# Parallelization of Multilevel Preconditioners Constructed from Inverse-Based ILUs on Shared-Memory Multiprocessors

J.I. Aliaga[1] M. Bollhöfer[2] A.F. Martín[1]
E.S. Quintana-Ortí[1]

[1]Deparment of Computer Science and Engineering, Univ. Jaume I (Spain)
{aliaga,martina,quintana}@icc.uji.es

[2]Institute of Computational Mathematics, TU-Braunschweig (Germany)
m.bollhoefer@tu-braunschweig.de

September, 2007

The solution of SPARSE linear systems is ubiquitous:

- Matrix market collection
  `http://math.nist.gov/MatrixMarket/`
- UF sparse matrix collection
  `http://www.cise.ufl.edu/research/sparse/matrices/`

Two main approaches to “attack” the problem:

- Direct methods (sparse LU, sparse Cholesky, etc.) may fail due to excessive fill-in of the factors
- Iterative methods (PCG, GMRES, etc.) may present slow convergence or even divergence

Convergence of iterative methods can be accelerated via preconditioning:

- Incomplete LU
- Sparse approximate inverse
- ...

Temporal cost of preconditioning can be reduced using high-performance computing and parallel platforms:

- Distributed-memory: Difficult to program, high cost of communication specially hurts sparse linear algebra
- Shared-memory: lack of scalability

Mid-term goal: Computation of parallel preconditioners on shared-memory architectures

- These platforms provide enough computational power (16–64 processors) for many practical applications
- With the advent of manycore, scalability will improve

Focus: First results obtained from a parallel preconditioner that employs the same techniques as those of ILUPACK using OpenMP on a CC-NUMA platform

# Outline

1. Motivation and Introduction
2. Overview of ILUPACK
   - Incomplete LU factorization
   - ILUPACK
3. Computation of Parallel Preconditioners
   - Trees and sparse matrices
   - Parallel algorithm
   - Parallel implementation
4. Experimental Results
   - Experimental Framework
   - Task (dependency) trees
   - Preconditioner computation results
5. Concluding Remarks

# Outline

1. Motivation and Introduction
2. Overview of ILUPACK
   - Incomplete LU factorization
   - ILUPACK
3. Computation of Parallel Preconditioners
   - Trees and sparse matrices
   - Parallel algorithm
   - Parallel implementation
4. Experimental Results
   - Experimental Framework
   - Task (dependency) trees
   - Preconditioner computation results
5. Concluding Remarks

# Incomplete LU factorization

Given

$$Ax = b,$$

compute $L$ and $U^T$ (sparse) unit lower triangular and $D$ diagonal so that,

$$M = LDU, \quad A = M - R,$$

while controlling the fill-in by means of dropping techniques. The application of the iterative method on the preconditioned system, accelerates its convergence.

$$M^{-1}Ax = M^{-1}b$$

Many "succesful" ILU factorization preconditioners:

- ILU(0), ILU(1) . . . ILU($p$): level-of-fill ILU with parameter $p$
- ILUT: ILU with threshold
- ILUTP: ILUT with partial pivoting
- ILU with inverse-based dropping
- Multilevel ILU
- . . .

**ILUPACK library implements multilevel inverse-based ILUs**

# ILUPACK

In each level:

- Initial static ordering (+ scaling) of the coefficient matrix

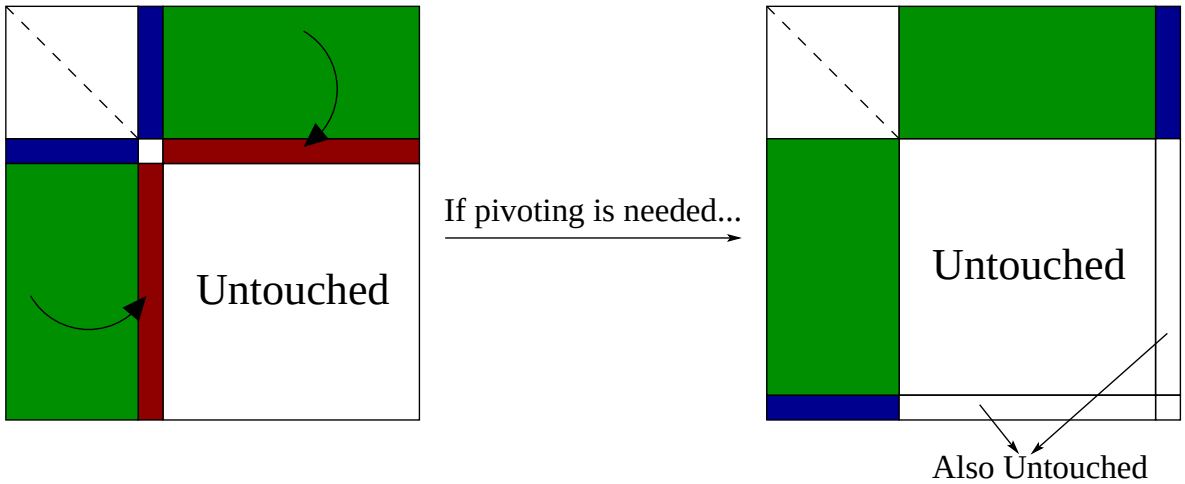
$$P^T A Q = \tilde{A}$$

  (MLND, RCM, MMD, ...)
- Incomplete LU factorization with controlled dropping
- Also, it uses diagonal pivoting to control the growth of

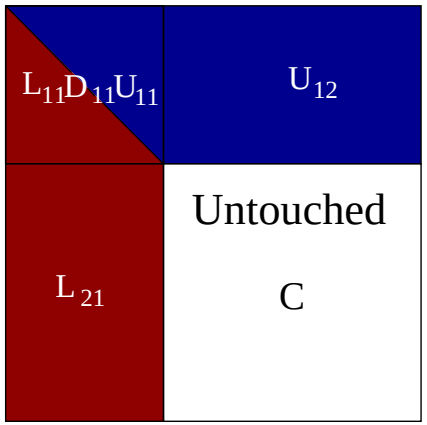
$$\|L^{-1}\| \quad \text{and} \quad \|U^{-1}\|$$

- The pivoted part forms the input matrix of the next level

- Crout variant of LU factorization with two different criteria for dropping entries in the factors:
  - "Small" entries
  - Entries with "small" influence on the growth of $\|L^{-1}\|,\ \|U^{-1}\|$
- Diagonal pivoting to control growth of $\|L^{-1}\|,\ \|U^{-1}\|$



If the "untouched part" only contains pivoted elements ...

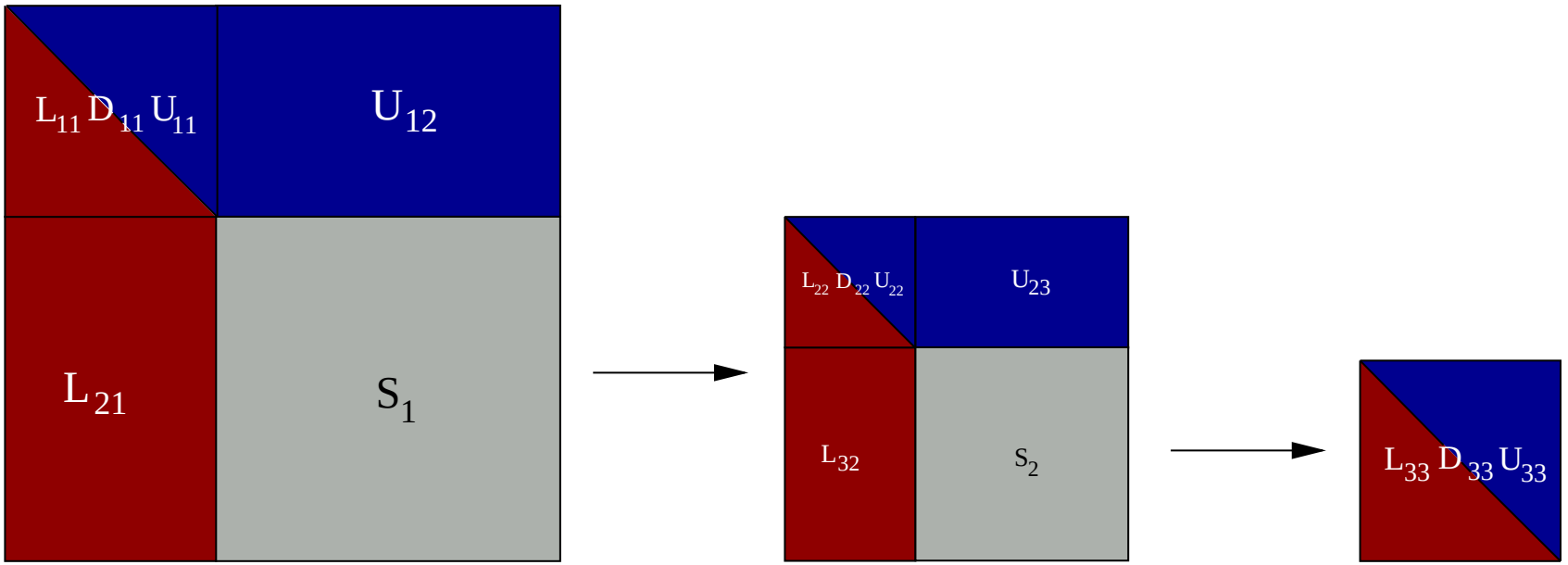



- Obtain the *approximate Schur complement* of $C$

$$S = C - L_{21} D_{11} U_{12},$$

  applying certain dropping techniques
- Repeat the procedure to factorize $S$ (multilevel recursion)

# A three level inverse-based multilevel ILU example



# Outline

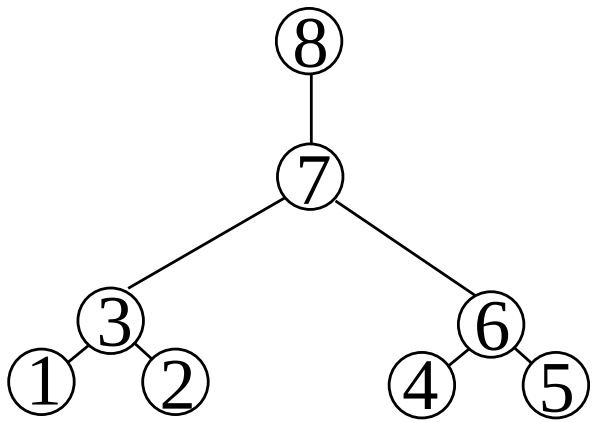
1. Motivation and Introduction
2. Overview of ILUPACK
   - Incomplete LU factorization
   - ILUPACK
3. Computation of Parallel Preconditioners
   - Trees and sparse matrices
   - Parallel algorithm
   - Parallel implementation
4. Experimental Results
   - Experimental Framework
   - Task (dependency) trees
   - Preconditioner computation results
5. Concluding Remarks

# Elimination tree

Matrix (sparsity pattern)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | × | | × | | | | | |
| 2 | | × | × | | | | | × |
| 3 | × | × | × | | | | × | |
| 4 | | | | × | | × | | |
| 5 | | | | | × | × | | × |
| 6 | | | | × | × | × | × | |
| 7 | | | × | | | × | × | |
| 8 | | × | | | × | | | × |

Elimination tree

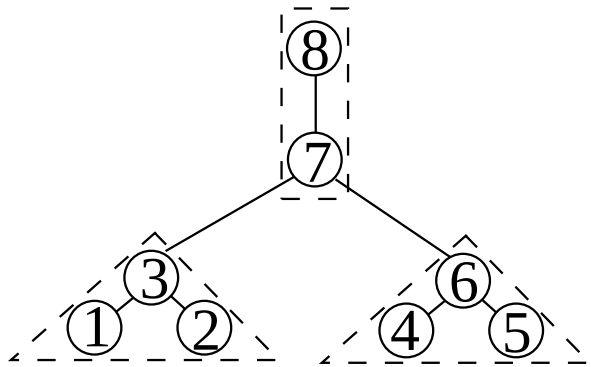



The elimination tree shows groups of rows/columns which can be factorized *independently* as, e.g., {1,2,3} and {4,5,6}

Matrix (sparsity pattern)

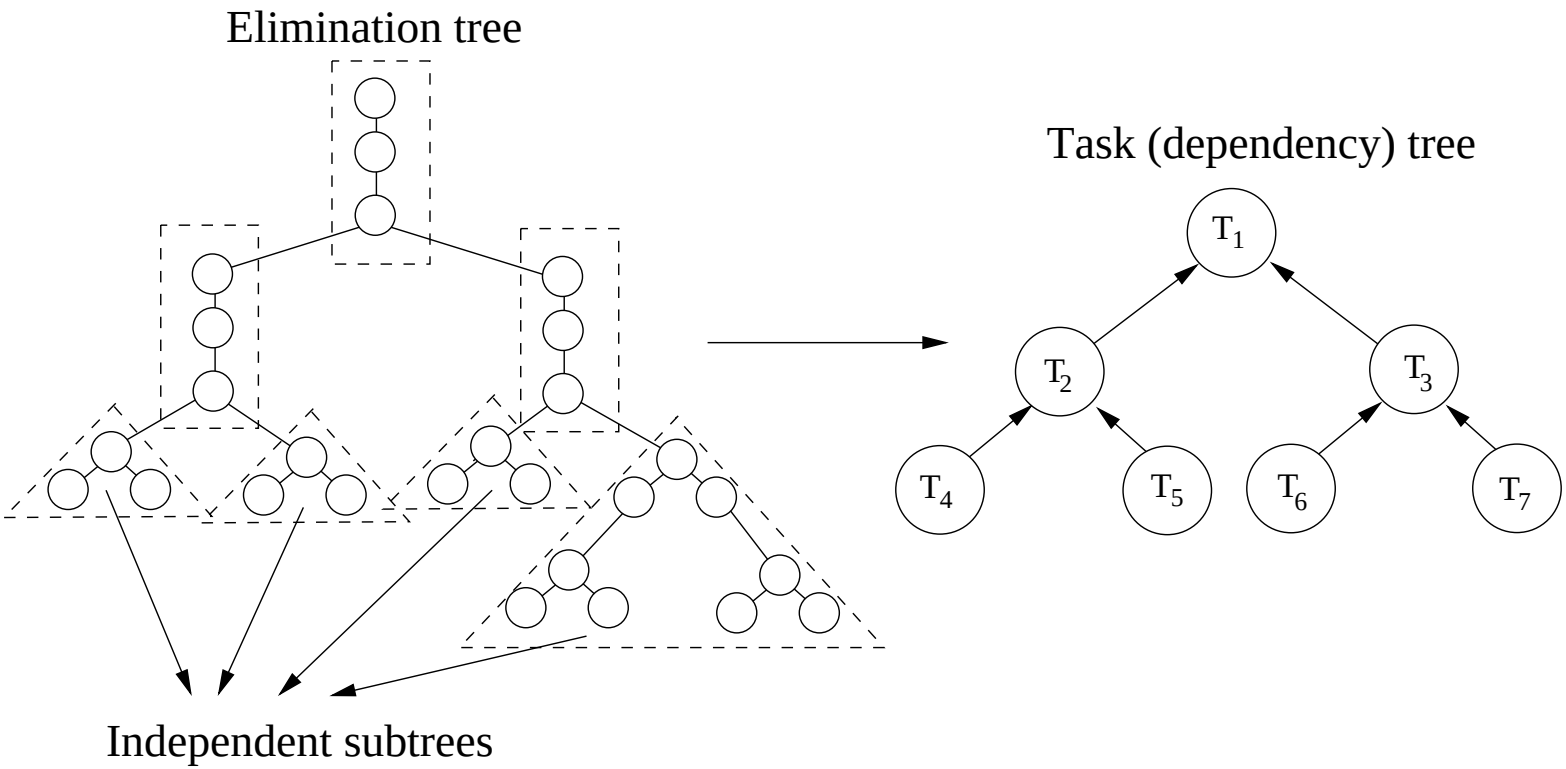
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | × | | × | | | | | |
| 2 | | × | × | | | | | × |
| 3 | × | × | × | | | | × | |
| 4 | | | | × | | × | | |
| 5 | | | | | × | × | | × |
| 6 | | | | × | × | × | × | |
| 7 | | | × | | | × | × | |
| 8 | | × | | | × | | | × |

Elimination tree




The elimination tree shows groups of rows/columns which can be factorized *independently* as, e.g., {1,2,3} and {4,5,6}

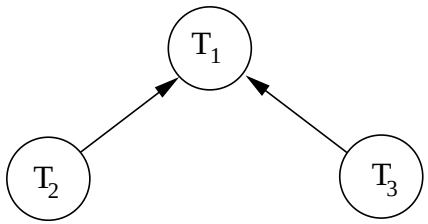
# Task (dependency) tree




The task tree shows tasks which can be executed *concurrently (i.e., in parallel)*

The task tree defines a block partition of the matrix.

Consider a task tree of height one:

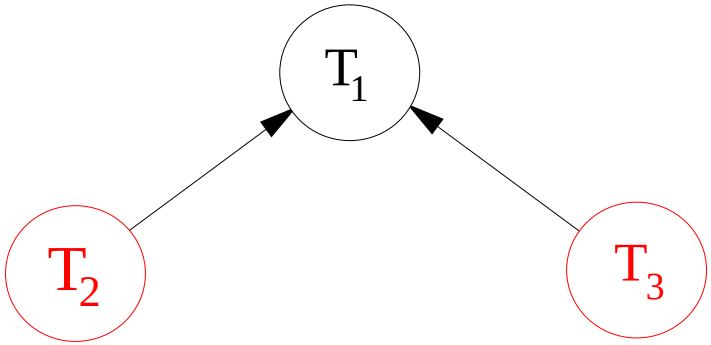



$$\rightarrow \quad A = \left(\begin{array}{c|c|c} A_{11} & 0 & A_{13} \\ \hline 0 & A_{22} & A_{23} \\ \hline A_{13}^T & A_{23}^T & A_{33} \end{array}\right)$$

To compute a preconditioner of $A$ in parallel, we assign submatrices to each leaf of the task tree:

$$\left(\begin{array}{c|c} A_{11} & A_{13} \\ \hline A_{13}^T & A_{33}^1 \end{array}\right) \rightarrow T_2, \quad \left(\begin{array}{c|c} A_{22} & A_{23} \\ \hline A_{23}^T & A_{33}^2 \end{array}\right) \rightarrow T_3, \quad \text{with } A_{33}^1 + A_{33}^2 = A_{33}$$

# Parallel algorithm

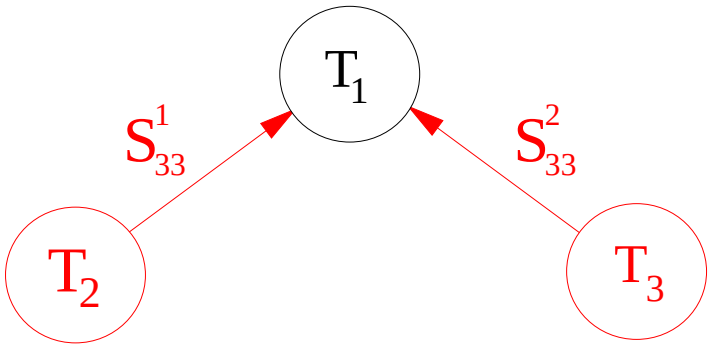



Tasks $T_2$, $T_3$ use ILUPACK to factorize, respectively,

$$\left(\begin{array}{c|c} A_{11} & A_{13} \\ \hline A_{13}^T & A_{33}^1 \end{array}\right) \rightarrow \left(\begin{array}{c|c} L_{11}, D_{11}, U_{11} & U_{13} \\ \hline L_{13} & S_{33}^1 \end{array}\right)$$

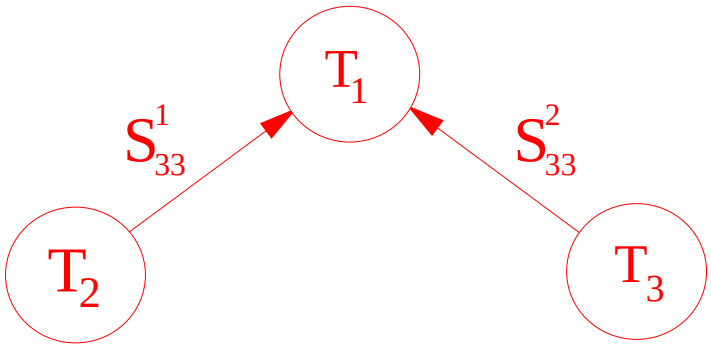
$$\left(\begin{array}{c|c} A_{22} & A_{23} \\ \hline A_{23}^T & A_{33}^2 \end{array}\right) \rightarrow \left(\begin{array}{c|c} L_{22}, D_{22}, U_{22} & U_{23} \\ \hline L_{23} & S_{33}^2 \end{array}\right)$$

# Parallel algorithm




Tasks $T_2$, $T_3$ interact with $T_3$ to send, resp., $S_{33}^1$, $S_{33}^2$

# Parallel algorithm




Tasks $T_2$, $T_3$ interact with $T_3$ to send, resp., $S_{33}^1$, $S_{33}^2$

Task $T_1$ uses ILUPACK to factorize the submatrix $S_{33}^1 + S_{33}^2$

$$S_{33}^1 + S_{33}^2 \rightarrow L_{33}, D_{33}, U_{33}$$

One forced level appears in the parallel preconditioner

$$\left(\begin{array}{cc|c} L_{11}, D_{11}, U_{11} & 0 & U_{13} \\ 0 & L_{22}, D_{22}, U_{22} & U_{23} \\ \hline L_{31} & L_{32} & S_{33} \end{array}\right) \quad \rightarrow \quad L_{33}, D_{33}, U_{33}$$

First Level Second Level

Then, the parallel preconditioner is not the same as the preconditioner computed by ILUPACK!

# Task (dependency) tree *whishes*

In a "good" task tree (from the parallelization viewpoint):

- Parallelism:
  - The leaves *should* concentrate the major part of the cost
  - The number of leaves *should* enable a high degree of concurrency (taking into account that the overhead usually increases with their number)
- Load balancing: In general, the leaves *should* have homogenous costs

# Achieving task tree *whishes*

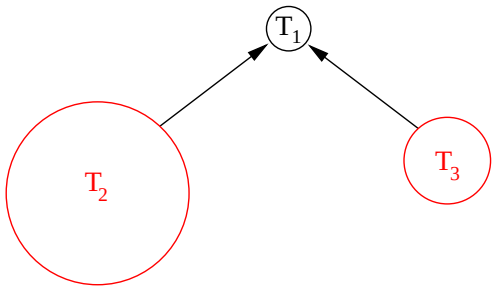
Our approach combines:

- MLND (METIS): yields a best-effort balanced elimination tree with a high degree of concurrency
- Task splitting: divides those leaves with higher computational cost into finer-grain tasks to improve load balancing

# Achieving task tree *whishes*

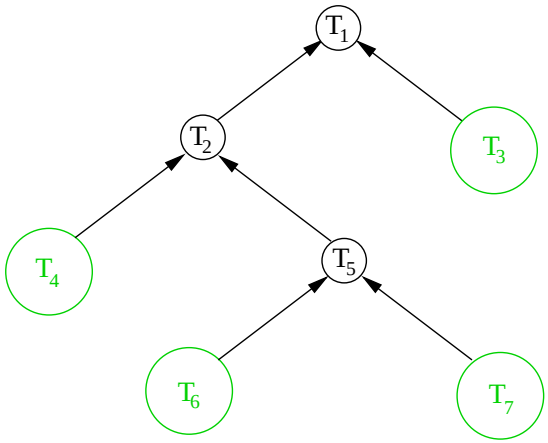
## 2 Processors

Bad load balancing




Task splitting

Load balancing improvement




Static task generation (i.e., before execution) !

# Achieving task tree *whishes*

. . . but

- the costs of the tasks are unknown *a priori*
  $\longrightarrow$ use an heuristic of the cost (weight): currently, the number of nonzeros in the submatrix to factorize
- the costs of the tasks can be heterogenous
  $\longrightarrow$ use dynamic scheduling for load balancing

# Parallel implementation

- OpenMP implementation
- Maintain a queue of tasks ready for execution (dependencies fulfilled)
- Priorize execution of leaves with larger weights
- Idle threads extract tasks from this structure and execute them
- When a task is completed, the task tree is examined to find out a new ready task, which is enqueued

# Outline

1. Motivation and Introduction
2. Overview of ILUPACK
   - Incomplete LU factorization
   - ILUPACK
3. Computation of Parallel Preconditioners
   - Trees and sparse matrices
   - Parallel algorithm
   - Parallel implementation
4. Experimental Results
   - Experimental Framework
   - Task (dependency) trees
   - Preconditioner computation results
5. Concluding Remarks

# Miscellaneous

- SGI Altix 350 CC-NUMA SMM with:
  - 16 Intel Itanium2@1.5 GHz
  - 32 GBytes of RAM shared via a SGI NUMAlink interconnect
- No attempt to exploit CC-NUMA data locality yet
- IEEE double precision
- -O3 Intel Compiler optimization level
- One thread scheduled per processor
- In both the serial and parallel algorithms:
  - MLND as initial static ordering
  - ILUPACK default values

# Benchmark Matrices

SPD matrices obtained from the UF sparse matrix collection

| Code | Matrix name | Rows/Cols. | Nonzeros |
|---|---|---|---|
| M1 | *GHS_psdef/Apache1* | 80800 | 542184 |
| M2 | *Schmid/Thermal1* | 82654 | 574458 |
| M3 | *Schenk_AFE/Af_0_k101* | 503625 | 17550675 |
| M4 | *GHS_psdef/Inline_1* | 503712 | 36816342 |
| M5 | *GHS_psdef/Apache2* | 715176 | 4817870 |
| M6 | *GHS_psdef/Audikw_1* | 943695 | 77651847 |

# Task splitting

- Guided by leaf tasks weights (heuristic cost)
- Splitting criteria: Let
  - $w_i$ the weight of leaf task $T_i$,
  - $W$ the overall matrix weight (i.e., the number of nonzero elements of the matrix), and
  - $p$ the number of threads

  Split those leaves with ratio $\frac{w_i}{W}$ ...
  - larger than $\frac{1}{p} \rightarrow$ option A $\rightarrow$ $num\_leaves \geq p$
  - larger than $\frac{1}{2p} \rightarrow$ option B $\rightarrow$ $num\_leaves \geq 2p$

# Task (dependency) trees

In order to measure how $w_i$'s is balanced among threads, we use:

$$c_w = \frac{\sigma_w}{\bar{w}}$$

with $\sigma_w$ and $\bar{w}$, the std. dev. and the avg. of thread weights, assuming a static mapping (i.e., before execution) of leaf tasks to threads

The closer $c_w$ to 0, the better the weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 2 | 4 | 11 | 23 | 24 | 0 | 0 | 1 | 29 | 5 |
| M2 | 3 | 5 | 9 | 17 | 19 | 0 | 1 | 2 | 32 | 5 |
| M3 | 2 | 4 | 9 | 16 | 18 | 0 | 2 | 5 | 33 | 6 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 4 | 11 | 24 | 43 | 45 | 0 | 1 | 1 | 26 | 5 |
| M2 | 5 | 9 | 19 | 33 | 37 | 0 | 1 | 0 | 26 | 5 |
| M3 | 4 | 9 | 18 | 32 | 37 | 0 | 4 | 0 | 27 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 2 | 4 | 11 | 23 | 24 | 0 | 0 | 1 | 29 | 5 |
| M2 | 3 | 5 | 9 | 17 | 19 | 0 | 1 | 2 | 32 | 5 |
| M3 | 2 | 4 | 9 | 16 | 18 | 0 | 2 | 5 | 33 | 6 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 4 | 11 | 24 | 43 | 45 | 0 | 1 | 1 | 26 | 5 |
| M2 | 5 | 9 | 19 | 33 | 37 | 0 | 1 | 0 | 26 | 5 |
| M3 | 4 | 9 | 18 | 32 | 37 | 0 | 4 | 0 | 27 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 2 | 4 | 11 | 23 | 24 | 0 | 0 | 1 | 29 | 5 |
| M2 | 3 | 5 | 9 | 17 | 19 | 0 | 1 | 2 | 32 | 5 |
| M3 | 2 | 4 | 9 | 16 | 18 | 0 | 2 | 5 | 33 | 6 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 4 | 11 | 24 | 43 | 45 | 0 | 1 | 1 | 26 | 5 |
| M2 | 5 | 9 | 19 | 33 | 37 | 0 | 1 | 0 | 26 | 5 |
| M3 | 4 | 9 | 18 | 32 | 37 | 0 | 4 | 0 | 27 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 3 | 6 | 12 | 16 | 21 | 5 | 6 | 7 | 27 | 8 |
| M5 | 2 | 7 | 11 | 27 | 27 | 0 | 0 | 0 | 33 | 1 |
| M6 | 2 | 4 | 8 | 16 | 17 | 0 | 3 | 4 | 31 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |
| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
| M4 | 6 | 12 | 21 | 33 | 42 | 1 | 2 | 4 | 15 | 10 |
| M5 | 7 | 11 | 27 | 50 | 50 | 0 | 0 | 1 | 33 | 1 |
| M6 | 4 | 8 | 17 | 32 | 32 | 0 | 1 | 1 | 27 | 0 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 3 | 6 | 12 | 16 | 21 | 5 | 6 | 7 | 27 | 8 |
| M5 | 2 | 7 | 11 | 27 | 27 | 0 | 0 | 0 | 33 | 1 |
| M6 | 2 | 4 | 8 | 16 | 17 | 0 | 3 | 4 | 31 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |
| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
| M4 | 6 | 12 | 21 | 33 | 42 | 1 | 2 | 4 | 15 | 10 |
| M5 | 7 | 11 | 27 | 50 | 50 | 0 | 0 | 1 | 33 | 1 |
| M6 | 4 | 8 | 17 | 32 | 32 | 0 | 1 | 1 | 27 | 0 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Task (dependency) trees

| Matrix | Option A #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 3 | 6 | 12 | 16 | 21 | 5 | 6 | 7 | 27 | 8 |
| M5 | 2 | 7 | 11 | 27 | 27 | 0 | 0 | 0 | 33 | 1 |
| M6 | 2 | 4 | 8 | 16 | 17 | 0 | 3 | 4 | 31 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B #Leaf Tasks | | | | | $c_w$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 6 | 12 | 21 | 33 | 42 | 1 | 2 | 4 | 15 | 10 |
| M5 | 7 | 11 | 27 | 50 | 50 | 0 | 0 | 1 | 33 | 1 |
| M6 | 4 | 8 | 17 | 32 | 32 | 0 | 1 | 1 | 27 | 0 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves weight balancing!

# Speed-up vs. Load balancing

- Speed-up computed with respect to the parallel algorithm using the same task tree on a single processor
- In order to measure how $t_i$'s (costs of leaf tasks) are balanced among threads, we use:

$$c_t = \frac{\sigma_t}{\bar{T}}$$

with $\sigma_t$, $\bar{T}$, the std. dev. and the avg. of thread costs, taking into account how tasks have been (dynamically) mapped to threads (i.e., after execution)

The closer $c_t$ to 0, the better the load balancing in the computation of leaves!

# Speed-up vs. Load balancing

| Matrix | Option A Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.87 | 6.37 | 7.19 | 10.18 | 1 | 1 | 11 | 26 | 14 |
| M2 | 1.99 | 3.81 | 7.20 | 7.60 | 12.53 | 0 | 3 | 5 | 28 | 7 |
| M3 | 1.96 | 3.89 | 6.85 | 7.90 | 13.02 | 0 | 4 | 7 | 27 | 9 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.79 | 6.56 | 8.96 | 11.13 | 1 | 2 | 8 | 11 | 7 |
| M2 | 1.99 | 3.81 | 7.08 | 9.70 | 15.91 | 0 | 3 | 6 | 13 | 5 |
| M3 | 1.96 | 3.69 | 7.28 | 7.46 | 13.54 | 1 | 5 | 4 | 13 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Speed-up vs. Load balancing

| Matrix | Option A Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.87 | 6.37 | 7.19 | 10.18 | 1 | 1 | 11 | 26 | 14 |
| M2 | 1.99 | 3.81 | 7.20 | 7.60 | 12.53 | 0 | 3 | 5 | 28 | 7 |
| M3 | 1.96 | 3.89 | 6.85 | 7.90 | 13.02 | 0 | 4 | 7 | 27 | 9 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.79 | 6.56 | 8.96 | 11.13 | 1 | 2 | 8 | 11 | 7 |
| M2 | 1.99 | 3.81 | 7.08 | 9.70 | 15.91 | 0 | 3 | 6 | 13 | 5 |
| M3 | 1.96 | 3.69 | 7.28 | 7.46 | 13.54 | 1 | 5 | 4 | 13 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Speed-up vs. Load balancing

| Matrix | Option A Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.87 | 6.37 | 7.19 | 10.18 | 1 | 1 | 11 | 26 | 14 |
| M2 | 1.99 | 3.81 | 7.20 | 7.60 | 12.53 | 0 | 3 | 5 | 28 | 7 |
| M3 | 1.96 | 3.89 | 6.85 | 7.90 | 13.02 | 0 | 4 | 7 | 27 | 9 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1 | 1.97 | 3.79 | 6.56 | 8.96 | 11.13 | 1 | 2 | 8 | 11 | 7 |
| M2 | 1.99 | 3.81 | 7.08 | 9.70 | 15.91 | 0 | 3 | 6 | 13 | 5 |
| M3 | 1.96 | 3.69 | 7.28 | 7.46 | 13.54 | 1 | 5 | 4 | 13 | 7 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Speed-up vs. Load balancing

| Matrix | Option A | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Speed-up | | | | | $c_t$ (%) | | | | |
| M4 | 1.91 | 3.41 | 6.69 | 7.13 | 11.35 | 4 | 13 | 10 | 26 | 15 |
| M5 | 1.99 | 3.84 | 7.45 | 7.98 | 13.61 | 0 | 2 | 3 | 31 | 4 |
| M6 | 1.88 | 3.25 | 5.38 | 9.69 | 9.92 | 5 | 21 | 30 | 16 | 35 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Speed-up | | | | | $c_t$ (%) | | | | |
| M4 | 1.93 | 3.70 | 6.99 | 10.45 | 13.77 | 3 | 6 | 9 | 7 | 8 |
| M5 | 1.97 | 3.92 | 7.70 | 10.17 | 14.08 | 0 | 0 | 1 | 14 | 3 |
| M6 | 1.94 | 3.35 | 7.22 | 10.55 | 13.38 | 3 | 19 | 8 | 9 | 12 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Speed-up vs. Load balancing

| Matrix | Option A Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 1.91 | 3.41 | 6.69 | 7.13 | 11.35 | 4 | 13 | 10 | 26 | 15 |
| M5 | 1.99 | 3.84 | 7.45 | 7.98 | 13.61 | 0 | 2 | 3 | 31 | 4 |
| M6 | 1.88 | 3.25 | 5.38 | 9.69 | 9.92 | 5 | 21 | 30 | 16 | 35 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 1.93 | 3.70 | 6.99 | 10.45 | 13.77 | 3 | 6 | 9 | 7 | 8 |
| M5 | 1.97 | 3.92 | 7.70 | 10.17 | 14.08 | 0 | 0 | 1 | 14 | 3 |
| M6 | 1.94 | 3.35 | 7.22 | 10.55 | 13.38 | 3 | 19 | 8 | 9 | 12 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Speed-up vs. Load balancing

| Matrix | Option A Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 1.91 | 3.41 | 6.69 | 7.13 | 11.35 | 4 | 13 | 10 | 26 | 15 |
| M5 | 1.99 | 3.84 | 7.45 | 7.98 | 13.61 | 0 | 2 | 3 | 31 | 4 |
| M6 | 1.88 | 3.25 | 5.38 | 9.69 | 9.92 | 5 | 21 | 30 | 16 | 35 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

| Matrix | Option B Speed-up | | | | | $c_t$ (%) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M4 | 1.93 | 3.70 | 6.99 | 10.45 | 13.77 | 3 | 6 | 9 | 7 | 8 |
| M5 | 1.97 | 3.92 | 7.70 | 10.17 | 14.08 | 0 | 0 | 1 | 14 | 3 |
| M6 | 1.94 | 3.35 | 7.22 | 10.55 | 13.38 | 3 | 19 | 8 | 9 | 12 |
| #Procs. | 2 | 4 | 8 | 12 | 16 | 2 | 4 | 8 | 12 | 16 |

Halving #procs. for the same task tree improves load balancing!

# Preconditioner execution time

| Matrix | Serial Algorithm Time (secs.) | Parallel Algorithm Option A Time (secs.) | | | | |
|---|---|---|---|---|---|---|
| M1 | 2.81 | 1.36 | 0.66 | 0.38 | 0.32 | 0.23 |
| M2 | 1.32 | 0.64 | 0.32 | 0.17 | 0.16 | 0.01 |
| M3 | 24.5 | 12.2 | 6.13 | 3.43 | 2.98 | 1.8 |
| #Procs. | 1 | 2 | 4 | 8 | 12 | 16 |
| Matrix | Serial Algorithm Time (secs.) | Parallel Algorithm Option B Time (secs.) | | | | |
| M1 | 2.81 | 1.29 | 0.64 | 0.35 | 0.23 | 0.18 |
| M2 | 1.32 | 0.62 | 0.31 | 0.17 | 0.12 | 0.01 |
| M3 | 24.5 | 12.2 | 6.43 | 3.21 | 2.12 | 1.62 |
| #Procs. | 1 | 2 | 4 | 8 | 12 | 16 |

# Preconditioner execution time

| Matrix | Serial Algorithm Time (secs.) | Parallel Algorithm Option A Time (secs.) | | | | |
|---|---|---|---|---|---|---|
| M4 | 143 | 68.2 | 36.7 | 16.9 | 15.4 | 9.28 |
| M5 | 36.9 | 18.2 | 9.41 | 4.75 | 4.44 | 2.61 |
| M6 | 320 | 152 | 89.4 | 68.5 | 51.8 | 51.2 |
| #Procs. | 1 | 2 | 4 | 8 | 12 | 16 |
| Matrix | Serial Algorithm Time (secs.) | Parallel Algorithm Option B Time (secs.) | | | | |
| M4 | 143 | 64.9 | 30.6 | 15.1 | 9.56 | 7.07 |
| M5 | 36.9 | 18.3 | 9.04 | 4.61 | 3.40 | 2.46 |
| M6 | 320 | 150 | 110 | 70.2 | 64.9 | 51.3 |
| #Procs. | 1 | 2 | 4 | 8 | 12 | 16 |

# Speed-up vs. Load balancing

| Matrix | Option A | | | |
|---|---|---|---|---|
| | Time (secs.) | Speed-up | $c_t$ (%) | $c_w$ (%) |
| M6 | 51.2 | 9.92 | 35 | 7 |
| #Procs. | 16 | | | |

*num_leaves* $\approx p$ and heterogeneous costs $\rightarrow$ bad performance

| Matrix | Option B | | | |
|---|---|---|---|---|
| | Time (secs.) | Speed-up | $c_t$ (%) | $c_w$ (%) |
| M5 | 2.46 | 14.08 | 3 | 1 |
| #Procs. | 16 | | | |

Good load balancing $\rightarrow$ good performance

Good news: the leaves concentrate the major part of the cost!

# Outline

1. Motivation and Introduction
2. Overview of ILUPACK
   - Incomplete LU factorization
   - ILUPACK
3. Computation of Parallel Preconditioners
   - Trees and sparse matrices
   - Parallel algorithm
   - Parallel implementation
4. Experimental Results
   - Experimental Framework
   - Task (dependency) trees
   - Preconditioner computation results
5. Concluding Remarks

# Conclusions

- We have presented a parallel preconditioner for the iterative solution of sparse linear systems on parallel architectures with shared-memory
- The preconditioner shares the principles with ILUPACK
- MLND ordering, task splitting, and dynamic scheduling are employed to improve load balancing, raise the degree of concurrency, and reduce the execution time
- In most cases, the parallel performance is satisfactory

# Future Work

- To compare and contrast the numerical properties of the preconditioner in ILUPACK and our parallel preconditioner.
- To parallelize the application of the preconditioner to the system.
- To exploit data locality on CC-NUMA architectures.
- To design parallel preconditioners for non-SPD linear systems.
- To develop MPI parallel preconditioners.

# Questions?