Self Adapting Numerical Software (SANS-Effort) for Scientific Computing

Jack Dongarra
Innovative Computing Lab
University of Tennessee
and
Computer Science and Math Div
Oak Ridge National Lab
http://www.cs.utk.edu/~dongarra/

2

Challenges in Achieving High Performance on Today's Systems

- **Diversity of execution environments**
  - **Growing complexity of modern microprocessors.**
    - Deep memory hierarchies
    - Out-of-order execution
    - Instruction level parallelism
  - **Growing diversity of platform characteristics**
    - SMPs
    - Clusters (employing a range of interconnect technologies)
    - Highly parallel systems (> 100K processors)
    - Grids (heterogeneity, wide range of characteristics)
- **Wide range of application needs**
  - **Dimensionality and sizes**
  - **Data structures and data types**
  - **Languages and programming paradigms**

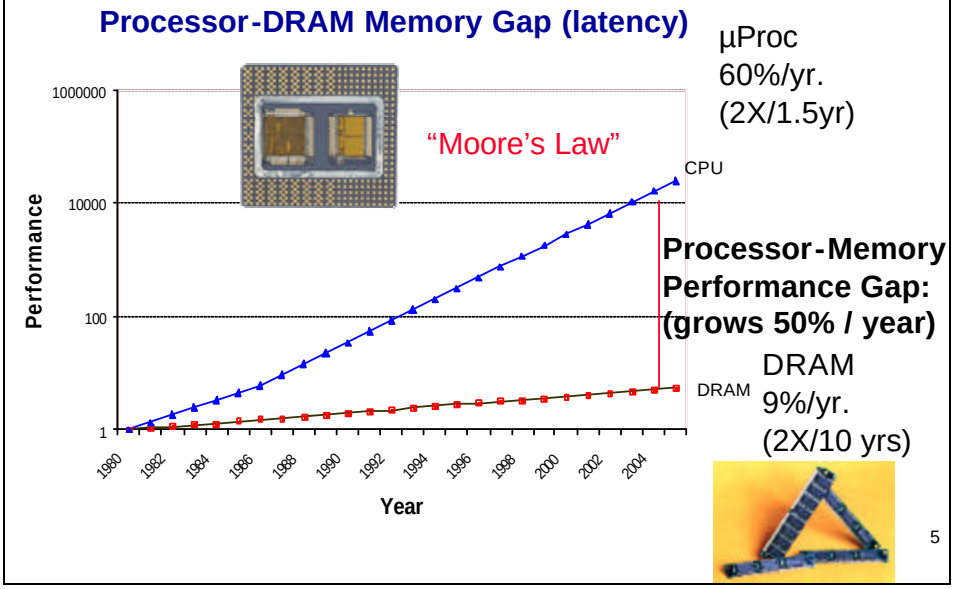
3

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.**
  - **Need for quick/dynamic deployment of optimized routines.**

4

Where Does the Performance Go? or Why Should I Care About the Memory Hierarchy?



5

Optimizing Computation and Memory Use

- **Computational optimizations**
  - **Theoretical peak:(# fpus)*(flops/cycle) * Mhz**
    - **Pentium 4: (1 fpu)*(2 flops/cycle)*(2.8 Ghz) = 5600 MFLOP/s**
- **Operations like:**
  - $a = x^T y$: **2 operands (16 Bytes) needed for 2 flops; at 5600 Mflop/s will requires 5600 MWord/s bandwidth**
- **Memory optimization**
  - **Theoretical peak: (bus width) * (bus speed)**
    - **Pentium 4: (32 bits)*(533 Mhz) = 2132 MB/s = 266 MWord/s**

6

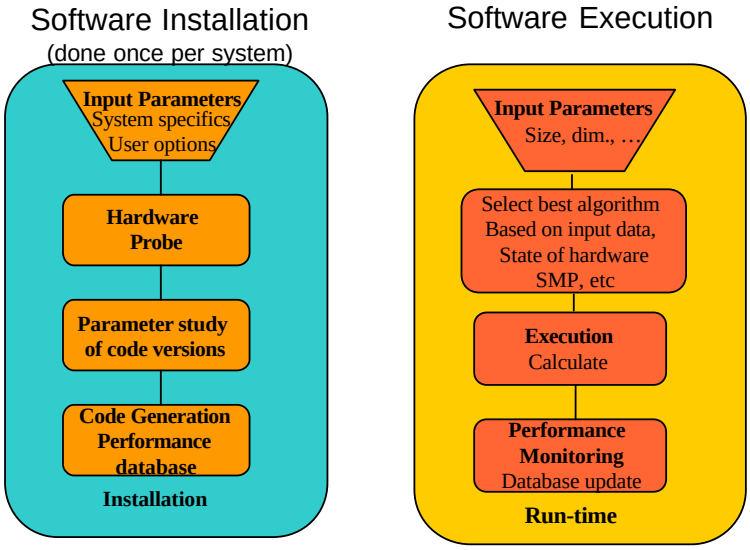
Levels Of Adaptivity

Adaptivity can apply to several levels in a scientific computing environment

- **On the processor-network level: optimization of the kernels for the specific architecture**
  - **Processor: investigate processor hardware characteristics and optimize for them, eg memory hierarchy.**
  - **Network: investigate connectivity, latency, bandwidth, congestion, load**
- **The parallel environment in which the code is run.**
  - **Adaptation to the parallel system or grid**
- **Interfacing to the user code: algorithmic decisions**
  - **Adaptation to user data: investigate user data and make decisions based thereon**

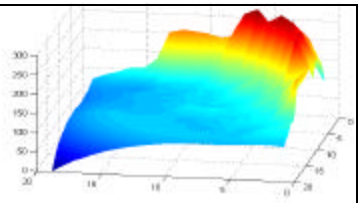
7

Performance Tuning Methodology



8

Software Generation Strategy - ATLAS BLAS

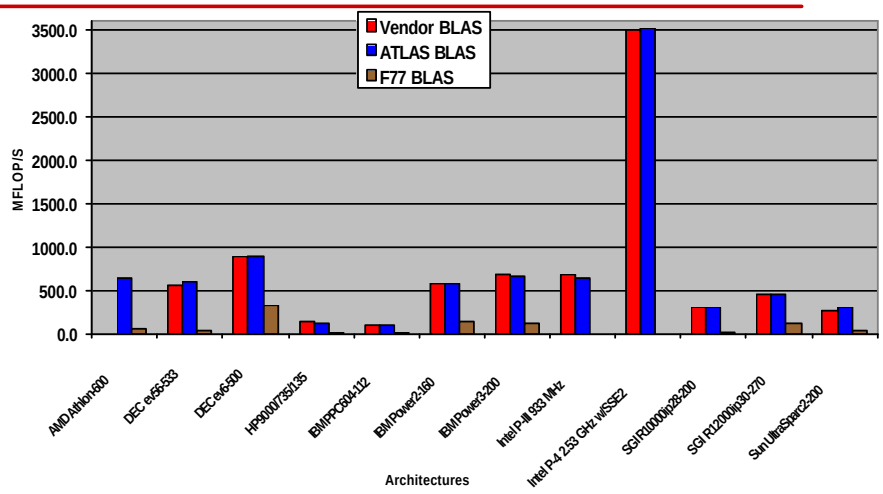


- Parameter study of the hw
- Generate multiple versions of code, w/difference values of key performance parameters
- Run and measure the performance for various versions
- Pick best and generate library
- Level 1 cache multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- 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, …

9

See: http://icl.cs.utk.edu/atlas/ for the ATLAS software

ATLAS (DGEMM n = 500)

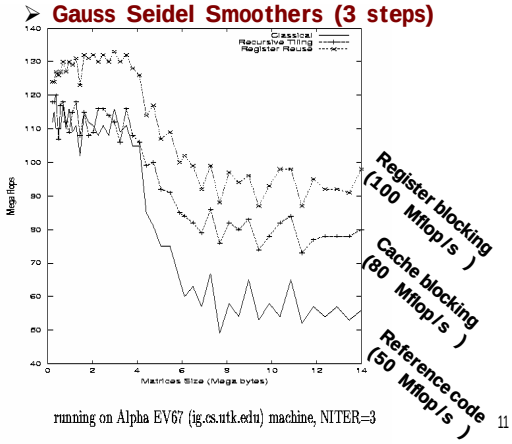


- ATLAS is faster than all other portable BLAS implementations and it is comparable with machine-specific libraries provided by the vendor.
- Looking at sparse operations

10

Efforts To Go Beyond Dense Kernel Operations

- **Kernels**
  - Sparse matrix-vector multiply (SpMV): $y=A*x$
  - Sparse triangular solve (SpTS): $x=T^{-1}*b$
  - $y=AA^T*x$, $y=A^TA*x$
  - matrix triple-product ($R*A*R^T$), Powers ($y=A^k*x$),
- **Optimization techniques (implementation space)**
  - Register blocking
  - Cache blocking
  - Multiple dense vectors (x)
  - A has special structure (*e.g* symmetric, banded, …)
  - Hybrid data structures (*e.g* splitting, switch-to-dense, .
  - Matrix reordering
- **Optimizes for processor characteristics for multigrid software**
  - Integration of smoother and residual, prolongation/restriction
  - Gauss Seidel Smoothers (3 steps)

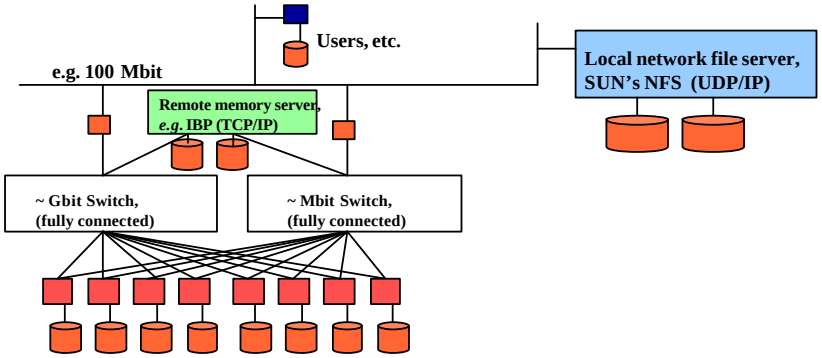


running on Alpha EV67 (ig.cs.utk.edu) machine, NITER=3

11

LAPACK For Clusters

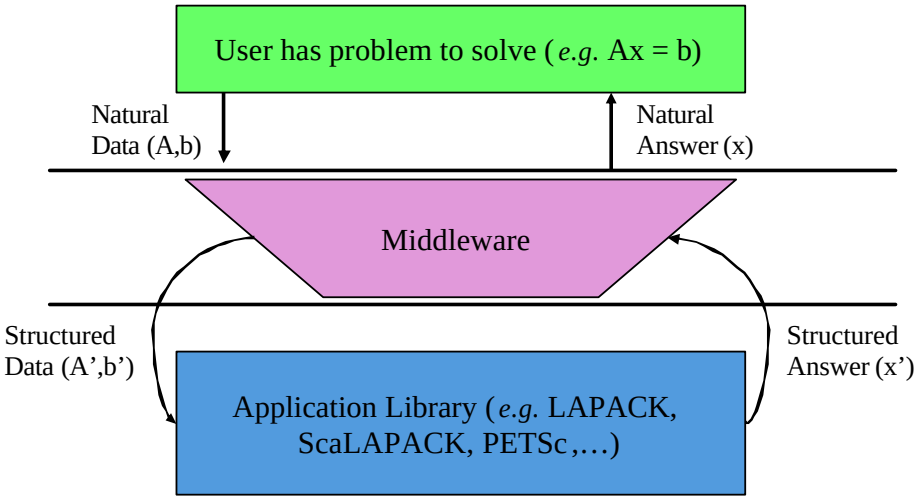
- Developing middleware which couples cluster system information with the specifics of a user problem to launch cluster based applications on the "best" set of resource available.



- Using ScaLAPACK as the prototype software, but developing a framework

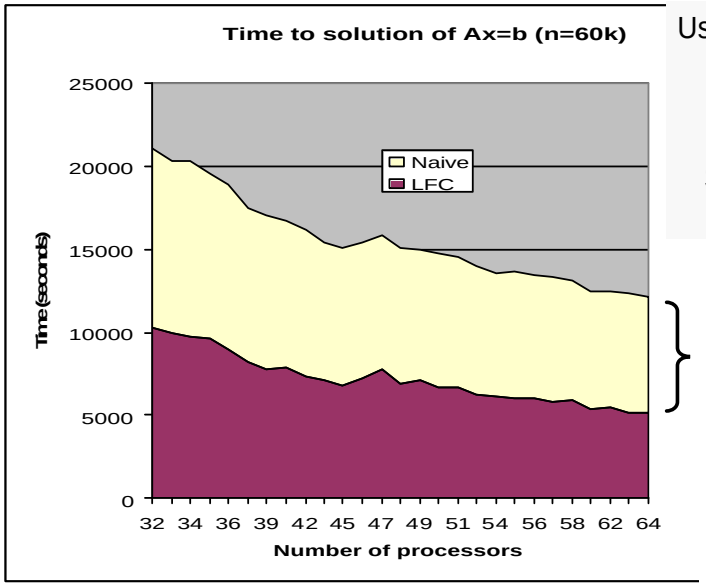
12

User Interface/Middleware



13

LFC Performance Results



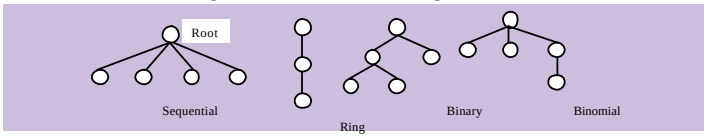
Using up to 64 of AMD 1.4 GHz processors at Ohio Supercomputer Center

Increasing margin of potential user error

4

Self Adapting for Message Passing

- **Communication libraries**
  - **Optimize for the specifics of one's configuration.**
  - **A specific MPI collective communication algorithm implementation** may not **give best results on all platforms.**
  - **Choose collective communication parameters that give best results for the system when the system is assembled.**

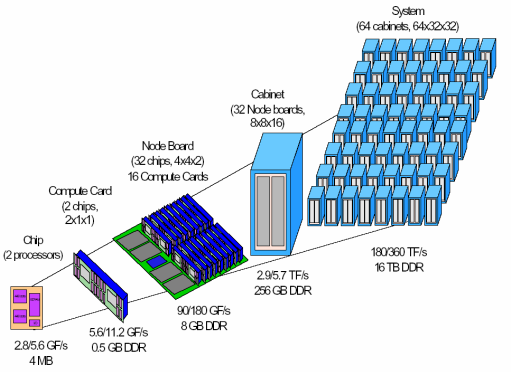


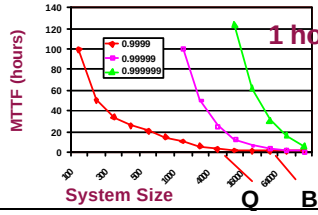
- **Algorithm layout and implementation**
  - **Look at the different ways to express implementation**

15

Fault Tolerance in the Computation

- The next generation of DOE ASCI computers are being designed with 131,000 processors (IBM Blue Gene L)
- Failures for such a system is likely to be just a few minutes away.
- Application checkpoint/restart is today's typical fault tolerance method.
- However, checkpoint & system reboot time approaching MTTF





1 hour reliability

16

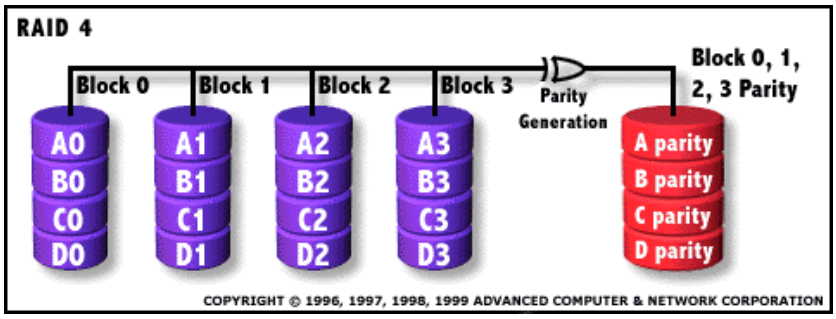
Algorithm Based Fault Tolerance Using Diskless Check Pointing

- Not transparent, has to be built into the algorithm
- N processors will be executing the computation.
  - Each processor maintains their own checkpoint locally
- M (M << N) extra processors maintain coding information so that if 1 or more processors die, they can be replaced
- Look at M = 1 (parity processor)
- FT-MPI based on MPI 1.3 with FT similar to what was done in PVM.

17

How Diskless Check Pointing Works

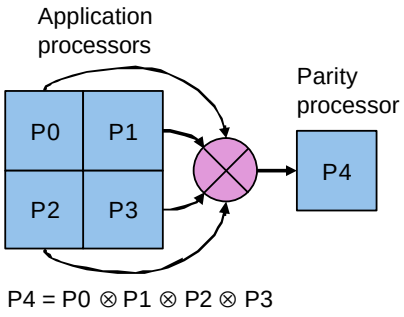
- **Similar to RAID for disks.**



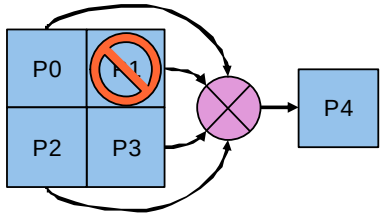
- **If X = A XOR B then this is true:**
  X XOR B = A
  A XOR X = B

Diskless Checkpointing

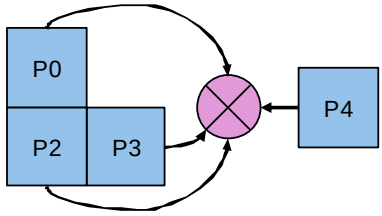
- **The N application processors (4 in this case) each maintain their own checkpoints locally.**
- **M extra processors maintain coding information so that if 1 or more processors die, they can be replaced.**
- **Will describe for m=1 (parity)**
- **If a single processor fails, then its state may be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



Diskless Checkpointing

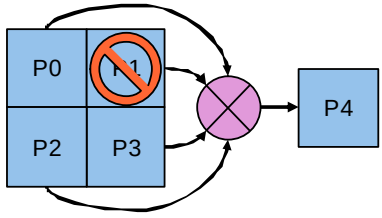


P1 = P0 ⊗ P2 ⊗ P3 ⊗ P4

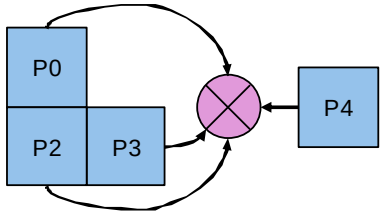


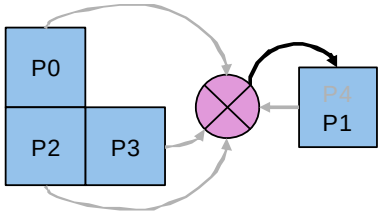
20

Diskless Checkpointing



P4 takes on the identity of P1
and the computation continues





21

Algorithm Based

- Built into the algorithm
  - Not transparent
  - Allows for heterogeneity
- Developing prototype examples for ScaLAPACK and iterative methods for Ax=b

22

PCG Iterative Equation Solver

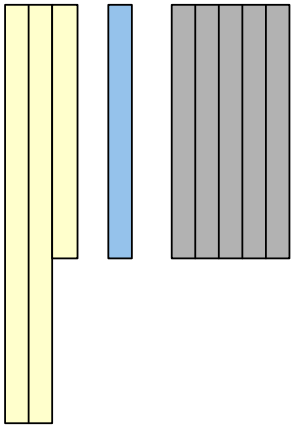
- Given a large, sparse matrix A and a vector B, determine the vector x such Ax=b.
- Chose an initial vector x and iteratively refine it until Ax=b to some error tolerance.
- A is stored as a compressed set of arrays and 5 vectors needed to carryout the iteration.
- Each iteration the 5 vectors are updated using the original matrix A and right hand side b.

23

CG Data Storage

Think of the data like this

A b 5 vectors

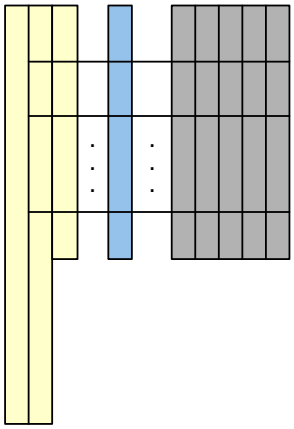


24

Parallel version

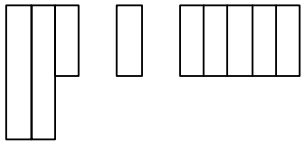
Think of the data like this

A b 5 vectors



Think of the data like this on each processor

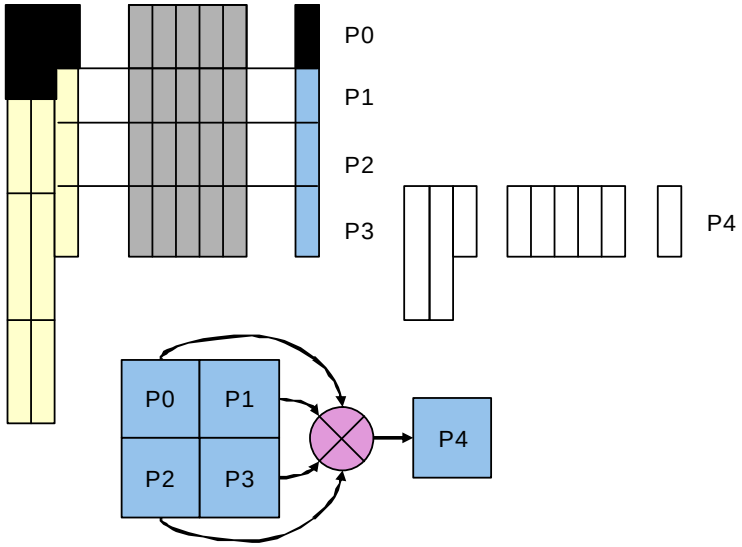
A b 5 vectors



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

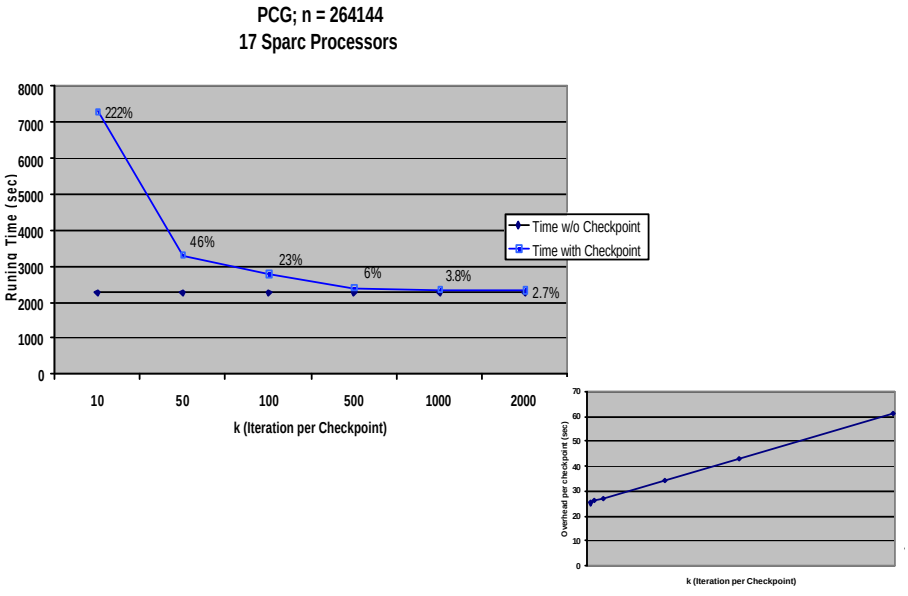
25

Diskless version



26

Diskless Checkpointing Experiments



Diskless Checkpointing

- Diskless checkpointing can be a viable technique for fast frequent checkpointing.
- Converts disk overhead into network overhead
- For numerical libraries, the checkpointing interval can have little effect on total overhead.
- Can apply to other algorithms like matrix decompositions, i.e. LU, QR, Cholesky.
  - Interesting issues about when failure occurs need to roll back computation to checkpoint
  - Undo computation to last checkpoint and then recover

28

Research Directions

- Self Adapting Numerical Software
- Fault tolerant algorithms
- Parameterizable & Annotated libraries
- "Grid" (network) enabled strategies

A new division of labor between compiler writers, library writers, and algorithm developers and application developers will emerge.

29

Collaborators / Support

- **ATLAS**
  - Clint Whaley, FSU
  - Antoine Petitet, Sun
- **LFC**
  - Kenny Roche, UTK
  - Piotr Luszczek, UTK
  - Jeffery Chen, UTK
- **SALSA/BeBOP**
  - Victor Eijkhout, UTK
  - David Keyes, CU
  - Bill Gropp, ANL
  - Jim Demmel, UCB
  - Kathy Yelick, UCB
- **Diskless Checkpointing**
  - Jim Plank, UTK

- **Thanks**

ALLIANCE

NPACI

NSF
Next Generation Software (NGS)

SciDAC
Scientific Discovery through Advanced Computing

TOPS
Terascale Optimal PDE Simulations

30