

SuperLU: Sparse Direct Solvers

X. Sherry Li

xsli@lbl.gov

ACTS Collection Workshop

September 4, 2002

- ◆ Introduction
- ◆ Use of SuperLU in large scale applications
- ◆ Overview of the algorithms
- ◆ Sparse matrix distribution and parallel performance
- ◆ Summary

- ◆ Solve general sparse linear system $A x = b$.
 - Example: A of dimension 10^5 , only $10 \sim 100$ nonzeros per row
- ◆ Algorithm: Gaussian elimination (LU factorization: $A = LU$), followed by lower/upper triangular solutions.
 - Store only nonzeros and perform operations only on nonzeros.
- ◆ Efficient implementation for high-performance architectures.
- ◆ Software portable on many platforms.

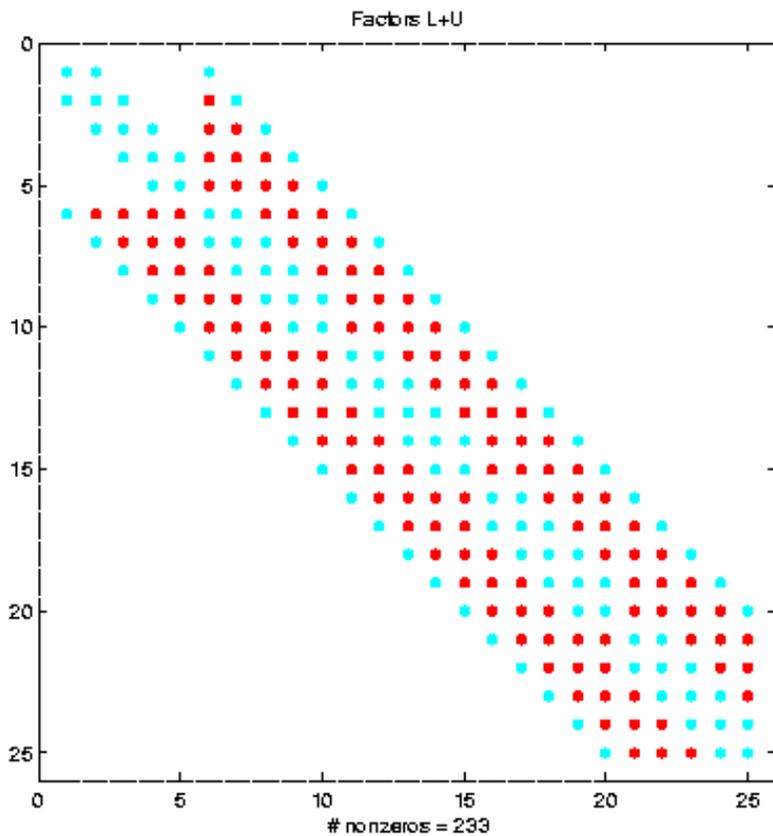
	SuperLU	SuperLU_MT	SuperLU_DIST
Platform	Serial	SMP	Distributed
Language (callable from F77)	C	C + Pthread (or pragmas)	C + MPI
Data type	Real/complex, Single/double	Real, double	Real/complex, Double

- ◆ Source, Users' Guide, papers available:

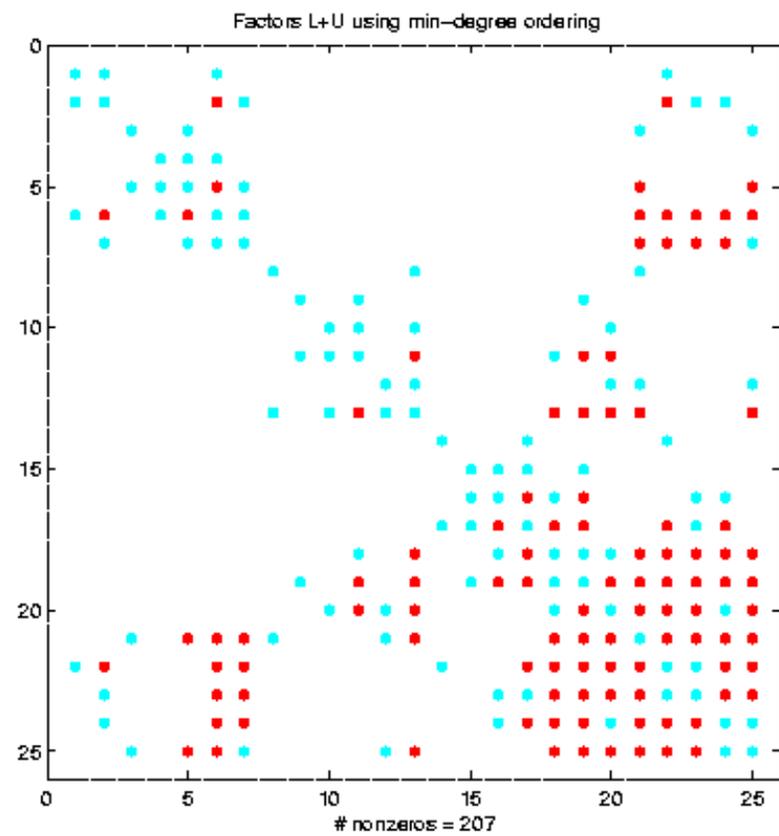
www.nersc.gov/~xiaoye/SuperLU

- ◆ Original zero entry A_{ij} becomes nonzero in L or U.

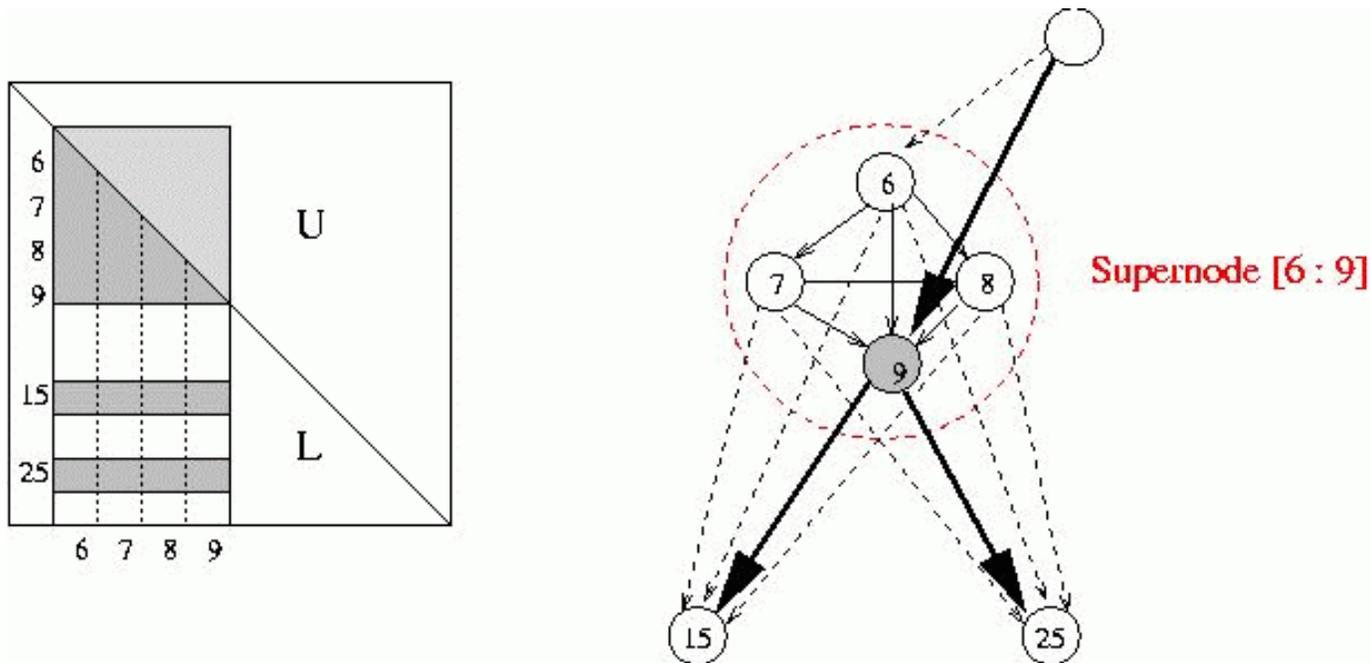
Natural order: nonzeros = 233



Min. Degree order: nonzeros = 207



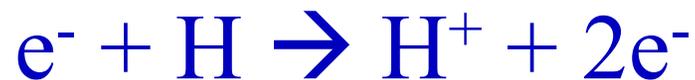
- ◆ Exploit dense submatrices in the L & U factors



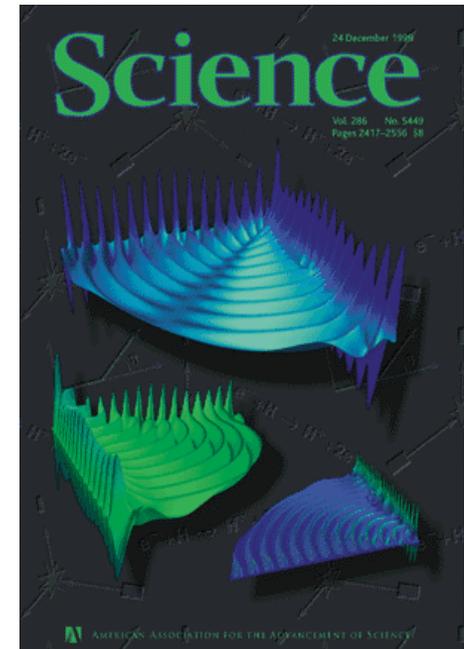
- ◆ Why are they good?

- Permit use of Level 3 BLAS
- Reduce inefficient indirect addressing (scatter/gather)
- Reduce graph algorithms time by traversing a coarser graph

- ◆ First solution to a long-standing unsolved problem of scattering in a quantum system of 3 charged particles. [Recigno, Baertschy, Isaacs & McCurdy, *Science*, 24 Dec 1999]
- ◆ The simplest nontrivial example is the ionization of a hydrogen atom by collision with an electron.

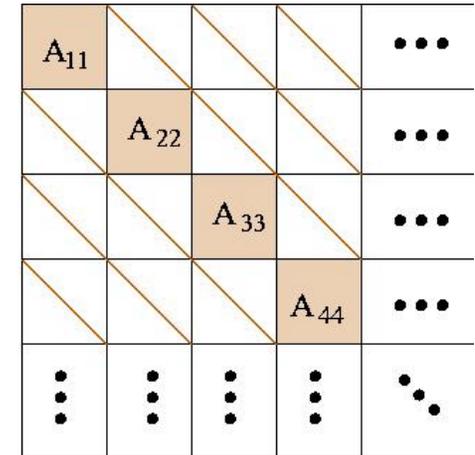


- ◆ Seek the particles' wave functions represented by the time-independent Schrodinger equation.



- ◆ Sparse, complex (non-Hermitian), unsymmetric linear systems.

- Diagonal blocks have the structure of 2D finite difference Laplacian matrices.
- Off-diagonal block is a diagonal matrix.
- Between 6 to 24 blocks, each of dimension between 200K and 350K → Total dimension as large as 8.4M.



- ◆ SuperLU_DIST as block diagonal preconditioner.

$$M^{-1}A x = M^{-1}b$$

$$M = \text{diag}(A_{11}, A_{22}, A_{33}, \dots)$$

- ◆ 12 to 35 iterations @ 2 to 3 minutes/iteration on 24 processors of IBM SP.

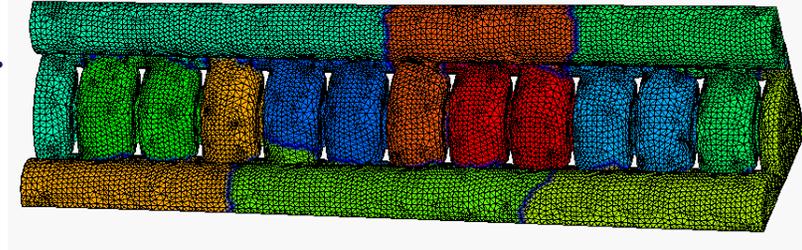
SuperLU in Accelerator Cavity Design

- ◆ Model large, complex cavities accurately for Next Generation Accelerator.
- ◆ Maxwell equation in electromagnetic simulation.
- ◆ Finite element methods lead to large sparse generalized eigensystem $K x = \lambda M x$.
- ◆ Seek interior eigenpairs, tightly clustered.
- ◆ Need to speed up Lanczos convergence by shift-invert \rightarrow Seek largest eigenpairs, well separated, of the transformed system.

$$M (K - \sigma M)^{-1} x = \mu M x$$

$$\mu = 1 / (\lambda - \sigma)$$

- ◆ Shifted linear system is ill-conditioned and needs to be solved accurately \rightarrow Hard for iterative solvers!



- ◆ Build exact shift-invert eigen solver with SuperLU_DIST integrated into Lanczos code PARPACK.

- ◆ DDS model on IBM SP
 - Damped, Detuned Structure, include linear and quadratic elements
 - 380K with 15.8M nonzeros
 - ~4.2 solves per eigenpair @ 24 seconds/solve on 8 procs.
 - 1.3M with 20.1M nonzeros
 - ~4.5 solves per eigenpair @ 39 seconds/solve on 32 procs.

- ◆ Solution accurate to more than 12 sig. digits in double precision.

- ◆ Sparse LU factorization: $P_r A P_c^T = L U$
 - Choose permutations P_r and P_c for numerical stability, minimizing fill-in, and maximizing parallelism.

- ◆ Phases for sparse direct solvers.
 1. Order equations & variables to minimize fill-in.
 - NP-hard, so use heuristics based on combinatorics.
 2. Symbolic factorization.
 - Identify supernodes, set up data structures and allocate memory for L & U.
 3. Numerical factorization – usually dominates total time.
 - How to pivot?
 4. Triangular solutions – usually less than 5% total time.

- ◆ In SuperLU_DIST, only numeric phases are parallel so far.

- ◆ Goal of pivoting is to control element growth in L & U for stability
- ◆ Example: partial pivoting during factorization: $PA = LU$ (GEPP)
 - Used in sequential SuperLU and SuperLU_MT
- ◆ Partial pivoting implies:
 - Dynamic change of fill patterns of L & U
 - Must interleave symbolic & numerical factorizations
 - Lots of small messages
 - Slow on parallel machines with high latency
- ◆ Static pivoting used in SuperLU_DIST (GESP)
 - Before factorization, scale and permute A to maximize diagonal: $P_r D_r A D_c = A'$
 - During factorization of $A' = LU$, replace tiny pivots by $\sqrt{\epsilon} \|A\|$, without changing data structures for L & U
 - If needed, use a few steps of iterative refinement after the first solution
 - Symbolic and numerical factorizations decoupled

◆ Local greedy heuristics

- Minimum degree (upper bound on fill-in)

- [Tinney/Walker `67, George/Liu `79, Liu `85, Amestoy/Davis/Duff `94, Ashcraft `95, Duff/Reid `95, et al.]

- Minimum deficiency (actual fill-in)

- [Tinney/Walker `67, Ng/Raghavan `97, et al.]

◆ Global graph partitioning heuristics

- Nested dissection [George `73]

- Multilevel schemes [Hendrickson/Leland `94, Karypis/Kumar `95, et al.]

- Spectral bisection [Simon et al. `90-`95, et al.]

- Geometric and spectral bisection [Chan/Gilbert/Teng `94]

◆ Hybrid of the above two [Ashcraft/Liu `96, Hendrickson/Rothberg `97]

- ◆ Use symmetric ordering for Cholesky of $A^T A$
 - If $R^T R = A^T A$ and $PA = LU$, then for any row permutation P ,
 $\text{struct}(L+U) \subseteq \text{struct}(R^T+R)$ [George/Ng '87]
 - Making R sparse tends to make L & U sparse
 - Strategy:
 1. Find a good symmetric ordering P_c from $A^T A$
 2. Apply P_c columns of A : $A' = A P_c^T$
 $A'^T A' = (A P_c^T)^T (A P_c^T) = P_c (A^T A) P_c^T$

- ◆ Column minimum degree based solely on A
 - Matlab; Larimore/Davis/Gilbert/Ng '98

- ◆ Use symmetric ordering for Cholesky of A^T+A
 - If $R^T R = A^T+A$ and $A = LU$, then $\text{struct}(L+U) \subseteq \text{struct}(R^T+R)$
 - Making R sparse tends to make L & U sparse
 - Strategy:
 1. Find a good symmetric ordering P_c from A^T+A
 2. Apply P_c to both rows and columns of A : $A' = P_c A P_c^T$
 $\text{struct}(A') = \text{struct}(P_c A P_c^T) \subseteq \text{struct}(P_c(A^T+A) P_c^T)$

- ◆ Use symmetric ordering based solely on A
 - Work in progress [Amestoy/Li/Ng '02]

◆ SuperLU library contains routines:

- Form $A^T A$
- Form $A^T + A$
- MMD (Multiple Minimum Degree, courtesy of Joseph Liu)
- COLAMD: www.netlib.org/linalg/colamd/

◆ You may use any other – just input a permutation vector to SuperLU

Example:

- (Par)Metis: www-users.cs.umn.edu/~karypis/metis/
- Chaco: www.cs.sandia.gov/~bahendr/chaco.html
- ...

		GEPP, COLAMD		GESP, AMD(A^T+A)	
Matrix	N	Fill (10^6)	Flops (10^9)	Fill (10^6)	Flops (10^9)
BBMAT	38744	49.8	44.6	40.2	34.0
ECL32	51993	73.5	120.4	42.7	68.4
MEMPLUS	17758	4.4	5.5	0.15	0.002
TWOTONE	120750	22.6	8.8	11.9	8.0
WANG4	26068	27.7	35.3	10.7	9.1

◆ Cholesky [George/Liu '81 book]

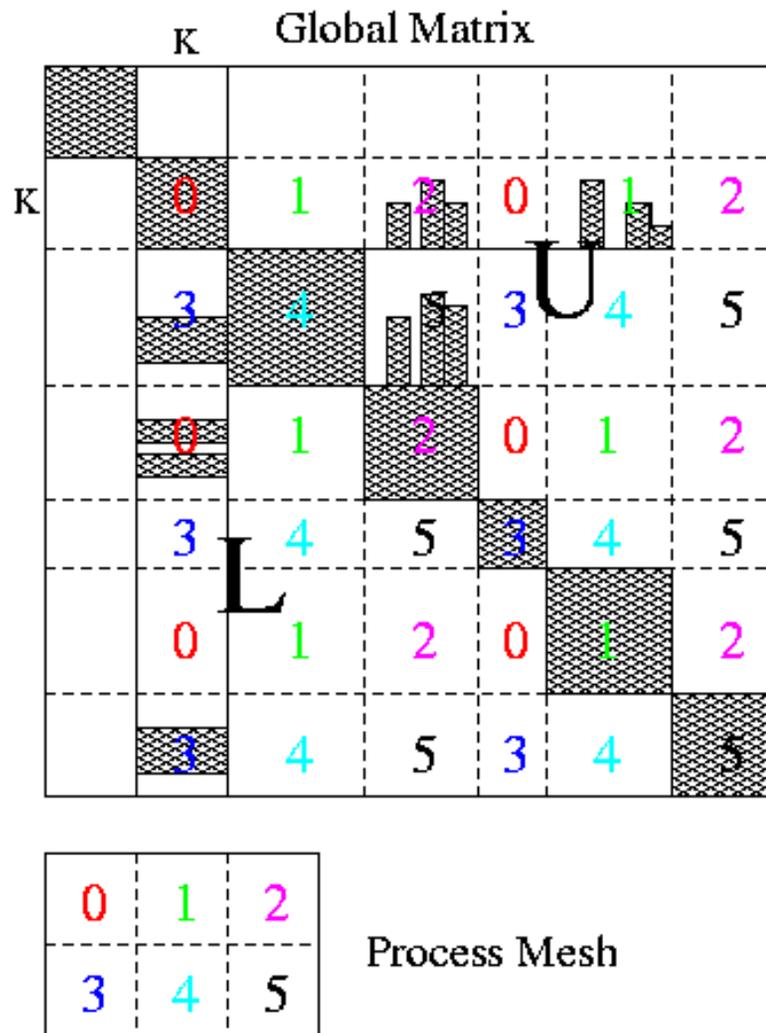
- Use elimination graph of L and its transitive reduction (elimination tree)
- Complexity linear in output: $O(\text{nnz}(L))$

◆ LU

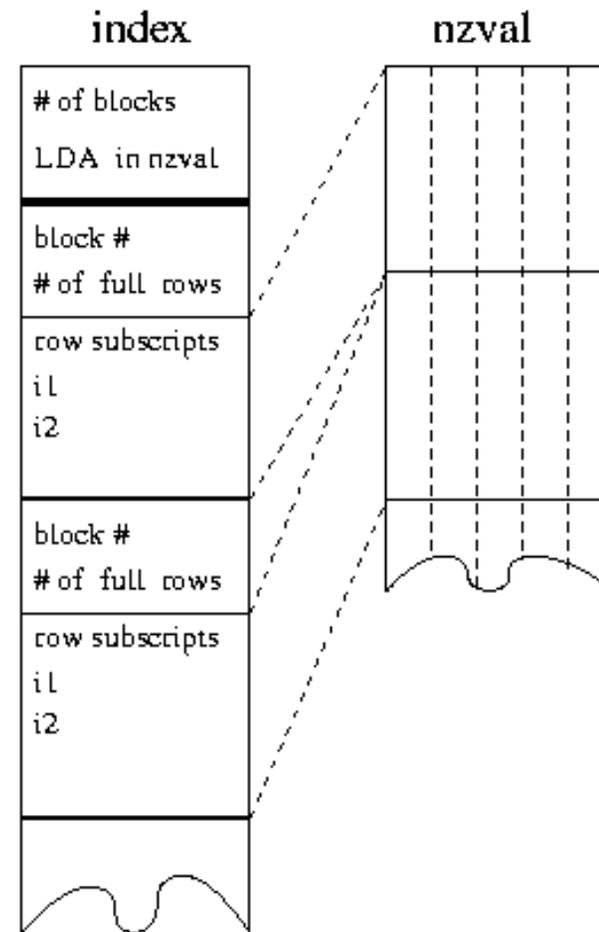
- Use elimination graphs of L & U and their transitive reductions (elimination DAGs) [Tarjan/Rose '78, Gilbert/Liu '93, Gilbert '94]
- Improved by symmetric structure pruning [Eisenstat/Liu '92]
- Improved by supernodes
- Complexity greater than $\text{nnz}(L+U)$, but much smaller than $\text{flops}(LU)$

- ◆ Sequential SuperLU
 - Enhance data reuse in memory hierarchy by calling Level 3 BLAS on the supernodes
- ◆ SuperLU_MT
 - Exploit both coarse and fine grain parallelism
 - Employ dynamic scheduling to minimize parallel runtime
- ◆ SuperLU_DIST
 - Enhance scalability by static pivoting and 2D matrix distribution

2D Block Cyclic Layout and Data Structures



Storage of block column of L



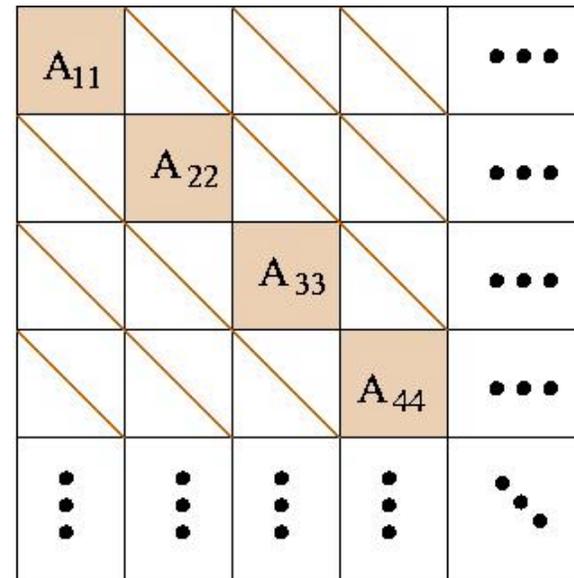
Create 2D Process Grid from MPI Communicator

- ◆ The 2D process grid/communicator must be created from an existing base MPI communicator (e.g., MPI_COMM_WORLD).
- ◆ SuperLU uses the newly created communicator for all the internal communications.

- ◆ Example:

$$M^{-1}A x = M^{-1} b$$

$$M = \text{diag}(A_{11}, A_{22}, A_{33}, \dots)$$



Two Ways to Create a SuperLU Process Grid

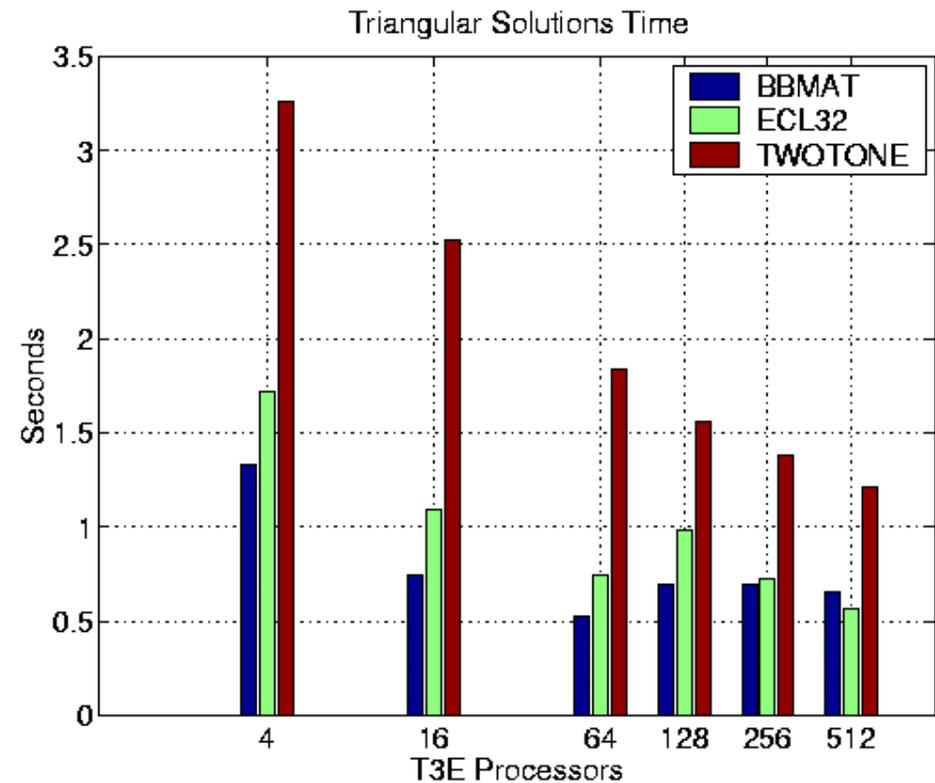
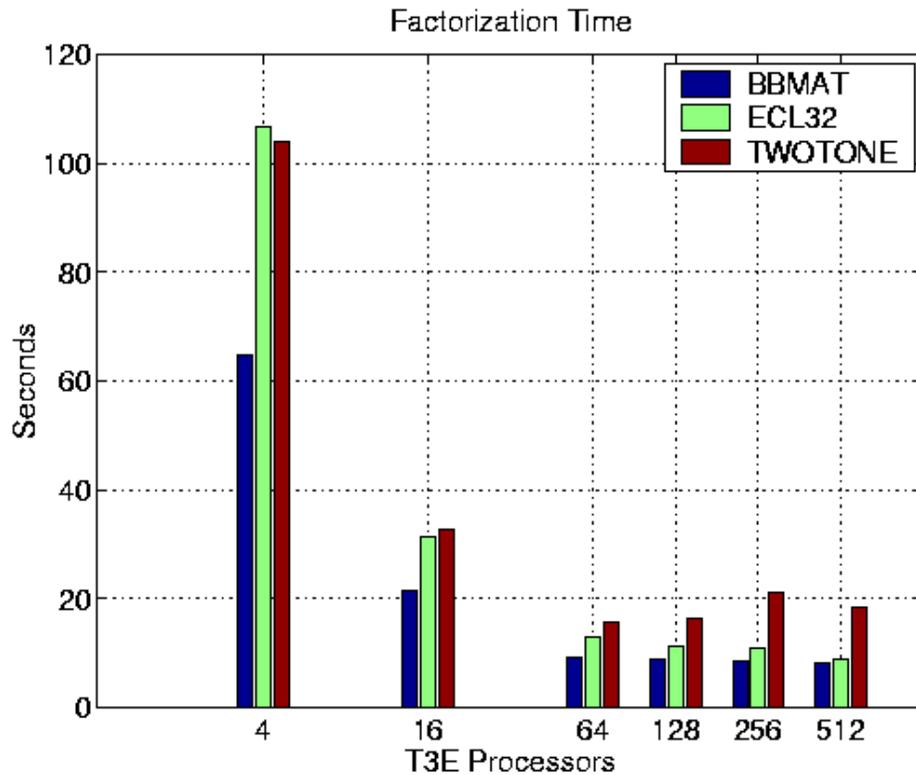


- ◆ `Superlu_gridinit(MPI_Comm Bcomm, int nprow, int npc, gridinfo_t *grid);`
 - This maps the first $nprow \times npc$ processes in the MPI communicator Bcomm to SuperLU 2D grid.
- ◆ `Superlu_gridmap(MPI_Comm Bcomm, int nprow, int npc, int usermap[], int ldumap, gridinfo_t *grid);`
 - This maps an arbitrary set of $nprow \times npc$ processes in the MPI communicator Bcomm to SuperLU 2D grid. The ranks of the selected MPI processes are given in Usermap[] array. For example:

	0	1	2
0	11	12	13
1	14	15	16

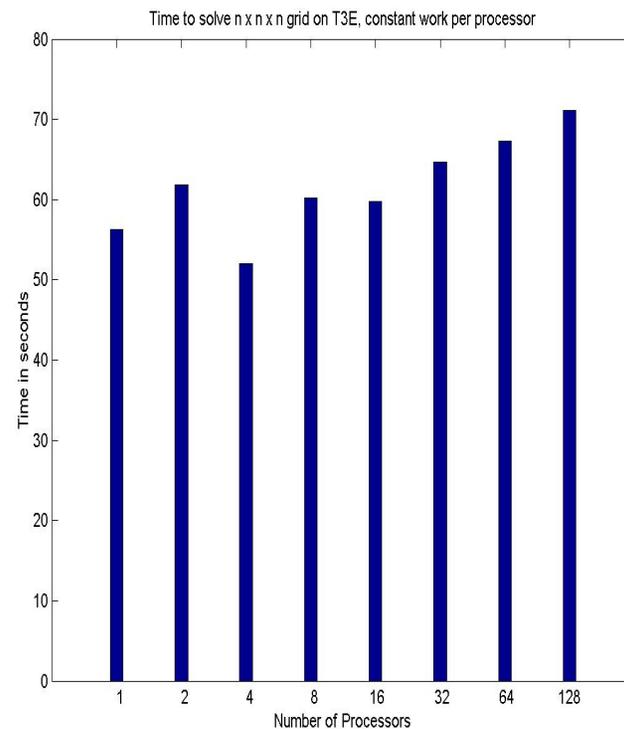
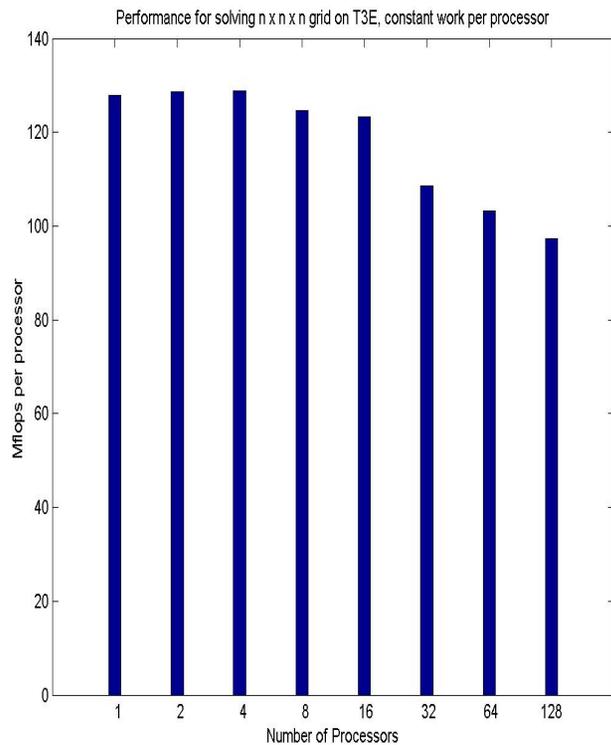
Example Matrices

Matrix	Source	Symm	N	nnz(A)	nnz(L+U)	Flops
BBMAT	Fluid flow	.54	38,744	1.77M	40.2M	31.2G
ECL32	Device sim.	.93	51,993	.38M	42.7M	68.4G
TWOTONE	Circuit sim.	.43	120,750	1.22M	11.9M	8.0G



◆ T3E

- 3D $K \times K \times K$ cubic grids, scale $N^2 = K^6$ with P for constant work per processor
- Up to 12.5 Gflops on 128 processors



- ◆ IBM SP: $K = 100$, $N = 1M$, 49 Gflops (267 Seconds)

◆ LAPACK-style interface

- Simple and expert driver routine
- Computational routines
- Comprehensive testing routines and example programs

◆ Functionalities

- Minimum degree ordering [MMD, Liu '85] applied to $A^T A$ or $A^T + A$
- User-controllable pivoting
 - Pre-assigned row and/or column permutations
 - Partial pivoting with threshold
- Solving transposed system
- Equilibration
- Condition number estimation
- Iterative refinement
- Componentwise error bounds [Skeel '79, Arioli/Demmel/Duff '89]

- ◆ Good implementations of sparse LU on high-performance machines
- ◆ More sensitive to latency than dense case
- ◆ Need more families of unsymmetric test matrices
- ◆ Continuing developments – being funded by DOE TOPS SciDAC and NSF NPACI programs
 - Improve triangular solution
 - ILU preconditioner
 - Parallel ordering and symbolic factorization
 - Integrate into applications
- ◆ “Eigentemplates” book (www.netlib.org/etemplates) for survey of other sparse direct solvers
 - LL^T , LDL^T , LU