Recvr	543.4	800.6	1078.4
Time for Recovery	31.6	33.4	34.9
Overhead for Recovery	7.0	4.8	3.6

An Example: ScaLAPACK Matrix Multiplication

A =

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

B =

10	10	20	20
10	10	20	20
30	30	40	40
30	30	40	40

C =

100	100	200	200
100	100	200	200
300	300	400	400
300	300	400	400

Assume the original matrix are distributed into a 2 by 2 processor grid with a 2D block cyclic distribution, where both the row block size and the column block size are 1.

Encode matrices into 3 by 3 processor grid:

A^r =

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4
4	4	6	6
4	4	6	6

B^c =

10	10	20	20	30	30
10	10	20	20	30	30
30	30	40	40	70	70
30	30	40	40	70	70

C^f =

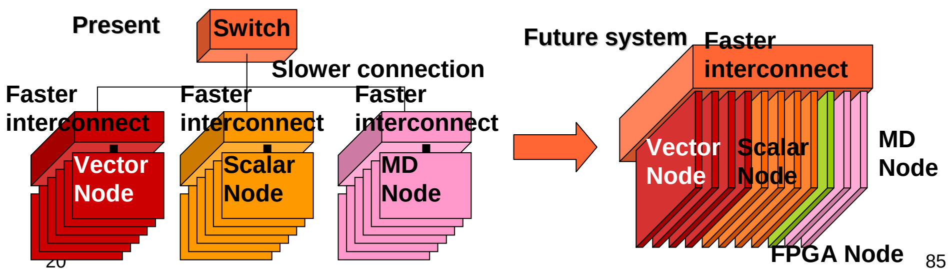
100	100	200	200	300	300
100	100	200	200	300	300
300	300	400	400	700	700
300	300	400	400	700	700
400	400	600	600	1000	1000
400	400	600	600	1000	1000

PDGEMM operates on A, B, and C

FT-PDGEMM operates on A^c, B^r and C^f

Japanese: Tightly-Coupled Heterogeneous System

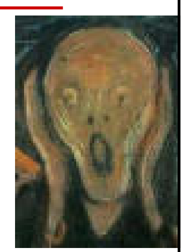
- Would like to get to 10 PetaFlop/s by 2011
- Scalable, fits any computer center
  - Size, cost, ratio of components
- Easy and low-cost to develop new component
- Scale merit of components

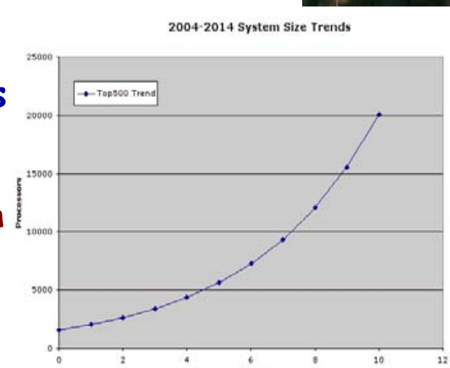


85

How Big Is Big?

- Every 10X brings new challenges
  - 64 processors was once considered large
    - it hasn't been "large" for quite a while
  - 1024 processors is today's "medium" size
  - 8096 processors is today's "large"
    - we're struggling even here

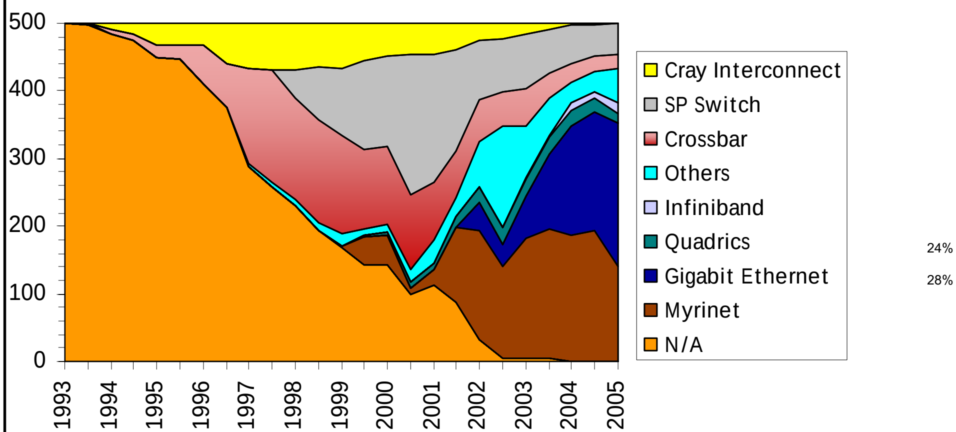




- 100K processor systems
  - are in construction
  - we have fundamental challenges in dealing with machines of this size
  - … and little in the way of programming support

20

Interconnects / Systems

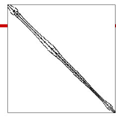


Myrinet and GigE > 50% of market

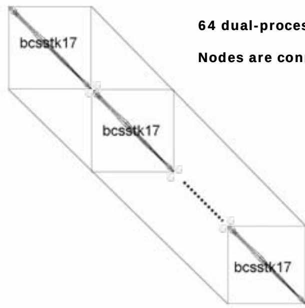
Real Crisis With HPC Is With The Software

- **Programming is stuck**
  - Arguably hasn't changed since the 60's
- **It's time for a change**
  - Complexity is rising dramatically
    - highly parallel and distributed systems
      - From 10 to 100 to 1000 to 10000 to 100000 of processors!!
    - multidisciplinary applications
- **A supercomputer application and software are usually much more long-lived than a hardware**
  - Hardware life typically five years at most.
  - Fortran and C are the main programming models
- **Software is a major cost component of modern technologies.**
  - The tradition in HPC system procurement is to assume that the software is free.
- **We have too few ideas about how to solve this problem.**

PCG: Performance with Different MPI Implementations



bcsstk17:
- **The size is:** 10974 x 10974
- **Non-zeros:** 428650
- **Sparsity:** 39 non-zeros per row on average
- **Source:** Linear equation from elevated pressure vessel



64 dual-processor 2.4 GHz AMD Opteron nodes

Nodes are connected with a Gigabit Ethernet.

N	Procs	LAM-7.0.4	MPICH2-1.0	FT-MPI
165K	15	522.5	536.3	517.8
329K	30	532.9	542.9	532.2
658K	60	545.5	553.0	546.5
1317K	120	674.3	624.4	622.9

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

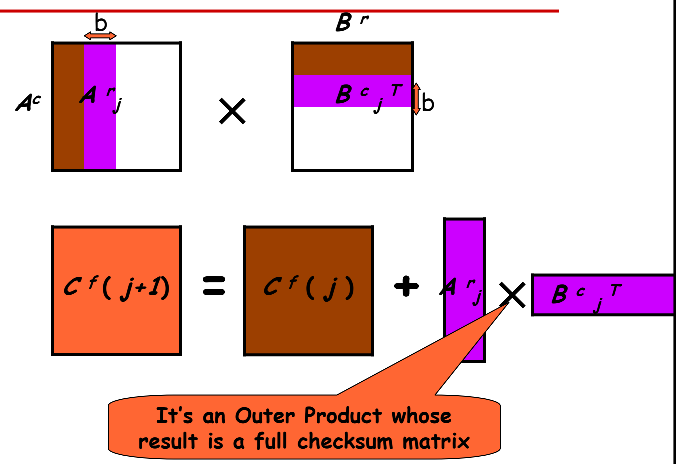
Self Adapting Numerical Software

- Optimizing software to exploit the features of a given system has historically been an exercise in hand customization.
  - Time consuming and tedious
  - Hard to predict performance from source code
  - Must be redone for every architecture and compiler
    - Software technology often lags hardware/architecture
    - Best algorithm may depend on input, so some tuning may be needed at run-time.
- With good reason scientists expect their computing tools to serve them and not the other way around.
- There is a need for quick/dynamic deployment of optimized routines.
  - ATLAS, PhiPAC, BeBoP, Spiral, FFTW, GCO, …

An Example: Matrix Multiplication

FT-PDGEMM operates on A^c, B^r and C^f :

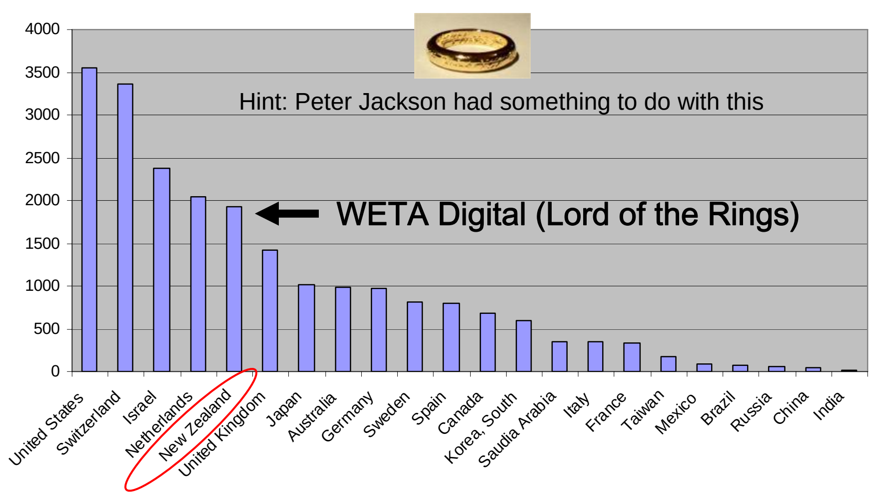
At the j^{th} iteration:



- Therefore:
  At the end of each iteration, the checksum relationship in A^r, B^c, and C^f will be maintained
- Conclusion:
  Single failure during computation can be recovered from the checksum relationship

KFlop/s per Capita (Flops/Pop)

Based on the June 2005 - Top500 only



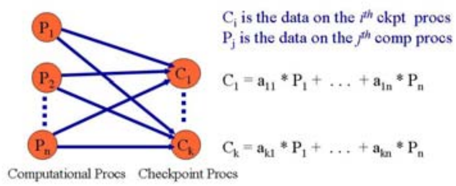
Has nothing to do with the 47.2 million sheep in NZ

Reed-Solomon Approach

*A*P = C, where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$



Reed-Solomon Approach

*A*P = C, where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ \not{P_2} \\ \not{P_3} \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Say 2 processors fail, P_2 and P_3.

Reed-Solomon Approach

$A*P = C$, *where A is k x p made up of random numbers, P is p x n, C is k x n*

Here using 4 processors and 3 Ckpt processors:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Say 2 processors fail, P_2 and P_3.

Take a subset of A's (colunm 2 and 3) and solve for P_2 and P_3.

$$\begin{pmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

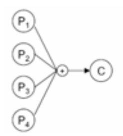
Could use GF(2). Signal processing aps do this. In that case, A is Vandermonde or Cauchy matrix. (Need to have any subset of A be non singular.) We use A as a random matrix.

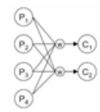
PCG: Impact of Round-Off Errors in Recovery

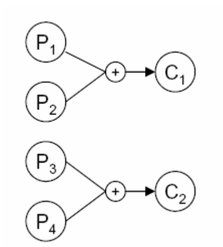
- **If no failure occurs**
  - **PCG computation is not affected by the round-off errors of checkpoint**
- **Whenever there is a failure**
  - **The recovered data is not exactly the same as original data due to round-off errors in the recovery, however…**

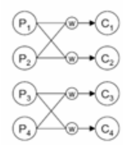
# of Iters	0 proc	1 proc	2 proc	3 proc	4 proc	5 proc
1.0e-10	2917	2918	2918	2915	2917	2917
1.0e-12	3141	3136	3142	3138	3140	3147
1.0e-14	3383	3385	3387	3384	3385	3393
1.0e-16	3599	3596	3595	3590	3601	3599
1.0e-18	3806	3809	3814	3802	3806	3802

Run PCG with 120 computation processors until the relative residual ||r|| / ||b|| < 10^{-i}.
Simulate some process failures at the 2000th iteration by exiting some processes.
The above table reports the number of iterations for different number of processes failures.









PCG: Impact of Round-Off Errors in Recovery

- **If no failure occurs**
  - **PCG computation is not affected by the round-off errors of checkpoint**
- **Whenever there is a failure**
  - **The recovered data is not exactly the same as original data due to round-off errors in the recovery, however…**

# of Iters	0 proc	1 proc	2 proc	3 proc	4 proc	5 proc
1.0e-10	2917	2918	2918	2915	2917	2917
1.0e-12	3141	3136	3142	3138	3140	3147
1.0e-14	3383	3385	3387	3384	3385	3393
1.0e-16	3599	3596	3595	3590	3601	3599
1.0e-18	3806	3809	3814	3802	3806	3802

Run PCG with 120 computation processors until the relative residual $||r|| / ||b|| < 10^{-i}$.
Simulate some process failures at the 2000th iteration by exiting some processes.
The above table reports the number of iterations for different number of processes failures.

Next Steps

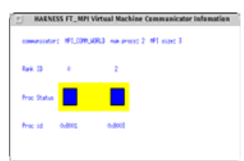
Investigate ideas for 1K to 10K processors, then to BG/L.

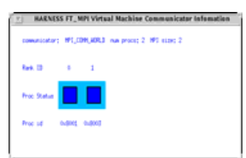
- Software to determine the checkpointing interval and number of checkpoint processors from the machine characteristics.
  - Perhaps use historical information.
- Local checkpoint and restart algorithm.
  - Coordination of local checkpoints.
  - Processors hold backups of neighbors.
- Have the checkpoint processes participate in the computation and do data rearrangement when a failure occurs.
  - Use p processors for the computation and have k of them hold checkpoint.
- Generalize the ideas to provide a library of routines to do the diskless check pointing.
- Look at "real applications" and investigate "Lossy" algorithms.
- FT-MPI available today and one of the contributions to Open MPI.

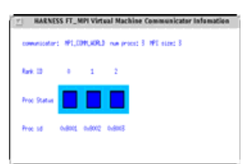
20 99

FT-MPI Failure Recovery Modes

- ABORT: Just do as other MPI implementations.
- BLANK: Leave hole in communicator.
- SHRINK: Re-order processes to make a contiguous communicator.
  - Some ranks change
- REBUILD: Re-spawn lost processes and add them to MPI_COMM_WORLD.







20 100

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.
  - Complete reimplementation, not based on other implementations.
- Gives the application the possibility to recover from a process-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

20 101

Sum Computed; Not XOR

$C_1 + C_2 + \ldots C_K = C_{K+1}$

To recover from a lose of C_2 :

$C_2 = C_{K+1} - C_1 - C_3 - \ldots C_K$

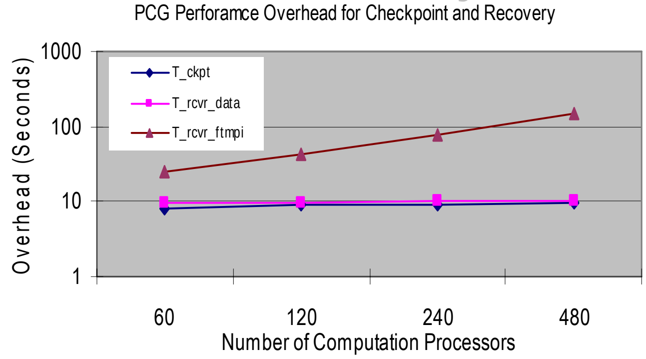
- For a single failure XOR is fine.
- For more than one failure will require GF(2) arithmetic
  - OK for the XOR but need to solve a system of equations in GF(2), will need +, *, / over GF(2)
- Starting to think of reversing the computation to get back to checkpoint state.
- Think of running the program backwards until reaching the checkpoint state.

  $y_i = y_i + a*x_i$
  - Undo computation by:

  $y_i = y_i - a*x_i$
  - Round off errors generated getting back to ckpt

20 102

PCG: Preliminary Performance



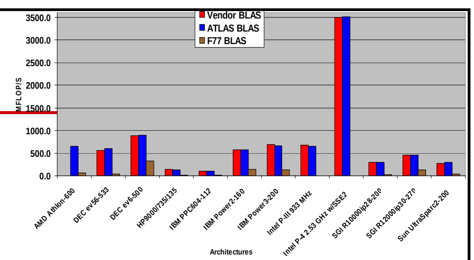
IBM RS/6000 SP w/176 Winterhawk II thin nodes (each with four 375 MHz Power3-II processors)

Run PCG for 5000 iterations and take checkpoint every 1000 iterations (about 5 minutes)
Simulate a failure of one node by exiting 4 processes at the 3000-th iteration.
Matrix size scales with the processors used, i.e. 60 procs: n=658,440; 480 procs: n=5.2M

Time (Sec)	T_pcg_comp	T_ckpt	T_rcvr_data	T_rcvr_ftmpi	T_tot
60 procs	1399.1	8.0	9.8	24.8	1441.7
120 procs	1429.3	9.2	9.9	42.1	1490.5
240 procs	1461.1	9.2	10.0	77.2	1557.5
480 procs	1531.1	9.7	10.1	146.1	1697.0

20 103

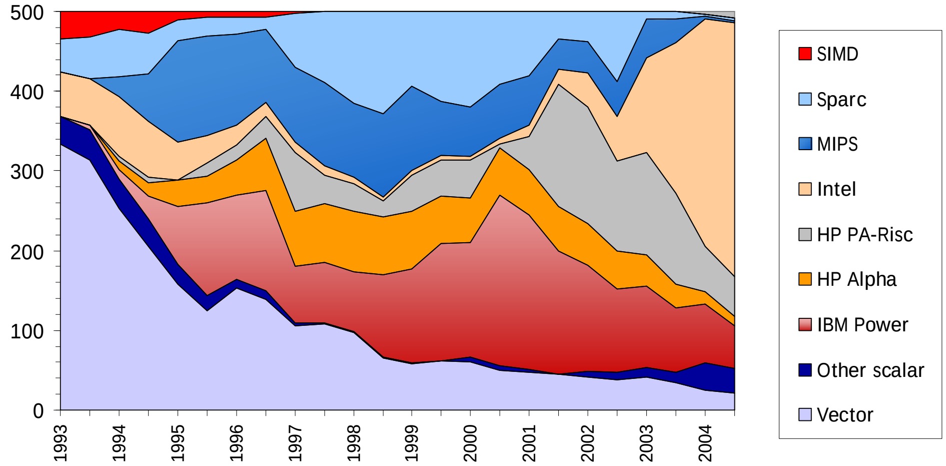
Software Generation Strategy - ATLAS BLAS



- Parameter study of the hw
- Generate multiple versions of code, w/difference values of key performance parameters
- Run and measure the performance for various versions
- Pick best and generate library
- Level 1 cache multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- Similar to FFTW and Johnsson, UH
- Takes ~ 20 minutes to run, generates Level 1,2, & 3 BLAS
- "New" model of high performance programming where critical code is machine generated using parameter optimization.
- Designed for modern architectures
  - Need reasonable C compiler
- Today ATLAS in used within various ASCI and SciDAC activities and by Matlab, Mathematica, Octave, Maple, Debian, Scyld Beowulf, SuSE,…

20 See: http://icl.cs.utk.edu/atlas/ joint with Clint Whaley & Antoine Petitet 104

Processor Types



Today's Processors

- pipelining (superscalar, OOO, VLIW, branch prediction, predication)
- simultaneous multithreading (SMT, Hyper-Threading, multi-core)
- SIMD vector instructions (VIS, MMX/SSE, AltiVec)
- caches and the memory hierarchy
- Intel added 36 instructions per year to IA-32, or 3 instructions per month!

Motivation Self Adapting Numerical Software (SANS) Effort

- Optimizing software to exploit the features of a given system has historically been an exercise in hand customization.
  - Time consuming and tedious
  - Hard to predict performance from source code
  - Must be redone for every architecture and compiler
    - Software technology often lags architecture
    - Best algorithm may depend on input, so some tuning may be needed at run-time.
- There is a need for quick/dynamic deployment of optimized routines.

20 107

Linpack (100x100) Analysis

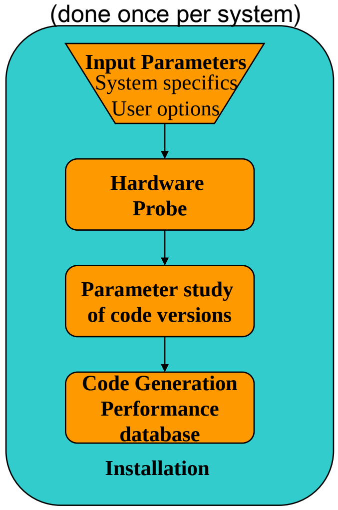
- Compaq 386/SX20 SX with FPA - .16 Mflop/s
- Pentium IV – 2.8 GHz – 1.3 Gflop/s
- 12 years ➔ we see a factor of ~ 8125
- Moore's Law says something about a factor of 2 every 18 months or a factor of 256 over 12 years
- Seem to be missing a factor of 32 …
  - Clock speed increase = 128x
  - External Bus Width & Caching –
    - 16 vs. 64 bits = 4x
  - Floating Point -
    - 4/8 bits multi vs. 64 bits (1 clock) = 8x
  - Compiler Technology = 2x
- However the theoretical peak for that Pentium 4 is 5.6 Gflop/s and here we are only getting 1.3 Gflop/s
  - Still a factor of 4.25 off of peak

Complex set of interaction between
Users' applications
Algorithm
Programming language
Compiler
Machine instruction
Hardware
Many layers of translation from the application to the hardware
Changing with each generation

20 108

Performance Tuning Methodology

Software Installation
(done once per system)



Software Generation Strategy - **ATLAS BLAS**
http://www.netlib.org/atlas/

- Parameter study of the hw
- Generate multiple versions of code, w/difference values of key performance parameters
- Run and measure the performance for various versions
- Pick best and generate library
- Optimize over 8 parameters
  - Cache blocking
  - Register blocking (2)
  - FP unit latency
  - Memory fetch
  - Interleaving loads & computation
  - Loop unrolling
  - Loop overhead minimization
- Similar to FFTW

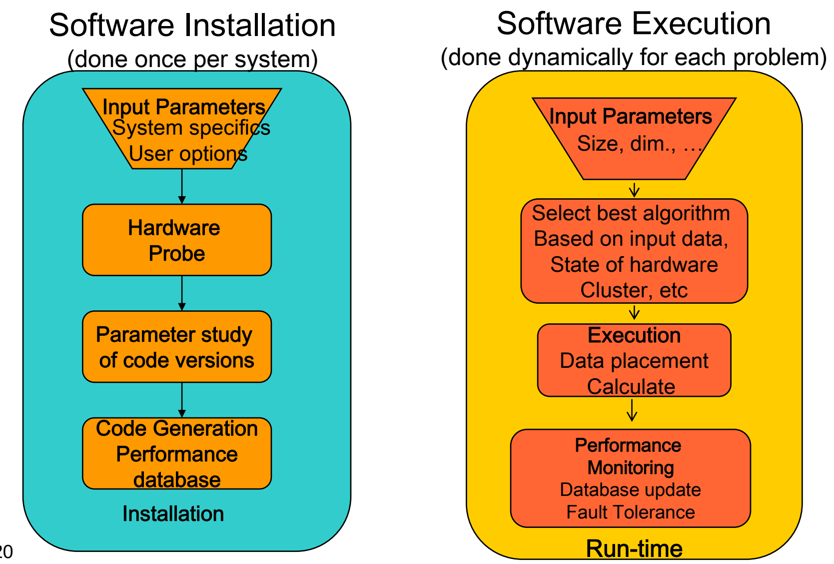
20 109

Self Adapting Numerical Software - SANS Effort

- Provide software technology to aid in high performance on commodity processors, clusters, and grids.
- Pre-run time (library building stage) and run time optimization.
- Integrated performance modeling and analysis
- Automatic algorithm selection – polyalgorithmic functions
- Automated installation process
- Can be expanded to areas such as communication software and selection of numerical algorithms

20 110

Performance Tuning Methodology

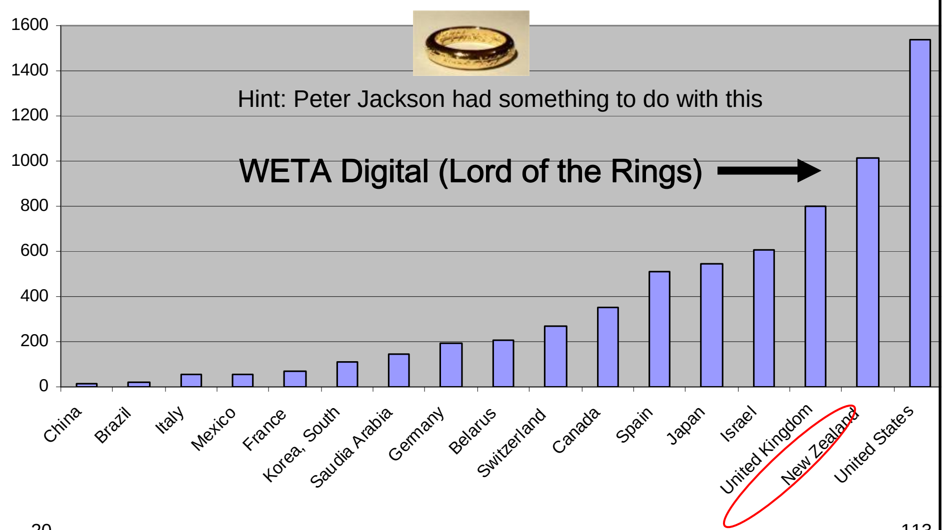


20

111

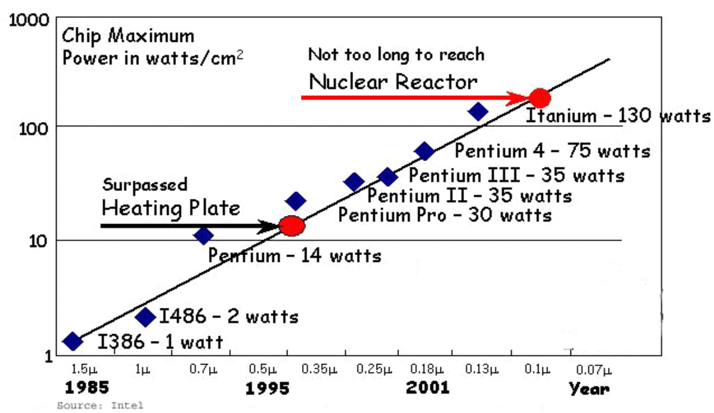
KFlop/s per Capita (Flops/Pop)

Based on the November 2004 Top500 only



Has nothing to do with the 47.2 million sheep in NZ

Today's CPU Architecture



Moore's Law for Power Consumption

Heat is becoming an unmanageable problem

NASA Ames: SGI Altix Columbia 10,240 Processor System (#3)

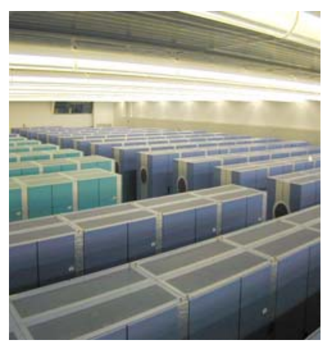
- Architecture: Hybrid Technical Server Cluster
- Vendor: SGI based on Altix systems
- Deployment: 2004
- Node:
  - 1.5 GHz Itanium-2 Processor
  - 512 procs/node (20 cabinets)
  - Dual FPU's / processor

- System:
  - 20 Altix NUMA systems @ 512 procs/node = 10240 procs
  - 320 cabinets (estimate 16 per node)
  - Peak: 61.4 Tflop/s ; LINPACK: 52 Tflop/s
- Interconnect:
  - FastNumaFlex (custom hypercube) within node
  - Infiniband between nodes
- Pluses:
  - Large and powerful DSM nodes
- Potential problems (Gotchas):
  - Power consumption - 100 kw per node (2 Mw total)

20 115

(Japanese) Earth Simulator (#4)

- Architecture: Custom Vector Cluster
- Vendor: NEC
- Deployment Date: 2002
- Node:
  - 500 MHz/1GHz SX-6 vector processor
  - 8 pe's/node
  - 8 vector pipes/ pe
  - 8 Gflops/processor peak



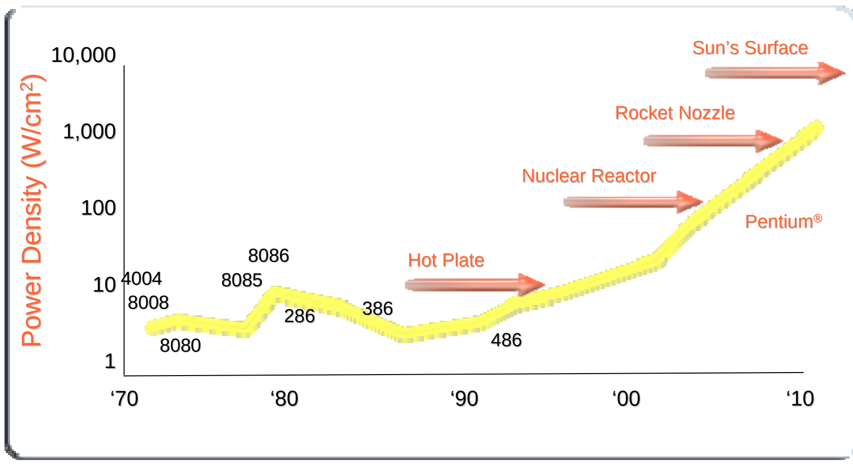
- System:
  - 5120 processors / 640 cabinets
  - Peak: 41.1 Tflop/s
- Interconnect:
  - Custom 640x640 crossbar
- Pluses:
  - High fraction of peak (30% typical)
- Gotchas:
  - No internet access (currently)
  - Cost (estimated $350 M)

20 116

Today's CPU Architecture:

Heat becoming an unmanageable problem

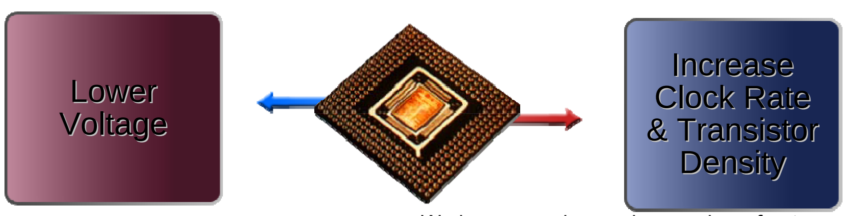
Increasing the number of gates into a tight knot and decreasing the cycle time of the processor

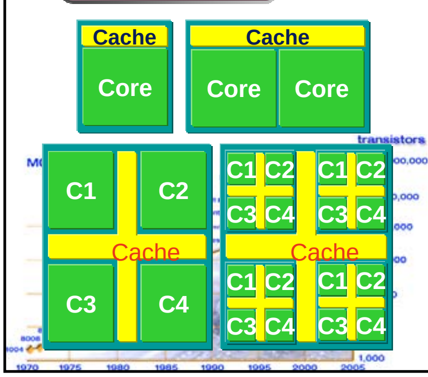


Intel Developer Forum, Spring 2004 - Pat Gelsinger
(Pentium at 90 W)

Square relationship between the cycle time and power.

Increasing CPU Performance: A Delicate Balancing Act





We have seen increasing number of gates on a chip and increasing clock speed.

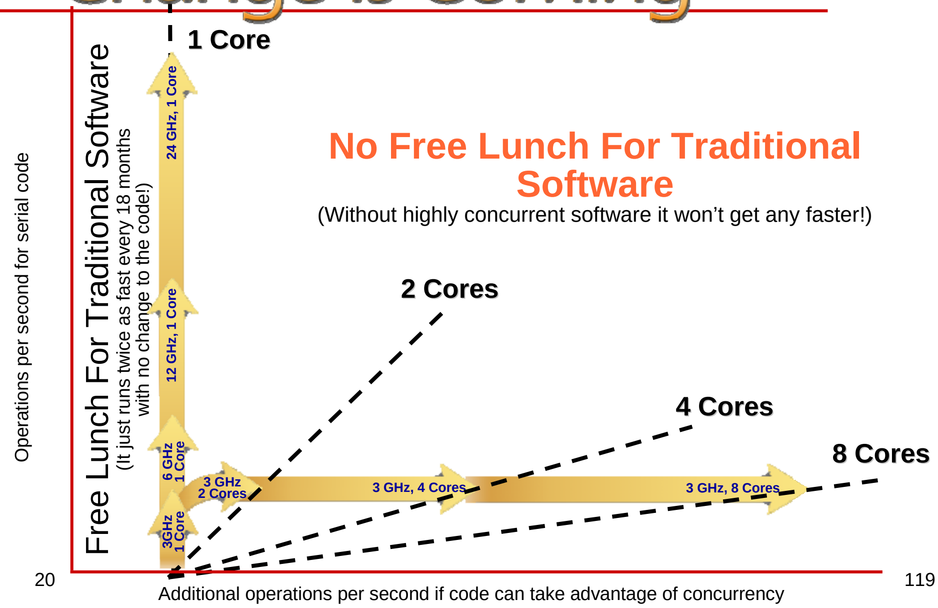
Heat becoming an unmanageable problem, Intel Processors > 100 Watts

We will not see the dramatic increases in clock speeds in the future.

However, the number of gates on a chip will continue to increase.

Intel Yonah will double the processing power on a per watt basis.

Change Is Coming

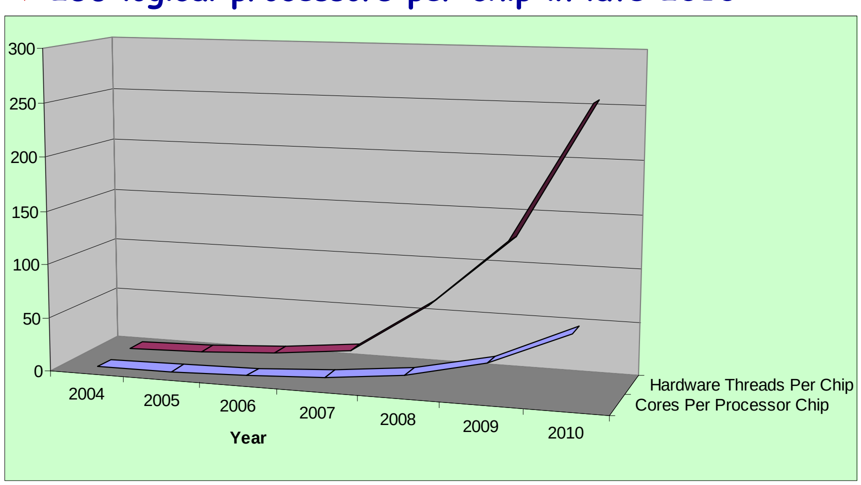


20

119

CPU Desktop Trends 2004-2010

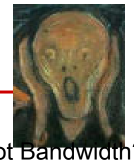
- Relative processing power will continue to double every 18 months
- 256 logical processors per chip in late 2010



120

Commodity Processor Trends

Bandwidth/Latency is the Critical Issue, not FLOPS



Got Bandwidth?

	Annual increase	Typical value in 2005
Single-chip floating-point performance	59%	4 GFLOP/s
Front-side bus bandwidth	23%	1 GWord/s = 0.25 word/flop
DRAM latency	(5.5%)	70 ns = 280 FP ops = 70 loads

20 Source: *Getting Up to Speed: The Future of Supercomputing*, National Research Council, 222 pages, 2004, National Academies Press, Washington DC, ISBN 0-309-09502-6. 121