SIAM Conference on Computational Science and Engineering

Short Course on the ACTS Collection:

Robust and High Performance Libraries for Computational Sciences

SuperLU
(Sparse Direct Solver)

Osni Marques

Lawrence Berkeley National Laboratory (LBNL)

oamarques@lbl.gov

Outline

- Overview of the software
- Some background of the algorithms
- Sparse matrix distribution and user interface
- Example program (Fortran 90 interface)
- Applications

What is SuperLU

- Solve general sparse linear system $A\,x = b$.
  - Example: A of dimension $10^{5,}$ only 10 ~ 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.

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

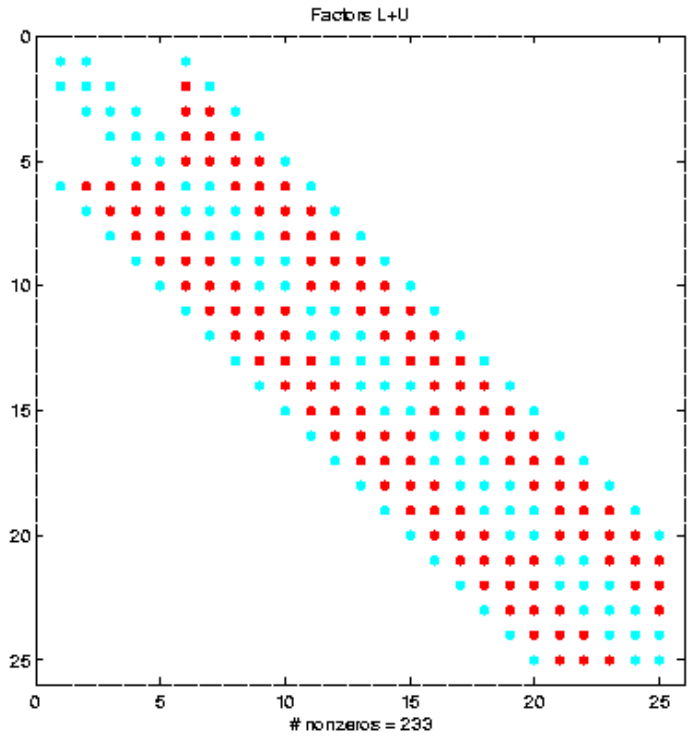
Contents of the SuperLU Library

- 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^TA 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]

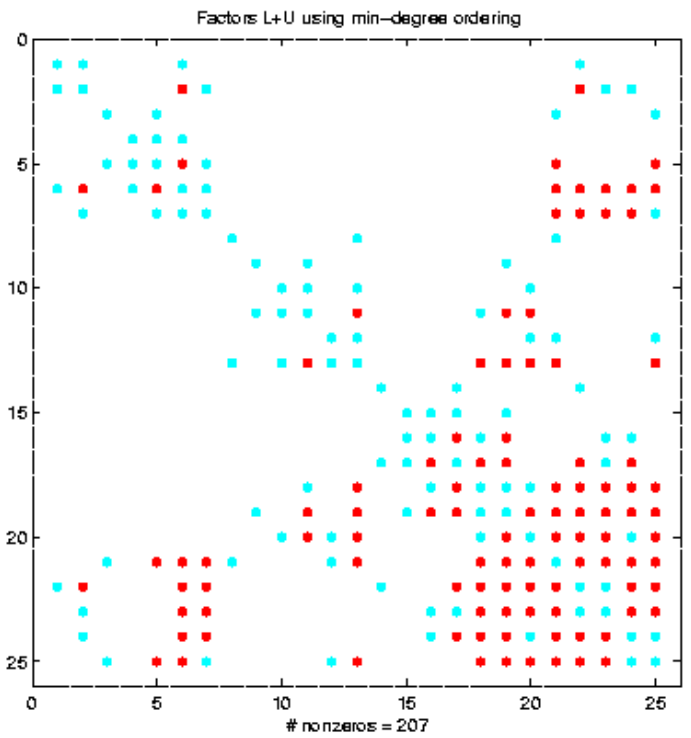
Fill-in in Sparse GE

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

Natural order: nonzeros = 233

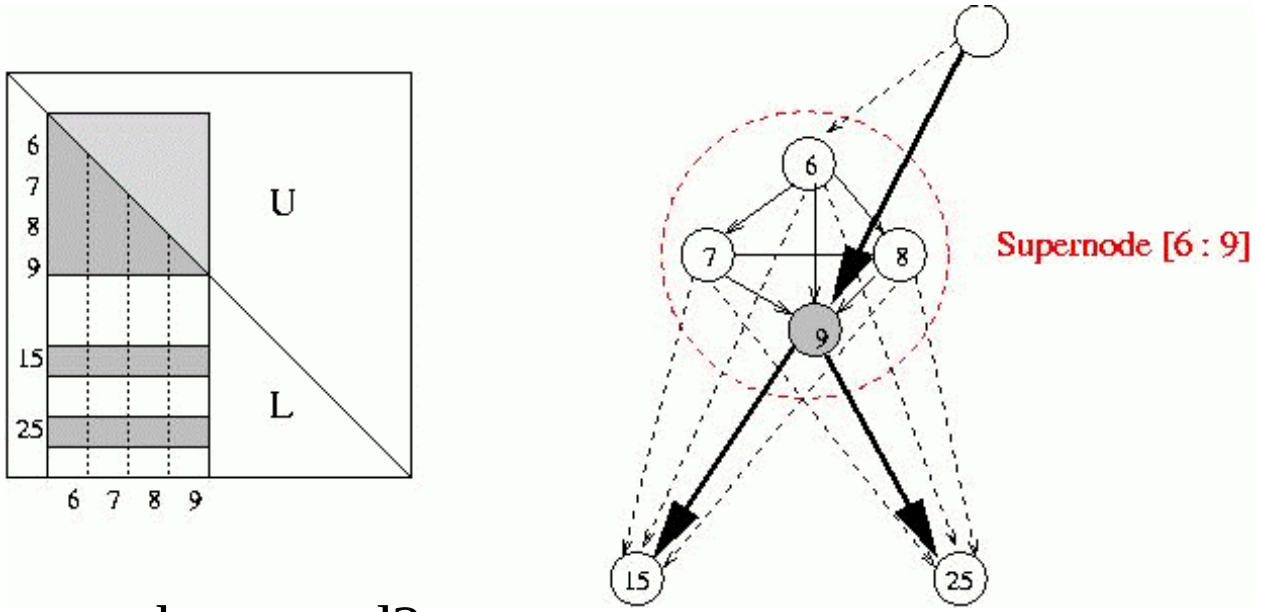


Min. Degree order: nonzeros = 207



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 and 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 and 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

Numerical Pivoting

- Goal of pivoting is to control element growth in L and 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 the data structures for L and U
  - If needed, use a few steps of iterative refinement after the first solution (quite stable in practice)

Ordering for LU (unsymmetric)

- Can use a symmetric ordering on a symmetrized matrix

- Case of partial pivoting (sequential SuperLU):

  Use ordering based on A^TA

  - If $R^TR = A^TA$ and $PA = LU$, then for any row permutation P, struct($L+U$) $\subseteq$ struct(R^T+R) [George/Ng `87]
  - Making R sparse tends to make L and U sparse

- Case of static pivoting (SuperLU_DIST):

  Use ordering based on A^T+A

  - If $R^TR = A^T+A$ and $A = LU$, then struct($L+U$) $\subseteq$ struct(R^T+R)
  - Making R sparse tends to make L and U sparse . . .
  - Can find better ordering based solely on A, without symmetrization [Amestoy/Li/Ng `03]

Ordering Interface in SuperLU

- Library contains the following routines:
  - Ordering algorithms: MMD [J. Liu], COLAMD [T. Davis]
  - Utilities: form A^T+A , A^TA

- 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, . . . );
. . .
```

Symbolic Factorization

- Cholesky [George/Liu `81]
  - Use elimination graph of L and its transitive reduction (elimination tree)
  - Complexity linear in output: O(nnz(L))
- LU
  - Use elimination graphs of L and 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)

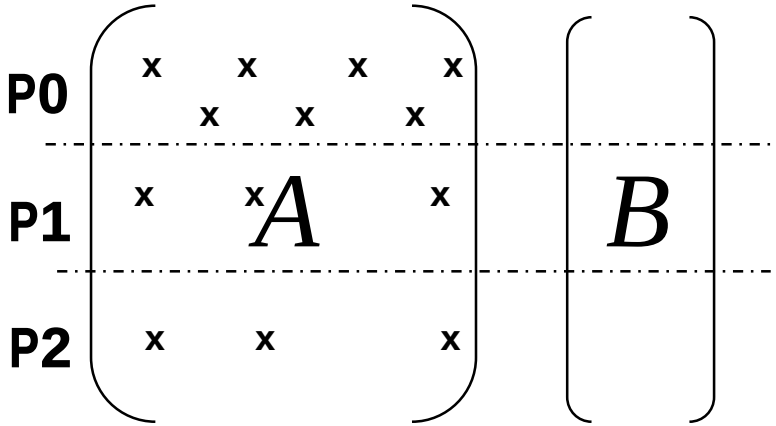
Numerical Factorization

- 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

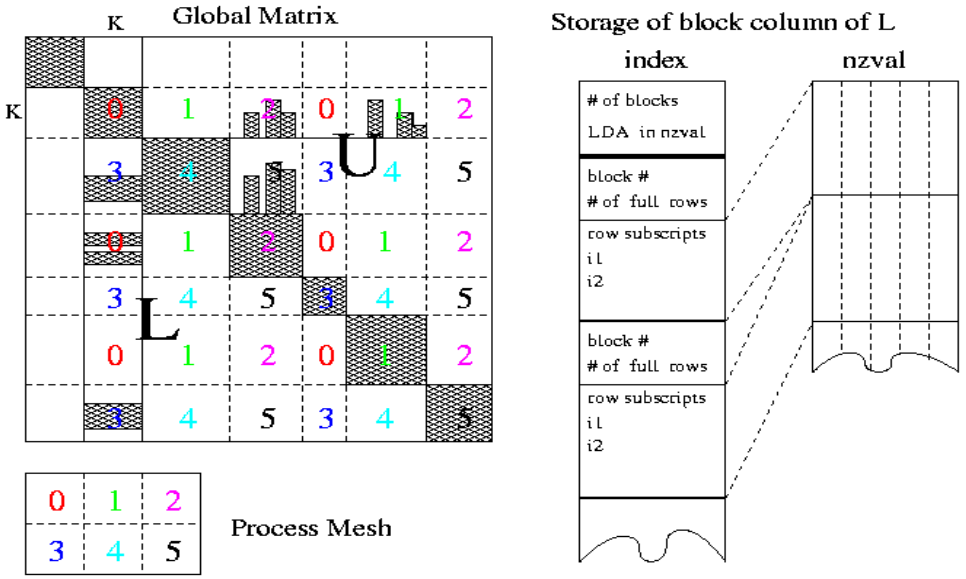


Local A stored in

Compressed Row Format

  - Natural for users, and consistent with other popular packages: PETSc, Aztec, etc.

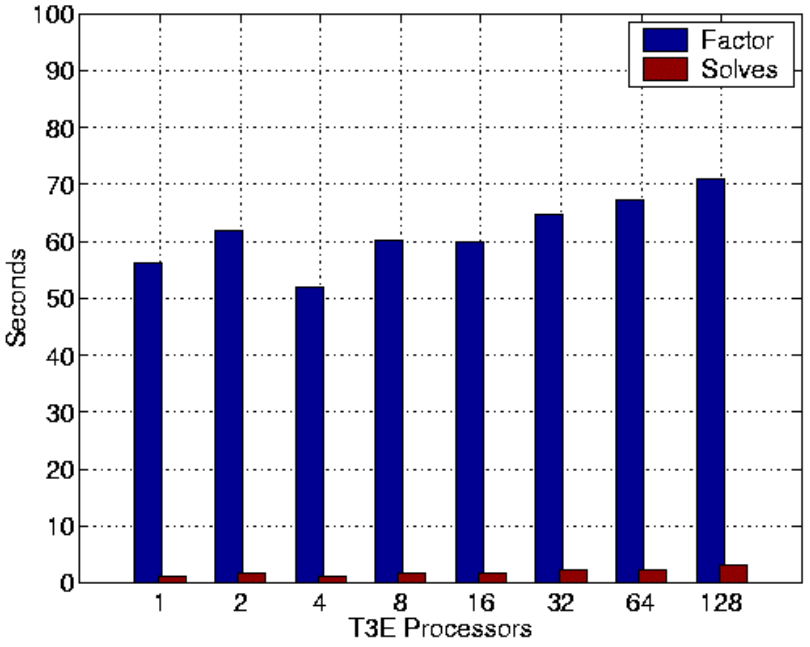
2D Block Cyclic Layout for *L* and *U*

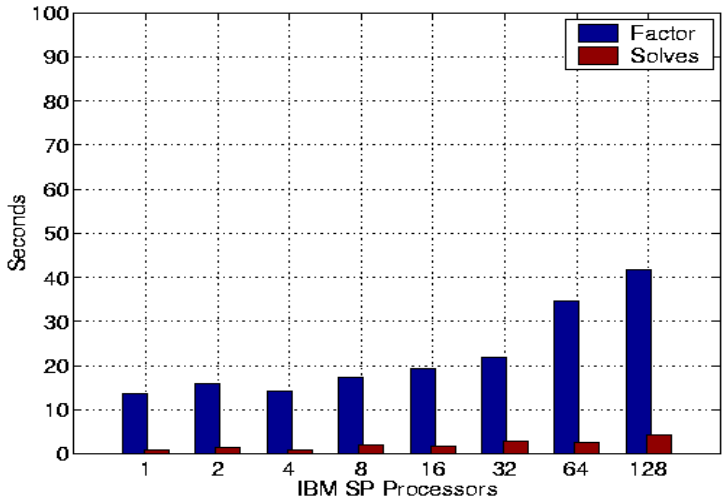


- Better for GE scalability, load balance
- Library has a "re-distribution" phase to distribute the initial values of A to the 2D block-cyclic data structure of *L* and *U*.
  - All-to-all communication, entirely parallel
  - < 10% of total time for most matrices

Scalability

- 3D KxKxK 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)





SuperLU_DIST Example Program (C)

- 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

Pddrive.c (1/2)

```
#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);
```

Pddrive.c (2/2)

```
/* 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);
```

```
   /* Release the SuperLU process grid */
   superlu_gridexit(&grid);

   /* Terminate the MPI execution
   environment */
   MPI_Finalize();

}
```

SuperLU_DIST Example Program (Fortran 90)

- 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

f_pddrive.f90 (1/2)

```
      program f_pddrive
      use superlu_mod
      include 'mpif.h'

! Declarations
      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

! 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)
```

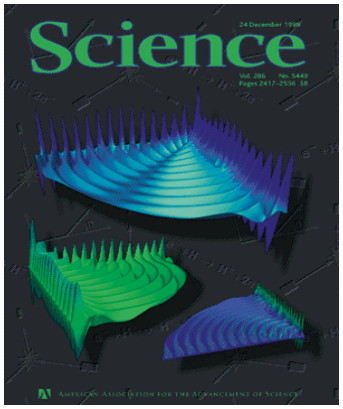
```
! Initialize MPI environment
     call mpi_init(ierr)

! Set up the distributed input matrix A
     call  f_dcreate_dist_matrix(A, m, n, nnz, values,
                rowind, colptr, grid)

! Set the default solver options
      call f_set_default_options(options)

! 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)
```

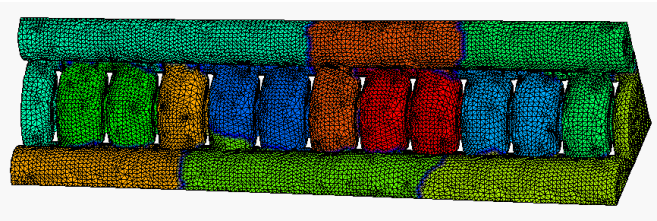
f_pddrive.f90 (2/2)

```
! Deallocate SuperLU allocated storage
      call f_PStatFree(stat)
      call f_ScalePermstructFree(ScalePermstruct)
      call f_Destroy_LU(n, grid, LUstruct)
      call f_LUstructFree(LUstruct)

! Release the SuperLU process grid
       call f_superlu_gridexit(grid)

! Destroy Fortran handles pointing to the C objects
      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_SuperLUStat(stat)

! Terminate the MPI execution environment
      call mpi_finalize(ierr)

      stop
      end
```

Applications

- Used to solve open Quantum Mechanics problem (Science, 24 Dec 1999):
  - *n = 736 K* on 64 PEs,
    Cray T3E in 5.7 minutes
  - *n = 1.8 M* on 24 PEs,
    ASCI Blue Pacific in 24 minutes
- Eigenmodes of accelerator cavities:
  - Quadratic Finite Element discretization (Omega3P)
  - $\boldsymbol{K}\,\boldsymbol{x} = \lambda\,\boldsymbol{M}\,\boldsymbol{x}$, with $\boldsymbol{K}$ and $\boldsymbol{M}$ large, sparse and symmetric.
  - Parallel exact shift-invert eigensolver
  - Problem of size 380698 with 15844364 nonzeros (*npes = 8*)
  - Early tests show that the computation of ~100 eigenvalues is faster than the current eigensolver in the electromagnetic simulation code (which can compute only a few eigenvalues at the moment)





Adoptions of SuperLU

- Industrial
  - Mathematica
  - FEMLAB
  - Python
  - HP Mathematical Library
  - NAG (planned)
- Academic/Lab:
  - In other ACTS Tools: PETSc, Hyper
  - NIMROD (simulate fusion reactor plasmas)
  - Omega3P (accelerator design, SLAC)
  - OpenSees (earthquake simluation, UCB)
  - DSpice (parallel circuit simulation, SNL)
  - Trilinos (object-oriented framework encompassing various solvers, SNL)
  - NIKE (finite element code for structural mechanics, LLNL)

Summary

- Efficient implementations of sparse LU on high-performance machines
- More sensitive to latency than dense case
- Continuing developments funded by DOE/SciDAC/TOPS
  - Integrate into more applications
  - Improve triangular solution
  - Parallel ordering and symbolic factorization
  - ILU preconditioner
- Survey of other sparse direct solvers in "Eigentemplates" book (*www.netlib.org/etemplates*): LL^T, LDL^T, LU
- See also *http://acts.nersc.gov/events/Workshop2004/slides/superlu.pdf* (by Sherry Li) for more details and applications