Parallel analysis and scaling

A. Buttari and B. Uçar, CNRS-IRIT Toulouse,
CNRS-LIP Lyon

MUMPS Users Group Meeting, April 2010

Sparse direct solvers: the three phases

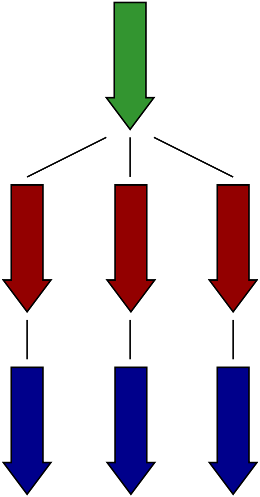
The solution of a sparse system with the MUMPS solver is achieved in three phases:

1. The **Analysis** phase
   - Fill-reducing pivot order
   - Symbolic factorization
   - Scaling
   - Amalgamantion
   - Mapping
   - ...
2. The **Factorization** phase
   - $LU = PA$
3. The **Solve** phase
   - Forward/backward substitutions

Sparse direct solvers: the three phases

The solution of a sparse system with the MUMPS solver is achieved in three phases:

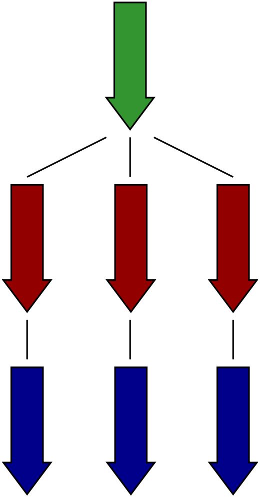
1. The **Analysis** phase
   - Fill-reducing pivot order
   - Symbolic factorization
   - Scaling
   - Amalgamantion
   - Mapping
   - ...
2. The **Factorization** phase
   - $LU = PA$
3. The **Solve** phase
   - Forward/backward substitutions



Sparse direct solvers: the three phases

The solution of a sparse system with the MUMPS solver is achieved in three phases:

1. The **Analysis** phase
   - Fill-reducing pivot order
   - Symbolic factorization
   - Scaling



Towards a Parallel Analysis

An approach to parallelization of the analysis

Briefly:

Problem: the sequential analysis of very large scale problems can be expensive

- memory consumption
- time to completion

Solution: parallelization of the analysis

An approach to parallelization of the analysis

Briefly:

Problem: the sequential analysis of very large scale problems can be expensive

- memory consumption
- time to completion

Solution: parallelization of the analysis

1. Parallel ordering of the problem using an external tool such as PT-SCOTCH or ParMETIS on $struct(A + A^T)$

An approach to parallelization of the analysis

Briefly:

Problem: the sequential analysis of very large scale problems can be expensive

- memory consumption
- time to completion

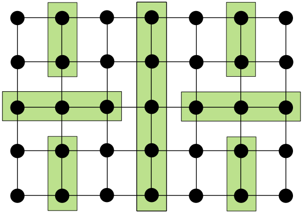
Solution: parallelization of the analysis

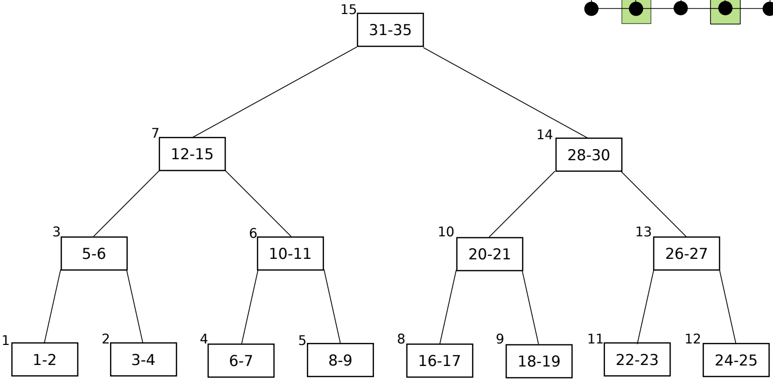
1. Parallel ordering of the problem using an external tool such as PT-SCOTCH or ParMETIS on $\textit{struct}(A + A^T)$
2. Parallel symbolic factorization based on quotient graphs and restarting techniques

Pahsel: parallel ordering

ParMETIS

- nested dissection stops at NP subdomains
- works only on 2^k processors
- quality of ordering degrades NP
- fast

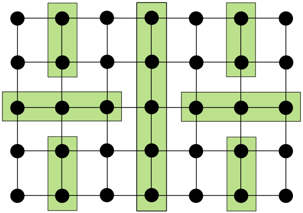


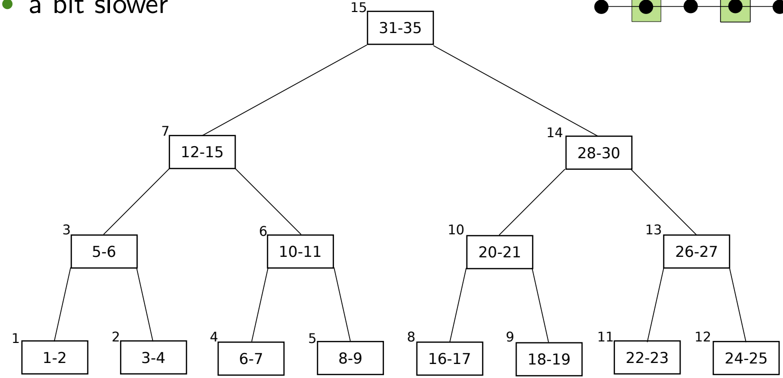


Pahsel: parallel ordering

PT-SCOTCH

- nested dissection does not stop at NP subdomains
- works on any number of processors
- quality of ordering is independent from NP
- a bit slower

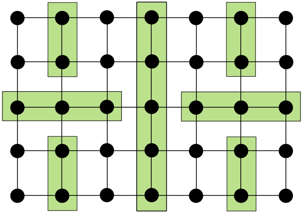


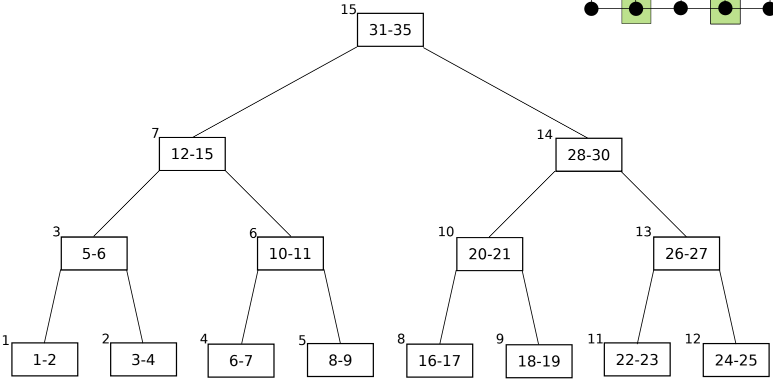


Pahsel: parallel ordering

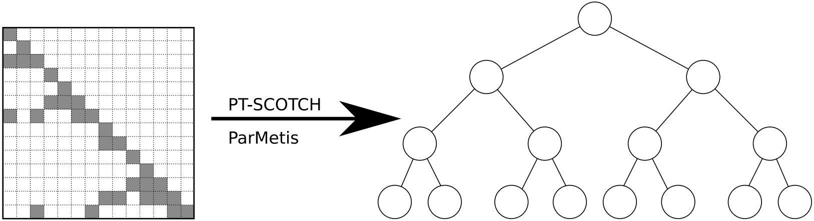
Both

- get a distributed graph in input
- return an ordering and a separators tree on output



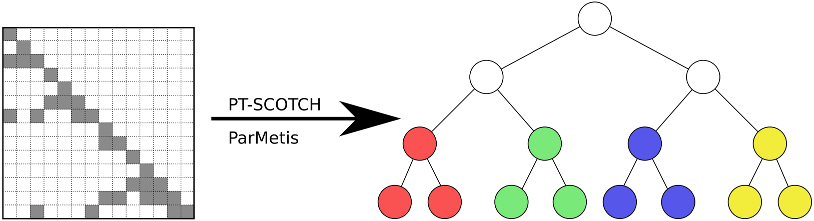


Parallel ordering and symbolic facto



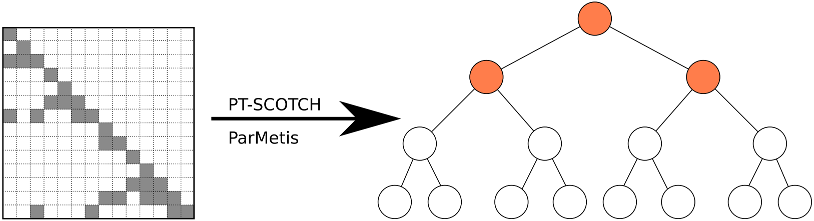
1 – First pass adjacency graph of the matrix to a parallel ordering tool (PT-SCOTCH or ParMetis). As a result, a pivotal order and a binary separators tree are returned

Parallel ordering and symbolic facto



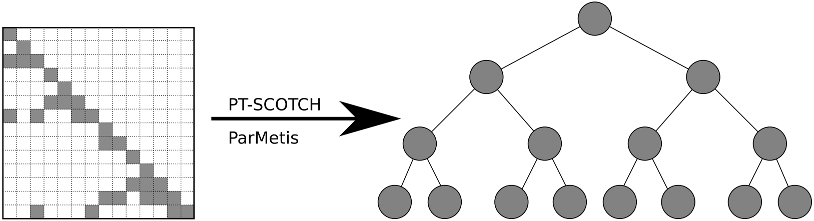
2 – Then each processor separately performs the symbolic elimination of the variables contained in a subtree. This symbolic factorization is based on the usage of quotient graphs with a restarting technique that mixes left and right looking factorization methods

Parallel ordering and symbolic facto



3 – The host processor eliminates the variables in the top part of the tree using the same technique

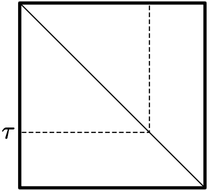
Parallel ordering and symbolic facto



4 – The distributed data are merged into a centralized data structure that is used in subsequent steps of the analysis phase like *amalgamation*, *mapping* etc.

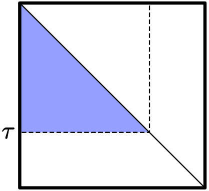
Parallel ordering and symbolic facto

- Quotient graphs
  - keep the cost limited to $O(nnz)$ thanks to techniques like nodes absorption and redundant edges elimination
  - ease the coupling between bottom and top part since the result of the symbolic facto on the subdomains can be represented as a clique in the quotient graph of the top-tree
- Restarting



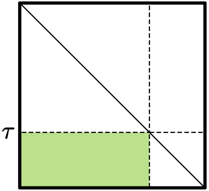
Parallel ordering and symbolic facto

- Quotient graphs
  - keep the cost limited to $O(nnz)$ thanks to techniques like nodes absorption and redundant edges elimination
  - ease the coupling between bottom and top part since the result of the symbolic facto on the subdomains can be represented as a clique in the quotient graph of the top-tree
- Restarting
  1. in pivotal steps $1, ..., \tau$ are processed and only the adjacency information for variables $1 - \tau$ is updated in a right-looking way



Parallel ordering and symbolic facto

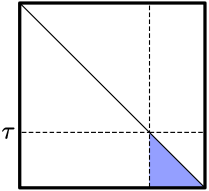
- Quotient graphs
  - keep the cost limited to $O(nnz)$ thanks to techniques like nodes absorption and redundant edges elimination
  - ease the coupling between bottom and top part since the result of the symbolic facto on the subdomains can be represented as a clique in the quotient graph of the top-tree
- Restarting
  1. in pivotal steps $1, ..., \tau$ are processed and only the adjacency information for variables $1 - \tau$ is updated in a right-looking way
  2. **restart**: the adjacency information of variables $\tau - n$ is updated with respect to elements $1 - \tau$ in a left-looking way



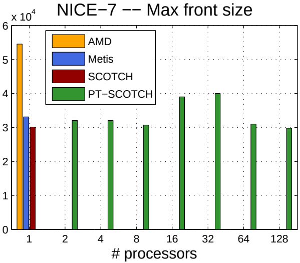
Parallel ordering and symbolic facto

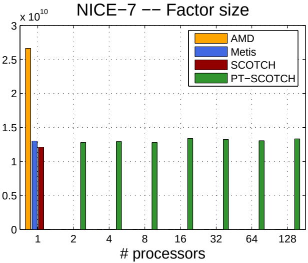
- Quotient graphs
  - keep the cost limited to $O(nnz)$ thanks to techniques like nodes absorption and redundant edges elimination
  - ease the coupling between bottom and top part since the result of the symbolic facto on the subdomains can be represented as a clique in the quotient graph of the top-tree

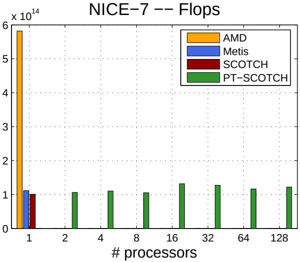
- Restarting
  1. in pivotal steps $1, ..., \tau$ are processed and only the adjacency information for variables $1 - \tau$ is updated in a right-looking way
  2. **restart**: the adjacency information of variables $\tau - n$ is updated with respect to elements $1 - \tau$ in a left-looking way
  3. apply steps 1 and 2 recursively on variables $\tau - n$

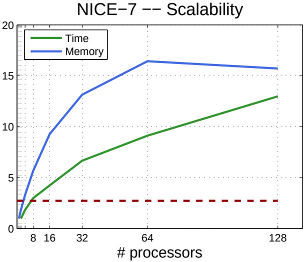


NICE-7: N=8159758, NNZ=669172552

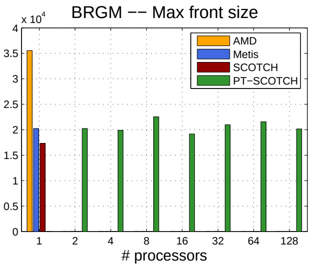

NICE−7 −− Max front size

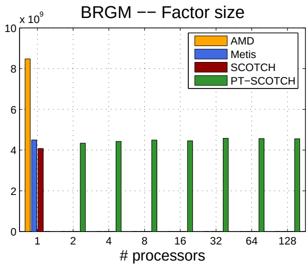

NICE−7 −− Factor size

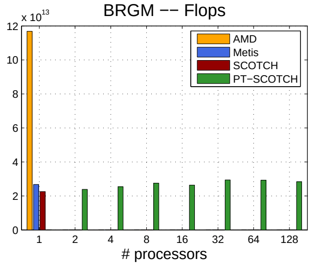

NICE−7 −− Flops

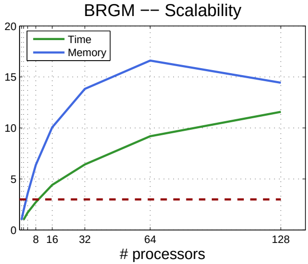

NICE−7 −− Scalability

BRGM: N=3699643, NNZ=307580395

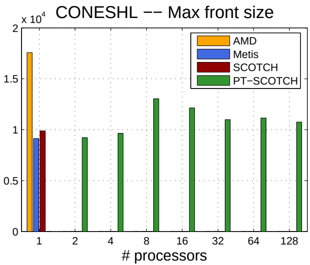

BRGM -- Max front size

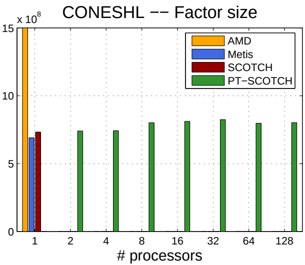

BRGM -- Factor size

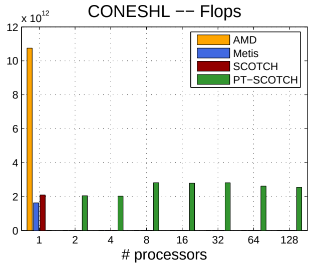

BRGM -- Flops

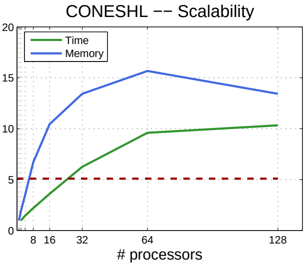

BRGM -- Scalability

CONESHL: N=1262212, NNZ=84753352

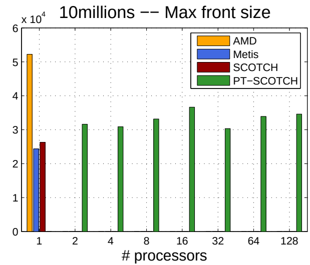

CONESHL -- Max front size

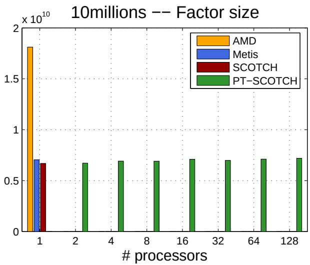

CONESHL -- Factor size

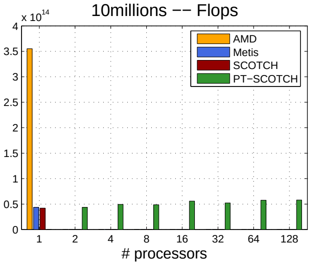

CONESHL -- Flops

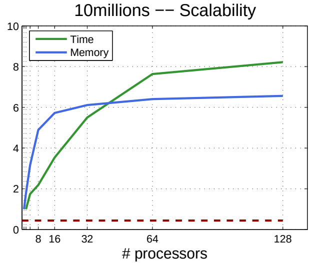

CONESHL -- Scalability

10millions: N=10423737, NNZ=167122005


10millions -- Max front size

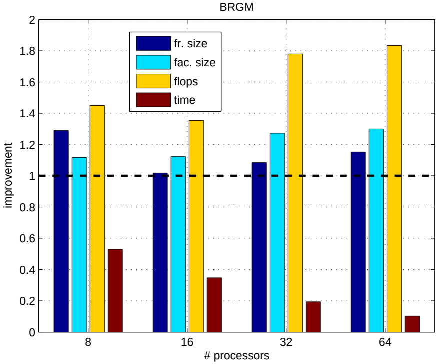

10millions -- Factor size


10millions -- Flops


10millions -- Scalability

BRGM: ParMETIS vs PT-SCOTCH

- Better ordering with PT-SCOTCH
- ParMETIS faster than PT-SCOTCH
- ParMETIS ordering degrades with increasing parallelism



Parallel analysis: interface

The behavior of the parallel analysis is defined by two parameters:

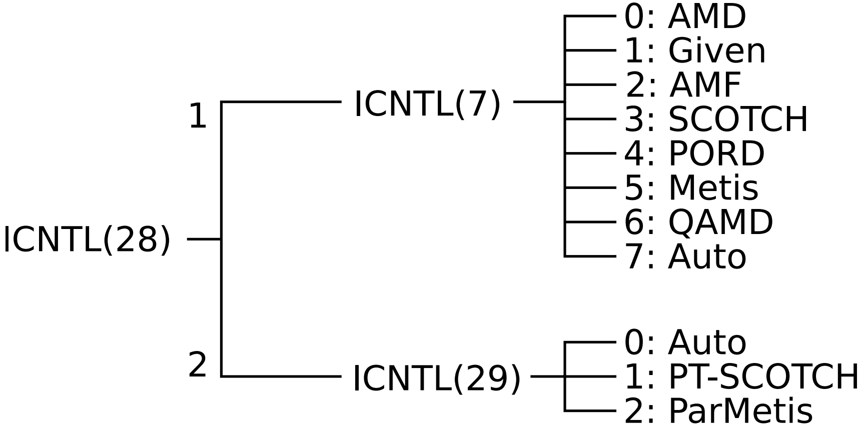
ICNTL(28) Analysis type:

- 0: Automatic decision (always =1 for the moment)
- 1: Sequential analysis
- 2: Parallel analysis

ICNTL(29) Ordering method for the parallel analysis

- 0: Automatic decision (always =1 for the moment)
- 1: PT-SCOTCH
- 2: ParMetis

Parallel analysis: interface



Parallel Scaling

Matrix scaling

Definition

Given an $m \times n$ sparse matrix $\mathbf{A}$, find diagonal matrices $\mathbf{D}_1 > 0$ and $\mathbf{D}_2 > 0$ such that all rows and columns of the scaled matrix

$$\hat{\mathbf{A}} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$$

have equal norm.

Motivations

- Good pivoting strategy, numerical/optimal properties.
- Scaling combined with permutations can avoid many numerical difficulties [Duff and Pralet, SIMAX(2005)] during LU factorization:
  - Provides (weak) diagonal dominance,
  - Increases robustness of the factorization algorithms,
  - May improve the condition number.

The sequential algorithm (Ruiz 2001)

```
$\mathbf{D_1}^{(0)} \leftarrow \mathbf{I}_{m\times m}$        $\mathbf{D_2}^{(0)} \leftarrow \mathbf{I}_{n\times n}$
for $k = 1, 2, \ldots$ until convergence do
    $\mathbf{D}_R \leftarrow \mathrm{diag}\left(\sqrt{\|\mathbf{r}_i{}^{(k)}\|_\ell}\right)\; i = 1, \ldots, m$
    $\mathbf{D}_C \leftarrow \mathrm{diag}\left(\sqrt{\|\mathbf{c}_j{}^{(k)}\|_\ell}\right)\; j = 1, \ldots, n$
    $\mathbf{D_1}^{(k+1)} \leftarrow \mathbf{D_1}^{(k)}\, \mathbf{D}_R{}^{-1}$
    $\mathbf{D_2}^{(k+1)} \leftarrow \mathbf{D_2}^{(k)}\, \mathbf{D}_C{}^{-1}$
    $\mathbf{A}^{(k+1)} \leftarrow \mathbf{D_1}^{(k+1)}\mathbf{A}\mathbf{D_2}^{(k+1)}$
end for
```

Reminder

$\|\mathbf{x}\|_\infty = \max\{|x_i|\}$

$\|\mathbf{x}\|_1 = \sum |x_i|$

Notes

ℓ: any vector norm (usually ∞- and 1-norms)

Convergence is achieved when

$$\max_{1\le i\le m}\left\{|1 - \|\mathbf{r}_i{}^{(k)}\|_\ell|\right\} \le \varepsilon \text{ and } \max_{1\le j\le n}\left\{|1 - \|\mathbf{c}_j{}^{(k)}\|_\ell|\right\} \le \varepsilon$$

Features

Some properties (Ruiz 2001)

- Preserves symmetry; permutation independent; amenable to parallelization,
- With ∞-norm, linear convergence with asymptotic rate of $1/2$,
- With 1-norm, results are similar to those of the other well-known algorithms; convergence under certain conditions.

Practical considerations

- Numerical tests toward investigating the effects on LU decomposition, preconditioning [Duff and Pralet, SIMAX(2005)],
- Sequential codes also available in HSL library as MC77 [Ruiz (2001)],
- Parallel codes have been plugged into MUMPS.

Parallelization: Data distribution

Data: $\hat{\mathbf{A}}^{(k)}$, $\mathbf{A}$, $\mathbf{D_1}^{(k)}$ $\mathbf{D_2}^{(k)}$, $\mathbf{D_R}$ and $\mathbf{D_C}$.

The scaled matrix $\hat{\mathbf{A}}^{(k)}$

Do not store $\hat{\mathbf{A}}^{(k)} = \mathbf{D_1}^{(k)}\mathbf{A}\mathbf{D_2}^{(k)}$ explicitly; access $a_{ij}{}^{(k)}$ by

$$d_1{}^{(k)}(i) \times |a_{ij}| \times d_2{}^{(k)}(j)$$

- Distribute $\mathbf{A}, \mathbf{D_1}$, and $\mathbf{D_2}$. At every iteration $\mathbf{D_R}$ and $\mathbf{D_C}$ are computed afresh.
  - Matrix $\mathbf{A}$ is already distributed (in another context). Each processor holds a set of entries a_{ij} and their indices (i, j).
  - Partition the diagonal elements of $\mathbf{D_1}$ and $\mathbf{D_2}$ among processors.

Problem definition

Given a partition on $\mathbf{A}$, find the best partitions for $\mathbf{D_1}$ and $\mathbf{D_2}$.

Parallelization: Dependencies

Local computations

Each processor p should use each (i, j, a_{ij}) triplet to compute partial results on $d_R(i)$ and $d_C(j)$, e.g., in ∞-norm, sets

$$d_R{}^p(i) = \max\left\{ d_1{}^{(k)}(i) \times |a_{ij}| \times d_2{}^{(k)}(j) : a_{ij} \in p \right\}$$

Communication operations

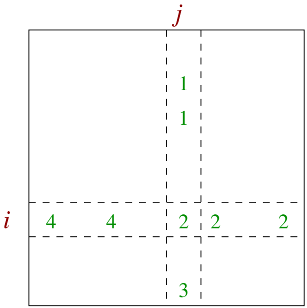
The partial results should be combined/reduced for each $d_1{}^{(k+1)}(i)$. The owner of $d_1(i)$ should set, in ∞-norm,

$$d_1{}^{(k+1)}(i) = d_1{}^{(k)}(i) \times \frac{1}{\sqrt{\max\{d_R{}^p(i) : 1 \le p \le P\}}}.$$

The owner should send $d_1{}^{(k+1)}(i)$ back to the contributing processors.

- Similar discussion for $d_2(j)$.

Parallelization: ∞-norm algorithm for step k



Row $\mathbf{r}_i$

Processors 2 and 4 contribute to $d_1{}^{(k+1)}(i)$. Whichever owns $d_1(i)$, receives one unit of data and sends one unit of data after computing the final $d_1{}^{(k+1)}(i)$.

Column $\mathbf{c}_j$

Processors 1, 2, and 3 contribute to $d_2{}^{(k+1)}(j)$. Whichever owns $d_2(j)$, receives two units of data and sends two units of data after computing the final $d_2{}^{(k+1)}(j)$.

Parallelization: Communication requirements

Common communication cost metric: the total volume.

Communication for $\mathbf{D}_1$

- The volume of data a processor receives while reducing a ${d_1}^{(k+1)}(i)$ is equal to the volume of data it sends after computing ${d_1}^{(k+1)}(i)$.
- Nonzeros in row $\mathbf{r}_i$ are split among $s_r(i)$ processors
  - All contribute to ${d_1}^{(k+1)}(i)$.
  - Reduction on $s_r(i)$ partial results.
  - If one of those $s_r(i)$ processors owns $d_1(i)$, $s_r(i) - 1$ partial results will be send to the owner.
  - If owned by somebody else, then $s_r(i)$ partial results will be send to the owner.

Communication for $\mathbf{D}_2$

Similar observations.

Parallelization: Partitioning $\mathbf{D}_1$ and $\mathbf{D}_2$

Communication requirements

Nonzeros in row $\mathbf{r}_i$ are split among $s_r(i)$ processors: total volume of communication is equal to

$$2 \times \sum (s_r(i) - 1)$$

(half for receiving contributions, half for sending back the results).

- The total volume of communication is the same for any $d_1(i)$ to processor assignment as long as that processor has at least one nonzero from row $\mathbf{r}_i$.

Similar observation for the column $\mathbf{c}_j$.

Twice the requirements of parallel sparse matrix-vector multiply operation.

Computations and communication requirements

Computations (per iteration)

Op.	SpMxV	1-norm	∞-norm
add	$\mathrm{nnz}(\mathbf{A})$	$2 \times \mathrm{nnz}(\mathbf{A})$	0
mult	$\mathrm{nnz}(\mathbf{A})$	$2 \times \mathrm{nnz}(\mathbf{A}) + m + n$	$2 \times \mathrm{nnz}(\mathbf{A}) + m + n$
comparison	0	0	$2 \times \mathrm{nnz}(\mathbf{A})$

Communication (per iteration)

The communication operations both in the 1-norm and ∞-norm algorithms are the same as those in the computations

$$\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$$
$$\mathbf{x} \leftarrow \mathbf{A}^T\mathbf{y}$$

when the partitions on $\mathbf{x}$ and $\mathbf{y}$ are equal to those on $\mathbf{D}_2$ and $\mathbf{D}_1$.

Parallelization: Our partitioning approach

What we did?

- Use simple strategies while ensuring that each scaling entry is assigned to a processor that contributes to that entry.

$d_1(i)$ assigned to the processor p that has an entry a_{ij} with j giving $\min\{|i-j|\}$; in case of ties to the processor with the smallest rank.

$d_2(j)$ assigned to the processor p that has an entry a_{ij} with i giving $\min\{|i-j|\}$; in case of ties to the processor with the smallest rank.

What could be done?

The freedom can be used to optimize some other metrics [U. and Aykanat, SISC(2004); Bisseling and Meesen, ETNA(2005)].

- Communication cost: Minimize number of messages, maximum volume/message per processor.
- Balance the number of $d_1(i)$ and/or $d_2(j)$ per processor.

Parallelization results: Speedup values

matrix	Seq. Time (s.)	Number of processors 2	4	8	16
aug3dcqp	8.30	1.7	2.9	4.1	4.5
	3.06	1.9	3.8	4.3	3.6
a2nnsnsl	20.71	1.8	3.1	4.0	4.8
	7.24	1.5	1.8	2.1	3.3
a0nsdsil	20.92	1.8	3.1	4.0	4.6
	7.22	1.5	1.8	2.1	3.2
lhr71	78.25	2.0	3.8	7.3	13.5
	18.10	2.0	3.4	6.8	14.0
G3_circuit	455.25	1.8	3.8	7.4	14.0
	173.11	1.9	3.3	6.9	14.5
thermal2	573.24	2.0	3.9	7.6	14.4
	208.20	1.6	3.4	6.5	13.1

- Averages of 10 different partitions (with PaToH [Çatalyürek and Aykanat, Tech.Rep (1999)]),
- PC cluster with a Gigabit Ethernet switch (Intel Pentium IV 2.6 GHz), PC cluster with an Infiniband interconnect (dual 150 Opteron AMD processors)

Best three and worst three speedup values are shown—speedup tends to be higher with larger number of nonzeros.

Settings in MUMPS and some numerical results

Default Behavior (ICNTL(8)=7)

One ∞-norm scaling, three 1-norm scaling.

Behaved better than a number of alternatives (among about 700 matrices, without scaling for 46 matrices parameter ICNTL(14) had to be adjusted; with this setting for 15 matrices).

Typically reduces the number of delayed pivots and off-diagonal pivoting, and hence reduces the memory requirements

	Flops ($\times 10^6$)		#entries in factors ($\times 10^6$)			
			estimated		effective	
scaling	OFF	ON	OFF	ON	OFF	ON
C-54	281	209	1.42	1.42	1.76	1.58
a0nsdsil	7.7	2.5	0.42	0.42	0.57	0.42

Conclusions

Thank you.

Parallel analysis and scaling are good $\ddot\smile$