The Situation:

Parallel scientific applications are typically

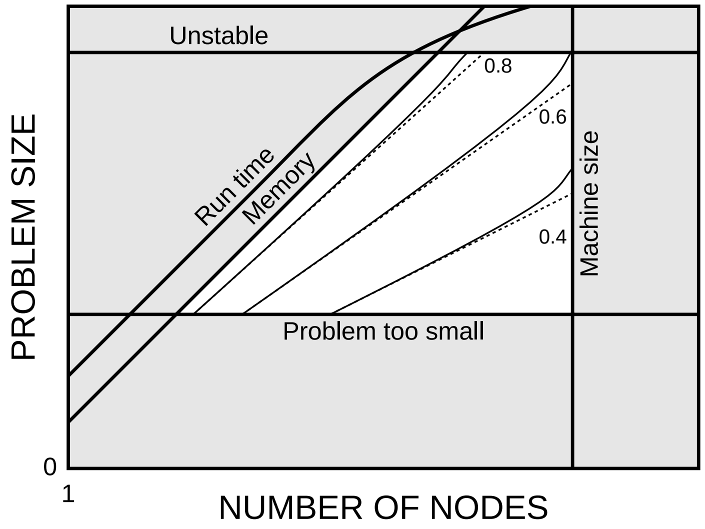
- written from scratch, or manually adapted from sequential programs
- using simple SPMD model
- in Fortran or C
- with explicit message passing primitives

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

Scalability

- A parallel algorithm is scalable if the concurrent efficiency can be held constant as the number of processors increases by increasing the problem size.
- The *isoefficiency function*, $\rho_\epsilon(N_p)$, **gives the problem size necessary to maintain some fixed efficiency, ϵ, as N_p increases.**



Memory Contention:

- The single most important bottleneck on high-performance machines—even with multiple load and store pipelines.

Objective:

- Keep the ratio of memory references to floating-point operations as low as possible.
- Key is to retain data in the highest level of the memory hierarchy.
  - vector registers
  - cache/local memory

Transfer Rate

Maximum rate of transfer of information to or from a memory device.

Increasing the computational power without a corresponding increase in the bandwidth of data to memory can create a serious bottleneck.

MFLOPS are easy—Bandwidth is difficult

One processor of a CRAY C90 can transfer 750 Mwords/sec.

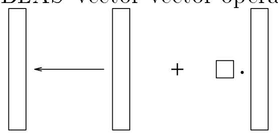
Complete system a total of 12 Gwords/sec as the peak memory bandwidth.

MFLOPS and Memory Bandwidth

Machine	Peak MFLOPS	Peak Transfer (MW/s)	Ratio
CRAY 1	160	80	.5
CRAY X-MP/4	940	1411	1.5
CRAY Y-MP/8	2667	4000	1.5
CRAY T-90 32	32000	48000	1.5
CRAY 2S	1951	970	.5
Cyber 205	400	600	1.5
ETA 10G	644	966	1.5
Fujitsu VP-200	533	533	1
Fujitsu VP-400	1066	1066	1
IBM 3090/600-VF	798	400	.5
NEC SX-2	1300	2000	1.5
Hitachi 820/80	3000	2000	.67
IBM RS/6000-590	266	266	1
Convex C-210	50	25	.5
Alliant FX/80	188	22	.12
Ardent Titan 4	64	32	.5

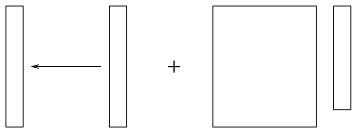
Level 1, 2, and 3 BLAS

- Level 1 BLAS–vector-vector operations



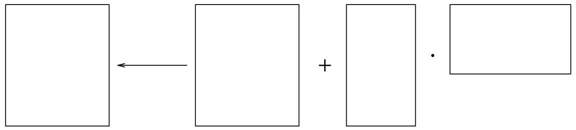
$y \leftarrow y + \alpha x$, also dot product, ...

- Level 2 BLAS–matrix-vector operations



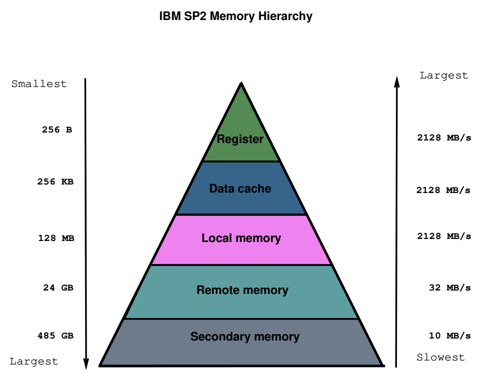
$y \leftarrow y + Ax$, also triangular solve, rank-1 update

- Level 3 BLAS–matrix-matrix operations



$A \leftarrow A + BC$, also block triangular solve, rank-k update

Why Higher Level BLAS?



- Can only do arithmetic on data at top
  $\longrightarrow$ keep active data as close to top of hierarchy as possible
- Higher level BLAS lets us do this:

		mem ref	flops	flops/mem ref
Level 1 BLAS	$y \leftarrow y + \alpha x$	3n	2n	2/3
Level 2 BLAS	$y \leftarrow y + Ax$	n^2	$2n^2$	2
Level 3 BLAS	$A \leftarrow A + BC$	$4n^2$	$2n^3$	n/2

- On parallel machines
  higher level BLAS $\longrightarrow$increase granularity $\longrightarrow$ lower synchronization cost

Blocked Algorithms have been developed for:

- LU factorization
- Cholesky factorization
- Factorization of symmetric indefinite matrices
- Matrix inversion
- Banded LU and Cholesky factorization
- QR factorization
- Form Q or $Q^T B$
- Orthogonal reduction to:
  - Hessenberg
  - symmetric tridiagonal
  - bidiagonal
- Block QR iteration for nonsymmetric eigenvalue problems

LAPACK

- Linear algebra library in Fortran 77:
  - solution of systems of equations
  - solution for eigenvalue problems
- Combine algorithms from LINPACK and EISPACK into a single package
- Efficient on a wide range of high-performance computers
  - Single and double precision routines
  - Complex counterparts
- User interface similar to LINPACK
- Built on the Level 1, 2, and 3 BLAS

- In LAPACK, most of the parallelism is in the BLAS.

  Advantage of writing parallel programs with the BLAS:

  - <u>CLARITY</u> - code is shorter and easier to read
  - <u>MODULARITY</u> - gives programmer larger building blocks
  - <u>PERFORMANCE</u> - manufacturers will provide machine-specific BLAS
  - <u>PORTABILITY</u> - machine dependency are confined to the BLAS

LAPACK timing results for a cray Y/MP (Mflops)

Name	Matrix size N					
	32	64	128	256	512	1024
LU						
SGETRF (1 proc)	40	108	195	260	290	304
(2 proc)	32	91	229	408	532	588
(4 proc)	32	90	260	588	914	1097
(8 proc)	32	90	205	375	1039	1974
Cholesky						
SPOTRF (1 proc)	34	95	188	259	289	301
(2 proc)	29	84	221	410	539	594
(4 proc)	29	84	252	598	952	1129
(8 proc)	29	84	273	779	1592	2115
QR Factorization						
SGEQRF (1 proc)	54	139	225	275	294	301
(2 proc)	50	134	256	391	505	562
(4 proc)	50	136	292	612	891	1060
(8 proc)	50	133	328	807	1476	1937

The BLAS as the Key to Portability

Speed in megaflops of Level 2 and Level 3 BLAS operations on a CRAY C90

(all matrices are of order 1000; U is upper triangular)

Number of processors:	1	2	4	8	16
Level 2: $y \leftarrow \alpha Ax + \beta y$	899	1780	3491	6783	11207
Level 3: $C \leftarrow \alpha AB + \beta C$	900	1800	3600	7199	14282
Level 2: $x \leftarrow Ux$	852	1620	3063	5554	6953
Level 3: $B \leftarrow UB$	900	1800	3574	7147	13281
Level 2: $x \leftarrow U^{-1}x$	802	1065	1452	1697	1558
Level 3: $B \leftarrow U^{-1}B$	896	1792	3578	7155	14009

Level 1 BLAS : for vector operations, such as $y \leftarrow \alpha x + y$

Level 2 BLAS : for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$

Level 3 BLAS : for matrix-matrix operations, such as $C \leftarrow \alpha AB + \beta C$

Here, A, B and C are matrices, x and y are vectors, and α and β are scalars.

Block Algorithms and their Derivation

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

and equating coefficients of the j^{th} column, we obtain:

$$\begin{aligned} a_j &= U_{11}^T u_j \\ a_{jj} &= u_j^T u_j + u_{jj}^2. \end{aligned}$$

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

$$\begin{aligned} U_{11}^T u_j &= a_j \\ u_{jj}^2 &= a_{jj} - u_j^T u_j. \end{aligned}$$

Here is the body of the code of the LINPACK routine SPOFA, which implements the above 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
```

```
    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
```

- change by itself is sufficient to make big gains in performance on a number of machines
  - from 72 to 251 megaflops for a matrix of order 500 on one processor of a CRAY Y-MP
    * 81% of the peak speed of matrix-matrix multiplication on this processor

To derive a block form of Cholesky factorization, we write the defining equation in partitioned form thus:

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

Equating submatrices in the second block of columns, we obtain:

$$\begin{aligned} A_{12} &= U_{11}^T U_{12} \\ A_{22} &= U_{12}^T U_{12} + U_{22}^T U_{22}. \end{aligned}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution to the equation

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

by a call to the Level 3 BLAS routine STRSM; and then we can compute U_{22} from

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}.$$

Speed in megaflops of Cholesky factorization $A = U^T U$ for $n = 500$

Machine:	IBM 3090 VF	CRAY Y-MP	CRAY Y-MP
Number of processors:	1	1	8
j-variant: LINPACK	23	72	72
j-variant: using Level 2 BLAS	24	251	378
j-variant: using Level 3 BLAS	49	287	1225
i-variant: using Level 3 BLAS	50	290	1414

```
   DO 10 J = 1, N, NB
      JB = MIN( NB, N-J+1 )
      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
```

This code runs at 49 megaflops on a 3090, more than double the speed of the LINPACK code. On a CRAY Y-MP, the use of Level 3 BLAS squeezes a little more performance out of one processor, but makes a large improvement when using all 8 processors.

Clock Speeds of Computers in Timing Results

	Clock Speed	
CONVEX C-4640	135 MHz	7.41 ns
CRAY C90	240 MHz	4.167 ns
DEC 3000-500X Alpha	200 MHz	5.0 ns
IBM POWER2 model 590	66 MHz	15.15 ns
IBM RISC Sys/6000-550	42 MHz	23.81 ns
SGI POWER CHALLENGE	75 MHz	13.33 ns

Speed in megaflops of SGETRF/DGETRF for square matrices of order n

	No. of processors	Block size	Values of n	
			100	1000
CONVEX C-4640	1	64	274	711
CONVEX C-4640	4	64	379	2588
CRAY C90	1	128	375	863
CRAY C90	16	128	386	7412
DEC 3000-500X Alpha	1	32	53	91
IBM POWER2 model 590	1	32	110	168
IBM RISC Sys/6000-550	1	32	33	56
SGI POWER CHALLENGE	1	64	81	201
SGI POWER CHALLENGE	4	64	79	353

Speed in megaflops of SPOTRF/DPOTRF for matrices of order n with UPLO = 'U'

	No. of processors	Block size	Values of n	
			100	1000
CONVEX C-4640	1	64	120	546
CONVEX C-4640	4	64	150	1521
CRAY C90	1	128	324	859
CRAY C90	16	128	453	9902
DEC 3000-500X Alpha	1	32	37	83
IBM POWER2 model 590	1	32	102	247
IBM RISC Sys/6000-550	1	32	40	72
SGI POWER CHALLENGE	1	64	74	199
SGI POWER CHALLENGE	4	64	69	424

Speed in megaflops of DSYTRF for matrices of order n with UPLO = 'U' on an IBM POWER2 model 590

Block size	Values of n	
	100	1000
1	62	86
64	68	165

Examples of Block Algorithms in LAPACK

- traditional algorithm for QR factorization is based on the use of elementary Householder-Householder transformation - blocked form matrices of the general form

$$H = I - rvv^T$$

where v is a column vector and r is a scalar. This leads to an algorithm with very good vector performance, especially if coded to use Level 2 BLAS.

- The key to developing a block form of this algorithm is to represent a product of b elementary Householder matrices of order n as a block form of a Householder matrix.

$$H_1 H_2 \dots H_b = I - VTV^T$$

Speed in megaflops of SGEQRF/DGEQRF for square matrices of order n

	No. of processors	Block size	Values of n	
			100	1000
CONVEX C-4640	1	64	81	521
CONVEX C-4640	4	64	94	1204
CRAY C90	1	128	384	859
CRAY C90	16	128	390	7641
DEC 3000-500X Alpha	1	32	50	86
IBM POWER2 model 590	1	32	108	208
IBM RISC Sys/6000-550	1	32	30	61
SGI POWER CHALLENGE	1	64	61	190
SGI POWER CHALLENGE	4	64	39	342

Speed in megaflops of reductions to condensed forms on an IBM POWER2 model 590

(all matrices are square of order n)

	Block size	Values of n	
		100	1000
DSYTRD	1	137	159
	16	82	169
DGEBRD	1	90	110
	16	90	136
DGEHRD	1	111	113
	16	125	187

Execution time and Megaflop rates for SGEMV/DGEMV and SGEMM/DGEMM

	SGEMV/DGEMV				SGEMM/DGEMM			
	Values of $n = m = k$							
	100		1000		100		1000	
	Time	Mflops	Time	Mflops	Time	Mflops	Time	Mflops
CONVEX C-4640 (1 proc)	.00009	208	.0079	252	.0027	718	2.554	783
CONVEX C-4640 (4 proc)	.00008	229	.0032	616	.0008	2308	.659	3031
CRAY C90 (1 proc)	.00003	797	.0022	899	.0023	863	2.222	900
CRAY C90 (16 procs)	.00002	822	.0003	6385	.0002	11215	.140	14282
DEC 3000-500X Alpha	.00043	47	.0935	21	.0162	123	17.301	116
IBM POWER2 model 590	.00023	88	.0120	167	.0085	233	7.800	256
IBM RISC Sys/6000-550	.00072	28	.0500	40	.0299	67	26.631	75
SGI POWER CHALLENGE (1 proc)	.00017	116	.0494	41	.0078	256	7.452	268
SGI POWER CHALLENGE (4 procs)	.00016	125	.0065	306	.0052	386	1.889	1059

Note that the synthetic megaflop rating is much higher than the true megaflop rating for
SSYEVD/DSYEVD in Table **??**; this is because SSYEVD/DSYEVD performs many fewer floating point operations than the standard algorithm, SSYEV/DSYEV.

"Standard" floating point operation counts for LAPACK drivers for n-by-n matrices

Driver	Options	Operation Count
xGESV	1 right hand side	$.67 \cdot N^3$
xGELS	1 right hand side	$1.33 \cdot N^3$
xGEEV	eigenvalues only	$10.00 \cdot N^3$
xGEEV	eigenvalues and right eigenvectors	$26.33 \cdot N^3$
xSYEVD	eigenvalues only	$1.33 \cdot N^3$
xSYEVD	eigenvalues and eigenvectors	$9.00 \cdot N^3$
xGESVD	singular values only	$2.67 \cdot N^3$
xGESVD	singular values and left and right singular vectors	$21.00 \cdot N^3$

Performance of SGESV/DGESV for n-by-n matrices

	No. of proc.	nb	Values of n: 100 Time	100 $\frac{\text{Time}}{\text{T(MM)}}$	100 Mflops	1000 Time	1000 $\frac{\text{Time}}{\text{T(MM)}}$	1000 Mflops
CONVEX C-4640	1	64	.003	1.07	244	.948	.37	705
CONVEX C-4640	4	64	.002	2.31	317	.278	.42	2404
CRAY C90	1	128	.002	.78	377	.775	.35	863
CRAY C90	16	128	.002	10.78	351	.092	.66	7270
DEC 3000-500X Alpha	1	32	.014	.86	49	7.656	.44	87
IBM POWER2 model 590	1	32	.006	.73	109	3.970	.51	168
IBM RISC Sys/6000-550	1	32	.020	.67	34	11.840	.45	56
SGI POWER CHALLENGE	1	64	.009	1.15	80	3.371	.45	198

Performance of SGELS/DGELS for n-by-n matrices

	No. of proc.	nb	Values of n: 100 Time	100 $\frac{\text{Time}}{\text{T(MM)}}$	100 Mflops	1000 Time	1000 $\frac{\text{Time}}{\text{T(MM)}}$	1000 Mflops
CONVEX C-4640	1	64	.025	8.97	55	2.901	1.14	461
CONVEX C-4640	4	64	.022	25.37	62	1.386	2.10	965
CRAY C90	1	128	.004	1.86	318	1.568	.71	853
CRAY C90	16	128	.004	23.18	329	.186	1.33	7209
DEC 3000-500X Alpha	1	32	.038	2.34	37	15.488	.90	86
IBM POWER2 model 590	1	32	.017	1.98	82	6.630	.85	202
IBM RISC Sys/6000-550	1	32	.060	2.00	23	22.560	.85	59
SGI POWER CHALLENGE	1	64	.030	3.84	46	7.077	.95	189

Performance of SGEEV/DGEEV, eigenvalues only

	No. of proc.	nb	Values of n: 100 Time	100 $\frac{\text{Time}}{\text{T(MM)}}$	100 True Mflops	100 Synth Mflops	1000 Time	1000 $\frac{\text{Time}}{\text{T(MM)}}$	1000 True Mflops	1000 Synth Mflops
CONVEX C-4640	1	64	.320	114.9	34	31	62.081	24.3	151	161
CONVEX C-4640	4	64	.317	365.6	34	32	57.051	86.5	165	175
CRAY C90	1	128	.138	59.5	74	73	22.536	10.1	410	444
CRAY C90	16	128	.139	772.2	73	72	18.982	135.5	483	527
DEC 3000-500X ALPHA	1	32	.382	23.5	28	26	335.231	19.4	28	30
IBM POWER2 model 590	1	32	.200	23.3	54	50	128.810	16.5	73	78
IBM RISC Sys/6000-550	1	32	.640	21.3	17	16	558.190	21.0	17	18
SGI POWER CHALLENGE	1	64	.219	28.0	50	46	78.535	10.5	120	127

Performance of SGEEV/DGEEV, eigenvalues and right eigenvectors

	No. of proc.	nb	Values of n: 100 Time	100 $\frac{\text{Time}}{\text{T(MM)}}$	100 True Mflops	100 Synth Mflops	1000 Time	1000 $\frac{\text{Time}}{\text{T(MM)}}$	1000 True Mflops	1000 Synth Mflops
CONVEX C-4640	1	64	.465	166.9	58	57	139.423	54.6	170	189
CONVEX C-4640	4	64	.460	530.6	59	57	130.542	197.8	183	202
CRAY C90	1	128	.181	78.0	137	146	42.416	19.1	550	621
CRAY C90	16	128	.181	1005.6	137	146	35.802	255.7	641	736
DEC 3000-500X Alpha	1	32	.847	52.1	32	31	920.321	53.2	26	29
IBM POWER2 model 590	1	32	.350	40.8	77	75	462.110	59.2	52	57
IBM RISC Sys/6000-550	1	32	1.220	40.7	22	22	1404.250	52.7	17	19
SGI Power Challenge	1	64	.368	47.1	73	72	262.583	35.2	91	100

Performance of SSYEVD/DSYEVD, eigenvalues only, UPLO='U'

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
CONVEX C-4640	1	64	.044	15.79	39	30	7.833	3.07	173	170
CONVEX C-4640	4	64	.039	44.98	45	35	4.228	6.41	320	315
CRAY C90	1	128	.013	5.73	116	100	2.237	1.01	604	596
CRAY C90	16	128	.013	73.00	115	100	.853	6.09	1585	1563
DEC 3000-500X Alpha	1	32	.057	3.51	27	24	38.117	2.20	36	35
IBM POWER2 model 590	1	32	.026	3.11	72	50	8.460	1.08	160	158
IBM RISC/6000-550	1	32	.080	2.67	19	17	27.300	1.03	50	49
SGI POWER CHALLENGE	1	64	.042	5.37	41	32	12.530	1.68	108	106

Performance of SSYEVD/DSYEVD, eigenvalues and eigenvectors, UPLO='U'

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
CONVEX C-4640	1	64	.086	30.9	67	104	12.544	4.9	357	718
CONVEX C-4640	4	64	.077	88.8	75	117	5.923	8.9	755	1520
CRAY C90	1	128	.044	18.9	133	205	6.586	2.9	682	1367
CRAY C90	16	128	.042	231.9	139	216	1.691	12.1	2657	5322
DEC 3000-500X Alpha	1	32	.167	10.3	35	54	74.876	4.3	60	120
IBM POWER2 model 590	1	32	.079	9.3	72	112	21.950	2.8	174	410
IBM RISC Sys/6000-550	1	32	.250	8.3	23	36	71.660	2.7	53	126
SGI POWER CHALLENGE	1	64	.099	12.7	59	91	29.220	3.9	154	308

Performance of SGESVD/DGESVD, singular values only

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
CONVEX C-4640	1	64	.059	21.2	49	45	18.775	7.3	143	142
CONVEX C-4640	4	64	.051	58.8	56	53	7.547	11.4	355	353
CRAY C90	1	128	.023	9.9	124	116	4.146	1.9	647	643
CRAY C90	16	128	.023	125.5	126	118	1.385	9.9	1936	1925
DEC 3000-500X Alpha	1	32	.088	5.4	32	30	81.281	4.7	33	33
IBM POWER2 model 590	1	32	.050	5.8	57	53	19.060	2.4	141	140
IBM RISC Sys/6000-550	1	32	.140	4.7	20	19	64.830	2.4	41	41
SGI POWER CHALLENGE	1	64	.067	8.6	43	40	34.569	4.6	78	77

Performance of SGESVD/DGESVD, singular values and left and right singular vectors

	No. of proc.	nb	Values of n							
			100				1000			
			Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops	Time	$\frac{\text{Time}}{\text{T(MM)}}$	True Mflops	Synth Mflops
CONVEX C-4640	1	64	.202	72.5	85	104	101.08	39.6	151	208
CONVEX C-4640	4	64	.186	214.5	92	113	78.19	118.5	195	269
CRAY C90	1	128	.103	44.4	135	203	29.78	13.4	409	705
CRAY C90	16	128	.104	577.8	134	202	24.17	172.6	504	869
DEC 3000-500X Alpha	1	32	.475	29.2	36	44	781.53	45.2	20	27
IBM POWER2 model 590	1	32	.190	22.2	90	111	664.05	85.1	23	32
IBM RISC Sys/6000-550	1	32	.720	24.0	24	29	1467.83	55.1	10	14
SGI POWER CHALLENGE	1	64	.216	27.6	67	97	238.22	32.0	53	88

History of block algorithms for matrix problems.

- Early algorithms involved use of a small main memory, using tape as secondary storage:

  (Barron and Swinnerton-Dyer, 1960)
  (Charles, 1960)
  (McKellar and Coffman, 1969),

- Recent work centers on use of vector register, cache, local memory, main memory or solid-state disks:

  (Berry et al., 1986)
  (Bischof and Van Loan, 1987)
  (Bucher and Jordan, 1984)
  (Calahan, 1986)
  (Dongarra and Hewitt, 1985)
  (Dongarra and Sorensen, 1986)
  (Gallivan, Jalbey, Meier, 1987)
  (IBM, 1986)
  (Robert and Sguazzero, 1987)
  (Schreiber, 1986)

LAPACK / ScaLapack A Scalable Library for Distributed Memory Machines

- **LAPACK success (from HP-48G to CRAY C-90)**
  - **Targets workstations, vector machines, shared memory**
  - **Initial release in April 1992**
  - **336,095 requests since then (as of 4/95)**
  - **Dense and banded matrices**
  - **Linear systems, least squares, eigenproblems**
  - **Manual available from SIAM**
  - **Being translated into Japanese and Russian**
  - **Second release and manual in 9/94**
- **Provide useful scientific program libraries for computational scientists working on new parallel architectures**
  - **Move to new generation high performance machines - SP-1, 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

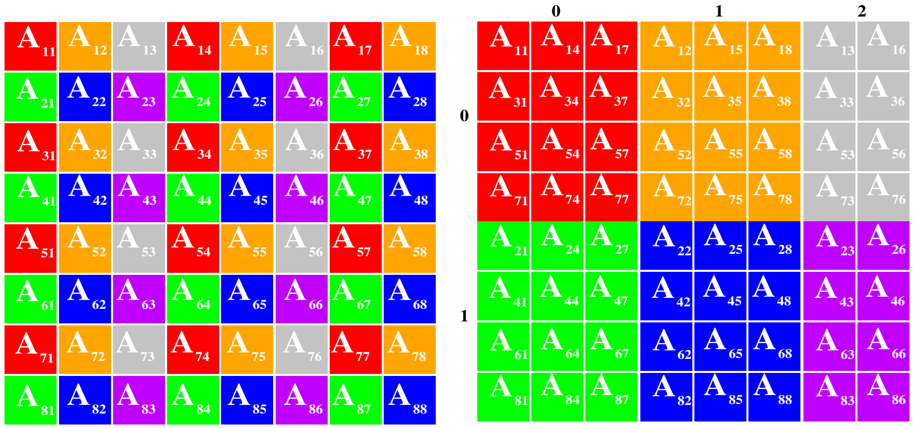
- Standards
  - Portable
  - Vendor-supported
  - Scalable
- High efficiency
- Protect programming investment, if possible
- Avoid low-level details
- Scalability

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
- **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.

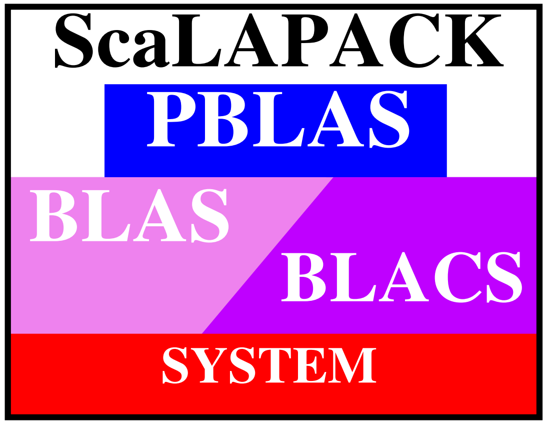
Communication Library for Linear Algebra

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.

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.

ScaLAPACK IMPLEMENTATION

PBLAS (Ease of Use and programming)

- **Array descriptor**: rigorous definition of distributed matrices,
- PBLAS use global view of the matrix operands,
- Extended subset of the BLAS allowing global addressing of distributed matrices avoiding complex local indexing,
- **Code reusability**, interface very close to sequential BLAS:

```
 CALL  DGEXXX( M, N, A( I, J ), 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 )
```

$\downarrow$

```
 CALL PDGEXXX( M, N, A, I, J, 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 )
```


Parallel Level 2 and 3 BLAS: PBLAS

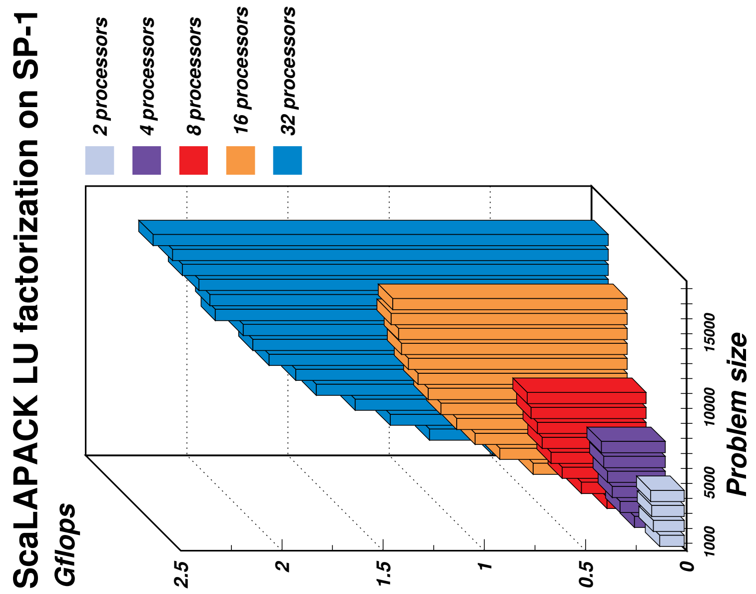
Goal

- 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 $\rightarrow$ ScaLAPACK

BLAS $\rightarrow$ PBLAS (BLAS, BLACS)

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



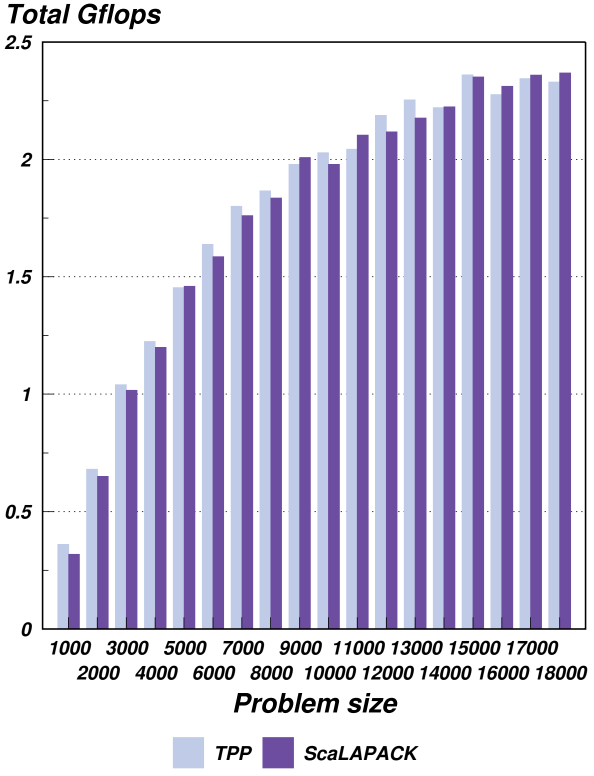
SEQUENTIAL LU FACTORIZATION CODE

```
   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
```

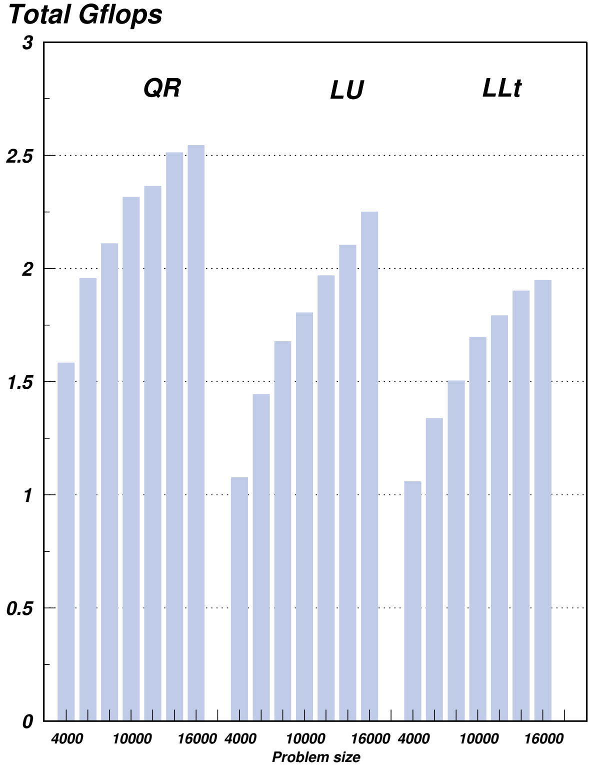
PARALLEL LU FACTORIZATION CODE

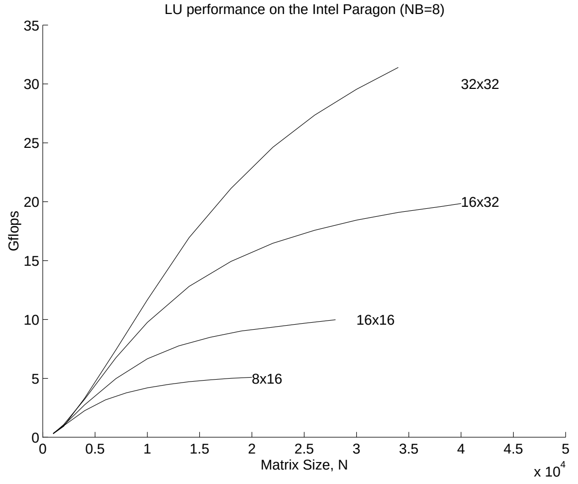
```
   DO 10 J = JA, JA+MIN(M,N)-1, ADESC(4)
      JB = MIN( MIN(M,N)-J+JA, ADESC(4) )

      Factor diagonal and subdiagonal blocks and test for exact
      singularity.

      CALL PDGETF2( M-J+JA, JB, A, IA+J-JA, J, ADESC, IPIV, INFO )

      Apply interchanges to columns JA:J-JA.

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

         Compute block row of U.

         CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
  $                   N-J-JB+JA, ONE, A, IA+J-JA, J, ADESC, A,
  $                   IA+J-JA, J+JB, ADESC )
         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, IA+J+JB-JA, J,
  $                      ADESC, A, IA+J-JA, J+JB, ADESC, ONE,
  $                      A, IA+J+JB-JA, J+JB, ADESC )
         END IF
      END IF
10 CONTINUE
```

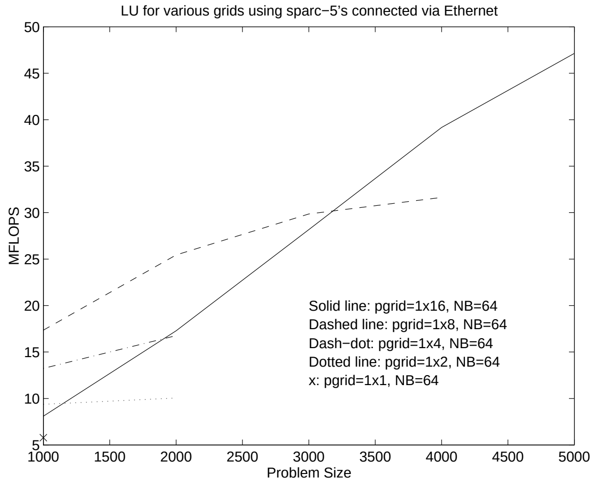
TPP vs. ScaLAPACK on 32 nodes SP1

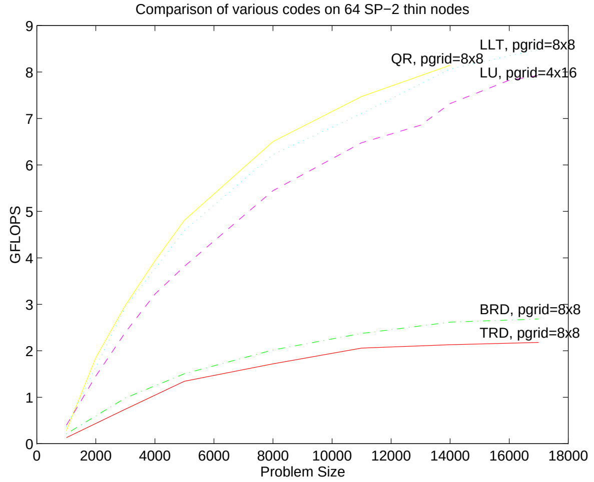


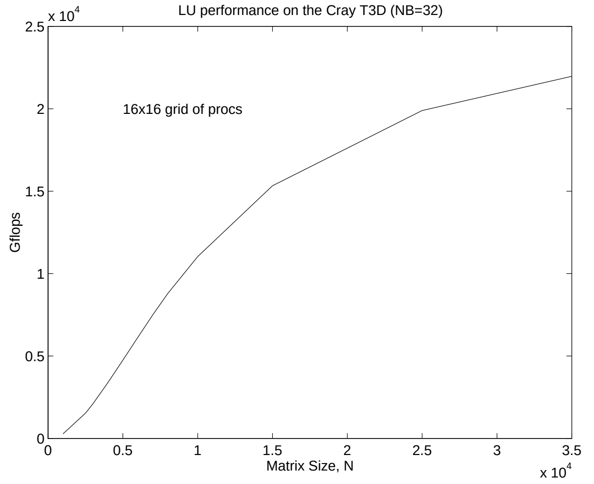
ScaLAPACK on 32 nodes SP1



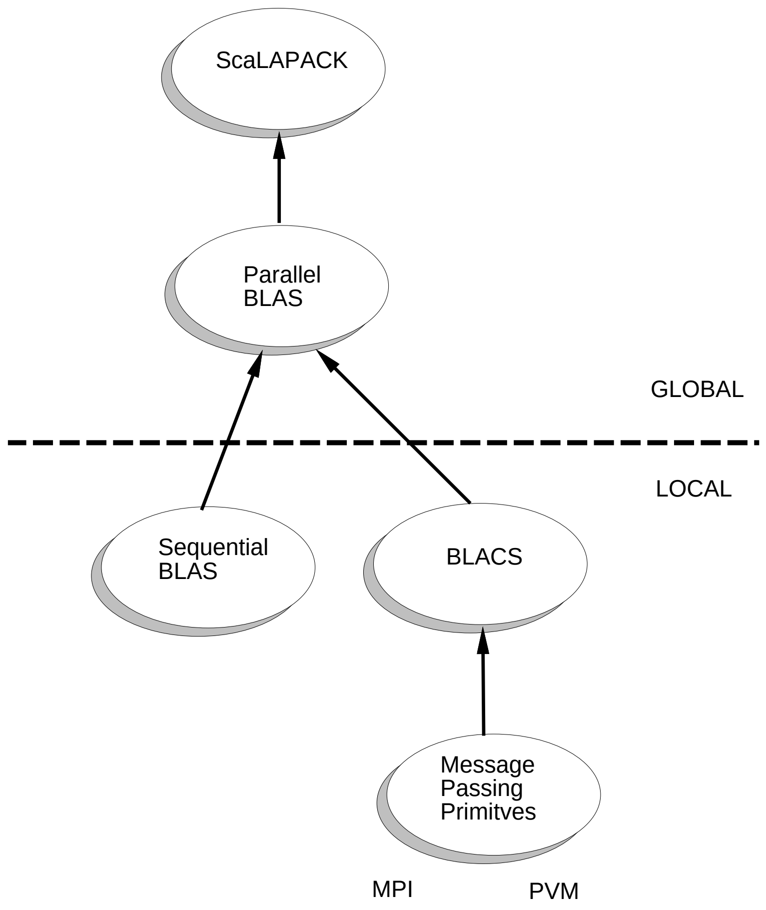

LU performance on the Intel Paragon (NB=8)
Gflops
Matrix Size, N
x 10^4
32x32
16x32
16x16
8x16


LU for various grids using sparc−5's connected via Ethernet
MFLOPS
Problem Size
Solid line: pgrid=1x16, NB=64
Dashed line: pgrid=1x8, NB=64
Dash−dot: pgrid=1x4, NB=64
Dotted line: pgrid=1x2, NB=64
x: pgrid=1x1, NB=64


Comparison of various codes on 64 SP−2 thin nodes
GFLOPS
Problem Size
LLT, pgrid=8x8
QR, pgrid=8x8
LU, pgrid=4x16
BRD, pgrid=8x8
TRD, pgrid=8x8


LU performance on the Cray T3D (NB=32)
x 10^4
Gflops
16x16 grid of procs
Matrix Size, N
x 10^4

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.

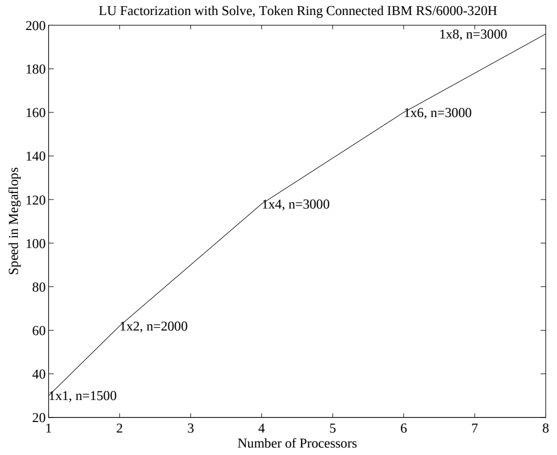
BLACS (Basic Linear Algebra Communication Subroutines) is a software package to perform communication at the same level as the BLAS are for computation.

- **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 $\rightarrow$ 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

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**

Different users with different needs.

The traditional library users, who make 500 requests/day to netlib and keep NAG and IMSL in business, have the following priorities:

- **Easy interface with hidden details**
- **Reliability; the code should fail as rarely as possible**
- **Speed**

The HPCC community, on the other hand, which wants to solve the largest, hardest problems as quickly as possible, seems to want

- **Speed**
- **Access to internal details to tune data structures to the application**
- **Algorithms which are fast for the particular application even if not reliable as general methods**

Differing priorities
Different approaches to algorithms and software.

WHY ITERATIVE METHODS?

When direct methods too expensive

When multigrid not feasable

— iterative methods may do —

ADVANTAGE OF ITERATIVE METHODS:

modest in time per step

modest in memory requirements

DISADVANTAGE: not robust

ILLUSTRATION

Finite element space model (Simon '89)

Realistic model: $n = 5 \star 10^9$

Direct Solver at 1 TFLOP: $520,040$ years

(exploitable parallelism)

Conjugate Gradients: $\sim$ 500 iteration steps

With 1 TFLOP speed: 30 minutes

(parallelism exploitable?)

ITERATIVE METHODS - BACKGROUND

$Ax = b,$ write: $A = I - (I - A)$

$$(I - (I - A))\, x = b$$

$$\Downarrow$$

BASIC ITERATION METHOD:

$$\begin{aligned} x_i &= b + (I - A)x_{i-1} \\ &= x_{i-1} + r_{i-1} \\ &= x_{i-2} + r_{i-2} + r_{i-1} \end{aligned}$$

BASIC ITERATION METHOD

$$x_i = x_0 + r_0 + r_1 + \cdots + r_{i-1}$$

from $x_i = x_{i-1} + r_{i-1}$

$A\times \Rightarrow Ax_i = Ax_{i-1} + Ar_{i-1}$

$$\begin{aligned} b- \quad \Rightarrow \quad \underbrace{b - Ax_i}_{r_i} &= \underbrace{b - Ax_{i-1}} - Ar_{i-1} \\ &= r_{i-1} - Ar_{i-1} \\ &= (I-A)r_{i-1} \end{aligned}$$

$$\begin{aligned} x_i &= x_0 + r_0 + (I-A)r_0 + \cdots + (I-A)^{i-1}r_0 \\ &= x_0 + \left[r_0, Ar_0, A^2r_0, \ldots, A^{i-1}r_0\right] \end{aligned}$$

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**

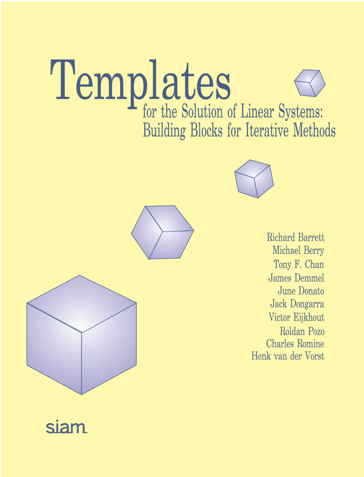
Need good design to choose proper objects and abstractions to balance

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

Motivation from Iterative Methods

Algorithm sketches for

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



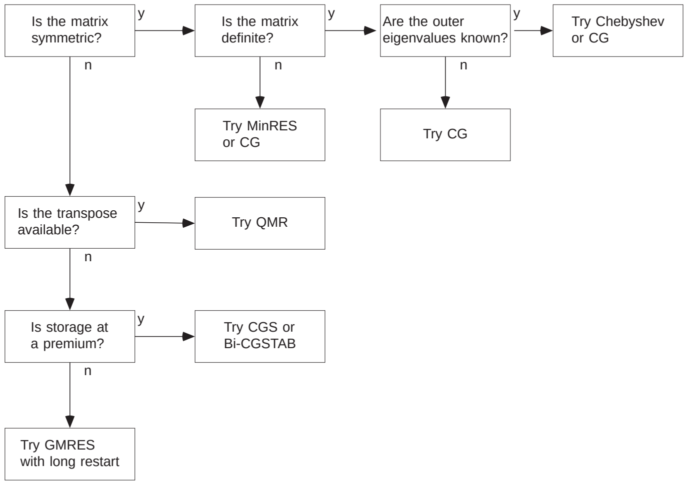
Templates for Sparse Matrix Computations

- **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**

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

ii



Flowchart with Suggestions for the Selection of Iterative Methods

$x^{(0)}$ 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

Top of Eigentemplates Decision Tree

- **Problem type:**
  - $\boxed{A - \lambda I,\ A = A^*}$
    - $A = H^*H$, H **given (point to SVD).**
    - $\boxed{\textit{Other (}\text{Symmetric Subtree}\textit{).}}$
  - **SVD of** A
  - $A - \lambda I$, **general** A
  - $A - \lambda B$, $A = A^*, B = B^*$ **and positive definite**
  - $A - \lambda B$, A **and** B **square**
  - **Other**
  - **Don't know**

Symmetric Subtree

- **Desired Eigenvalue Information:**
  - **All eigenvalues.**
  - $\boxed{\textit{Some eigenvalues}}$**:**
    - * **Compute** λ_i **through** λ_j **where** $\lambda_1 \leq \cdots \leq \lambda_n$
    - * $\boxed{\textit{Eigenvalues in } [\alpha, \beta].}$
    - * **Other.**

- **Desired Eigenvector Information Desired?:**
  - $\boxed{\textit{Yes.}}$
  - **No**

- **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.**

Yes, we can please everyone!

- We propose that
  - NHSE - National High Performance Software Exchange
  - GAMS - Guide to Available Math Software
  - Netlib
  - Templates
  - NA textbooks
  - Annotated bibliographies
  - Computing environments á la Matlab or Maple
  - On-line compute servers (comp-u-bots) for bigger problems

  should all be part of one seamless whole.

- We propose an E-book containing
  - Linearly ordered text
  - Multiple access modes via hypertext
    - Table of contents of book
    - Index of book
    - A decision-tree querying the user to pick the right algorithm
    - Pointers to other trees (netlib, GAMS)
  - Multiple levels of algorithmic description
    - "Template" in pseudo-code, further pointers to common notation, etc.
    - One-page description of algorithm
    - Matlab implementation
    - Fortran, C, C++ implementations
    - Detailed text (references to literature) ("Numerical Linear Algebra", Demmel)
    - Pointers to literature
- C++ implementation of LinSolve or EigSolve incorporating parts of decision tree (Pozo)
- Open architecture for future growth
- Hard-nosed editing for most reliable parts

Future Work

- **Publically distribute detailed outline**
  Superset of likely content
- **Get (potential) user feedback**
  (workshop planned)
- **Keep linear version of book $\leq$ 200 pages**
  (not all material)
- **Q: Will a good design put NA faculty out of business?**
  A: Better we do it than someone else.

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

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

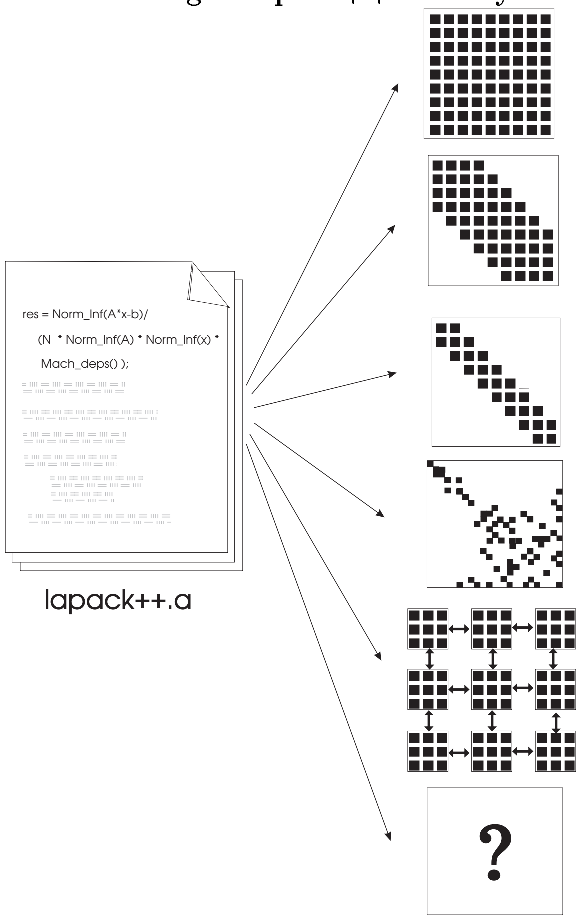
F77 LAPACK / BLAS

```
        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
$\begin{pmatrix} a_{00} & a_{01} & & & \\ a_{10} & a_{11} & a_{12} & & \\ a_{20} & a_{21} & a_{22} & a_{23} & \\ & a_{31} & a_{32} & a_{33} & a_{34} \\ & & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} * & a_{01} & a_{12} & a_{23} & a_{34} \\ a_{00} & a_{11} & a_{22} & a_{33} & a_{44} \\ a_{10} & a_{21} & a_{32} & a_{43} & * \\ a_{20} & a_{31} & a_{42} & * & * \end{matrix}$

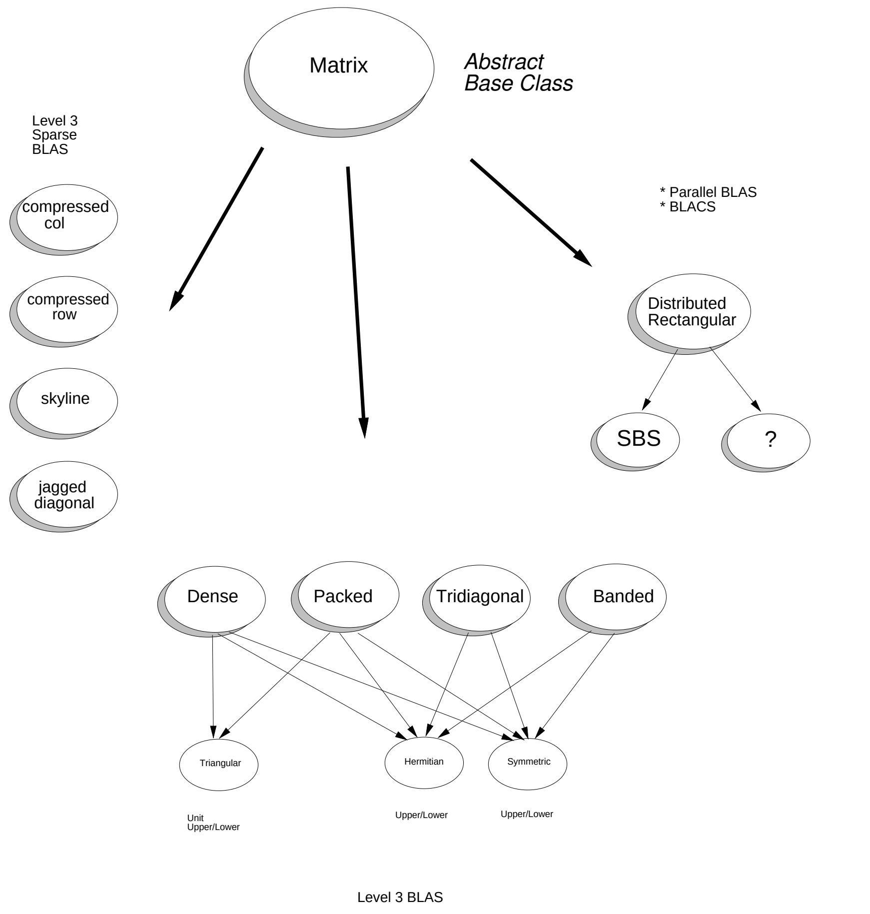
Triangular Packed

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`	$\begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\underbrace{a_{00}\, a_{10}\, a_{20}\, a_{30}}\ \underbrace{a_{11}\, a_{21}\, a_{31}}\ \underbrace{a_{22}\, a_{32}}\ a_{33}$

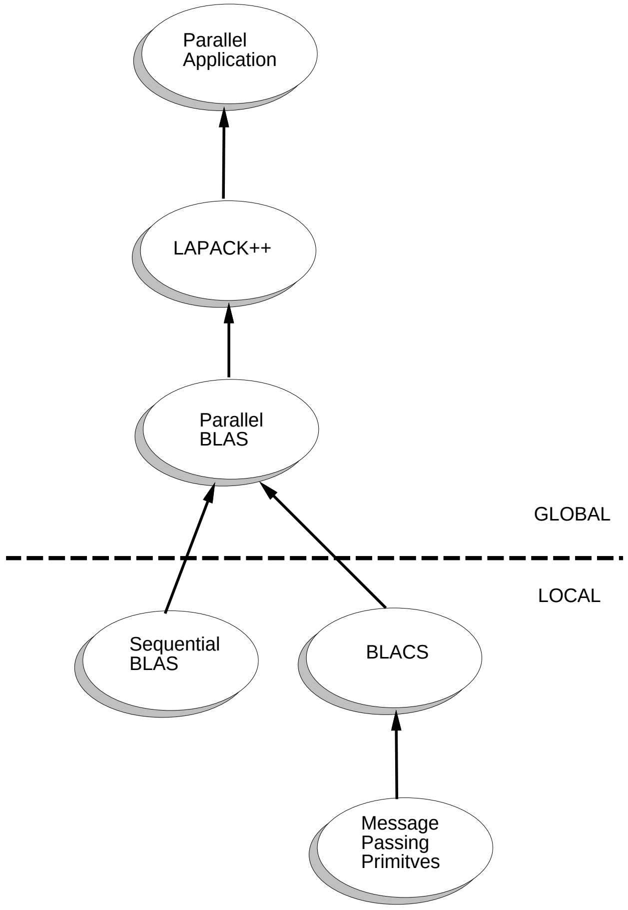
Orthogonal

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

Linear Algebra Classes



Object Oriented Software Framework

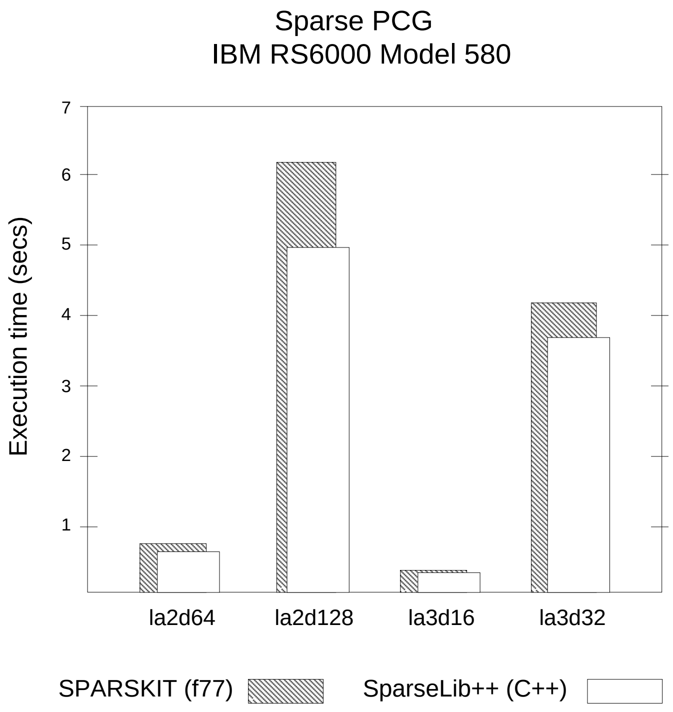


Iterative Methods Meta-Library (IML++)

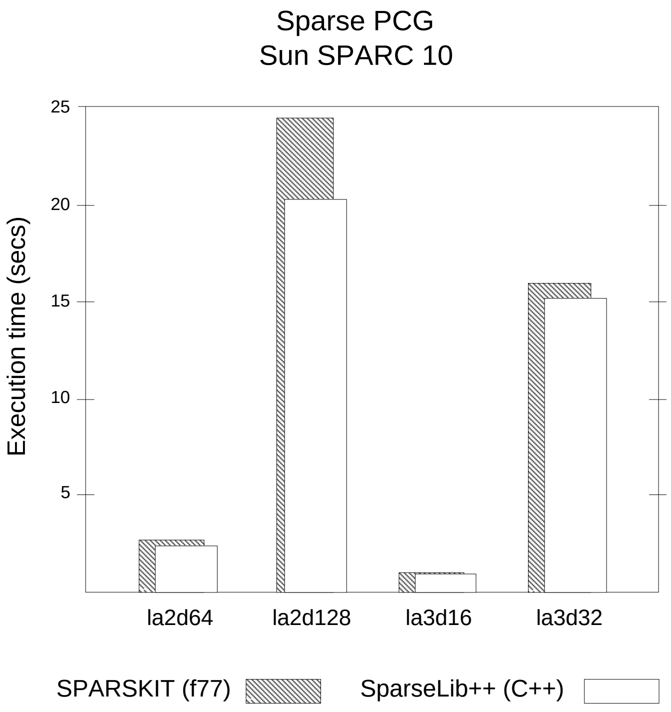
- **Stationary Methods (SOR)**
- **Conjugate Gradient (CG)**
- **Generalized Minimal Residual (GMRES)**
- **Minimum Residual (MINRES)**
- **Quasi-Minimal Residual (QMR)**
- **Chebyshev Iteration (Cheb)**
- **Conjugate Gradient Squared (CGS)**
- **Biconjugate Gradient (BiCG)**
- **Biconjugate Gradient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

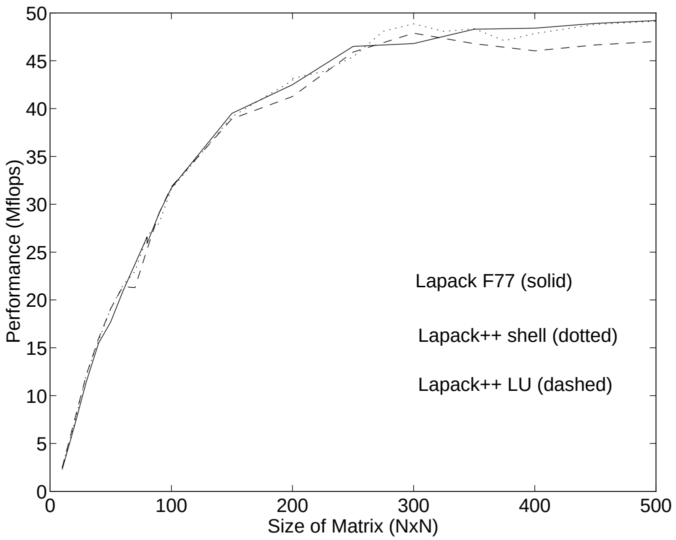


PCG using compressed row matrices; optimized Fortran (xlf -O) Sparskit and C++ (xlC -O) SparseLib++ (Sparse Blas).



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.

LAPACK
ScaLAPACK
For High Performance Machines

Tony Chan	UCLA
James Demmel	U. C. Berkeley
Jack Dongarra	U. of Tennessee and ORNL
Mike Heath	U. of Illinois
Roldan Pozo	U. of Tennessee
Danny Sorensen	Rice U.
David Walker	ORNL

E. Anderson	Z. Bai
C. Bischof	J. Du Croz
A. Greenbaum	S. Hammarling
A. McKenney	S. Ostrouchov

References:

- Mosaic URL:
  - http://www.netlib.org
  - http://www.netlib.org/lapack
  - http://www.netlib.org/scalapack
  - http://www.netlib.org/lapack/lawns
  - http://www.netlib.org/linalg/html_templates/Templates.html
  - http://www.netlib.org/c++/lapack++
- anonymous ftp netlib2.cs.utk.edu
  - cd lapack; get index
  - cd scalapack; get index
  - cd lapack/lawns; get index
  - cd linalg; get index
  - cd c++; get index