The challenge of the solve phase of a multicore solver

Jonathan Hogg
Jennifer Scott

Rutherford Appleton Laboratory

IMA Numerical Linear Algebra and Optimization, Birmingham, 13 September 2010

Sparse Direct Solvers

Solve $A\mathbf{x} = \mathbf{b}$ using a Cholesky factorization:

$$PAP^T = LL^T$$

Where A is...

- Large
- Sparse
- Positive Definite

Variants for more general matrices — LDL^T, LU.

Sparse Direct Solvers

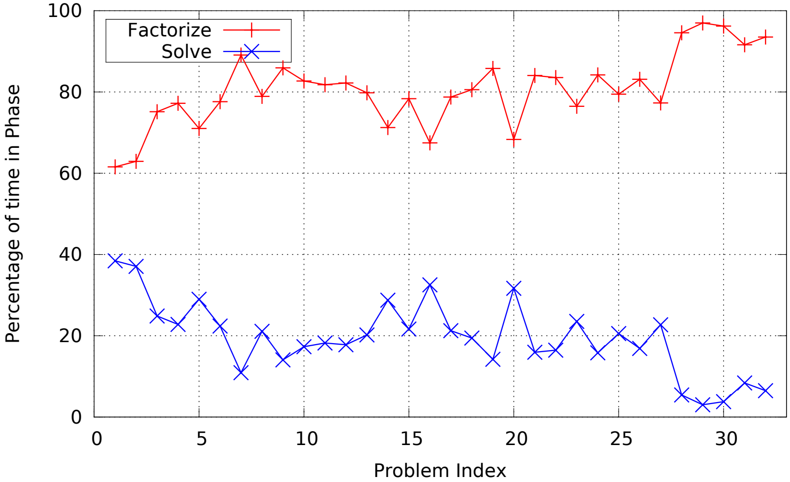
Four phases:

Ordering	Reduce fill-in (AMD, MeTiS, SCOTCH, ...)
Analyse	Construct data structures, plan computation
Factorize	Numerical computation
Solve	Triangular solves with L and L^T

Traditionally: Factorize phase by far the most time consuming:

$$1 \text{ Factorize} = 50\text{-}100 \text{ Solves}$$

Multicore. Oh dear.



Factorize parallel. Solve serial.

Science & Technology Facilities Council

So? Just parallelize the solve!

Lots of ways of doing this:

- Threaded BLAS/OpenMP do loops

So? Just parallelize the solve!

Lots of ways of doing this:

- Threaded BLAS/OpenMP do loops
- Assembly tree parallelism

So? Just parallelize the solve!

Lots of ways of doing this:

- Threaded BLAS/OpenMP do loops
- Assembly tree parallelism
- DAG-based methods

So? Just parallelize the solve!

Lots of ways of doing this:

- Threaded BLAS/OpenMP do loops
- Assembly tree parallelism
- DAG-based methods

So? Just parallelize the solve!

Lots of ways of doing this:

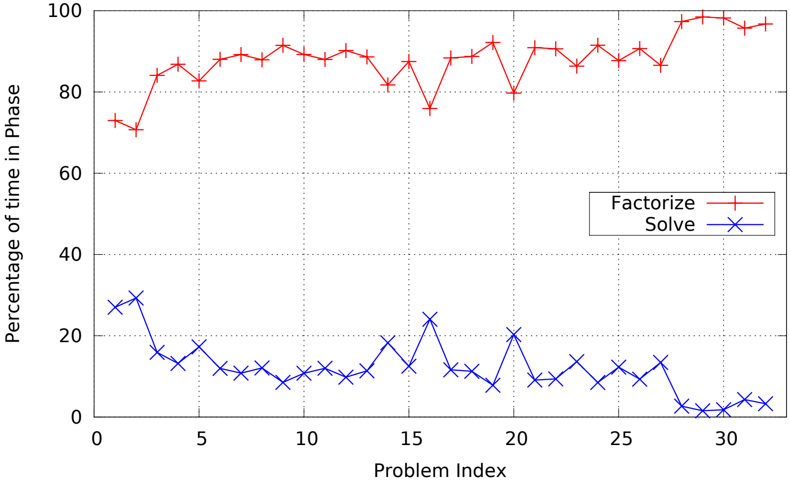
- Threaded BLAS/OpenMP do loops
- Assembly tree parallelism
- DAG-based methods

Regardless of how we did this...

...only a 2–3 times speedup

Science & Technology Facilities Council

Not much better...



Factorize and solve parallel.

Why?

Typically:

	Flops	Cache Misses
Factor	$> 98\%$	60–80%
Solve	$< 2\%$	20–40%

Speedup on a single quad-core: < 1.30
Speedup on two quad-cores: < 3.00

Solve is memory bound.

Multicore $=$ more cores, same memory bandwidth.

Is this a problem?

Is this a problem?

Yes.

Lots of use cases for multiple sequential solves following a factorize.

Is this a problem?

Yes.

Lots of use cases for multiple sequential solves following a factorize.

Iterative Refinement classical case. Use multiple solves to reduce the error.

Is this a problem?

Yes.

Lots of use cases for multiple sequential solves following a factorize.

Iterative Refinement classical case. Use multiple solves to reduce the error.

Mixed Precision precondition high precision iterative methods with low precision factorization.

Is this a problem?

Yes.

Lots of use cases for multiple sequential solves following a factorize.

Iterative Refinement classical case. Use multiple solves to reduce the error.

Mixed Precision precondition high precision iterative methods with low precision factorization.

Preconditioning in general often uses incomplete factorizations.

Is this a problem?

Yes.

Lots of use cases for multiple sequential solves following a factorize.

Iterative Refinement classical case. Use multiple solves to reduce the error.

Mixed Precision precondition high precision iterative methods with low precision factorization.

Preconditioning in general often uses incomplete factorizations.

Interior-point Methods multiple correctors and often aggressive iterative refinement.

Can we fix it?

Need to exploit the cache.

or

Need to trade off more computation for less memory bandwidth.

Can we fix it?

Need to exploit the cache.

or

Need to trade off more computation for less memory bandwidth.

Combined factor-solve?

Can we fix it?

Need to exploit the cache.

or

Need to trade off more computation for less memory bandwidth.

Combined factor-solve?

Compression?

Combined factor-solve

Forward substitution can be done at same time as factorization.
(data is already in cache!)

Well established in out-of-core solvers.

Halves time for first solve

But... doesn't help on subsequent solves.

Compression

Two *theoretical* limits:

Compression ratio How much bandwidth do we save?

Decompression rate How many extra cycles does this take?

Will never go faster by more than the compression ratio.

Compression

Two *theoretical* limits:

Compression ratio How much bandwidth do we save?

Decompression rate How many extra cycles does this take?

Will never go faster by more than the compression ratio.

Exploit two levels of compression:

Algorithm specific We have deliberately introduced extra zeros to make the factorize go faster, does removing them help?

Generic Use a generic compression algorithm such as gzip, bzip2, LZO...

Compression ratios

	nemin = 32	LZO nemin = 32		nemin = 1		LZO nemin = 1	
	size	size	**CR**	size	**CR**	size	**CR**
1.	579	333	**1.74**	283	**2.04**	269	**2.14**
2.	981	567	**1.73**	489	**2.01**	463	**2.12**
3.	315	283	**1.11**	292	**1.08**	269	**1.17**
4.	407	328	**1.24**	342	**1.19**	303	**1.34**
5.	10829	10310	**1.05**	10304	**1.05**	10009	**1.08**

(size is storage for L in Megabytes)

1. CEMW/tmt_sym
2. Schmid/thermal2
3. GHS_psdef/crankseg_1
4. DNVS/shipsec1
5. GHS_psdef/audikw_1

Science & Technology Facilities Council

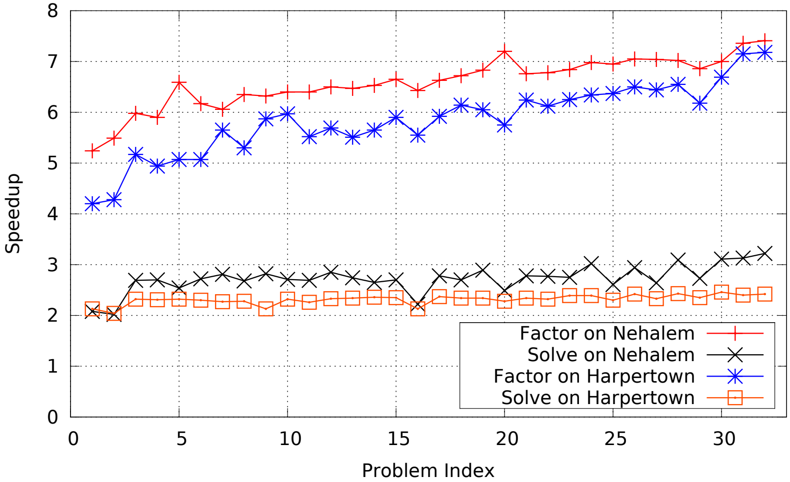
In practice...

Not even getting close to theoretical maxima.

	4 cores			8 cores			**CR**
	without	with	**ratio**	without	with	**ratio**	
1.	0.44	0.38	**1.16**	0.30	0.27	**1.11**	**1.74**
2.	0.76	0.67	**1.13**	0.53	0.48	**1.10**	**1.73**
3.	0.19	0.19	**1.00**	0.11	0.11	**1.00**	**1.11**
4.	0.25	0.24	**1.04**	0.15	0.14	**1.07**	**1.24**
5.	6.90	6.92	**1.00**	3.61	3.59	**1.01**	**1.05**

(times in seconds)

Are architectures improving?



Conclusions

- Multicore is not SMP
- Memory-bound operations are a problem
- Solve phase of direct methods now significant
- No easy fix

Any questions?
Any solutions?

Time distribution, all phases.