High Performance Sparse Gaussian Elimination
Algorithms and Software

Xiaoye Sherry Li
xiaoye@nersc.gov
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

Joint work with
Jim Demmel (UC Berkeley) and John Gilbert (Xerox PARC)

Outline

- Background of the sparse direct solvers.
- Supernodal GEPP – blocking for locality.
  - On cache-based workstations
  - On shared memory multiprocessors
- Results on several machines.
  - IBM RS/6000-590 (at UCB)
  - DEC AlphaServer 8400 (at LLNL)
  - Cray C90 & J90 (at NERSC)
  - SGI Power Challenge (with help of Ed Rothberg then at SGI)
  - SGI/Cray Origin2000 (at NCSA)
- Distributed memory algorithms.
- Available software.

Direct Solvers for Sparse Linear Systems

- Gaussian elimination (LU, LL^T, LDL^T factorizations), followed by lower/upper triangular solves.
  - Dense: $PA = LU$ — permutation for stability
  - Sparse: $P_r A P_c^T = LU$ — permutations for stability and sparsity of L, U

- **Distinct steps for sparse matrices.**
  1. Order equations $\&$ variables to minimize fill in the L, U factors
     heuristics based on combinatorics
  2. Symbolic factorization
     set up data structures and allocate memory for L, U
  3. Numerical factorization
     usually dominates total time
  4. Triangular solves
     usually $< 5\%$ time

Ordering for Sparse Cholesky

- Local greedy heuristics.
  - Minimum degree [Tinney/Walker '67, George/Liu '79, Liu '85, Amestoy/Davis/Duff '94, Ashcraft '95, Duff/Reid '95]
  - Minimum deficiency (fill-in) [Tinney/Walker '67, Ng/Raghavan '97]

- Global graph partitioning heuristics.
  - Nested dissection [George '73]
  - Multilevel schemes [Hendrickson/Leland '94, Karypis/Kumar '95]
  - Spectral bisection [Simon *et al.* '90-'95]
  - Geometric and spectral bisection [Chan/Gilbert/Teng '94]

- Hybrid of the above two [Ashcraft/Liu '96, Hendrickson/Rothberg'97].

Ordering for Sparse LU

- Symmetric ordering for Cholesky of $A^T A$.
  - If $R^T R = A^T A$ and $PA = LU$, then for any P, the nonzero structure of $L + U$ is contained in the nonzero structure of $R^T + R$. [George/Ng '87]
  - Making R sparse tends to make $L + U$ sparse.
  - Strategy:
    1. Find good symmetric ordering P_c from $A^T A$
    2. Apply P_c to columns of A

$$(AP_c^T)^T (AP_c^T) = P_c (A^T A) P_c^T \ .$$

- Column minimum degree based solely on A.

  [Matlab, Davis/Gilbert/Ng/Peyton]

- Markowitz – unsymmetric variant of minimum degree.

  [Duff/Erisman/Reid '86 book]

  usually performed together with numeric factorization.

Symbolic Factorization

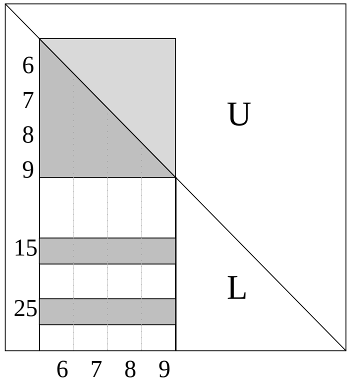
- Cholesky [George/Liu '81 book]
  - Use elimination graph of L and its transitive reduction (elimination tree)
  - Complexity linear in output $O(nnz(L))$
- LU
  - Use elimination graphs of L, U and their transitive reductions (elimination DAGs) [Tarjan/Rose '78, Gilbert/Liu '93, Gilbert '94]
  - Improved by symmetric structure pruning [Eisenstat/Liu '92]
  - Improved by supernodes
  - Complexity greater than $nnz(L+U)$, yet much smaller than $flops(LU)$

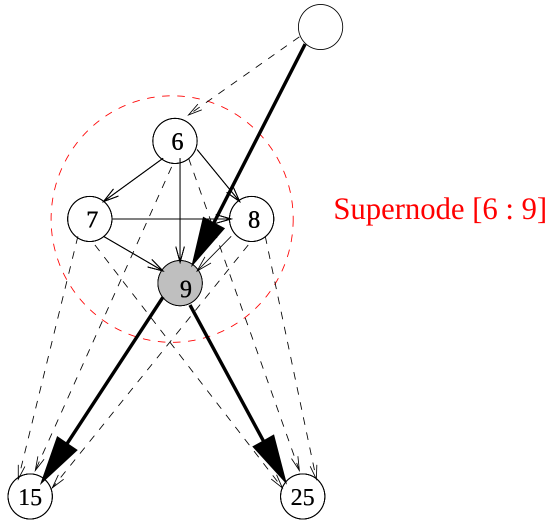
Numerical Factorization

- Usually the most expensive step
- Many existing codes
  - Okay for small problems, slow for large ones.
  - Do not work well on new architectures:
    - hierarchical memory system
    - superscalar processor
    - multiple processors
- Our aims of this research
  - Study state-of-the-art algorithms and architectures
  - Develop library quality softwre and transfer technologies to users

Unsymmetric Supernode [Eisenstat/Gilbert/Liu '93]

- Exploit dense submatrices in the L & U factors of $PA = LU$.





- Why are supernodes good?
  - Permit use of higher level BLAS.
  - Reduce inefficient indirect addressing (scatter/gather).
  - Reduce symbolic time by traversing a coarser graph.

Supernode-Panel factorization

for column $j = 1$ **to** n **step** w **do**

$F(:, j{:}j + w - 1) = A(:, j{:}j + w - 1)$;

(1) Symbolic factorization [Gilbert/Peierls '88, Gilbert/Li '94]

Determine which supernodes update $F(:, j{:}j + w - 1)$;

(2) Numeric update

for each updating supernode $(r{:}s) < j$ in order **do**

- Triangular solve

  $U(r : s, j : j + w - 1) =$

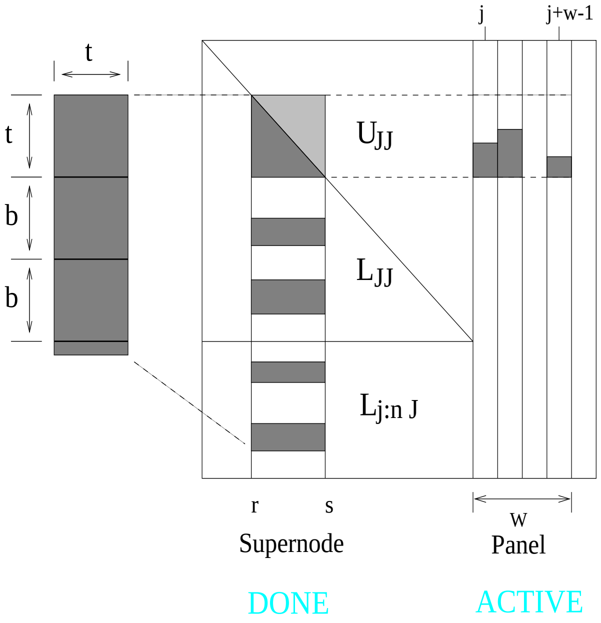
  $\quad L(r : s, r : s) \backslash F(r : s, j : j + w - 1)$;

- Matrix update

  $F(s + 1 : n, j : j + w - 1) -=$

  $\quad L(s + 1 : n, r : s) * U(r : s, j : j + w - 1)$;

end for;



(3) Inner factorization for $F(j : n, j + w - 1)$

- Row pivoting for each column;
- Detect supernode boundary;
- Symmetric structure pruning; [Eisenstat/Liu '92]

end for;

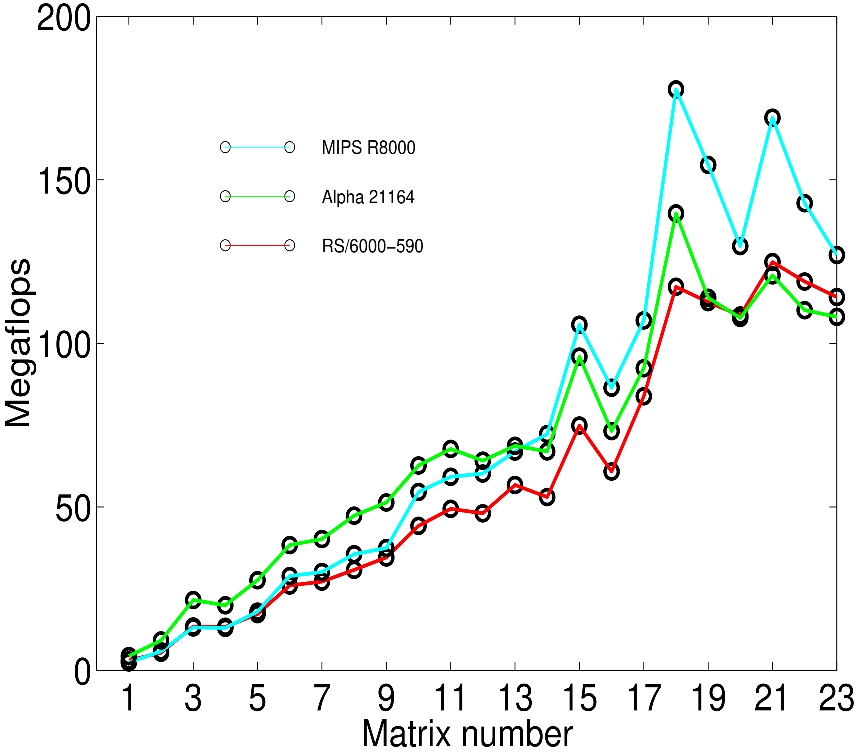
Test Matrices

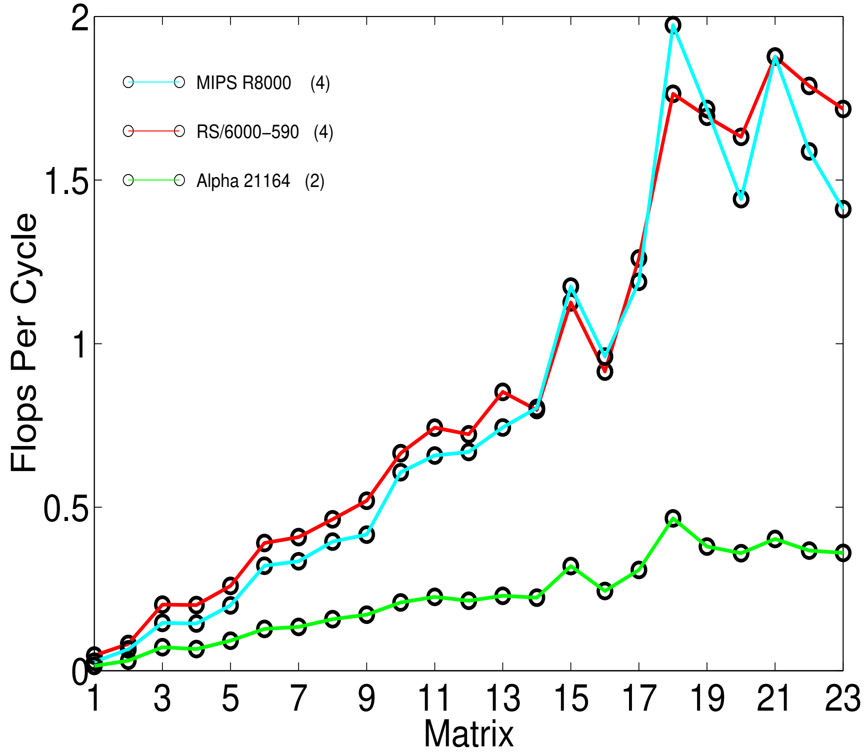
	Name	N	$nnz(A)$	$\frac{nnz(A)}{N}$	$\frac{nnz(F)}{nnz(A)}$	#flops (10^6)	$\frac{\#flops}{nnz(F)}$
1	MEMPLUS	17758	99147	5.6	1.4	1.8	12.5
2	GEMAT11	4929	33108	6.7	2.8	1.5	16.3
3	RDIST1	4134	94408	22.8	3.6	12.9	38.1
4	ORANI678	2529	90158	35.6	3.1	14.9	53.3
5	MCFE	765	24382	31.8	2.8	4.1	59.9
6	LNSP3937	3937	25407	6.5	16.8	38.9	91.1
7	LNS3937	3937	25407	6.5	17.7	44.8	99.7
8	SHERMAN5	3312	20793	6.3	12.0	25.2	101.3
9	JPWH991	991	6027	6.1	23.4	18.0	127.7
10	SHERMAN3	5005	20033	4.0	21.6	60.6	139.8
11	ORSREG1	2205	14133	6.4	28.5	59.8	148.6
12	SAYLR4	3564	22316	6.3	29.3	104.8	160.0
13	SHYY161	76480	329762	4.3	23.2	1571.6	205.8
14	GOODWIN	7320	324772	44.4	9.6	665.1	213.9
15	VENKAT01	62424	1717792	27.5	7.6	3219.9	247.9
16	INACCURA	16146	1015156	62.9	9.8	4118.7	414.3
17	AF23560	23560	460598	19.6	30.4	6363.7	454.9
18	DENSE1000	1000	1000000	1000	1.0	666.2	666.2
19	RAEFSKY3	21200	1488768	70.2	11.8	12118.7	690.7
20	EX11	16614	1096948	66.0	23.8	26814.5	1023.1
21	WANG3	26064	177168	6.8	74.9	14557.5	1095.5
22	RAEFSKY4	19779	1316789	66.6	20.3	31283.4	1172.6
23	VAVASIS3	41029	1683902	41.0	29.2	89209.3	1813.5

- Sorted in increasing order of $Reuse\_ratio = \frac{\#flops}{nnz(F)}$.
- Columns are preordered by Matlab's minimum degree algorithm.

Uniprocessor Factorization Rates

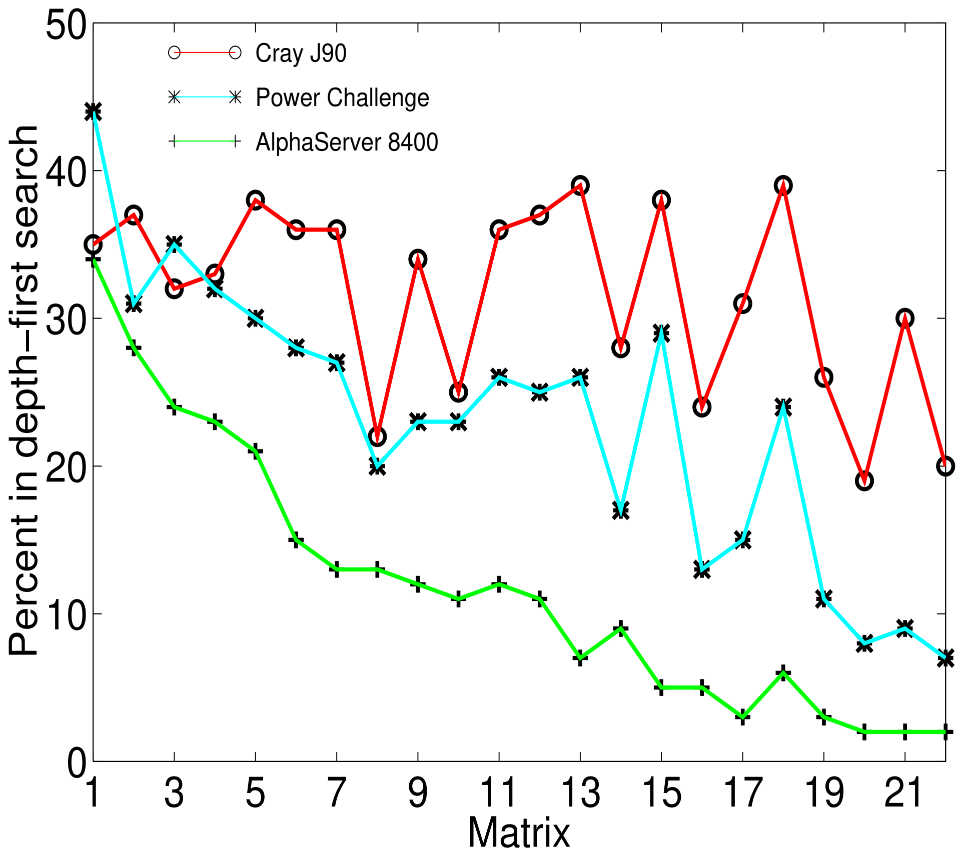
- Time includes everything except column preordering.





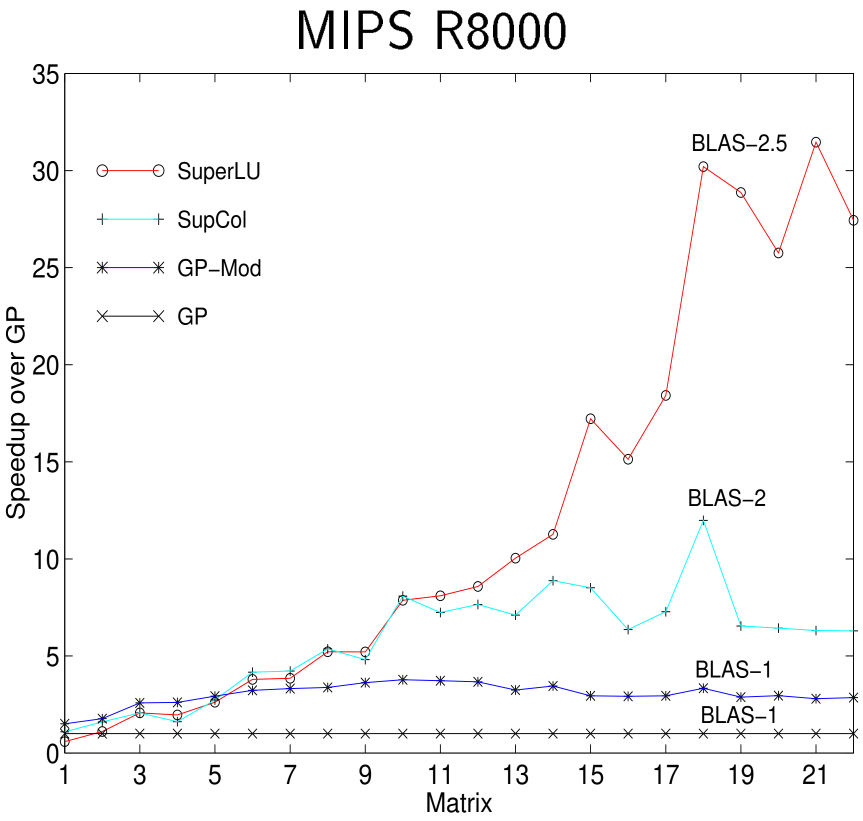
Fraction of Time in Symbolic Steps on 1-CPU

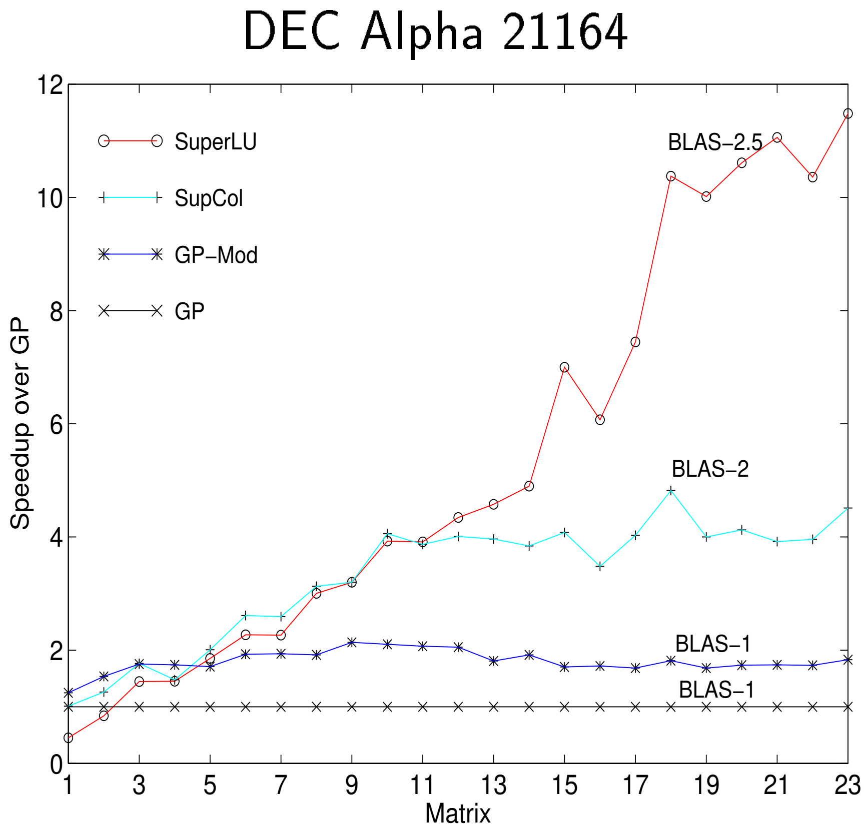
- Show relative strength of integer vs. floating-point speed
- Roughly carry over to shared memory parallel code



Improvement Over Earlier *LU* Codes

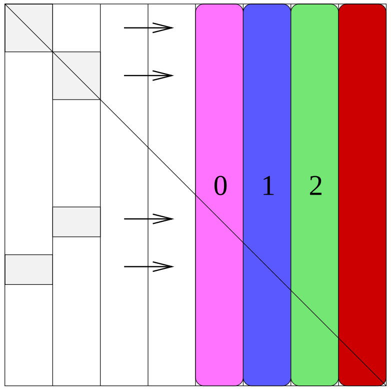
- **GP** (Gilbert/Peierls '88, in Matlab)
- **GP-mod** (Eisenstat/Liu '92, in Matlab 5.0)
- **SupCol** (Eisenstat/Gilbert/Liu '93)
- **SuperLU** (Demmel/Gilbert/Li '95)





Shared Memory SuperLU [Li '96, Demmel/Gilbert/Li '97]

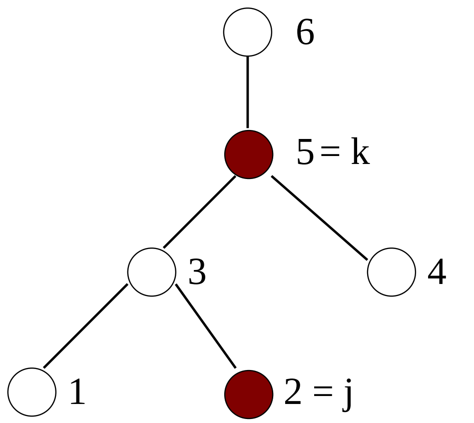
- One panel factorized by one processor; pipeline execution of several panels.
- Central scheduler dynamically assigns panel tasks to processors.
- Retain uniprocessor optimizations:
  - Symmetric structural pruning
  - "BLAS 2.5" kernel
- Highly portable – include POSIX threads interface
  - Cray C90 & J90
  - DEC Alpha servers
  - SGI Power Challenge
  - SGI/Cray Origin 2000
  - Sun Ultra Enterprise servers



Column Elimination Tree [Gilbert/Ng '93]

- Each column of the matrix has one vertex in the tree.
- Exhibits column dependencies during the elimination.
  1. If column j updates column k, then j is a descendant of k;
  2. Conversely, if j is a descendant of k, column j **may** or **may not** update column k (depending on numerical pivoting).
- More accurate update edges are detected on the fly.
- Computing elimination tree takes time almost linear in $nnz(A)$.

$$A = \begin{pmatrix} 1 & & \bullet & & \bullet & \bullet \\ & 2 & \bullet & & \bullet & \\ & \bullet & 3 & & & \\ \bullet & & & 4 & \bullet & \\ & & & & 5 & \bullet \\ \bullet & & & \bullet & & 6 \end{pmatrix}$$



The Parallel Scheduling Loop [after Gilbert '88]

Each processor ... (SPMD paradigm)

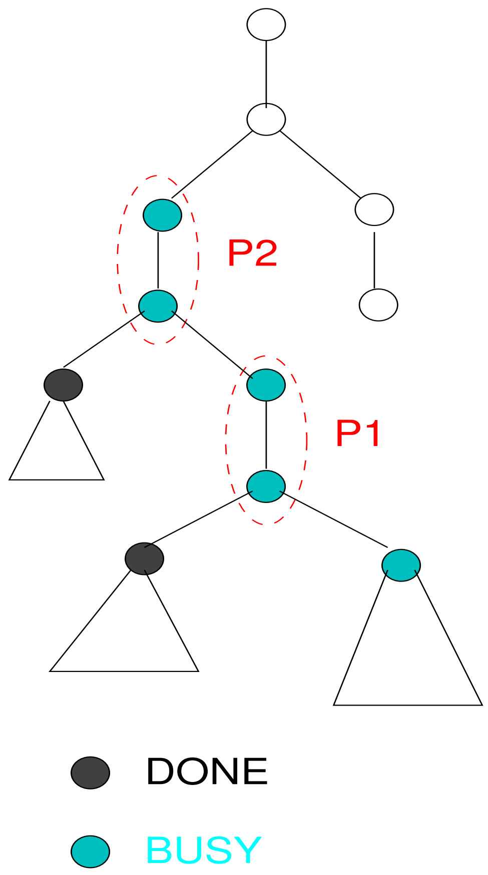
Shared task queue initialized with leaves;

while (there are more panels) do
    panel := GetTask (queue);

    (1) panel_symbolic_factor(panel);
        • skip all BUSY descendant supernodes;

    (2) panel_numeric_factor(panel);
        • updates from all DONE supernodes;
        • wait for BUSY supernodes to become DONE;

    (3) inner factorization(panel);
end while;



Some Details of the Parallel Algorithm

- Significant algorithmic changes in order to make parallelism easier.
- Parallelism from
  - Independent subtrees of column etree
  - Pipelining: may start work on parent before all descendants finish
  - Panels must be limited to chains in etree
- Priorities of shared task queue
  1. Assign parent if processor just finished last child, else
  2. Assign panels without busy children, else
  3. Assign node with unfinished children
- Avoid lock contention in pruning $G(L)$ and traversing $G(L)$ for symbolic factorization.
- Allocate contiguous memory for supernodes to enable BLAS
  - Use fundamental, not maximal, supernodes
  - Use (conservative) nonzero structure of Householder matrix in $A = QR$
  - Usually $< 15\%$ overhead in a dynamic scheme

Parallel Machines

- Some characteristics

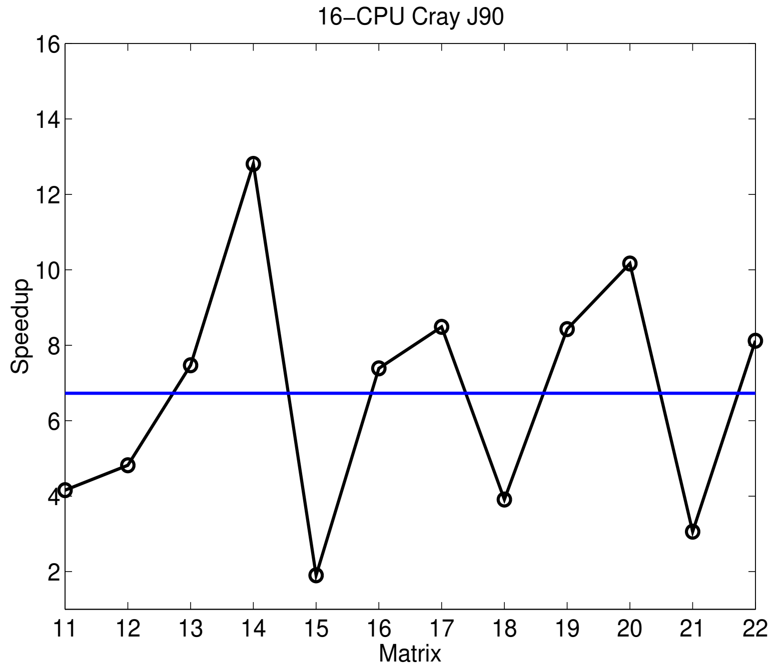
Machine	CPUs	Bus Bandwidth	Read Latency	Memory Size	Mflops Per CPU
Cray C90	8	245.8 GB/s	96 ns	640 MB	960
Cray J90	16	51.2 GB/s	330 ns	640 MB	200
Power Challenge	16	1.2 GB/s	252 ns	4 GB	360
AlphaServer	8	1.6 GB/s	260 ns	4 GB	600

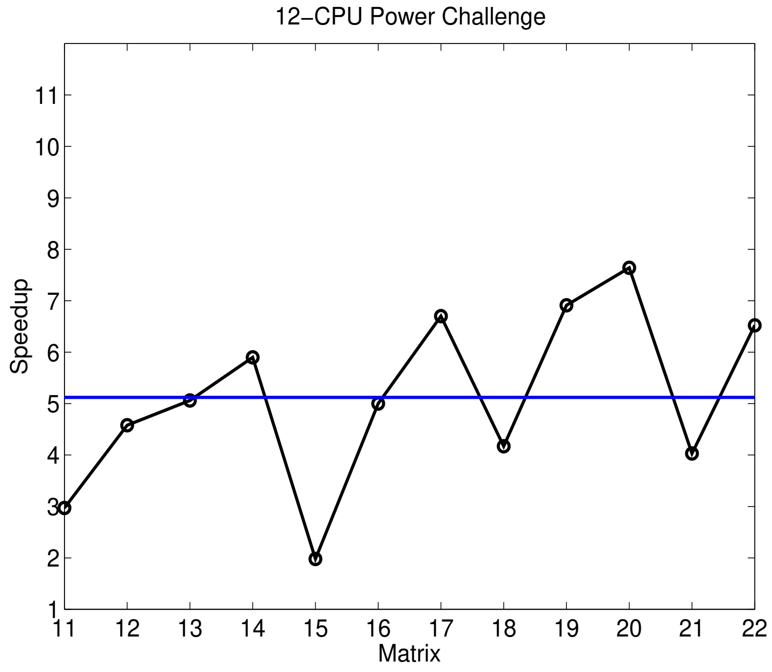
- Cost of lock acquisition and release, in microseconds (cycles)

Machine	$P = 1$	$P = 4$	P=8
Cray C90	1.34 (323)	1.09 (261)	1.40 (336)
Cray J90	2.67 (267)	4.17 (417)	4.42 (442)
Power Challenge	1.13 (102)	1.98 (179)	2.02 (182)
AlphaServer 8400	0.98 (294)	2.26 (678)	2.71 (814)

Speedup

- Compared with the best sequential runtime achieved by SuperLU.
- Time includes everything except column preordering.





Performance Highlights

- Large matrices on Origin 2000

Matrix	N	$nnz(A)$	P=1	P=8	P=18	Speedup
EX11	16614	1096948	209	33	20	10
RAEFSKY4	19779	1316789	229	39	25	9
BBMAT	38744	1771722	605	166	64	9
VAVASIS3	41092	1683902	598	136	75	8
TWOTONE	120750	1224224	313	58	38	8

- A 3-D flow calculation (matrix 20, EX11)

Machine	CPUs	Speedup	Mflops	Percent peak
C90	8	6	2583	33%
J90	16	12	831	25%
Power Challenge	12	7	1002	23%
Origin 2000	20	10	1335	17%
AlphaServer 8400	8	7	781	17%

Overhead in Parallelization

- Sources of overhead
  - Reduced uniprocessor performance due to smaller blocking.
  - Accessing critical sections.
  - Idle time due to task dependencies or inefficient scheduling.
- Notation
  - T_s = the best serial time obtained from SuperLU
  - T_1 = the execution time of the parallel code on one processor
  - T_P = the parallel execution time on P processors
  - T_I = the total idle time of all processors
- How to measure? $P\,T_P \approx T_1 + T_I$.
  - Uniprocessor inefficiency

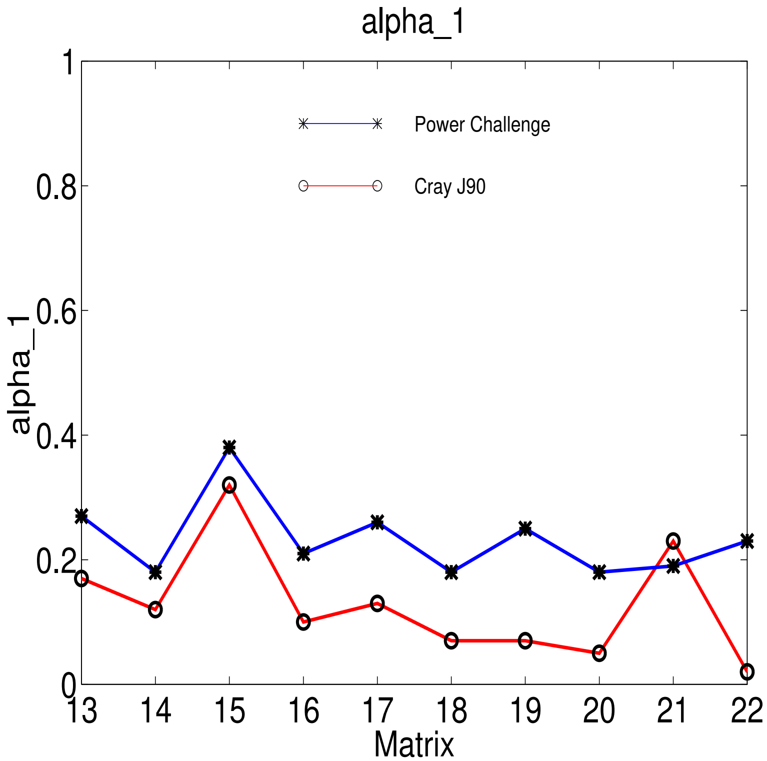
$$\alpha_1 = \frac{T_1 - T_s}{T_1} = 1 - \frac{T_s}{T_1} \ .$$

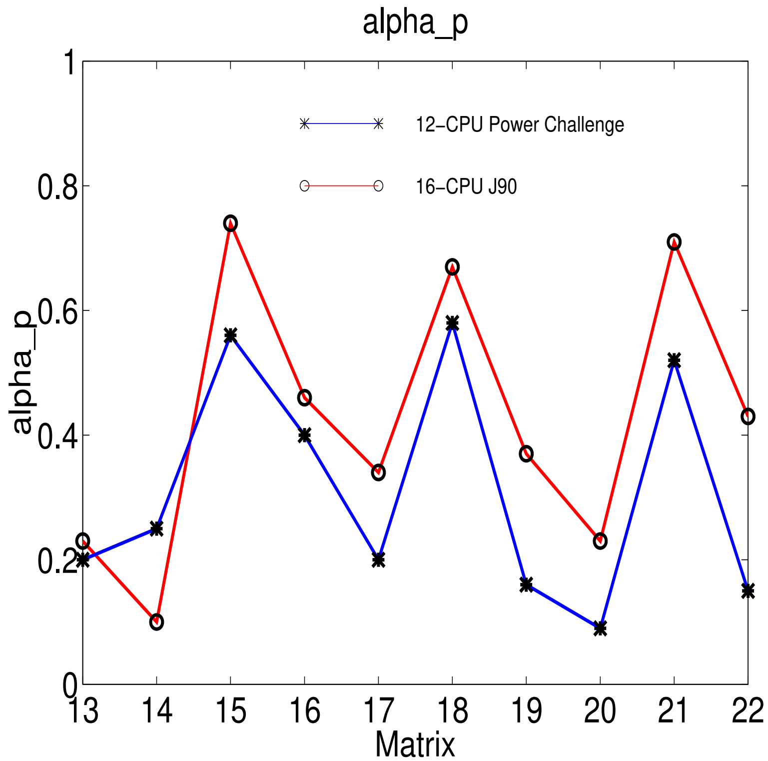
  - Multiprocessor inefficiency

$$\alpha_p = \frac{T_I}{P\,T_P} \ .$$

Measured Uniprocessor and Multiprocessor Inefficiencies

- $0 \leq \alpha \leq 1$, and smaller is better.
- Overall parallel efficiency $E \approx (1 - \alpha_1) \cdot (1 - \alpha_p)$.

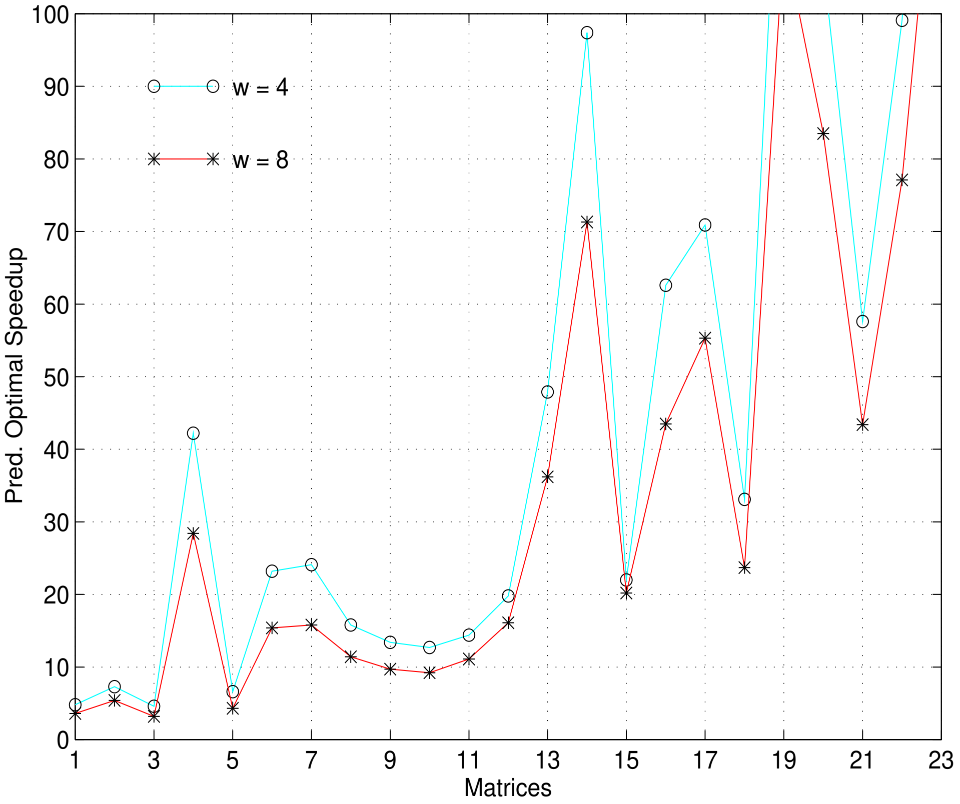




Predicted Optimal Speedup of the 1D Algorithm

Simulations based on

- Each floating-point operation takes one unit of time
- Optimistic scheduling on infinite number of processors
- Memory access or communication is free



Available Software

- SuperLU available on Netlib

  http://www.netlib.org/scalapack/prototype/

  – Real and complex, single and double precisions

- SuperLU_MT shared memory code (ALPHA version)

  ftp.cs.berkeley.edu : pub/src/lapack/SuperLU/superlu_mt.tar.gz

  http://www.nersc.gov/research/SCG/Sherry/

  Will soon appear on Netlib.

  – Double precision real

- Papers on Web

  http://www.nersc.gov/research/SCG/Sherry

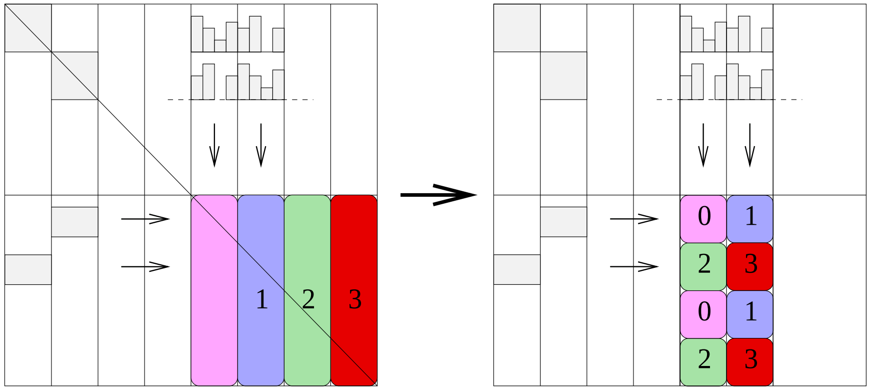
SuperLU and SuperLU_MT

- LAPACK-style interface
  - Simple and expert driver routines
  - Computational routines
  - Comprehensive testing routines
- Functions
  - Minimum degree ordering [MMD, Liu '85] applied to $A^T A$ or $A + A^T$
  - User-input to control pivoting
    - pre-assigned row and/or column permutations
    - partial pivoting with threshold
  - Transposed system
  - Equilibration
  - Condition number estimation
  - Iterative refinement to improve accuracy
  - Componentwise error bounds [Skeel '79, Arioli/Demmel/Duff '89]

Challenges On Distributed Memory Machines

- Numerical pivoting requires dynamic adaptation of data structures
  - Dynamic scheduling and load balancing too expensive
  - Symbolic algorithm too fine-grained

- Scalability requires 2D matrix partitioning [Schreiber '93, Rothberg '93, Gupta/Kumar '95]



+ Higher level of concurrency
– More synchronizations
– More complex data structures

Some Distributed LU Efforts

- Network of transputers [van der Stappen/Bisseling/van der Vorst '93]
  - Distribute matrix in a grid fashion
  - Perform a parallel search for compatible pivots and a parallel rank-m update of the reduced matrix
  - Okay on transputers with low computation speed relative to communication

- Cray T3E [Fu/Jiao/Yang '97]
  - Use QR data structures, but perform LU operation (even on the explicit zeros)
    Let H be the symbolic Householder matrix used to produce $A = QR$. Then the nonzero structures of L and U are contained in those of H and R for all possible pivot orders. [George/Ng '87, George/Liu/Ng '88]
  - Static data distribution and task scheduling
  - Extra fills and operations can be arbitrarily large.

Our Research On Distributed LU [Demmel/Li '97]

- Many engineering problems can be solved without pivoting or with local restricted pivoting.
- GENP plus many "tricks" to ensure stability
  - Control element growth
    - extra precision (e.g., use both single and double)
    - low rank modification to A
  - Iterative refinement
  - ...
- On cluster of SMPs, can perform local pivoting on each SMP.
- Research issues
  - Understand trade-offs between stability and costs
  - Design software to choose appropriate techniques automatically

Extra Precision

- Standard backward error bound for GENP $A + E = LU$

$$|E_{ij}| \leq O(\varepsilon_{single})(|L| \cdot |U|)_{ij}$$

- Accumulating inner products in double precision, we expect

$$|E_{ij}| \leq O(\varepsilon_{single})(|U| + |L| \cdot diag(|U|))_{ij}$$

- Storing either U_{ij} or L_{ij} in double precision if either U_{ij} or $(|L| \cdot diag(|U|))_{ij}$ too large, we expect

$$|E_{ij}| \leq O(\varepsilon_{double})(|U| + |L| \cdot diag(|U|))_{ij}$$

  Need dynamic precision control

- Need more than single and double?

Low Rank Modification

- A rank k correction to A results in a rank k correction to A^{-1}.

  Sherman-Morrison-Woodbury formula: [Golub/Van Loan book]

  Let U, V be $n \times k$ matrices

$$(A + \underbrace{UV^T}_{Rank-k})^{-1} = A^{-1} - A^{-1}U \underbrace{(I + V^T A^{-1} U)^{-1}}_{k \times k} V^T A^{-1}$$

  Example: During Gaussian elimination, replace a few tiny diagonal pivots by

  – $\varepsilon_{single}||A||$

  – large values, and correct the solution at the end.

$$\begin{pmatrix} a_{ii} + \Delta a_{ii} & & \\ & \ddots & \\ \vdots & & a_{jj} + \Delta a_{jj} \\ & & \vdots \end{pmatrix}$$

Other Interests

- Blocked ILU factorization as preconditioner.
- Parallel sparse triangular solve.
  important in distributed memory code, ILU and eigenvalue computation
- Distributed parallel symbolic algorithm.
  enable solutions of even larger problems
- Updating and downdating.

Software Survey On Sparse Direct Solvers

See Table 6.1 in my thesis.