



# *Fault Tolerance in Linear Algebra Algorithms and Software*

Jack Dongarra  
University of Tennessee  
and  
Oak Ridge National Laboratory

2/15/2006

1



## Fault Tolerance: Motivation

### ◆ Trends in HPC:

- High end systems with thousand of processors
- Move to multicore chips

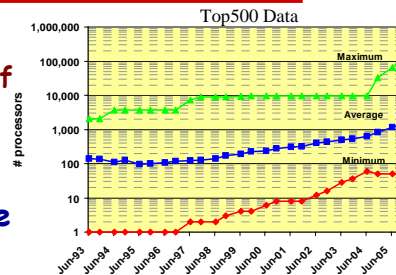
### ◆ Increased probability of a node failure

- 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.



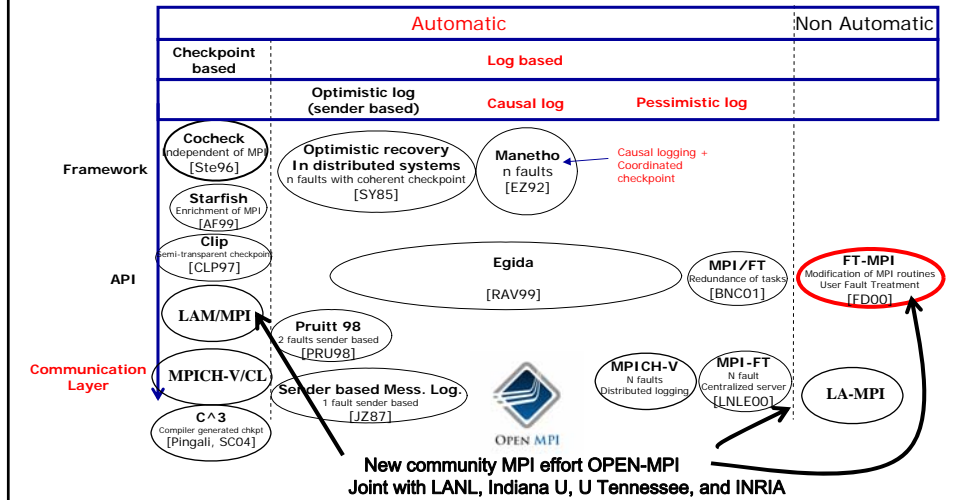
35

2



# 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.



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

- ◆ Define the behavior of MPI in case an 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 reimplementaion, 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



## 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
  - 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

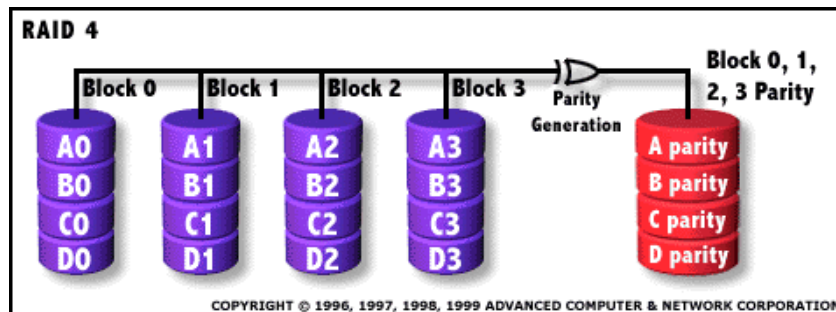
33

5



## Disk-less Checkpointing

- ◆ **Similar to RAID for disks.**



- ◆ **If  $X = A \text{ XOR } B$  then this is true:**
  - $X \text{ XOR } B = A$
  - $A \text{ XOR } X = B$

33

6

**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 :  $T = \# \text{ of procs } * \text{ size\_ckpt } / \text{ bandwidth}$

33 7

**First Approach: Diskless Checkpointing**  
(K. Li & J. Plank, et. al.)

- 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):

$T = \log ( \# \text{ of procs } ) * \text{size\_ckpt } / \text{ bandwidth } + \log ( \# \text{ of procs } ) * \text{latenc}$

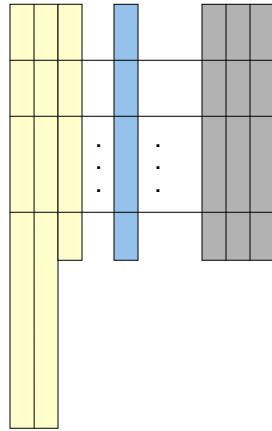
33



# CG Parallel Version

Think of the data like this

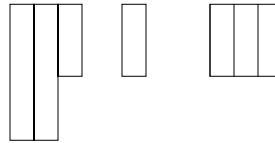
A    b    3 vectors



33

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.

9



# FT PCG Algorithm Analysis

Compute  $r^{(0)} = b - Ax^{(0)}$  for some initial guess  $x^{(0)}$

for  $i = 1, 2, \dots$

  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.

33

Global operation in Checkpoint: encoding the local checkpoint.

10



# FT PCG Algorithm Analysis

```

Compute  $r^{(0)} = b - Ax^{(0)}$  for some initial guess  $x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  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.

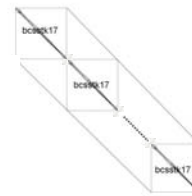
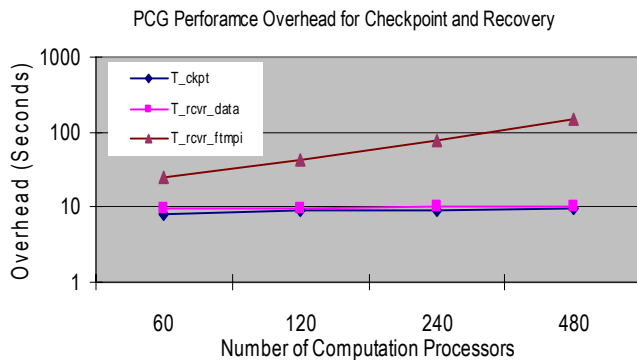
33

Global operation in Checkpoint: encoding the local checkpoint.  
Global operation in checkpoint can be localized by sub-group.

11



# 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:  $n=5.2M$

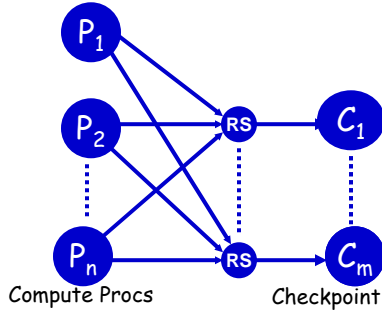
Time (Sec)	Time w/o checkpoint	Checkpoint time	Data Recovery time	System Recovery time	Total time to recover from fault
60 procs	1399.1	8.0	9.8	24.8	1441.7
120 procs	1429.3	9.2	9.9	42.1	1490.5
240 procs	1461.1	9.2	10.0	77.2	1557.5
480 procs	1531.1	9.7	10.1	146.1	1697.0

33

12



## 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

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

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

If there are  $k$  ( $\leq m$ ) 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.

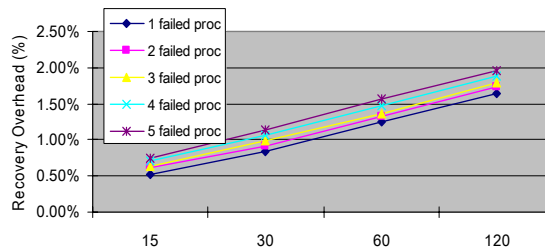
The checkpoint overhead (assume pipelined encoding):

$$T \approx m * \{ (1 + O(1/\text{size\_ckpt}^{0.5})) * \text{size\_ckpt} / \text{bandwidth} + \# \text{of procs} * \text{latency} \}^{13}$$



## PCG: Performance Overhead of Recovery

PCG Performance Overhead for Performing Recovery



Number of Computation Processors  
 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) <sup>14</sup>



## 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

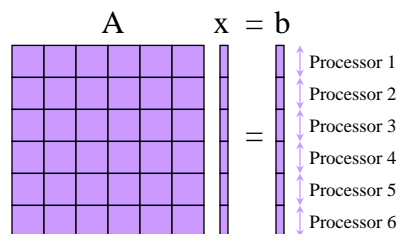
33

15



## Lossy Algorithm : Basic Idea

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



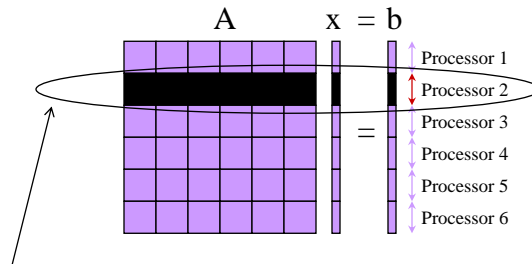
33

16



## 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?

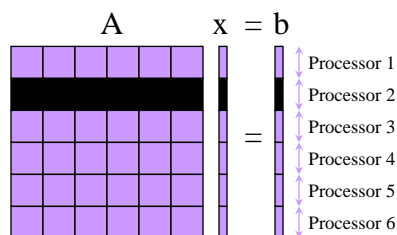
33

17



## 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)

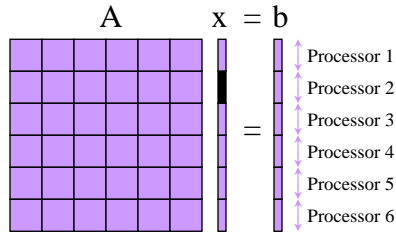
33

18



# 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

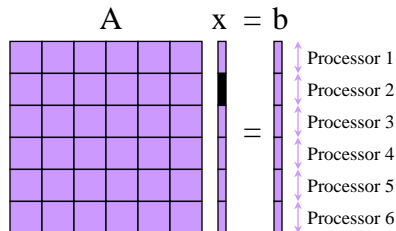
33

19



# 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

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

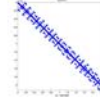
$$x_2 = A_{22}^{-1} (b_2 - \sum_{i \neq 2} A_{2i} x_i)$$

33

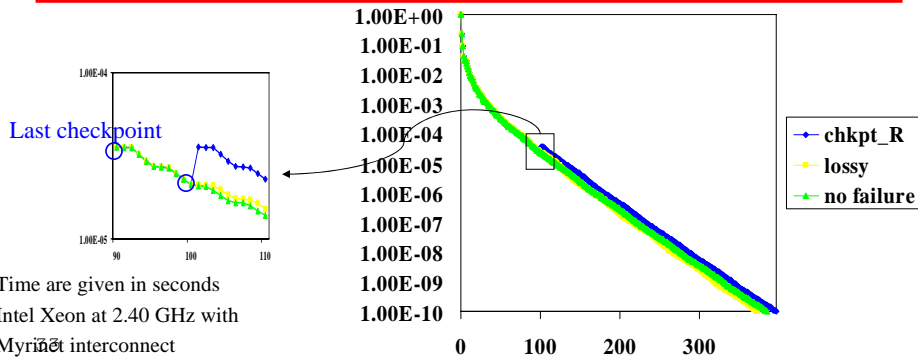
20



## Using GMRES(30) Non Symetric Matrix



stomach; n=213,360; nnz=3,021,648; tol=10 <sup>-10</sup> ; #procs=16; n <sub>f</sub> =13,335; nnz=185,541									
recovery	iter <sub>f</sub>	#iter	T <sub>Wall</sub>	T <sub>Chkpt</sub>	T <sub>Roll</sub>	T <sub>Recov</sub>	T <sub>I</sub>	T <sub>II,a,b</sub>	T <sub>III</sub>
lossy	no	385	38.89						
chkpt <sub>o</sub>	no	385	41.04	1.92					
lossy	100	372	42.38		1.56	5.38	1.03	0.33	3.91
chkpt <sub>R</sub>	100	395	45.49	1.92	2.40	1.68	1.02	0.32	0.20



21



## Third Approach: Matrix-Vector Multiplication with Checksum Matrix

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

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

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

Matrix and vectors stored by rows on processors.

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

<sup>33</sup> 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.

22



## 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

$$M = \begin{pmatrix} M_{11} & \dots & M_{1q} \\ \vdots & \dots & \vdots \\ M_{p1} & \dots & M_{pq} \end{pmatrix}, \text{ where } M_{ij} \text{ is the local matrix on processor } (i, j).$$

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

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

- For  $p \times q$  processors need extra  $p + q + 1$  processors to maintain the checksum.

33

23



## An Example: ScaLAPACK/PBLAS Matrix Multiplication

$$\begin{pmatrix} A_{11} & \dots & A_{1q} \\ \vdots & \dots & \vdots \\ A_{p1} & \dots & A_{pq} \\ \sum_{i=1}^p A_{i1} & \dots & \sum_{i=1}^p A_{iq} \end{pmatrix} * \begin{pmatrix} B_{11} & \dots & B_{1p} & \sum_{j=1}^p B_{1j} \\ \vdots & \dots & \vdots & \vdots \\ B_{q1} & \dots & B_{qp} & \sum_{j=1}^p B_{qj} \end{pmatrix} = \begin{pmatrix} C_{11} & \dots & C_{1p} & \sum_{j=1}^p C_{1j} \\ \vdots & \dots & \vdots & \vdots \\ C_{p1} & \dots & C_{pp} & \sum_{j=1}^p C_{pj} \\ \sum_{i=1}^p C_{i1} & \dots & \sum_{k=1}^p C_{ip} & \sum_{i=1}^p \sum_{j=1}^p C_{ij} \end{pmatrix}$$

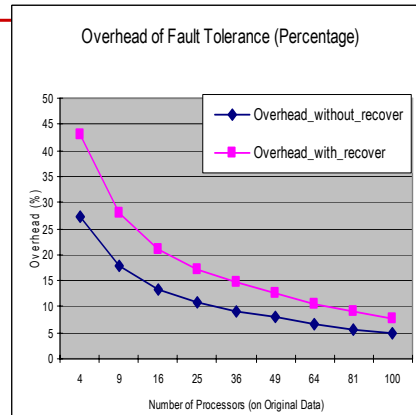
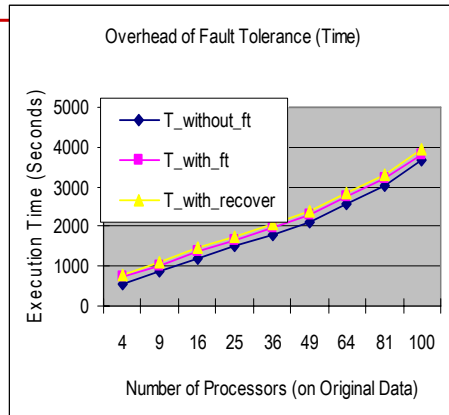
- Single failure during computation can be recovered from the checksum relationship
- By using a floating-point version Reed-Solomon code, multiple failures can be tolerated

33

24



## 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

33    ➤  $O(1/(p*n)) \rightarrow 0, \text{ as } p \rightarrow \infty$

25



## Next Steps

- ◆ Overheads not so great.
- ◆ 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.

33

26



## 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

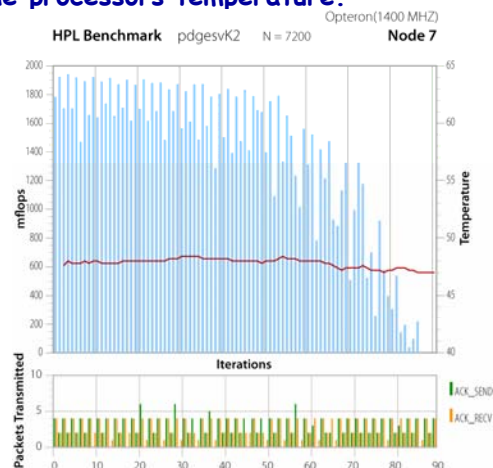
33

27



## 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.



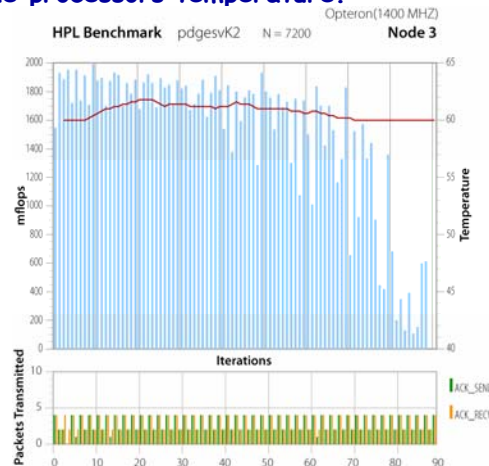
33

28



## 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.



## 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.

33

30



## 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



Web [Bilder](#) [Groups](#) [Verzeichnis](#) [News](#) [Froogle](#) [Desktop](#) [Mehr »](#)  
dongarra [Erweiterte Suche](#)  
  [Einstellungen](#)  
[Sprachtools](#)

Suche:  Das Web  Seiten auf Deutsch  Seiten aus Deutschland

[Werbung](#) - [Unternehmensangebote](#) - [Alles über Google](#) - [Google.com in English](#)

[Machen Sie Google zu Ihrer Startseite!](#)

33

©2006 Google