

SuperLU: Sparse Direct Solver

X. Sherry Li

xqli@lbl.gov

<http://crd.lbl.gov/~xiaoye>

ACTS Collection Workshop

August 6, 2003

- ◆ Status of the software
- ◆ Some background of the algorithms
- ◆ Sparse matrix distribution and user interface
- ◆ Dissection of two applications
 - Quantum mechanics (linear system)
[with M. Baertschy, C. W. McCurdy, T. N. Rescigno, W. A. Isaacs]
 - Accelerator design (eigenvalue problem)
[with P. Husbands, C. Yang]

- ◆ 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 and portable implementation for high-performance architectures; flexible interface.

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

- ◆ Friendly interface for Fortran users
- ◆ Continuing fundings from DOE TOPS SciDAC and NSF NPACI programs

◆ LAPACK-style interface

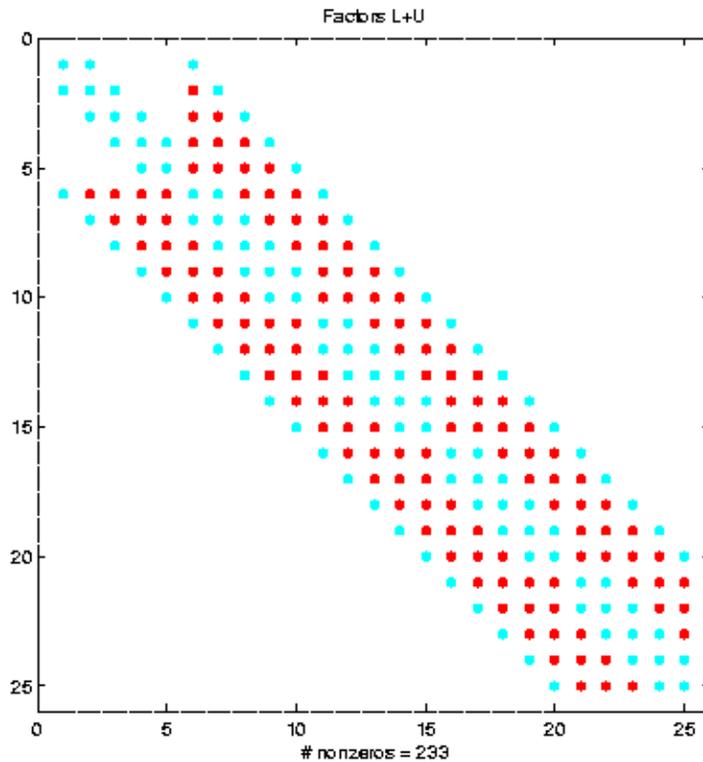
- Simple and expert driver routines
- Computational routines
- Comprehensive testing routines and example programs

◆ Functionalities

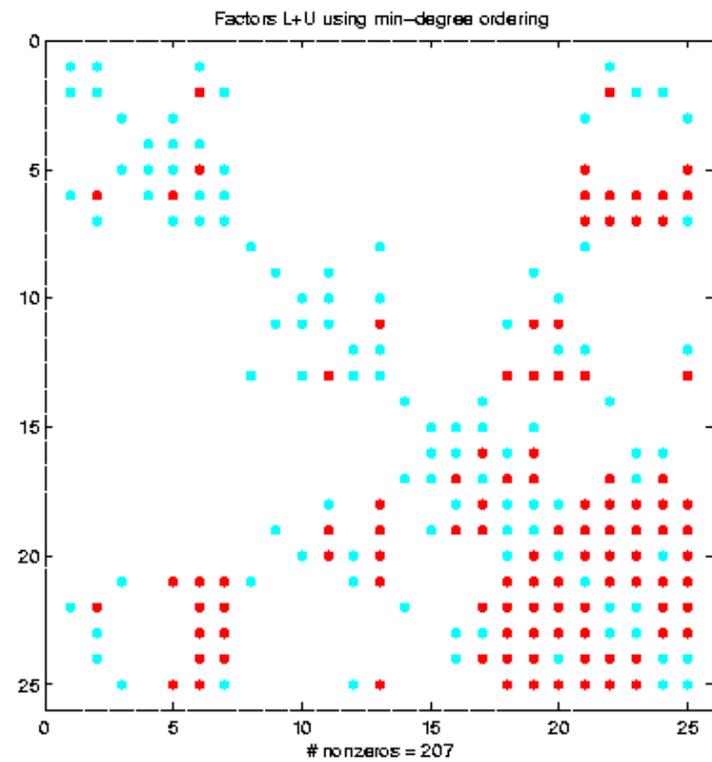
- Minimum degree ordering [MMD, Liu `85] applied to $A^T A$ or $A^T + A$
- User-controllable pivoting
 - Pre-assigned row and/or column permutations
 - Partial pivoting with threshold
- Solving transposed system
- Equilibration
- Condition number estimation
- Iterative refinement
- Componentwise error bounds [Skeel `79, Arioli/Demmel/Duff `89]

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

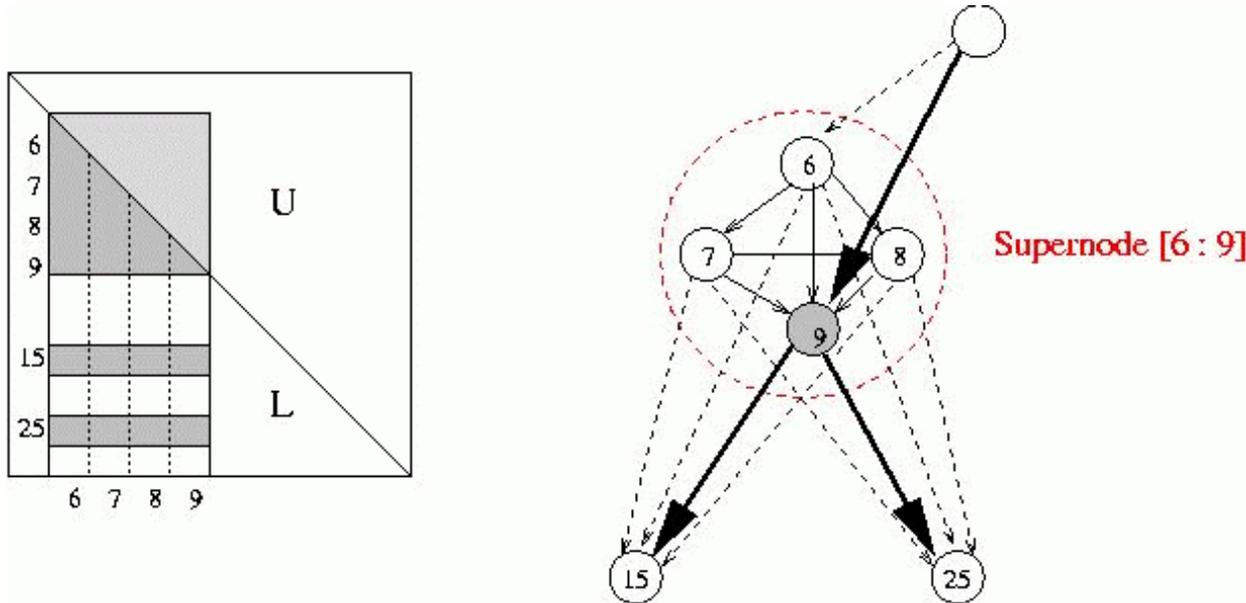
Natural order: nonzeros = 233



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

- ◆ 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
 - For sparse factorizations, often relax the pivoting rule to trade with better sparsity and parallelism (e.g., threshold pivoting, static pivoting, . . .)
- ◆ **Partial pivoting** used in sequential SuperLU (GEPP)
 - Can force diagonal pivoting (controlled by diagonal threshold)
 - Hard to implement scalably for sparse factorization
- ◆ **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
 - ➔ Quite stable in practice

- ◆ 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 two [Ashcraft/Liu `96, Hendrickson/Rothberg `97]

- ◆ Can use a symmetric ordering on a symmetrized matrix . . .
- ◆ Case of partial pivoting (sequential SuperLU):
Use ordering based on $A^T A$
 - 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 . . .
- ◆ Case of static pivoting (SuperLU_DIST):
Use ordering based on 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 . . .
 - Can find better ordering based solely on A , without symmetrization [Amestoy/Li/Ng `03]

- ◆ SuperLU library contains the following 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
- ...

Ordering Comparison



		GEPP, COLAMD (SuperLU)		GESP, AMD(A^T+A) (SuperLU_DIST)	
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]

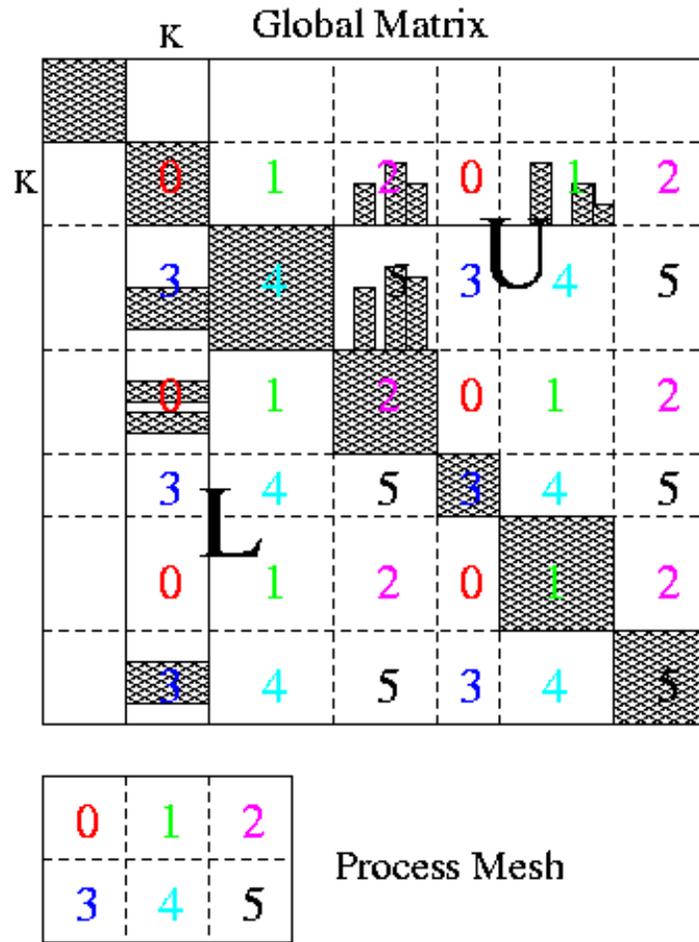
- Use elimination graph of L and its transitive reduction (elimination tree)
- Complexity linear in output: $O(\text{nnz}(L))$

◆ LU

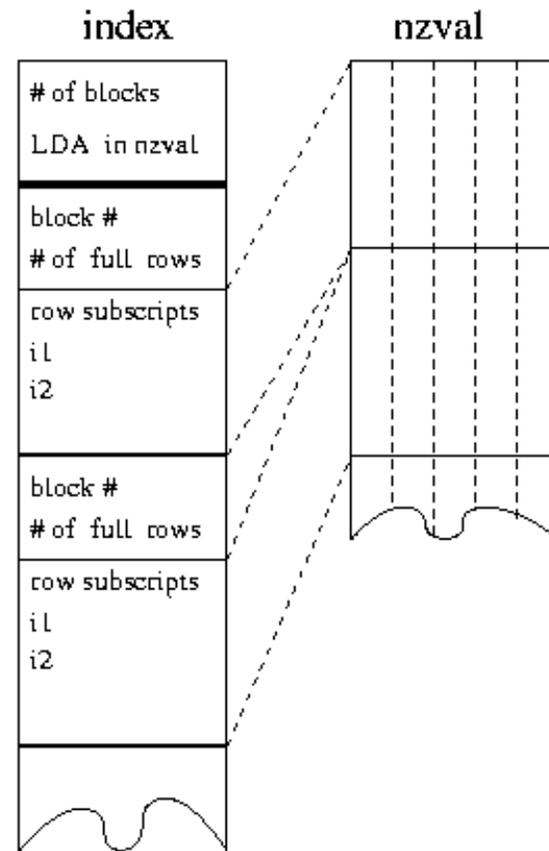
- Use elimination graphs of L & U and their transitive reductions (elimination DAGs) [Tarjan/Rose `78, Gilbert/Liu `93, Gilbert `94]
- Improved by symmetric structure pruning [Eisenstat/Liu `92]
- Improved by supernodes
- Complexity greater than $\text{nnz}(L+U)$, but much smaller than $\text{flops}(LU)$

- ◆ 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 for L and U



Storage of block column of L



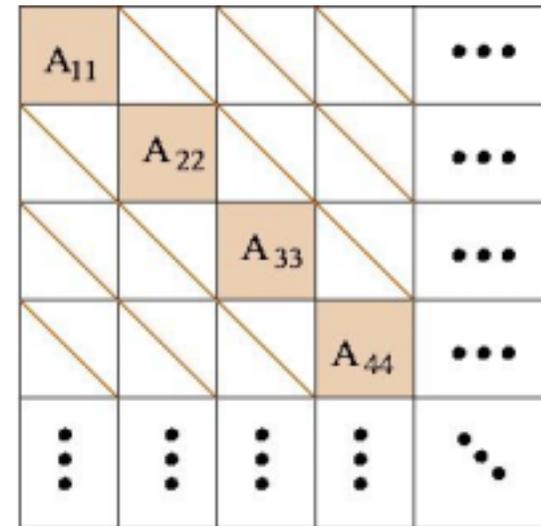
- ◆ The 2D process grid/communicator must be created from an existing base MPI communicator (e.g., MPI_COMM_WORLD).
- ◆ SuperLU uses the newly created communicator for all the internal communications.

- ◆ Example:

Solving a preconditioned linear system

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

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



Two Ways to Create a SuperLU Process Grid



◆ `Superlu_gridinit(MPI_Comm Bcomm, int nrow, int ncol, gridinfo_t *grid);`

- This maps the first $nrow \times ncol$ processes in the MPI communicator Bcomm to SuperLU 2D grid.

◆ `Superlu_gridmap(MPI_Comm Bcomm, int nrow, int ncol, int usermap[], int ldumap, gridinfo_t *grid);`

- This maps an arbitrary set of $nrow \times ncol$ processes in the MPI communicator

Bcomm to SuperLU 2D grid. The ranks of the selected MPI processes are given in `usermap[]` array. For example:

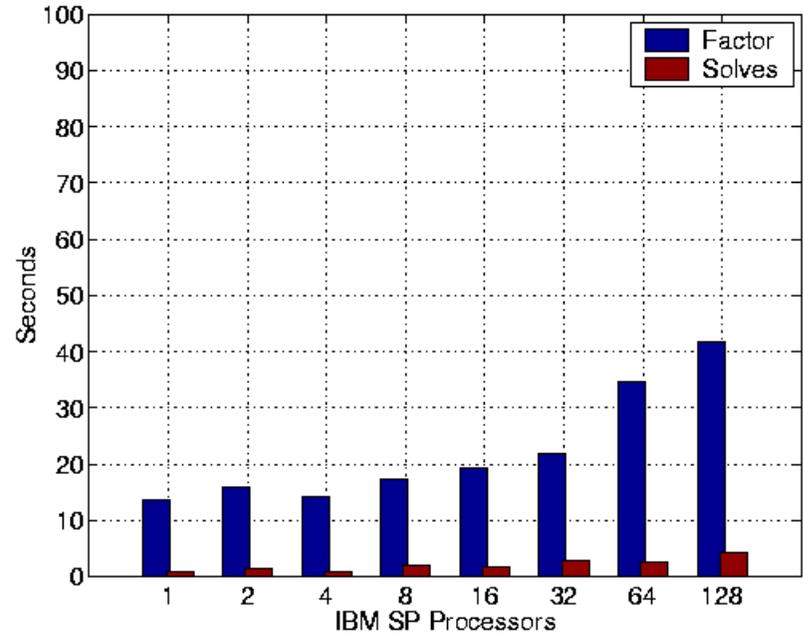
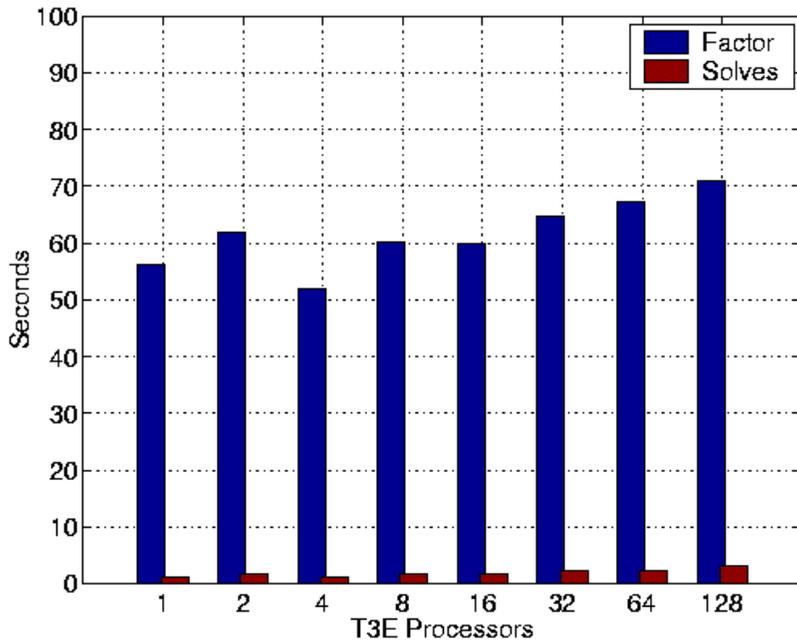
	0	1	2
0	11	12	13
1	14	15	16

- ◆ Inputs A (sparse) and B (dense) are distributed by block rows
- ◆ Each process has a structure to store local part of A:
 - Number of local rows
 - Number of local nonzeros
 - First row number of the block
 - (nzval, colind, rowptr) – *Compressed Row Storage*
- ◆ The library has a “distribution” phase to re-distribute the initial values of A to the 2D block-cyclic data structure of L & U.
 - All-to-all communication, entirely parallel
 - < 10% of total time for most matrices

Scalability



- ◆ 3D $K \times K \times K$ cubic grids, scale $N^2 = K^6$ with P for constant work per processor
- ◆ Achieved 12.5 and 21.2 Gflops on 128 processors
- ◆ Performance sensitive to communication latency
 - Cray T3E latency: 3 microseconds (~ 2702 flops)
 - IBM SP latency: 8 microseconds (~ 11940 flops)



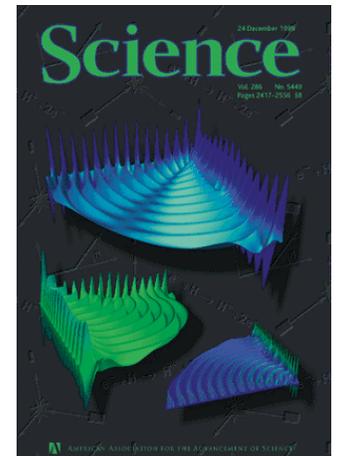
- ◆ Check ordering
- ◆ Diagonal pivoting is preferable
 - E.g., matrix is diagonal dominant, or SPD, . . .
- ◆ Need good BLAS library
 - May need adjust block size for each architecture
(Parameters modifiable in routine `sp_ienv()`)
 - Larger blocks better for uniprocessor
 - Smaller blocks better for parallelism and load balance
 - Open problem: automatic tuning for block size?

Example 1: Quantum Mechanics

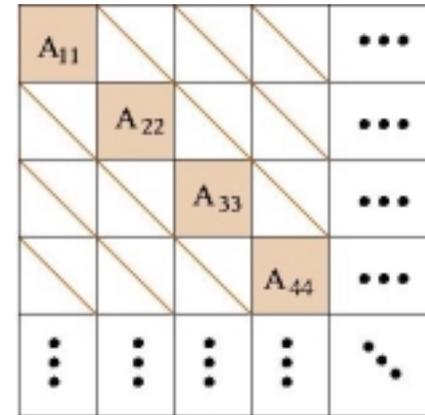
- ◆ Scattering in a quantum system of three charged particles
- ◆ Simplest example is ionization of a hydrogen atom by collision with an electron:



- ◆ Seek the particles' wave functions represented by the time-independent Schrodinger equation
- ◆ First solution to this long-standing unsolved problem [Recigno, McCurdy, et. al. Science, 24 Dec 1999]



- ◆ Finite difference leads to **complex, unsymmetric** systems, very ill-conditioned
 - Diagonal blocks have the structure of 2D finite difference Laplacian matrices
 - Very sparse: nonzeros per row ≤ 13
 - Off-diagonal block is a diagonal matrix
 - Between 6 to 24 blocks, each of order between 200K and 350K
 - Total dimension up to 8.4 M



- ◆ Too much fill if use direct method . . .

- ◆ SuperLU_DIST as block-diagonal preconditioner for CGS iteration

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

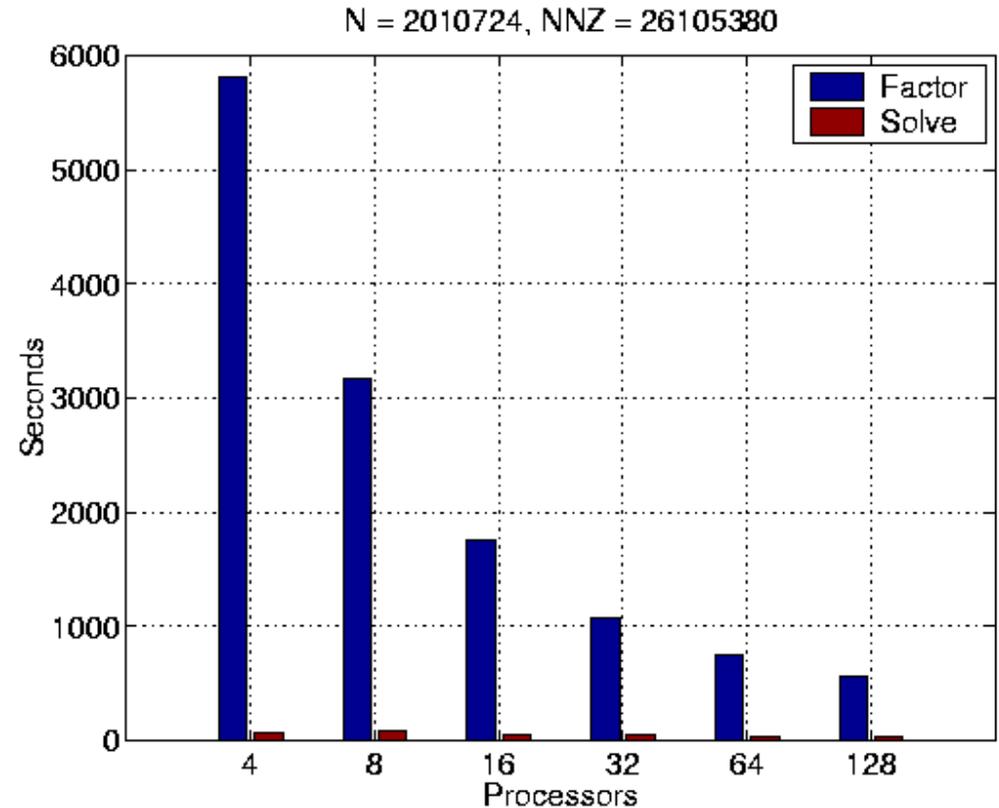
$$M = \text{diag}(A_{11}, A_{22}, A_{33}, \dots)$$

- ◆ Run multiple SuperLU_DIST simultaneously for diagonal blocks
- ◆ **No pivoting, nor iterative refinement**
- ◆ 96 to 280 iterations @ 1 ~ 2 minute/iteration using 64 IBM SP processors
 - **Total time ~ 1 to 8 hours**

One Block Timings on IBM SP



- ◆ Complex, unsymmetric
- ◆ $N = 2\text{ M}$, $NNZ = 26\text{ M}$
- ◆ Fill-ins using Metis: 1.3 G (50x fill)
- ◆ Factorization speed
 - 10x speedup (4 to 128 P)
 - Up to 30 Gflops

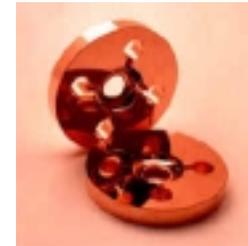


Example 2: Accelerator Cavity Design

- ◆ Calculate cavity mode frequencies and field vectors
- ◆ Solve Maxwell equation in electromagnetic field
- ◆ Omega3P simulation code developed at SLAC

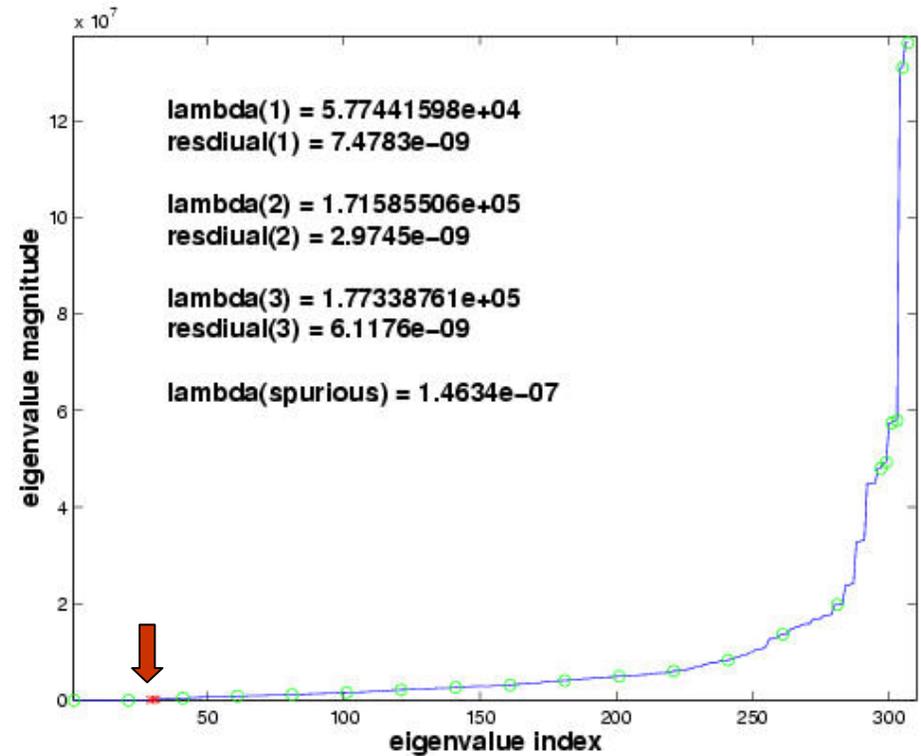


Omega3P model of a 47-cell section of the 206-cell Next Linear Collider accelerator structure



Individual cells used in accelerating structure

- ◆ Finite element methods lead to large sparse generalized eigensystem $\mathbf{K} \mathbf{x} = \lambda \mathbf{M} \mathbf{x}$
- ◆ Real symmetric for lossless cavities; Complex symmetric when lossy in cavities
- ◆ Seek interior eigenvalues (tightly clustered) that are relatively small in magnitude



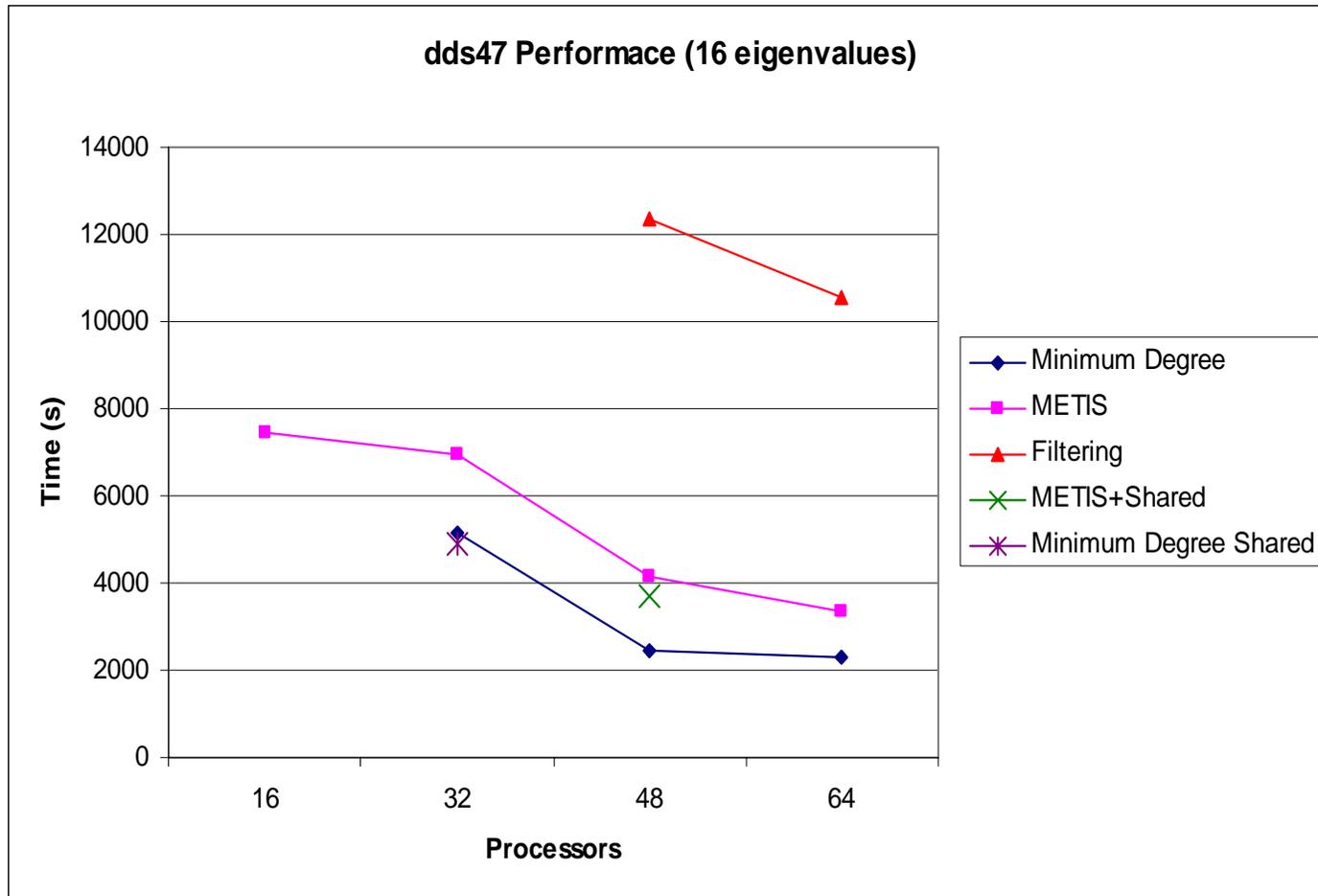
- ◆ Speed up Lanczos convergence by shift-invert
 - ➔ Seek largest eigenvalues, well separated, of the transformed system

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

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

- ◆ The Filtering algorithm [Y. Sun]
 - Inexact shift-invert Lanczos + JOCC
- ◆ We added exact shift-invert Lanczos (ESIL)
 - PARPACK for Lanczos
 - SuperLU_DIST for shifted linear system
 - No pivoting, nor iterative refinement

◆ Total eigensolver time: $N = 1.3$ M, $NNZ = 20$ M



- ◆ DDS47, quadratic elements
 - $N = 7.5 \text{ M}$, $NNZ = 304 \text{ M}$
 - 6 G fill-ins using Metis

- ◆ 24 processors (8x3)
 - Factor: 3,347 s
 - 1 Solve: 61 s
 - Eigensolver: 9,259 s (~2.5 hrs)
 - 10 eigenvalues, 1 shift, 55 solves

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