

Libraries, Tools, and PSEs for Petaflops Systems

Jack Dongarra

University of Tennessee
and
Oak Ridge National Laboratory

1

Future Architectures

- ◆ **Petaflops Architectures**
 - Faster processors, high processor counts
 - Deep memory hierarchy
 - High levels of parallelism required

2

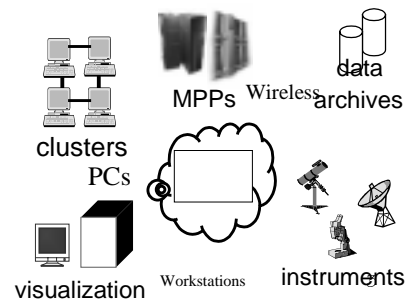
Future Architectures

◆ Petaflops Architectures

- Faster processors, high processor counts
- Deep memory hierarchy
- High levels of parallelism required

◆ Grid based computing

- Networks geographically distributed
- Different parallelism and communication
- Nodes and links have varying performance



Convergence of Designs

◆ Petaflops machines look like Grids

- Many different processor types
- Processors connected by networks of different speeds
- Fault tolerance a major problem
- "Grid in a box"
 - » multiple, heterogeneous, processing nodes, interconnected with multiple levels of networks and involving multiple levels of memory hierarchies

Software Technology & Performance

- ◆ Tendency to focus on hardware
- ◆ Software required to bridge an ever widening gap
 - hardware capabilities and user needs
- ◆ Gaps between usable and deliverable performance is very steep
 - Performance only if the data and controls are setup just right
 - » Otherwise, dramatic performance degradations, very unstable situation
 - » Will become more unstable
- ◆ Challenge of Libraries, PSEs and Tools is formidable with Tflop/s level, even greater with Pflops, some might say insurmountable.

5

Software Issues:

- ◆ Predictability and robustness of accuracy and performance.
- ◆ Run-time resource management and dynamic algorithm selection.
- ◆ Support for a multiplicity of programming environments and plugability.
- ◆ Reproducibility, fault tolerant, and auditability of the computations.
- ◆ New algorithmic techniques for latency tolerant and miserly bandwidth applications.
- ◆ Support for long running computations

6

Major Challenge - Adaptivity

- ◆ These characteristics have major implications for applications that require performance guarantees.
- ◆ Adaptivity is a key so applications can function appropriately...
 - as resource utilization and availability change,
 - as processors and networks fail,
 - as old components are retired,
 - as new systems are added, and
 - as both software and hardware on existing systems are updated and modified.

7

Numerical Libraries

- ◆ 20 years ago
 - 1 Mflop/s - Scalar based
 - » Linpack, Level 1 BLAS, loop unrolling
- ◆ 10 years ago
 - 1 Gflop/s - Vector & SMP computing, cache aware
 - » LAPACK, Level 2 & 3 BLAS, block partitioned, latency tolerant
- ◆ Today
 - 1 Tflop/s - Highly parallel, network based, message passing
 - » ScaLAPACK, data decomposition, communication/computation
- ◆ 10 years away
 - 1 Pflop/s - Many more levels MH, combination/grids&HPC
 - » More adaptive, LT and bandwidth aware, fault tolerant, extended precision, attention to SMP nodes

8

Numerical Algorithms and Software

- ◆ Numerical computing will be adaptive, iterative, exploratory, and intelligent.
- ◆ Determinism in numerical computing will be gone.
 - After all, its not reasonable to ask for exactness in numerical computations.
 - Audibility of the computation, reproducibility at a cost
- ◆ Importance of floating point arithmetic will be undiminished.
 - 16, 32, 64, 128 bits and beyond.
 - Standards being developed
- ◆ New methods, multipole methods and their descendants will be ubiquitous.
- ◆ Standards are critical, need to evolve

9

Enhanced ScaLAPACK Eigensolver in ASCI Application (M. Sears, K.Stanley, J. Demmel,...)

- ◆ MP-Quest is a simulation code for materials modeling
 - Surface structures at SNL. (Pb-Zn-Ti)
- ◆ MP-Quest will account for 10-20% of all cycles on ASCI-Red
 - Symmetric Eigensolver accounts for 25% of cycles in MP-Quest
 - Hence 2.5% of Ascii Red cycles in MP-Quest's symmetric eigensolver.
- ◆ ScaLAPACK's Sym Eig solver is 10x faster than MP-Quest's
 - 90% savings in cycles by using ScaLAPACK
- ◆ Right now large problems are too slow because of the eigensolver.
- ◆ Sustain 605 Gflop/s (ScaLAPACK part at 684 Gflop/s)
- ◆ Extrapolation to Petaflops machine:
 - 20-30% of peak on a one hour problem on a petaflop machine.
 - 50% of peak for a $10^6 \times 10^6$ problem taking 11 hours.

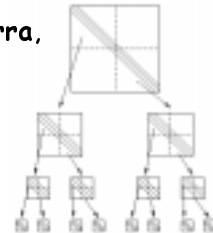
10

New Algorithms/Software

◆ Symmetric λ -problem

- **Divide and Conquer** (J. Cuppen, J. Dongarra, D. Sorensen, L. Jessup, F. Tisseur, ...)

- » recursively split problem
- » generates lots of parallel work



- **Holy Grail** (J. Demmel, B. Parlett, I. Dhillon, ...)

- » Optimal output complexity $O(n^2)$ and embarrassingly parallel
- » Works by redefining clusters of eigenvalues to get relative gap instead of absolute gap.

◆ Devil's in the details in terms of implementation

11

Changing Algorithmic Approach

- ◆ **Compute by the fast, possibly unreliable, method - "living dangerously"**
 - Check a posteriori whether a problem occurred
 - Fix any problems by the reliable, perhaps much slower method
- ◆ **Predictability and robustness of accuracy and performance**
 - May require higher precision (128 bit floating point)
 - Perhaps interval arithmetic may play a role.
 - » However problems with composibility
- ◆ **Reproducibility, fault tolerance, and auditability**

12

Performance Issues - Cache & Bandwidth

- ◆ Performance instability
 - Small changes may cause dramatic changes in delivered performance.
- ◆ Latency tolerant and bandwidth parsimonious algorithms and software are critical
 - Recompute rather than store/load
- ◆ Need to help the compiler
- ◆ Have a hard time today getting performance
 - Only going to get harder

13

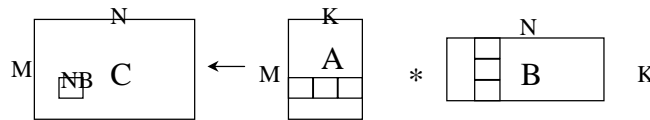
How To Get Performance From Commodity Processors?

- ◆ Today's processors can achieve high-performance, but this requires extensive machine-specific hand tuning.
- ◆ Routines have a large design space w/many parameters
 - Blocking sizes, loop nesting permutations, loop unrolling depths, software pipelining strategies, register allocations, and instruction schedules.
 - Complicated interactions with the increasingly sophisticated microarchitectures of new microprocessors.
 - Very unstable
- ◆ Not long ago no tuned BLAS for Pentium for Linux.
- ◆ Need for quick/dynamic deployment of optimized routines.
- ◆ ATLAS - Automatic Tuned Linear Algebra Software
 - Adapts to differing architectures via code generation + timing
 - PhiPac from Berkeley
 - FFTW from MIT

14

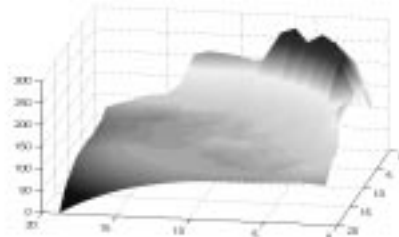
Adaptive Approach for Level 3 BLAS

- ◆ Do a parameter study of the operation on the target machine, done once.
- ◆ Only generated code is on-chip multiply
- ◆ BLAS operation written in terms of generated on-chip multiply
- ◆ All transpose cases coerced through data copy to 1 case of on-chip multiply
 - Only 1 case generated per platform

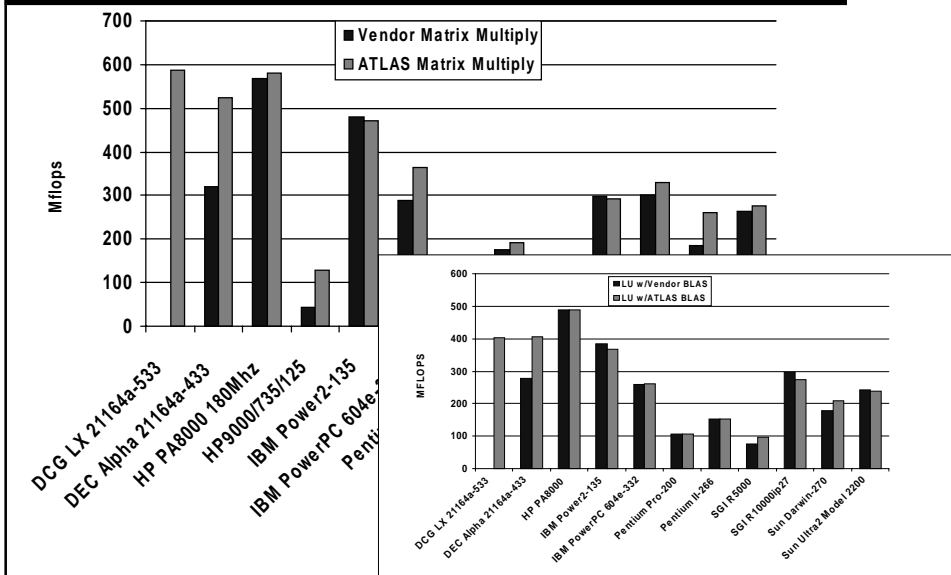


Code Generation Strategy

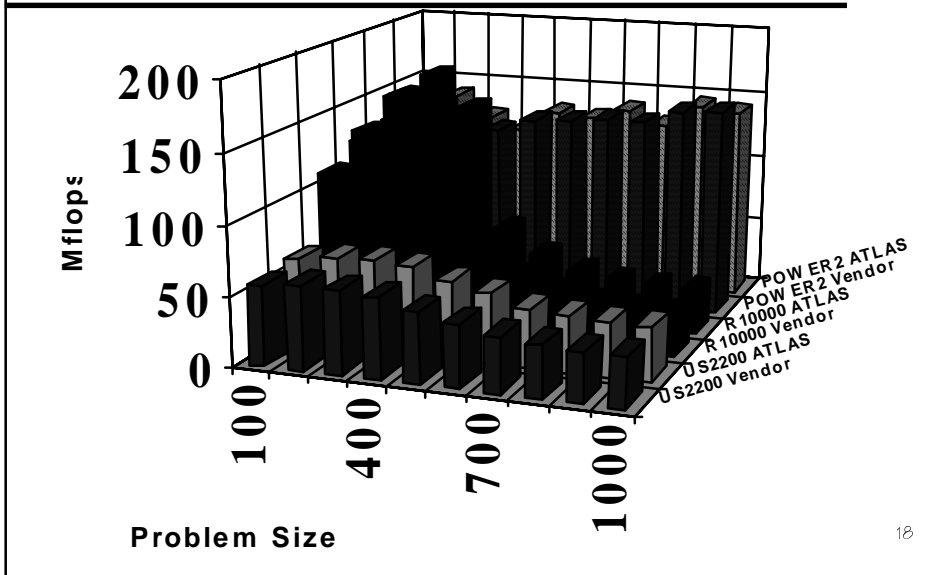
- ◆ On-chip multiply optimizes for:
 - TLB access
 - L1 cache reuse
 - FP unit usage
 - Memory fetch
 - Register reuse
 - Loop overhead minimization
- ◆ Takes a couple of hours to run.
- ◆ Code is iteratively generated & timed until optimal case is found. We try:
 - Differing NBs
 - Breaking false dependencies
 - M, N and K loop unrolling



ATLAS Across Various Architectures 500x500 Matrix Problem



Left looking LU (GEMV based) across platforms



Future work

- ◆ **Threading**
- ◆ **Runtime adaptation**
 - Sparsity analysis
 - Iterative code improvement
- ◆ **Adaptive libraries**
- ◆ **Specialization for user applications**
- ◆ **See: <http://www.netlib.org/atlas/>**
 - Official release next week

19

Fault Tolerance - Diskless (RAID) Checkpointing - Built into software

(J. Plank, J. Dongarra)

- ◆ **Maintain a system checkpoint in memory**
 - All processors may be roll back if necessary
 - Use m extra processors to encode checkpoints so that if up to m processors fail, their checkpoints may be restored
 - No reliance on disk
- ◆ **Checksum and reverse communication**
 - Checkpoint less frequently
 - Reverse the computation of the non-failed processors back to previous checkpoint
- ◆ **Idea to build into library routines**
 - System or user can dial it up
 - Working prototype for MM, LU, LL^T , QR, sparse solvers (built on PVM)

20

Research Directions

- ◆ Parameterizable libraries
- ◆ Annotated libraries
- ◆ Hierarchical algorithm libraries
- ◆ "Grid" (network) enabled strategies

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

21

Many Active Tools Projects

- ◆ Akenti
- ◆ AppLeS
- ◆ Arcade
- ◆ CIF
- ◆ Condor
- ◆ CUMULVUS
- ◆ EveryWare
- ◆ Globus
- ◆ Habanero
- ◆ Harness
- ◆ IceT
- ◆ IPG NAS-NASA
- ◆ JINI
- ◆ Llava
- ◆ Legion
- ◆ NCSA Workbench Project
- ◆ NEOS
- ◆ NetSolve
- ◆ NINF
- ◆ Ninja
- ◆ PAWS
- ◆ PARDIS
- ◆ POEMS
- ◆ Sweb
- ◆ Teraweb
- ◆ UNICORE
- ◆ WebFlow

22

Tools/Debuggers

- ◆ TotalView is the best parallel debugger available.
 - limitation is its scalability
 - only used in a homogeneous environment
- ◆ Vampir performance analysis tool for parallel MPI programs
 - scalability is a problem, tracefile can be huge.
 - hard to see what's going on when you have lots of processes
- ◆ Research prototype tools Paradyne, Virtue, and DPCL
 - look promising
 - yet to be proven
 - not yet robust finished enough to be truly useful at this stage for HPC end users.
- ◆ Computational Steering
 - Autopilot, Cumulvs, SciRun
- ◆ Performance counter which will provide a portable interface to a common set of performance metrics across platforms
- ◆ Survey: <http://www.nhse.org/~browne/perf-tools-review/>

23

Challenges

- ◆ Dynamic reconfiguration
- ◆ optimization for distributed targets
- ◆ performance monitoring and control strategies
 - needs deep integration across compilers, tools, and runtime systems
 - performance contracts and dynamic reconfiguration

24

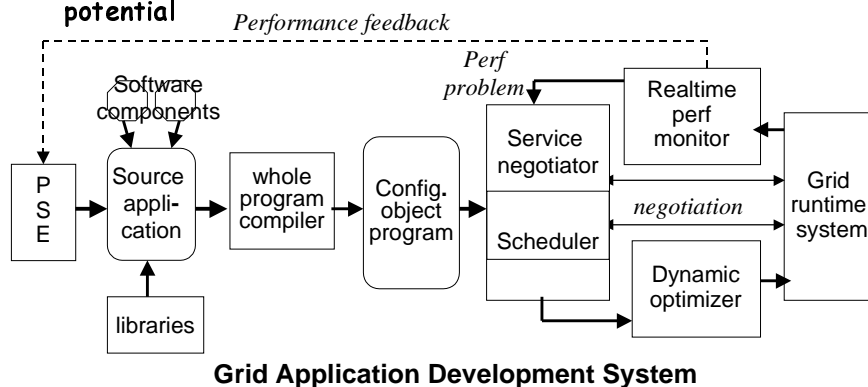
Challenges for LTP Research

- ◆ **Complex Architectures**
 - more parallelism
 - deeper memory hierarchies
 - heterogenous system
 - » late binding
- ◆ **Complex Applications**
 - multiple (programs, languages, parallelism styles)
 - irregular, adaptive, dynamic computations
 - Script based system
- ◆ **Issues**
 - performance bottlenecks
 - portable high performance and predictable computing

25

The Brave New World

- ◆ "Grid-aware" programming will require comprehensive development and execution environment
 - Performance economy required to leverage system, application potential



Conclusions

- ◆ **Need for software capitalization**
 - take the research products into useable, harden products.
- ◆ **Libraries**
 - Tools on top of this
 - Component repositories

27

Contributors to These Ideas

- ◆ Fran Berman
 - ◆ Henri Casanova
 - ◆ Frederica Darema
 - ◆ Jim Demmel
 - ◆ Ian Foster
 - ◆ Dennis Gannon
 - ◆ Al Geist
 - ◆ Andrew Grimshaw
 - ◆ Lennart Johnsson
 - ◆ Ken Kennedy
 - ◆ Carl Kesselman
 - ◆ Dan Reed
 - ◆ Subhash Saini
 - ◆ Clint Whaley
- ◆ **For additional information see...**
 - icl.cs.utk.edu/
 - www.netlib.org/atlas/
 - www.cs.utk.edu/netsolve/
 - www.netlib.org/utk/people/JackDongarra/

28