Overview of ScaLAPACK

Jack Dongarra

Computer Science Department
University of Tennessee

and

Mathematical Sciences Section
Oak Ridge National Laboratory

(http://www.netlib.org/utk/people/JackDongarra.html)

The Situation:

Parallel scientific applications are typically

- written from scratch, or manually adapted from sequential programs
- using simple SPMD model
- in Fortran or C
- with explicit message passing primitives

Which means

- similar components are coded over and over again,
- codes are difficult to develop and maintain
- debugging particularly unpleasant...
- difficult to reuse components
- not really designed for long-term software solution

What should math software look like?

- *Many* more possibilities than shared memory
- Possible approaches
  - Minimum change to status quo
    - Message passing and object oriented
  - Reusable templates –
    requires sophisticated user
  - Problem solving environments
    - Integrated systems à la Matlab, Mathematica
      use with heterogeneous platform
  - Computational server, like netlib/xnetlib f2c
- Some users want high perf., access to details
  Others sacrifice speed to hide details
- Not enough penetration of libraries into hardest applications on vector machines
- Need to look at applications and
  rethink mathematical software

The NetSolve Client

Virtual Software Library
Multiplicity of interfaces → Ease of use

Programming interfaces

- C interface
- FORTRAN interface

Interactive interfaces

- Non-graphic interfaces
  - MATLAB interface
  - UNIX-Shell interface
- Graphic interfaces
  - TK/TCL interface
  - HotJava Browser interface (WWW)

The NetSolve Agent

Intelligent Agent → Optimum use of Resource

The agent is permanently aware of :

- The configuration of the NetSolve system
- The characteristics of each NetSolve resource
- The load of each resource
- The functionalities of each resource

Each client requests informs the agent about :

- The type of problem to solve
- The size of the problem

The NetSolve Agent chooses the NetSolve resource :

Load Balancing Strategy

The NetSolve System

The system is a set of computational servers

- Heterogeneity supported (XDR)
- Dynamic addition of resource
- Fault Tolerance

Each server uses the installed scientific packages

Example of packages :

- BLAS
- LAPACK
- ScaLAPACK
- Ellpack

Connection to other systems

- NEOS

LAPACK / ScaLapack
A Scalable Library for Distributed Memory Machines

- LAPACK success (from HP-48G to CRAY C-90)
  - Targets workstations, vector machines, shared memory
  - 449,015 requests since then (as of 10/95)
  - Dense and banded matrices
  - Linear systems, least squares, eigenproblems
  - Manual available from SIAM, html, Japanese translation
  - Second release and manual in 9/94
  - Used by Cray, IBM, Convex, NAG, IMSL, ...
- Provide useful scientific program libraries for computational scientists working on new parallel architectures
  - Move to new generation high performance machines - SP-2, T3D, Paragon, Clusters of WS, ...
  - Add functionality (generalized eigenproblem, sparse matrices, ...)

- New challenges
  - No standard software
    Fortran 90, HP Fortran, PVM, MPIbut ...
  - Many flavors of message passing
    Need standard - BLACS
  - Highly varying ratio of

$$r = \frac{\text{computation speed}}{\text{communication speed}}$$

  - Many ways to layout data
  - Fastest parallel algorithm sometimes less stable numerically

Needs

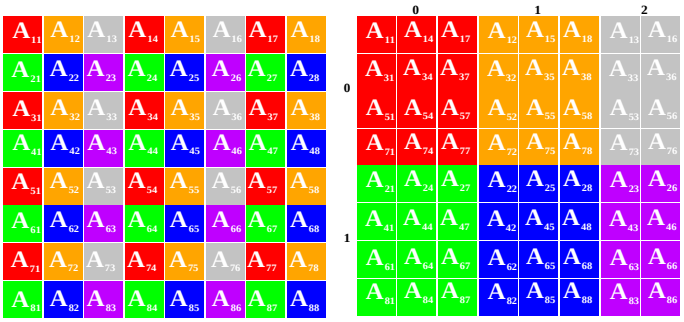
- **Standards**
  - **Portable**
  - **Vendor-supported**
  - **Scalable**
- **High efficiency**
- **Protect programming investment, if possible**
- **Avoid low-level details**
- **Scalability**

ScaLAPACK – OVERVIEW

Goal: Port Linear Algebra PACKage (LAPACK) to distributed-memory environments.

- **Scalability and Performance**
  - Optimized compute and communication engines
  - Blocked data access (level 3 BLAS) yields good node performance
  - 2-D Block-cyclic data decomposition yields good load balance
- **Portability**
  - BLAS: optimized compute engine
  - BLACS: communication kernel (PVM, MPI, ...)
- **Simple calling interface**
  - Stay as close as possible to LAPACK in calling sequence, storage, etc.
- **Ease of Programming / Maintenance**
  - Modularity: Build rich set of linear algebra tools
    - BLAS
    - BLACS
    - PBLAS
  - Whenever possible, use reliable LAPACK algorithms.

2-DIMENSIONAL BLOCK CYCLIC DISTRIBUTION



Global (left) and distributed (right) views of matrix

- **Ensure good load balance ⇒ Performance and scalability**,
- Encompass a large number (but not all) data distribution schemes,
- Need redistribution routines to go from distribution to the other,
- Not intuitive.

Communication Library for Linear Algebra

Basic Linear Algebra Communication Subroutines

Purpose of the BLACS:

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations, improve program readability and the self-documenting quality of code.
- Promote efficiency by identifying frequently-occurring operations of linear algebra which can be optimized on various computers.
- Program portability is improved through standardization of kernels without giving up efficiency since optimized versions will be available.

BLACS – BASICS

- Processes are embedded in a **2-dimensional grid**.

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

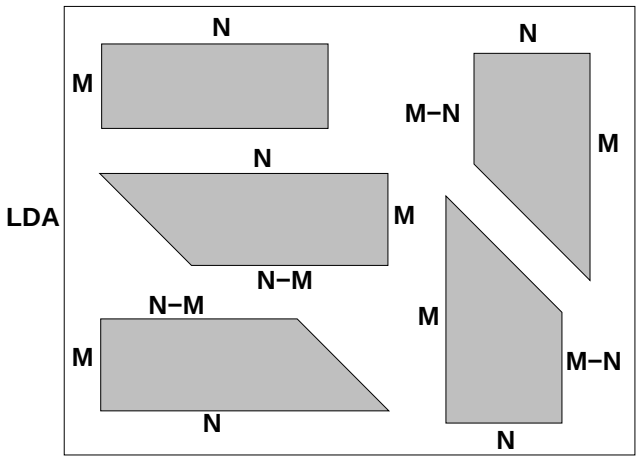
- An operation which involves more than one sender and one receiver is called a **"scoped operation"**. Using a 2D-grid, there are 3 natural scopes:

SCOPE	MEANING
Row	All processes in a process row participate.
Column	All processes in a process column participate.
All	All processes in the process grid participate.

→ use of `SCOPE` parameter.

BLACS – BASICS

- **Operations are matrix based and ID-less**



- **4 Types of BLACS routines: point-to-point communication, broadcast, combine operations and support routines.**

BLACS – COMMUNICATION ROUTINES

1. **Send/Recv**

Send (sub)matrix from one process to another:

□`xxSD2D( ICONTXT, M, N, A, LDA, RDEST, CDEST )`

□`xxRV2D( ICONTXT, M, N, A, LDA, RSRC, CSRC )`

□ (Data type)	xx (Matrix type)
`I`: INTEGER, `S`: REAL, `D`: DOUBLE PRECISION, `C`: COMPLEX, `Z`: DOUBLE COMPLEX.	`GE`: GEneral rectangular matrix, `TR`: TRapezoidal matrix.

```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
   CALL DGESD2D( ICONTXT, 5, 1, X, 5, 1, 0 )
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
   CALL DGERV2D( ICONTXT, 5, 1, Y, 5, 0, 0 )
END IF
```


BLACS – COMMUNICATION ROUTINES

2. **Broadcast**

Send (sub)matrix to all processes or subsection of processes in `SCOPE`, using various distribution patterns (`TOP`):

□`xxBS2D( ICONTXT, SCOPE, TOP, M, N, A, LDA )`

□`xxBR2D( ICONTXT, SCOPE, TOP, M, N, A, LDA, RSRC, CSRC )`

SCOPE	TOP
`'Row'`	`' '` (default)
`'Column'`	'Increasing Ring'
`'All'`	'1-tree' ...

```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 ) THEN
   IF( MYCOL.EQ.0 ) THEN
      CALL DGEBS2D( ICONTXT, 'Row', ' ', 2, 2, A, 3 )
   ELSE
      CALL DGEBR2D( ICONTXT, 'Row', ' ', 2, 2, A, 3, 0, 0 )
   END IF
END IF
```

BLACS – COMBINE OPERATIONS

3. **Global combine operations**

Perform element-wise SUM, |MAX|, |MIN| operations, on rectangular matrices:

```
□GSUM2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,
         RDEST, CDEST )

□GAMX2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,
         RA, CA, RCFLAG, RDEST, CDEST )

□GAMN2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,
         RA, CA, RCFLAG, RDEST, CDEST )
```

- `RDEST = -1` indicates that the result of the operation should be left on all processes selected by `SCOPE`,
- For |MAX|, |MIN|, when `RCFLAG = -1`, `RA` and `CA` are not referenced; otherwise, these arrays are set on output with the coordinates of the process owning the corresponding maximum (resp. minimum) element in absolute value of `A`. Both arrays must be at least of size $M \times N$ and their leading dimension is specified by `RCFLAG`.

BLACS – ADVANCED TOPICS

The **BLACS context** is the BLACS mechanism for partitioning communication space. A defining property of a context is that a message in a context cannot be sent or received in another context. The BLACS context includes the definition of a grid, and each process coordinates in it.

It allows the user to

- create arbitrary groups of processes,
- create multiple overlapping and/or disjoint grids,
- isolate each process grid so that grids do not interfere with each other.

→ the BLACS context is compatible with the MPI communicator

BLACS – REFERENCES

- BLACS software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/blacs`,
  - (anonymous) ftp `ftp.netlib.org`: `cd blacs; get index`
  - email `netlib@www.netlib.org` with the message:

    `send index from blacs`
- Comments and questions can be addressed to

  `blacs@cs.utk.edu`

- J. Dongarra and R. van de Geijn, *Two dimensional Basic Linear Algebra Communication Subprograms*, Technical Report UT CS-91-138, LAPACK Working Note #37, University of Tennessee, 1991.
- R. C. Whaley, *Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures*, Technical Report UT CS-94-234, LAPACK Working Note #73, University of Tennessee, 1994.
- J. Dongarra and R. C. Whaley, *A User's Guide to the BLACS v1.0*, Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.
- Message Passing Interface Forum, *MPI: A Message Passing Interface Standard*, *International Journal of Supercomputer Applications and High Performance Computing*, 8(3–4), 1994.
- A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Cambridge, Massachusetts, 1994.

PBLAS – INTRODUCTION

Parallel Basic Linear Algebra Subprograms for distributed memory MIMD computers.

- **Similar functionality** as the BLAS: distributed vector-vector, matrix-vector and matrix-matrix operations,
- **Simplification of the parallelization** of dense linear algebra codes: especially when implemented on top of the BLAS,
- **Clarity:** code is shorter and easier to read,
- **Modularity:** gives programmer larger building blocks,
- **Program portability:** machine dependency are confined to the PBLAS (BLAS and BLACS).

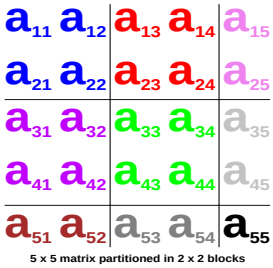
PBLAS – STORAGE CONVENTIONS

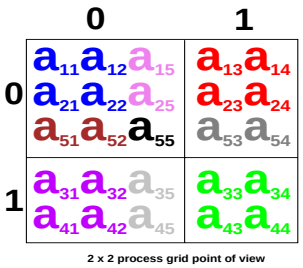
- An M_-by-N_ **matrix is block partitionned** and these MB_-by-NB_ blocks are distributed according to the

  2-dimensional block-cyclic scheme
  $\Rightarrow$ **load balanced computations, scalability**,

- Locally, the scattered **columns are stored contiguously** (FORTRAN "Column-major")

  $\Rightarrow$ **re-use of the BLAS** (leading dimension LLD_).



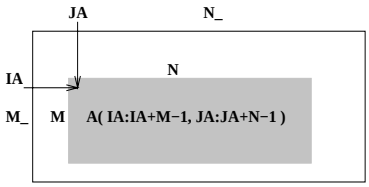


Descriptor DESC_**:** 8-Integer array describing the matrix layout, containing M_, N_, MB_, NB_, RSRC_, CSRC_, CTXT_ and LLD_, where (RSRC_, CSRC_) are the coordinates of the process owning the first matrix entry in the grid specified by CTXT_.

Ex: M_=N_=5, MB_=NB_=5, RSRC_=CSRC_= 0, LLD_ $\geq$ 3 (in process row 0), and 2 (in process row 1).

PBLAS – ARGUMENT CONVENTIONS

- **Global view of the matrix operands**, allowing global addressing of distributed matrices (hiding complex local indexing),



- **Code reusability**, interface very close to sequential BLAS:

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

CALL  DGEMM( 'No Transpose', 'No Transpose',
$     M-J-JB+1, N-J-JB+1, JB, -ONE, A(J+JB,J),
$     LDA, A(J,J+JB), LDA, ONE, A(J+JB,J+JB),
$     LDA )
                          ↓
CALL PDGEXXX( M, N, A, IA, JA, DESCA )

CALL PDGEMM( 'No Transpose', 'No Transpose',
$     M-J-JB+1, N-J-JB+1, JB, -ONE, A, J+JB,
$     J, DESCA, A, J, J+JB, DESCA, ONE, A,
$     J+JB, J+JB, DESCA )
```


PBLAS – SPECIFICATIONS

```
DGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX,
       BETA, Y, INCY )
                          ↕
PDGEMV( TRANS, M, N, ALPHA, A, IA, JA, DESCA,
        X, IX, JX, DESCX, INCX,
        BETA, Y, IY, JY, DESCY, INCY )
```

- In the PBLAS, the increment specified for vectors is always global. So far only INCX=1 and INCX=DESCX(1) are supported.

BLAS	PBLAS
INTEGER LDA	INTEGER IA, JA, DESCA(8)
INTEGER INCX	INTEGER INCX, IX, JX, DESCX(8)
A, LDA	A, IA, JA, DESCA
X, INCX	X, IX, JX, DESCX, INCX

- PBLAS matrix transposition routine (REAL):

```
PDTRAN( M, N, ALPHA, A, IA, JA, DESCA, BETA,
        C, IC, JC, DESCC )
```

- Level 1 BLAS functions have become PBLAS subroutines. Ouput Scalar correct in operand scope.

>> **SPMD PROGRAMMING MODEL** <<

PBLAS – REFERENCES

- PBLAS software and documentation can be obtained via:
  - WWW: http://www.netlib.org/scalapack,
  - (anonymous) ftp ftp.netlib.org:

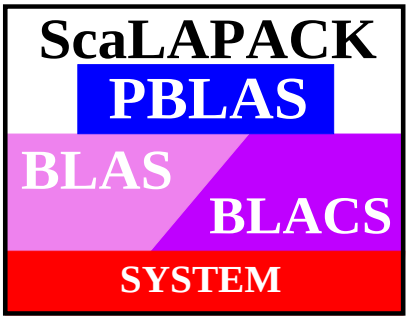
    cd scalapack; get index
  - email netlib@www.netlib.org with the message:

    send index from scalapack
- Comments and questions can be addressed to

  scalapack@cs.utk.edu

- J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Technical Report UT CS-95-292, LAPACK Working Note #100, University of Tennessee, 1995.

ScaLAPACK IMPLEMENTATION



BLAS (Performance and Portability)

- Blocked data access (level 3 BLAS) yields good local performance

BLACS (Performance and Portability)

- Dedicated machine dependent communication kernel,
- 2D array-based and ID-less communication,
- Code re-entrance, multiple overlapping process grids,
- Safe co-existence with other parallel libraries.

Parallel Level 2 and 3 BLAS: PBLAS

Goal

- Port sequential library for shared memory machine that use the BLAS to distributed memory machines with little effort.
- Reuse the existing software by hiding the message passing in a set of BLAS routines.
- Parallel implementation of the BLAS that understands how the matrix is layed out and when called can perform not only the operation but the required data transfer.

LAPACK → ScaLAPACK

BLAS → PBLAS (BLAS, BLACS)

- Quality (maintenance)
- Portability (F77 - C)
- Efficiency - Reuseability (BLAS, BLACS)
- Hide Parallelism in (P)BLAS

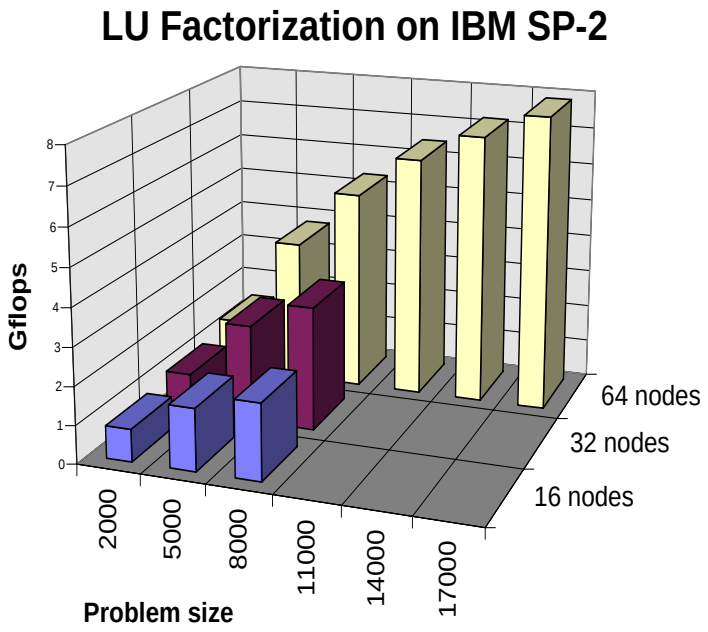
SEQUENTIAL LU FACTORIZATION CODE

```
      DO 20 J = 1, MIN( M, N ), NB
         JB = MIN( MIN( M, N )-J+1, NB )

         Factor diagonal and subdiagonal blocks and test for exact
         singularity.

         CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ),
     $                IINFO )

         Adjust IINFO and the pivot indices.

         IF( IINFO.EQ.0 .AND. IINFO.GT.0 ) IINFO = IINFO + J - 1
         DO 10 I = J, MIN( M, J+JB-1 )
            IPIV( I ) = J - 1 + IPIV( I )
   10    CONTINUE

         Apply interchanges to columns 1:J-1.

         CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )

         IF( J+JB.LE.N ) THEN

            Apply interchanges to columns J+JB:N.

            CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
     $                   IPIV, 1 )

            Compute block row of U.

            CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $                  JB, N-J-JB+1, ONE, A( J, J ), LDA,
     $                  A( J, J+JB ), LDA )
            IF( J+JB.LE.M ) THEN

               Update trailing submatrix.

               CALL DGEMM( 'No transpose', 'No transpose',
     $                     M-J-JB+1, N-J-JB+1, JB, -ONE,
     $                     A( J+JB, J ), LDA, A( J, J+JB ), LDA,
     $                     ONE, A( J+JB, J+JB ), LDA )
            END IF
         END IF
   20 CONTINUE
```

PARALLEL LU FACTORIZATION CODE

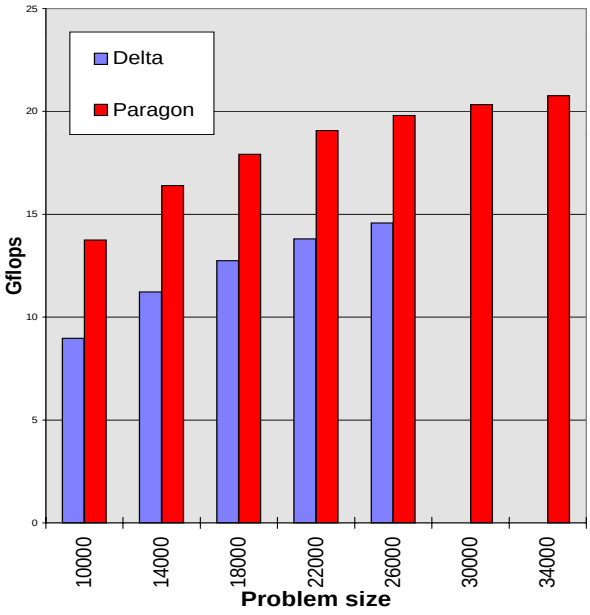
```
      DO 10 J = JA, JA+MIN(M,N)-1, DESCA( 4 )
         JB = MIN( MIN(M,N)-J+JA, DESCA( 4 ) )
         I = IA + J - JA

         Factor diagonal and subdiagonal blocks and test for exact
         singularity.

         CALL PDGETF2( M-J+JA, JB, A, I, J, DESCA, IPIV, IINFO )

         Adjust IINFO and the pivot indices.

         IF( IINFO.EQ.0 .AND. IINFO.GT.0 )
     $      IINFO = IINFO + J - JA

         Apply interchanges to columns JA:J-JA.

         CALL PDLASWP( 'Forward', 'Rows', J-JA, A, IA, JA, DESCA,
     $                 J, J+JB-1, IPIV )

         IF( J-JA+JB+1.LE.N ) THEN

            Apply interchanges to columns J+JB:JA+N-1.

            CALL PDLASWP( 'Forward', 'Rows', N-J-JB+JA, A, IA,
     $                    J+JB, DESCA, J, J+JB-1, IPIV )

            Compute block row of U.

            CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $                   JB, N-J-JB+JA, ONE, A, I, J, DESCA, A, I,
     $                   J+JB, DESCA )
            IF( J-JA+JB+1.LE.M ) THEN

               Update trailing submatrix.

               CALL PDGEMM( 'No transpose', 'No transpose',
     $                      M-J-JB+JA, N-J-JB+JA, JB, -ONE, A,
     $                      I+JB, J, DESCA, A, I, J+JB, DESCA,
     $                      ONE, A, I+JB, J+JB, DESCA )
            END IF
         END IF
   10 CONTINUE
```


PERFORMANCE RESULTS



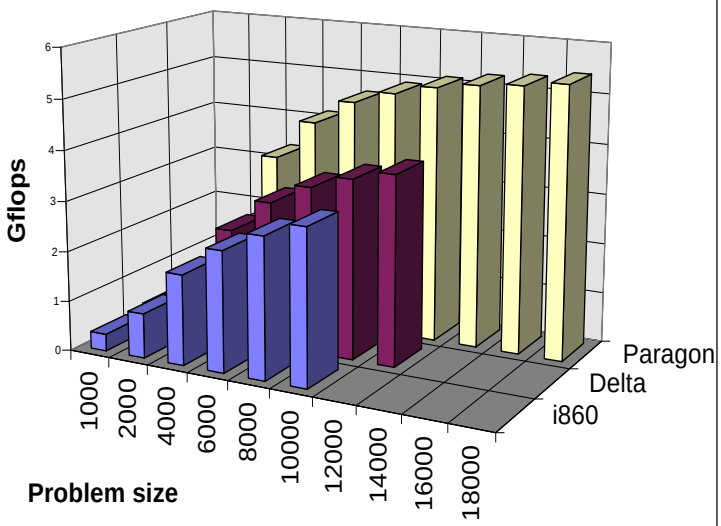
PERFORMANCE RESULTS

QR Factorization on Intel machines (512 processors)

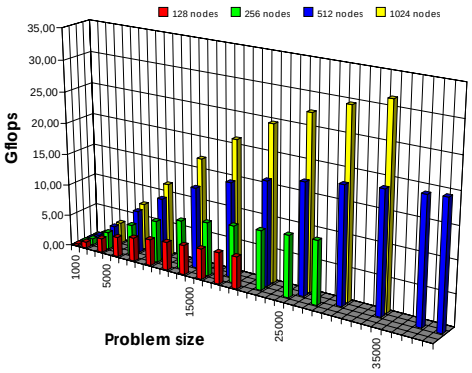


PERFORMANCE RESULTS

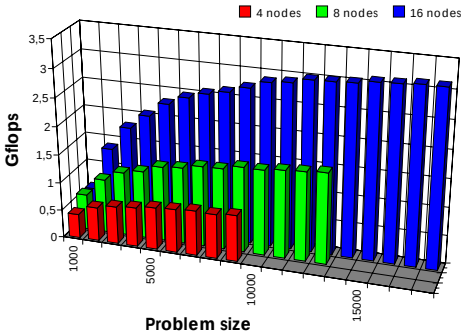
QR Factorization on 128 nodes Intel Machines



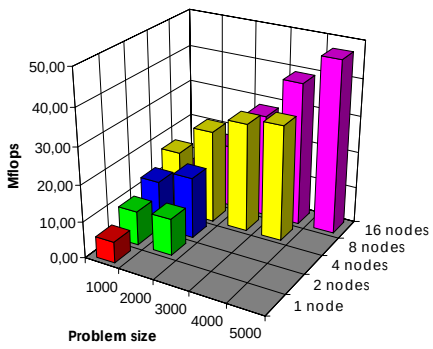
LU Factorization on Intel Paragon

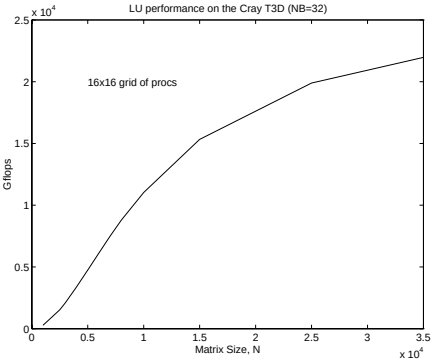


LU Factorization on IBM SP-2



LU Factorization on Sparc-5 (Ethernet)





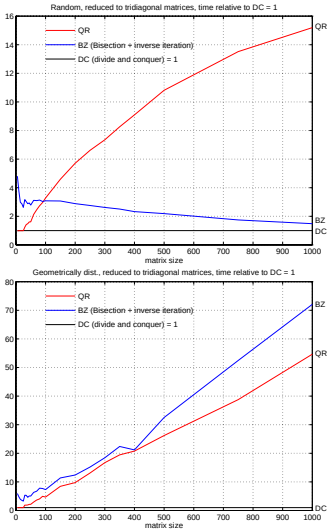
Symmetric Eigenproblem $Ax = \lambda x,\ A = A^T$

- Two phases:
  Phase 1: Reduce A to tridiagonal T:
  $$A = UTU^T, \quad T = \begin{bmatrix} a_1 & b_1 & & 0 \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{bmatrix}$$
  Phase 2: Find e-vals, vectors of T: $T = Q\Lambda Q^T$
  So $A = (UQ)\Lambda(UQ)^T \equiv V\Lambda V^T$
- Phase 2 is bottleneck (3x Phase 1)
- LAPACK: Cuppen's divide-and-conquer method
- ScaLAPACK:
  - Bisection + inverse iteration with limited reorthogonalization: fast but may fail to compute orthogonal eigenvectors if eigenvalues clustered
  - QR iteration: slower but reliable

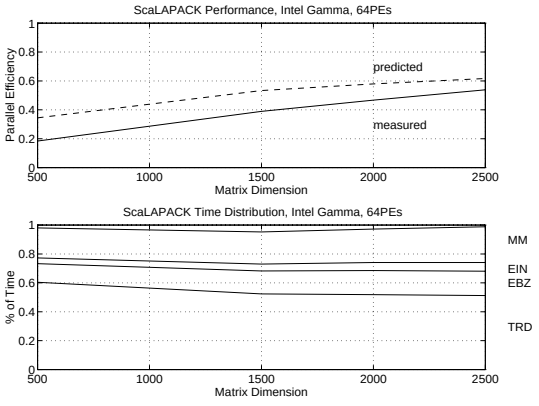
Outline of Cuppen's Divide and Conquer

1. Goal: compute $T = Q\Lambda Q^T,\ \Lambda = \mathrm{diag}(\lambda_i),\ QQ^T = I$
2. $$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix} = \left[\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right] + vv^T$$
3. Solve $T_1 = Q_1\Lambda_1 Q_1^T$ and $T_2 = Q_2\Lambda_2 Q_2^T$ recursively
4. $$T = \left[\begin{array}{c|c} Q_1\Lambda_1 Q_1^T & \\ \hline & Q_2\Lambda_2 Q_2^T \end{array}\right] + vv^T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot (D + zz^T) \cdot \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix}$$
   where
   $$D = \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} \text{ and } z = \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} v$$
5. Eigenvalues λ of $D + zz^T$ are roots of "secular equation"
   $$1 + \sum_i \frac{z_i^2}{\lambda - d_{ii}} = 0$$
6. Eigenvectors v_j of $D + zz^T$ are $(D - \lambda_j I)^{-1} z$
7. Eigenvectors of T are $\begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot V$

Speedups with Divide-and-Conquer



ScaLAPACK Performance on 64 Processor Intel Gamma Expert Driver - Bisection + Inverse iteration



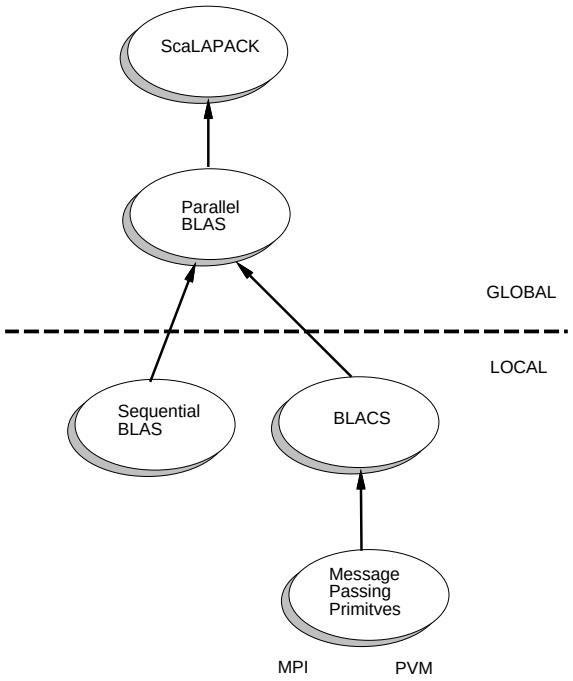
ScaLAPACK Sparse matrix activities

- ARPACK - Arnoldi-Pack - D. Sorensen
  - Algorithms for symmetric and nonsymmetric linear systems, eigenproblems, generalized eigenproblems
  - User's matrix-vector product, rest serial
  - Based on clever identity for compressing partial Arnoldi factorization
- CAPSS - Cartesian Parallel Sparse Solver - P. Raghavan and M. Heath
  - Parallel sparse Cholesky
  - For finite difference/finite element problems
  - Usese coordinates from underlying mesh
  - Cartesian nested dissection to layout data
- Sparse LU with Pivoting
  - Exploits data locality using supernodal approach
  - Up to 120 Mflops on IBM RS6000/590, depending on sparsity pattern
  - Fastest code available on many test cases, along with UMFPACK
  - To be released soon

WHAT IS AVAILABLE TO YOU ?

- **Tools**
  - BLACS (CMMD, MPL, NX, PVM)
  - PBLAS
  - REDISTRIBUTION
  - PUMMA
- **Factorizations**
  - LU + solve
  - Cholesky + solve
  - QR, QL, LQ, RQ, QRP
- **Matrix inversions**
- Q, QC, Q^TC, CQ, CQ^T
- **Reductions**
  - to Hessenberg form
  - to symmetric tridiagonal form
  - to bidiagonal form
- **Symmetric eigensolver**

Software Framework



PVM calls within the BLACS

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of serial, parallel, and vector computers hooked together by a network to appear as one large computer.

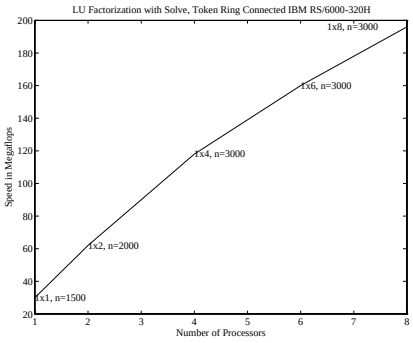
BLACS (Basic Linear Algebra Communication Subroutines) is a software package to perform communication at the same level as the BLAS are for computation.

- **BLACS have been written in terms of calls to PVM.**
- **Allows network of workstations to be used with minimal change.**

MPI – Message Passing Interface

Standard for Message Passing

- **BLACS → MPI underway**



LU Results on IBM Cluster

Proc	Time	Speedup	Proc	Mflop/s	Speedup
1	1030		1 (n=1500)	33	
2	403	2.6	2 (n=2000)	64	1.9
4	155	6.6	4 (n=3000)	116	3.5
6	115	8.9	6 (n=3000)	156	4.7
8	93	11.0	8 (n=3000)	194	5.9

SCALAPACK – ONGOING WORK

- **HPF**
  - HPF supports the 2D-block-cyclic distribution used by ScaLAPACK
  - HPF like calling sequence (Global indexing scheme)
- **Portability: MPI implementation of the BLACS**
- **More flexibility added to the PBLAS**
- **More testing and timing programs**
- **Condition estimators**
- **Iterative refinement of linear system solutions**
- **SVD**
- **Linear Least Square solvers**
- **Banded systems**
- **Non symmetric eigensolvers**
- ...