Software Standards and

Tools for Concurrent Computing

Jack Dongarra

Computer Science Department
University of Tennessee

and

Mathematical Sciences Section
Oak Ridge National Laboratory

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

Participants

Ed Anderson (Cray)
Zhaojun Bai (U Kentucky)
Michael Berry (U Tenn)
Adam Bojanczyk (Cornell U)
Tony Chan (UCLA)
Inderjit Dhillon (UC Berkeley)
Jack Dongarra (U Tenn, ORNL)
Jeremy Du Croz (NAG)
Vince Fernando (NAG)
Sven Hammarling (NAG)
Steve Huss-Lederman (SRC)
Velvel Kahan (UC Berkeley)
Rencang Li (ORNL)
Alan McKenney (NYU)
Beresford Parlett (UC Berkeley)
Chris Puscasiu (UC Berkeley)
Roldan Pozo (NIST)
Huan Ren (UC Berkeley)
Charles Romine (ORNL)
Eunice Santos (UC Berkeley)
Dan Sorensen (Rice U)
Xiaobai Sun (Duke U)
Anna Tsao (SRC)
Henk van der Vorst (Utrecht U)
Krešimir Veselić (U Hagen)
Clint Whaley (U Tenn)
Majed Sidani (Cray)
Richard Barrett (LANL)
Chris Bischof (ANL)
Soumen Chakrabarti (UC Berkeley)
Jaeyoung Choi (U Tenn)
June Donato (ORNL)
Zlatko Drmač (U Hagen)
Victor Eijkhout (UCLA)
Ming Gu (UC Berkeley, LBL)
Mike Heath (U Illinois)
Bo Kågström (U Umeå)
Youngbae Kim (U Tenn)
Xiaoye Li (UC Berkeley)
Susan Ostrouchov (U Tenn)
Antoine Petitet (U Tenn)
Peter Poromaa (U Umeå)
Padma Raghavan (U Tenn)
Howard Robinson (UC Berkeley)
Jeff Rutter (UC Berkeley)
Ivan Slapničar (U Split)
Ken Stanley (UC Berkeley)
Bernard Tourancheau (ENS - Lyon)
Robert van de Geijn (U Texas)
Paul Van Dooren (U Illinois)
David Walker (ORNL)
Kathy Yelick (UC Berkeley)

With the cooperation of
Cray, IBM, Convex, DEC, Fujitsu, NEC, NAG, IMSL

Supported by ARPA, NSF, DOE

Three Basic Ways of Improving Computer Productivity

- **Increasing component operating speed**
- **Parallelizing the process of calculation**
- **Making computers that are specialized for a certain class of problem**

Why Parallel Computers?

- **Desire to solve bigger, more realistic applications problems.**
- **Fundamental limits are being approached.**
- **More cost effective solution**

Example: Weather Prediction (Navier-Stokes) with 3D Grid around the Earth

$$6 \quad variables \begin{cases} temperature \\ pressure \\ humidity \\ 3-wind \quad velocity \end{cases}$$

- **1 Kilometer Cells**
- **10 slices** $\rightarrow 5 \times 10^9$ **cells**
- **each cell is 8 bytes,** 2×10^{11} **Bytes** $= 200$ **GBytes**
- **at each cell will perform 100 ops/cell**
- **1 minute time step**
- $\frac{100ops/cell \times 5 \times 10^9 cells}{1min \times 60sec/min} = 8GFlop/s$

Goals of a Parallel Architecture

- **High Performance:**
  - **Performance > fastest uniprocessors**
- **Competitive cost-performance:**
  - **Cost-performance is competitive with workstations**
- **Scalable performance and cost-performance**
- **General-purpose: cost-effective for a wide range of applications**

Scalable Multiprocessors

What is Required?

- **Must scale the local memory bandwidth linearly.**
- **Must scale the global interprocessor communication bandwidth.**
- **Scaling memory bandwidth cost-effectively requires separate, distributed memories.**
- **Cost-effectiveness also requires best price-performance in individual processors.**

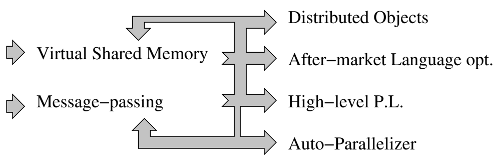
What we get

- **Compelling Price/Performance**
- **Tremendous scalability**
- **Tolerable entry price**
- **Tackle intractable problems**

Architecture Alternatives

- **Distribute, private memories using message-passing to communicate among processors.**
- **Single address space implemented with physically distributed memories.**
- **Both approaches require:**
  - **Attention to locality of access.**
  - **Scalable interconnection technology.**

Distributed Computing



Nevermid all that: Focus on Message−passing

Build a good message−passing system

Can write MP applications

Can build other abstraction on top

Performance Numbers Comparing Various RISC Processors
Using the Linpack Benchmark
(All based on actual runs.)
Linpack n=100 based on Fortran code only.
Ax=b n=1000 is based on a blocked algorithm (LAPACK routine)
using the Level 3 BLAS as provided by the vendor.

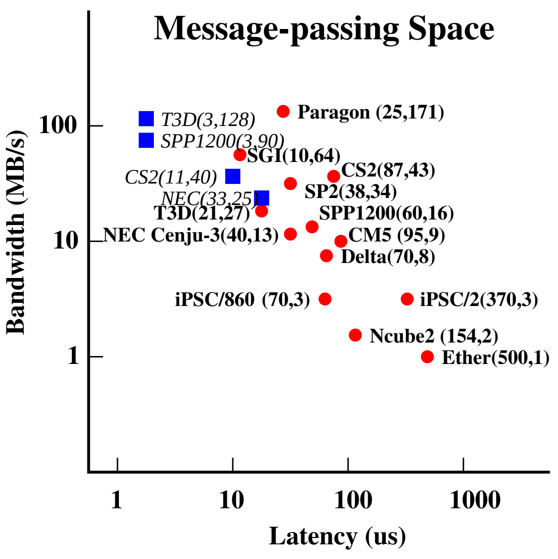
Machine	MHz	Cycle time nsec	Linpack n=100 Mflops	% peak	Ax = b n=1000 Mflops	% peak	Theor Peak Mflops
DEC Alpha	300	3.3	140	23%	411	69%	600
IBM Power2	66	15	130	48%	236	89%	266
SGI Power	75	13.3	101	33%	260	87%	300
HP 735	100	10	61	31%	107	54%	198
DEC Alpha	200	5	43	22%	155	78%	200
DEC Alpha	182	5	39	21%	141	77%	182
IBM RS-580	66	15	38	30%	104	83%	125
DEC Alpha	160	6	36	23%	114	71%	160
DEC Alpha	150	7	30	20%	107	71%	150
IBM RS-550	42	24	26	31%	70	83%	84
HP 730/750	66	15	24	36%	47	71%	66
SGI Indy2/Crim	100	10	15	30%	32	64%	50
IBM RS-530	25	40	15	30%	42	84%	50
KSR (1 proc)	40	25	15	38%	31	78%	40
Intel i860	40	25	10	25%	34	85%	40
CRAY T90	454	2.2	522	29%	1576	88%	1800
CRAY C90	238	4.2	387	41%	902	95%	952
CRAY J90	100	10	115	51%	193	96%	200
CRAY Y-MP	166	6	161	48%	324	97%	333
CRAY X-MP	118	8.5	121	51%	218	93%	235
CRAY 3	481	2.1	241	25%			962
CRAY 2	244	4.1	120	25%	384	79%	488
CRAY 1	80	12.5	27	17%	110	69%	160

Table 1: Multiprocessor Latency and Bandwidth.

Machine	OS	Latency $n = 0$ (μs)	Bandwidth $n = 10^6$ (MB/s)	$n_{1/2}$ bytes	Theoretical Bandwidth (MB/s)
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	63	15	1000	250
Convex SPP1200 (sm m-n)	SPP-UX 3.0.4.1	11	71	1000	250
Cray T3D (sm)	MAX 1.2.0.2	3	128	363	300
Cray T3D (PVM)	MAX 1.2.0.2	21	27	1502	300
Intel Paragon	OSF 1.0.4	29	154	7236	175
Intel Paragon	SUNMOS 1.6.2	25	171	5856	175
Intel Delta	NX 3.3.10	77	8	900	22
Intel iPSC/860	NX 3.3.2	65	3	340	3
Intel iPSC/2	NX 3.3.2	370	2.8	1742	3
IBM SP-1	MPL	270	7	1904	40
IBM SP-2	MPI	35	35	3263	40
KSR-1	OSF R1.2.2	73	8	635	32
Meiko CS2 (sm)	Solaris 2.3	11	40	285	50
Meiko CS2	Solaris 2.3	83	43	3559	50
nCUBE 2	Vertex 2.0	154	1.7	333	2.5
nCUBE 1	Vertex 2.3	384	0.4	148	1
NEC Cenju-3	Env. Rel 1.5d	40	13	900	40
NEC Cenju-3 (sm)	Env. Rel 1.5d	34	25	400	40
SGI	IRIX 6.1	10	64	799	1200
TMC CM-5	CMMD 2.0	95	9	962	10
Ethernet	TCP/IP	500	0.9		1.2
FDDI	TCP/IP	900	9.7		12
ATM-100	TCP/IP	900	3.5		12

Table 2: Computation Performance.

Machine	OS	Clock cycle MHz	(nsec)	Linpack 100 Mfls	(ops/cl)	Linpack 1000 Mfls	(ops/cl)	Latency us	(cl)
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	100	(8.33)	65	(.54)	123	(1.02)	63	(7560)
Convex SPP1200 (sm m-n)								11	(1260)
Cray T3D (sm)	MAX 1.2.0.2	150	(6.67)	38	(.25)	94	(.62)	3	(450)
Cray T3D (PVM)								21	(3150)
Intel Paragon	OSF 1.0.4	50	(20)	10	(.20)	34	(.68)	29	(1450)
Intel Paragon	SUNMOS 1.6.2							25	(1250)
Intel Delta	NX 3.3.10	40	(25)	9.8	(.25)	34	(.85)	77	(3080)
Intel iPSC/860	NX 3.3.2	40	(25)	9.8	(.25)	34	(.85)	65	(2600)
Intel iPSC/2	NX 3.3.2	16	(63)	.37	(.01)	–	(–)	370	(5920)
IBM SP-1	MPL	62.5	(16)	38	(.61)	104	(1.66)	270	(16875)
IBM SP-2	MPI	66	(15.15)	130	(1.97)	236	(3.58)	35	(2310)
KSR-1	OSF R1.2.2	40	(25)	15	(.38)	31	(.78)	73	(2920)
Meiko CS2 (MPI)	Solaris 2.3	90	(11.11)	24	(.27)	97	(1.08)	83	(7470)
Meiko CS2 (sm)								11	(990)
nCUBE 2	Vertex 2.0	20	(50)	.78	(.04)	2	(.10)	154	(3080)
nCUBE 1	Vertex 2.3	8	(125)	.10	(.01)	–	(–)	384	(3072)
NEC Cenju-3	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	40	(3000)
NEC Cenju-3(sm)	Env Rev 1.5d	75	(13.3)	23	(.31)	39	(.52)	34	(2550)
SGI Power Challenge	IRIX 6.1	90	(11.11)	126	(1.4)	308	(3.42)	10	(900)
TMC CM-5	CMMD 2.0	32	(31.25)	–	(–)	–	(–)	95	(3040)



Challenges in developing distributed memory libraries

- How to integrate libraries?
  - no standard software
  - many parallel languages
  - many flavors of message passing
  - various parallel programming models
  - assumptions made about parallel environment
    - granularity
    - topology
    - overlapping of communication / computation
    - development tools
- Where is the data?
  - who owns it?
  - optimal data distribution not a function of individual routines but of overall integration of components
- Who determines data layout and/or transformations?
  - determined by user?
  - determined by library developer?
  - choose from a small set?
  - allow for dynamic data distributions?
  - load balancing?

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

Problem Solving Environments

MATLAB Mathematica

- **Interactive**
- **Graphical interface for specific problems**
- **Algorithm and data structures hidden**
- **Workstation / computational server**
- **Heterogeneous platforms**

The NetSolve Client

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**

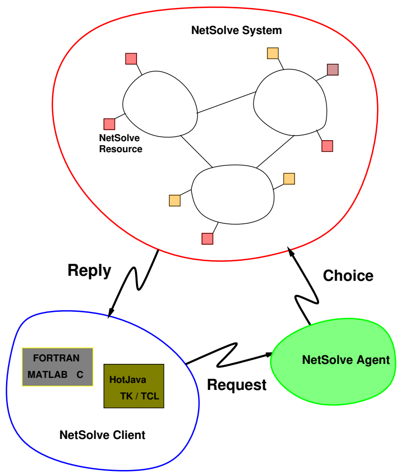
Each server uses the installed scientific packages

Example of packages :

- **BLAS**
- **LAPACK**
- **ScaLAPACK**
- **Ellpack**

Connection to other systems

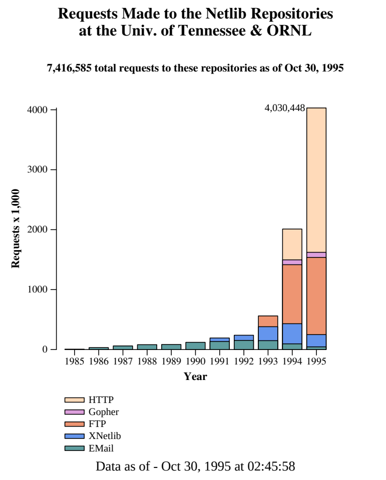
- **NEOS**



NetSolve : The Solution Engine

Netlib - Network Access to Mathematical Software and Data

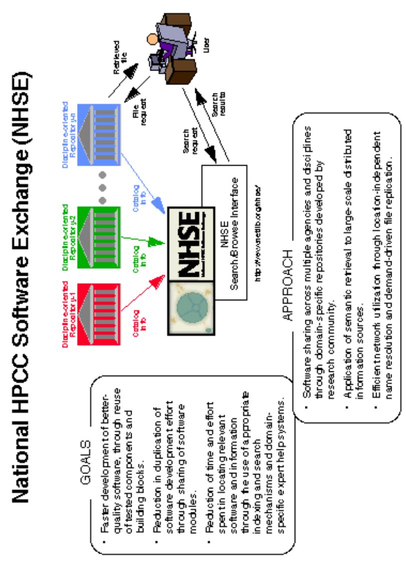
- Began in 1985
  - JD and Eric Grosse, AT&T Bell Labs
- Motivated by the need for cost-effective, timely distribution of high-quality mathematical software to the community.
- Designed to send, by return electronic mail, requested items.
- Automatic mechanism for the disseminate of public domain software.
  - Still in use and growing
  - Mirrored at a number of sites
    - ∗ netlib2.cs.utk.edu
    - ∗ netlib1.epm.ornl.gov
    - ∗ research.att.com
    - ∗ netlib.no
    - ∗ unix.hensa.ac.uk
    - ∗ ftp.zip-berlin.de



Breakdown of requests to each Netlib library

(an alphabetical listing is also available.)
Data as of 10/30/95 at 02:59:34

Library Name	Number of accesses
lapack	448,920
pvm3	358,474
linpack	248,863
slatec	237,228
blas	171,696
linalg	123,473
eispack	122,410
slatec/src	113,593
toms	110,740
clapack	108,079
c++	93,009
f2c	92,529
benchmark	82,552
master	68,367
f2c/src	63,180
minpack	58,578
fn	56,849
fftpack	56,596
na-digest	47,595
port	47,289
slatec/lin	44,742
hence	44,209
confdb	36,799
c++/answerbook	36,271
slatec/chk	35,487

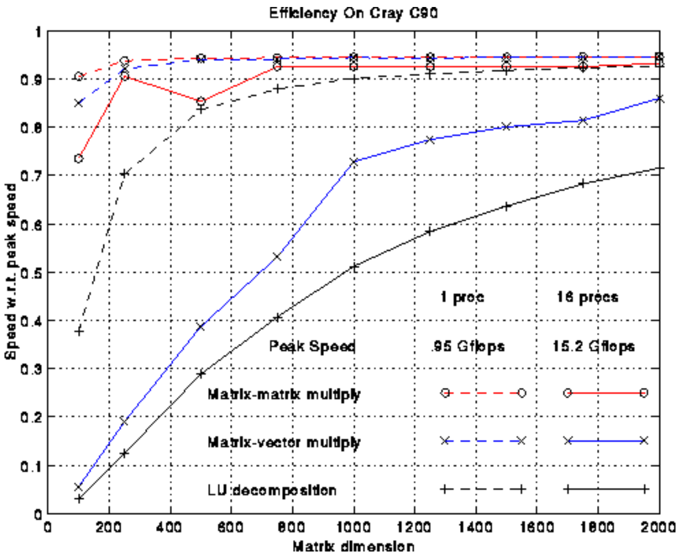


Current libraries under development

- Mutlicomputer Toolbox (Bangalore, Skjellum [92]), C, data parallelism, large-scale libraries for dense and sparse systems.
- Simplified Linear Equations Solvers, SLES (W. Gropp, B. Smith, [93]) C, SPMD data parallelism, direct and iterative methods
- ScaLapack Fortran 77, SPMD, data parallelism, linear solvers, factorizations, and eigenvalue problems; supports 2-D SBS decomposition
- CMSSL (Thinking Machines, CM-2 [90], CM-5 [93]), CM-Fortran, SIMD data parallelism, block-cyclic distributions
- pC++ library (Gannon et. al., [92]) parallel extension to C++ (parblock),
- P++ (Quinlan [92]) distributed structured grids, adaptive mesh refinement, supports 1-D decomposition, C++ with explicit message passing.
- ...

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

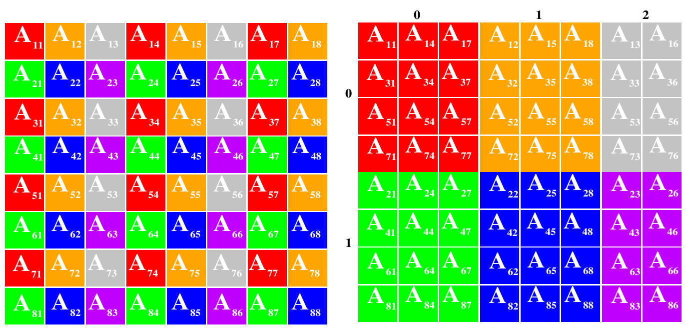
- **Standards**
  - **Portable**
  - **Vendor-supported**
  - **Scalable**
- **High efficiency**
- **Protect programming investment, if possible**
- **Avoid low-level details**
- **Scalability**

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 – 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.

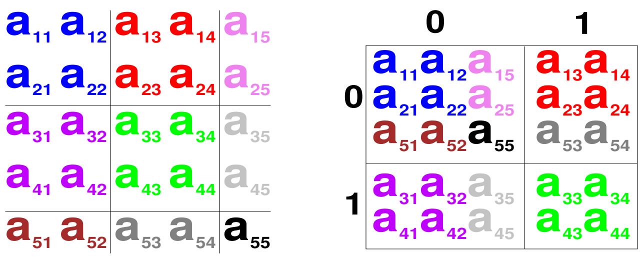
- **Similar functionality** as the BLAS: distributed vector-vector, matrix-vector and matrix-matrix operations,
- **Simplification of the parallelization** of dense linear algebra codes: especially when implemented on top of the BLAS,
- **Clarity:** code is shorter and easier to read,
- **Modularity:** 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
  ⇒ **load balanced computations, scalability**,
- Locally, the scattered **columns are stored contiguously** (FORTRAN "Column-major")

  ⇒ **re-use of the BLAS** (leading dimension `LLD_`).



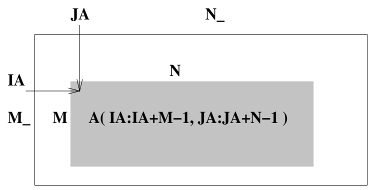
5 x 5 matrix partitioned in 2 x 2 blocks — 2 x 2 process grid point of view

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 )
```

$\updownarrow$

```
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 <<

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 INFO and the pivot indices.

      IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = 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 INFO and the pivot indices.

      IF( INFO.EQ.0 .AND. IINFO.GT.0 )
  $      INFO = 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
```


PBLAS – REFERENCES

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

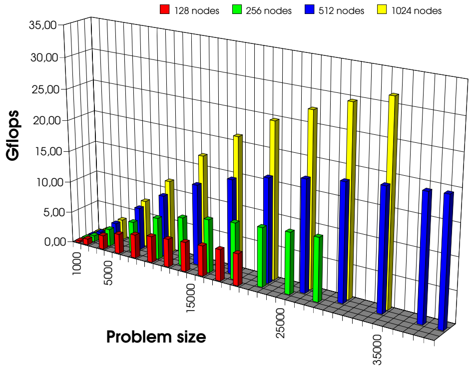
    `cd scalapack; get index`
  - email `netlib@www.netlib.org` with the message:

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

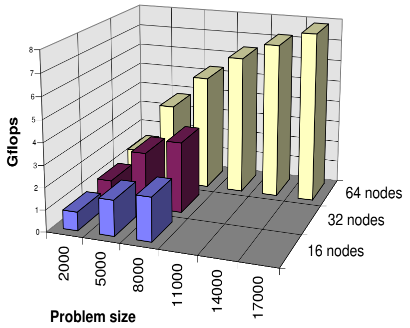
  `scalapack@cs.utk.edu`

- 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.

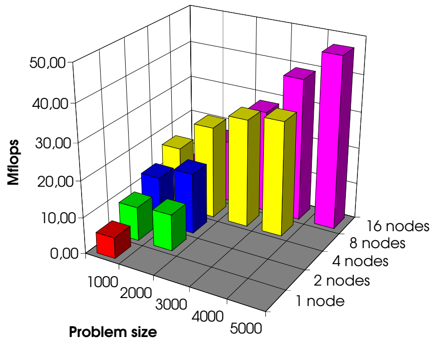
LU Factorization on Intel Paragon

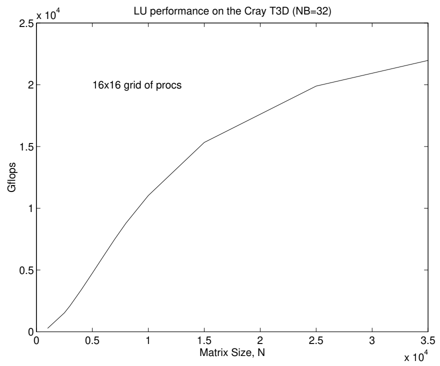


LU Factorization on IBM SP-2

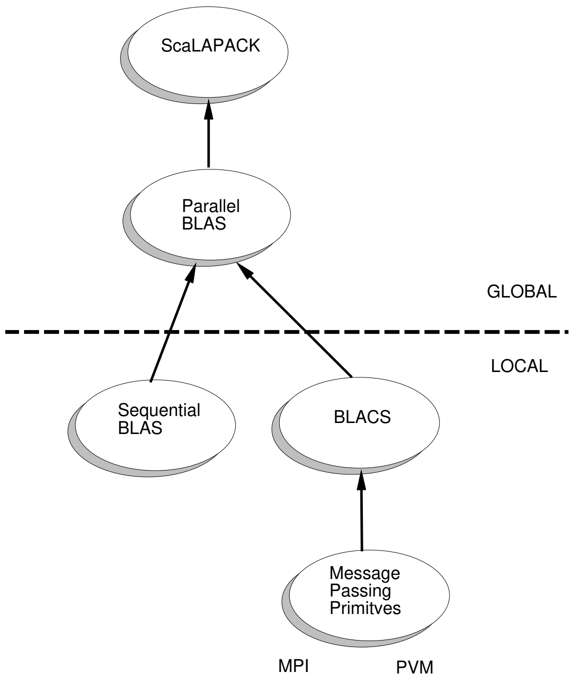


LU Factorization on Sparc-5 (Ethernet)





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.

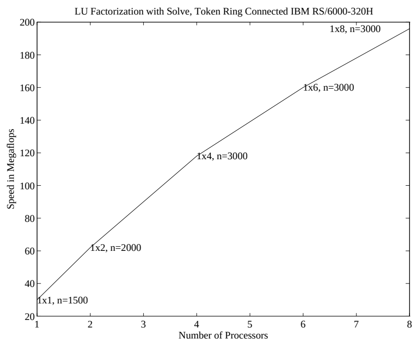
BLACS (Basic Linear Algebra Communication Subroutines) is a software package to perform communication at the same level as the BLAS are for computation.

- 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

What is MPI?

A standard message passing interface

- explicit message passing in
- application programs on
- MIMD distributed memory concurrent computers.

Why is a Standard Needed?

- Portability and ease-of-use,
- Provides hardware vendors with well-defined set of routine to implement efficiently,
- Pre-requisite for the development of concurrent software industry,
- Will lead to more widespread use of concurrent computers.

MPI

- Message passing paradigm widely used on parallel machines, and well understood, but each vendor has their own variant.
- Lack of a standard has significantly impeded the development of portable software and libraries for message-passing machines.
- MPI defines a set of routines which are useful to a wide range of users and efficiently implementable on a wide range of computers.
- Intended to become a widely used standard, gradually replacing the vendor- specific and other interfaces used by C and Fortran programs today.
- MPIF is not sanctioned or supported by any official standards organization, but virtually all of the major vendors have participated.

Communication Scope

- In MPI, a process is specified by:
  - a group
  - a rank relative to the group $(0, 1, 2, \ldots, N-1)$
- A message label is specified by:
  - a message context
  - a message tag relative to the context
- Groups are used to partition process space
- Contexts are used to partition "message label space"
- Groups and contexts may be bound together to form a communicator object
- A communicator defines the scope of a communication operation

Contexts

- A context is a non-wildcardable component of,
  - the message label,
  - the communication scope.
- A context is identified by an integer ID.
- Context are used to create independent message streams.
- Contexts are used to disambiguate message selection when an application calls a library routine that performs message passing. Indeterminacy may arise,
  - if processes enter the library routine asynchronously,
  - if processes enter the library routine synchronously, but there are outstanding communication operations.

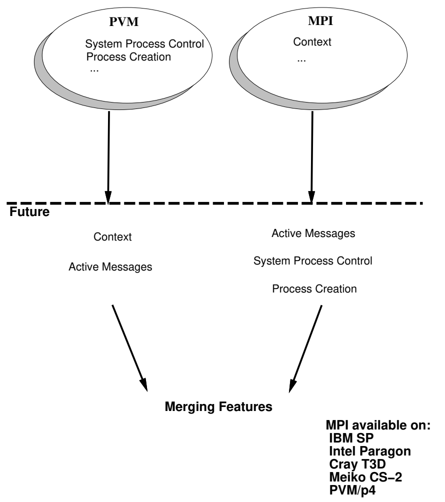
Use of Contexts

- Different phases of an application are assigned different contexts to avoid their messages being confused.
- Library routines need to be passed a group and an array of contexts so they can construct the communicators that they need.

Strategy – Near Term Approach

- Choose most important routines in library
  - One sided factorizations LU, LL^T, QR
  - Two sided factorizations UHU^T, UTU^T, UBV^T
  - Eigenvalues for symmetric and non-symmetric problem
  - Large sparse eigenproblems
  - Sparse direct solvers
  - Explore idea of templates
  - Implement toolbox approach
  - Develop prototype software
- Choose most significant programming models
  - Explicit message passing (BLACS)
  - Data Parallel (Fortran 90, HPF ...)
- Choose most important machines
  - Cray T3D, Intel Paragon, IBM SP
  - Other (TERA, Meiko...)
- Software initially Fortran 77 and C, then Fortran 90, and HPF
- Compare approaches

PVM and MPI Future



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
- ...

Fortran 90
Array Operations and Intrinsics

- A = A + B
- A = A * B ($a_{ij} = a_i j b_i j$)
- WHERE (A ¿ 0.0) SQRT(A)
- scalar = DOT˙PRODUCT(x, y)
- y = alpha*MATMUL(A, x) + beta*y
- C = alpha*MATMUL(A, B) + beta*C
- C = MATMUL(CONJG(TRANSPOSE(A)), B)

Upper Triangular Solve $Ux = b$

```
x = b
DO i = n, 1, -1
   x(i)      = x(i)/u(i,i)
   x(1:i-1) = x(1:i-1) - u(1:i-1,i)*x(i)
END DO
```

$$UX = B$$

```
X = B
DO i = n, 1, -1
   x(i,:)      = x(i,:)/u(i,i)
   x(1:i-1,:) = x(1:i-1,:) - u(1:i-1,i)*x(i,:)
END DO
```


High Performance Fortran (HPF)

- Fortran 90 plus distribution directives
- A few parallel constructs (eg FORALL)
- Version 1.0 was published on May 3rd, 1993
- Almost certain to become a de facto standard
- Supported by many manufactures
- Likely to be part for future Fortran Standard (??)

SCALAPACK – REFERENCES

- ScaLAPACK software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/scalapack`,
  - WWW: `http://www.netlib.org/lapack/lawns`,
  - (anonymous) ftp `netlib2.cs.utk.edu`:
    - cd scalapack; get index
    - cd lapack/lawns; get index
  - email `netlib@ornl.gov` with the message:

    `send index from scalapack`
- Comments and questions can be addressed at

  `scalapack@cs.utk.edu`
- LAPACK Working Notes:
  #43, #55, #57, #58, #61, #65, #73, #80, #86, #91,
  #92, **#93**, **#94**, **#95**, #96, **#100**.
- J. Dongarra and D. Walker, *Software Libraries for Linear Algebra Computations on High Performance Computers*, SIAM Review, Vol. 37, (2), pp. 151 – 180, 1995.

Who needs numerical software?

- **Kinds of users and their priorities:**
  - **Engineers and scientists generally, who make 12,000 requests/day to netlib and keep NAG and IMSL in business:**
    1. **Easy interface, hidden details**
    2. **Reliability**
    3. **Speed**
  - **HPCC community, solving biggest and hardest problems:**
    1. **Speed**
    2. **Access to details for tuning**
    3. **Reliable enough for their application**
  - **Teachers and Students**
    1. **Ease of understanding**
    2. **Access to some details for learning**
- **Can we please everyone with one "medium"?**

Iterative Methods

- No single iterative method that can solve any given sparse linear system in reasonable time and with reasonable memory requirements.
- Rate of convergence depends on certain properties of the system, and these properties are often unknown in practice.
- For many very large linear systems, iterative methods are at this moment the only choice.
- Necessary to have a variety of iterative methods available.
- Through Templates the user is guided into the world of iterative methods, with only a minimum of linear algebra knowledge required.
- Using Templates the user can build software at several levels of sophistication, depending on requirements and goals.

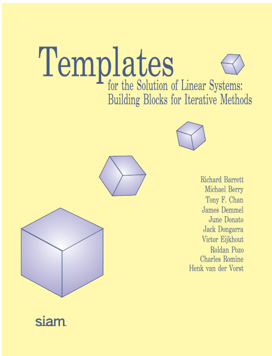
Iterative Methods

- The iteration schemes consist of one or two matrix vector products, a couple of inner-products and a few vector updates.
- The algorithms also allow for some form of preconditioning, which means that in each step a linear system $Mz = r$ has to be solved, where M has to be specified by the user.
- These operations are pretty well understood, and it is easy to see how the algorithms work.
- These basic iteration Templates may be used for simple computational experiments in order to get familiarized with the basic iteration methods and their mutual differences.
- The user is given extensive advice in the manual on how to expand the basic codes to reliable software for practical problems.

Motivation from Iterative Methods

Algorithm sketches for

- Gauss Siedel, Jacobi SOR, SSOR, etc.
- PCG, Bi-CG, QMR, GMRES, etc.
- Preconditioners
- Domain Decomposition
- Parallelism



Templates for Sparse Matrix Computations

- **Alternate to software**
- **Describe the basic features of the algorithms**
- **Language not so important**
- **Communication between disciplines**
- **Clear documentation**
- **Test cases**
- **Provide understanding of the results**
- **Allow customization**
- **Serve as a pedagogical role**
- **Retain the delicate numerical details**
- **Describe parallel opportunities**

A Template for an Algorithm Includes

1. A high-level description of the algorithm.
2. A description of when it is effective, including conditions on the input, and estimates of the time, space or other resources required. If there are natural competitors, they will be referenced.
3. A description of available refinements and user-tunable parameters, as well as advice on when to use them.
4. Pointers to complete or partial implementations, perhaps in several languages or for several architectures (each parallel architecture). These implementation expose those details suitable for user-tuning, and hide the others.
5. Numerical examples, on a common set of examples, illustrating both easy cases and difficult cases.
6. Trouble shooting advice.
7. Pointers to texts or journal articles for further information.

In addition to individual templates, there will be a *decision tree* to help steer the user to the right algorithm, or subset of possible algorithms, based on a sequence of questions about the nature of the problem to be solved.

Building Blocks for Iterative Solution of Linear Systems

- **Algorithm sketches for**
  - **Algorithm**
    - **Jacobi, Gauss-Seidel, SOR**
    - **CG, Precond-CG, CG-Squared, Bi-CG**
    - **Bi-CG-Stab, QMR, GMRES, Chebychev**
  - **Matlab script**
  - **Fortran code in a 2-D array**
  - **Fortran code using reverse commmunication**
  - **Preconditioners**
    - **Jacobi, SSOR, Incomplete Factoriation, Polynominal**
  - **Convergence tests**
  - **Performance Summary**
  - **Sparse Data Structures**
  - **Hints on parallel implementation**
  - **Available from SIAM or http://www.netlib.org/templates/index.html**

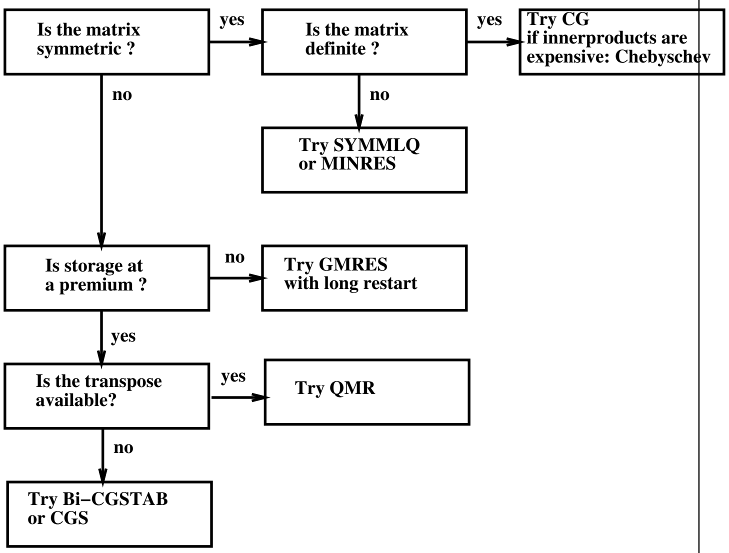
Three Potential User Communities for such Tools

- **The "HPCC" (High Performance Computing and Communication) community consists of those scientists trying to solve the largest, hardest applications in their fields. They desire high speed, access to algorithmic details for performance tuning, and reliability for their problem.**
- **Engineers and scientists generally desire easy-to-use, reliable software, that is also reasonably efficient.**
- **Students and teachers want simple but generally effective algorithms, which are easy to explain and understand.**

- **Matrix data type:**
  - **General Dense.**
  - **Banded Dense.**
  - $\boxed{\textit{General Sparse.}}$

    Preferred Operations (in order of decreasing power):
    - $\boxed{\textit{Solve } (A - \sigma I)x = b}$
    - **Solve** $Ax = b$
    - **Multiply** $x = A \cdot b$
    - **Don't know.**
  - **Sparse with Special Structure.**
  - **Dense but compactly representable.**
  - **Other.**

- **Machine type:**
  - **Serial.**
  - **Parallel.**
    - **Shared Memory.**
    - **Distributed Memory.**



$x^{(0)}$ is an initial guess
for $j = 1, 2, ...$
  Solve r from $Mr = b - Ax^{(0)}$
  $v^{(1)} = r/\|r\|_2$
  $s := \|r\|_2 e_1$
  for $i = 1, 2, ..., m$
    Solve w from $Mw = Av^{(i)}$
    for $k = 1, ..., i$
      $h_{k,i} = (w, v^{(k)})$
      $w = w - h_{k,i} v^{(k)}$
    end
    $h_{i+1,i} = \|w\|_2$
    $v^{(i+1)} = w / h_{i+1,i}$
    apply $J_1, ..., J_{i-1}$ on $(h_{1,i}, ..., h_{i+1,i})$
    construct J_i, acting on ith and $(i+1)$st component
    of $h_{.,i}$, such that $(i+1)$st component of $J_i h_{.,i}$ is 0
    $s := J_i s$
    if $s(i+1)$ is small enough then (UPDATE($\tilde{x}, i$) and quit)
  end
  UPDATE($\tilde{x}, m$)
end

In this scheme UPDATE($\tilde{x}, i$)
replaces the following computations:

Compute y as the solution of $Hy = \tilde{s}$, in which
the upper $i \times i$ triangular part of H has $h_{i,j}$ as
its elements (in least squares sense if H is singular),
$\tilde{s}$ represents the first i components of s
$\tilde{x} = x^{(0)} + y^{(1)} v^{(1)} + y^{(2)} v^{(2)} + ... + y^{(i)} v^{(i)}$
$s^{(i+1)} = \|b - A\tilde{x}\|_2$
if $\tilde{x}$ is an accurate enough approximation then quit
else $x^{(0)} = \tilde{x}$

Figure 2.6: The Preconditioned GMRES(m) Method

MAIN-PROGRAM

```
SUBROUTINE CG(N, B, X, WORK, LDW,
     ITER, RESID, MATVEC, PSOLVE,
     INFO)
10   CALL  CGREVCOM()

c    return from CGREVCOM
c    with job request (IJOB)
     IF (IJOB .eq. 1) CALL MATVEC()
     IF (IJOB .eq. 2) CALL PSOLVE()
c    do matrix-vector multiply with X
     IF (IJOB .eq. 3) CALL MATVEC()
c    do stopping test #2
     IF (IJOB .eq. 4) CALL STOPTEST2()
     IF (IJOB .eq. -1) GO TO 20
     GO TO 10

20   RETURN
     END
```

revcom return

```
SUBROUTINE CGREVCOM(N, B, X, WORK, LDW,
     ITER, RESID, INFO, NDX1, NDX2, SCLR1,
     SCLR2, IJOB)
c    do matrix-vector multiply with X
     IJOB = 3; RETURN
c    ITERATIVE METHOD MAIN LOOP

c         do preconditoner solve
          IJOB = 2; RETURN

c         do matrix-vector multiply
          IJOB = 1; RETURN

c         do stopping test
          IJOB = 4; RETURN
c    END OF ITERATIVE METHOD MAIN LOOP
c    all done, return to MAIN-PROGRAM
     IJOB = -1; RETURN

     STOP
     END
```

Note 1: Diagram only represents reverse communication interactions.

Note 2: CGREVCOM() logic to resume execution at appropriate place not shown.

```
*     First time call always init.
      IJOB = 1

 1    CONTINUE

          CALL CGSREVCOM(N, B, X, WORK, LDW, ITER, RESID, INFO,
     $                   NDX1, NDX2, SCLR1, SCLR2, IJOB)

*         On a return from REVCOM() we use the table
*         to figure out what is reqd.
          IF (IJOB .eq. -1) THEN
*           revcom wants to terminate, so do it.
            GOTO 2
          ELSEIF (IJOB .eq. 1) THEN
*           call matvec.
            CALL MATVEC(SCLR1, WORK(NDX1), SCLR2, WORK(NDX2))
          ELSEIF (IJOB .eq. 2) THEN
*           call solve.
            CALL PSOLVE(WORK(NDX1), WORK(NDX2))
          ELSEIF (IJOB .eq. 3) THEN
*           call matvec with X.
            CALL MATVEC(SCLR1, X, SCLR2, WORK(NDX2))
         ELSEIF (IJOB .EQ. 4) THEN
*           do stopping test 2
*           if FirstTime, set info  so that BNRM2 is computed.
            IF( FTFLG ) INFO = -1
            CALL STOPTEST2(N, WORK(NDX1), B, BNRM2, RESID, TOL, INFO)
            FTFLG = .FALSE.
         ENDIF
*        done what revcom asked, set IJOB & go back to it.
         IJOB = 2
         GOTO 1
*     come here to terminate
 2    CONTINUE
*
      RETURN
*     End of CGS
      END
```



```
function [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
%  -- Iterative template routine --
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, et al, SIAM Publications, 1993.
%     (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%  [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
% cg.m solves the symmetric positive definite linear system Ax=b
% using the Conjugate Gradient method with preconditioning.
% input   A        REAL symmetric positive definite matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         M        REAL preconditioner matrix
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it
  flag = 0;  iter = 0;                        % initialization
  bnrm2 = norm( b );
  if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
  r = b - A*x;
  error = norm( r ) / bnrm2;
  if ( error < tol ) return, end
  for iter = 1:max_it                         % begin iteration
     z  = M \ r;
     rho = (r'*z);
     if ( iter > 1 ),                         % direction vector
        beta = rho / rho_1;
        p = z + beta*p;
     else
        p = z;
     end
     q = A*p;
     alpha = rho / (p'*q );
     x = x + alpha * p;                      % update approximation vector
     r = r - alpha*q;                        % compute residual
     error = norm( r ) / bnrm2;              % check convergence
     if ( error <= tol ), break, end
     rho_1 = rho;
  end
  if ( error > tol ) flag = 1; end           % no convergence
% END cg.m
```

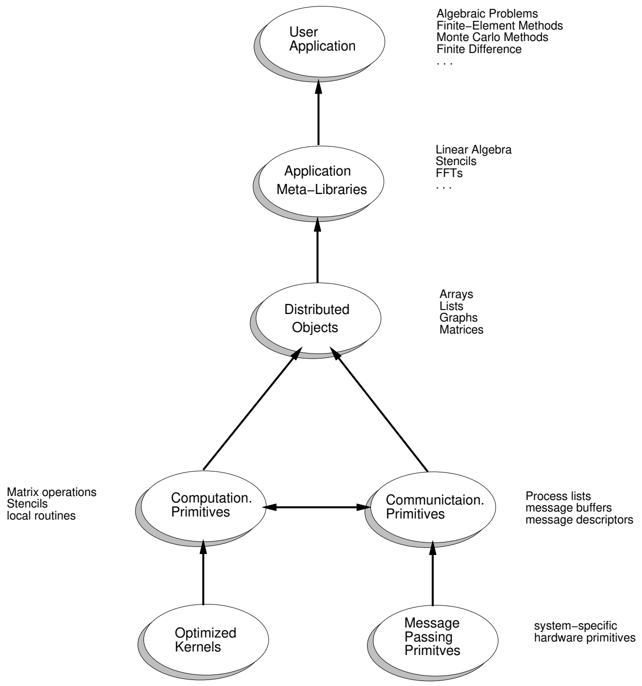

A Practical Solution: Meta-Libraries

- **separate the data implementation details out of the fundamental algorithms**
- **describe library algorithms in terms of abstract base classes (inheritance)**
- **specify only the interface**
- **utilize late-binding (polymorphism) to determine data distributions and implementations**

Need good design to choose proper objects and abstractions to balance

- **performance**
- **reusability**
- **generality**
- **control**
- **flexibility**

Software Hierachy



Templates for Sparse Matrix Computations

- Alternate to software
- Describe the basic features of the algorithms
- Language not so important
- Communication between disciplines
- Clear documentation
- Test cases
- Provide understanding of the results
- Allow customization
- Serve as a pedagogical role
- Retain the delicate numerical details
- Describe parallel opportunities

Building Blocks for Iterative Solution of Linear Systems

- Some users want algorithms which are fast for their particular application even if not reliable as general methods.
- They want the tools to customize iterative methods for solving the linear systems that arise in their specific problem.
- Offer reusable software templates
  - Algorithm
    - Jacobi, Gauss-Seidel, SOR
    - CG, Precond-CG, CG-Squared, Bi-CG
    - Bi-CG-Stab, QMR, GMRES, Chebychev
  - Matlab script
  - Fortran code in a 2-D array
  - Preconditioners
  - Convergence tests
  - Performance Summary
  - Data Structures
  - Hints on parallel implementation

Preconditioned Conjugate Gradient Method for $Ax = b$

Initial $r^{(0)} = b - Ax^{(0)}$
for $i = 1, 2, \ldots$
  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;
end

```
r = b - A*x;
for (int i=1; i<maxiter; i++){
    z = M.solve(r);
    rho = r * z;
    if (i==1)
       p = z;
    else{
       beta = rho1/ rho0;
       p = z + p * beta;
    }
    q = A*p;
    alpha = rho1 / (p*q);
    x += alpha * p;
    r -= alpha * q;
    if (norm(r)/normb < tol) break;
}
```


OO Sparse Matrix Libraries

- motivated by "Building Blocks for Iterative Solution of Linear Systems"
- based on Level 3 Sparse BLAS interface
- support distributed formats: compressed row / column
- implementing parallel Level 3 Sparse BLAS
- inital development of conventional sparse matrix classes
  - compressed column (Harwell/Boeing)
  - compressed row
  - jagged diagonal storage
  - compressed diagonal storage
  - block compressed row/col storage
  - skyline storage

OO Sparse Matrix Libraries (contd.)

- **Iterative algorithms**
  - Stationary Methods (SOR),
  - Conjugate Gradient (CG),
  - Generalized Minimal Residual (GMRES),
  - Minimum Residual (MINRES),
  - Quasi-Minimal Residual (QMR),
  - Chebyshev Iteration (Cheb),
  - Conjugate Gradient Squared (CGS),
  - Biconjugate Gradient (BiCG),
  - Biconjugate Gradient Stabilized (Bi-CGSTAB)
- **Preconditioners are user-defined**
  - **some basic stationary preconditioners provided**
  - **application specific**
  - **more difficult to parallelize**
- **Current Development**
  - **Phase I: development of conventional sparse matrix classes:**
  - **Phase II: development of shared memory sparse matrix**
  - **Phase III: development of distributed memory sparse matrix objects with Level 3 Sparse BLAS**

A Simple Example

$$\frac{\|Ax-b\|_\infty}{N\|A\|_\infty\|x\|_\infty\epsilon} < 1$$

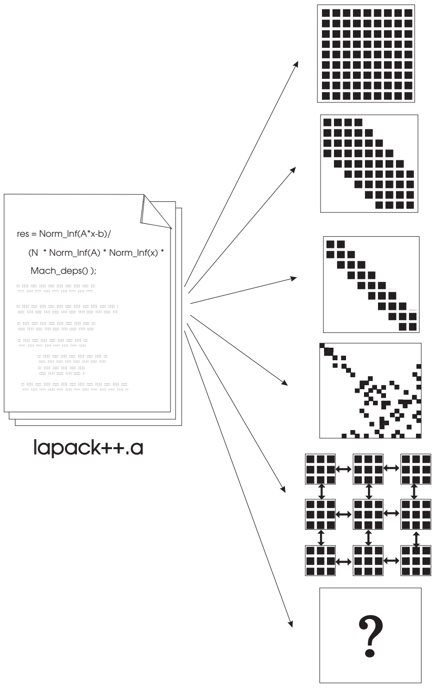
F77 LAPACK / BLAS

```
      DCOPY(N, AXMB, 1, B, 1)
      DGEMV('No Transpose', N, N, 1.0, A, N, X, 1, -1.0, AXMB, 1)
      DO 100 I=1,N
100   R(I) = DASUM(N, A(I), N)
      INDEX = IDAMAX(N, R, 1)
      RES = DASUM(N,AXMB,1) / ( N* R(INDEX) * DASUM(N,X)
     &          * DLMACH('e'))
```

LAPACK++

```
res =           Norm_Inf(A*x-b) /
       (N * Norm_Inf(A) * Norm_Inf(x) * Mach_deps() );
```


One Single Lapack++ Library



Conventional LAPACK Storage Formats

Hermitian

Storage	Hermitian matrix A	Storage in array A
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ \bar{a}_{01} & a_{11} & a_{12} & a_{13} \\ \bar{a}_{02} & \bar{a}_{12} & a_{22} & a_{23} \\ \bar{a}_{03} & \bar{a}_{13} & \bar{a}_{23} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} \\ * & a_{11} & a_{12} & a_{13} \\ * & * & a_{22} & a_{23} \\ * & * & * & a_{33} \end{matrix}$
Lower	$\begin{pmatrix} a_{00} & \bar{a}_{10} & \bar{a}_{20} & \bar{a}_{30} \\ a_{10} & a_{11} & \bar{a}_{21} & \bar{a}_{31} \\ a_{20} & a_{21} & a_{22} & \bar{a}_{32} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & * & * & * \\ a_{10} & a_{11} & * & * \\ a_{20} & a_{21} & a_{22} & * \\ a_{30} & a_{31} & a_{32} & a_{33} \end{matrix}$

Triangular

Storage	Triangular matrix A	Storage in array A
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ & a_{11} & a_{12} & a_{13} \\ & & a_{22} & a_{23} \\ & & & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} \\ * & a_{11} & a_{12} & a_{13} \\ * & * & a_{22} & a_{23} \\ * & * & * & a_{33} \end{matrix}$
Lower	$\begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & * & * & * \\ a_{10} & a_{11} & * & * \\ a_{20} & a_{21} & a_{22} & * \\ a_{30} & a_{31} & a_{32} & a_{33} \end{matrix}$

Banded

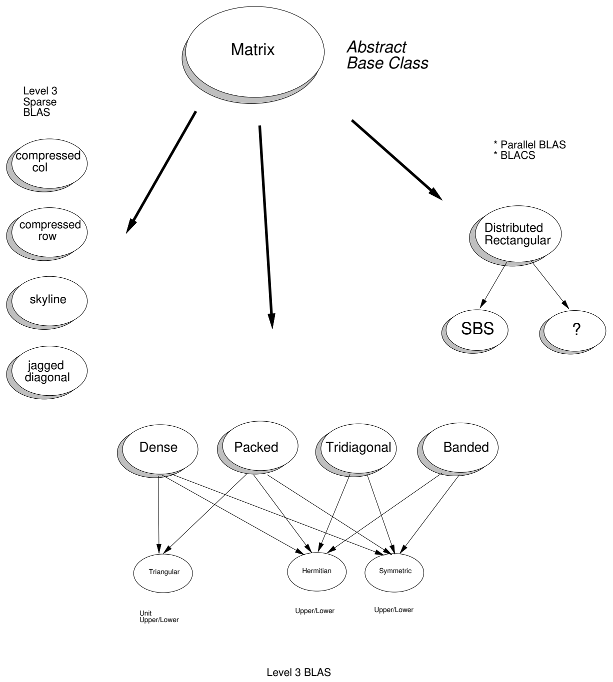
Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{00} & a_{01} & & & \\ a_{10} & a_{11} & a_{12} & & \\ a_{20} & a_{21} & a_{22} & a_{23} & \\ & a_{31} & a_{32} & a_{33} & a_{34} \\ & & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} * & a_{01} & a_{12} & a_{23} & a_{34} \\ a_{00} & a_{11} & a_{22} & a_{33} & a_{44} \\ a_{10} & a_{21} & a_{32} & a_{43} & * \\ a_{20} & a_{31} & a_{42} & * & * \end{matrix}$

Triangular Packed

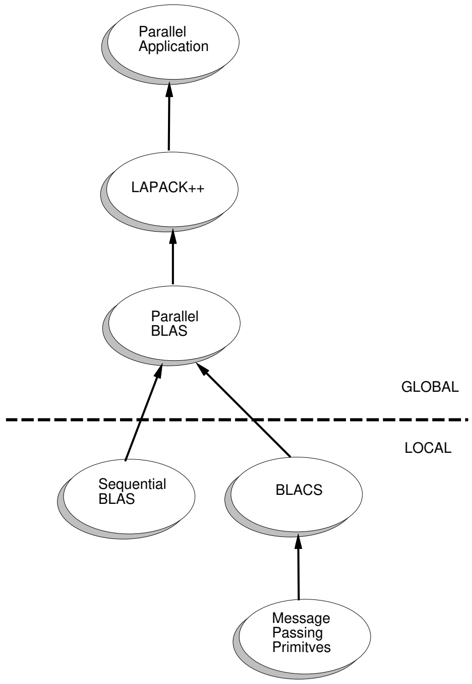
Storage	Triangular matrix A	Packed storage in array AP
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ & a_{11} & a_{12} & a_{13} \\ & & a_{22} & a_{23} \\ & & & a_{33} \end{pmatrix}$	$a_{00}\ \underbrace{a_{01}\ a_{11}}\ \underbrace{a_{02}\ a_{12}\ a_{22}}\ \underbrace{a_{03}\ a_{13}\ a_{23}\ a_{33}}$
Lower	$\begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\underbrace{a_{00}\ a_{10}\ a_{20}\ a_{30}}\ \underbrace{a_{11}\ a_{21}\ a_{31}}\ \underbrace{a_{22}\ a_{32}}\ a_{33}$

Orthogonal $Q = \underbrace{H_1 H_2 \ldots H_q}_{\text{Householder vectors}}$

Linear Algebra Classes



Object Oriented Software Framework

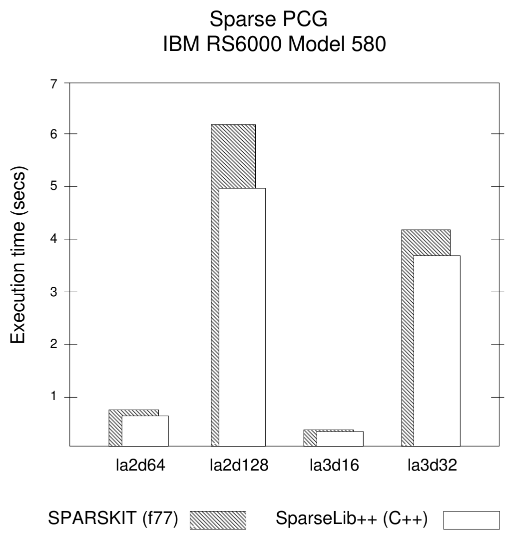


Iterative Methods Meta-Library (IML++)

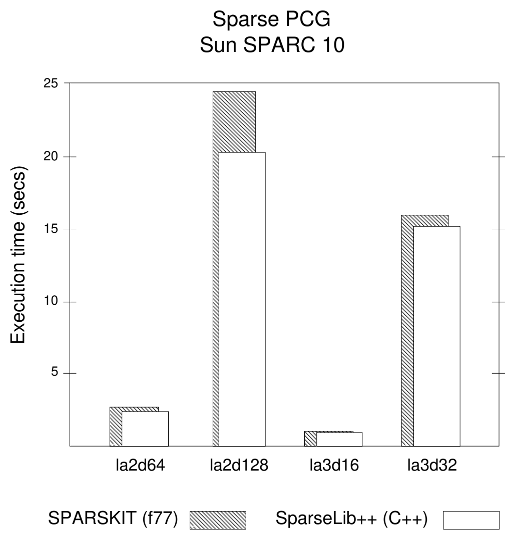
- Stationary Methods (SOR)
- Conjugate Gradient (CG)
- Generalized Minimal Residual (GMRES)
- Minimum Residual (MINRES)
- Quasi-Minimal Residual (QMR)
- Chebyshev Iteration (Cheb)
- Conjugate Gradient Squared (CGS)
- Biconjugate Gradient (BiCG)
- Biconjugate Gradient Stabilized (Bi-CGSTAB)

SparseLib++ Matrix Classes

- coordinate
- compressed column (Harwell/Boeing)
- compressed row
- jagged diagonal storage
- compressed diagonal storage
- block compressed row/col storage
- skyline storage

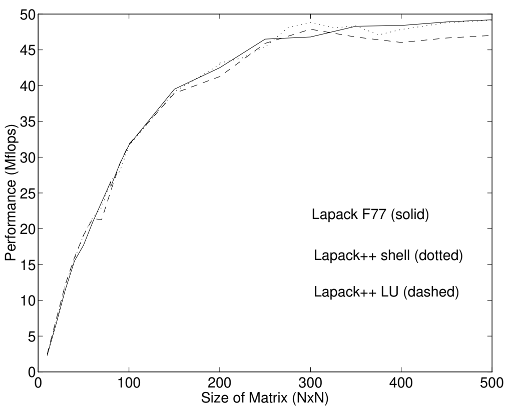


PCG using compressed row matrices; optimized Fortran (xlf -O) Sparskit and C++ (xlC -O) SparseLib++ (Sparse Blas).



PCG using compressed row matrices; optimized Fortran (f77 -O) Sparskit and C++ (g++ -O) SparseLib++ (Sparse Blas).

What does the C++ design cost?



Performance of LAPACK++ LU factorization on the IBM RS/6000 Model 550 workstation, using GNU g++ v. 2.3.1 and BLAS routines from the IBM ESSL library.

Level 3 Sparse Blas
Large (Duff, Marrone, Radicatti)
(S. Carney, M. Heroux, G. Li, and K. Wu)

- **sparse matrix products,**

$$C \leftarrow \alpha\ op(A)\ B + \beta C$$

- **solution of triangular systems,**

$$C \leftarrow \alpha D\ op(A)^{-1}\ B + \beta C$$

- **reordering of a sparse matrix (permutations),**

$$A \leftarrow A\ op(P)$$

- **conversion of one data format to another,**

$$A' \leftarrow A$$

where α and β are scalars
B and C are dense (multiple vectors)
D is a (block) diagonal matrix,
A and A' are sparse
$op(A)$ is either A or A^T.

- **framework for extending data structures**

Object Oriented interface

```
XYYYMM( TRANSA, M, N, K, ALPHA, args(A),
        B, LDB, BETA, C, LDC, WORK, LWORK)
```

Point entry:

COO	**Coordinate**
CSC	**Comressed sparse column**
CSR	**Compressd sparse row**
DIA	**Sparse diagonal**
ELL	**Ellpack/Itpack**
JAD	**Jagged diagonal**
SKY	**Skyline**

Block entry:

BCO	**Block coordinate**
BSC	**Block comressed sparse column**
BSR	**Block compressd sparse row**
BDI	**Block sparse diagonal**
BEL	**Block Ellpack/Itpack**
VBR	**Variable block compressed sparse row**