A Comparison of 1-D and 2-D Data Mapping for Sparse LU Factorization with Partial Pivoting

Cong Fu, **Xiangmin Jiao** and **Tao Yang**
Department of Computer Science
University of California
Santa Barbara, CA 93106

Difficulties in parallelizing sparse LU

- Data structure variations due to partial pivoting.
- Unpredictable dependency structures and processor loads.
- Dynamic scheduling has high run-time overhead compared to relatively fine grain computation.
- Exploit memory hierarchy and improve the cache performance.

$\Longrightarrow$ **An open problem.**

Related research on Sparse LU

- Unsymmetric multi-frontal method (Davis, Duff).
- SuperLU (Demmel, Eisenstat, Gilbert, Li and Liu), parallelized on shared memory machines, but not on distributed memory machines (DMM).
- Static symbolic factorization (George and Ng). Implemented on a shared memory machine.

No efficient implementation of sparse LU on DMMs.

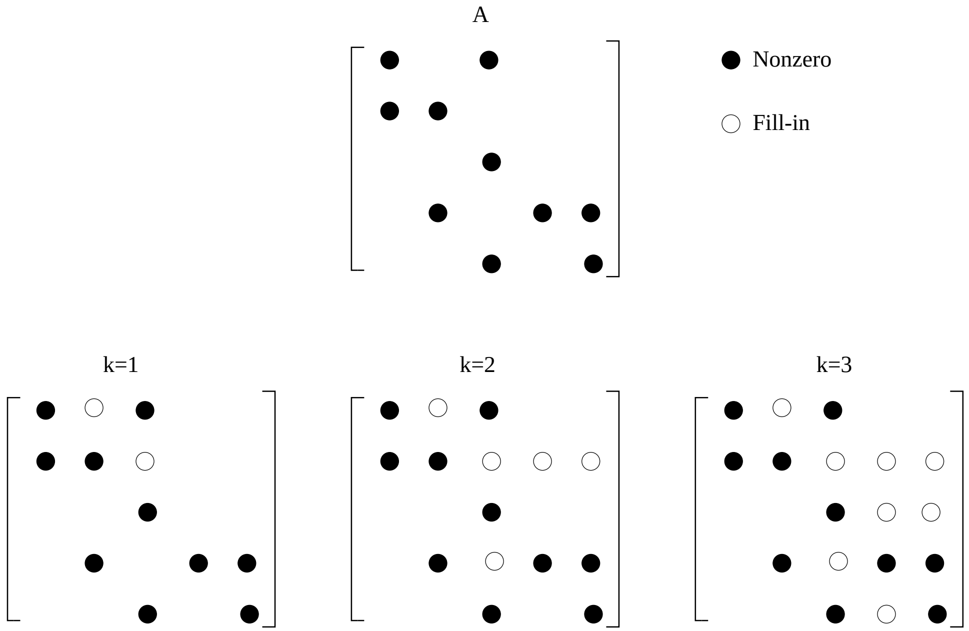
The S^* approach

[SC'96]

- **Avoid data structure variation** by using a static symbolic factorization scheme.
- **Improve cache performance** using nonsymmetric L/U supernode partitioning techniques to effectively identify dense structures.
- **Exploit irregular parallelism** by using graph scheduling techniques and efficient run-time support (the RAPID system).

Static symbolic factorization

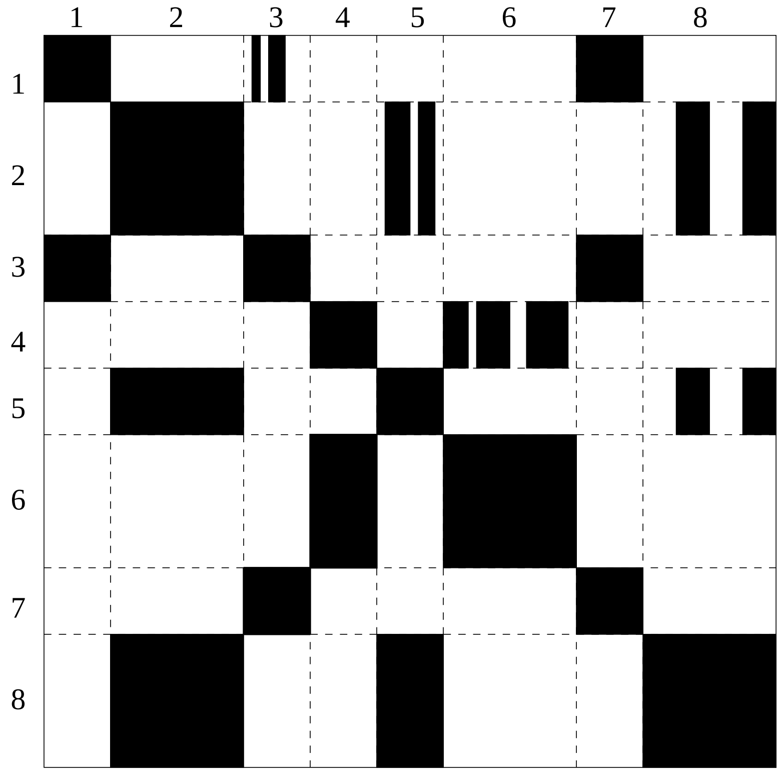
Idea: [George and Ng, 1987] consider all possible pivoting choices at each step.



Over-estimation is acceptable for most of test matrices (within 30-50% more non-zeros).

L/U supernode partitioning

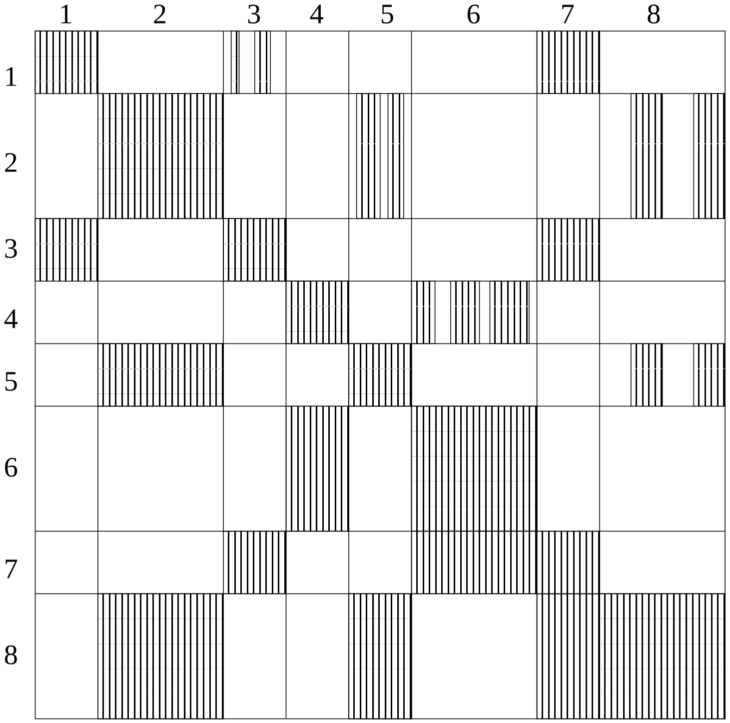
Identify dense structures in both L and U factors so that BLAS-3 operations can be used.



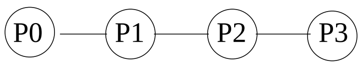
Data mapping

1-D panel based mapping.

- localize pivoting operations and row interchanges.
- reduce number of messages by aggregating the blocks.
- produce coarse grain tasks.

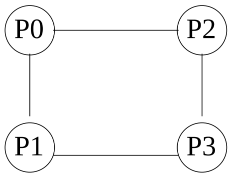


(a)



	1	2	3	4	5	6	7	8
1	P0		P2				P2	
2		P1			P0			P3
3	P0		P2				P2	
4				P3		P1		
5		P1			P0			P3
6				P3		P1		
7			P2			P1	P2	
8		P1			P0		P2	P3

(b)



	1	2	3	4	5	6	7	8
1	P0		P0				P0	
2		P3			P1			P3
3	P0		P0				P0	
4				P3		P3		
5		P2			P0			P2
6				P3		P3		
7			P0			P2	P0	
8		P3			P1		P1	P3

(c)

2-D block cyclic mapping

More scalable, but with following difficulties:

- Pivoting operations and row interchanges require frequent and well-synchronized inter-processor communication.
- Asynchronous parallelism requires delicate message buffer management. Space complexity may increase.
- Difficult to model 2-D irregular parallelism from sparse LU.

Key tasks in Sparse LU

for $k = 1$ **to** N

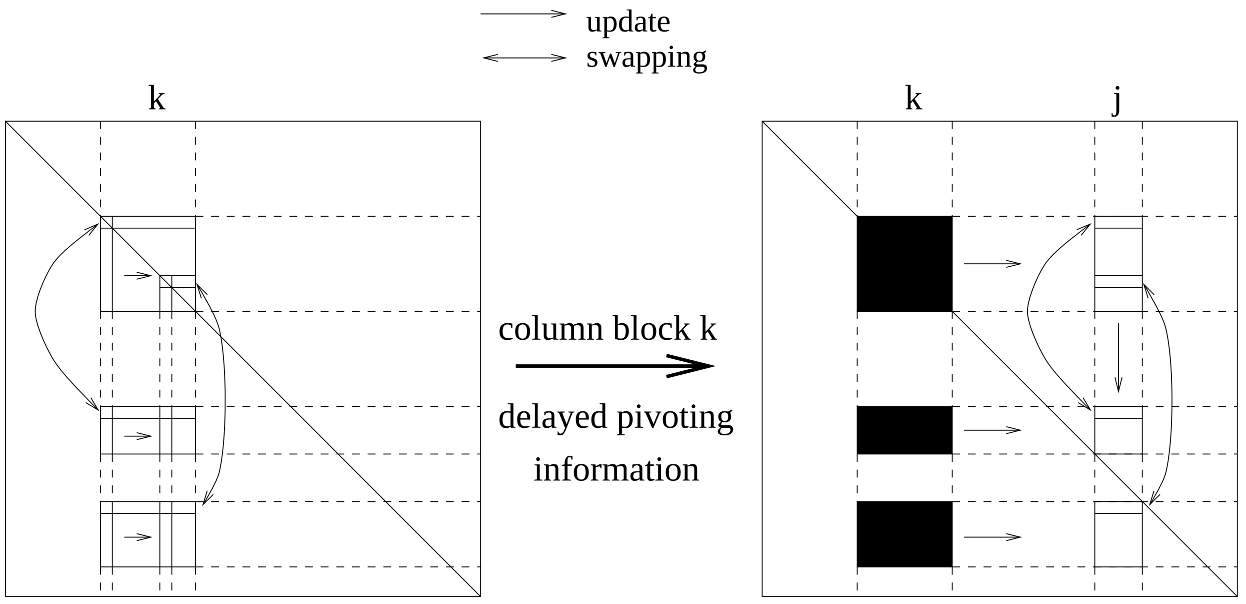
  Task $Factor(k)$ (Perform pivoting/updating on panel k);

  for $j = k + 1$ **to** N with $\mathcal{B}(k, j) \neq 0$

    Task $Update(k, j)$(Updating panel j using panel k);

  endfor

endfor



Factor(k): pivoting & local updating

Update(k,j): swapping and updating

Parallelism scheduling

1-D panel based mapping.

- **Compute-ahead** algorithm (CA).

  Pivoting process proceeds in the order of k. $Update()$ tasks can be overlapped.

- **Graph-based scheduling.** Exploit more parallelism, but use more memory because of out-of-order execution. Use a DAG to model parallelism, and use RAPID to schedule tasks.

2-D block cyclic mapping. Synchronous/asynchronous.

The RAPID run-time system

RAPID provides an API and run-time support for developing efficient irregular codes [ICS'96]:

1. User specifies shared data objects, tasks and data access patterns.
2. RAPID performs dependence analysis, task scheduling.
3. Its run-time support provides an efficient task execution protocol.

- Use asynchronous fast remote memory access for copying/buffering free communication.
- Optimize communication. e.g. Eliminate redundant sending and unnecessary synchronizations.

An asynchronous 2-D algorithm

Assume that the processor grid is of size $\sqrt{p} \times \sqrt{p}$.

Goal:

1. Use asynchronous execution and eliminate global barriers to maximize concurrency.
2. Control the complexity of buffer space.

• The **computation flow** is controlled by pivoting tasks $Factor(k)$. $Update()$ tasks can be overlapped in a maximum degree of $\sqrt{p}$.

• **Pivoting** within $Factor(k)$ is parallelized across a column of processors.

• The **updating** tasks are done concurrently.

A comparison 1-D and 2-D data mapping

Assume:

- N – The number of matrix panels.
- X – the number of nonzero lower triangular submatrices.
- Y – the number of nonzero upper triangular submatrices.
- Z – Total number of nonzero blocks. $Z = X + Y + N$
- b – submatrix block size.

For simplicity: assume: $Y \approx X \approx Z/2$ and $X, Y >> N$.

	1-D	2-D
Parallelism	O(N)	O(Z)
Extra buffer space	$\min(pNb^2, Zb^2)$	$2Nb^2$
Communication volume	$0.5pZb^2$	$\sqrt{p}Zb^2$
Number of messages	Nbp	$\sqrt{p}Z$

2-D approach is potentially more scalable, with more frequent communication.

Experiments

- Cray-T3D in SDSC, each node with 64Mbytes memory.
- Communication bandwidth: 126 MB/S, shmem_put() overhead: $2.7\mu s$.
- BLAS-3 DGEMM: 103 MFLOPS, BLAS-2 DGEMV: 85 MFLOPS, BLAS-1: 25 MFLOPS.

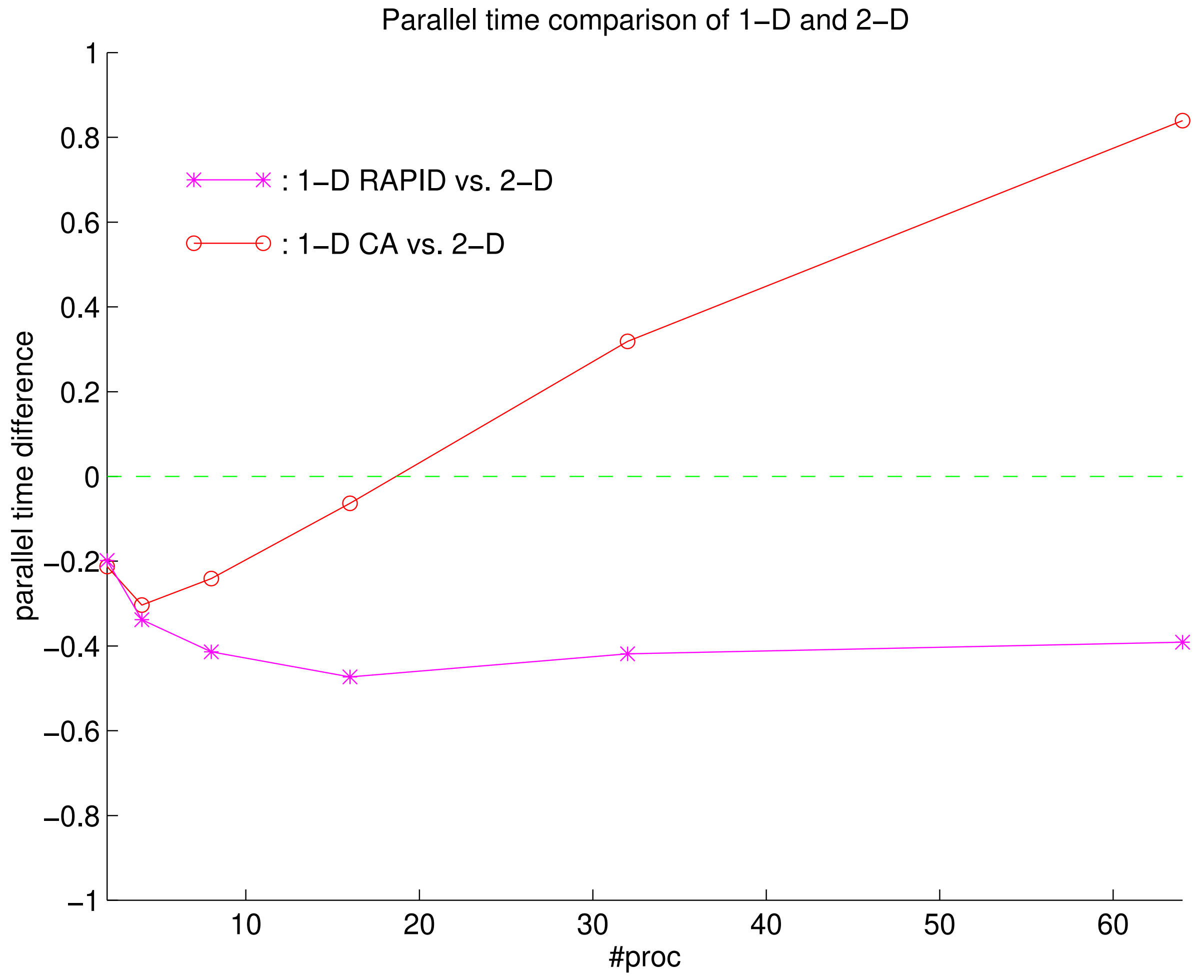
Recent megaflop performance results

2-D asynchronous algorithm.

	Row	NZ(A)	Ratio	P=16	P=32	P=64
goodwin	7320	324772	1.62	110.67	145.22	184.75
e40r0100	17281	553562	1.35	116.62	162.46	235.35
ex11	16614	1096948	1.35	304.99	501.77	802.59
raefsky4	19779	1316789	1.79	242.93	413.83	719.24
vavasis3	41092	1683902	1.31	525.35	885.1	1480.2

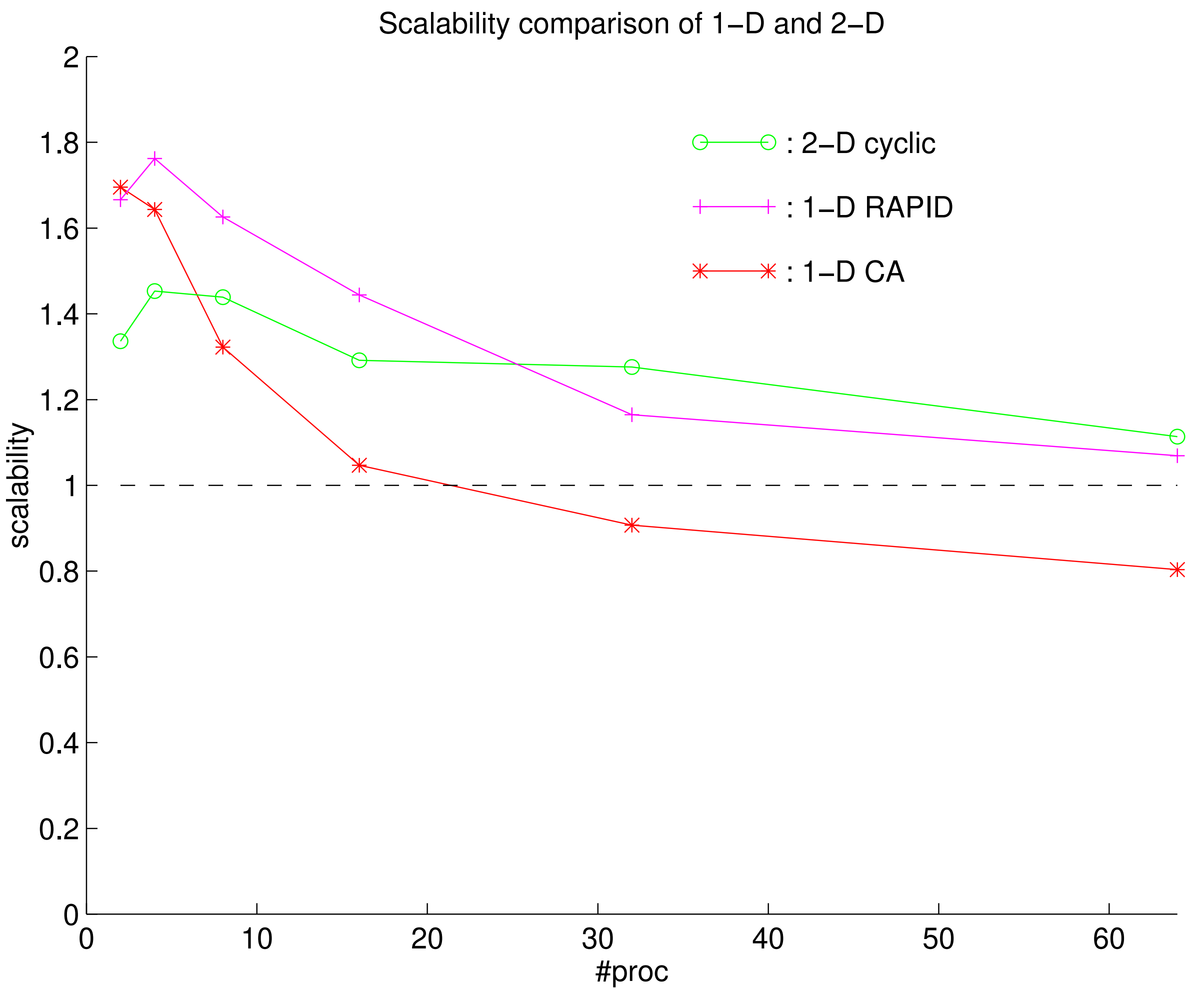
Per node performance up to 23.1 MFLOPS on 64 nodes.

Parallel time comparison



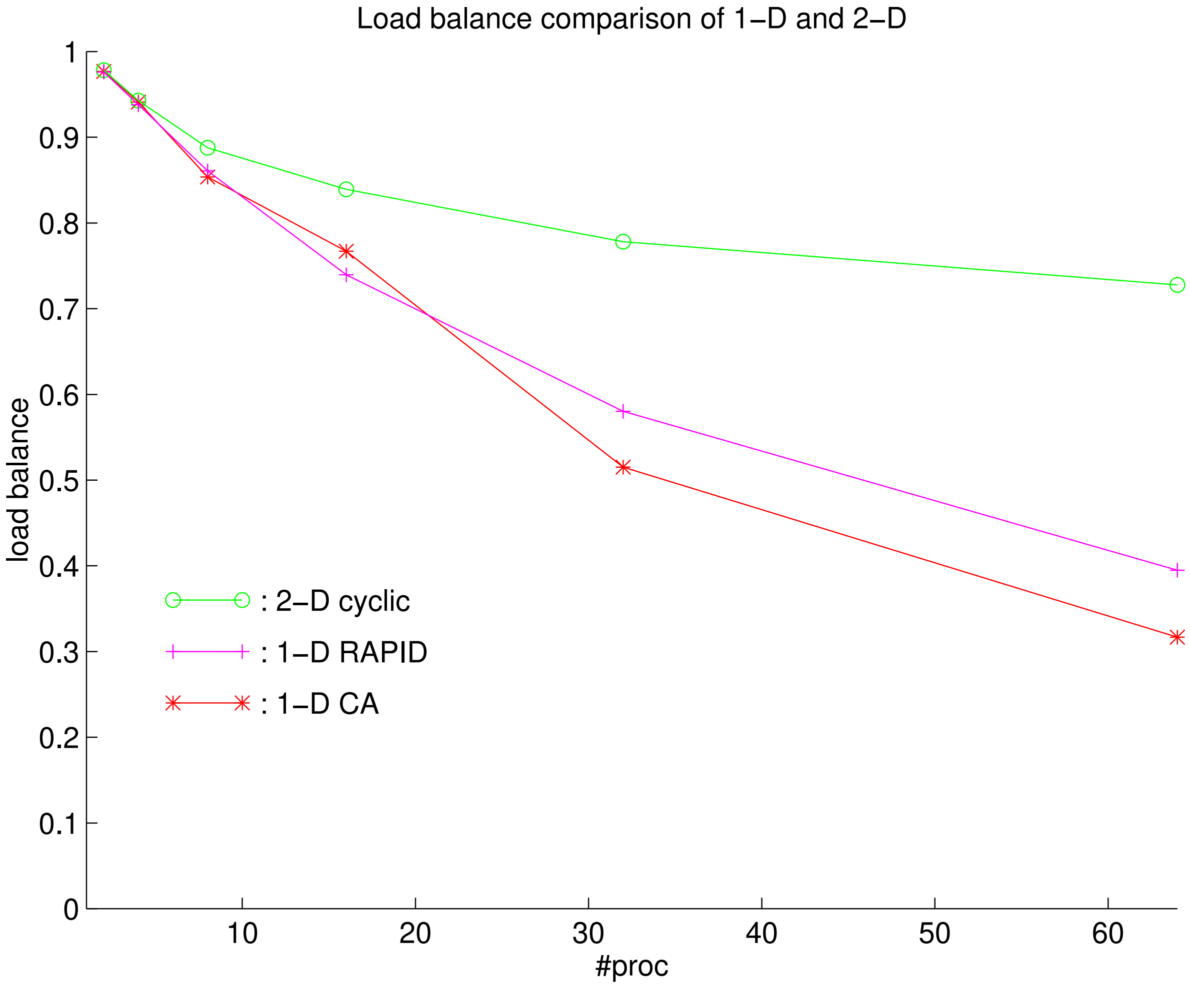
Scalability comparison

$S(2^i) = PT(2^{i-1})/PT(2^i)$, where $PT(2^i)$ is the parallel time on 2^i processors.



Load balance comparison

$$work_{total}/(P \cdot work_{max}).$$



Execution time distribution

Part	P=4		P=16		P=64	
	1-D	2-D	1-D	2-D	1-D	2-D
Factor() + waiting	18.4%	23.7%	56.3%	39.7%	81.2%	52.8%
Swapping	4.1%	6.8%	2.0%	8.8%	0.8%	7.9%

Importance of barrier-free design

Compare 2-D synchronous and asynchronous codes.

Improvement ratio: $1 - PT_{async}/PT_{sync}$.

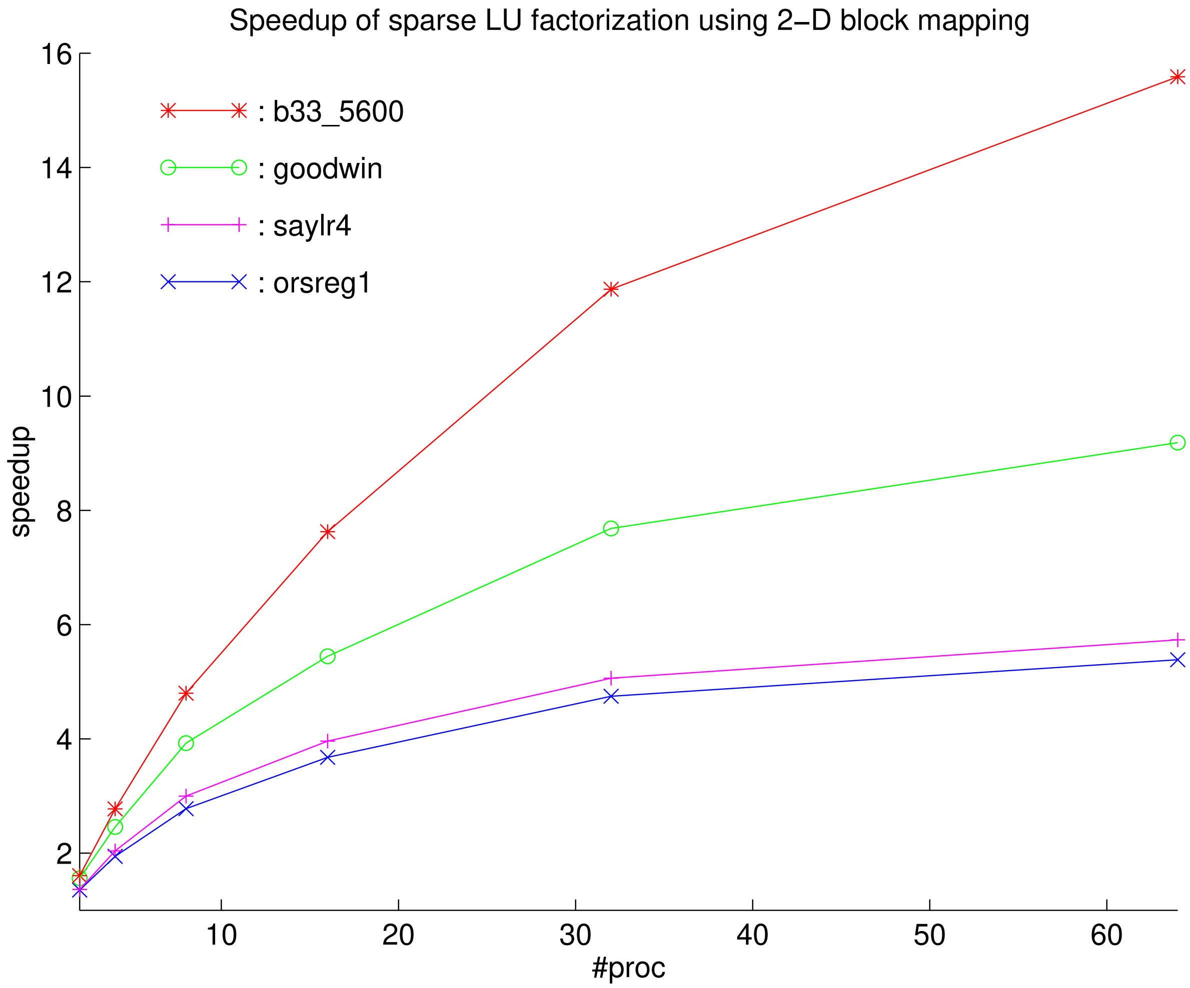
	P=2	P=4	P=8	P=16	P=32	P=64
Average	5.27%	7.13%	15.57%	16.95%	25.93%	26.07%

Conclusions

- 2-D mapping has better scalability, better load balancing, although the 2-D mapping requires frequent communication.
- 2-D code is memory efficient and has more potential to solve larger problems.
- Barrier-free design is important.
- Achieved 1.48 GFLOPS on 64 nodes of T3D.
- Static symbolic factorization works for most cases.

http://www.cs.ucsb.edu/Research/rapid_sweb/RAPID.html

Parallel speedups



Sequential code is 30-60% slower than SuperLU code.

Testing matrices

Matrix	Order	Description
goodwin	7320	fluid mechanics problem
e40r0100	17281	fluid dynamics problem
ex11	16614	3-D steady flow calculation
raefsky4	19779	buckling problem
vavasis3	41092	2-D PDE