Parallel Solution of Sparse Triangular Linear Systems on Multicore Platforms

İlke Çuğu

Department of Computer Engineering
Middle East Technical University

23/11/2018

Outline

Motivation

Sparse linear systems are found in many applications of science and engineering:

- Electromagnetics, circuit simulations, computational fluid dynamics, etc.

Sparse triangular systems arise in...

- Sparse matrix factorizations such as LU, QR, Cholesky, etc.
- Iterative solvers such as Gauss-Seidel, Successive Over Relaxations (SOR), Symmetric SOR, etc.

Parallel Sparse Triangular System Solvers

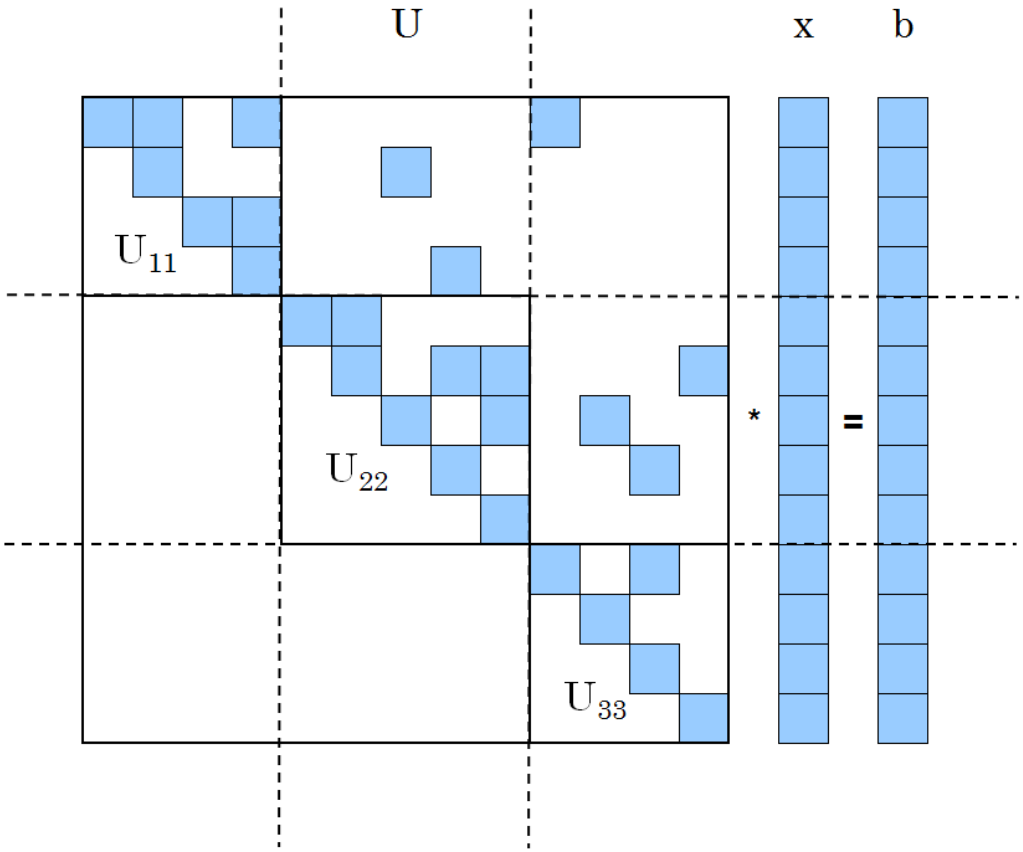
- Level-scheduling based methods
- Self-scheduling based methods
- Graph coloring based methods
- Block diagonal based methods
  - The proposed algorithm

The Algorithm - Origins

The Spike algorithm...

- is originally designed for banded systems
- is generalized for general sparse systems
- **is expanded and specialized for sparse triangular case by the proposed algorithm**

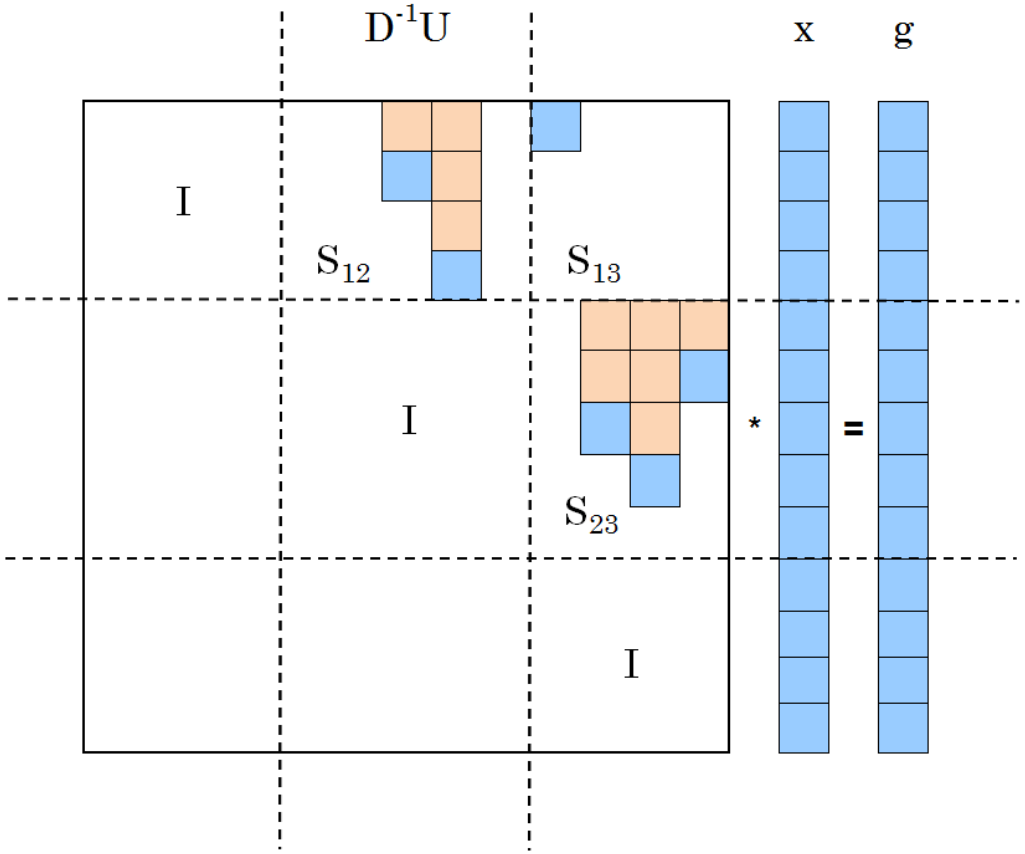
The Algorithm - The Original System



$$Ux = b$$

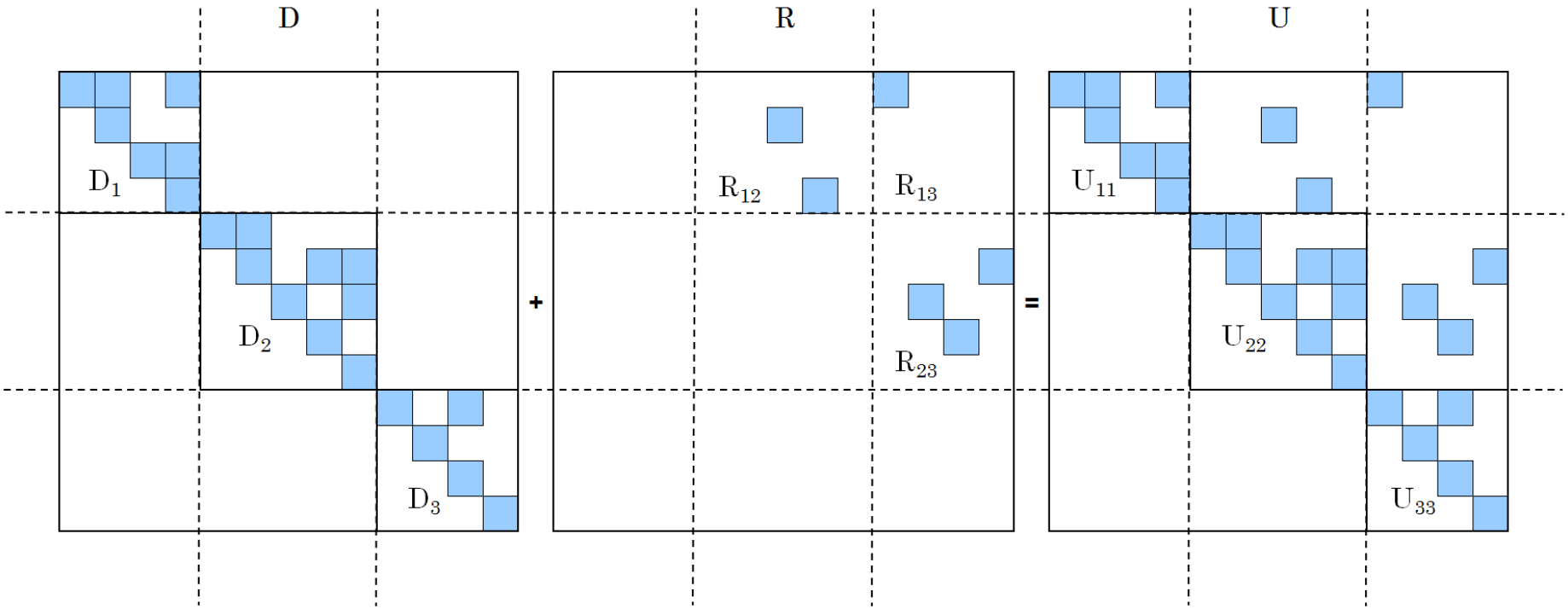
- The proposed algorithm is applicable to lower triangular case as well

The Algorithm - Structure of the Spike Matrix



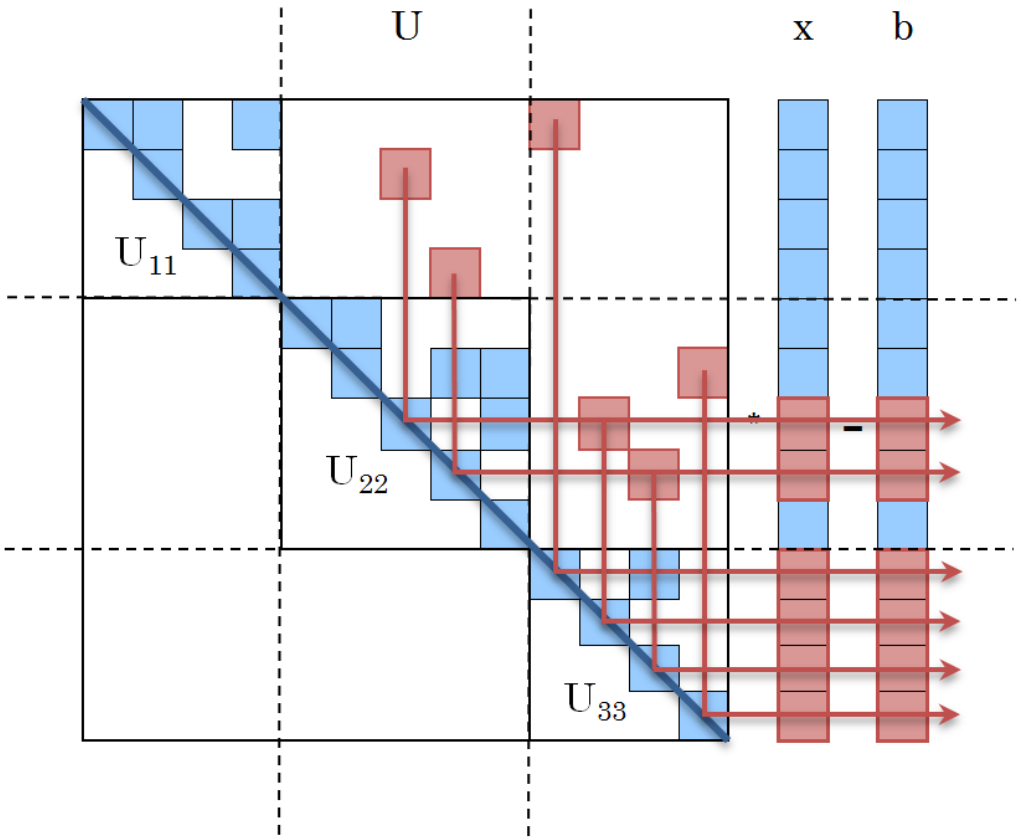
$$Sx = g$$

The Algorithm - Splitting U Matrix



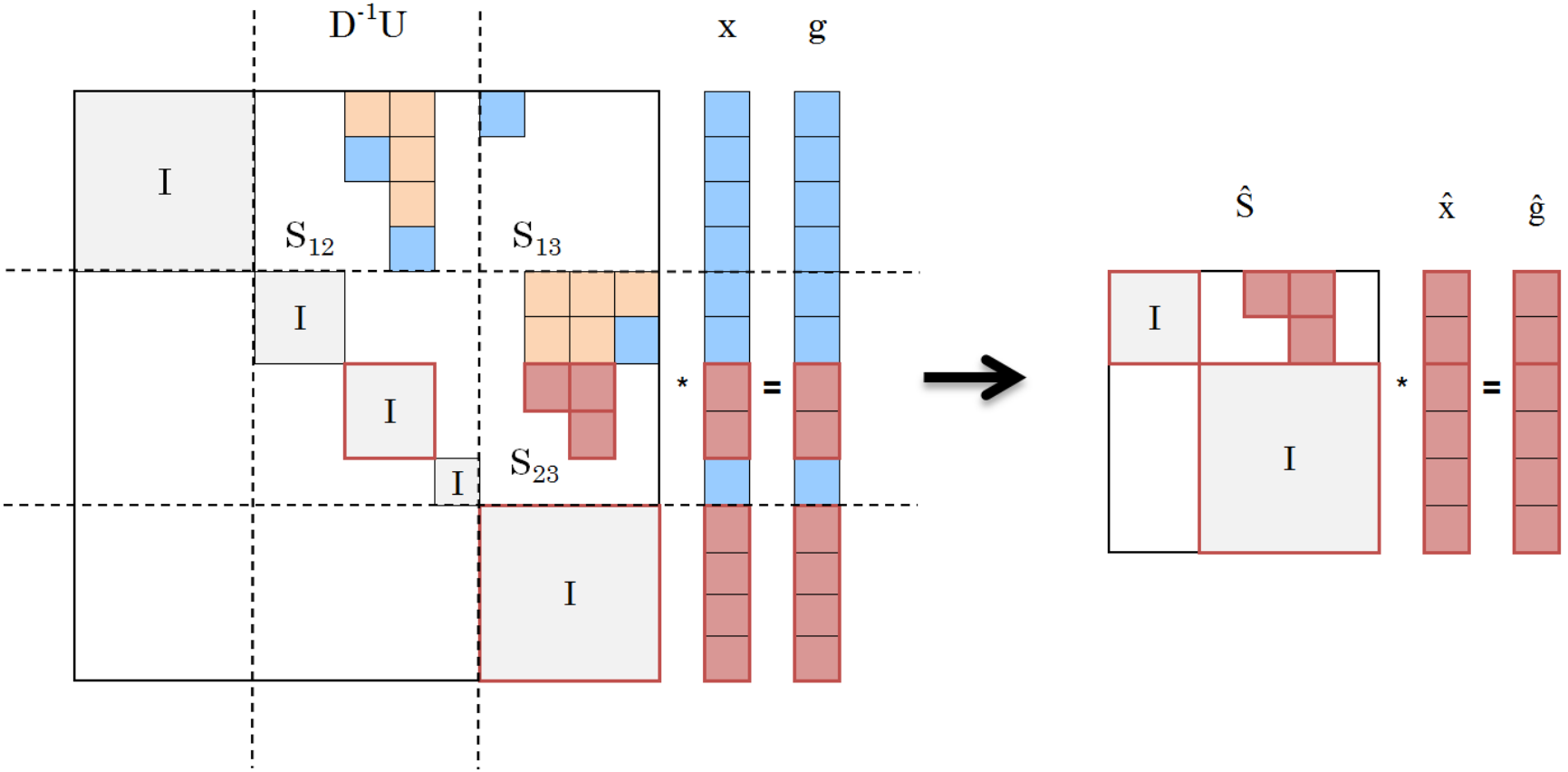
$$D + R = U$$

The Algorithm - Dependency Elements Metaphor



The illustration of *light beams* as dependency mappings

The Algorithm - The Reduced System



Construction of the reduced system

- $\widehat{S}$ is a $d \times d$ unit diagonal triangular matrix
- Solution of the reduced system requires $\mathcal{O}(nnz(\widehat{S}) - d)$ operations

The Algorithm - Preprocessing

Preprocessing phase covers operations independent from the right hand side vector b:

- Partitioning D matrix
- **Memory allocation for dense R and S parts**
- Compressing R into a dense form
- **Computing the partial S matrix**
- Load-balance optimization for the parallel blocks

The Algorithm - Solution

Algorithm 1 PSTRSV

Input: Partitioned and factored coefficient matrix $U = DS$, reduced coefficient matrix $\hat{S}$, together with associated dependency information and b, the right-hand side vector

Output: x, solution vector of $Ux = b$

for each thread $i = 1, 2, ..., t$ **do**

  if $hasReflection_i$ or $isOptimized_i$ **then**

    Solve the triangular system $D_i^{(m;b)} g_i^{(m;b)} = b_i^{(m;b)}$ for $g_i^{(m;b)}$

  end if

  Wait until all threads reach this point

  for a single thread i **do**

    Solve the reduced system $\widehat{S}\widehat{x} = \widehat{g}$ for $\widehat{x}$

    Update the solution vector $x \leftarrow \widehat{x}$

  end for

  Wait until all threads reach this point

  if $hasDependence_i$ **then**

    $b_i^{(t;m)} := b_i^{(t;m)} - (\hat{R}_i x + P_i x_i^{(b)})$

  end if

  if $hasReflection_i$ or $isOptimized_i$ **then**

    Solve the triangular system $D_i^{(t;m)} x_i^{(t;m)} = b_i^{(t;m)}$ for $x_i^{(t;m)}$

  else

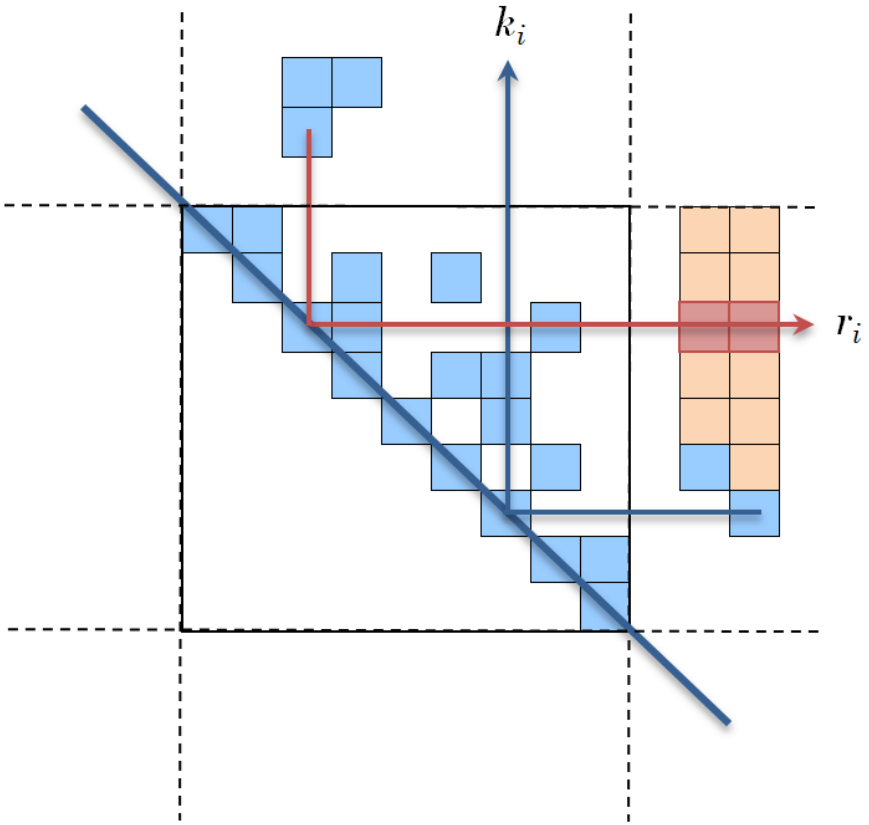
    Solve the triangular system $D_i x_i = b_i$ for x_i

  end if

end for

return x

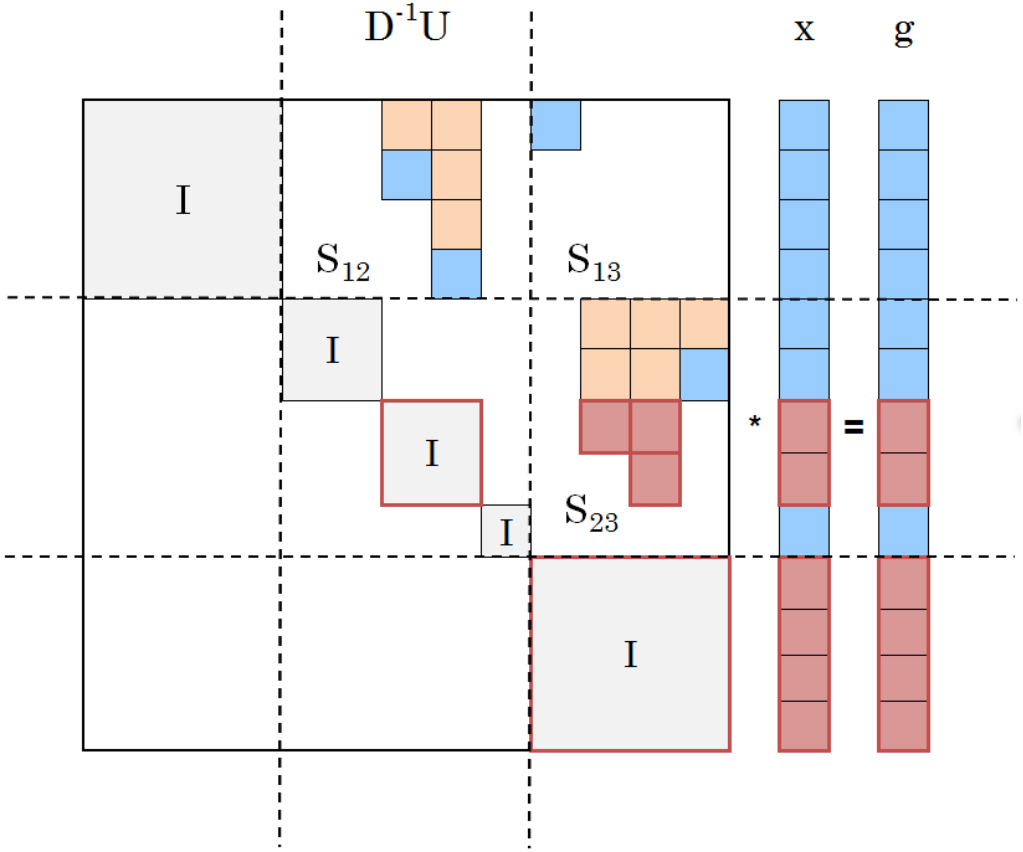
Performance Constraints - Key Parameters



Two of the key performance parameters

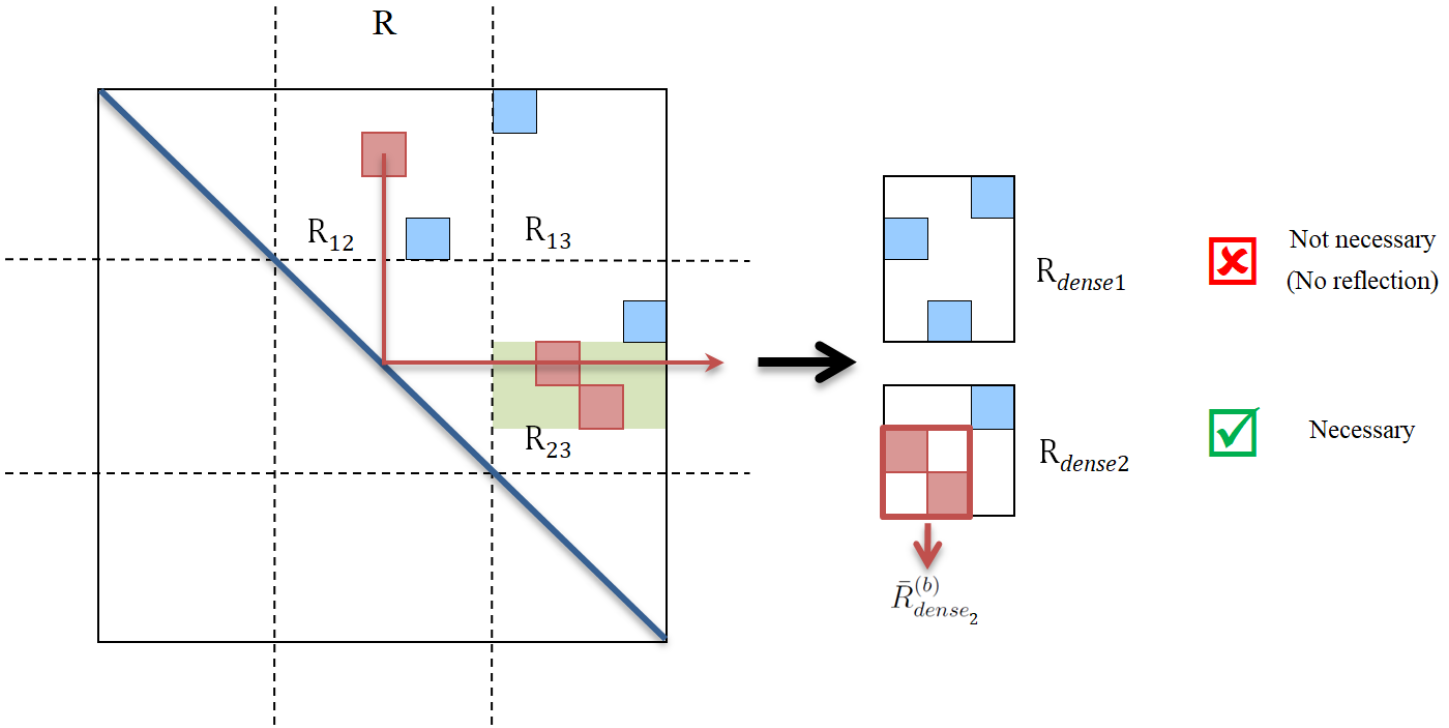
- reflection r_i: Row index of the top-most *light beam* for each R_i
- k_i: Row index of the bottom-most *dependency element* for each R_i
- $nnz(\widehat{S}) - d$: # of off-diagonal nonzeros in $\widehat{S}$

Performance Constraints - Preprocessing



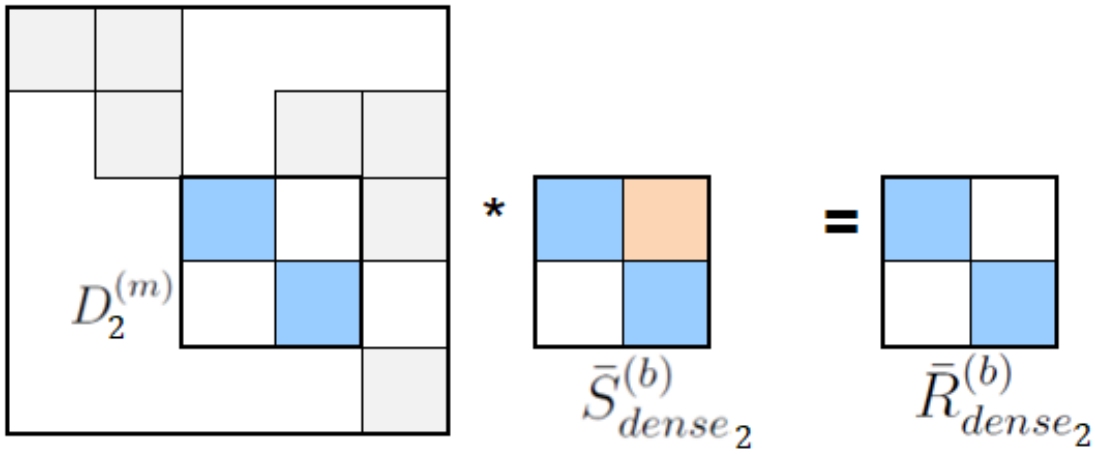
We only need to compute S matrix parts highlighted in red

Performance Constraints - $\bar{R}_i^{(b)}$ to $\bar{R}_{dense_i}^{(b)}$



We transform the sparse $\bar{R}_i^{(b)}$ matrix to dense $\bar{R}_{dense_i}^{(b)}$ matrix

Performance Constraints - Computing $\bar{S}^{(b)}_{dense_i}$



Solution of a sparse triangular system with multiple right hand side vectors

Performance Constraints - Solution

Ideal scenarios:

- for $d_i = 0, \forall i \in \{1, 2, ..., t\}$ there is no reduced system
- for $r_i > k_i, \forall i \in \{1, 2, ..., t\}$, $\widehat{S}$ is the identity matrix

Numerical Experiments - Environment

Hardware:

- 2 sockets
- in each an Intel(R) Xeon(R) CPU E5-2650 v3 processor
- 10 cores per processor (20 cores in total)
- 16 GB of memory

Software:

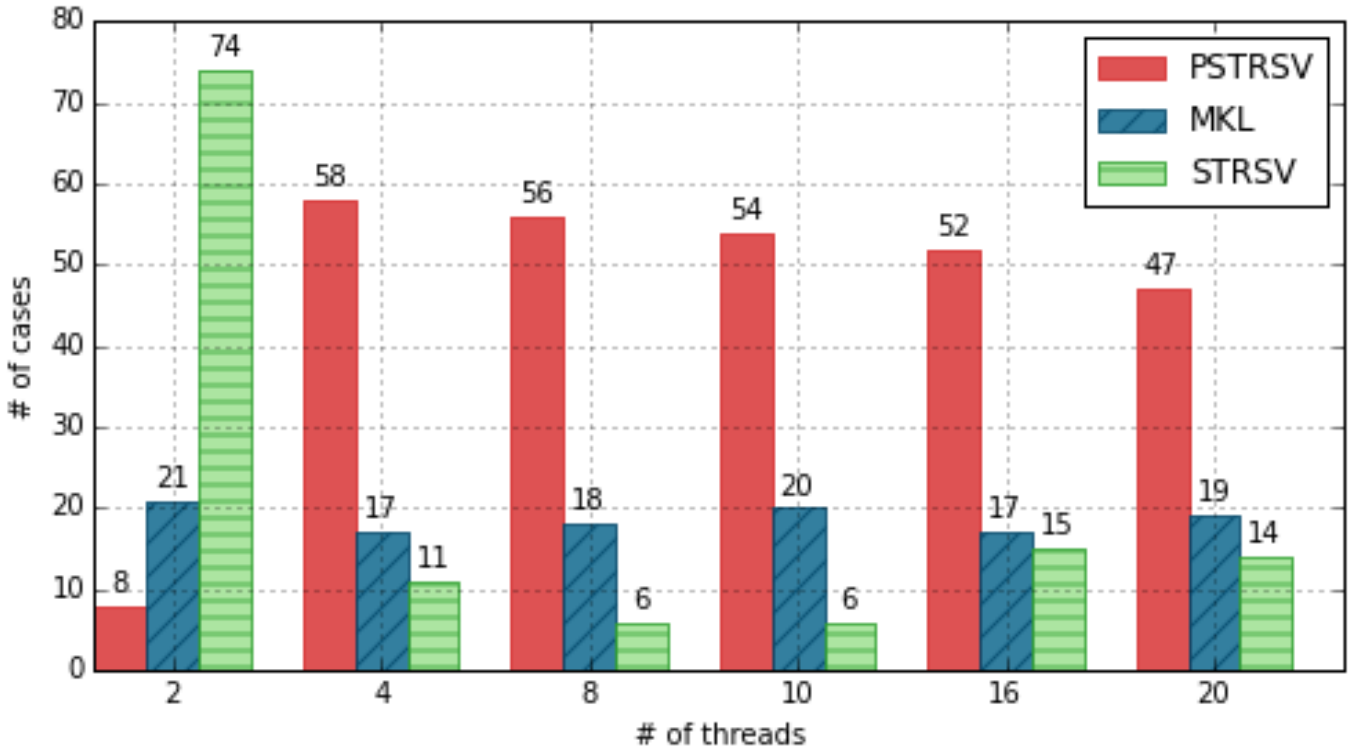
- Matrices are in Compressed Sparse Row (CSR) format
- Intel Math Kernel Library (MKL) 2018 is used
- PSTRSV is implemented in C with OpenMP
- KMP_AFFINITY = granularity = fine,compact,1,0

Numerical Experiments

In the experiments...

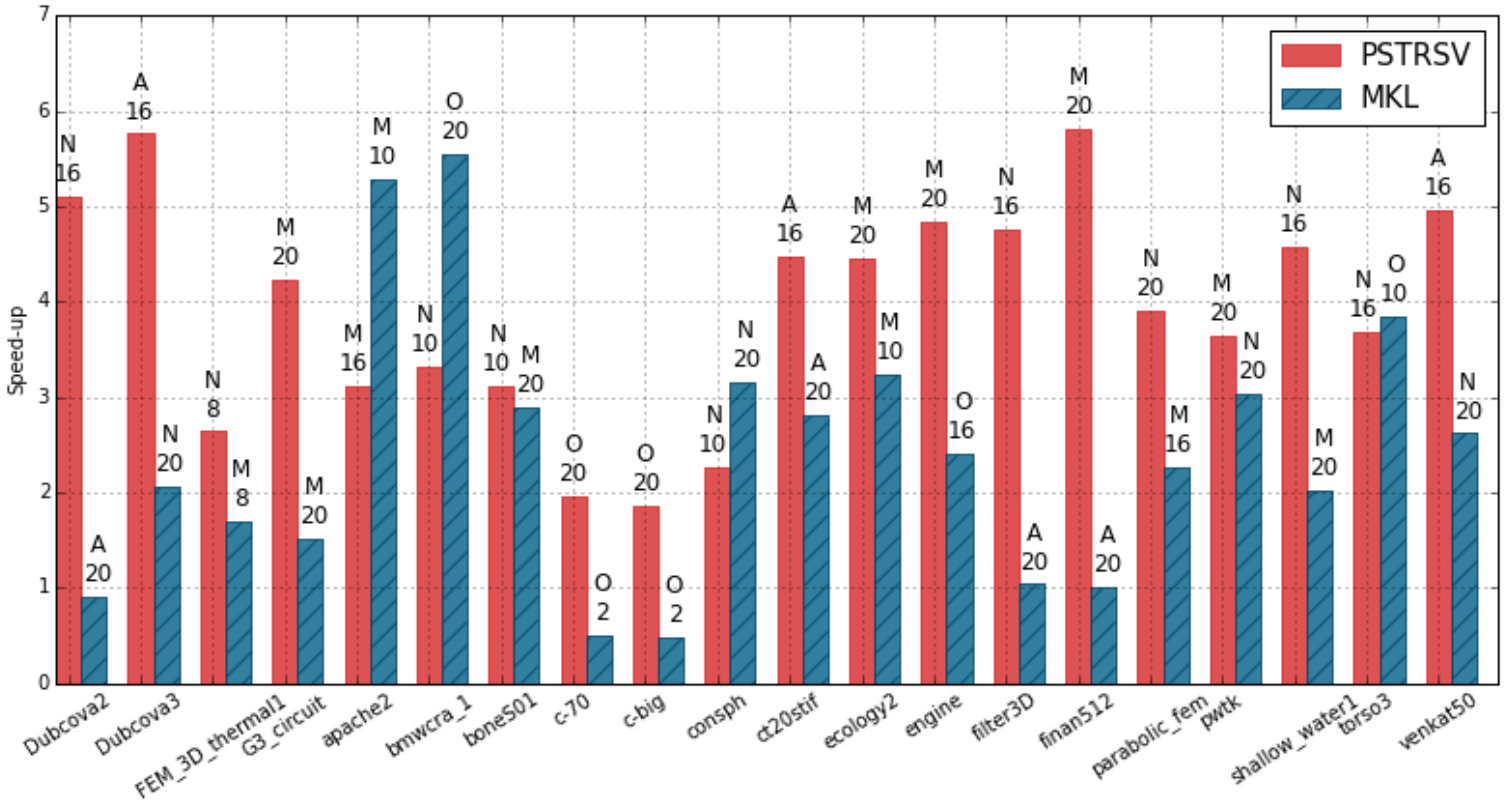
- 20 real world matrices are taken from SuiteSparse Matrix Collection
- METIS, AMD, ColPerm, NDP and RCM orderings are employed
- multithreaded sparse triangular solver of Intel MKL 2018 is used
- each run is repeated 1,000 times and to obtain the avg wallclock time

Numerical Experiments - Solution



Overall performance comparison

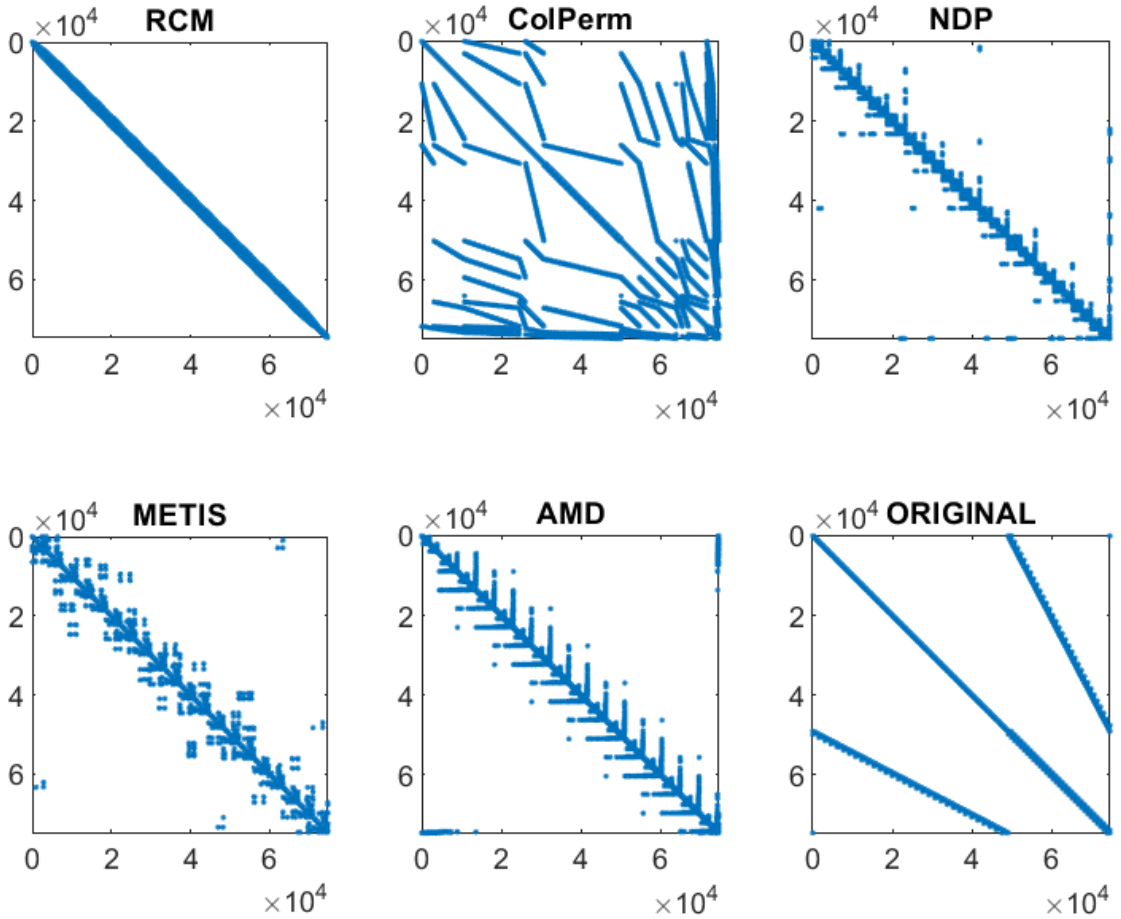
Numerical Experiments - Solution



The highest speed-ups achieved by PSTRSV and MKL

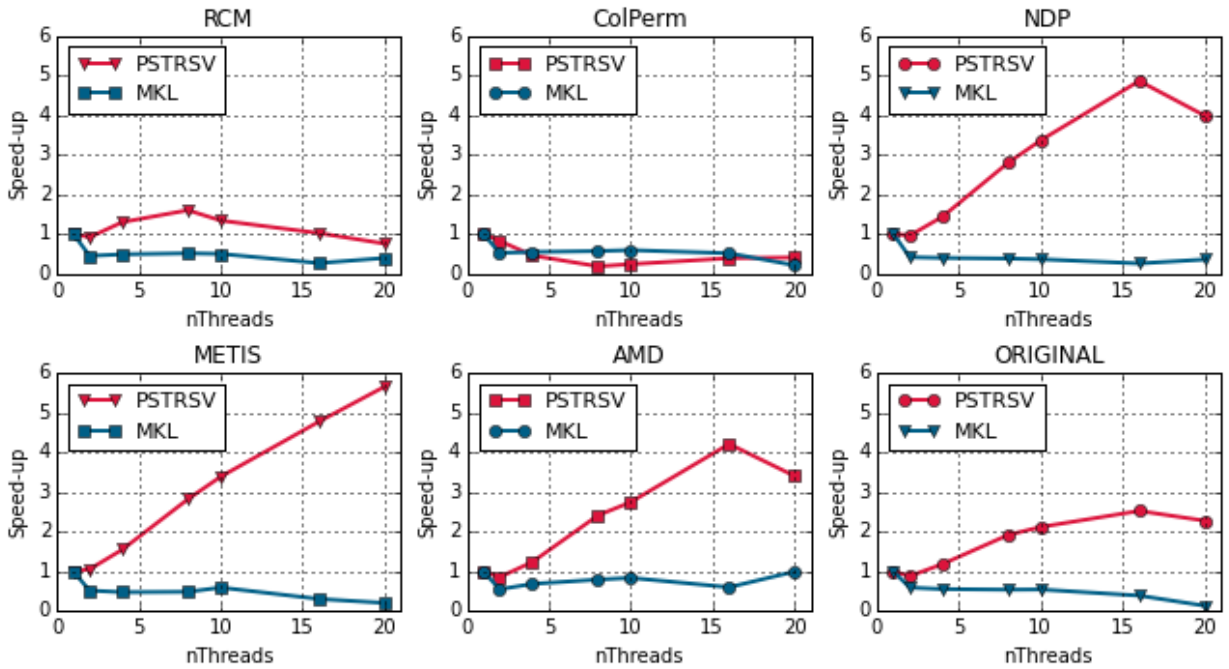
- PSTRSV cannot amortize the preprocessing overhead in 9/120 cases
- MKL cannot amortize the preprocessing overhead in 21/120 cases

Numerical Experiments - Closer Look



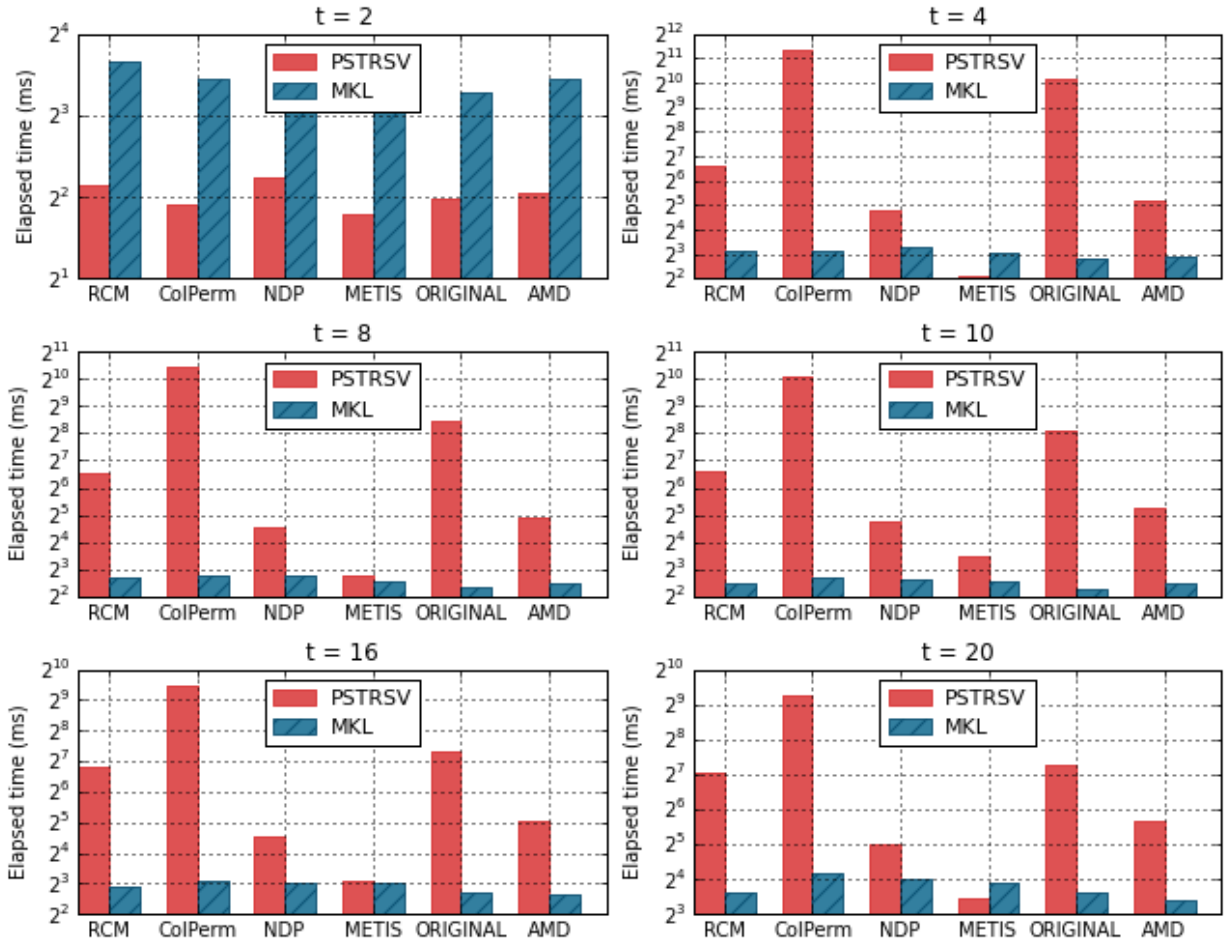
The illustration of *finan512* for different matrix reorderings

Numerical Experiments - Closer Look



The speed-up comparison for *finan512*

Numerical Experiments - Closer Look



The preprocessing time comparison for *finan512*

Numerical Experiments - Preprocessing

t	PSTRSV				MKL			
	min	max	avg	std	min	max	avg	std
2	2.40	75.22	26.97	204.20	4.11	251.50	78.77	616.41
4	4.02	5995.39	875.11	10215.81	2.82	131.36	46.50	338.93
8	4.07	2988.42	576.13	5972.42	2.17	114.80	32.89	244.67
10	4.17	2756.16	495.46	5161.81	2.58	118.37	31.32	242.35
16	4.41	2961.46	372.92	4223.28	0.19	115.57	27.41	206.61
20	4.12	2219.22	327.21	3500.55	0.44	264.46	35.85	332.74

Statistics of the preprocessing times of PSTRSV and MKL in milliseconds

- $t = 2$ is a special condition where $r_0 = 0$ and $k_1 = 0$ (no $\bar{R}_i^{(b)}$ or $\bar{S}_i^{(b)}$)

Summary

PSTRSV...

- is implemented in C with OpenMP
- benefits from METIS, AMD and NDP orderings
- is tested with matrices taken from SuiteSparse Matrix Collection
- outperforms MKL in $\sim 80\%$ of cases by a factor of 2.3 on average
- achieves best speed-ups with..
  - 9/20 cases: NDP
  - 6/20 cases: METIS
  - 3/20 cases: AMD
  - 2/20 cases: Original
- **is submitted to Journal of Parallel and Distributed Computing (in review)**

Future Work

Investigation on

- further optimization in the preprocessing phase
- other matrix ordering frameworks such as PaToH
- a specialized graph partitioning algorithm tailored for PSTRSV
- an MPI implementation of the proposed algorithm