



Challenges in Parallel Sparse Direct Linear Solvers

Jonathan Hogg

STFC Rutherford Appleton Laboratory

Perspectives on Parallel Numerical Linear Algebra
18th July 2012

Sparse Direct Solvers

Solve

$$Ax = b$$

where A is **Sparse**.

Direct Methods Factorize $A = LU$, solve $Ly = b$, $Ux = y$.

Black-box, robust, **compute-bound**.

Memory-hungry \Rightarrow slow for large matrices?.

Iterative Methods CG, GMRES, BiCGStab, etc.

Matrix-free. Fast? Efficient? **memory-bound**.

Non-robust, performance depends on preconditioner.



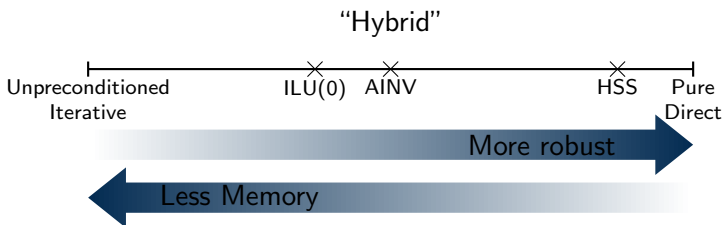
Sparse Direct Solvers

Solve

$$Ax = b$$

where A is **Sparse**.

New view: Spectrum



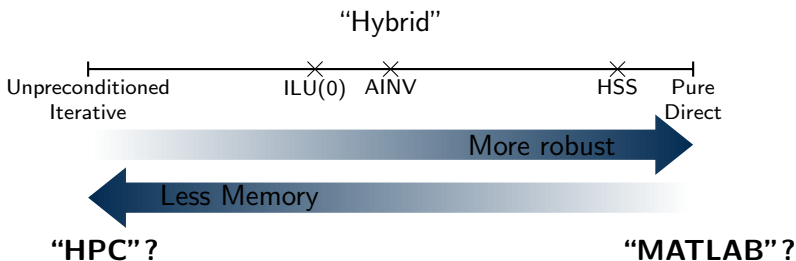
Sparse Direct Solvers

Solve

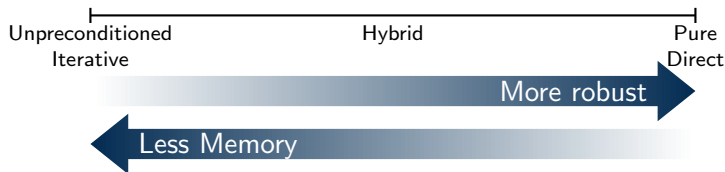
$$Ax = b$$

where A is **Sparse**.

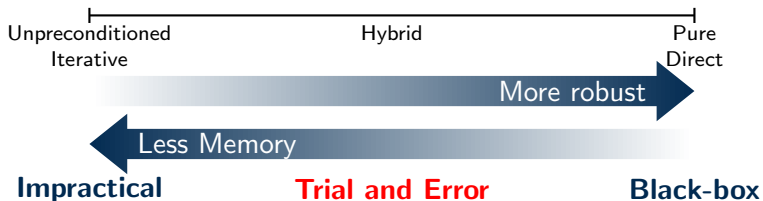
New view: Spectrum



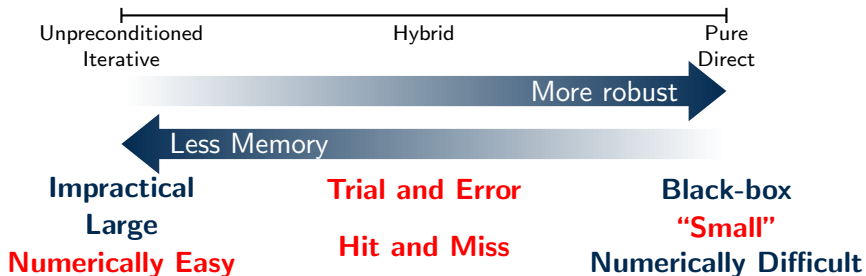
Horses for Courses



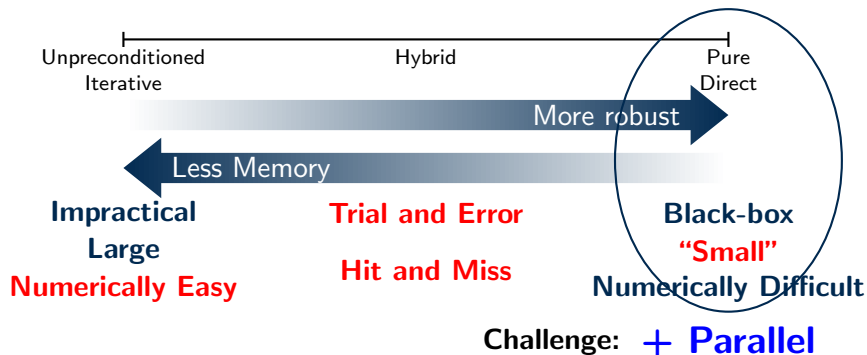
Horses for Courses



Horses for Courses



Horses for Courses



Challenge #1: “Small” + Parallel

We need to achieve **strong scaling**.

Example

Non-linear optimization solver, unknown problem origin

⇒ **Preconditioning difficult** (at best)!

Direct solver: solves 100 systems ($n = 35000$) to reach solution in 5 seconds. 95% of time in linear solver.

⇒ **0.05s per serial factorization**

Maybe 2 million flops with 250,000 non-zeroes (8 flops/non-zero)

2015 desktop: 16 CPU cores + 1024 GPU cores?

⇒ **Fewer than 250 non-zeroes per core!**



Challenge #1: “Small” + Parallel

We need to achieve **strong scaling**.

8 flops/non-zero \Rightarrow Communication is King!

Work by *Laura Grigori*, *Jim Demmel* and others:

Communication avoiding algorithms

A small world:

Avoid fine-grained communication — **latency hurts**.



Challenge #1: “Small” + Parallel

We need to achieve **strong scaling**.

8 flops/non-zero \Rightarrow Communication is King!

Work by *Laura Grigori*, *Jim Demmel* and others:

Communication avoiding algorithms

A small world:

Avoid fine-grained communication — **latency hurts**.

Assume flops are (almost) free: what can we do?

- ▶ Generic compression [bandwidth]
- ▶ Low-rank approximation (HSS preconditioning) [bandwidth]
- ▶ Speculative assumptions on numerical stability [latency]

Generic Compression

J.D. Hogg and J.A. Scott

A note on the solve phase of a multicore solver

RAL-TR-2010-007

Idea:

Compress data blocks before storing factors, decompress into cache before use. Otherwise 1 flops/non-zero in solve phase.

[LZO Compression Library](#) Higher compression than GZIP, *much* faster.



Generic Compression

J.D. Hogg and J.A. Scott

A note on the solve phase of a multicore solver

RAL-TR-2010-007

Idea:

Compress data blocks before storing factors, decompress into cache before use. Otherwise 1 flops/non-zero in solve phase.

LZO Compression Library Higher compression than GZIP, *much* faster.

Outcome:

Performance matched that of original algorithm:

Wait for more flops/unit bandwidth.



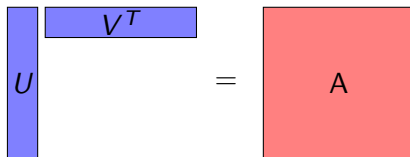
Low-rank approximation

Multiple works by *J. Xia, S. Chandrasekaran, M. Gu, X.S. Li* et al.

Idea:

Communicate low rank approximations not large dense matrices

Rank-revealing QR:



The diagram illustrates the Rank-revealing QR decomposition. On the left, a blue vertical rectangle labeled U is positioned next to a blue horizontal rectangle labeled V^T . To the right of these two rectangles is an equals sign. Further to the right is a red square labeled A . This visualizes the equation $U V^T = A$.

Flops are
cheap!

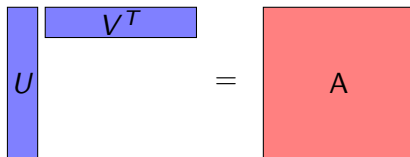
Low-rank approximation

Multiple works by *J. Xia, S. Chandrasekaran, M. Gu, X.S. Li* et al.

Idea:

Communicate low rank approximations not large dense matrices

Rank-revealing QR:


$$U V^T = A$$

Flops are
cheap!

Outcome:

Good *preconditioner* for some classes of matrix.

More work needed!



Speculative assumptions on numerical stability

PARDISO: *O. Schenk* et al.

Static pivoting, weighted matchings: *I.S. Duff* and others.

Idea:

Assume no pivoting is needed; don't do pivoting.



Speculative assumptions on numerical stability

PARDISO: *O. Schenk* et al.

Static pivoting, weighted matchings: *I.S. Duff* and others.

Idea:

Assume no pivoting is needed; don't do pivoting.

More Advanced Idea:

Put large entries on subdiagonal; only do local pivoting.



Speculative assumptions on numerical stability

PARDISO: *O. Schenk* et al.

Static pivoting, weighted matchings: *I.S. Duff* and others.

Idea:

Assume no pivoting is needed; don't do pivoting.

More Advanced Idea:

Put large entries on subdiagonal; only do local pivoting.

Outcome:

Works for majority of matrices.

But: Not for some *difficult* matrices — what direct solvers are for!

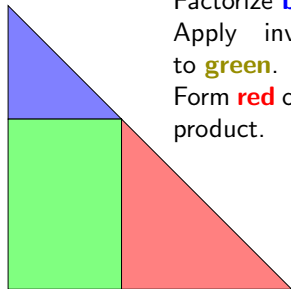
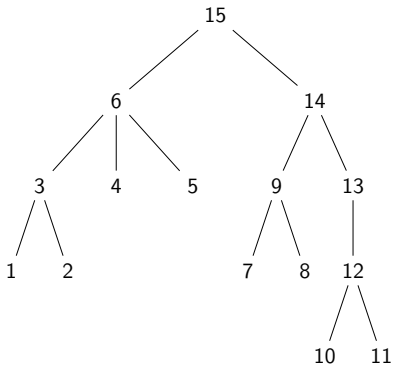


Challenge #2: Numerically difficult + Parallel

Need to do **pivoting** for stability — in parallel.

Sparse Direct Primer:

Organises into tree of dense linear algebra + sparse scatters

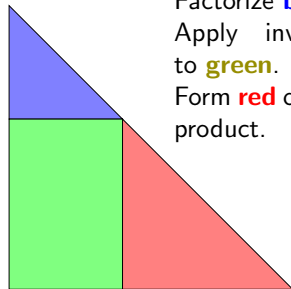
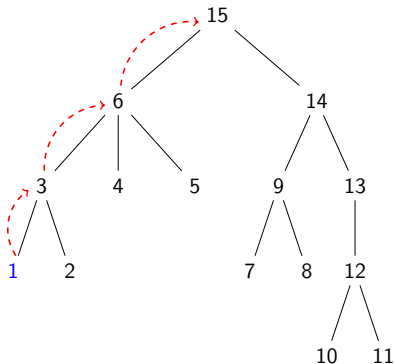


Challenge #2: Numerically difficult + Parallel

Need to do **pivoting** for stability — in parallel.

Sparse Direct Primer:

Organises into tree of dense linear algebra + sparse scatters



Factorize **blue**.
Apply inverse
to **green**.
Form **red** outer
product.

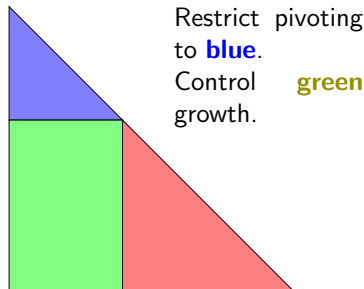
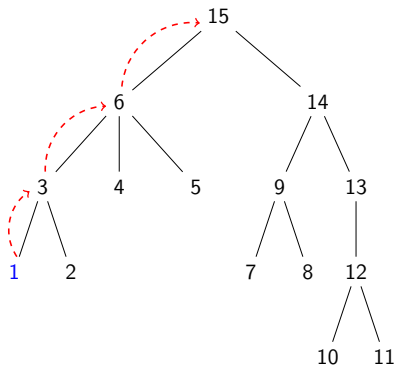


Challenge #2: Numerically difficult + Parallel

Need to do **pivoting** for stability — in parallel.

Sparse Direct Primer:

Organises into tree of dense linear algebra + sparse scatters



Challenge #2: Numerically difficult + Parallel

Need to do **pivoting** for stability — in parallel.

Observations:

- ▶ Want to start factorization of diagonal block *before* rest of column is ready.
- ▶ Even for difficult matrices, delayed pivots generally restricted to few subtrees.
- ▶ Assume pivoting will work; backtrack if it doesn't.
- ▶ Achieve the best of both worlds?



Challenge #2: Numerically difficult + Parallel

Need to do **pivoting** for stability — in parallel.

Otherwise:

- ▶ Currently test 1×1 and 2×2 pivots
- ▶ Use larger **block pivots**?
- ▶ Sparse analog to tournament pivoting?



Challenge #3: Bit-compatibility?

Bit-compatibility: Getting the same answer twice.

$$1 + (\epsilon/2 + \epsilon/2) \neq (1 + \epsilon/2) + \epsilon/2$$



Challenge #3: Bit-compatibility?

Bit-compatibility: Getting the same answer twice.

$$1 + (\epsilon/2 + \epsilon/2) \neq (1 + \epsilon/2) + \epsilon/2$$

Why would we not do this?

- ▶ If we don't, answers are still equally valid
- ▶ Less efficient: restrict parallelism, optimization
- ▶ More difficult to achieve
- ▶ Must be achieved by all libraries used



Challenge #3: Bit-compatibility?

Bit-compatibility: Getting the same answer twice.

$$1 + (\epsilon/2 + \epsilon/2) \neq (1 + \epsilon/2) + \epsilon/2$$

But it's very attractive

- ▶ Hard to debug without it: make it an option?
- ▶ Confuses non-expert users no end
- ▶ Methods built on top may behave unexpectedly:

e.g. Different local maxima found for non-linear optimization
Different iteration counts



Achieving bit-compatibility

Option #1: Add up in the same order

J.D. Hogg and J.A. Scott, HSL_MA97

Enforce ordering on additions:

$$((1 + 2) + 3) + 4 \quad \text{or} \quad (1 + 2) + (3 + 4).$$

Option #2: Add up in high precision

Use quad or double-double precision to store intermediate results

Ideally requires sufficient cache to hold intermediate results.



Task-based

Sparse task-based implementation *exist*: HSL_MA86, HSL_MA87, PaStiX.

Problems:

- ▶ Block alignments — need dynamic reblocking for best efficiency.
- ▶ Building on top of LAPACK/PLASMA — dynamic reblocking on same data desirable.
- ▶ Building on top of LAPACK/PLASMA — can we use the same task scheduler?
- ▶ Dynamic task sizing — splitting/merging across levels.
- ▶ Bit-compatibility?

Supernodal method



Supernodal method



Supernodal method



Supernodal method



Supernodal method



Tasking

- ▶ Each task may have its own way of blocking.
- ▶ Run in parallel — different optimal block sizes.
- ▶ Want to compose libraries.



Summary

“Direct” Methods Still required:

- ▶ Black-box solution
- ▶ Small problems
- ▶ Numerically difficult problems

Challenges:

1. Small + Parallel (strong scaling)
2. Accurate + Parallel (communication avoiding pivoting)
3. Bit-compatibility (software/user education)
4. Interface to rest of software stack (up *and* down)



**But iterative methods
aren't perfect either...**



Iterative methods challenges

If Matrix-vector product is main cost:

- ▶ Already Memory-bound
- ▶ Look for ways to use spare cycles \Rightarrow More expensive preconditioning?
- ▶ 2 or 4 M-v product not much more expensive than 1 M-v.
Can you exploit this?



Iterative methods challenges

If Matrix-vector product is main cost:

- ▶ Already Memory-bound
- ▶ Look for ways to use spare cycles \Rightarrow More expensive preconditioning?
- ▶ 2 or 4 M-v product not much more expensive than 1 M-v.
Can you exploit this?

Existing Efforts:

- ▶ *Mark Hoemmen* (Berkeley),
Communication Avoiding Krylov Methods
- ▶ Computes $[v, Av, A^2v, \dots, A^s v]$ simultaneously
- ▶ Uses QR for orthogonalize
- ▶ Need to use Chebyshev basis for stability





Thank you!