

SuperLU: Sparse Direct Solver

X. Sherry Li
xsli@lbl.gov

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

7th DOE ACTS Collection Workshop
August 24, 2006

Outline



- ◆ Overview of the software
 - ◆ Background of the algorithms
 - Differences between sequential and parallel solvers
 - ◆ Sparse matrix distribution and user interface
 - ◆ Example program, Fortran 90 interface
 - ◆ Application
-

What is SuperLU?



- ◆ Solve general sparse linear system $\mathbf{A} \mathbf{x} = \mathbf{b}$.
 - Example: \mathbf{A} of dimension 10^5 ,
cxdfsre only $10 \sim 100$ nonzeros per row
- ◆ Algorithm: Gaussian elimination (LU factorization: $\mathbf{A} = \mathbf{L}\mathbf{U}$), 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.

Software Status



	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
 - ◆ SuperLU_MT similar to SuperLU both numerically and in usage
-

- ◆ 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: $D_r A D_c$
 - Condition number estimation
 - Iterative refinement
 - Componentwise error bounds [Skeel '79, Arioli/Demmel/Duff '89]

- ◆ Industrial

- FEMLAB
- HP Mathematical Library
- NAG
- Python (NumPy, SciPy extensions)

- ◆ Academic/Lab:

- In other ACTS Tools: PETSc, Hypre
- NIMROD (simulate fusion reactor plasmas)
- Omega3P (accelerator design, SLAC)
- OpenSees (earthquake simulation, UCB)
- DSpice (parallel circuit simulation, SNL)
- Trilinos (object-oriented framework encompassing various solvers, SNL)
- NIKE (finite element code for structural mechanics, LLNL)

Review of Gaussian Elimination (GE)



- ◆ Solving a system of linear equations $Ax = b$
- ◆ First step of GE: (make sure α not too small . . . may need pivoting)

$$A = \begin{bmatrix} \alpha & \boxed{w^T} \\ \boxed{v} & \boxed{B} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I \end{bmatrix} \cdot \begin{bmatrix} \alpha & w^T \\ 0 & C \end{bmatrix}$$
$$C = B - \frac{v \cdot w^T}{\alpha}$$

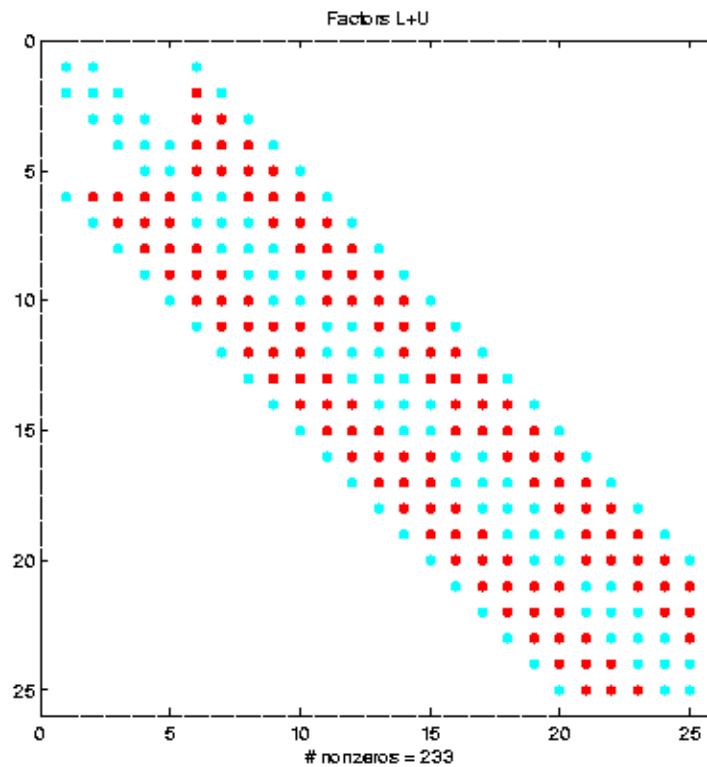
- ◆ Repeats GE on C
- ◆ Results in LU factorization ($A = LU$)
 - L lower triangular with unit diagonal, U upper triangular
- ◆ Then, x is obtained by solving two triangular systems with L and U

Fill-in in Sparse GE

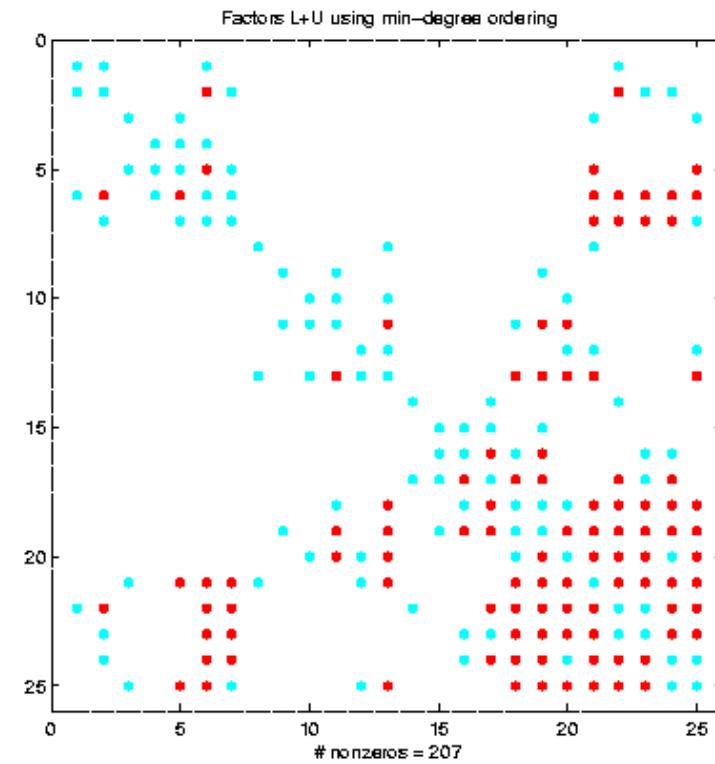


- ◆ Original zero entry A_{ij} becomes nonzero in L or U.

Natural order: nonzeros = 233



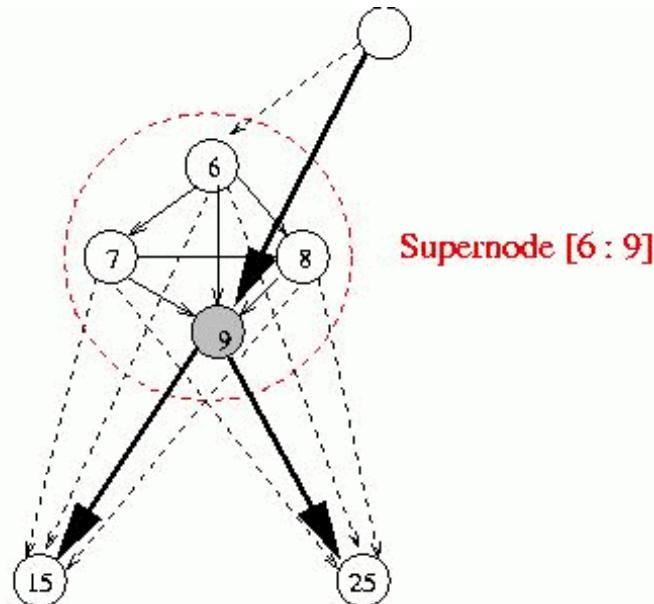
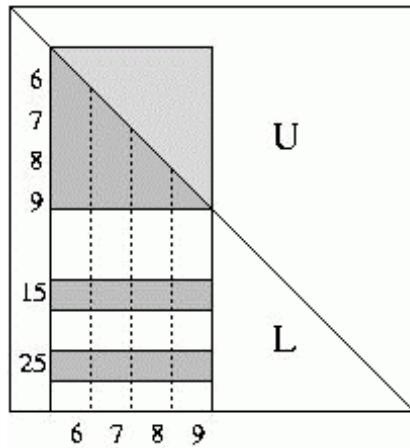
Min. Degree order: nonzeros = 207



Supernode



- ◆ 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

Overview of the Algorithms



- ◆ 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.
- ◆ Parallelization of Steps 1 and 2 are in progress.

- ◆ 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 factor, scale and permute A to maximize diagonal: $P_r D_r A D_c = A'$
 - During factor $A' = LU$, replace tiny pivots by $\sqrt{\varepsilon} \|A\|$, without changing data structures for L & U
 - If needed, use a few steps of iterative refinement after the first solution
 - ➔ Quite stable in practice

	s	x	x
x			
b	x	x	x

Ordering for Sparse Cholesky (1/2)



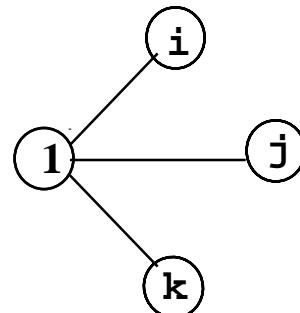
- ◆ Local greedy: **Minimum degree** (upper bound on fill-in)

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

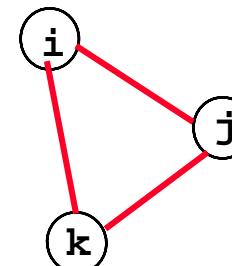
$$\begin{matrix} & \text{i} & \text{j} & \text{k} \\ \text{i} & \times & & \\ \text{j} & & \times & \\ \text{k} & & & \times \end{matrix}$$

Eliminate 1

$$\begin{matrix} & \text{i} & \text{j} & \text{k} \\ \text{i} & \times & & \\ \text{j} & & \bullet & \\ \text{k} & & & \bullet \end{matrix}$$



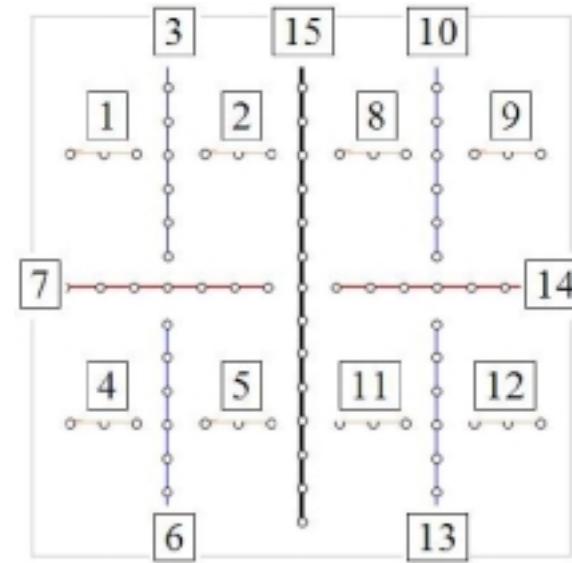
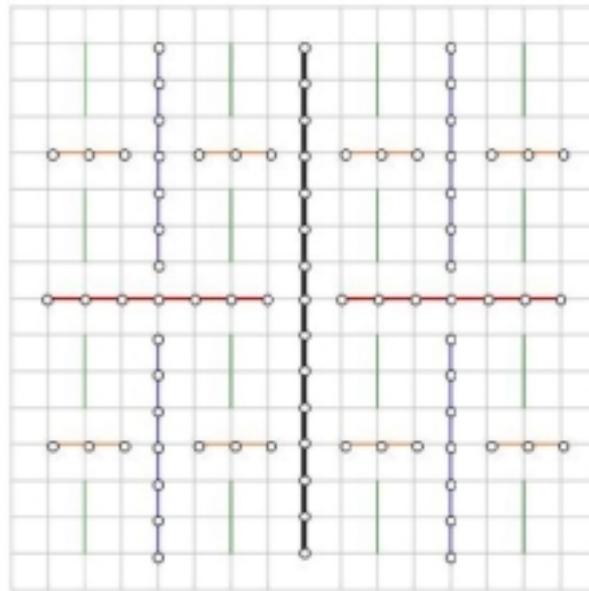
Eliminate 1



Ordering for Sparse Cholesky (2/2)



- ◆ Model problem: discretized system $Ax = b$ from certain PDEs, e.g., 5-point stencil on $n \times n$ grid, $N = n^2$
 - Factorization cost: $O(n^3)$
- ◆ Nested dissection ordering gave optimal complexity in exact arithmetic [George '73, Hoffman/Martin/Ross]

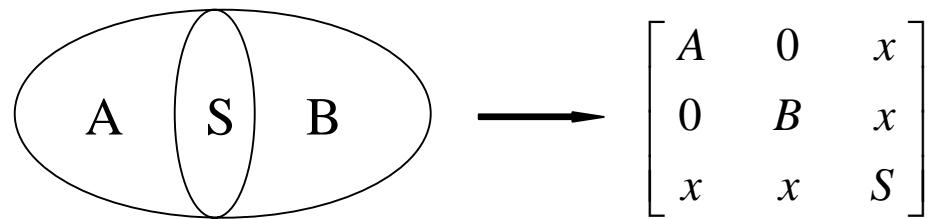


Ordering for Sparse Cholesky (2/2, cont.)



- ◆ Generalized nested dissection [Lipton/Rose/Tarjan '79]

- Global graph partitioning: top-down, divide-and-conquer
 - First level

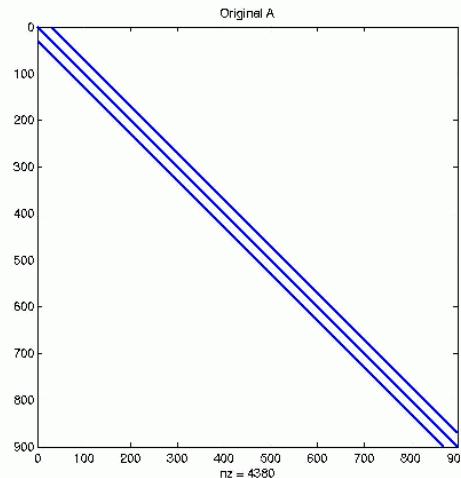


- Recurse on A and B
- ◆ Goal: find the smallest possible separator S at each level
 - Multilevel schemes:
 - Chaco [Hendrickson/Leland '94], Metis [Karypis/Kumar '95]
 - Spectral bisection [Simon et al. '90-'95]
 - Geometric and spectral bisection [Chan/Gilbert/Teng '94]

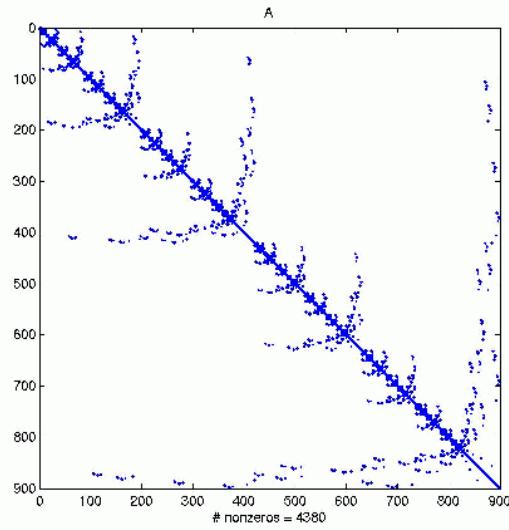
Ordering Based on Nested Dissection



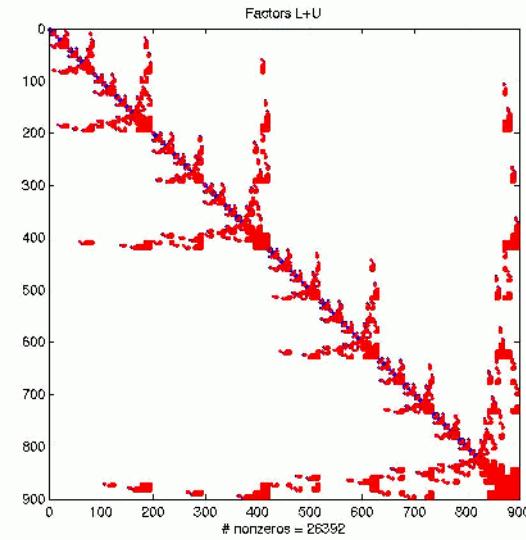
Original A



Permuted A



LU factors



Ordering for LU (unsymmetric)



- ◆ Can use a symmetric ordering on a symmetrized matrix . . .
- ◆ Case of partial pivoting (sequential SuperLU):
Use ordering based on $\mathbf{A}^T \mathbf{A}$
 - If $\mathbf{R}^T \mathbf{R} = \mathbf{A}^T \mathbf{A}$ and $\mathbf{P}\mathbf{A} = \mathbf{L}\mathbf{U}$, then for any row permutation \mathbf{P} ,
 $\text{struct}(\mathbf{L}+\mathbf{U}) \subseteq \text{struct}(\mathbf{R}^T + \mathbf{R})$ [George/Ng '87]
 - Making \mathbf{R} sparse tends to make \mathbf{L} & \mathbf{U} sparse . . .
- ◆ Case of static pivoting (SuperLU_DIST):
Use ordering based on $\mathbf{A}^T + \mathbf{A}$
 - If $\mathbf{R}^T \mathbf{R} = \mathbf{A}^T + \mathbf{A}$ and $\mathbf{A} = \mathbf{L}\mathbf{U}$, then $\text{struct}(\mathbf{L}+\mathbf{U}) \subseteq \text{struct}(\mathbf{R}^T + \mathbf{R})$
 - Making \mathbf{R} sparse tends to make \mathbf{L} & \mathbf{U} sparse . . .
 - Can find better ordering based solely on \mathbf{A} , without symmetrization
[Amestoy/Li/Ng '03]

- ◆ Library contains the following routines:
 - Ordering algorithms: MMD [J. Liu], COLAMD [T. Davis]
 - Utility routines: form $A^T + A$, $A^T A$
- ◆ Users may input any other permutation vector (e.g., using Metis, Chaco, etc.)

```
...
set_default_options_dist ( &options );
options.ColPerm = MY_PERMC; /* modify default option */
ScalePermstructInit ( m, n, &ScalePermstruct );
METIS ( . . . , &ScalePermstruct.perm_c );
...
pdgssvx ( &options, . . . , &ScalePermstruct, . . . );
...
```

Ordering Comparison



		GEPP, COLAMD (SuperLU)	GESP, AMD($A^T + A$) (SuperLU_DIST)
Matrix	N	Fill (10^6)	Fill (10^6)
BBMAT	38744	49.8	40.2
ECL32	51993	73.5	42.7
MEMPLUS	17758	4.4	0.15
TWOTONE	120750	22.6	11.9
WANG4	26068	27.7	10.7

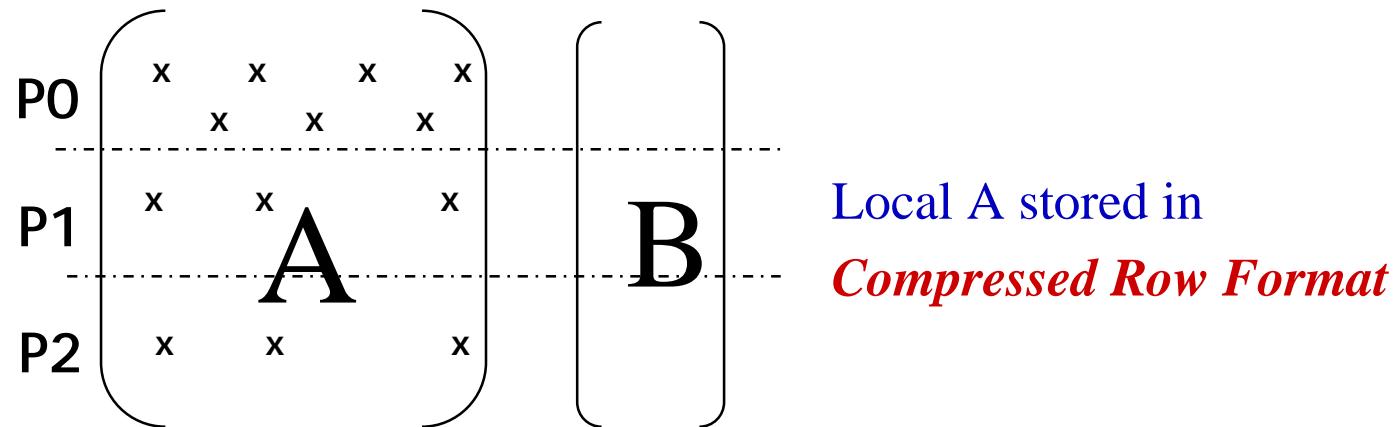
- ◆ Cholesky [George/Liu '81 book]
 - Use elimination graph of L and its transitive reduction (elimination tree)
 - Complexity linear in output: $O(nnz(L))$
- ◆ LU
 - Use elimination graphs of L & U and their transitive reductions (elimination DAGs) [Tarjan/Rose '78, Gilbert/Liu '93, Gilbert '94]
 -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

How to distribute the matrices?



- ◆ Matrices involved:
 - A, B (turned into X) – input, users manipulate them
 - L, U – output, users do not need to see them
- ◆ A (sparse) and B (dense) are distributed by block rows



- Natural for users, and consistent with other popular packages: e.g. PETSc

- ◆ Each process has a structure to store local part of A
(Distributed Compressed Row Format):

```
typedef struct {  
    int_t nnz_loc; /* number of nonzeros in the local submatrix */  
    int_t m_loc;   /* number of rows local to this processor */  
    int_t fst_row; /* global index of the first row */  
    void *nzval;   /* pointer to array of nonzero values, packed by row */  
    int_t *colind; /* pointer to array of column indices of the nonzeros */  
    int_t *rowptr; /* pointer to array of beginning of rows in nzval[]and colind[] */  
} NRformat_loc;
```

A is distributed on 2 processors:

P0	s	u	u	
	1	u		
		1	p	
			e	u
P1	1	1		r

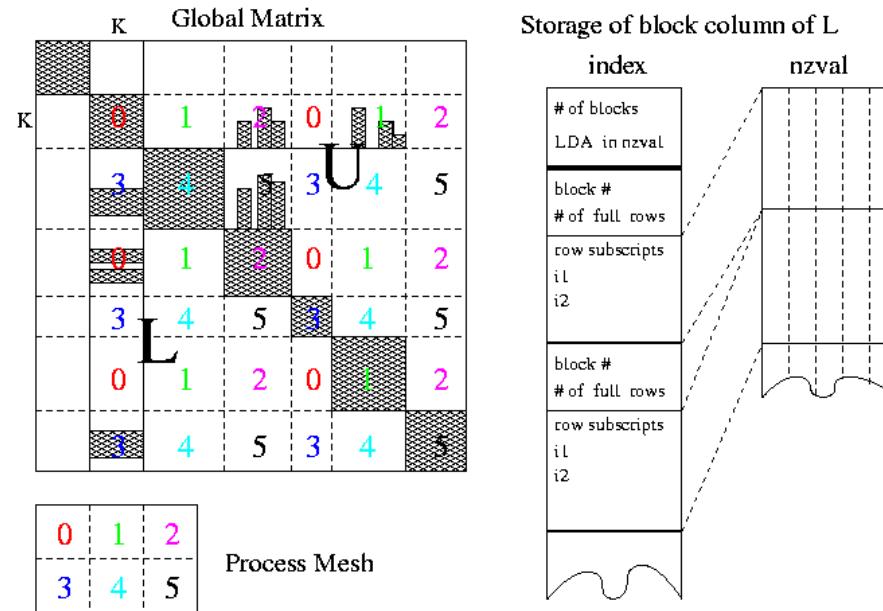
♦ Processor P0 data structure:

- nnz_loc = 5
- m_loc = 2
- fst_row = 0 /* 0-based indexing */
- nzval = { s, u, u, | 1, u }
- colind = { 0, 2, 3, | 0, 1 }
- rowptr = { 0, 3, 5 }

♦ Processor P1 data structure:

- nnz_loc = 7
- m_loc = 3
- fst_row = 2 /* 0-based indexing */
- nzval = { 1, p, | e, u, | 1, 1, r }
- colind = { 1, 2, | 3, 4, | 0, 1, 4 }
- rowptr = { 0, 2, 4, 7 }

2D Block Cyclic Layout for L and U



- ◆ Better for GE scalability, load balance
- ◆ Library has a “re-distribution” phase to 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

Process grid and MPI communicator



- ◆ Example: Solving a preconditioned linear system

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

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

→ use SuperLU_DIST for
each diag. block

0	1		
2	3		
		4	5
		6	7
		8	9
		10	11

- ◆ Need create 3 process grids, same logical ranks (0:3),
but different physical ranks
- ◆ Each grid has its own MPI communicator

Two ways to create a process grid



- ◆ Superlu_gridinit(MPI_Comm Bcomm, int nprow, int npcol, gridinfo_t *grid);
 - Maps the first nprow*npcol processes in the MPI communicator Bcomm to SuperLU 2D grid

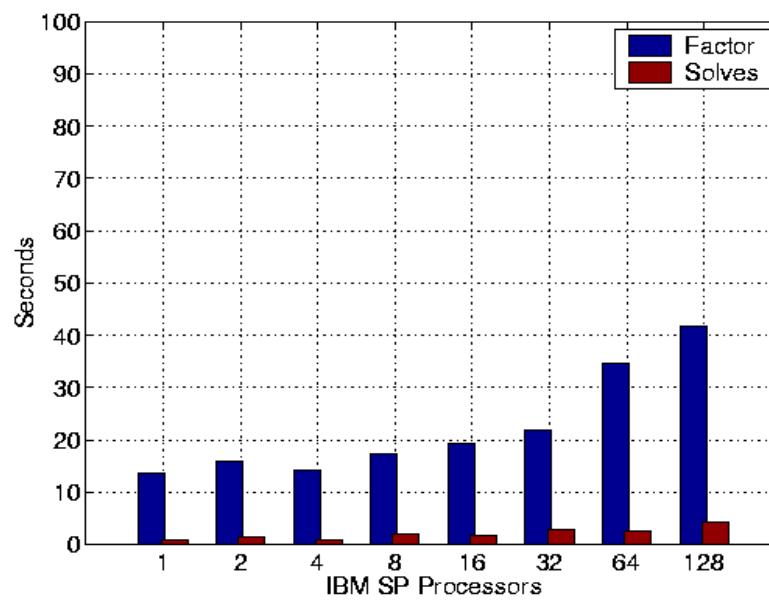
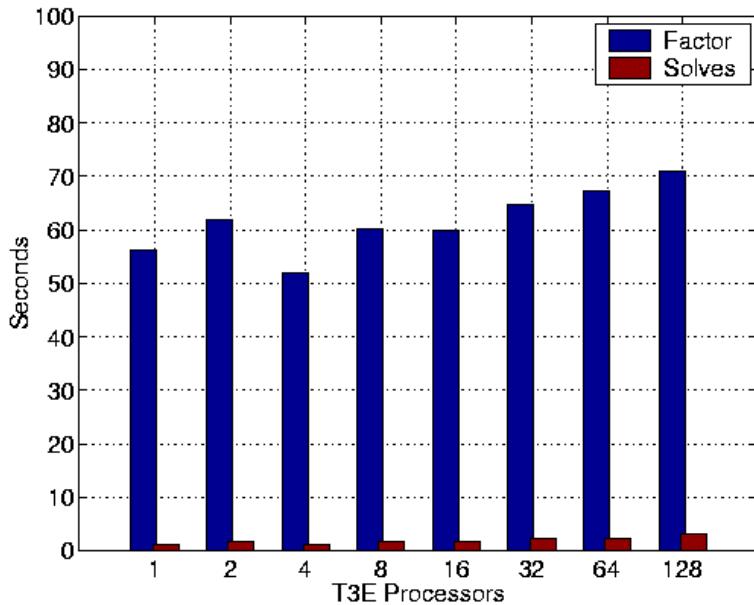
- ◆ Superlu_gridmap(MPI_Comm Bcomm, int nprow, int npcol, int usermap[], int ldumap, gridinfo_t *grid);
 - Maps an *arbitrary* set of nprow*npcol processes in the MPI communicator Bcomm to SuperLU 2D grid. The ranks of the selected MPI processes are given in usermap[] array. For example:

	0	1	2
0	11	12	13
1	14	15	16

Scalability of SuperLU_DIST



- ◆ Poisson's equation on a cube of size $N=n^3$, scale $N^2 = n^6$ with processors 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 diagonally dominant, or SPD, . . .
- ◆ Need good BLAS library (vendor, ATLAS, GOTO)
 - 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?

SuperLU_DIST Example Program



- ◆ SuperLU_DIST_2.0/EXAMPLE/pddrive.c

- ◆ Five basic steps
 1. Initialize the MPI environment and SuperLU process grid
 2. Set up the input matrices A and B
 3. Set the options argument (can modify the default)
 4. Call SuperLU routine PDGSSVX
 5. Release the process grid, deallocate memory, and terminate the MPI environment

EXAMPLE/pddrive.c



```
#include "superlu_ddefs.h"

main(int argc, char *argv[])
{
    superlu_options_t options;
    SuperLUStat_t stat;
    SuperMatrix A;
    ScalePermstruct_t ScalePermstruct;
    LUstruct_t LUstruct;
    SOLVEstruct_t SOLVEstruct;
    gridinfo_t grid;

    . . .

/* Initialize MPI environment */
MPI_Init( &argc, &argv );
. . .

/* Initialize the SuperLU process grid */
nprow = npcol = 2;
superlu_gridinit(MPI_COMM_WORLD, nprow,
                  npcol, &grid);

/* Read matrix A from file, distribute it, and set up the
right-hand side */
dcreate_matrix(&A, nrhs, &b, &ldb, &xtrue, &ldx,
               fp, &grid);

/* Set the options for the solver. Defaults are:
options.Fact = DOFACT;
options.Equil = YES;
options.ColPerm = MMD_AT_PLUS_A;
options.RowPerm = LargeDiag;
options.ReplaceTinyPivot = YES;
options.Trans = NOTRANS;
options.IterRefine = DOUBLE;
options.SolveInitialized = NO;
options.RefineInitialized = NO;
options.PrintStat = YES;
*/
set_default_options_dist(&options);
```

EXAMPLE/pddrive.c (cont.)



```
/* Initialize ScalePermstruct and LUstruct. */
ScalePermstructInit (m, n, &ScalePermstruct);
LUstructInit (m, n, &LUstruct);

/* Initialize the statistics variables. */
PStatInit(&stat);

/* Call the linear equation solver. */
pdgssvx (&options, &A, &ScalePermstruct, b,
          ldb, nrhs, &grid, &LUstruct,
          &SOLVEstruct, berr, &stat, &info );

/* Print the statistics. */
PStatPrint (&options, &stat, &grid);

/* Deallocate storage */
PStatFree (&stat);
Destroy_LU (n, &grid, &LUstruct);
LUstructFree (&LUstruct);
```

- ◆ SuperLU_DIST_2.0/FORTRAN/
- ◆ All SuperLU objects (e.g., LU structure) are **opaque** for F90
 - They are allocated, deallocated and operated in the C side and not directly accessible from Fortran side.
- ◆ C objects are accessed via **handles** that exist in Fortran's user space
- ◆ In Fortran, all handles are of type INTEGER
- ◆ Example:

$$A = \begin{bmatrix} s & u & u \\ l & u & \\ & l & p & \\ & & e & u \\ l & l & & r \end{bmatrix}, \quad s = 19.0, u = 21.0, p = 16.0, e = 5.0, r = 18.0, l = 12.0$$

```
program f_5x5
use superlu_mod
include 'mpif.h'
implicit none
integer maxn, maxnz, maxnrhs
parameter ( maxn = 10, maxnz = 100, maxnrhs
= 10 )
integer colind(maxnz), rowptr(maxn+1)
real*8 nzval(maxnz), b(maxn), berr(maxnrhs)
integer n, m, nnz, nrhs, ldb, i, ierr, info, iam
integer nprow, npcol
integer init
integer nnz_loc, m_loc, fst_row
real*8 s, u, p, e, r, l

integer(superlu_ptr) :: grid
integer(superlu_ptr) :: options
integer(superlu_ptr) :: ScalePermstruct
integer(superlu_ptr) :: LUstruct
integer(superlu_ptr) :: SOLVEstruct
integer(superlu_ptr) :: A
integer(superlu_ptr) :: stat
```

```
! Initialize MPI environment
call mpi_init(ierr)

! Create Fortran handles for the C structures used
! in SuperLU_DIST
call f_create_gridinfo(grid)
call f_create_options(options)
call f_create_ScalePermstruct(ScalePermstruct )
call f_create_LUstruct(LUstruct)
call f_create_SOLVEstruct(SOLVEstruct)
call f_create_SuperMatrix(A)
call f_create_SuperLUStat(stat)

! Initialize the SuperLU_DIST process grid
nprow = 1
npcol = 2
call f_superlu_gridinit
    (MPI_COMM_WORLD,
     nprow, npcol, grid)
call get_GridInfo(grid, iam=iam)
```

f_5x5.f90 (cont.)

```

! Set up the input matrix A
! It is set up to use 2 processors:
! processor 1 contains the first 2 rows
! processor 2 contains the last 3 rows
    m = 5
    n = 5
    nnz = 12
    s = 19.0
    u = 21.0
    p = 16.0
    e = 5.0
    r = 18.0
    l = 12.0
if ( iam == 0 ) then
    nnz_loc = 5
    m_loc = 2
    fst_row = 0      ! 0-based indexing
    nzval (1) = s
    colind (1) = 0   ! 0-based indexing
    nzval (2) = u
    colind (2) = 2
    nzval (3) = u
    colind (3) = 3
    nzval (4) = 1
    colind (4) = 0
    nzval (5) = u
    colind (5) = 1
    rowptr (1) = 0   ! 0-based indexing
    rowptr (2) = 3
    rowptr (3) = 5
else
    nnz_loc = 7
    m_loc = 3
    fst_row = 2      ! 0-based indexing
    nzval (1) = 1
    colind (1) = 1
    nzval (2) = p
    colind (2) = 2
    nzval (3) = e
    colind (3) = 3
    nzval (4) = u
    colind (4) = 4
    nzval (5) = 1
    colind (5) = 0
    nzval (6) = 1
    colind (6) = 1
    nzval (7) = r
    colind (7) = 4
    rowptr (1) = 0   ! 0-based indexing
    rowptr (2) = 2
    rowptr (3) = 4
    rowptr (4) = 7
endif

```

f_5x5.f90 (cont.)



```
! Create the distributed compressed row matrix
! pointed to by the F90 handle
call f_dCreate_CompRowLoc_Matrix_dist
    (A, m, n, nnz_loc, m_loc, fst_row,  &
     nzval, colind, rowptr, SLU_NR_loc, &
     SLU_D, SLU_GE)
! Setup the right hand side
nrhs = 1
call get_CompRowLoc_Matrix
    (A, nrow_loc=ldb)
do i = 1, ldb
    b(i) = 1.0
enddo

! Set the default input options
call f_set_default_options(options)

! Modify one or more options
Call set_superlu_options
    (options,ColPerm=NATURAL)
call set_superlu_options
    (options,RowPerm=NOROWPERM)
```

```
! Initialize ScalePermstruct and LUstruct
call get_SuperMatrix (A,nrow=m,ncol=n)
call f_ScalePermstructInit(m, n, ScalePermstruct)
call f_LUstructInit(m, n, LUstruct)

! Initialize the statistics variables
call f_PStatInit(stat)

! Call the linear equation solver
call f_pdgssvx(options, A, ScalePermstruct, b,
                ldb, nrhs, grid, LUstruct, SOLVEstruct,
                berr, stat, info)

! Deallocate the storage allocated by SuperLU_DIST
call f_PStatFree(stat)
call f_Destroy_SuperMatrix_Store_dist(A)
call f_ScalePermstructFree(ScalePermstruct)
call f_Destroy_LU(n, grid, LUstruct)
call f_LUstructFree(LUstruct)
```

f_5x5.f90 (cont.)



! Release the SuperLU process grid

```
call f_superlu_gridexit(grid)
```

! Deallocate the C structures pointed to by the

! Fortran handles

```
call f_destroy_gridinfo(grid)
```

```
call f_destroy_options(options)
```

```
call f_destroy_ScalePermstruct(ScalePermstruct)
```

```
call f_destroy_LUstruct(LUstruct)
```

```
call f_destroy_SOLVEstruct(SOLVEstruct)
```

```
call f_destroy_SuperMatrix(A)
```

```
call f_destroy_SuperLUDestroy(stat)
```

! Terminate the MPI execution environment

```
call mpi_finalize(ierr)
```

Stop

```
end
```

- ◆ Pddrive1.c:

Solve the systems with same A but different right-hand side.

- Reuse the factored form of A

- ◆ Pddrive2.c:

Solve the systems with the same sparsity pattern of A.

- Reuse the sparsity ordering

- ◆ Pddrive3.c:

Solve the systems with the same sparsity pattern and similar values

- Reuse the sparsity ordering and symbolic factorization

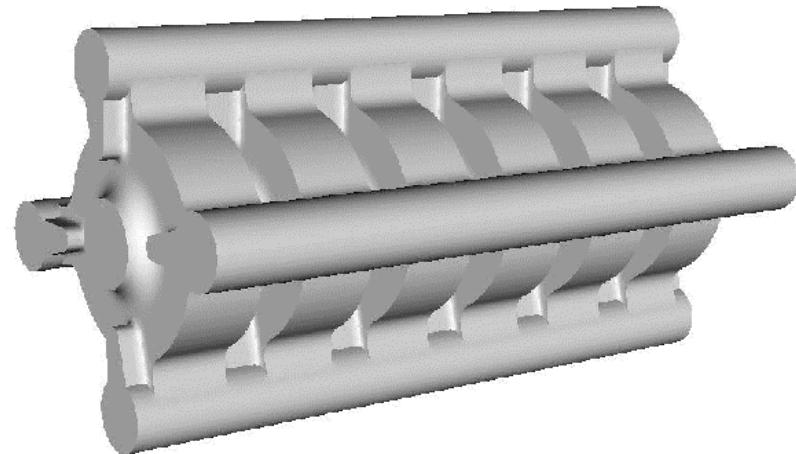
- ◆ Pddrive4.c:

Divide the processes into two subgroups (two grids) such that each subgroup solves a linear system independently from the other.

Application: Accelerator Cavity Design



- ◆ Electromagnetic applications
- ◆ Important for the design of International Linear Accelerator (ILC)
 - Stanford Linear Accelerator Center
- ◆ Curl-curl formulation of Maxwell's equation



$$\nabla \times (\nabla \times \mathbf{E}) - \lambda \mathbf{E} = 0 \text{ in } \Omega$$

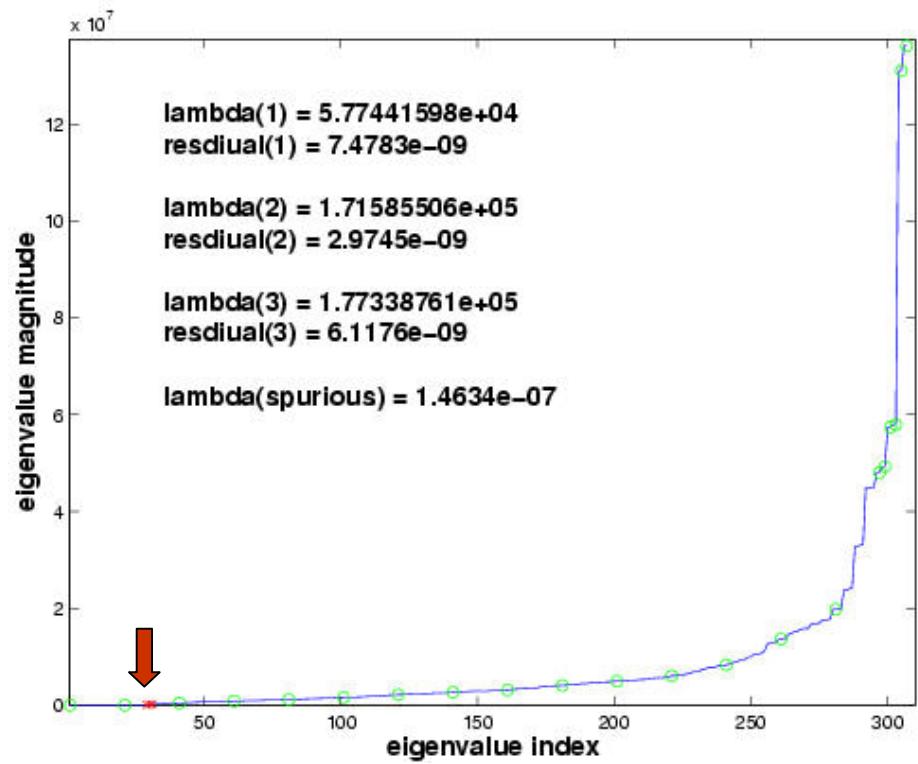
$$n \times \mathbf{E} = 0 \text{ on } \Gamma_E$$

$$n \times (\nabla \times \mathbf{E}) = 0 \text{ on } \Gamma_B$$

Accelerator (cont.)



- ◆ Finite element methods lead to large, sparse generalized eigensystem
 - $\mathbf{K} \mathbf{x} = \lambda \mathbf{M} \mathbf{x}$
- ◆ Real symmetric for closed cavities;
Complex symmetric, nonlinear for open cavities (with external coupling)
- ◆ Seek interior eigenvalues that are relatively small in magnitude



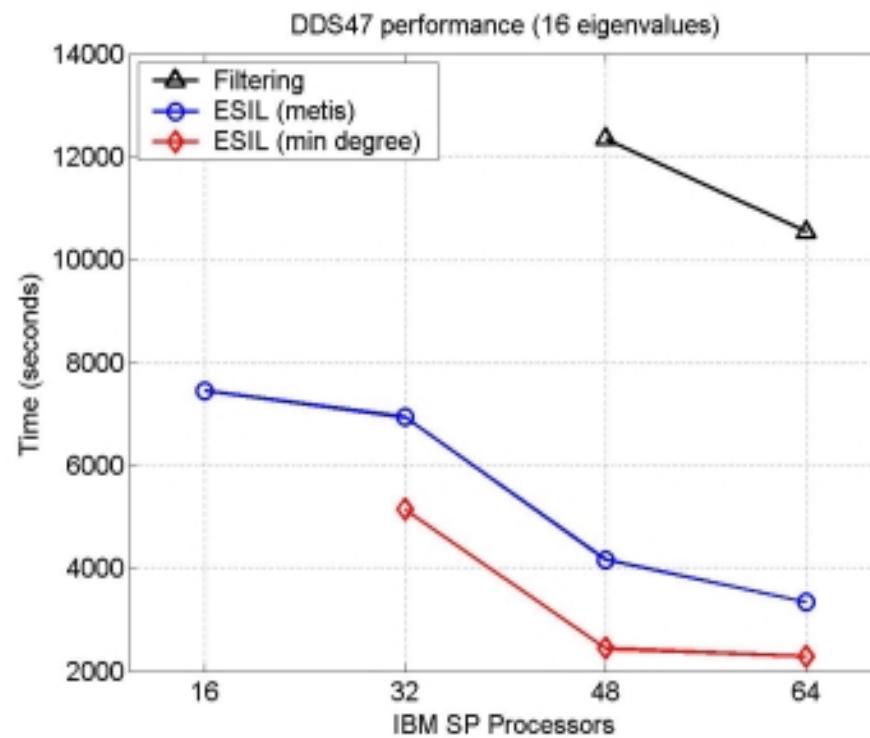
- ◆ Algorithm: Speed up Lanczos convergence by shift-invert
 - ➔ Seek largest eigenvalues, well separated, of the transformed system

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

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

- ◆ Shift-invert Lanczos (SIL)
 - PARPACK: implicitly restarted Arnoldi
 - Parallel SuperLU to solve the shifted linear system

- ◆ Total eigensolver time: $N = 1.3 \text{ M}$, $\text{NNZ} = 20 \text{ M}$



Largest Eigen Problem Solved So Far



- ◆ DDS47, quadratic elements
 - $N = 7.5 \text{ M}$, $\text{NNZ} = 304 \text{ M}$
 - 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

Summary



- ◆ Efficient implementations of sparse LU on high-performance machines
- ◆ More sensitive to latency than dense case
- ◆ Continuing developments funded by DOE TOPS program
 - Integrate into more applications
 - Parallel ordering and symbolic factorization
 - Improve triangular solution
- ◆ Survey of other sparse direct solvers: “Eigentemplates” book
(<http://crd.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf>)
 - LL^T , LDL^T , LU, QR