Elimination Forest Guided 2D Sparse LU Factorization

Kai Shen, Xiangmin Jiao and Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106

Preliminaries

- Gaussian elimination with partial pivoting: used for solving a linear system $Ax = b$ $(PA = LU)$.
- Applications: Newton's method for solving nonlinear systems, Device simulation, Fluid mechanics, Flow simulation, etc.
- Challenges for parallel sparse LU factorization:
  - Poor caching performance due to irregular data distribution.
  - Limited parallelism due to pivoting.
  - High space cost for asynchronous computation.

Summary of related work:

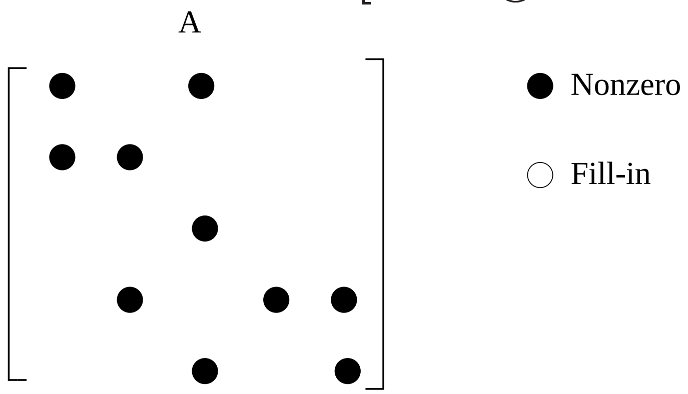
High performance sparse LU codes

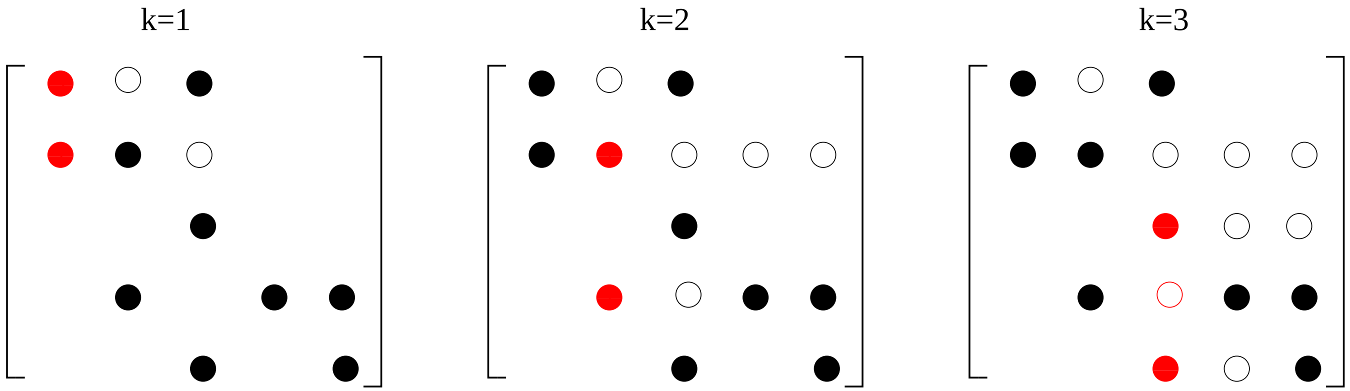
- Sequential machines
  - UMFPACK (Davis and Duff. 93-94)
  - SuperLU (Demmel, Eisenstat, Gilbert, Li and Liu. 95-96)
- Shared memory machines
  - SuperLU (Demmel, Gilbert, Li, 96-97). 2.4Gflops
- Distributed memory machines
  - $S*$ (Fu, Yang, 1996, using RAPID tool). 1.3Gflops
  - $S+$ (Shen, Jiao, Yang, 97-98). 11Gflops

Static symbolic factorization

Purpose: Avoid dynamic space allocation at runtime.

Idea: Statically predict all possible fill-ins and compute the structural bound for L and U factors. [George and Ng, 1987]





Static vs. dynamic symbolic factorization

Static symbolic factorization (S^+)

- More space (10-40% more for most of cases).
- More operations for numerical factorization (1.5-5 times).
- No cost for runtime symbolic factorization and space allocation.

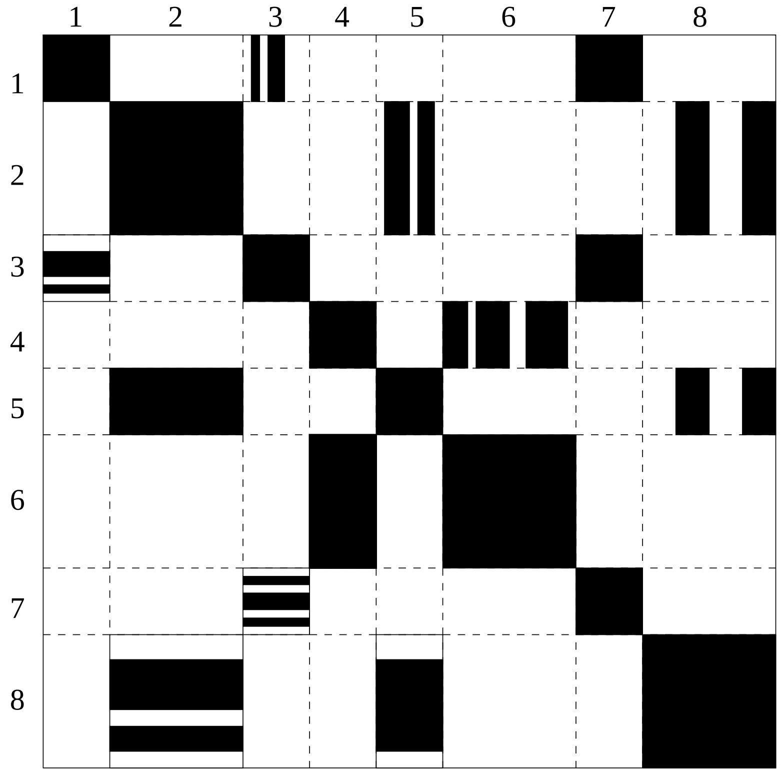
Dynamic symbolic factorization (SuperLU)

- Runtime symbolic processing takes about 45% of the total numerical processing time.

Dense structures identification

Observation: BLAS-3 delivers better cache performance than BLAS-2 (up to 34% time saving on T3E).

Solution: L/U supernode partitioning and amalgamation.



Efficient execution of parallel sparse LU

Sparse LU for a partitioned $N \times N$ matrix A

```
for k = 1 to N
        Perform task Factor(k);
        for j = k+1 to N with A_{k,j} ≠ 0
              Perform task Update(k, j);
        endfor
endfor
```

Efficient execution of parallel sparse LU requires:

- Find parallelism among $Factor()/Update()$ tasks;
- Fast execution of each individual task.

Fast Sparse LU code: S^+

New techniques:

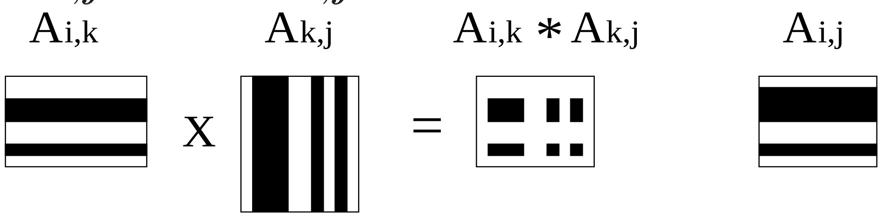
- Supernodal matrix multiplication GEMM to improve the kernel computation speed.
- Elimination forest guided supernode partitioning and amalgamation to identify dense structures.
- Elimination forest guided 2D asynchronous computation scheduling.
  - DAG representation requires more overhead.

Speeding up the kernel computation

Dominating computation in *Update*() tasks:

$A_{i,j} = A_{i,j} - A_{i,k} * A_{k,j}$.

Difficulty for high performance: BLAS-3 GEMM is not directly applicable because of the structural difference between $A_{i,k} * A_{k,j}$ and $A_{i,j}$.

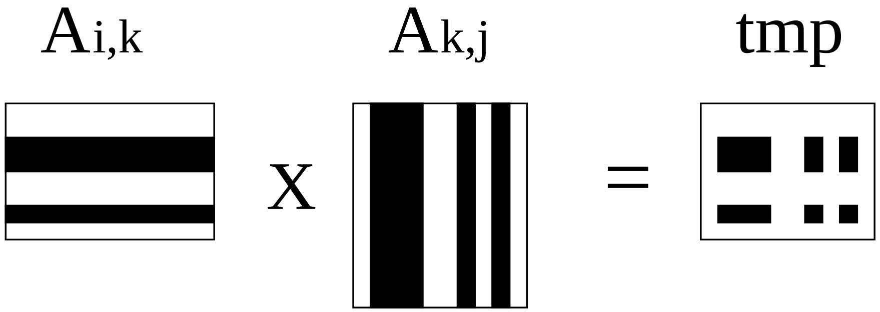


Possible solutions:

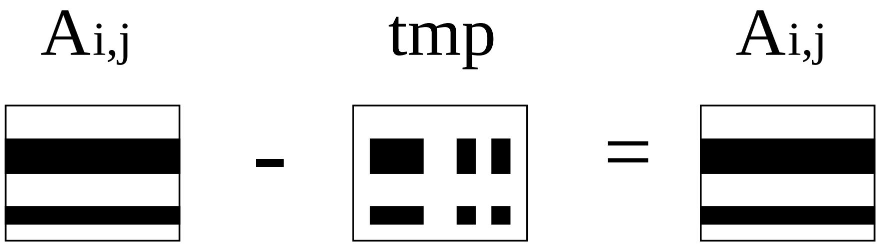
1. Use BLAS-1/BLAS-2 if BLAS-3 is not applicable.
2. Treat those submatrices as dense.

Our solution: Supernodal GEMM

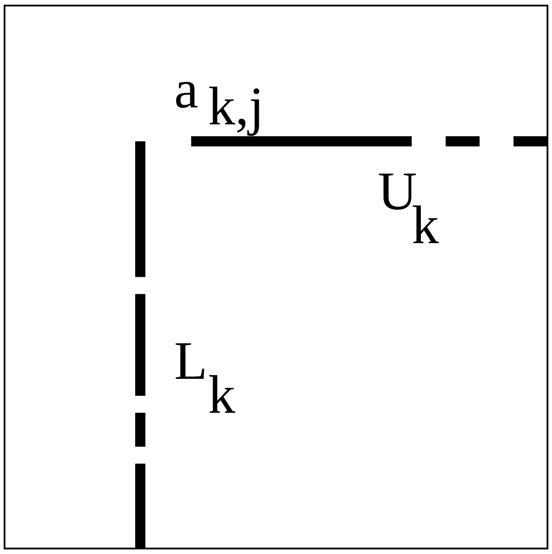
1. Use BLAS-3 GEMM to compute $A_{i,k} * A_{k,j}$. Store the result in a temporal block.



2. Merge the temporal block with $A_{i,j}$ using subtraction.



Elimination forest: Definition

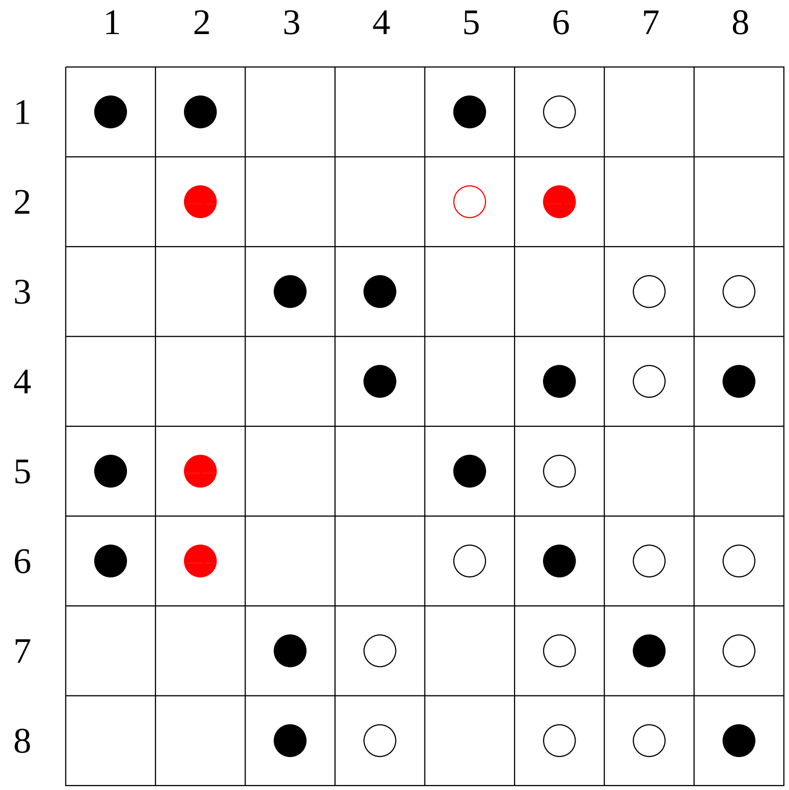


- $a_{k,j}$ — a matrix element after symbolic factorization.
- U_k — k-th row at U part.
- L_k — k-th column at L part.

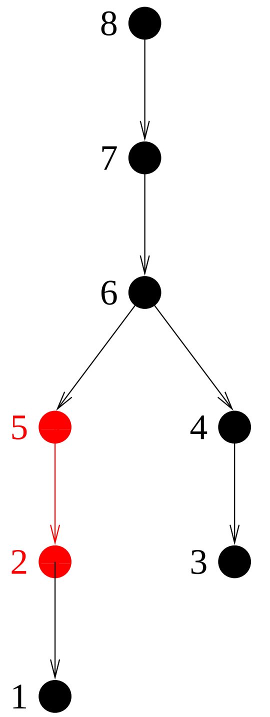
Elimination forest (E-forest):

- Each node corresponds to a column.
- There is an edge from j to k if a_{kj} is the first off-diagonal nonzero in U_k, and $|L_k| > 1$.

Example



Elimination Forest



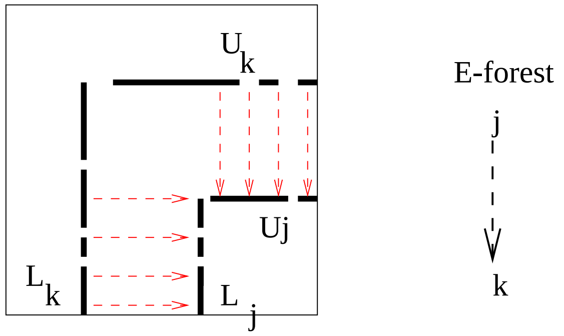
● Nonzeros in the original matrix

○ Fill-in entries generated by symbolic factorization

Why this definition?

- Reveal structural relationship: useful for supernode partitioning/amalgamation.

  Theorem 1: *If j is k's ancestor in the E-forest, $L_k - \{k, k+1, \cdots, j-1\} \subseteq L_j$ and $U_k - \{k, k+1, \cdots, j-1\} \subseteq U_j$.*



- Compact parallelism representation: useful for scheduling of parallel execution.

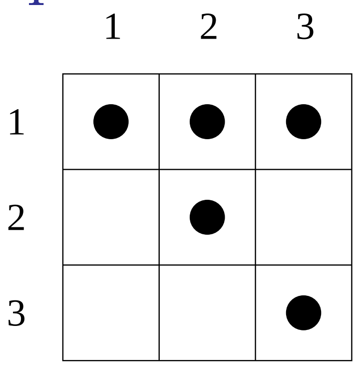
  Theorem 2: *L_k is used to directly or indirectly update L_j if and only if j is an ancestor of k.*

Why not other definitions?

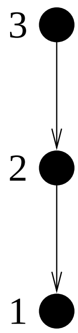
Other choices: Elimination tree of $A^T * A$; George and Ng's elimination forest.

- Not true for Theorems 1 and 2.
- Lead to more false dependence.

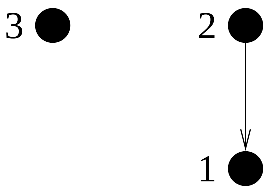
Example:



(a) Sparse matrix A



(b) Elimination tree of A^TA

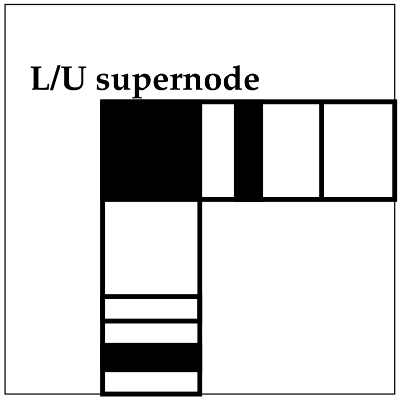


(c) Elimination forest under George and Ng's definition

(d) Elimination forest under our definition

Use E-forest in L/U supernode partitioning

L/U supernode:



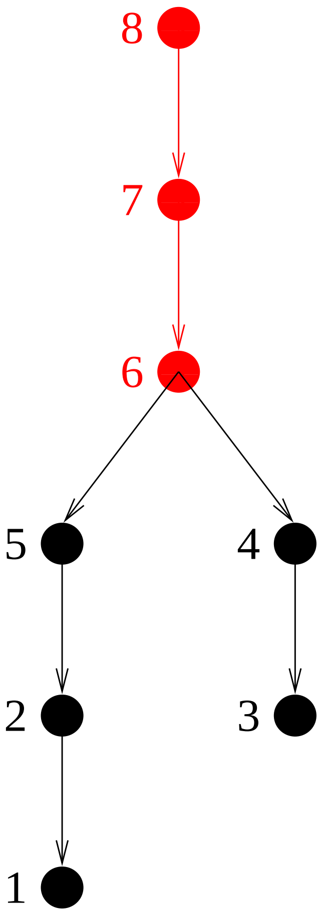
Algorithm: Group i and $i+1$ if i's parent is $i+1$ and $|L_i| = |L_{i+1}| + 1$.

Theorem 3: The produced supernode contains only dense subrows in the L part and dense subcolumns in the U part.

Complexity: $O(n)$ where n is the matrix order.

Example

Elimination Forest



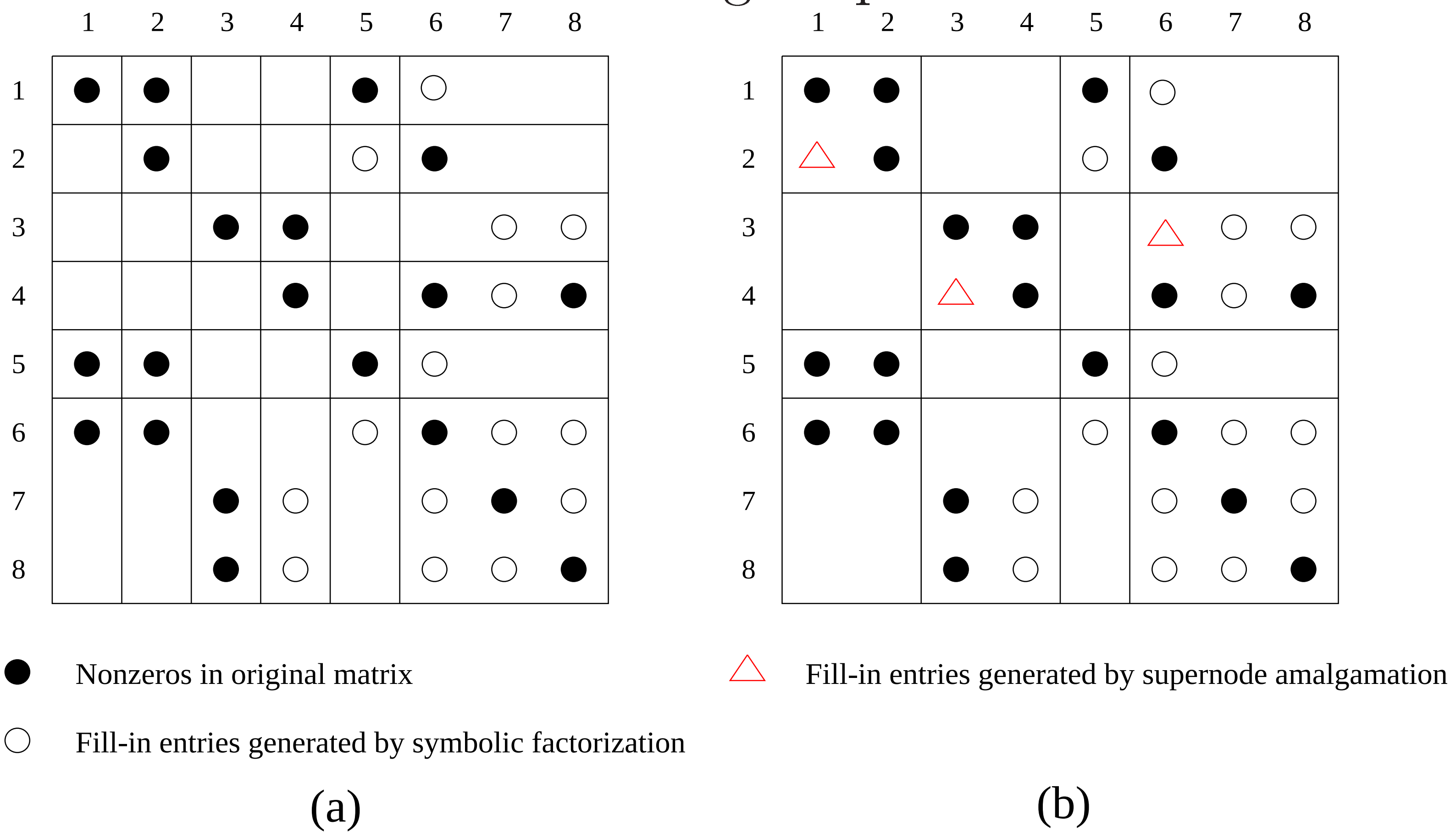
	1	2	3	4	5	6	7	8
1	●	●			●	○		
2		●			○	●		
3			●	●			○	○
4				●		●	○	●
5	●	●			●	○		
6	●	●			○	●	○	○
7			●	○		○	●	○
8			●	○		○	○	●

● Nonzeros in original matrix

○ Fill-in entries generated by symbolic factorization

Use E-forest for supernode amalgamation

Idea: Add more zeros to enlarge supernodes.



Algorithm: Merge i and $i+1$ if $i+1$ is the parent of i and use Theorem 1 to control the number of extra fill-ins.

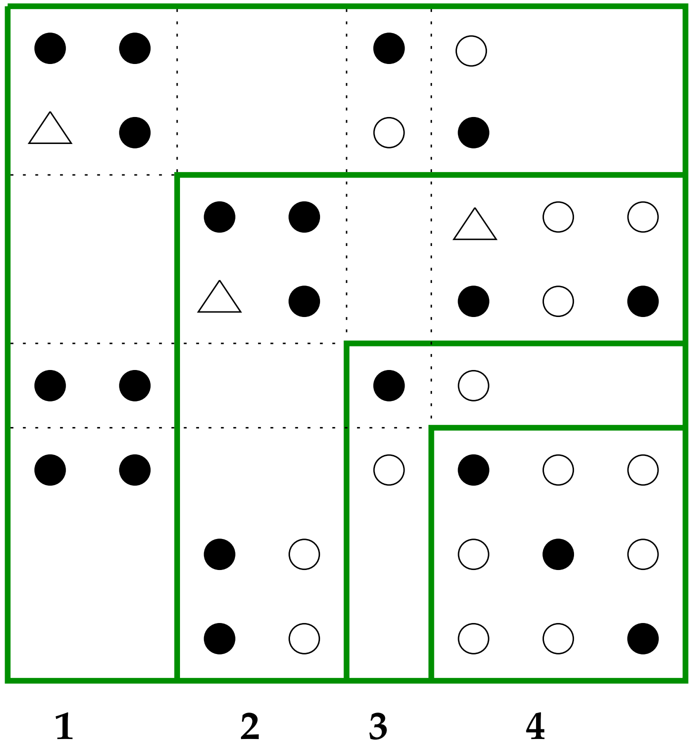
2D Parallel Execution

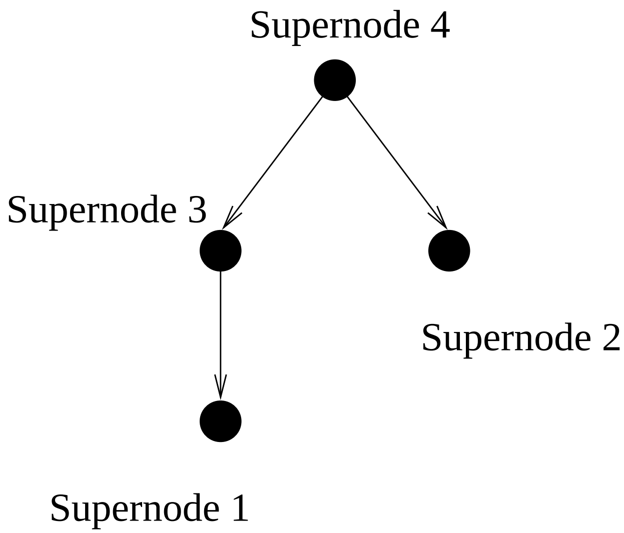
2D data mapping: Submatrices are distributed to 2D processor grid cyclically.

- Advantages over 1D data mapping: More scalable; both Factor(k) and Update(k,j) can be parallelized.
- Difficulties:
  - Need asynchronous execution without global barriers.
  - Pivoting operations and row interchanges require frequent and well-synchronized inter-processor communication.
  - Asynchronous execution requires delicate message buffer management. Space complexity may increase.

Possible parallelism control strategies: Basic asynchronous approach, Factor-ahead approach.

Supernodal E-forest for parallelism control





Theorem 4: Factor(k) can run concurrently with Factor(j) if they are not in the same path.

Key idea: Overlap updating stages and execute Factor() tasks as early as possible.

Overall sequential performance

Experiments are done at a single node of 300Mhz Cray T3E.

We use the operation count reported by SuperLU.

Megaflop count does not include extra operation introduced by static symbolic factorization.

Matrix	Sequential S^+		SuperLU		Sequential S^*		Exec. Time Ratio	
	Time	Mflops	Time	Mflops	Time	Mflops	$\frac{S^+}{SuperLU}$	$\frac{S^+}{S^*}$
sherman5	0.65	38.88	0.78	32.4	0.94	26.9	0.83	0.69
lnsp3937	1.48	28.52	1.73	24.4	2.0	21.1	0.86	0.74
lns3937	1.58	28.30	1.84	24.3	2.19	20.4	0.86	0.72
sherman3	1.56	39.52	1.68	36.7	2.03	30.4	0.93	0.77
jpwh991	0.52	33.38	0.56	31.0	0.69	25.2	0.93	0.75
orsreg1	1.60	39.11	1.53	40.9	2.04	30.7	1.05	0.78
saylr4	2.67	40.10	2.69	39.8	3.53	30.3	0.99	0.76
goodwin	10.26	65.28	-	-	17.0	39.4	-	0.60

Megaflop performance of S^+

on 300MHz Cray T3E

Matrix	P=8	P=16	P=32	P=64	P=128
goodwin	403.5	603.4	736.0	797.3	826.8
e40r0100	443.2	727.8	992.8	1204.8	1272.8
raefsky4	568.2	1072.5	1930.3	3398.1	5133.6
inaccura	495.5	803.8	1203.6	1627.6	1921.7
af23560	432.1	753.2	1161.3	1518.9	1844.7
fidap011	811.2	1522.8	2625.0	4247.6	6248.4
vavasis3	958.4	1864.8	3303.6	5640.4	8441.9

Megaflop rate of GEMM: 388Mflops.

Parallel performance of S^+ on 450MHz Cray T3E

Matrix	P=8		P=32		P=128	
	Time	Mflops	Time	Mflops	Time	Mflops
goodwin	1.21	553.5	0.69	970.6	0.67	999.6
e40r0100	4.06	611.3	1.87	1327.2	1.59	1560.9
raefsky4	38.62	804.2	11.54	2691.5	4.55	6826.0
inaccura	6.56	697.2	2.80	1633.4	1.91	2394.6
af23560	10.57	602.1	4.06	1567.5	2.80	2272.9
fidap011	21.58	1149.5	6.81	3642.7	3.04	8159.9
vavasis3	62.68	1423.6	19.26	4632.9	8.08	11043.5

Megaflop rate of GEMM: 464Mflops.

Effectiveness of proposed techniques

1. Supernodal GEMM:
   - 6-10% improvement over the BLAS-1-2-3 mixed approach.
2. E-forest guided partitioning/amalgamation:
   - 10-35% improvement over the previous method (only examine structures in L part).
3. E-forest guided scheduling:
   - 16-40% improvement over the basic asynchronous approach;
   - 6-17% improvement over the factor-ahead approach.

Conclusions

Contributions:

- Fastest parallel sparse *LU* factorization on distributed memory machines with reasonable space cost.
  Performance record: up to 11.04Gflops on 128 nodes 450MHz Cray T3E.
- Supernodal GEMM for improving kernel computation speed.
- E-forest guided matrix partitioning and parallel execution.

More information:

- http://www.cs.ucsb.edu/research/RAPID.html