The Challenge of Strong Scaling for Direct Methods

Jonathan Hogg and the NLAFET team

STFC Rutherford Appleton Laboratory

14 April 2016
SIAM Parallel Processing 2016
Paris, France

Introduction

Solve: $Ax = b$

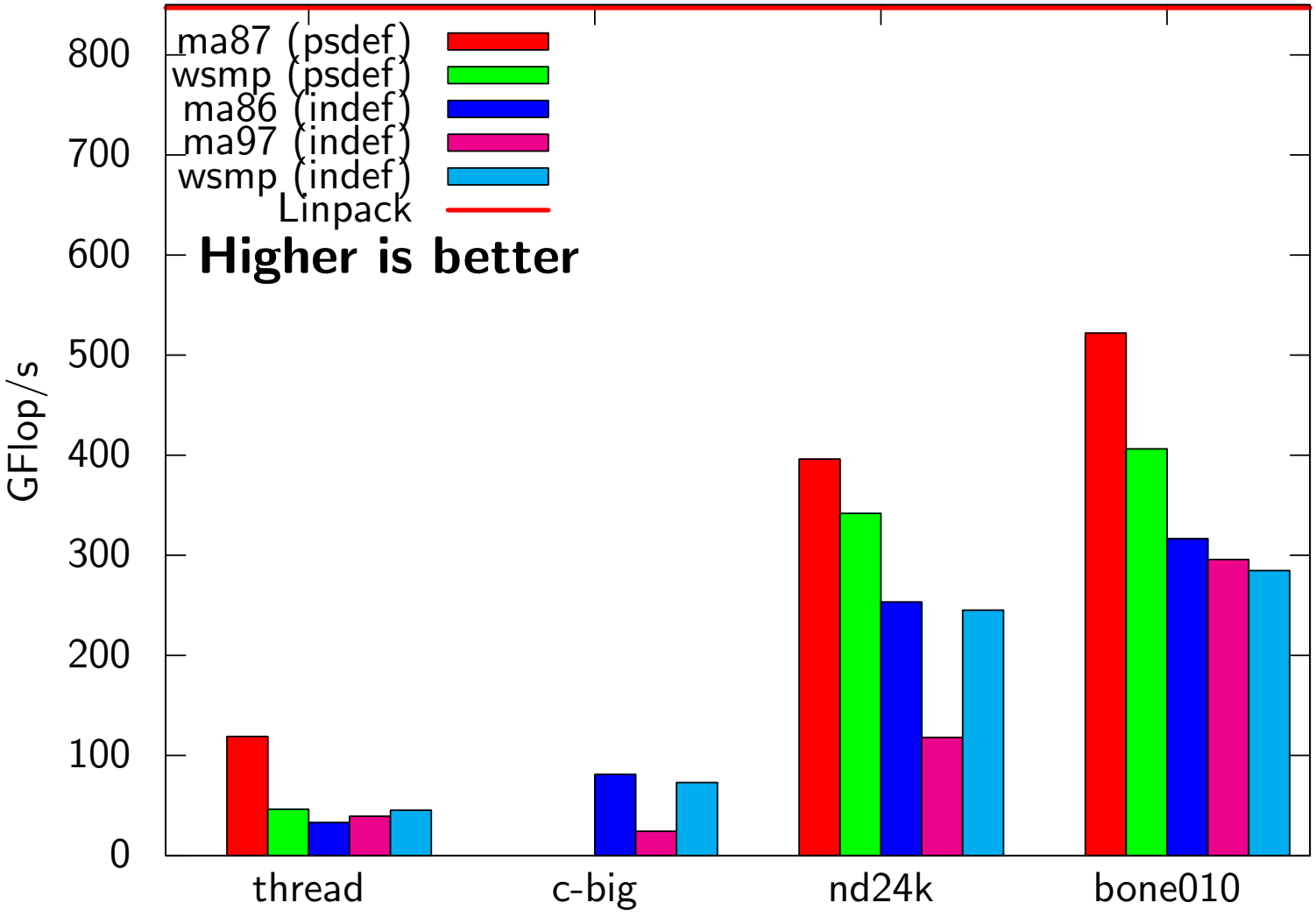
A is:

- Sparse
- Large
- Symmetric?
- Indefinite?

Algorithm should be:

- Fast
- Accurate
- Numerically Stable
- Bitwise-reproducible?

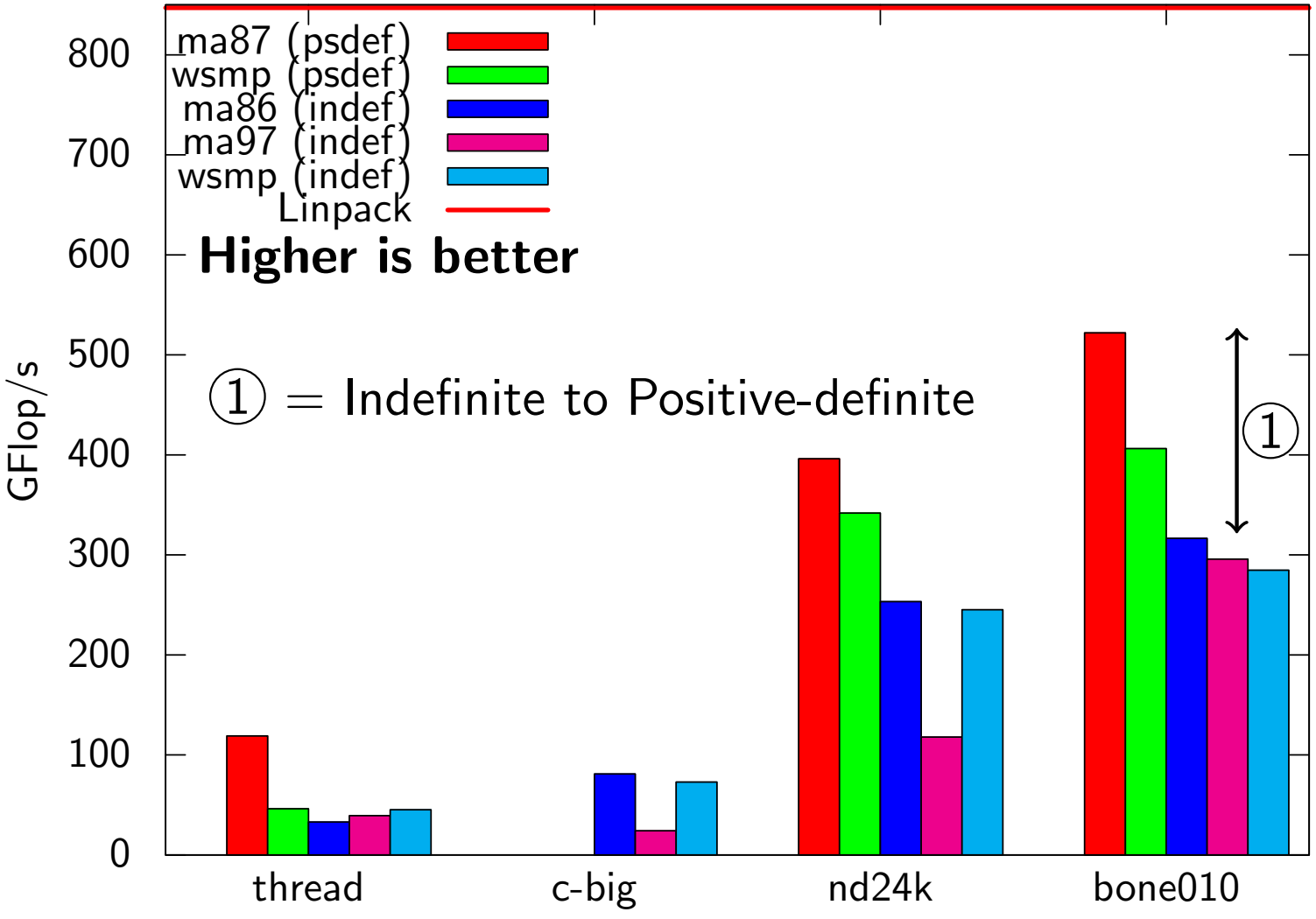
Current Performance Gaps



2×E5-2695 v3 (Haswell-EP) = 28 cores

Science & Technology Facilities Council

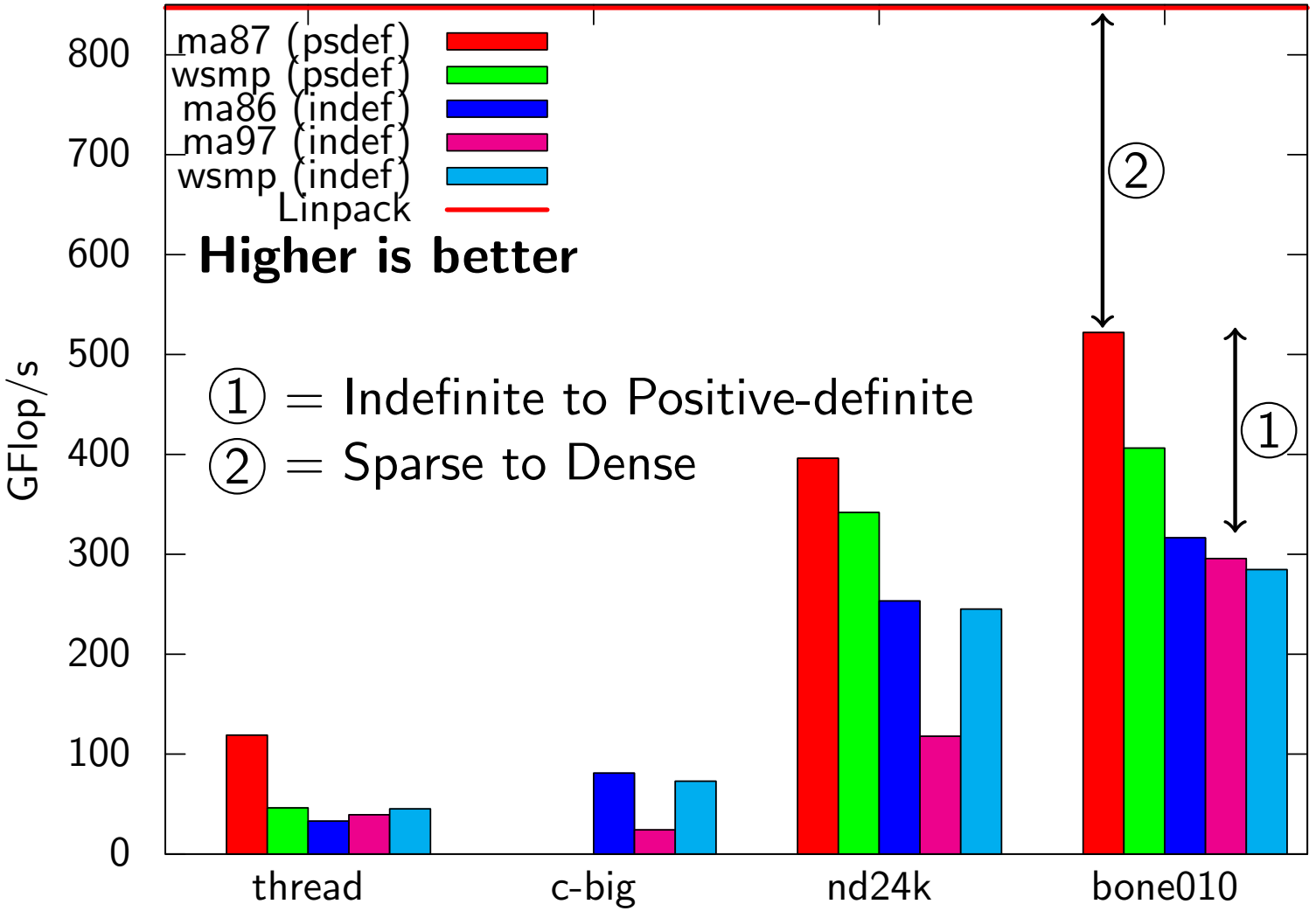
Current Performance Gaps



2×E5-2695 v3 (Haswell-EP) = 28 cores

Science & Technology Facilities Council

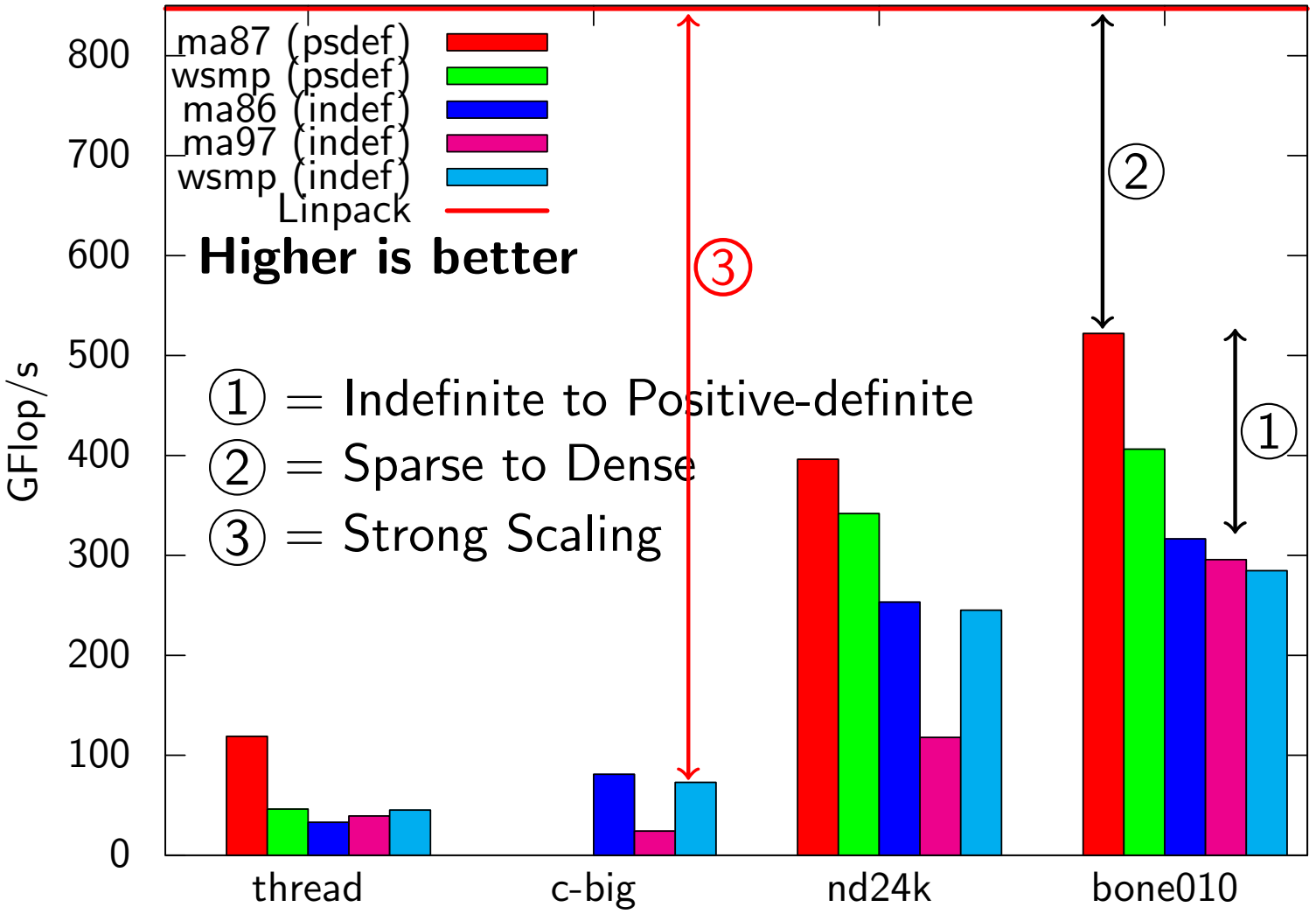
Current Performance Gaps



2×E5-2695 v3 (Haswell-EP) = 28 cores

Science & Technology Facilities Council

Current Performance Gaps



2×E5-2695 v3 (Haswell-EP) = 28 cores

Main focus of this talk

Strong scaling

Memory bound. Latency bound.

Vectorization? Probably.

Who cares?

Small problems. These problems are fast.
Performance irrelevant for a single factor/solve?

- Repeated solution adds up.
- e.g. Ipopt non-linear optimization solver.
- Problem *HVYCRASH*:
  - 0.03 sec per factor/several solves.
  - 748 iterations.
  - 25.0 sec total.

Who cares?

Small problems. These problems are fast.
Performance irrelevant for a single factor/solve?

- Repeated solution adds up.
- e.g. Ipopt non-linear optimization solver.
- Problem *HVYCRASH*:
  - 0.03 sec per factor/several solves.
  - 748 iterations.
  - 25.0 sec total.

If we're lucky:

Work extends to subtrees of larger problems.

Let's focus on the simplest problem

Sparse Cholesky

Theoretical limits?

Obviously matrix dependent.

DNVS/thread

A: $n = 29736, nnz = 4.44 \times 10^6$

L, metis 5, $nemin = 1$: $nnz = 2.43 \times 10^7, nflops = 3.65 \times 10^{10}$

L, metis 5, $nemin = 32$: $nnz = 2.54 \times 10^7, nflops = 3.72 \times 10^{10}$

	Cores	*Theoretical*	*LINPACK*	*HSL_MA87*	*PARDISO*
Desktop:	1×4	99.2 Gflop/s	93.4 Gflop/s	32.3 Gflop/s	32.4 Gflop/s
i7-3770S (Sandy Bridge)			94.2%	32.6%	32.7%
Server:	2×14	1030.4 Gflop/s	844.8 Gflop/s	163.7 Gflop/s	158.7 Gflop/s
E5-2695 v3 (Haswell)			82.0%	15.9%	15.4%

Stream triad 102 GB/s, limits to 20 TFlop/s $\Rightarrow$ memory bandwidth irrelevant?

How can we improve?

HSL_MA87

Task-based sparse solver

Performance breakdown

DNVS/thread

Factor (DPOTRF)						Oberwolfach/bone010 Max	Overall
#tasks	157	45	41	11	9	505	7425
Gflop/s	0.31	2.06	2.49	4.80	5.59	5.59	1.20
GB/s	0.07	0.30	0.39	0.52	0.52	0.53	0.21
Apply (DTRSM)							
#tasks	6	45	37	92	77	147	12534
Gflop/s	1.77	1.92	3.08	4.87	7.35	8.24	3.82
GB/s	0.23	0.16	0.25	0.31	0.46	0.52	0.26
Dense Update (DGEMM)							
#tasks	0	10	8	219	331	1018	27765
Gflop/s	0.00	5.20	11.08	16.58	20.96	33.14	23.03
GB/s	0.00	0.52	1.09	0.86	1.02	1.56	1.16
Sparse Update (DGEMM + Scatter)							
#tasks	475	705	854	520	82	1097	158022
Gflop/s	0.76	2.39	1.87	5.12	5.98	11.41	6.54
GB/s	0.06	0.07	0.08	0.11	0.13	0.15	0.11
IDLE %	32.57	14.43	28.50	18.57	52.79	0.07	17.39

Per core peak: 36.8 Gflop/s, "$>$3.64 GB/s"

The Problem:

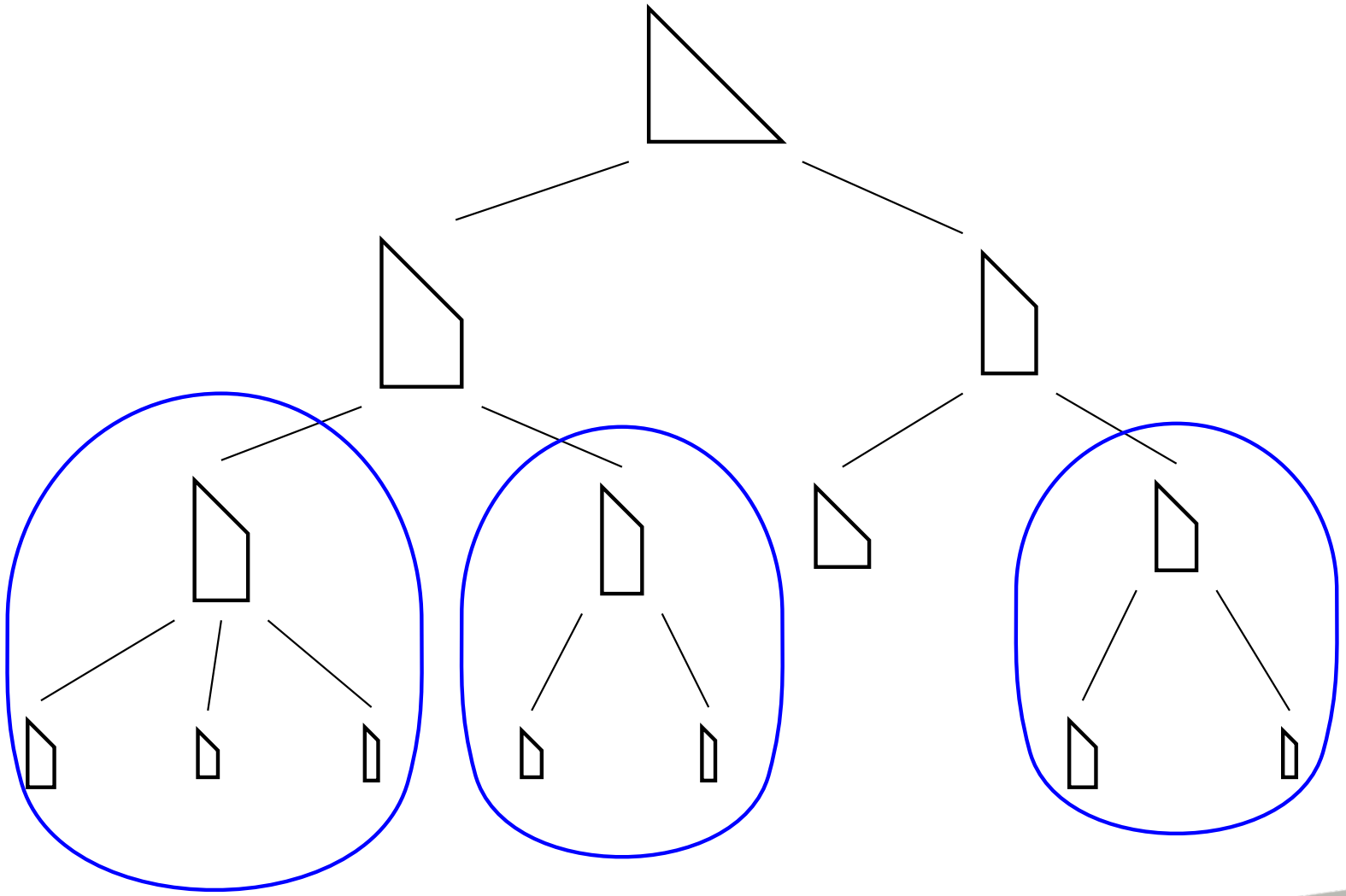
Small nodes near leaves of tree
$\Rightarrow$ inefficient

Solution #1:

Subtree as single task
(Merge vertically)

- "Sparse supernode amalgamation"
- Fully custom kernel?
- Limited parallelism? (difficult on GPU)

Merge Vertically



"Tree pruning" in HSL_MA87

	default		min storage	
	orig	pruned	orig	pruned
DNVS/thread	**0.44**	**0.44**	0.50	0.49
CEMW/tmt_sym	1.03	1.05	2.05	**0.56**
DNVS/shipsec5	**0.99**	1.03	1.10	1.10
Schmid/thermal2	1.65	1.95	3.69	**1.08**
ND/nd24k	6.76	**6.62**	8.39	8.26
Oberwolfach/bone010	**14.3**	14.7	22.4	21.3
GHS_psdef/audikw_1	17.6	**17.4**	28.3	29.5

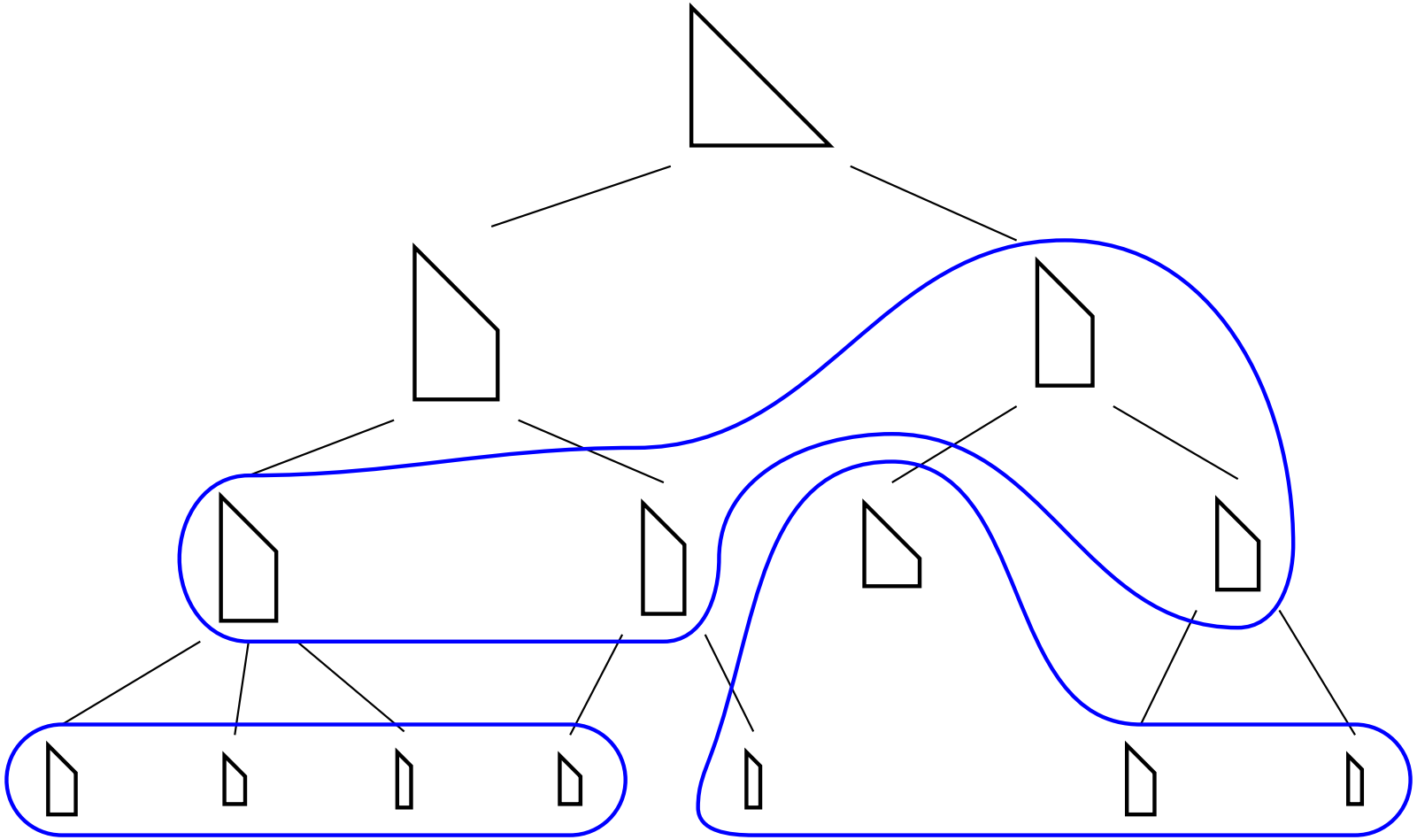
- Supernode amalgamation strategy important
- Performance seems largely to correspond to number of nodes
- Work ongoing for efficient kernel
  (but do we really want to write our own BLAS?)

Solution #2:

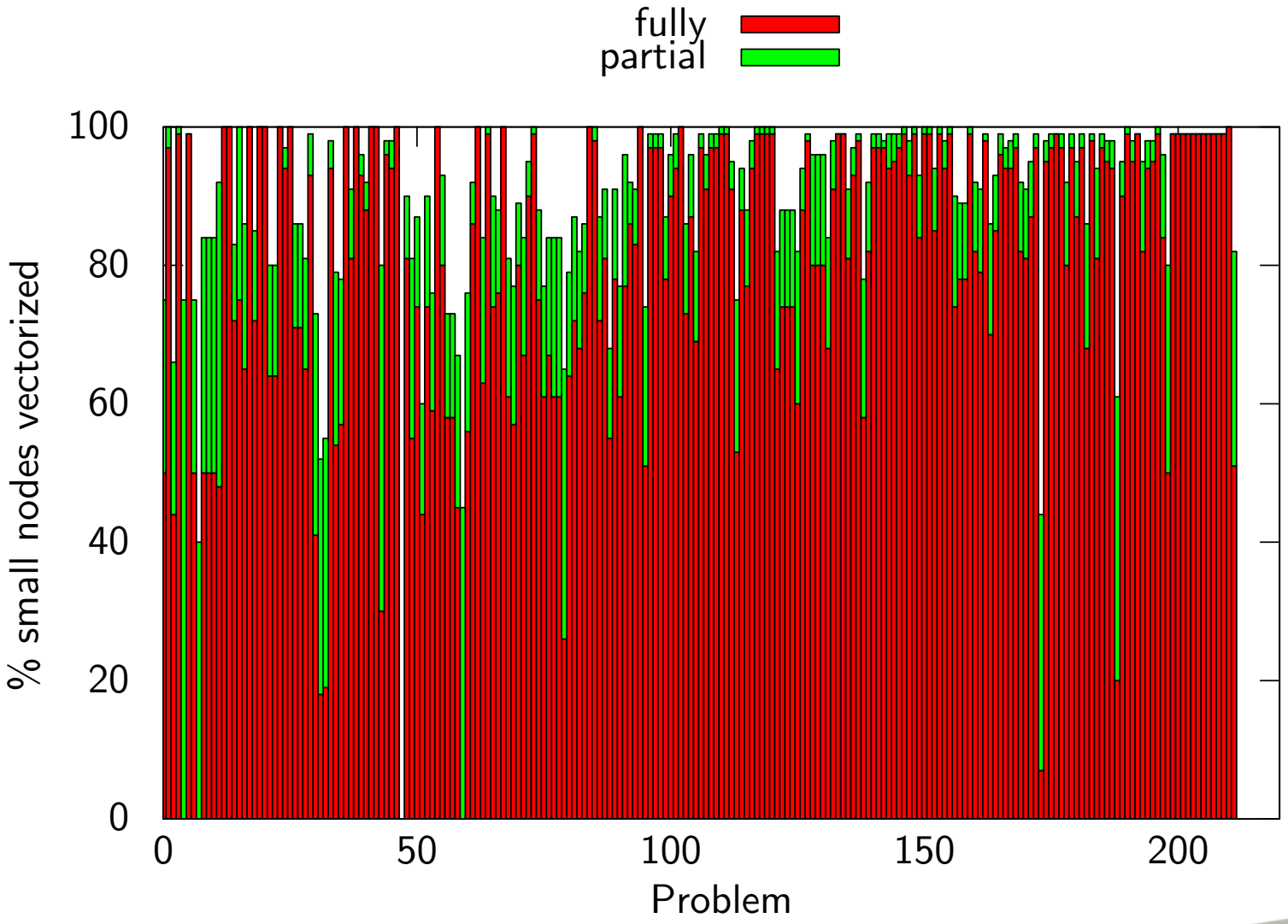
"Level-set" as single task
(Merge horizontally)

- Maps to batched BLAS
- Vectorizes well (eg GPU solver SSIDS)
- Adds synchronizations

Merge Horizontally



Viability

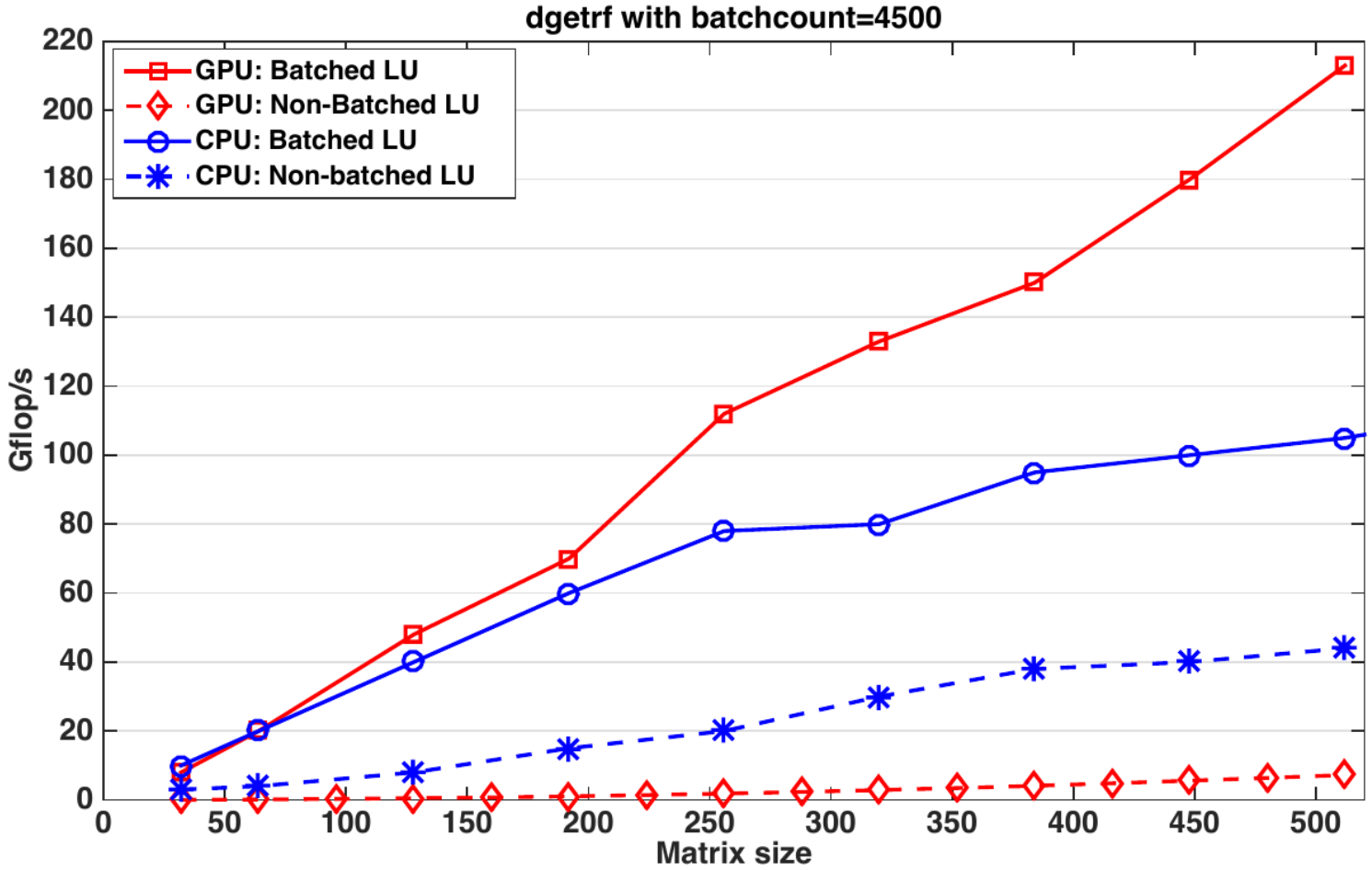


Pending batched BLAS

Open Questions

- Easy win from avoiding BLAS checking overheads on lots of small calls
- Get vectorization on small matrices, aligned loads etc.
- ... too much sparse assembly?
- ... can we combine with _syrk/_gemm calls?

Batched BLAS



Work by Stanimire Tomov, ICL, Tennessee

Going forwards

- Write some custom subtree kernels
- Align first-child indices and treat as dense?
- Test with batched BLAS from partners
- Build performance models
- Use these to guide tree amalgamation

Harder problems

Sparse LDL^T

Sparse LU ($A + A^T$ pattern)

Similar tricks should hopefully work...

... modulo pivoting

(Future work)

Even Harder problem

Sparse LU Unsymmetric pattern

How do we even do parallel Markowitz efficiently?
???

Thanks for listening!

Questions?

`http://www.nlafet.eu`

`http://www.numerical.rl.ac.uk/spral`

Say what now

An appendix?