Parallel Preconditioning of Linear Systems based on ILUPACK for Multithreaded Architectures

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

March, 2009

Motivation and Introduction

- We consider the **efficient** solution of linear systems

$$Ax = b$$

  where
  - $A \in \mathbb{R}^{n,n}$ is a **large** and **sparse** SPD coefficient matrix
  - $x \in \mathbb{R}^n$ is the sought-after solution
  - $b \in \mathbb{R}^n$ is a given r.h.s. vector
- The solution of this problem
  - Arises in very many large-scale application problems
  - Frequently the most time-consuming part of the computation

Motivation and Introduction

We propose to solve $Ax = b$ **iteratively** using

- Preconditioned Conjugate Gradient (PCG)
  - Among the best iterative approaches to solve SPD linear systems
- ILUPACK preconditioning techniques
  - Incomplete Cholesky (IC) factorization-based preconditioners
    $\rightarrow$ Multilevel IC (MIC) preconditioners: **Inverse-based** approach
  - Successful for a wide range of large-scale application problems with up to million of equations

Motivation and Introduction

- The large scale of the problems faced by ILUPACK makes it evident the need to reduce the time-to-solution via parallel computing techniques
- Development of ILUPACK-based solvers on shared-memory architectures
  - Parallelism revealed by nested dissection on adjacency graph G_A
    $\longrightarrow$ recursive application of vertex separator partitioning
  - Dynamic scheduling to improve load-balancing during execution
    $\longrightarrow$ run-time tuned for a specific platform/application problem
  - PCG operations conformal with the MIC parallelization
  - Implemented using standard tools: threads and OpenMP

Focus:

- Parallelization approach preserves semantics of the MIC
- Delivers a high-degree of concurrence for shared-memory architectures

Numerical results for two large-scale 3D application problems

Outline

Outline

IC factorization preconditioners

- The (root-free) sparse Cholesky decomp. computes $L \in \mathbb{R}^{n,n}$, sparse unit lower triangular, and $D \in \mathbb{R}^{n,n}$, diagonal with positive entries s.t.

$$A = LDL^T$$

- In the IC factorization A is factored as

$$A = \tilde{L}\tilde{D}\tilde{L}^T + E$$

  where $E \in \mathbb{R}^{n,n}$ is a "small" perturbation matrix consisting of those entries dropped during factorization
- $M = \tilde{L}\tilde{D}\tilde{L}^T$ is applied to the original system $Ax = b$ in order to accelerate the convergence of the PCG solver

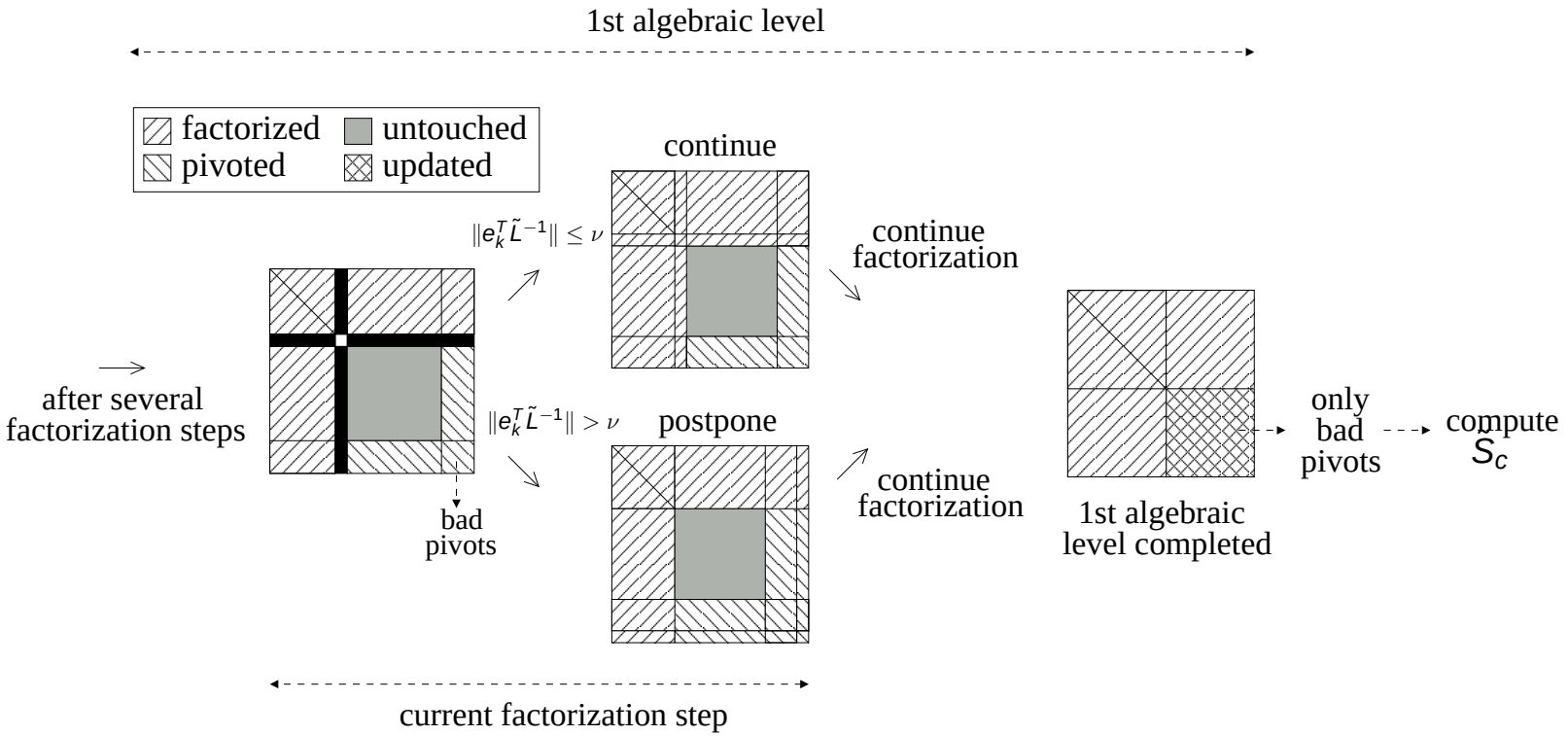
Inverse-based IC factorization preconditioners

- PCG solver converges quickly if the preconditioned matrix is close to I

$$\tilde{D}^{-1/2}\tilde{L}^{-1}A\tilde{L}^{-T}\tilde{D}^{-1/2} = I + \tilde{D}^{-1/2}\underbrace{\tilde{L}^{-1}E\tilde{L}^{-T}}_{F}\tilde{D}^{-1/2}$$

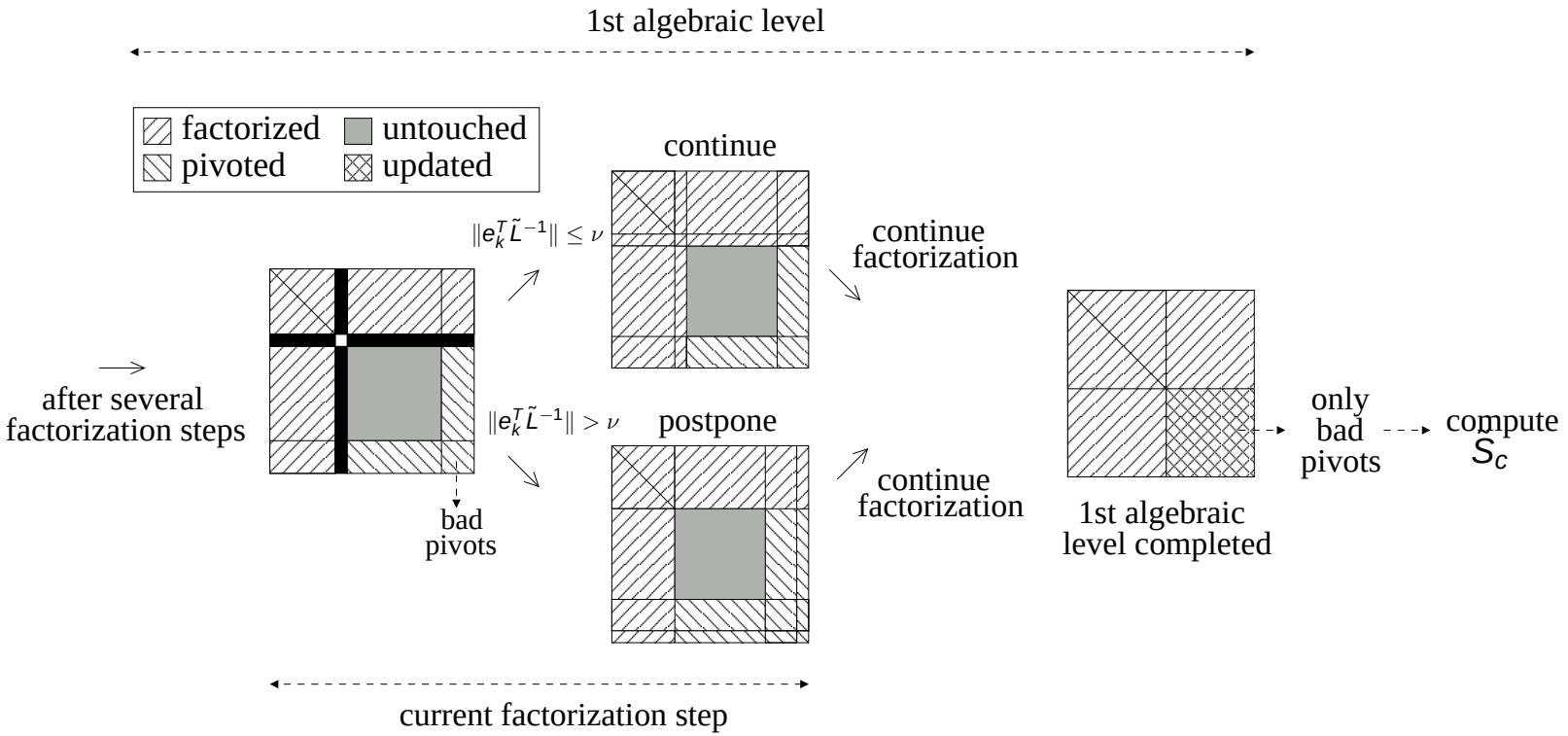
- $\|\tilde{D}^{-1/2}F\tilde{D}^{-1/2}\|$ must be “small” to construct effective preconditioners
- Inverse-based IC factorizations construct $M = \tilde{L}\tilde{D}\tilde{L}^T$ s.t. $\|\tilde{L}^{-1}\| \leq \nu$
  - $\nu = 5$ or $\nu = 10$ are good choices for many problems
  - Pivoting and **multilevel** methods are required in general

Multilevel Inverse-based IC factorization



- The IC decomp. employs **Inverse-based** pivoting
  - → Row/Col. k of $\tilde{L}$ / $\tilde{L}^T$ obtained at step k along with estimation $t_k \approx \|e_k^T \tilde{L}^{-1}\|$
  - → Row/Col. s.t. $t_k > \nu$ are moved to the bottom/right-end of the matrix
  - → Considered "bad" pivots with respect to the constraint $\|\tilde{L}^{-1}\| \le \nu$
- The IC factorization **stops** when the trailing principal only contains bad pivots
  - → The computation of the approximate Schur complement $\tilde{S}_c$ completes . . .

Multilevel Inverse-based IC factorization



- . . . a **partial** IC decomp. of a permuted system with $\tilde{L}_B^{-1}$ satisfying $\|\tilde{L}_B^{-1}\| \lessapprox \nu$

$$\hat{P}^T \hat{A} \hat{P} \equiv \begin{pmatrix} B & F^T \\ F & C \end{pmatrix} = \begin{pmatrix} \tilde{L}_B & 0 \\ \tilde{L}_F & I \end{pmatrix} \begin{pmatrix} \tilde{D}_B & 0 \\ 0 & \tilde{S}_c \end{pmatrix} \begin{pmatrix} \tilde{L}_B^T & 0 \\ \tilde{L}_F^T & I \end{pmatrix} + E$$

- The whole method is restarted on $\tilde{S}_c$ leading to a **multilevel** approach

Multilevel Inverse-based IC factorization

The MIC factorization can be expressed algorithmically as follows

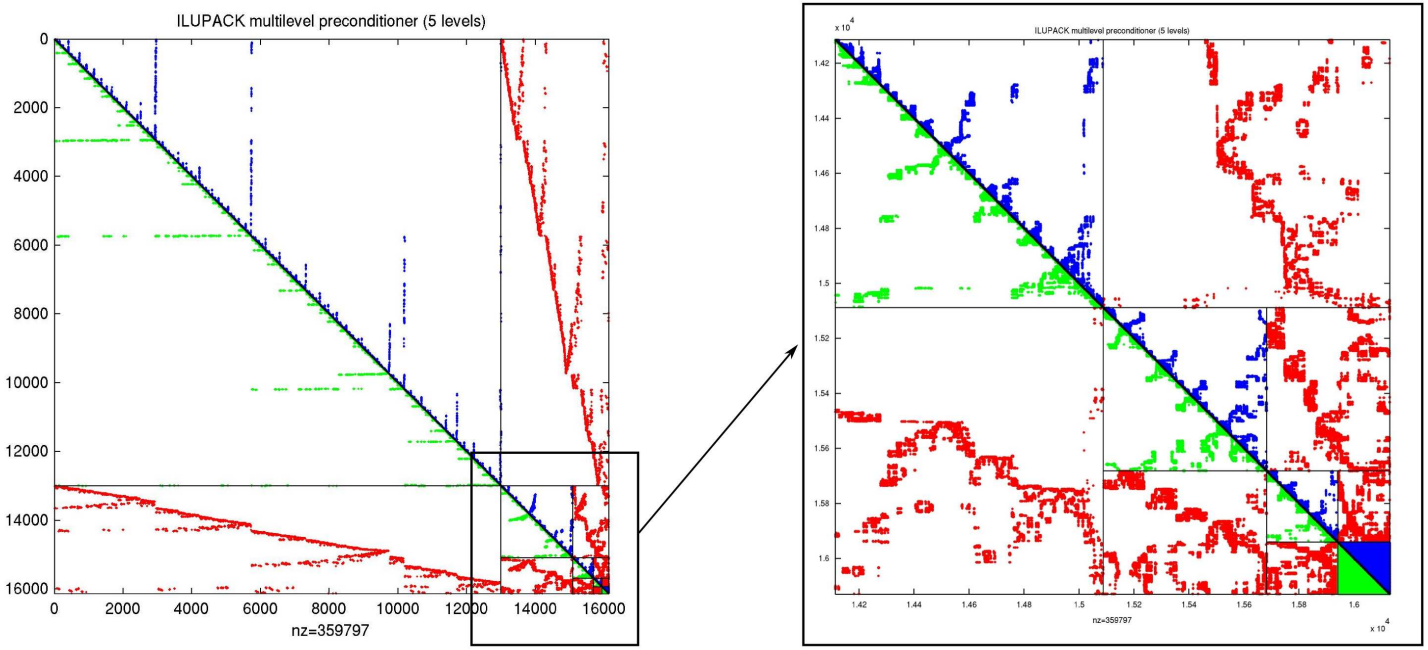
1. Reorder $A \rightarrow P^T A P = \hat{A}$ by some fill-reducing ordering matrix P
2. Compute a partial IC decomp. as by-product of inverse-based pivoting

$$\hat{P}^T \hat{A} \hat{P} \equiv \begin{pmatrix} B & F^T \\ F & C \end{pmatrix} = \begin{pmatrix} \tilde{L}_B & 0 \\ \tilde{L}_F & I \end{pmatrix} \begin{pmatrix} \tilde{D}_B & 0 \\ 0 & \tilde{S}_c \end{pmatrix} \begin{pmatrix} \tilde{L}_B^T & 0 \\ \tilde{L}_F^T & I \end{pmatrix} + E$$

3. Proceed to the next level by repeating steps 1 and 2 with $A \equiv \tilde{S}_c$ until $\tilde{S}_c$ is void or “dense enough” to be handled by a dense Cholesky solver

Multilevel Inverse-based IC factorization

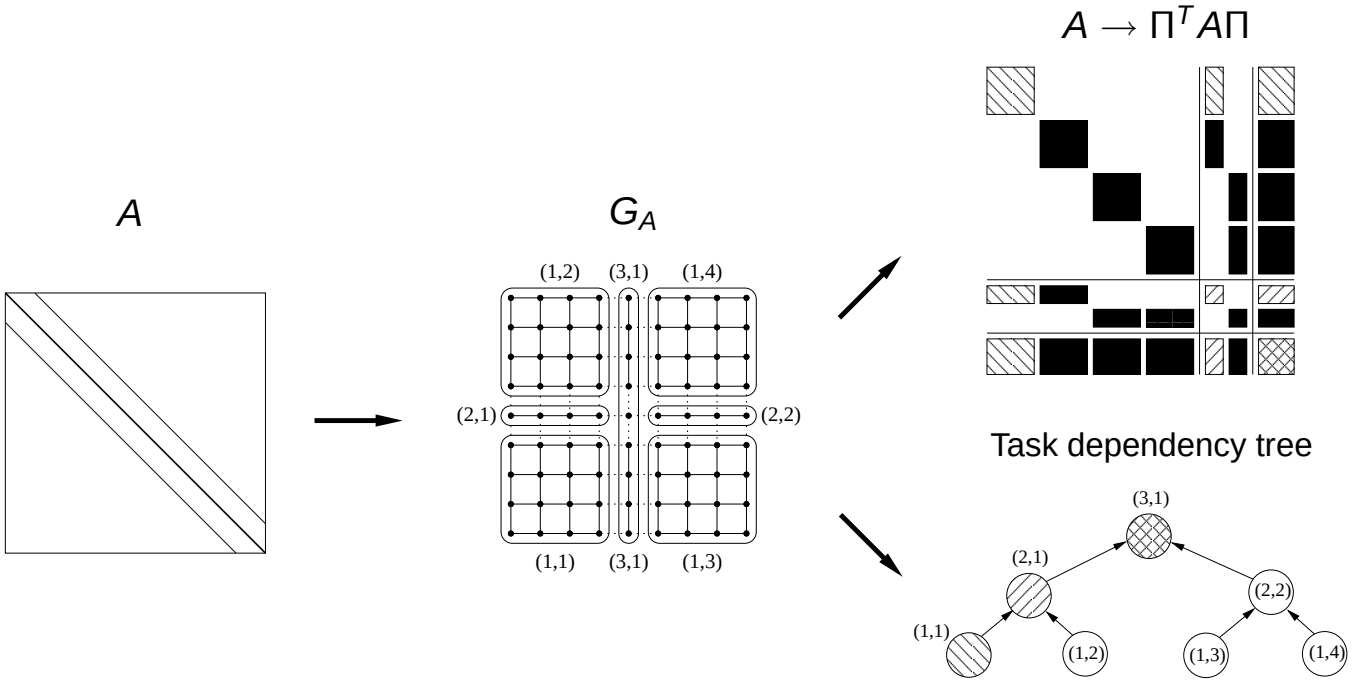
MIC factorization of five-point matrix arising from Laplace PDE discretization



$\nu = 5$ and $\tau = 10^{-3}$ leads to a MIC factorization with 5 levels

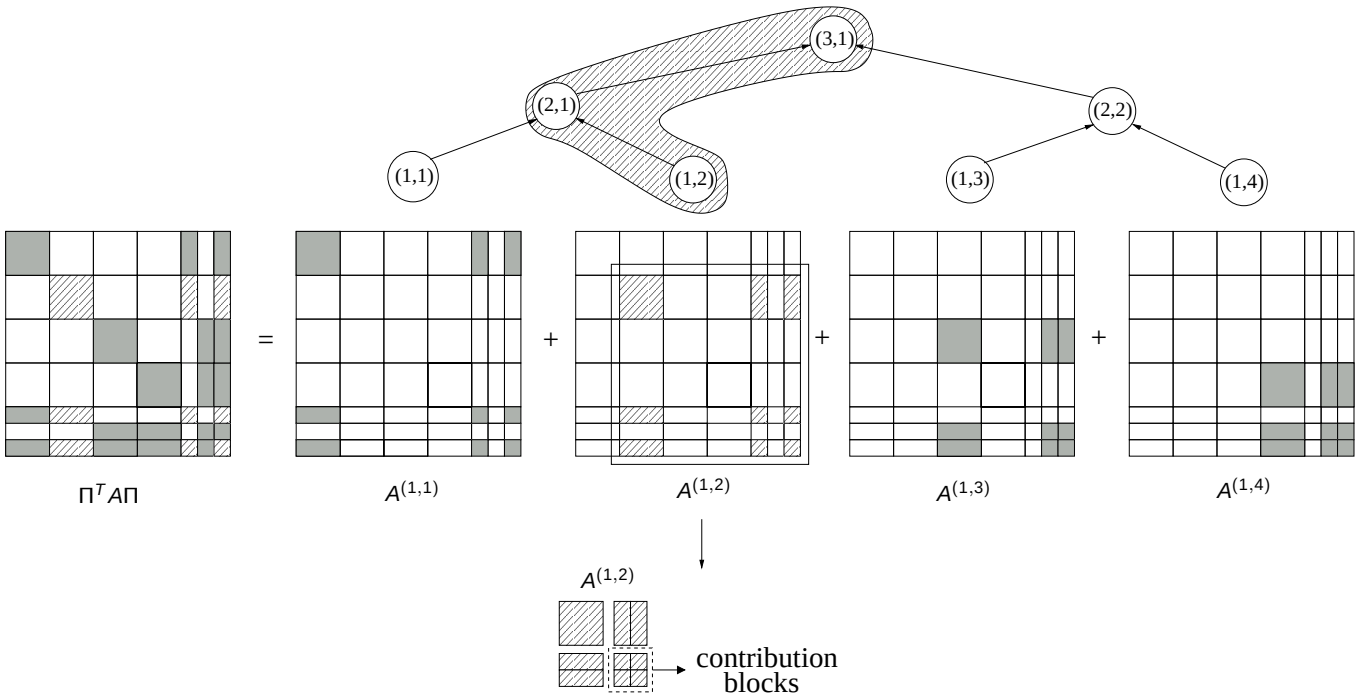
Outline

Nested dissection preprocessing step



- Parallelism revealed by nested dissection on adjacency graph G_A
  $\rightarrow$ fast and efficient **multilevel** (MLND) variants provided by SCOTCH package
- Parallelization driven by the dependencies “captured” in the task tree
  $\rightarrow$ subdomains **first** eliminated independently, **then** separators (2,1) and (2,2) in parallel, **finally** the root separator (3,1)

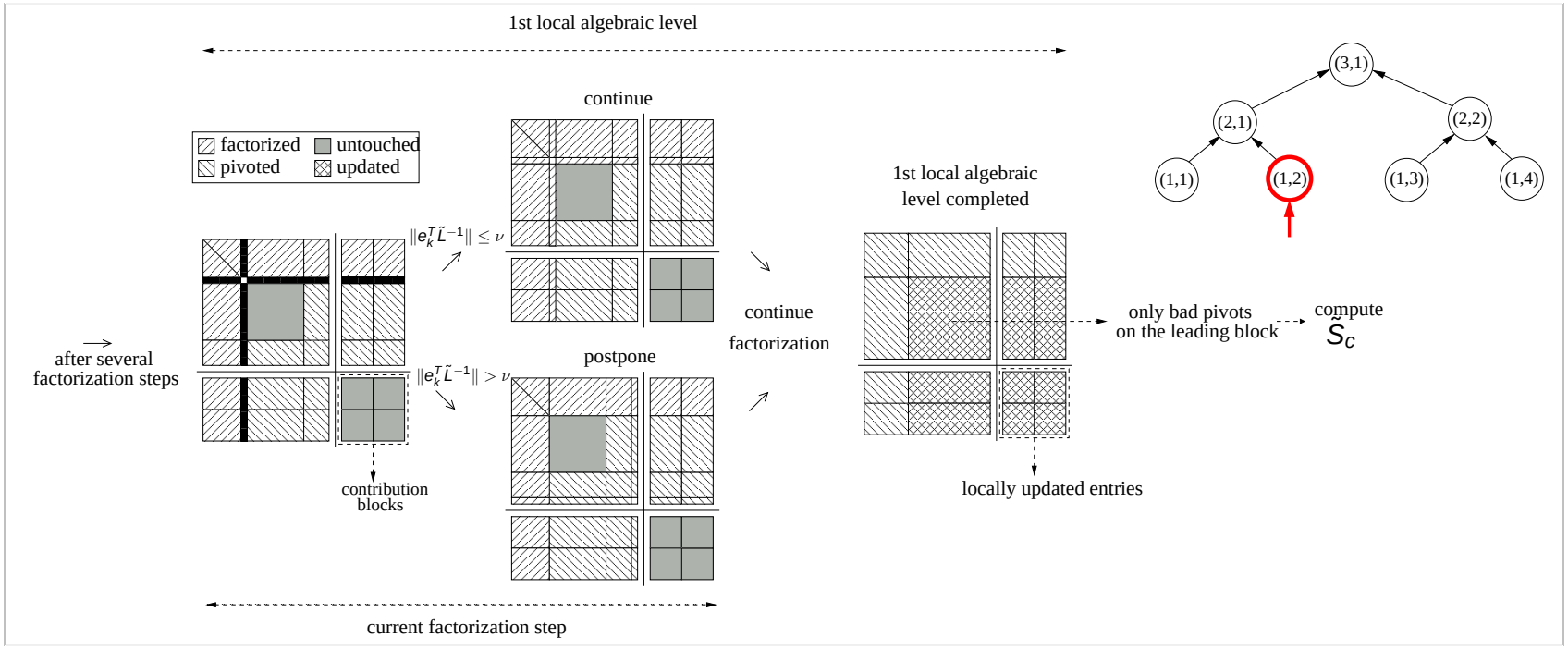
Parallel Multilevel Inverse-based IC factorization



- Disassemble $\Pi^T A \Pi$ into a sum of submatrices, one per leaf node
- The factorization of the diagonal blocks of $\Pi^T A \Pi$ can proceed in parallel as well as the updates on the blocks corresponding to all ancestor nodes
- The contribution blocks of each submatrix hold the updates on ancestor nodes during the factorization of the leading block

Parallel Multilevel Inverse-based IC factorization

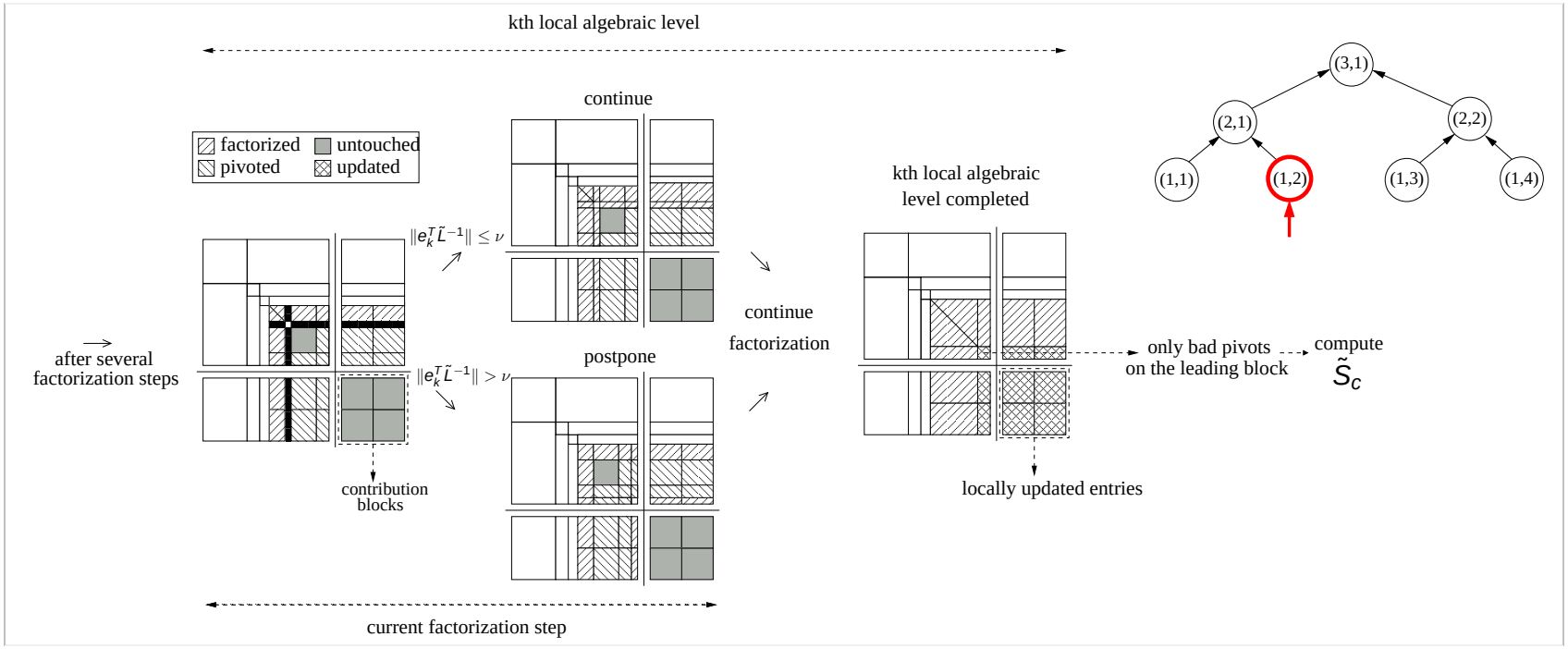
The tasks compute a local MIC **restricted to the leading block** of the submatrix



- The IC decomp. employs **restricted** Inverse-Based pivoting
  → Bad pivots are **only moved** to the bottom/right-end of the leading block
- The IC decomp. stops if the trailing of the **leading block** only contains bad pivots
  → Postponed pivots still “resolved” inside the task by entering next MIC level

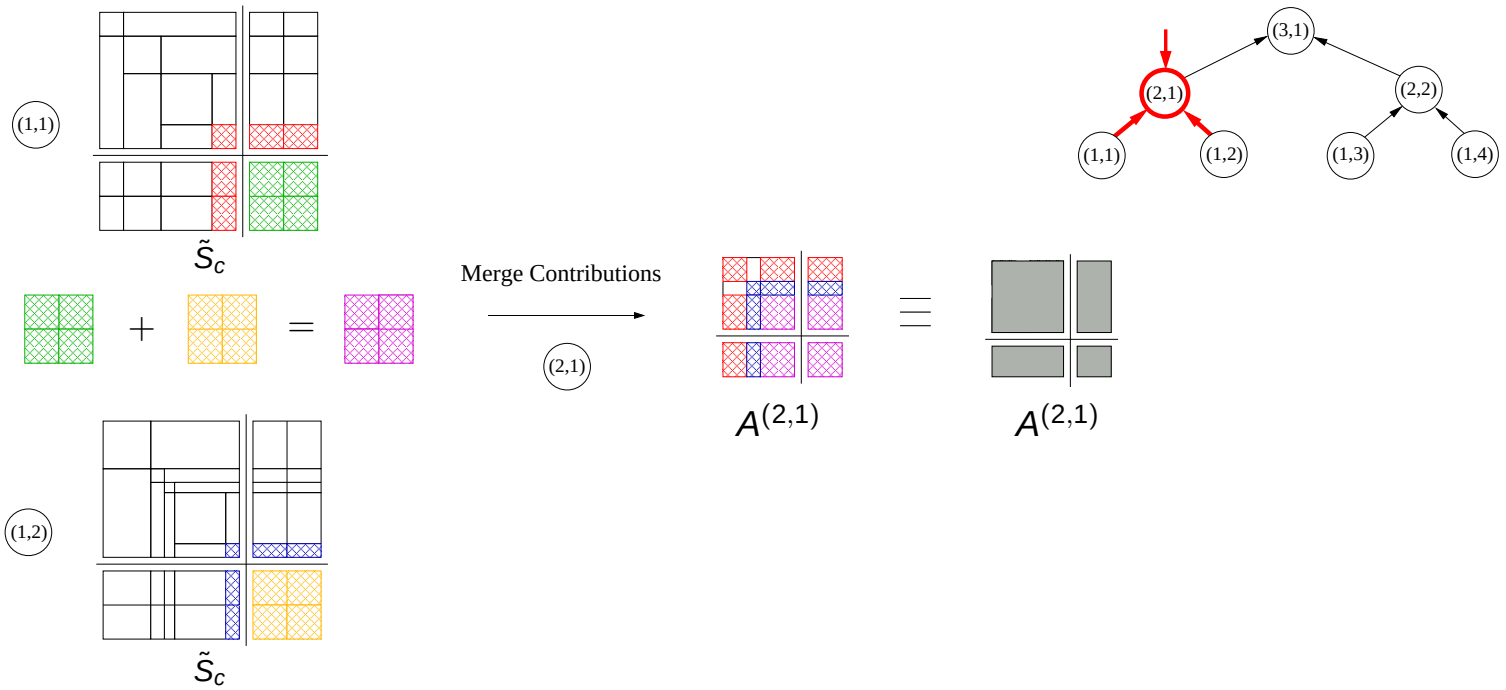
Parallel Multilevel Inverse-based IC factorization

The tasks compute a local MIC **restricted to the leading block** of the submatrix



- When the set of bad pivots is “small enough” the local MIC **stops**
  → The task “sends” the local contributions to the parent task

Parallel Multilevel Inverse-based IC factorization



- The parent task merge contributions resulting from its children
- A new submatrix is constructed for the local MIC decomp. of the separator task
  - $\longrightarrow$ Bad pivots resulting from its children moved on top of the leading block
  - $\longrightarrow$ Contributions blocks added

Parallel Multilevel Inverse-based IC factorization

The computation of a **leaf node** (i,j) can be expressed algorithmically as follows

1. Construct task submatrix $A^{(i,j)}$ from $\Pi^T A \Pi$
2. Reorder $A^{(i,j)} \rightarrow (P^{(i,j)})^T A^{(i,j)} P^{(i,j)} = \hat{A}^{(i,j)}$ by some fill-reducing ordering matrix $P^{(i,j)}$ restricted to the leading block of $A^{(i,j)}$
3. Compute a partial IC decomp. as by-product of inverse-based pivoting restricted to the leading block of $A^{(i,j)}$
4. Proceed to the next local level by repeating steps 2 and 3 until the set of bad pivots is “small enough”
5. “Send” local contributions to the parent task

Parallel Multilevel Inverse-based IC factorization

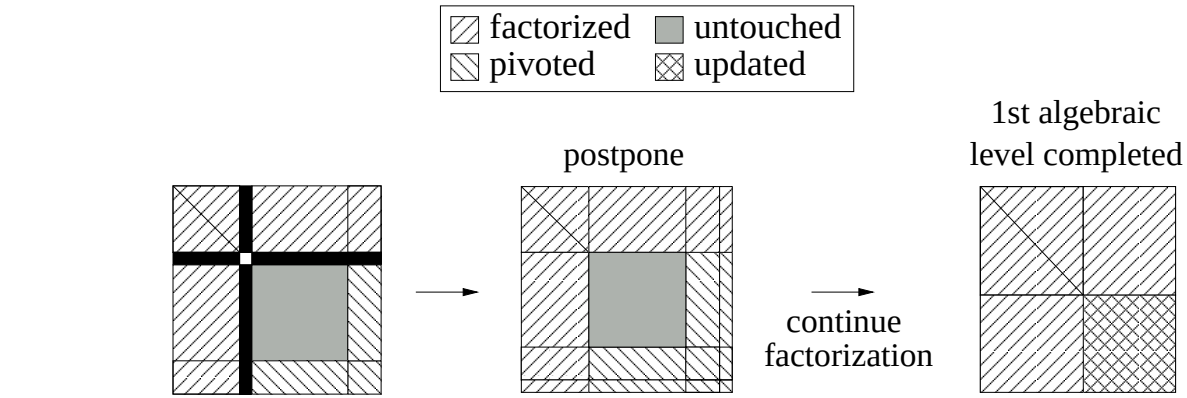
For a **separator intermediate task** (i,j) ...

1. Construct task submatrix $A^{(i,j)}$ by merging contributions of its children
2. Reorder $A^{(i,j)} \rightarrow (P^{(i,j)})^T A^{(i,j)} P^{(i,j)} = \hat{A}^{(i,j)}$ by some fill-reducing ordering matrix $P^{(i,j)}$ restricted to the leading block of $A^{(i,j)}$
3. Compute a partial IC decomp. as by-product of inverse-based pivoting restricted to the leading block of $A^{(i,j)}$
4. Proceed to the next local level by repeating steps 2 and 3 until the set of bad pivots is “small enough”
5. “Send” local contributions to the parent task

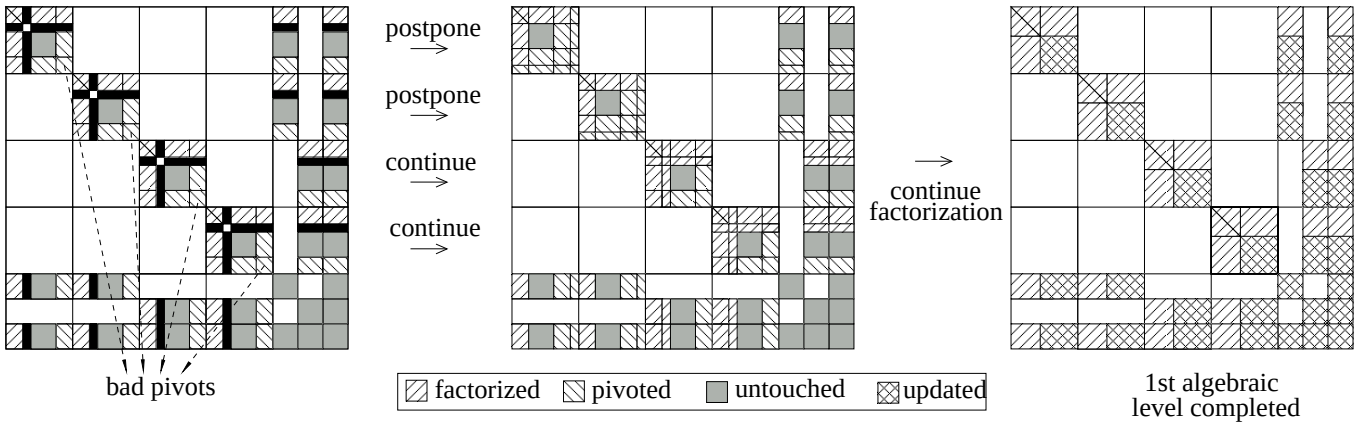
Parallel Multilevel Inverse-based IC factorization

For the **separator root task** (i,j) ...

1. Construct task submatrix $A^{(i,j)}$ by merging contributions of its children
2. Reorder $A^{(i,j)} \rightarrow (P^{(i,j)})^T A^{(i,j)} P^{(i,j)} = \hat{A}^{(i,j)}$ by some fill-reducing ordering matrix $P^{(i,j)}$ restricted to the leading block of $A^{(i,j)}$
3. Compute a partial IC decomp. as by-product of inverse-based pivoting restricted to the leading block of $A^{(i,j)}$
4. Proceed to the next local level by repeating steps 2 and 3 until the approximate Schur complement is “dense enough” or void
5. MIC factorization finished



MIC



Parallel MIC

How does MIC behave when applied to a system reordered by MLND?

- $\|e_k^T \tilde{L}^{-1}\|_\infty$ defined as $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
- Estimators $t_k \approx \|e_k^T \tilde{L}^{-1}\|_\infty$ are computed step by step along with IC decomp.
  $\rightarrow$ Compute specific vector $\hat{z}$ s.t. $t_k = |e_k^T \tilde{L}^{-1} \hat{z}|$ close to $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
  $\rightarrow$ $\hat{z}$ is computed while solving $\tilde{L} y = \hat{z}$ by column-oriented forward substitution
- At step k of this forward substitution process ...
  1. $t_k = |y_k|$ is computed by selecting $\hat{z}_k = 1$ or $\hat{z}_k = -1$
  2. Vector y is updated by $y_j := y_j - \tilde{l}_{jk} y_k$ for $j > k$ s.t. $\tilde{l}_{jk} \neq 0$
- $\hat{z}_k$ (y_k) is selected to maximize $|y_j|$ after the update
  $\rightarrow$ Likely that $|y_j|$ becomes large if involved in many updates
  $\rightarrow$ **Separator nodes expected to lead to more fill-in than subdomain nodes**

When the MIC is applied to a system reordered by MLND it is likely that separator nodes are initially rejected more often

How does MIC behave when applied to a system reordered by MLND?

- $\|e_k^T \tilde{L}^{-1}\|_\infty$ defined as $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
- Estimators $t_k \approx \|e_k^T \tilde{L}^{-1}\|_\infty$ are computed step by step along with IC decomp.
  $\rightarrow$ Compute specific vector $\hat{z}$ s.t. $t_k = |e_k^T \tilde{L}^{-1} \hat{z}|$ close to $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
  $\rightarrow$ $\hat{z}$ is computed while solving $\tilde{L}y = \hat{z}$ by column-oriented forward substitution
- At step k of this forward substitution process . . .
  1. $t_k = |y_k|$ is computed by selecting $\hat{z}_k = 1$ or $\hat{z}_k = -1$
  2. Vector y is updated by $y_j := y_j - \tilde{l}_{jk} y_k$ for $j > k$ s.t. $\tilde{l}_{jk} \neq 0$
- $\hat{z}_k$ (y_k) is selected to maximize $|y_j|$ after the update
  $\rightarrow$ Likely that $|y_j|$ becomes large if involved in many updates
  $\rightarrow$ **Separator nodes expected to lead to more fill-in than subdomain nodes**

When the MIC is applied to a system reordered by MLND it is likely that separator nodes are initially rejected more often

How does MIC behave when applied to a system reordered by MLND?

- $\|e_k^T \tilde{L}^{-1}\|_\infty$ defined as $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
- Estimators $t_k \approx \|e_k^T \tilde{L}^{-1}\|_\infty$ are computed step by step along with IC decomp.
  $\rightarrow$ Compute specific vector $\hat{z}$ s.t. $t_k = |e_k^T \tilde{L}^{-1} \hat{z}|$ close to $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
  $\rightarrow$ $\hat{z}$ is computed while solving $\tilde{L}y = \hat{z}$ by column-oriented forward substitution
- At step k of this forward substitution process ...
  1. $t_k = |y_k|$ is computed by selecting $\hat{z}_k = 1$ or $\hat{z}_k = -1$
  2. Vector y is updated by $y_j := y_j - \tilde{l}_{jk} y_k$ for $j > k$ s.t. $\tilde{l}_{jk} \neq 0$
- $\hat{z}_k$ (y_k) is selected to maximize $|y_j|$ after the update
  $\rightarrow$ Likely that $|y_j|$ becomes large if involved in many updates
  $\rightarrow$ **Separator nodes expected to lead to more fill-in than subdomain nodes**

When the MIC is applied to a system reordered by MLND it is likely that separator nodes are initially rejected more often

How does MIC behave when applied to a system reordered by MLND?

- $\|e_k^T \tilde{L}^{-1}\|_\infty$ defined as $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
- Estimators $t_k \approx \|e_k^T \tilde{L}^{-1}\|_\infty$ are computed step by step along with IC decomp.
  $\rightarrow$ Compute specific vector $\hat{z}$ s.t. $t_k = |e_k^T \tilde{L}^{-1} \hat{z}|$ close to $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
  $\rightarrow$ $\hat{z}$ is computed while solving $\tilde{L}y = \hat{z}$ by column-oriented forward substitution
- At step k of this forward substitution process ...
  1. $t_k = |y_k|$ is computed by selecting $\hat{z}_k = 1$ or $\hat{z}_k = -1$
  2. Vector y is updated by $y_j := y_j - \tilde{l}_{jk} y_k$ for $j > k$ s.t. $\tilde{l}_{jk} \neq 0$
- $\hat{z}_k$ (y_k) is selected to maximize $|y_j|$ after the update
  $\rightarrow$ Likely that $|y_j|$ becomes large if involved in many updates
  $\rightarrow$ **Separator nodes expected to lead to more fill-in than subdomain nodes**

When the MIC is applied to a system reordered by MLND it is likely that separator nodes are initially rejected more often

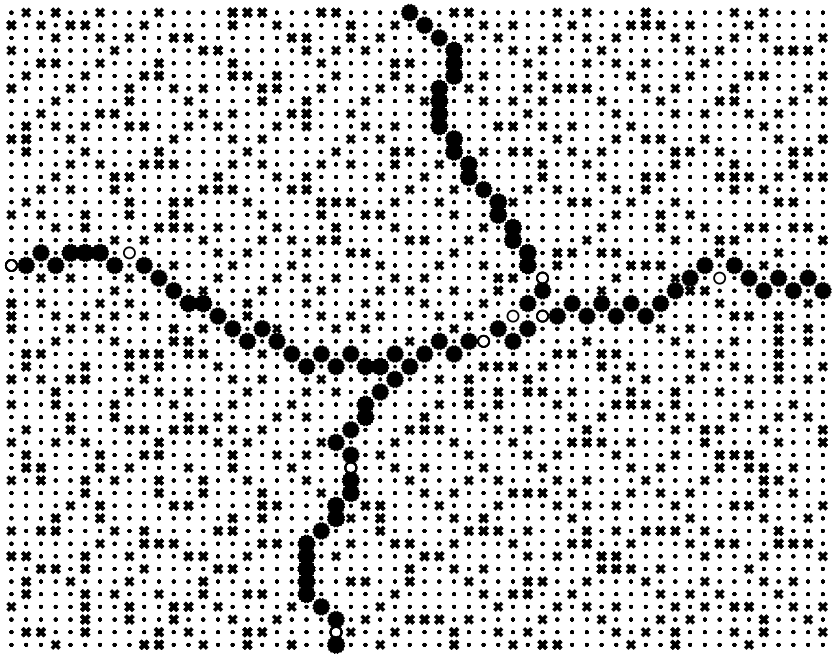
How does MIC behave when applied to a system reordered by MLND?

- $\|e_k^T \tilde{L}^{-1}\|_\infty$ defined as $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
- Estimators $t_k \approx \|e_k^T \tilde{L}^{-1}\|_\infty$ are computed step by step along with IC decomp.
  $\rightarrow$ Compute specific vector $\hat{z}$ s.t. $t_k = |e_k^T \tilde{L}^{-1} \hat{z}|$ close to $\max_{\|z\|_\infty=1} |e_k^T \tilde{L}^{-1} z|$
  $\rightarrow$ $\hat{z}$ is computed while solving $\tilde{L}y = \hat{z}$ by column-oriented forward substitution
- At step k of this forward substitution process ...
  1. $t_k = |y_k|$ is computed by selecting $\hat{z}_k = 1$ or $\hat{z}_k = -1$
  2. Vector y is updated by $y_j := y_j - \tilde{l}_{jk} y_k$ for $j > k$ s.t. $\tilde{l}_{jk} \neq 0$
- $\hat{z}_k$ (y_k) is selected to maximize $|y_j|$ after the update
  $\rightarrow$ Likely that $|y_j|$ becomes large if involved in many updates
  $\rightarrow$ **Separator nodes expected to lead to more fill-in than subdomain nodes**

When the MIC is applied to a system reordered by MLND it is likely that separator nodes are initially rejected more often

MIC applied to MLND-reordered finite-difference grid discretization of 2D Laplace PDE

- 1st MIC level
  → **87**% of separator nodes rejected, **only** 30% of subdomain nodes rejected



- **separator node rejected**
- ○ separator node accepted
- × subdomain node rejected
- · subdomain node accepted

- 2nd MIC level
  → **92**% of separator nodes rejected, **only** 35% of subdomain nodes rejected
- This tendency continues until the **bulk** of subdomain nodes has been accepted
  → Only after that the MIC begins to accept most of the separator nodes

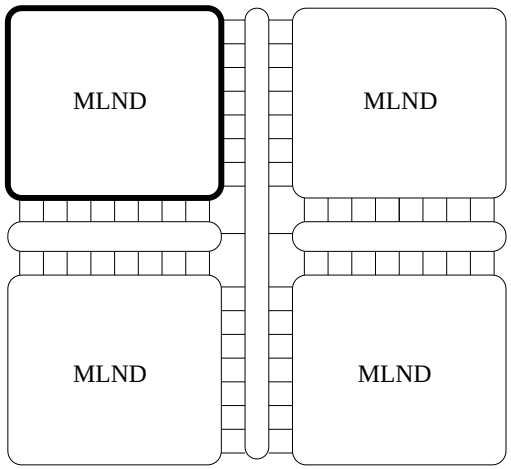
In the parallel MIC separator nodes are rejected by construction

Outline

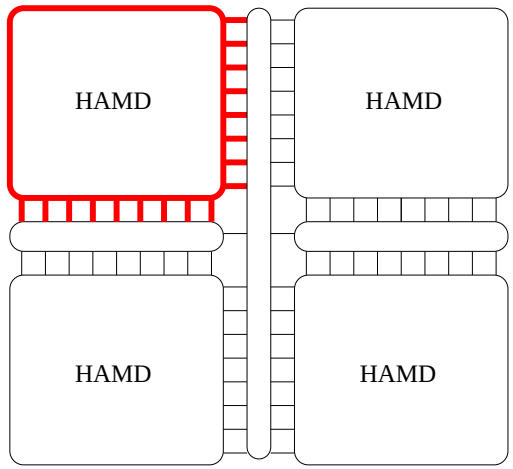
Experimental Framework

- SGI Altix 350 ccNUMA shared-memory multiprocessor:
  - 8 nodes, 2-processor-per-node, Intel Itanium2 1.5 GHz
  - 32 GBytes of RAM shared via a SGI NUMAlink interconnect
    $\rightarrow$ 4 GBytes per node
- Intel Compiler OpenMP 2.5 compliance
  $\rightarrow$ -O3 optimization level
- One thread binded per CPU + no thread migration during execution
- IEEE double precision

Experimental Framework



ND-HAMD-A



ND-HAMD-B

Two different approaches for the MLND preprocessing step and MIC reordering:

- ND-HAMD-A:
  - Subdomains reordered by MLND recursively in the preprocessing step
  - HAMD as MIC reordering **except** for the initial level of the leaf nodes
- ND-HAMD-B:
  - Truncated MLND stops if the degree of hw. parallelism matched
  - HAMD as MIC reordering
    → Subdomains ordered by HAMD at initial MIC level of the leaf nodes
    → This can be already done in parallel exploiting implicit parallelism

First Example Benchmark

We consider the Laplacian PDE equation

$$-\Delta u = f$$

in a $3D$ unit cube $\Omega = [0,1]^3$ with Dirichlet boundary conditions $u = g$ on $\partial\Omega$

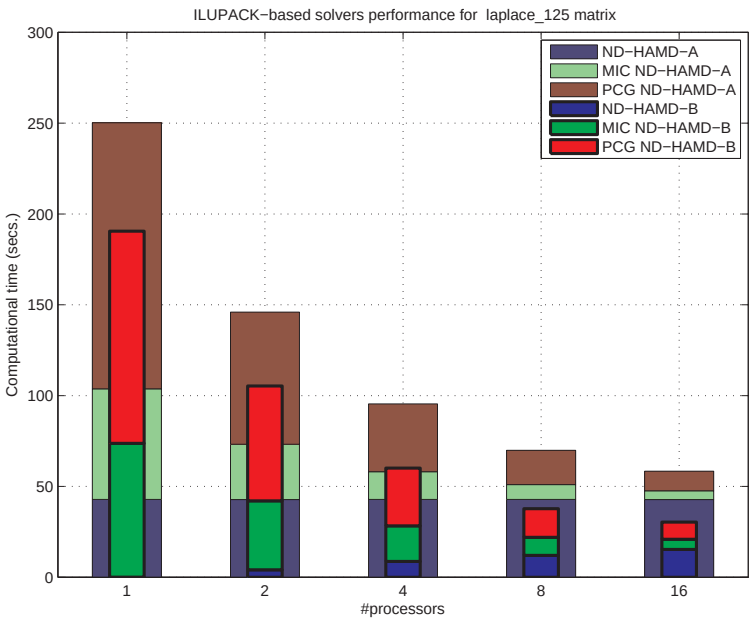
For the discretization . . .

- Ω replaced by a $N_x \times N_y \times N_z$ uniform grid
- Δu approximated by centered finite differences
  $\rightarrow$ seven-point-star difference stencil

We consider four SPD benchmark linear systems from this problem

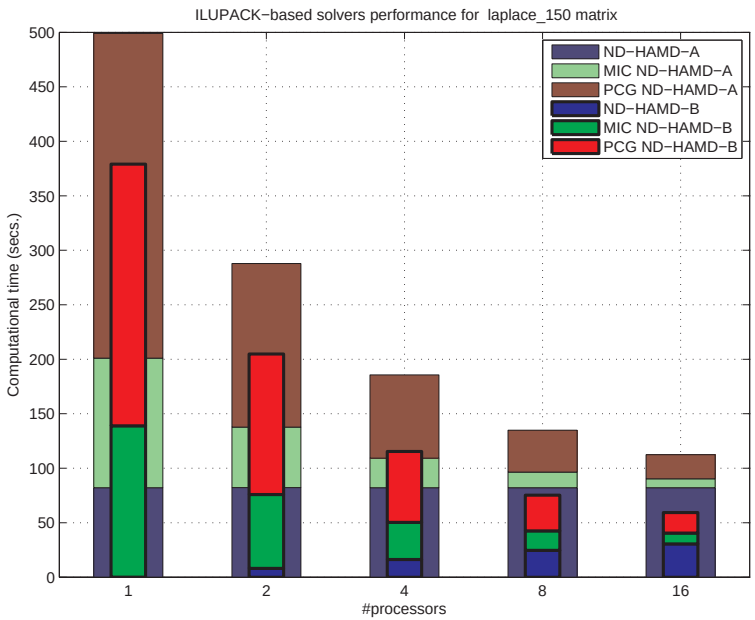
$N_x \times N_y \times N_z$	n	nnz_A/n
$100 \times 100 \times 100$	1,000,000	7
$125 \times 125 \times 125$	1,953,125	7
$150 \times 150 \times 150$	3,375,000	7
$200 \times 200 \times 200$	8,000,000	7

ILUPACK-based solvers execution time



$125 \times 125 \times 125$ Grid

$n = 1{,}953{,}125$



$150 \times 150 \times 150$ Grid

$n = 3{,}375{,}000$

n	p	MLND Option ND-HAMD-X	MLND T (sec.)	MIC nnz_L $\times 10^{-6}$	MIC T_p (sec.)	MIC S_p (T_1/T_p)	PCG #Iter.	PCG T_p (sec.)	PCG S_p (T_1/T_p)
100^3	1	A	19.8	15.0	30.0	1.0	64	61.3	1.0
	8	A	19.8	15.2	4.0	7.5	67	7.4	8.3
	16	A	19.8	15.3	2.5	12.3	67	4.3	14.1
	1	B	0.0	13.6	36.0	1.0	57	49.9	1.0
	8	B	5.7	14.0	4.9	7.4	63	6.6	7.6
	16	B	7.1	14.2	2.9	12.5	66	4.0	12.2
125^3	1	A	42.8	29.5	60.9	1.0	78	146.0	1.0
	8	A	42.8	29.8	8.2	7.5	81	19.4	7.5
	16	A	42.8	30.0	4.8	12.7	82	10.7	13.7
	1	B	0.0	26.8	73.7	1.0	68	116.8	1.0
	8	B	12.0	27.5	10.0	7.4	72	15.8	7.4
	16	B	15.3	28.0	5.6	13.2	78	9.4	12.4
150^3	1	A	82.2	51.1	108.0	1.0	90	296.8	1.0
	8	A	82.2	51.8	14.3	7.5	91	38.5	7.7
	16	A	82.2	52.0	8.1	13.3	93	21.5	13.8
	1	B	0.0	46.6	138.9	1.0	79	240.1	1.0
	8	B	24.7	47.8	17.8	7.8	83	32.8	7.3
	16	B	30.6	48.2	10.0	14.0	87	18.9	12.7
200^3	1	A	247.0	121.9	303.3	1.0	117	1,332.0	1.0
	8	A	247.0	123.2	35.2	8.6	118	122.2	10.9
	16	A	247.0	123.9	19.6	15.5	117	69.4	19.2
	1	B	0.0	111.1	366.0	1.0	103	1,010.2	1.0
	8	B	89.7	113.4	45.9	8.0	107	104.2	9.7
	16	B	108.0	114.6	24.9	14.7	107	57.7	17.5

number of nonzero entries of IC factor

number of PCG iterations

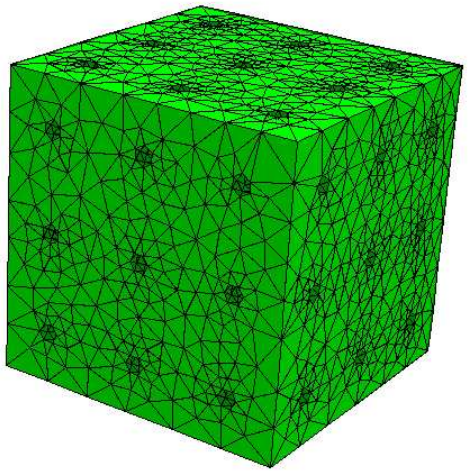
- The nnz slightly changes and # of iterations only increase slightly with p
  $\rightarrow$ **the parallelization approach preserves the semantics of the MIC**

Second Example Benchmark

The second example addresses an irregular 3*D* PDE problem

$$-\text{div}\,(A\ \text{grad}\ u) = f$$

in Ω where $A(x, y, z)$ is chosen with positive random coefficients



Ω

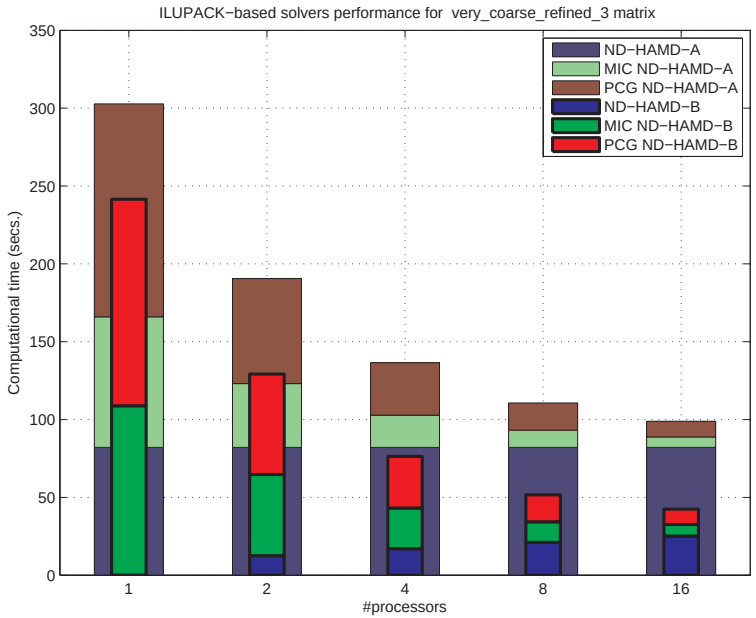
For the discretization . . .

- Linear finite elements are used
- Ω replaced by NETGEN-tool-generated mesh
  $\longrightarrow$ 5 levels of mesh refinement: VC, C, M, F, VF
  $\longrightarrow$ Each mesh refined further up to 3 times

We consider 12 SPD benchmark linear systems from this second example

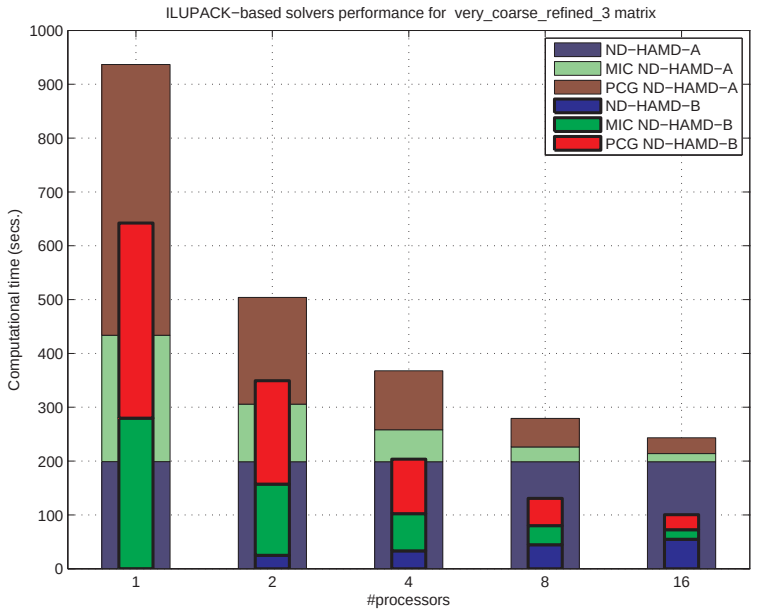
Id.	Code	Initial Mesh	# refs.	n	nnz_A	nnz_A/n
1	VC	very coarse	0	1,709	16,669	9.75
2	C	coarse	0	9,583	112,563	11.75
3	M	moderate	0	32,429	412,251	12.71
4	F	fine	0	101,296	1,368,594	13.51
5	VC2	very coarse	2	271,272	3,686,268	13.59
6	M1	moderate	1	297,927	4,134,255	13.88
7	VF	very fine	0	658,609	9,294,721	14.11
8	F1	fine	1	882,824	12,562,880	14.23
9	C2	coarse	2	906,882	12,854,824	14.17
10	VC3	very coarse	3	2,382,864	34,128,924	14.32
11	M2	moderate	2	2,539,954	36,768,808	14.48
12	VF1	very fine	1	5,413,520	78,935,174	14.58

ILUPACK-based solvers execution time



VC3 (3 refinements)

n=2,382,864



VF1 (1 refinement)

n=5,413,520

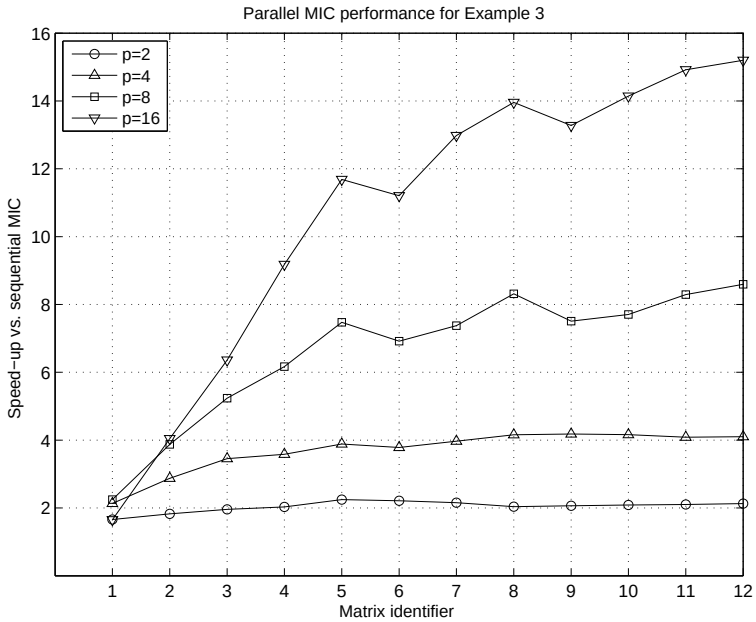
n	p	MLND Option ND-HAMD-X	MLND T (sec.)	MIC nnz_L $\times 10^{-6}$	MIC T_p (sec.)	MIC S_p (T_1/T_p)	PCG #Iter.	PCG T_p (sec.)	PCG S_p (T_1/T_p)
F	1	A	2.3	1.6	2.3	1.0	22	1.8	1.0
	8	A	2.3	1.6	0.4	6.0	24	0.3	6.4
	16	A	2.3	1.6	0.3	8.7	24	0.2	8.9
	1	B	0.0	1.5	2.6	1.0	23	1.8	1.0
	8	B	0.7	1.5	0.4	6.2	23	0.3	6.9
	16	B	0.9	1.5	0.3	9.4	23	0.2	9.3
M1	1	A	8.0	5.0	8.0	1.0	29	8.3	1.0
	8	A	8.0	5.1	1.3	6.2	29	1.2	7.2
	16	A	8.0	5.1	0.8	10.1	30	0.8	10.3
	1	B	0.0	4.7	9.8	1.0	28	7.7	1.0
	8	B	2.3	4.8	1.4	7.2	28	1.2	6.5
	16	B	2.7	4.8	0.9	11.1	29	0.8	9.9
VC3	1	A	82.1	44.9	83.9	1.0	52	136.8	1.0
	8	A	82.1	44.9	11.2	7.5	52	17.1	7.9
	16	A	82.1	44.9	6.5	13.0	53	10.0	13.7
	1	B	0.0	45.2	108.8	1.0	50	132.6	1.0
	8	B	21.5	45.0	13.3	8.2	52	17.1	7.7
	16	B	25.1	44.9	7.5	14.6	52	9.8	13.5
VF1	1	A	198.9	105.3	234.8	1.0	64	502.8	1.0
	8	A	198.9	104.2	27.0	8.7	63	51.7	9.7
	16	A	198.9	104.0	15.2	15.4	64	29.2	17.2
	1	B	0.0	100.7	279.7	1.0	60	362.3	1.0
	8	B	50.6	98.7	32.0	8.7	62	49.3	7.4
	16	B	60.9	99.1	17.4	16.1	62	27.7	13.3

number of nonzero entries of IC factor

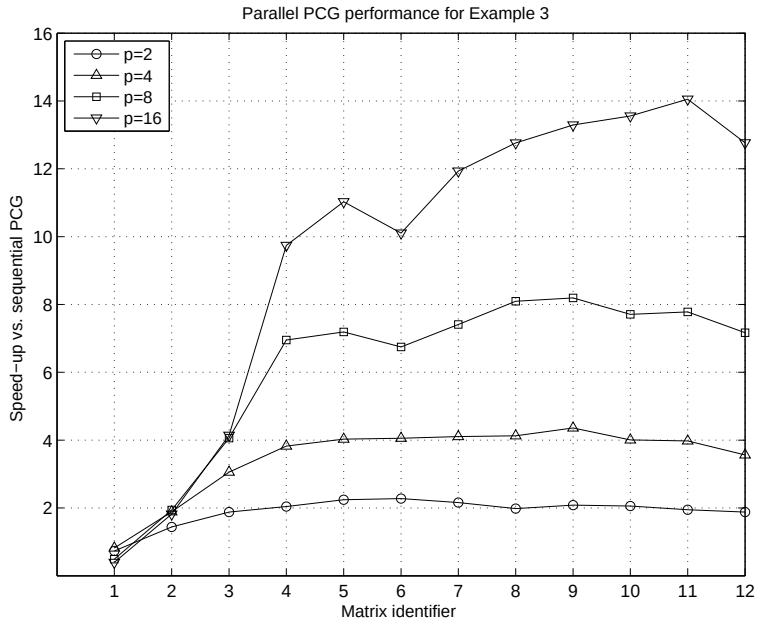
number of PCG iterations

- The nnz slightly changes and # of iterations only increase slightly with p
  $\rightarrow$ **the parallelization approach preserves the semantics of the MIC**

MIC and PCG stages parallel efficiency

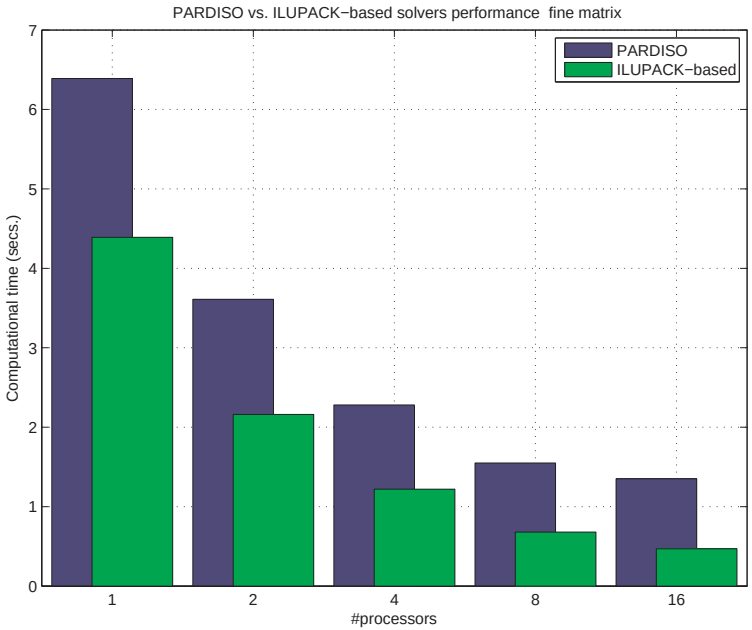


MIC Speed-Up

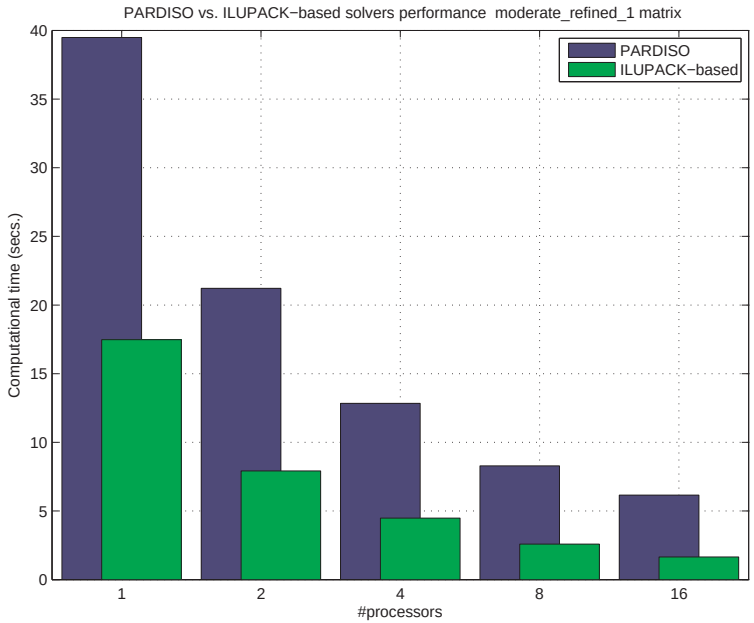


PCG Speed-Up

PARDISO vs. ILUPACK-based solvers

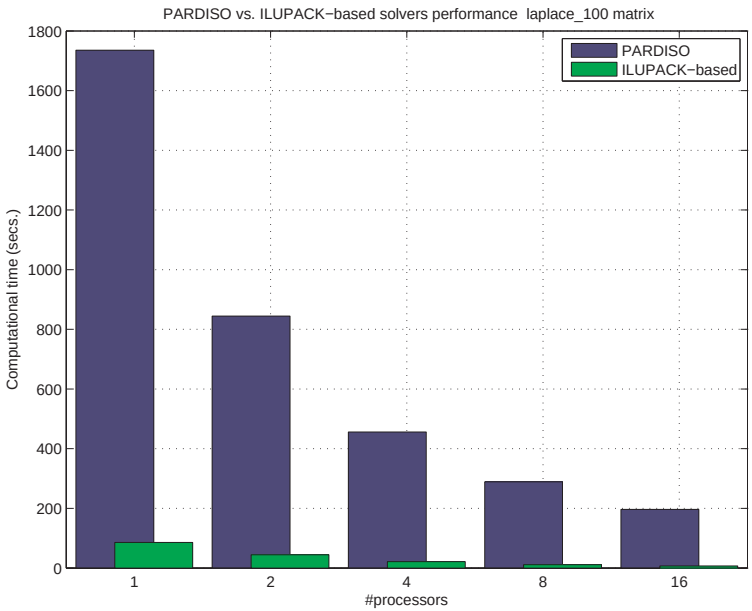


F mesh
n=101,296



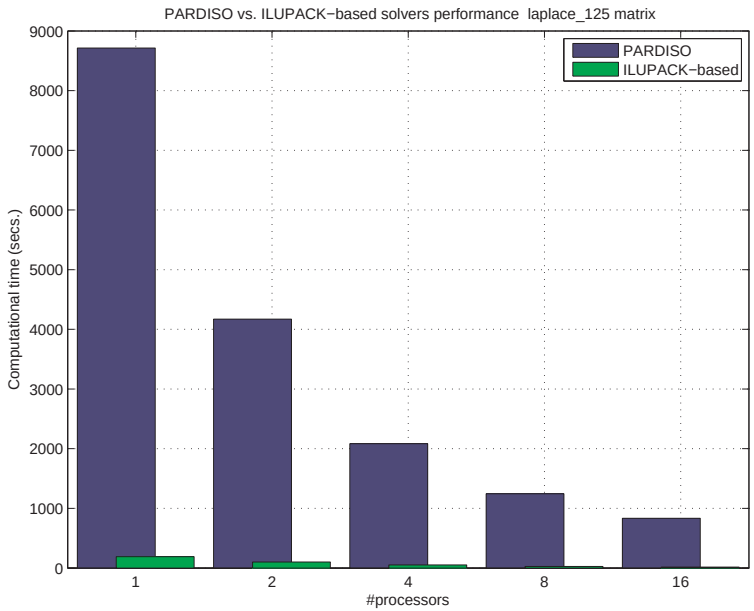
M1 (1 refinement)
n=297,927

PARDISO vs. ILUPACK-based solvers



$100 \times 100 \times 100$ Grid

n=1,000,000



$125 \times 125 \times 125$ Grid

n=1,953,125

Conclusions and Future work

The OpenMP ILUPACK parallelization . . .

- Preserves the semantics of the method in ILUPACK
- Delivers a high-degree of concurrence for shared-memory multiprocessors for the MIC and PCG stages
- The partitioning stage starts dominating the overall computation time as p increases

Ongoing work . . .

- Use parallel solutions for partitioning sparse matrices (PT-SCOTCH, ParMETIS)
- Extend the parallelization approach for the indefinite case
- Move to platforms with higher number of processors and larger scale problems to analyze the challenges to be faced and the limits of the parallelization approach

Questions?

Algorithm: computes parallel MIC decomp. of the reordered system $A \rightarrow \Pi^T A \Pi$

```
[ Π , T ] ← nested_dissection(G_A)        ▷ obtain task tree T and permutation Π
Q ← { leaves(T) }                         ▷ initialize Q with all leaves of T
mark all tasks of T as not executed
Begin parallel region
    pid ← get_process_identifier()
    repeat
        while pending tasks in Q do
            tid ← dequeue(Q)              ▷ remove ready task from the head of Q
            map [ tid ] ← pid             ▷ process pid in charge of task tid
            execute(tid)                  ▷ construct tid's submatrix and compute local MIC
            mark tid as executed
            if all dependencies of parent(tid) have been resolved then
                enqueue(parent(tid), Q)   ▷ insert new ready task at the tail of Q
            end
        end
    until not all tasks executed
End parallel region
```