Sparse Communication Avoiding Pivoting

Jennifer Scott
Joint work with Jonathan Hogg

STFC Rutherford Appleton Laboratory

CSC Workshop Lyon
July 2014

Sparse direct solvers

Want to solve $N \times N$ system:

$$Ax = b$$

where A is large, sparse and, for this talk,
symmetric and indefinite

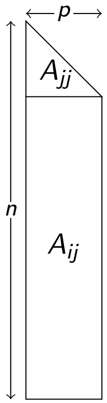
using a factorization

$$A = LDL^T$$

where L is unit lower triangular and D block diagonal with
1×1 and 2×2 bocks.

LDL^T factorization

Most solvers work with supernodes that are held as dense matrices of form:



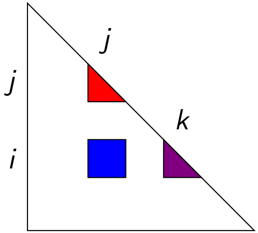
(1)

where $N \gg n \gg p$.

LDL^T factorization

Work by blocks:

- Factorize dense blocks on diagonal using dense algorithm $A_{jj} = L_{jj} D_{jj} L_{jj}^T$
- "Divide" remainder of column by diagonal block $L_{ij} = A_{ij} L_{jj}^{-T}$
- Update remaining matrix to right as $A_{ik} = A_{ik} - L_{ij} D_{jj} {L_{kj}}^T$



Threshold Partial Pivoting (TPP)

Higham (1997) proves sufficient condition for stability of LDL^T factorization is

$$\|L\|_\infty \|D\|_\infty \|L^T\|_\infty \leq const_n \|A\|_\infty$$

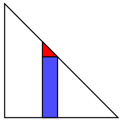
for a modest constant $const_n$, provided linear systems involving 2×2 pivots are solved in a norm-wise backward stable fashion.

So we want to bound the entries of L.

Threshold Partial Pivoting (TPP)

1×1 pivot test

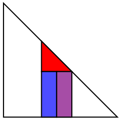
$$|a(q,q)| \geq u \max_{i>q} |a(i,q)|$$



Threshold Partial Pivoting (TPP)

1×1 pivot test

$$|a(q,q)| \ge u \max_{i>q} |a(i,q)|$$



2×2 pivot test

$$\left| \begin{pmatrix} a(q,q) & a(q,q+1) \\ a(q,q+1) & a(q+1,q+1) \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{i>q+1} |a(i,q)| \\ \max_{i>q+1} |a(i,q+1)| \end{pmatrix} \le \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}$$

where $0 < u \le 0.5$ controls balance between stability and sparsity.

Threshold Partial Pivoting (TPP)

Problem: comparisons with all entries in candidate column(s). Thus

- The whole column must be up-to-date (limits parallelism)
- Communication needed between diagonal block and off-diagonal blocks

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: “Normalize” entries of A

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: “Normalize” entries of A
- Ordering: Permute large entries onto subdiagonal and then pick 2×2 pivots

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: "Normalize" entries of A
- Ordering: Permute large entries onto subdiagonal and then pick 2×2 pivots
- Restricted pivoting: Limit checking to diagonal block and perturb pivots that are too small (no delays)

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: “Normalize” entries of A
- Ordering: Permute large entries onto subdiagonal and then pick 2×2 pivots
- Restricted pivoting: Limit checking to diagonal block and perturb pivots that are too small (no delays)
- Use factorization as preconditioner e.g. iterative refinement

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: “Normalize” entries of A
- Ordering: Permute large entries onto subdiagonal and then pick 2×2 pivots
- Restricted pivoting: Limit checking to diagonal block and perturb pivots that are too small (no delays)
- Use factorization as preconditioner e.g. iterative refinement

Our experience:
Combinations of these works for $\sim$95% of real matrices.

Alternatives?

Various *a priori* treatments to reduce/eliminate need for pivoting:

- Scaling: “Normalize” entries of A
- Ordering: Permute large entries onto subdiagonal and then pick 2×2 pivots
- Restricted pivoting: Limit checking to diagonal block and perturb pivots that are too small (no delays)
- Use factorization as preconditioner e.g. iterative refinement

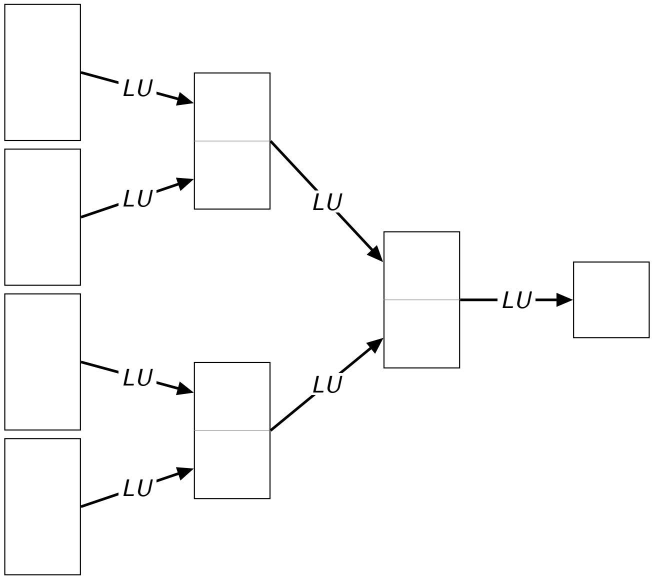
Our experience:
Combinations of these works for ∼95% of real matrices.

For remaining numerically challenging 5% we need pivoting!

What's done in the dense case?

- Key difference: in the dense case, pivots may be chosen from off-diagonal block as well as from diagonal block.
- CALU algorithm of Grigori, Demmel and Xiang (2011) uses tournament pivoting.
- Supernode is recursively bisected into sections upon which an LU factorization is performed to select the best p pivot rows.

Tournament pivoting

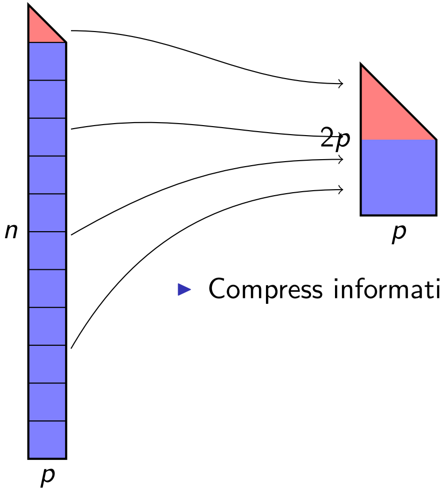


Tournament pivoting

- Tournament pivoting performs an optimal amount of communication that is asymptotically less than Gaussian elimination with partial pivoting.
- Hence, it is faster on platforms where communication is expensive.
- Furthermore, it has been shown to be stable in practice.

But as pivots are selected from within off-diagonal blocks, approach cannot be accommodated within a traditional sparse symmetric factorization without significant additional fill and departure from pre-planned data structures.

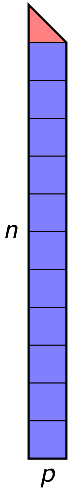
Compressed pivoting

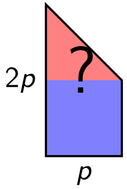


Idea:

- Compress information into small matrix

Compressed pivoting

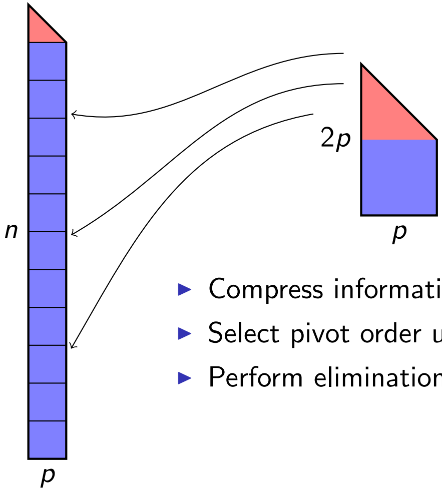




Idea:

- Compress information into small matrix
- Select pivot order using small matrix

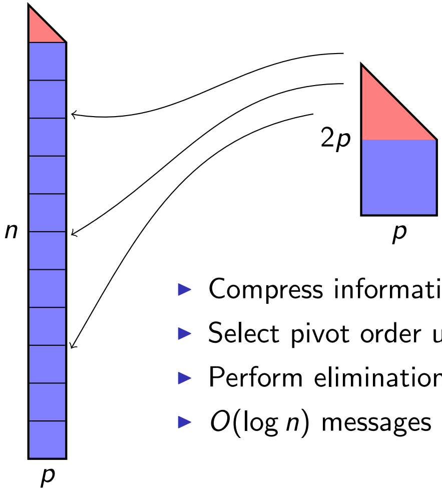
Compressed pivoting



Idea:

- Compress information into small matrix
- Select pivot order using small matrix
- Perform eliminations on supernodal matrix

Compressed pivoting

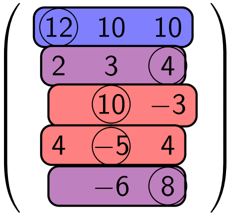


Idea:

- Compress information into small matrix
- Select pivot order using small matrix
- Perform eliminations on supernodal matrix
- $O(\log n)$ messages rather than $O(p \log n)$

Strict Compressed Pivoting

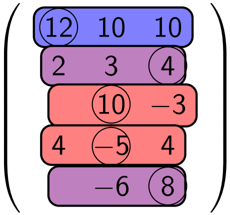
1. Partition the rows of the supernodal matrix into sets according to the column of maximum index of largest entry in the row.



Partitioned rows

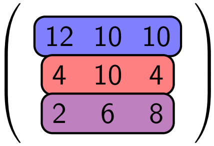
Strict Compressed Pivoting

1. Partition the rows of the supernodal matrix into sets according to the column of maximum index of largest entry in the row.
2. Represent each set by single row: take maximum $|a_{ij}|$



$$\begin{pmatrix} \textcircled{12} & 10 & 10 \\ 2 & 3 & \textcircled{4} \\ & \textcircled{10} & -3 \\ 4 & \textcircled{-5} & 4 \\ & -6 & \textcircled{8} \end{pmatrix}$$

Partitioned rows

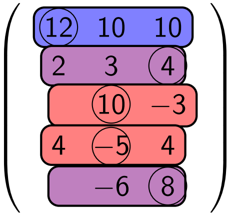


$$\begin{pmatrix} 12 & 10 & 10 \\ 4 & 10 & 4 \\ 2 & 6 & 8 \end{pmatrix}$$

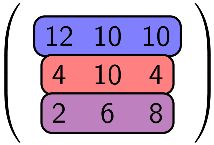
Compressed matrix

Strict Compressed Pivoting

1. Partition the rows of the supernodal matrix into sets according to the column of maximum index of largest entry in the row.
2. Represent each set by single row: take maximum $|a_{ij}|$
3. Update using a “worst-case” formula



Partitioned rows



Compressed matrix

Strict Compressed Pivoting

This approach is

- Provably backwards stable
- Sometimes too pessimistic (rejects pivots TPP would have accepted leading to more delayed pivots and hence denser factors and more work/memory)

Relaxed Compressed Pivoting

1. For each column, pick a “representative” row: largest $|a_{ij}|$
2. Apply standard threshold partial pivoting.

$$\begin{pmatrix} \textcircled{12} & 10 & 10 \\ 2 & 3 & 4 \\ & \textcircled{10} & -3 \\ 4 & -5 & 4 \\ & -6 & \textcircled{8} \end{pmatrix}$$

Partitioned rows

Relaxed Compressed Pivoting

1. For each column, pick a “representative” row: largest $|a_{ij}|$
2. Apply standard threshold partial pivoting.

$$\begin{pmatrix} \textcircled{12} & 10 & 10 \\ 2 & 3 & 4 \\ & \textcircled{10} & -3 \\ 4 & -5 & 4 \\ & -6 & \textcircled{8} \end{pmatrix}$$

Partitioned rows

$$\begin{pmatrix} 12 & 10 & 10 \\ & 10 & -3 \\ & -6 & 8 \end{pmatrix}$$

Compressed matrix

Relaxed Compressed Pivoting

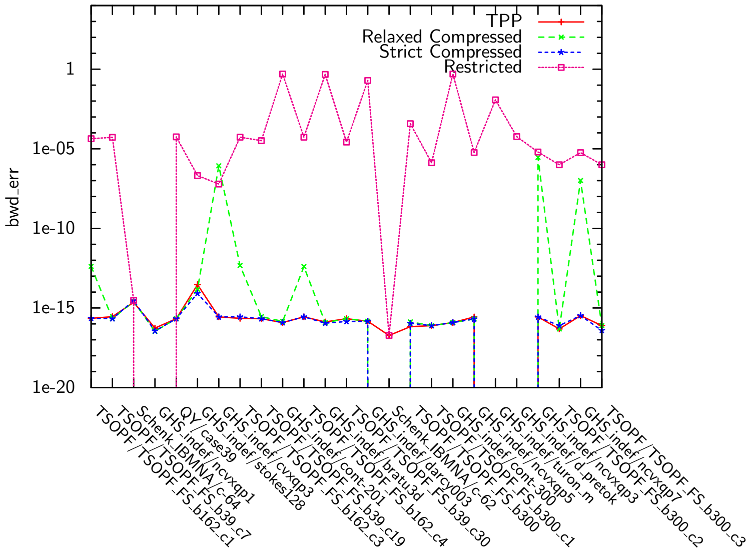
This approach is

- Not backwards stable!
- But we find it is stable in practice (see results)

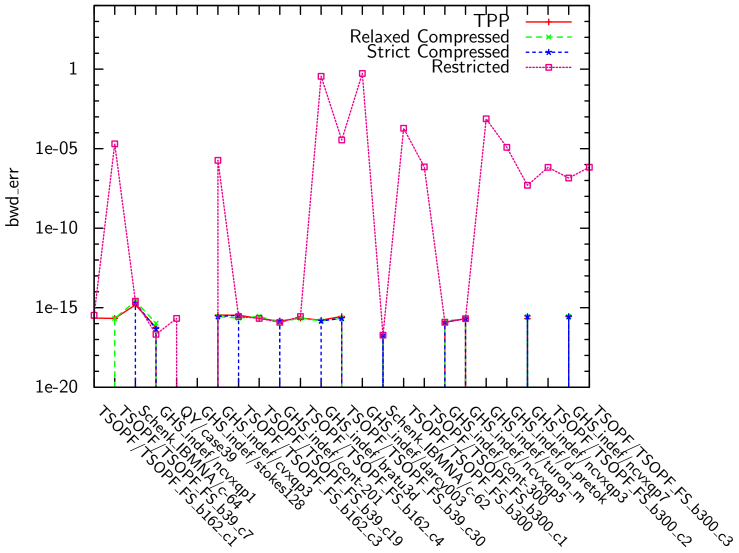
Numerical experiments

- 25 tough indefinite problems from UFL selected
- Problems scaled using symmetric version of `MC64`
- Sparse solver `HSL_MA97` run on 2×8 core machine
- 10 steps of iterative refinement used

Results: numerical stability default ordering



Results: numerical stability matching-based ordering



Results: numerical stability

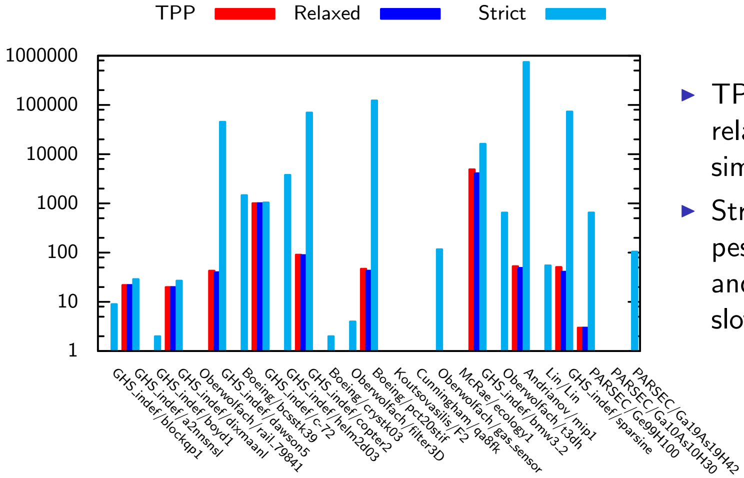
Experiments show:

- TPP and strict compressed pivoting always good
- Relaxed compressed pivoting is better than restricted pivoting
- Matching-based ordering can really help ... but restricted pivoting still unstable for some problems.

Experiment on delayed pivots

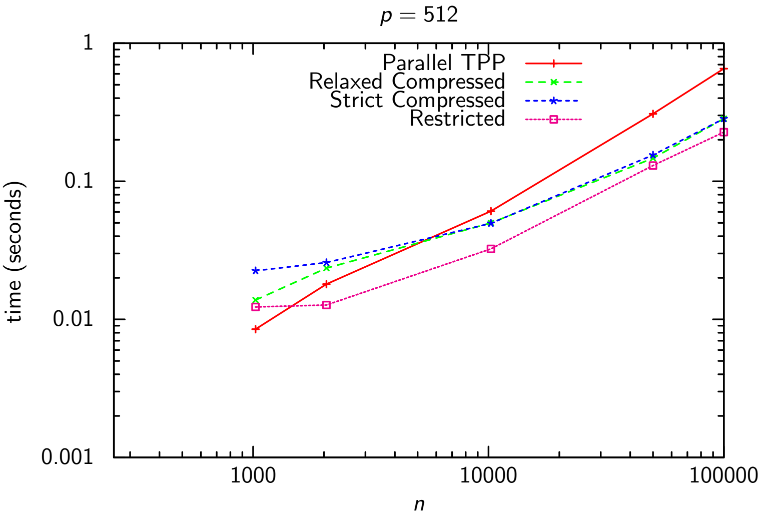
- Want to check out whether the proposed compressed pivoting leads to an increase in the number of delayed pivots.
- We consider a set of 25 general problems from UFL.
- These are not such tough problems ... TPP does not result in a large number of delayed pivots.

Results: delays

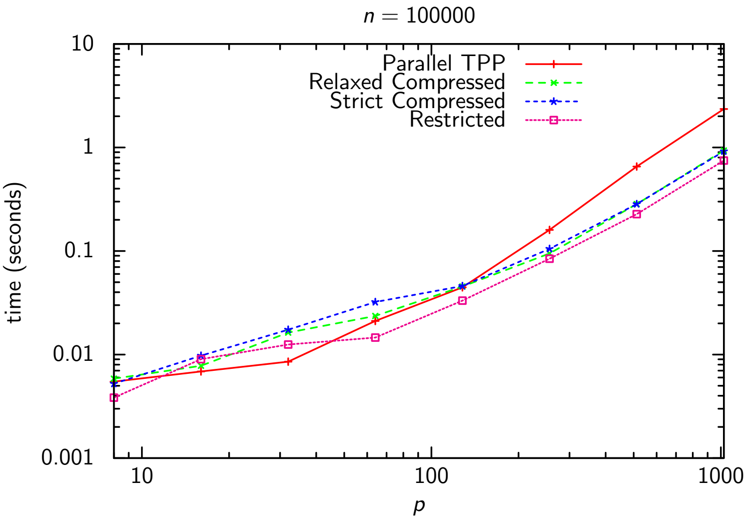


- TPP and relaxed very similar
- Strict very pessimistic and hence slow

Results: speed $p = 512$



Results: speed $n = 100000$



Summary of findings

- Compressed pivoting 2+ times faster on large problems
- Restricted pivoting not good enough for tough problems
- Strict compressed pivoting guarantees backwards stability
- Relaxed compressed pivoting works well and cheaper in practice

Another approach

Try-it-and-see pivoting:

- Store a copy of the supernodal matrix before pivoting
- Factorize diagonal block without reference to off-diagonal blocks
- Do numerical test on entries of L_{ij} *a posteriori*
- Back-track if test fails
- **Still need a fall back plan ... pivots may be delayed**

Our new GPU code (Hogg, Ovtchinnikov and Scott)

Some key aspects of SSIDS:

- Multifrontal code, primarily for indefinite systems. All frontal matrices (including small ones) factored on GPU.
- Incorporates ‘try-it-and-see’ pivoting.
- Both factorization and subsequent solves performed on GPU.
- Bit-compatible results.

Implementation was challenging! Report available with details.

Times(s) and Speedup: Factor+Solve

Problem	CPU	GPU	Speedup
GHS_indef/c-71	2.76	0.64	4.3
GHS_indef/ncvxqp3	4.75	1.61	2.9
Schenk_IBMNA/c-big	11.8	2.35	5.0
DNVS/shipsec1	1.51	0.61	2.5
GHS_psdef/bmwcra_1	2.09	0.93	2.3
DNVS/ship_003	3.12	1.08	2.9
ND/nd6k	6.42	1.36	4.7
PARSEC/Si5H12	14.6	2.20	6.7
Andrianov/mip1	0.82	0.38	2.2

CPU: Westmere-EP

- $2\times$ E5620 $=$ 8 cores [77 GFlop/s, 160W TDP]

GPU: Fermi

- C2050 GPU [515 GFlop/s, 238W TDP]

GPU to CPU flops ratio about $7\times$

Times(s) and Speedup: Factor+Solve

Problem	CPU	GPU	Speedup
GHS_indef/c-71	1.67	0.43	3.9
GHS_indef/ncvxqp3	2.31	1.42	1.6
Schenk_IBMNA/c-big	6.63	1.49	4.4
DNVS/shipsec1	0.63	0.40	1.6
GHS_psdef/bmwcra_1	0.77	0.60	1.3
DNVS/ship_003	1.24	0.68	1.8
ND/nd6k	2.84	0.82	3.5
PARSEC/Si5H12	7.20	1.32	5.4
Andrianov/mip1	0.40	0.28	1.5

GPU to CPU flops ratio about 3×

[TDP = total board power]

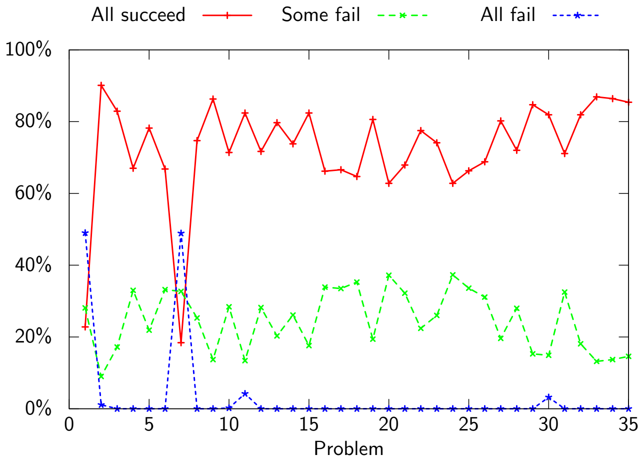
CPU: Sandybridge-EP

- 2×E5-2687W = 16 cores [397 GFlop/s, 300W TDP]

GPU: Kepler

- K20 GPU [1170 GFlop/s, 225W TDP]

Percentage of candidate pivot columns accepting a given number of pivots (candidate pivot columns can have at most 8 pivots).



Science & Technology
Facilities Council

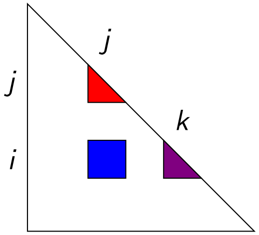
Thank you!

Paper on compressed pivoting to appear in SIMAX.

GPU code SSIDS is open source and is available from
`http://www.numerical.rl.ac.uk/spral`

Supported by EPSRC grants EP/I013067/1 and EP/J010553/1

Stability



- What if diagonal block is **singular**?
- What if off-diagonal entries **much larger** than diagonal entries?

Then factorization is **not** backwards stable