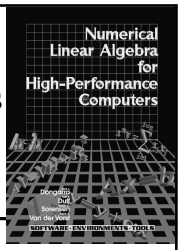
EuroPar'99 Tutorial on Linear Algebra Algorithms and Software for Large Scientific Problems



Jack Dongarra, U of Tennessee and ORNL
Iain Duff, Rutherford-Appleton Lab
Danny Sorensen, Rice University
Henk van der Vorst, Utrecht University

1

Outline

- Overview of HPC and Dense Problems 9:30-10:45
  - Jack Dongarra
- Sparse Direct Methods for $Ax = b$ 11:15-12:30
  - Iain Duff
- Sparse Iterative Methods for $Ax = b$ 2:30-3:45
  - Henk van der Vorst
- Sparse Iterative Methods for $Ax = \lambda x$ 4:15-5:30
  - Danny Sorensen

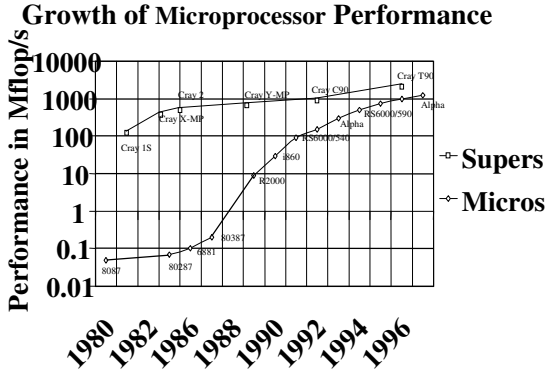
2

High-Performance Computing Today

- In the past decade, the world has experienced one of the most exciting periods in computer development.
- Microprocessors have become smaller, denser, and more powerful.
- The result is that microprocessor-based supercomputing is rapidly becoming the technology of preference in attacking some of the most important problems of science and engineering.

3

Growth of Microprocessor Performance



Scalable Multiprocessors

- Memory access from local memory must be adequate.
- Memory accesses across the machine requires a scalable global interprocessor communication network.
- Scaling memory bandwidth cost-effectively requires separate, distributed memories.
- Cost-effectiveness also requires best price-performance in individual processors.
- What we get:
  - Compelling Price/Performance
  - Tremendous scalability
  - Tolerable entry price
  - Tackle intractable problems

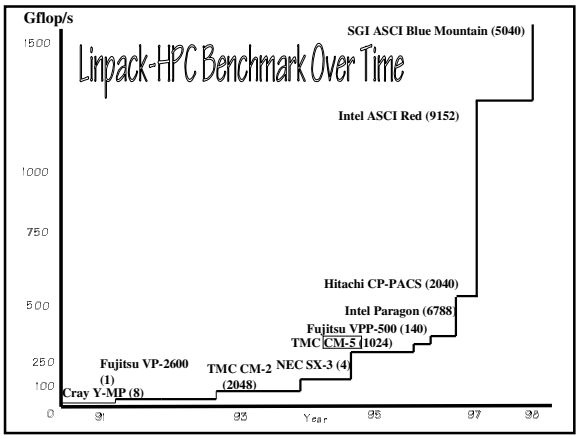
5

Performance Numbers on RISC Processors

- Using Linpack Benchmark

Machine	MHz	Linpack n=100 Mflop/s	Ax=b n=1000 Mflop/s	Peak Mflop/s
DEC Alpha	**612**	287(23%)	**764 (59%)**	1224
IBM RS/397	**160**	315 (49%)	**532 (83%)**	640
HP PA	**440**	375 (21%)	**1290 (73%)**	1760
SGI Origin 2K	**300**	173 (29%)	**553 (92%)**	600
SUN Ultra	**400**	183 (23%)	**552 (69%)**	800
Intel Xeon	**450**	98 (22%)	**295 (66%)**	450
Cray T90	**454**	705 (39%)	**1603 (89%)**	1800
Cray C90	**238**	387 (41%)	**902 (95%)**	952
Cray Y-MP	**166**	161 (48%)	**324 (97%)**	333
Cray X-MP	**118**	121 (51%)	**218 (93%)**	235
Cray J-90	**100**	106 (53%)	**190 (95%)**	200
Cray 1	**80**	27 (17%)	**110 (69%)**	160

6



The Maturation of Highly Parallel Technology

- Affordable parallel systems now out-perform the best conventional supercomputers.
- Performance per dollar is particularly favorable.
- The number of companies has shrunk and we have a few very capable systems.
- Reliability of systems is greatly improved.
- Third-party scientific and engineering applications are appearing.
- Business applications are appearing.
- Commercial customers, not just research labs, are acquiring systems.

Directions

- Move toward shared memory
  - SMPs and Distributed Shared Memory
  - Shared address space w/ deep memory hierarchy
- Clustering of shared memory machines for scalability
- Efficiency of message passing and data parallel programming
  - Helped by standards efforts such as MPI and HPF (PVM & OpenMP)

Challenges in Developing Distributed Memory Libaries

- How to integrate software?
  - Until recently no standards
  - Many parallel languages
  - Various parallel programming models
  - Assumptions about the parallel environment
    - » granularity
    - » topology
    - » overlapping of communication/computation
    - » development tools
- Where is the data
  - Who owns it?
  - Opt data distribution
- Who determines data layout
  - Determined by user?
  - Determined by library developer?
  - Allow dynamic datadist.
  - Load balancing

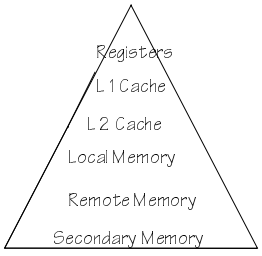
10

EISPACK and LINPACK

- EISPACK
  - Design for the algebraic eigenvalue problem, $Ax = \lambda x$ and $Ax = \lambda Bx$.
  - work of J. Wilkinson and colleaques in the 70's.
  - Fortran 77 software based on translation of ALGOL.
- LINPACK
  - Design for the solving systems of equations, $Ax = b$.
  - Fortran 77 software using the Level 1 BLAS.

11

Memory Hierarchy

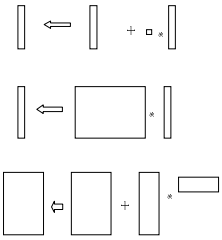


- Key to high performance in effective use of memory hierarchy
- True on all architectures

12

Level 1, 2 and 3 BLAS

- Level 1 BLAS
  Vector-Vector operations
- Level 2 BLAS
  Matrix-Vector operations
- Level 3 BLAS
  Matrix-Matrix operations

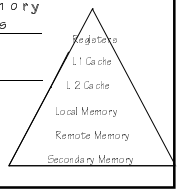


13

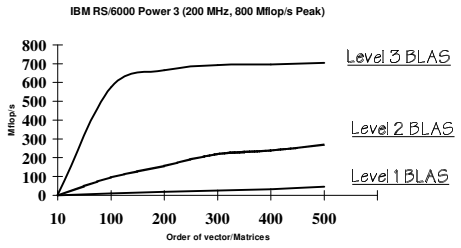
Why Higher Level BLAS?

- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

BLAS	Memory Refs	Flops	Flops/ Memory Refs
Level 1 $y = y + \alpha x$	$3n$	$2n$	2/3
Level 2 $y = y + Ax$	n^2	$2n^2$	2
Level 3 $C = C + AB$	$4n^2$	$2n^3$	$n/2$



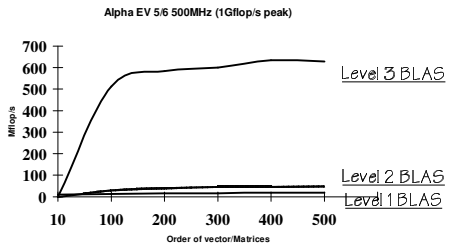
BLAS for Performance



- Development of blocked algorithms important for performance

15

BLAS for Performance



- Development of blocked algorithms important for performance

16

History of Block Partitioned Algorithms

- Early algorithms involved use of small main memory using tapes as secondary storage.
- Recent work centers on use of vector registers, level 1 and 2 cache, main memory, and "out of core" memory.

17

Blocked Partitioned Algorithms

- LU Factorization
- Cholesky factorization
- Symmetric indefinite factorization
- Matrix inversion
- QR, QL, RQ, LQ factorizations
- Form Q or Q^TC
- Orthogonal reduction to:
  - (upper) Hessenberg form
  - symmetric tridiagonal form
  - bidiagonal form
- Block QR iteration for nonsymmetric eigenvalue problems

18

LAPACK

- Linear Algebra library in Fortran 77
  - Solution of systems of equations
  - Solution of eigenvalue problems
- Combine algorithms from LINPACK and EISPACK into a single package
- Efficient on a wide range of computers
  - RISC, Vector, SMPs
- User interface similar to LINPACK
  - Single, Double, Complex, Double Complex
- Built on the Level 1, 2, and 3 BLAS

19

LAPACK

- Most of the parallelism in the BLAS.
- Advantages of using the BLAS for parallelism:
  - Clarity
  - Modularity
  - Performance
  - Portability

20

Derivation of Blocked Algorithms Cholesky Factorization $A = U^T U$

$$\begin{pmatrix} A_{11} & a_j & A_{13} \\ a_j^T & a_{jj} & \alpha_j \\ A_{13}^T & \alpha_j & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ u_j^T & u_{jj} & 0 \\ U_{13}^T & u_{jj} & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & u_j & U_{13} \\ 0 & u_{jj} & \mu_j^T \\ 0 & 0 & U_{33} \end{pmatrix}$$

Equating coefficient of the j^{th} column, we obtain

$$a_j = U_{11}^T u_j$$

$$a_{jj} = u_j^T u_j + u_{jj}^2$$

Hence, if U_{11} has already been computed, we can compute u_j and u_{jj} from the equations:

$$U_{11}^T u_j = a_j$$

$$u_{jj}^2 = a_{jj} - u_j^T u_j$$

21

LINPACK Implementation

- Here is the body of the LINPACK routine SPOFA which implements the method

```
      DO 30 J = 1, N
         INFO = J
         S = 0.0E0
         JM1 = J - 1
         IF( JM1.LT.1 ) GO TO 20
         DO 10 K = 1, JM1
            T = A( K, J ) - SDOT( K-1, A( 1, K ), 1,A( 1, J ), 1 )
            T = T / A( K, K )
            A( K, J ) = T
            S = S + T*T
   10    CONTINUE
   20    CONTINUE
         S = A( J, J ) - S
C     ...EXIT
         IF( S.LE.0.0E0 ) GO TO 40
         A( J, J ) = SQRT( S )
   30 CONTINUE
```


LAPACK Implementation

```
      DO 10 J = 1, N
         CALL STRSV( 'Upper', 'Transpose', 'Non-Unit', J-1, A, LDA, A( 1, J ), 1 )
         S = A( J, J ) - SDOT( J-1, A( 1, J ), 1, A( 1, J ), 1 )
         IF( S.LE.ZERO ) GO TO 20
         A( J, J ) = SQRT( S )
   10 CONTINUE
```

- This change by itself is sufficient to significantly improve the performance on a number of machines.
- From 72 to 251 Mflop/s for a matrix of order 500 on one processor of a CRAY Y-MP.
- However on 378 Mflop/s on 8 Procs. Of a CRAY Y-MP.
- Suggest further work needed.

Derivation of Blocked Algorithms

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{12}^T & A_{22} & A_{12} \\ A_{13}^T & A_{12}^T & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ U_{12}^T & U_{22}^T & 0 \\ U_{13}^T & U_{23}^T & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}$$

Equating coefficient of second block of columns, we obtain

$$A_{12} = U_{11}^T U_{12}$$

$$A_{22} = U_{12}^T U_{12} + U_{22}^T U_{22}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution of the following equations by a call to the Level 3 BLAS routine STRSM:

$$U_{11}^T U_{12} = A_{12}$$

LAPACK Blocked Algorithms

```
      DO 10 J = 1, N, NB
         CALL STRSM( 'Left', 'Upper', 'Transpose','Non-Unit', J-1, JB, ONE, A, LDA,
     $               A( 1, J ), LDA )
         CALL SSYRK( 'Upper', 'Transpose', JB, J-1,-ONE, A( 1, J ), LDA, ONE,
     $               A( J, J ), LDA )
         CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
         IF( INFO.NE.0 ) GO TO 20
   10 CONTINUE
```

- On Y-MP, L3 BLAS squeezes a little more out of 1proc, but makes a large improvement when using 8procs.

Cray Y-MP Cholesky, n=500	1 Proc	8 procs
j-variant LINPACK	72	72
j-variant Level 2 BLAS	251	378
j-variant Level 3 BLAS	287	1225
k-variant Level 3 BLAS	290	1414

25

LAPACK Contents

- Combines algorithms from LINPACK and EISPACK into a single package. User interface similar to LINPACK.
- Built on the L1, 2 and 3 BLAS, for high performance (manufacturers optimize BLAS)
- LAPACK does not provide routines for structured problems or general sparse matrices (i.e sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).

26

LAPACK Ongoing Work

- Add functionality
  - updating/downdating divide and conquer least squares,bidiagonal bisection, bidiagonal inverse iteration, band SVD, Jacobi methods, ...
- Move to new generation of high performance machines
  - IBM SPs, CRAY T3E, SGI Origin, clusters of workstations
- New challenges
  - New languages: FORTRAN 90, HP FORTRAN, ...
  - (CMMD, MPL, NX ...)
    - many flavors of message passing, need standard (PVM, MPI); BLACS
- Highly varying ratio ~~Computational speed~~
- Many ways to layout data,
- Fastest parallel algorithm sometimes less stable numerically.

27

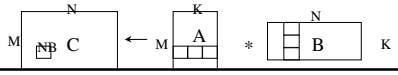
How To Get Performance From Commodity Processors?

- 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.
- A few months ago no tuned BLAS for Pentium for Linux.
- Need for quick/dynamic deployment of optimized routines.
- ATLAS - Automatic Tuned Linear Algebra Software
  - PhiPac from Berkeley

28

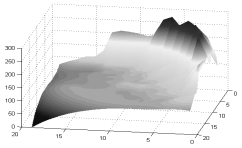
Adaptive Approach for Level 3

- 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 tranpose cases coerced through data copy to 1 case of on-chip multiply
  - Only 1 case generated per platform



Code Generation Strategy

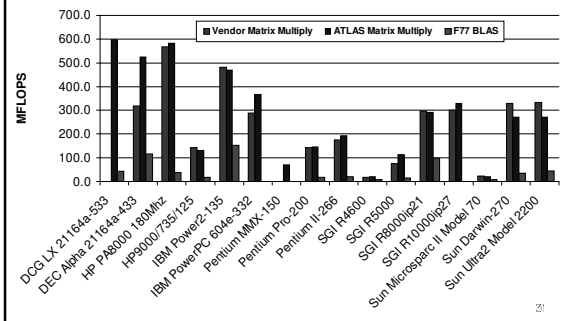
- On-chip multiply optimizes for:
  - TLB access
  - L1 cache reuse
  - FP unit usage
  - Memory fetch
  - Register reuse
  - Loop overhead minimization
- Takes a couple of hours to run.



- Code is iteratively generated & timed until optimal case is found. We try:
  - Differing NBs
  - Breaking false dependencies
  - M, N and K loop unrolling

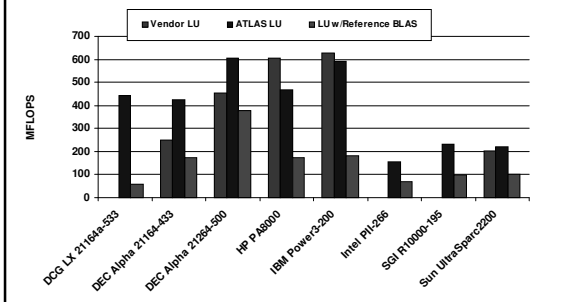
30

ATLAS 500x500 DGEMM Across Various Architectures



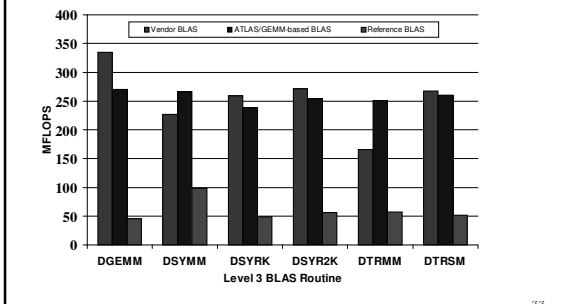
31

500 x 500 LU Right-Looking



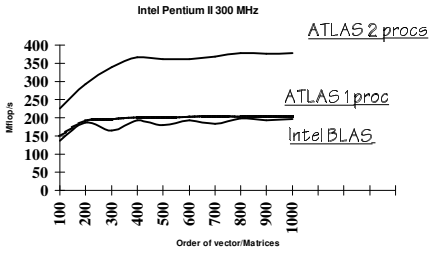
32

500x500 Recursive BLAS on UltraSparc 2200



33

Multithreaded BLAS for Performance



34

ATLAS

- Keep a repository of kernels for specific machines.
- Develop a means of dynamically downloading code
- Extend work to allow sparse matrix operations
- Extend work to include arbitrary code segments
- See: **http://www.netlib.org/atlas/**

35

ScaLAPACK

- Library of software dealing with dense & banded routines
- Distributed Memory - Message Passing
- MIMD Computers and Networks of Workstations
- Clusters of SMPs

36

Programming Style

- SPMD Fortran 77 with object based design
- Built on various modules
  - PBLAS Interprocessor communication
  - BLACS
    - » PVM, MPI, IBM SP, CRI T3, Intel, TMC
    - » Provides right level of notation.
  - BLAS
- LAPACK software expertise/quality
  - Software approach
  - Numerical methods

37

Overall Structure of Software

- Object based - Array descriptor
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out-of-core
- Using the concept of context

38

PBLAS

- Similar to the BLAS in functionality and naming.
- Built on the BLAS and BLACS
- Provide global view of matrix

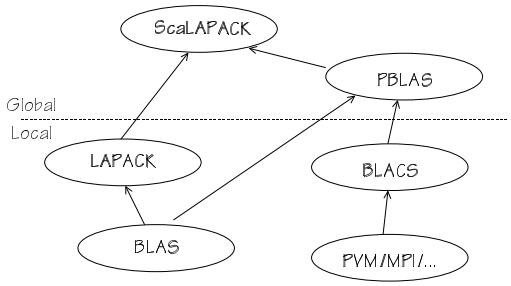
CALL DGEXXX (M, N, A(IA, JA), LDA,...)

⇩

CALL PDGEXXX(M, N, A, IA, JA, DESCA,...)

39

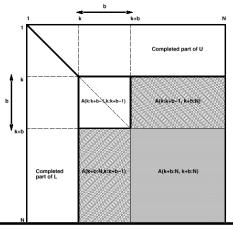
ScaLAPACK Structure



40

Choosing a Data Distribution

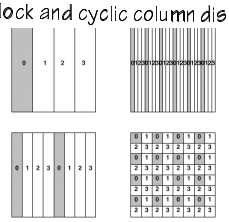
- Main issues are:
  - Load balancing
  - Use of the Level 3 BLAS



41

Possible Data Layouts

- 1D block and cyclic column distributions

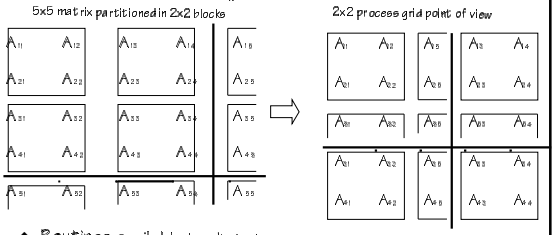


- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

42

Distribution and Storage

- Matrix is block-partitioned & maps blocks
- Distributed 2-D block-cyclic scheme



- Routines available to distribute/redistribute data.

43

Parallelism in ScaLAPACK

- Level 3 BLAS block operations
  - All the reduction routines
- Pipelining
  - QR Algorithm, Triangular Solvers, classic factorizations
- Redundant computations
  - Condition estimators
- Static work assignment
  - Bisection
- Task parallelism
  - Sign function eigenvalue computations
- Divide and Conquer
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function
- Cyclic reduction
  - Reduced system in the band solver
- Data parallelism
  - Sign function

44

ScaLAPACK - What's Included

Problem type Ax = b	SDrv	EDrv	Factor	Solve	Inv	Cond Est	Iter Refin
Triangular				X	X	X	X
SPD	X	X	X	X	X	X	X
SPD Banded	X		X	X			
SPD Tridiagonal	X		X	X			
General	X	X	X	X	X	X	X
General Banded	X		X	X			
General Tridiagonal	X		X	X			
Least squares	X		X	X			
GQR			X				
GRQ			X				

Ax = λx or Ax = λBx	SDrv	EDrv	Reduct	Solution
Symmetric (2 types)	X	X	X	X
General (2 types)			X	X
Generalized BSPD		X	X	X
SVD			X	X

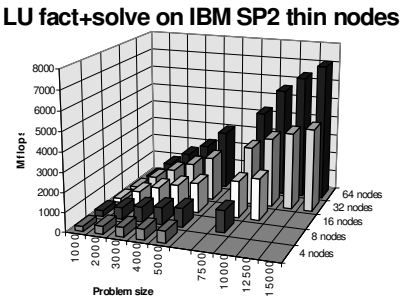
- Timing and Testing routines for almost all
- This is a large component of the package
- Prebuilt libraries available for SP, PGON, HPPA, DEC, Sun, RS6K

Heterogeneous Computing

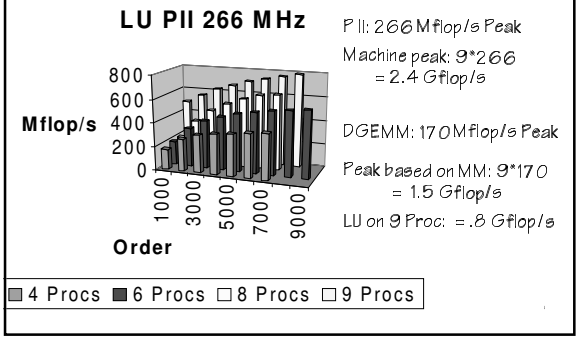
- Software intended to be used in this context
- Communication of ft. pt. numbers between processors
- Machine precision and other machine specific parameters
- Iterative convergence across clusters of processors
- Defensive programming required

46

Performance



ScaLAPACK on a Cluster



PII: 266 Mflop/s Peak

Machine peak: 9*266 = 2.4 Gflop/s

DGEMM: 170 Mflop/s Peak

Peak based on MM: 9*170 = 1.5 Gflop/s

LU on 9 Proc: = .8 Gflop/s

HPF Interface for ScaLAPACK

- Interface provided for a subset of routines
- Linear solvers (general, pos. def., least squares) & matrix multiply
- Given these declarations:

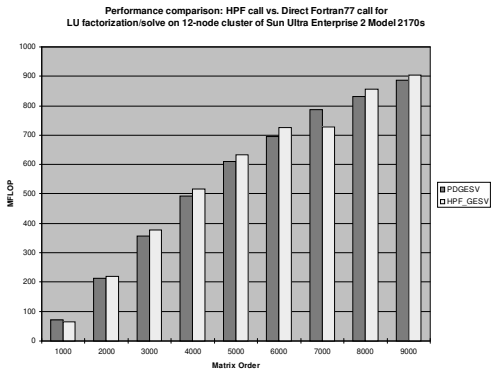
```
USE HPF_LAPACK
REAL, DIMENSION(1000,1000) :: A, B, C
!HPF$ DISTRIBUTE (CYCLIC(64), CYCLIC(64)) :: A, B, C
```

- Calls could be made as simple as:

```
CALL LA_GESV(A, B)
CALL LA_POSV(A, B)
CALL LA_GELS(A, B)
CALL LA_GEMM(A, B, C)
```

(same as F90)

43

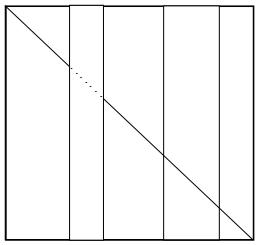


Out of Core Approach

- High-level I/O Interface
- ScaLAPACK uses a 'Right-looking' variant for LU, QR and Cholesky factorizations.
- A 'Left-looking' variant is used for Out-of-core factorization to reduce I/O traffic.
- Requires two in-core column panels.
- Imposes another level in the memory hierarchy.

51

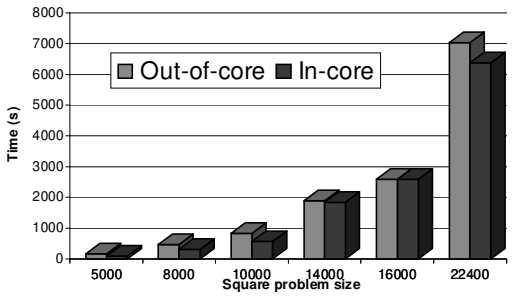
Out of Core Algorithm



- Hybrid approach
- Algorithm is "Left-Looking" in nature, but uses "Right-Looking" (ScaLAPACK) on Panel B
- Latency Tolerant
- Model for deep memory hierarchy

52

QR Factorization on 64 processors Intel Paragon



References

- http://www.netlib.org
- http://www.netlib.org/lapack
- http://www.netlib.org/scalapack
- http://www.netlib.org/lapack/lawns
- http://www.netlib.org/clapack
- http://www.netlib.org/c++/lapack++
- http://www.netlib.org/lapack90
- http://www.nhse.org
- http://www.netlib.org/scalapack/slug/scalapack_slug.html
- http://www.netlib.org/lapack/lug/lapack_lug.htm
- lapack@cs.utk.edu
- scalapack@cs.utk.edu

54