On the limits of (and opportunities for?) GPU acceleration

Aparna Chandramowlishwaran, Jee Choi, Kenneth Czechowski, Murat (Efe) Guney, Logan Moon, Aashay Shringarpure, Richard (Rich) Vuduc

HotPar'10, Berkeley — June 15, 2010

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers
David H. Bailey
June 11, 1991
Ref: Supercomputing Review, Aug. 1991, pg. 54--55

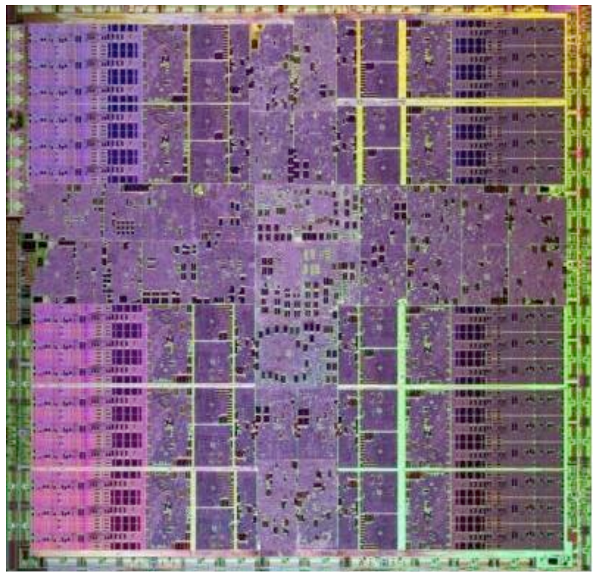
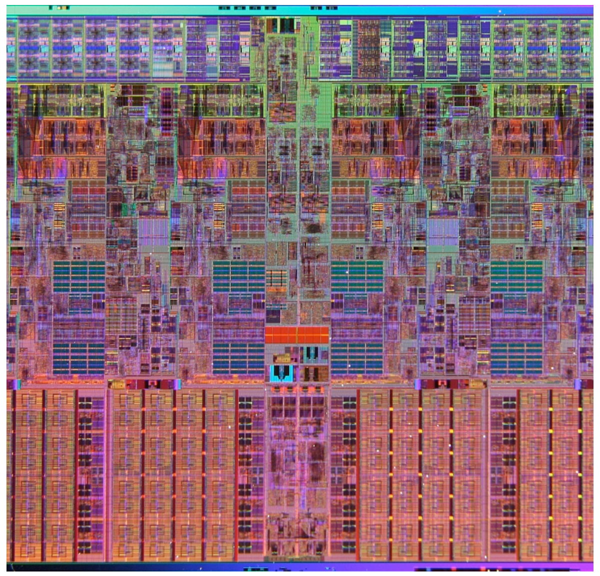
6. Compare your results against scalar, unoptimized code on Crays.

It really impresses the audience when you can state that your code runs several times faster than a Cray, currently the world's dominant supercomputer. Unfortunately, with a little tuning many applications run quite fast on Crays. Therefore you must be careful not to do any tuning on the Cray code. Do not insert vectorization directives, and if you find any, remove them. In extreme cases it may be necessary to disable all vectorization with a command line flag. Also, Crays often run much slower with bank conflicts, so be sure that your Cray code accesses data with large, power-of-two strides whenever possible. It is also important to avoid multitasking and autotasking on Crays --- imply in your paper that the one processor Cray performance rates you are comparing against represent the full potential of a $25 million Cray system.

Q: Port to GPU?

(Posed to me by Scott Klasky at ORNL)

- A: Given roughly same level of tuning & power*, …

GPU  $\cong$  CPU

- Meta-analysis, for **semi-irregular** sci. comp. + data analytics apps (sparse iterative + direct solvers; tree-based particle methods)

Summary of “limits”

- Bandwidth-bound: Aggregate bandwidth < 3x, restricted access patterns
- Compute-bound: 5 to 10x peak potential, but there is a multithreading granularity mismatch
- PCIe: Will architects replicate GPU memory system in on-die CPU/GPU?
- Productivity: To 0th order, tuning required on all platforms
  I.e., Bailey’s Rule #6

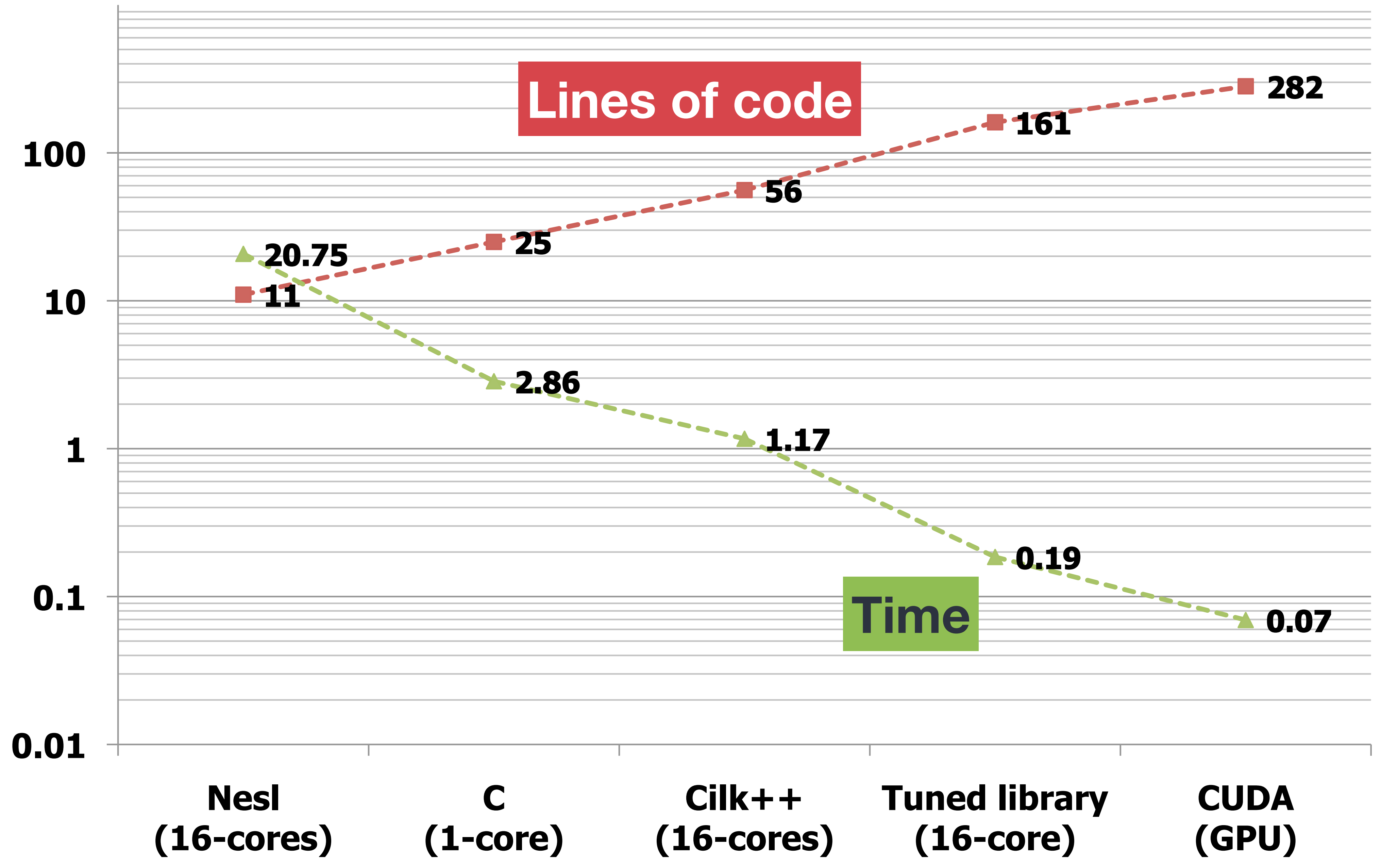
Limitations of this meta-analysis

- Mix of partial results, very rough back-of-the envelope, apples vs. oranges
  - But: “It’s all fruit” — *My Big Fat Greek Wedding*
- Narrow: Scientific computing? (yawn)
  - But: More physically realistic games & graphics, signal analysis, data analytics
- Limited to today’s platforms, and excludes Fermi. What about tomorrow?
  - But: “Prediction is very difficult, especially if it's about the future.” — N. Bohr

Other voices

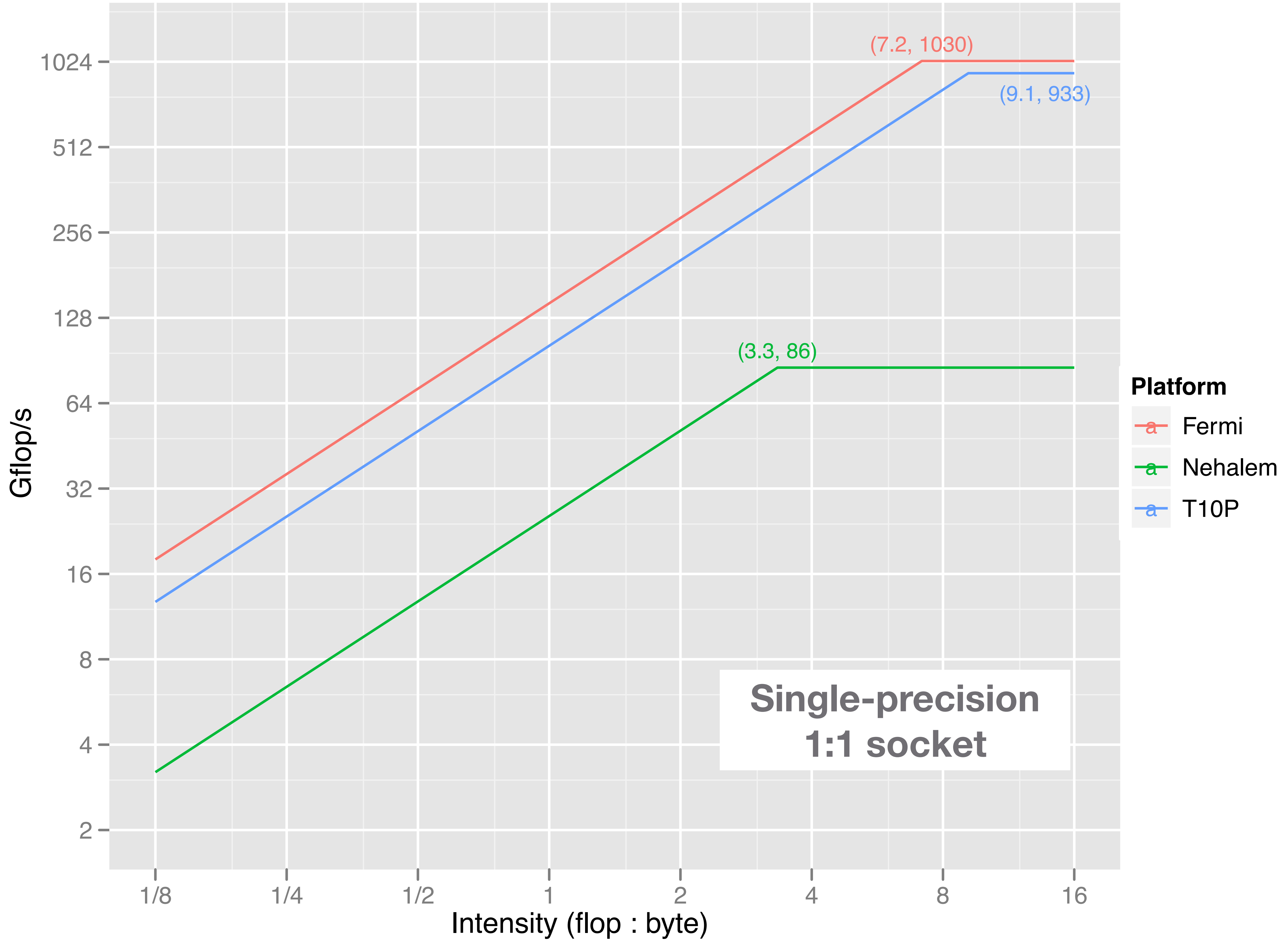
- Vishkin, et al., HotPar'10 poster!
- Bordawekar, et al. (IBM). "Believe it or Not! Multi-core CPUs Can Match GPU Performance for FLOP-intensive Application!" IBM Technical Report RC24982, April 2010.
- Lee, et al. (Intel). "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU." ISCA, June 2010.

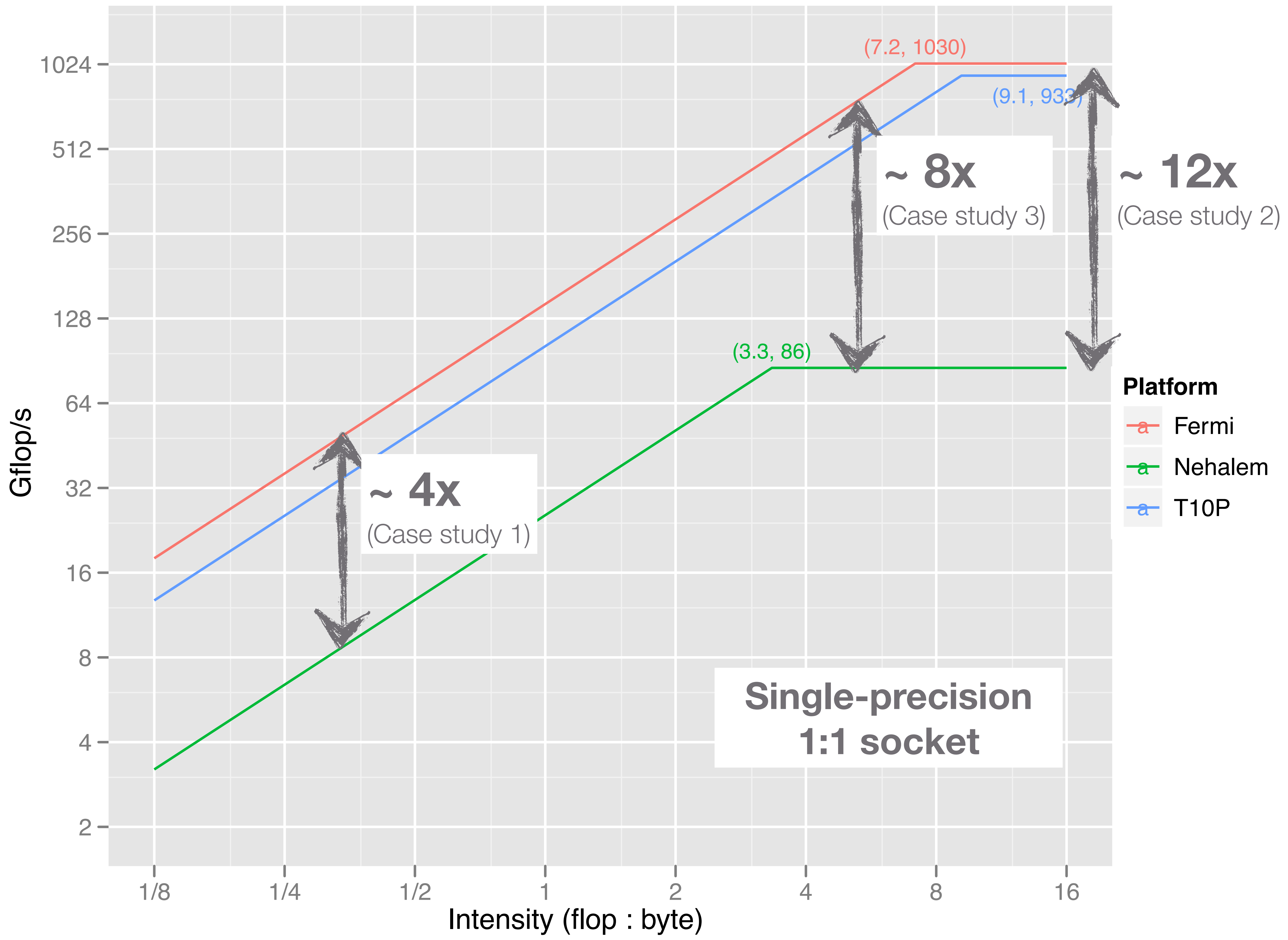
- Performance and productivity expectations



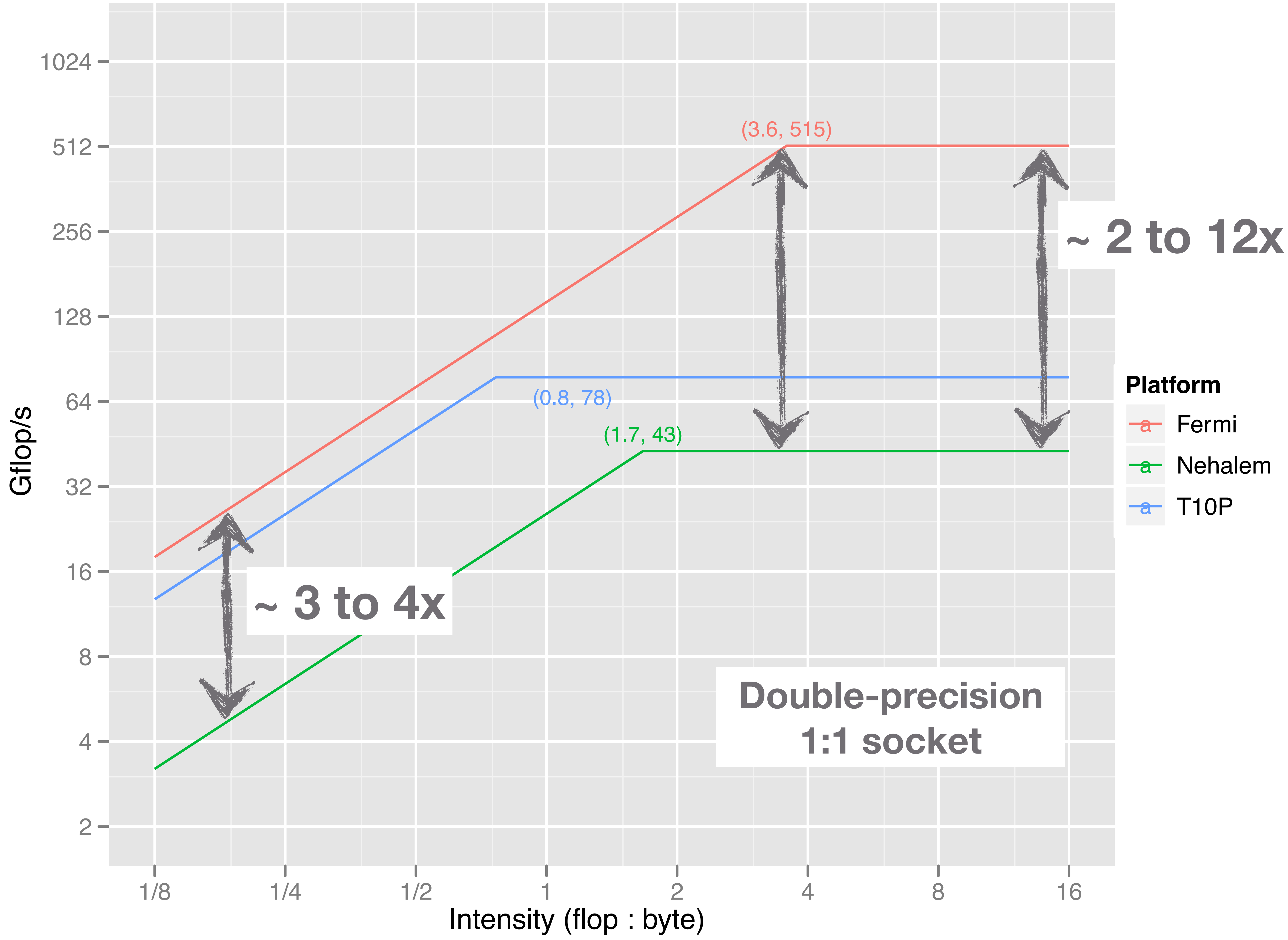
Worth the effort?

Algorithm: Parallel sorting


(7.2, 1030)
(9.1, 933)
(3.3, 86)
Gflop/s
1024
512
256
128
64
32
16
8
4
2
1/8
1/4
1/2
1
2
4
8
16
Intensity (flop : byte)
Platform
Fermi
Nehalem
T10P
Single-precision
1:1 socket



(7.2, 1030)
(9.1, 933)
(3.3, 86)
~ 8x
(Case study 3)
~ 12x
(Case study 2)
~ 4x
(Case study 1)
Single-precision
1:1 socket
Platform
Fermi
Nehalem
T10P
Gflop/s
Intensity (flop : byte)
1024
512
256
128
64
32
16
8
4
2
1/8
1/4
1/2
1
2
4
8
16



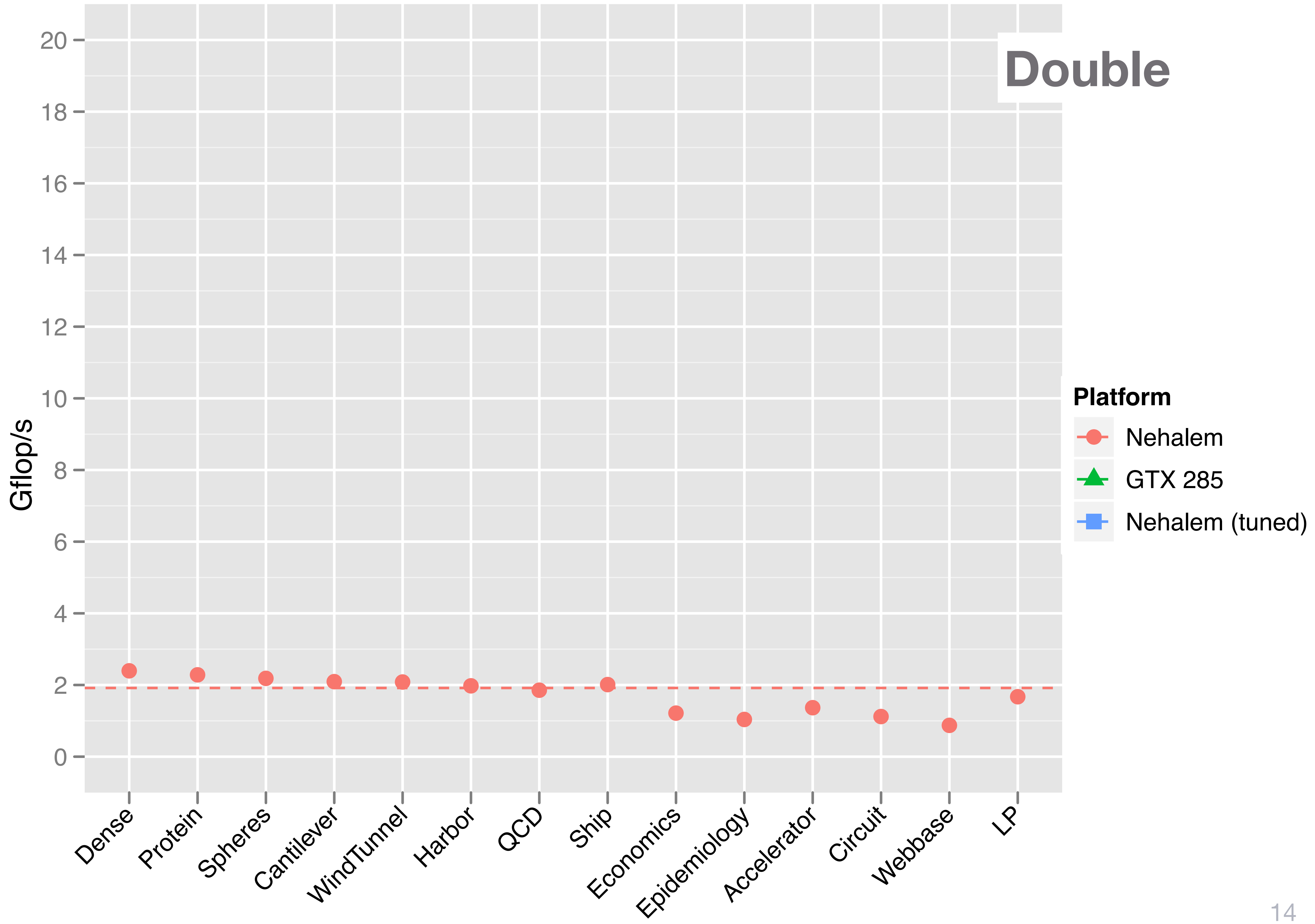
1024
512
256
128
64
32
16
8
4
2
Gflop/s
1/8
1/4
1/2
1
2
4
8
16
Intensity (flop : byte)
(3.6, 515)
(0.8, 78)
(1.7, 43)
~ 2 to 12x
~ 3 to 4x
Double-precision
1:1 socket
Platform
Fermi
Nehalem
T10P

- Case study 1: Sparse iterative solvers

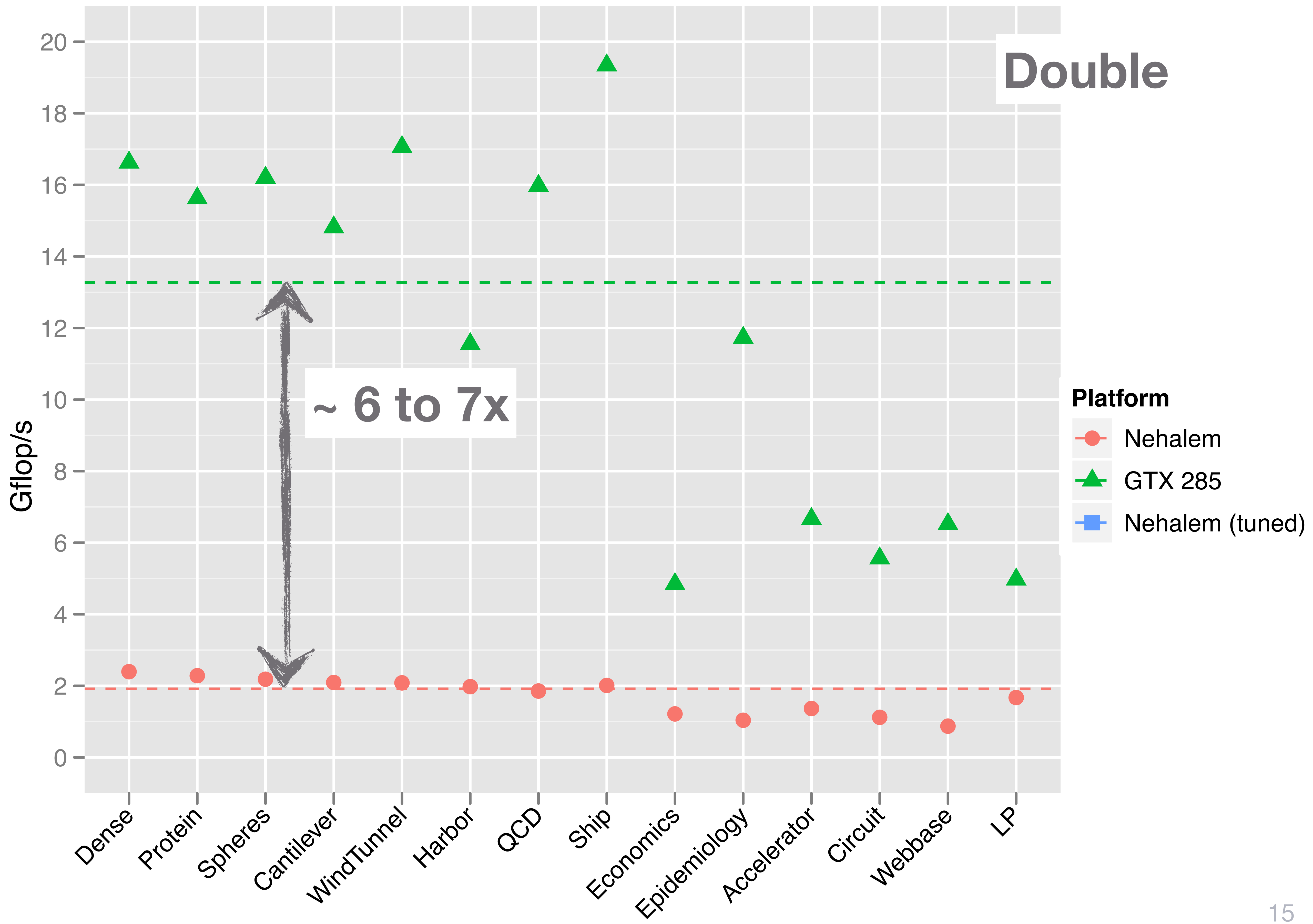
Anatomy of a sparse iterative solver

```
do {
  ...
  y ← A*x        "SpMV"
  ...
} while (!converged)
```

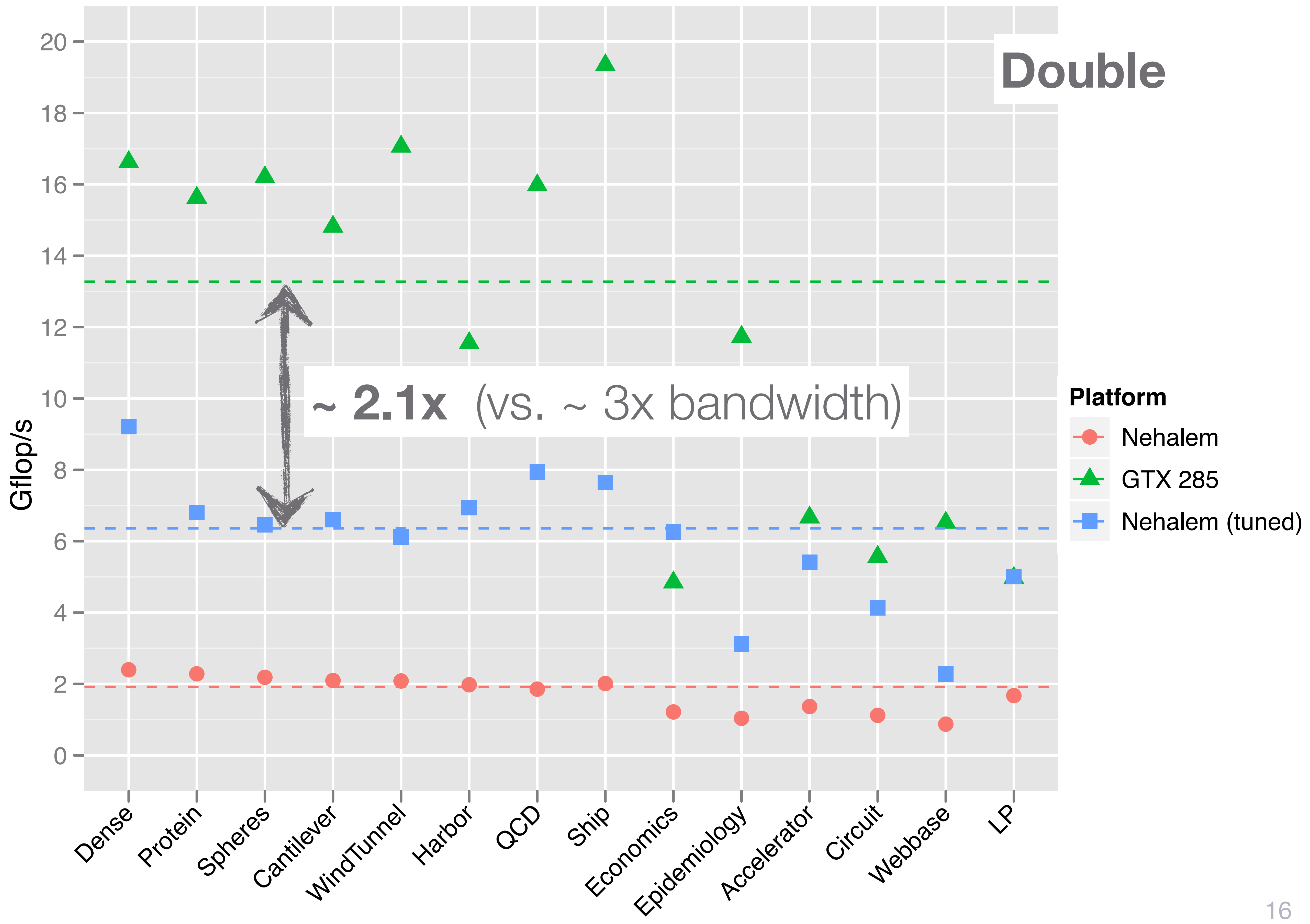
- Bottleneck: Sparse matrix-vector multiply (SpMV)
- Memory bandwidth-limited (stream A): GPU / CPU ~ 3x



S. Williams, N. Bell, J. Choi, M. Garland, L. Oliker, R. Vuduc. “SpMV on MC & Acc.” In *BDK book* (2010).



S. Williams, N. Bell, J. Choi, M. Garland, L. Oliker, R. Vuduc. “SpMV on MC & Acc.” In *BDK book* (2010).



S. Williams, N. Bell, J. Choi, M. Garland, L. Oliker, R. Vuduc. “SpMV on MC & Acc.” In *BDK book* (2010).

Beyond the kernel…

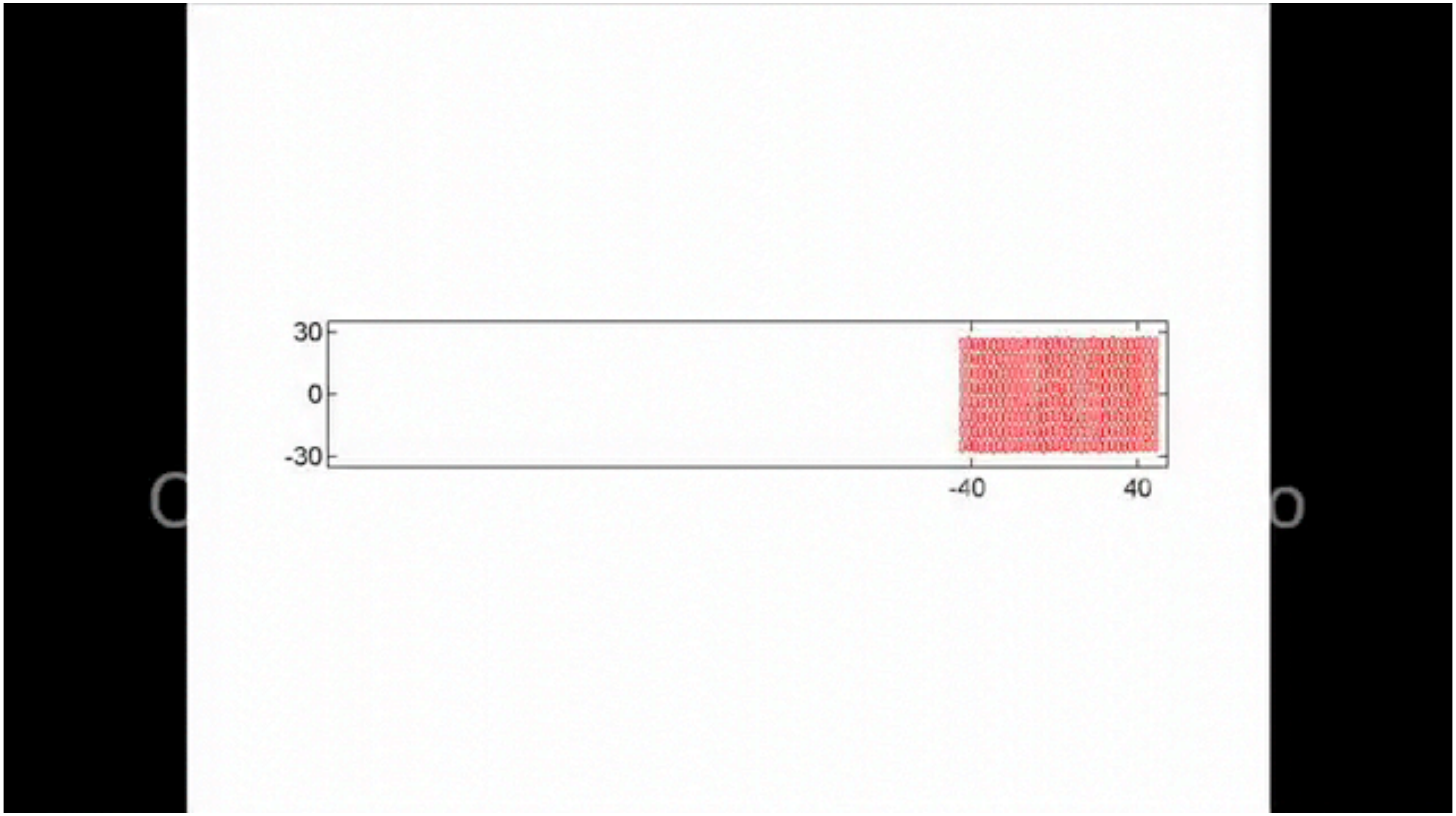
- Optimal data structures differ between CPU & GPU
  ⇒ Startup, transfer cost

- In distributed memory, need to transfer vectors
  ⇒ PCIe limits

- Need ~ 100 iterations to break even, ~ 840 to get 2x on actual solver
  ⇒ Big or not? Depends on app & input

- Case study 2: The Fast Multipole Method

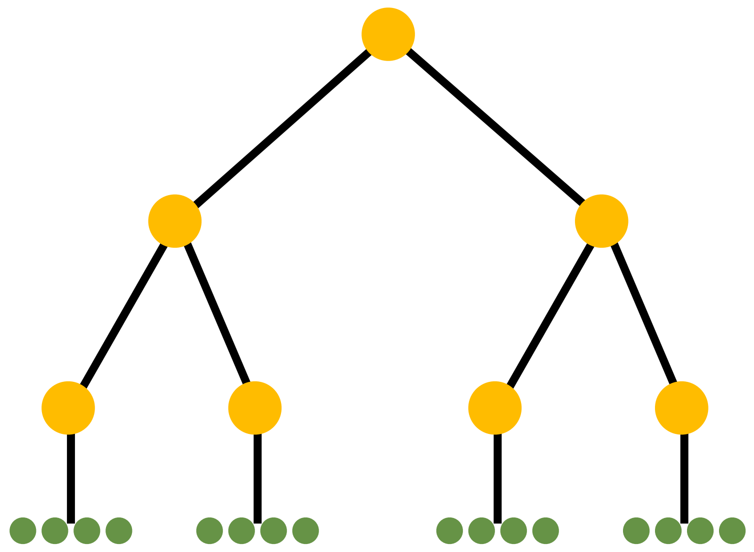
(1) Lashuk, et al., SC’09. (2) Chandramowlishwaran, Williams, et al., IPDPS’10.



Blood cell simulation

(movie) In 2-D with deformations

Anatomy of an FMM



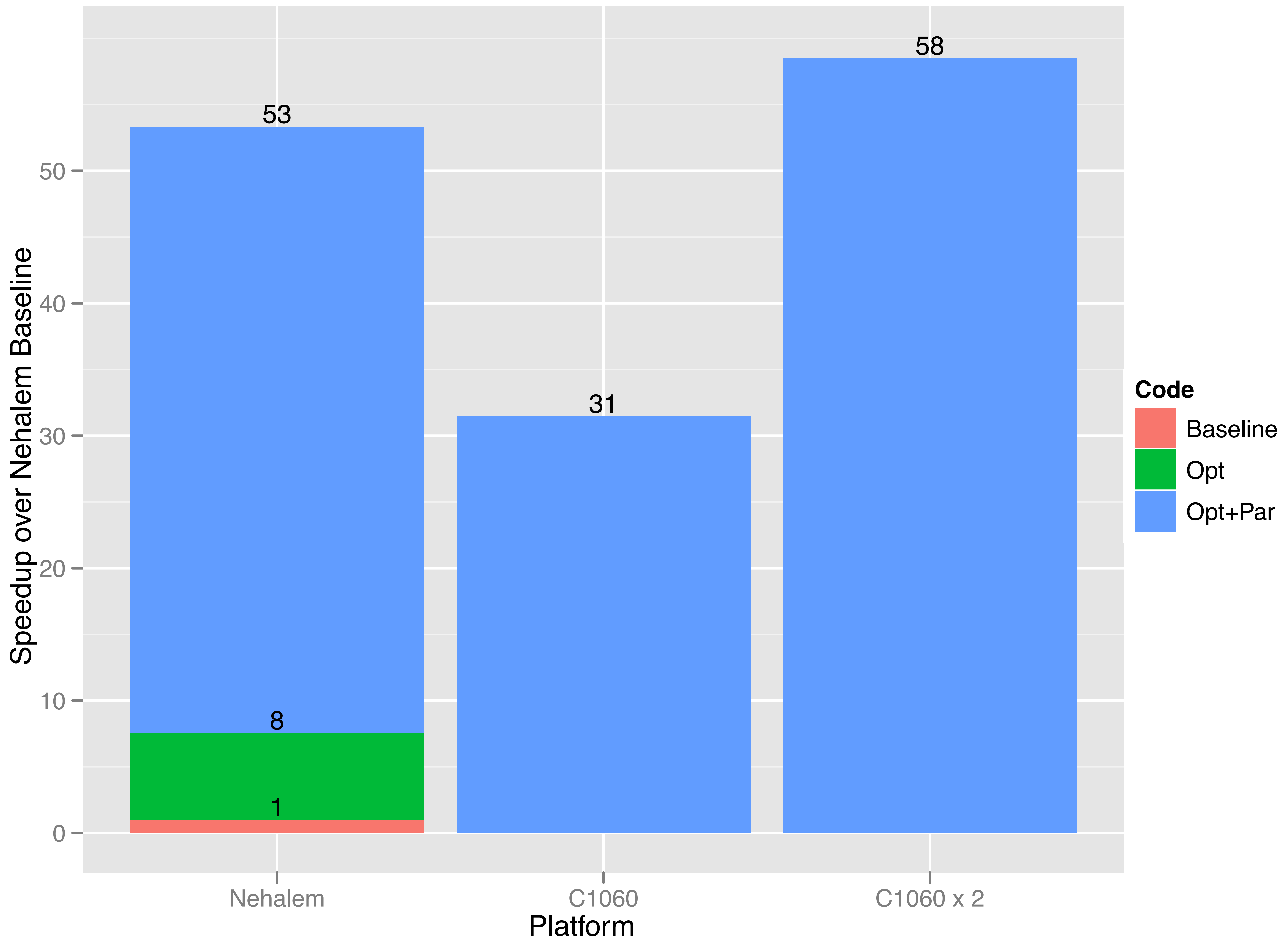
Want:
All pair interactions among green dots
→ O(n^2)

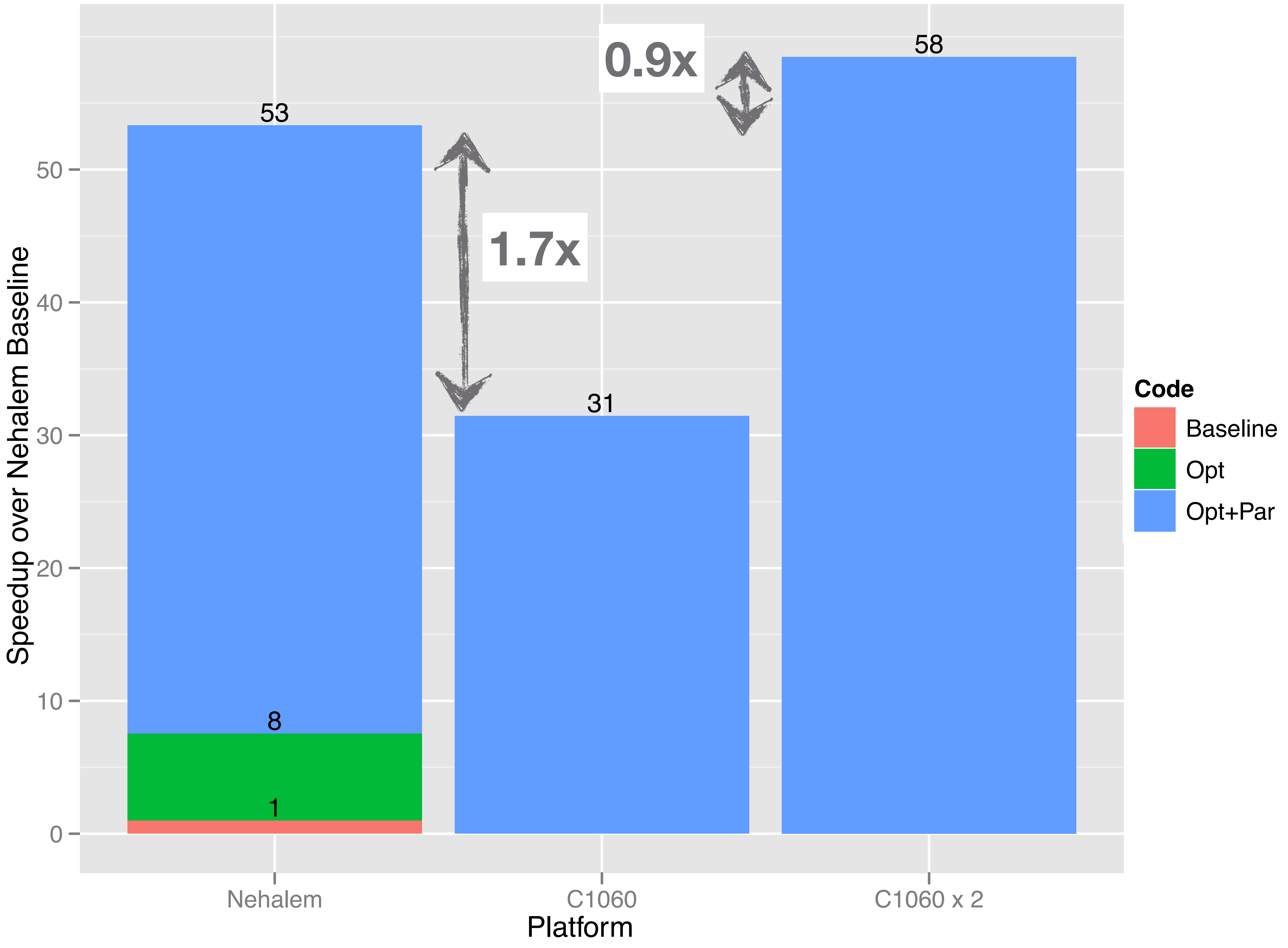
FMM idea:
Build tree, traverse & prune (approx.)
→ O(n log n), with accuracy guarantee

- Bottleneck: “Little” all-pair interactions (b^2) among leaves
- High compute intensity: 12x possible?

The story

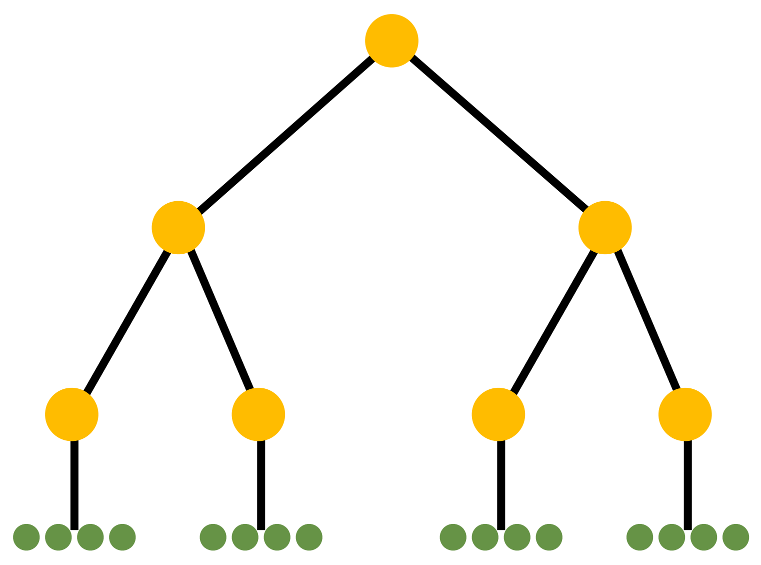
- Baseline: Lashuk, et al., SC'09
  - FMM that scales to 100k cores of "Kraken" machine @ UTK
  - Good overall scalability, but low within-node performance
- Try GPUs?
  - Gumerov & Duraiswami (JCP'08) suggest 30—60x speedups on GPUs
  - We successfully replicate on 256 GPU system at UIUC (1 MPI task / GPU)
- But, stubborn student (Aparna C.) is skeptical of speedup


53
8
1
31
58
Speedup over Nehalem Baseline
0
10
20
30
40
50
Nehalem
C1060
C1060 x 2
Platform
Code
Baseline
Opt
Opt+Par



0.9x
1.7x
58
53
31
8
1
50
40
30
20
10
0
Speedup over Nehalem Baseline
Nehalem
C1060
C1060 x 2
Platform
Code
Baseline
Opt
Opt+Par

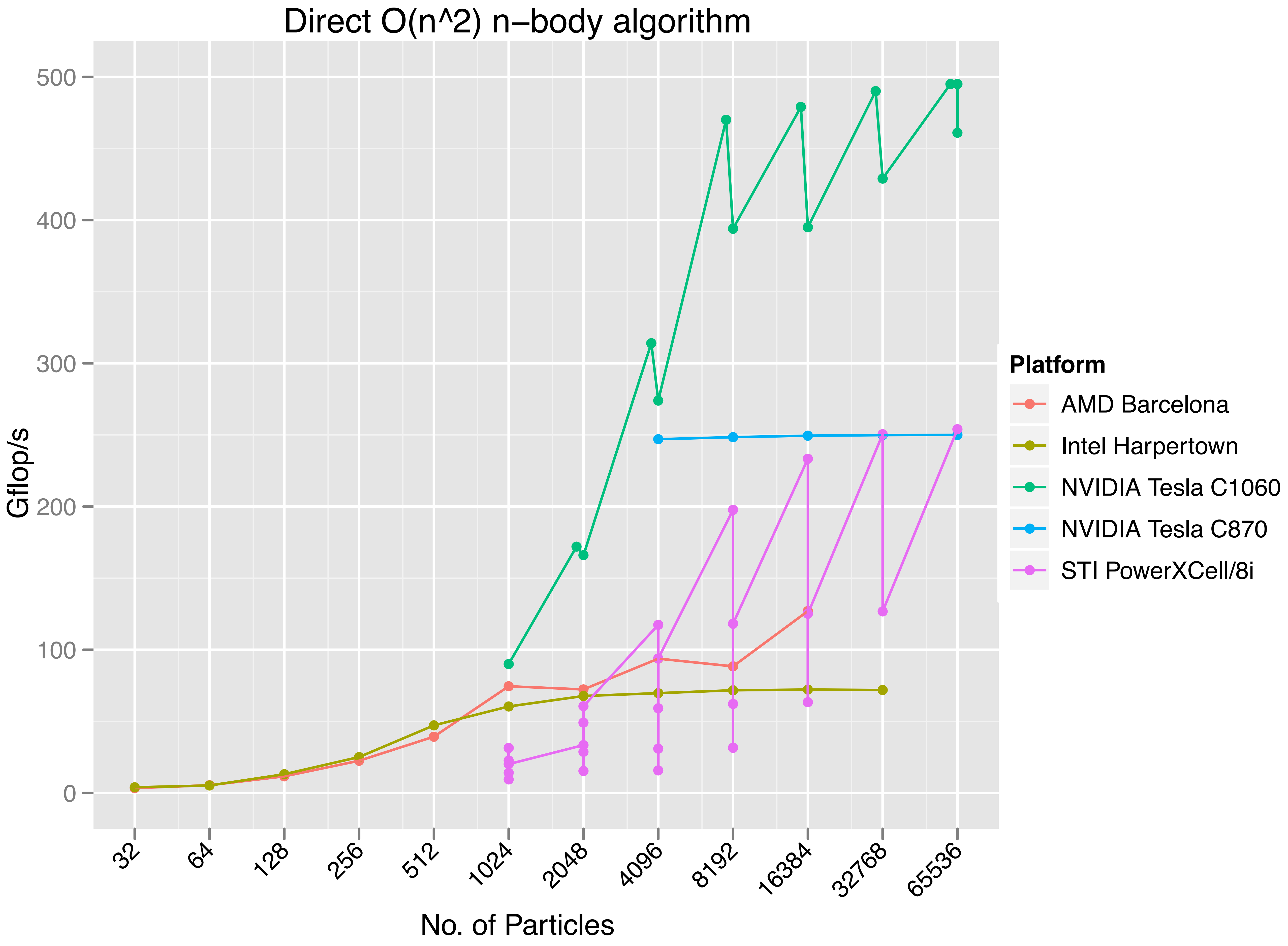
Recall: Anatomy of an FMM

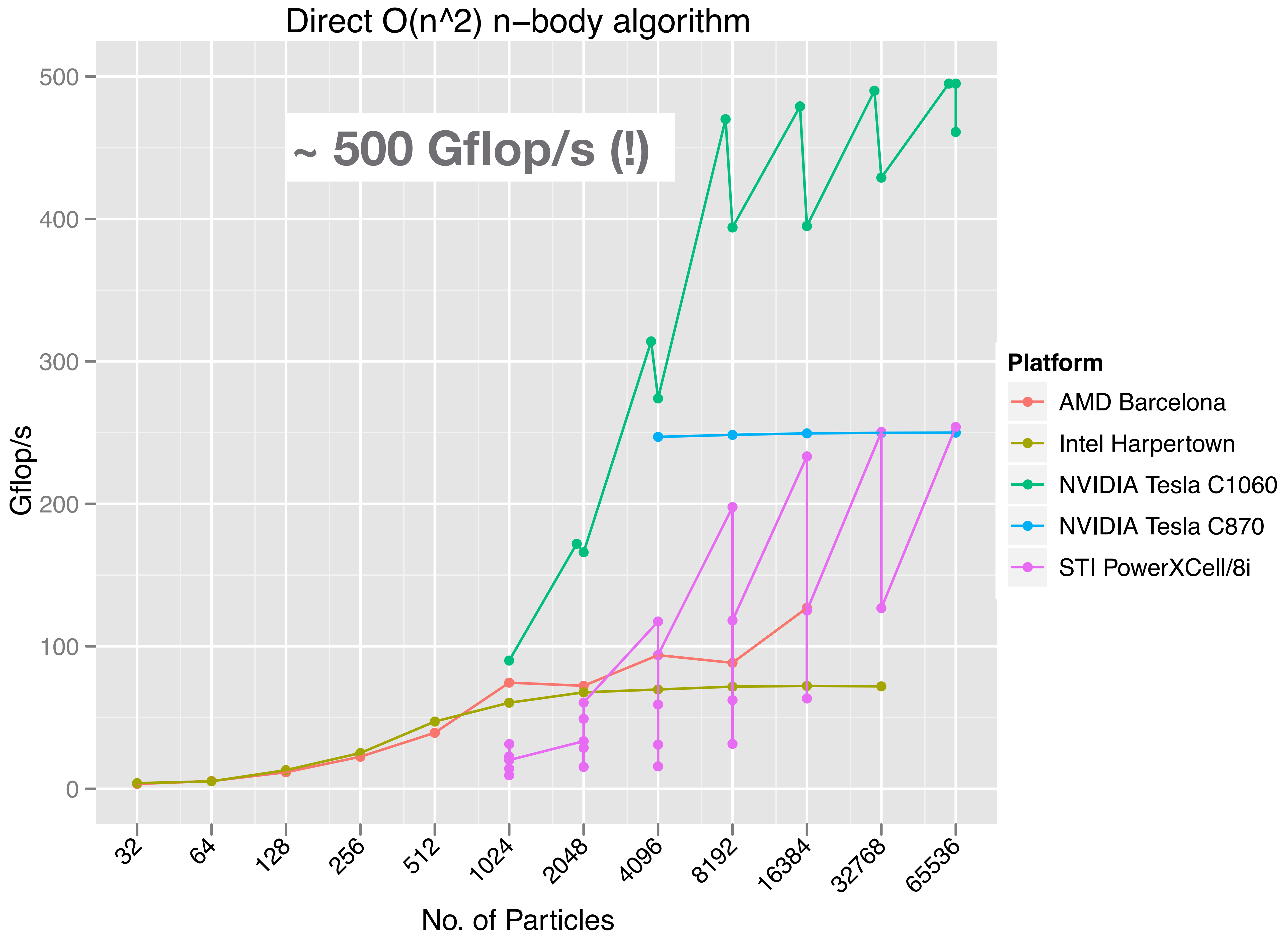


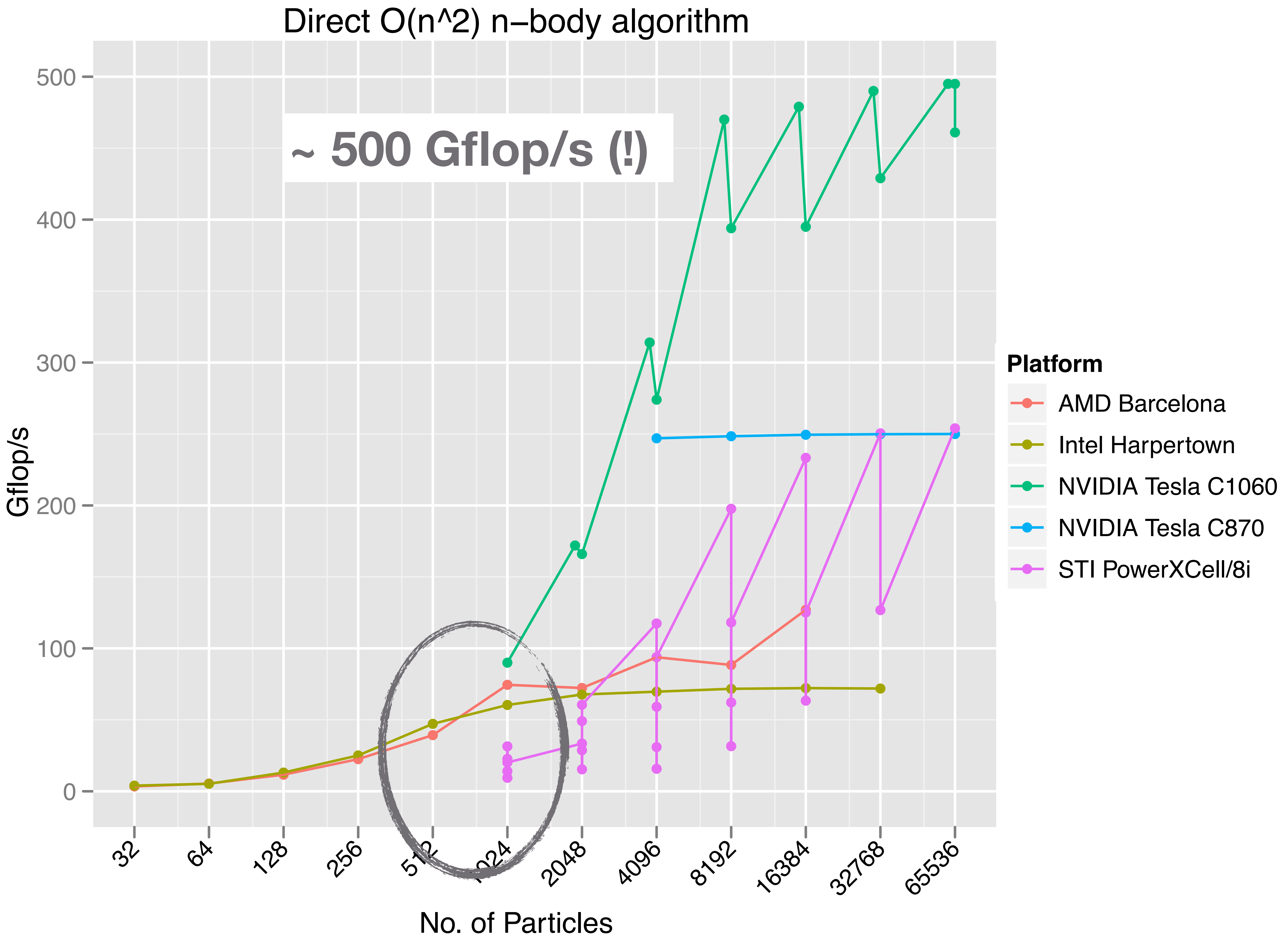
Want:
All pair interactions among green dots
→ O(n^2)

FMM idea:
Build tree, traverse & prune (approx.)
→ O(n log n), with accuracy guarantee

- Bottleneck: **"Little" all-pair interactions (b^2) among nodes**
- High compute intensity: 12x possible?


Direct O(n^2) n−body algorithm
Gflop/s
0
100
200
300
400
500
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
No. of Particles
Platform
AMD Barcelona
Intel Harpertown
NVIDIA Tesla C1060
NVIDIA Tesla C870
STI PowerXCell/8i

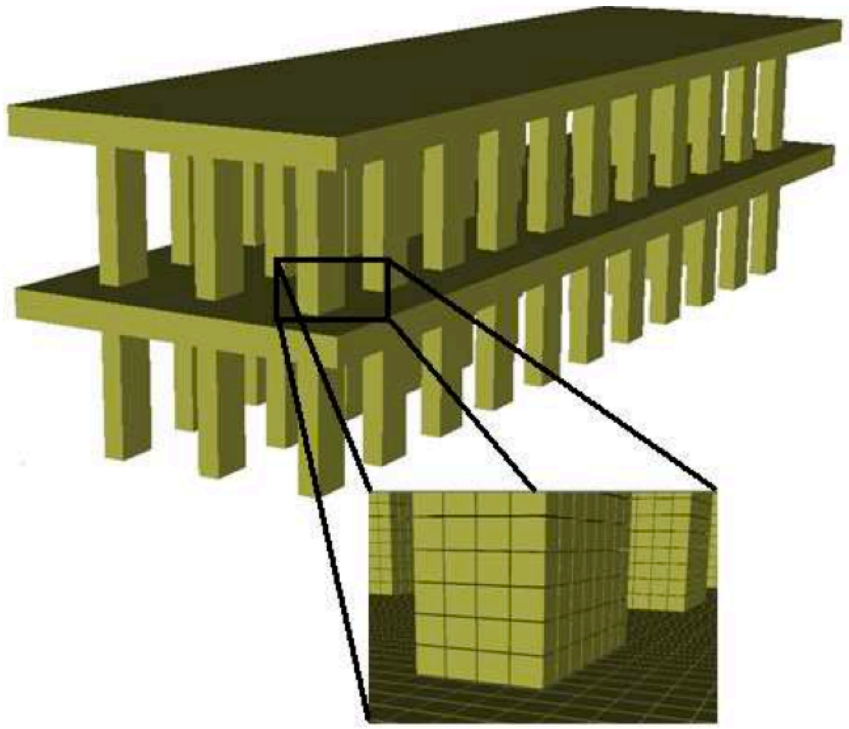

Direct O(n^2) n−body algorithm
~ 500 Gflop/s (!)
Gflop/s
0
100
200
300
400
500
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
No. of Particles
Platform
AMD Barcelona
Intel Harpertown
NVIDIA Tesla C1060
NVIDIA Tesla C870
STI PowerXCell/8i

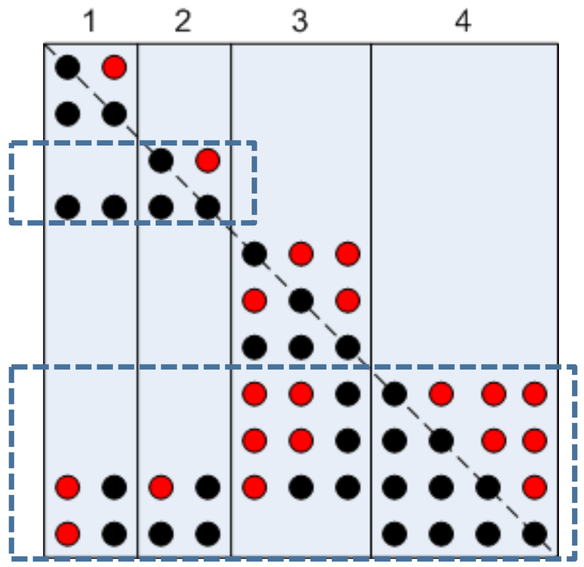

Direct O(n^2) n−body algorithm
~ 500 Gflop/s (!)
Gflop/s
0
100
200
300
400
500
No. of Particles
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Platform
AMD Barcelona
Intel Harpertown
NVIDIA Tesla C1060
NVIDIA Tesla C870
STI PowerXCell/8i

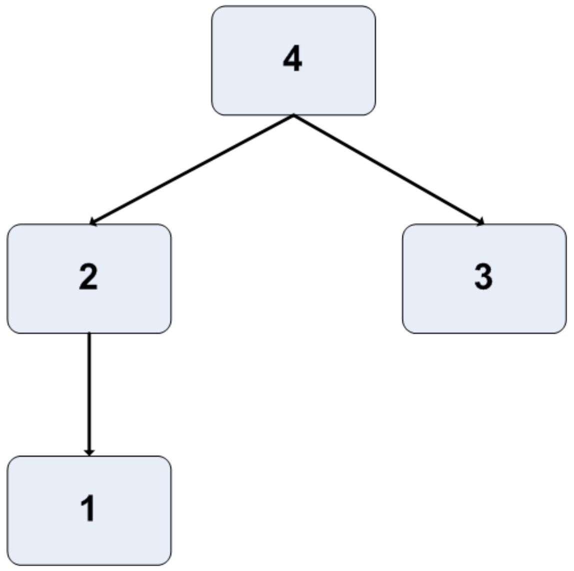
- Case study 3: Sparse direct solvers

(1) M. Efe Guney, Ph.D. Thesis, Georgia Tech, May ’10. (2) Guney, Czechowski, et al. (*in progress*)

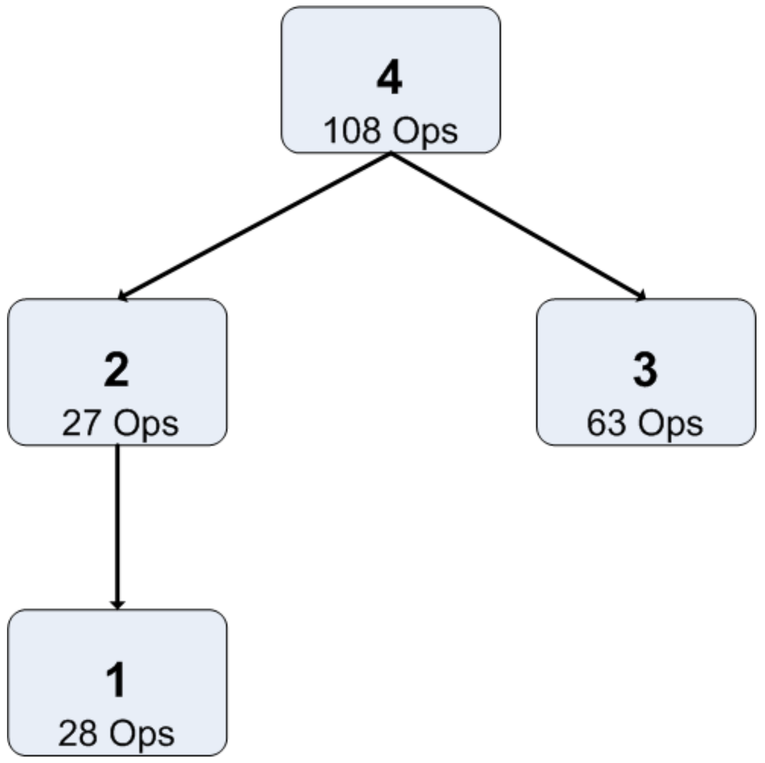
Anatomy of a sparse direct solver

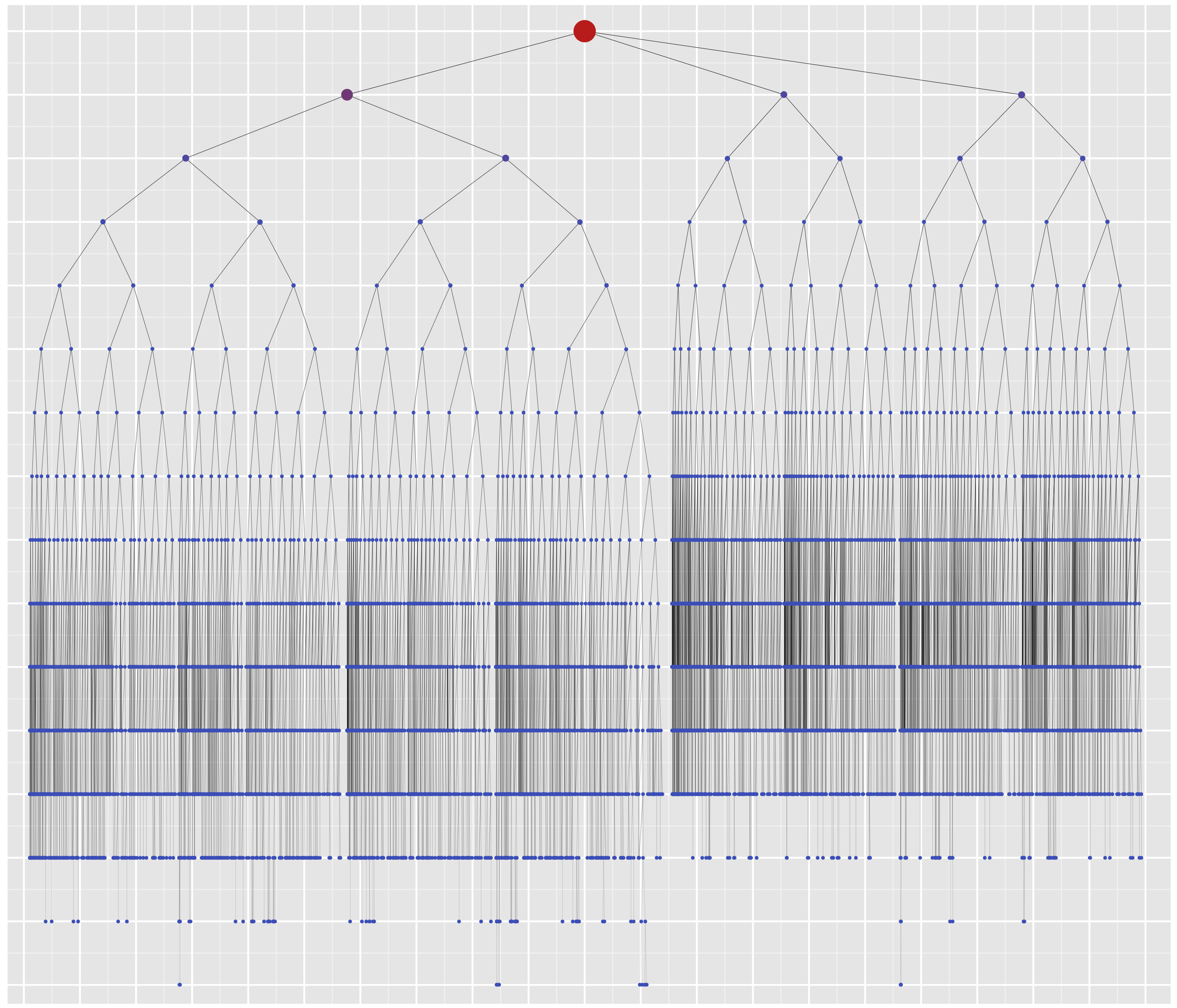






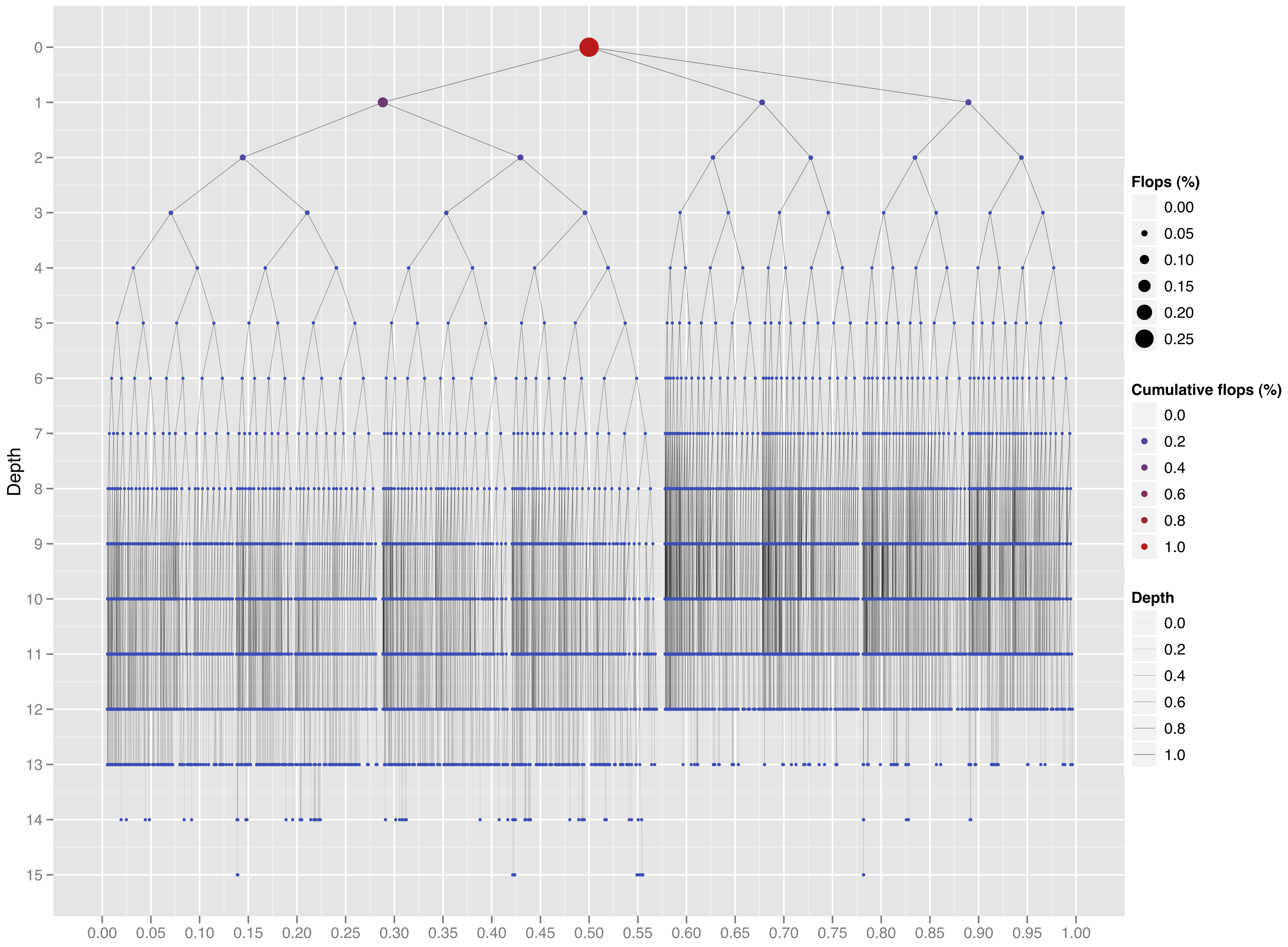
- Sparse Cholesky factorization, $A = L \cdot L^T$, where A & L are sparse
- Mixed compute intensity, average of ~ 4 flops : byte for sample problem



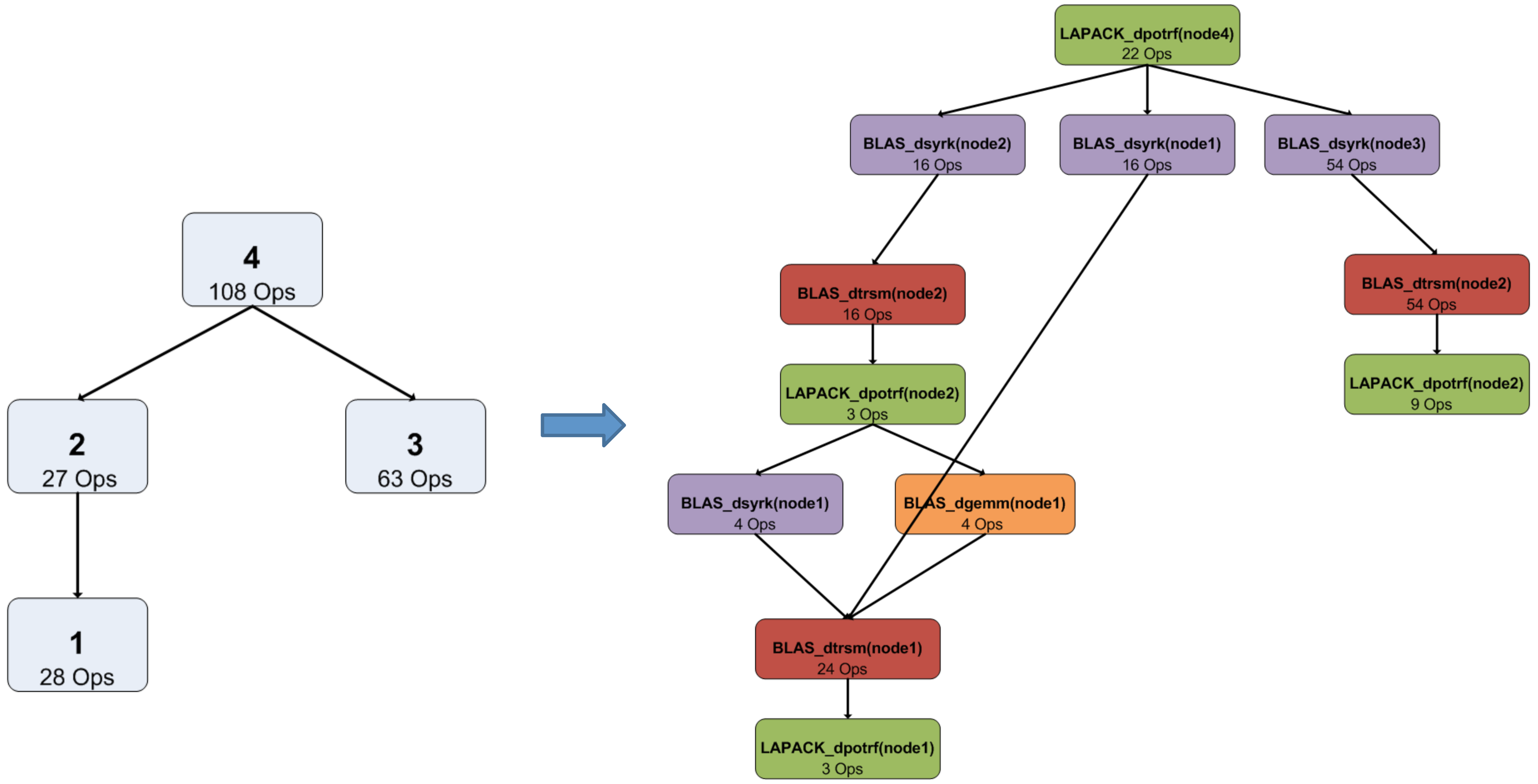


Real elimination tree example

Independent subtrees may be processed in parallel.

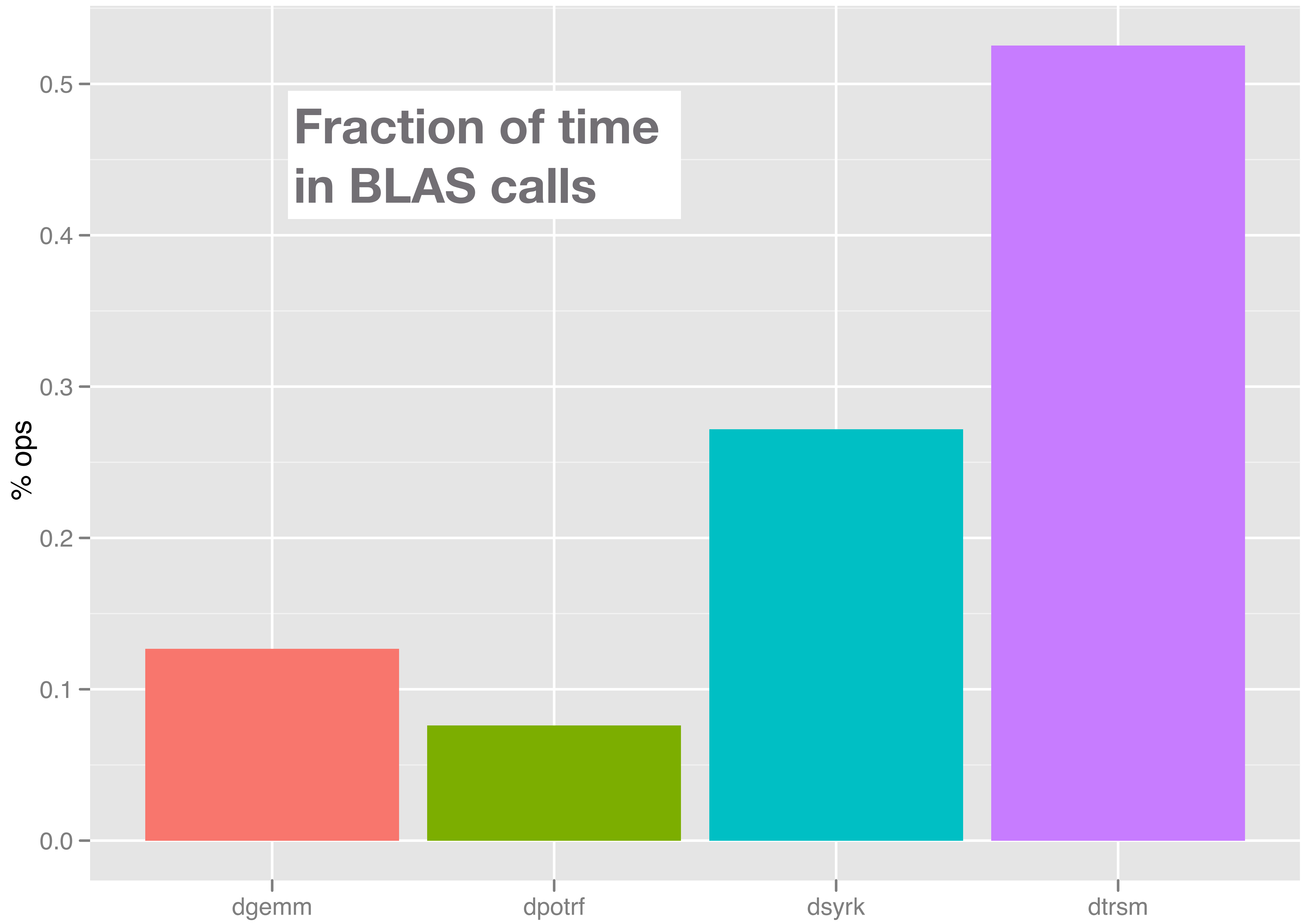


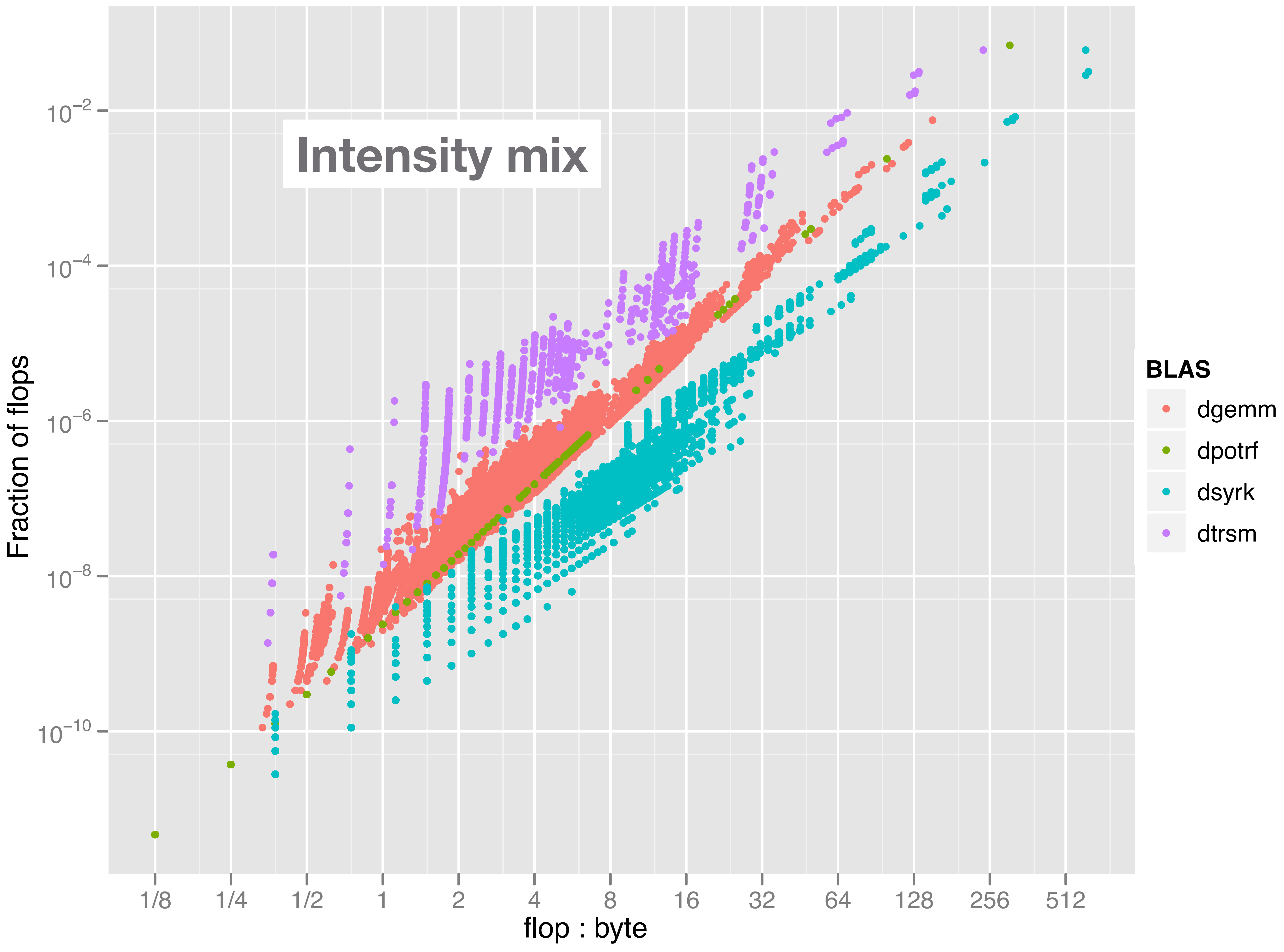
Thursday, July 1, 2010

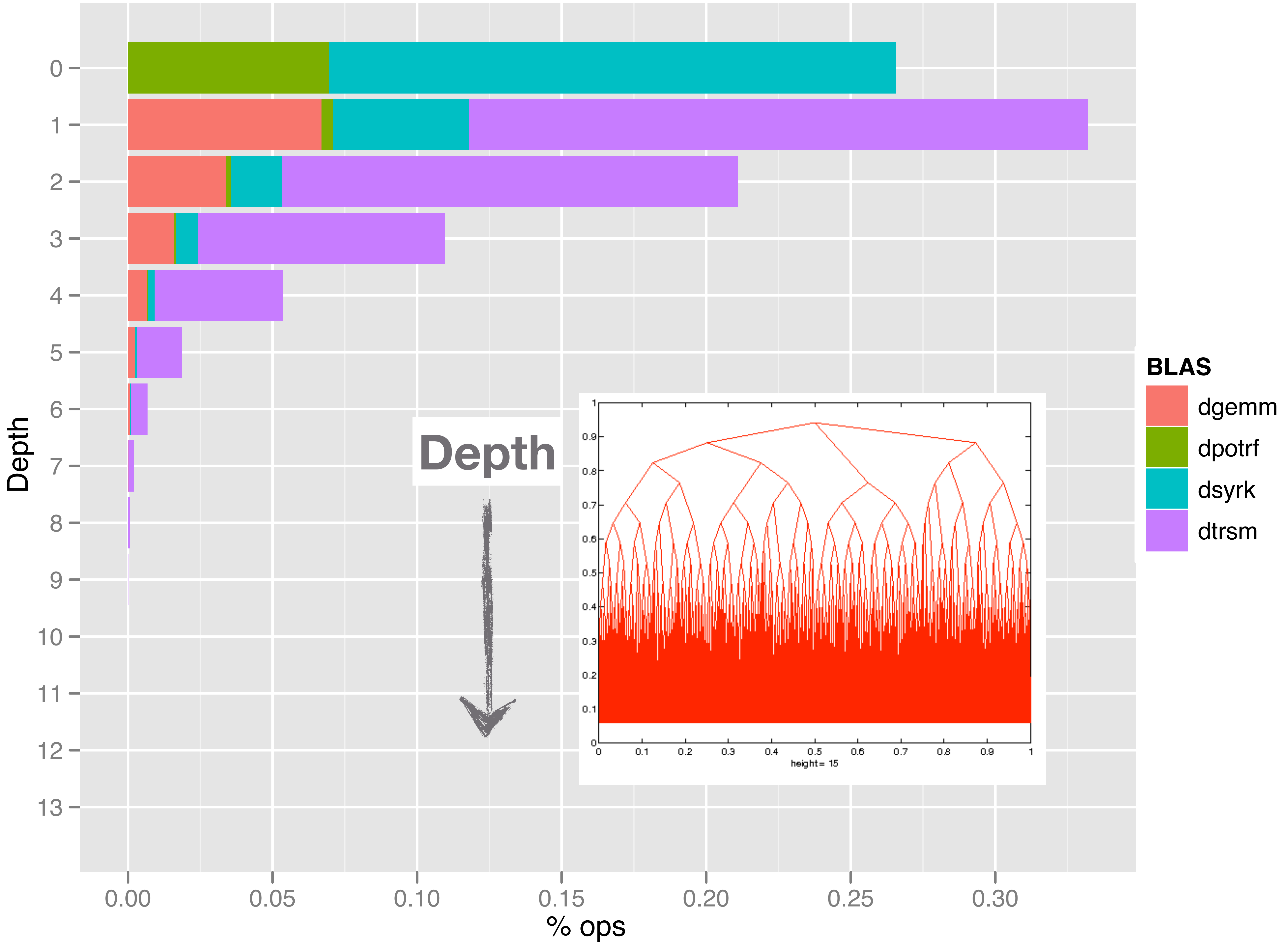


Finer-grained dependencies

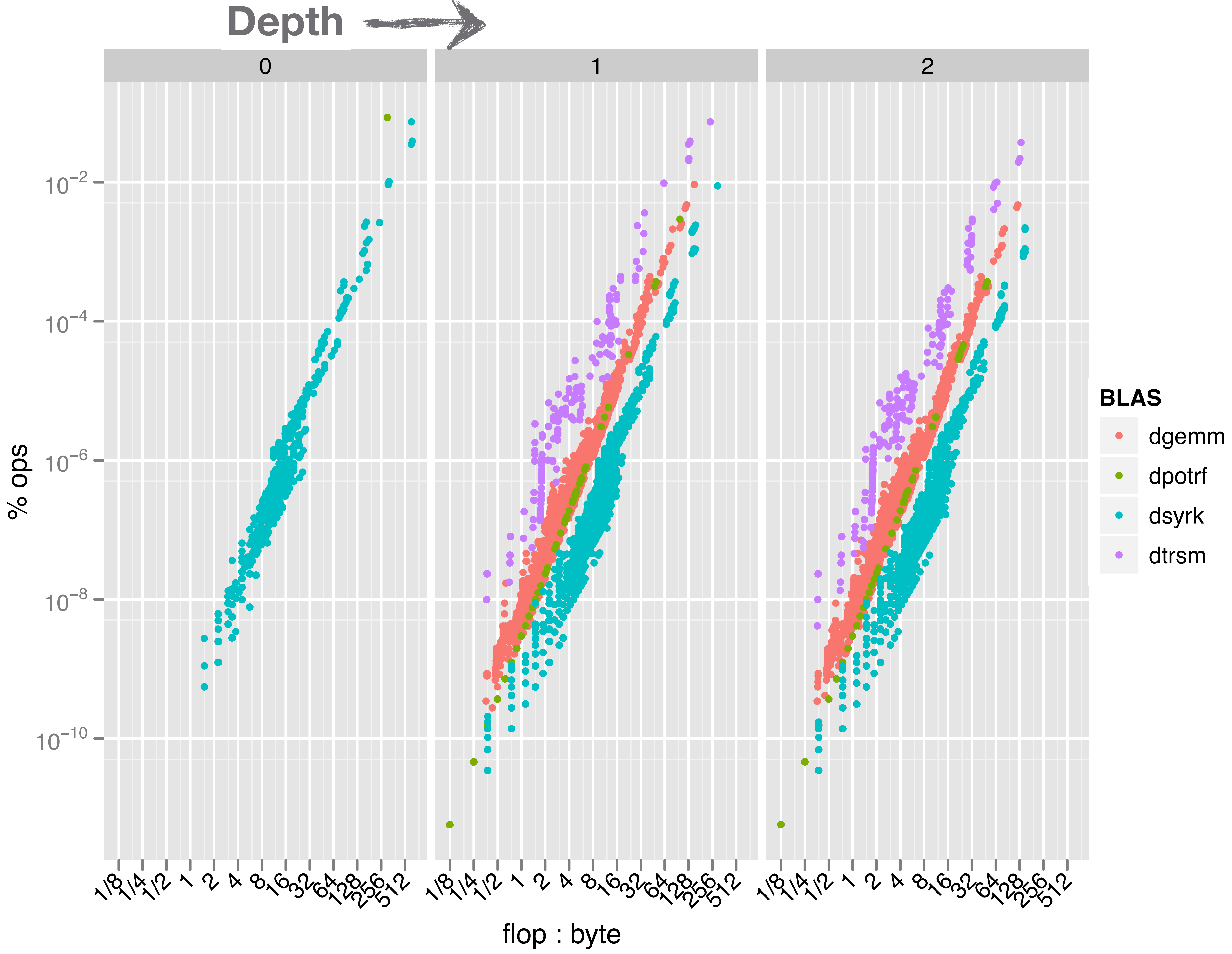
Colored circles on the right are BLAS calls on operands of varying size.

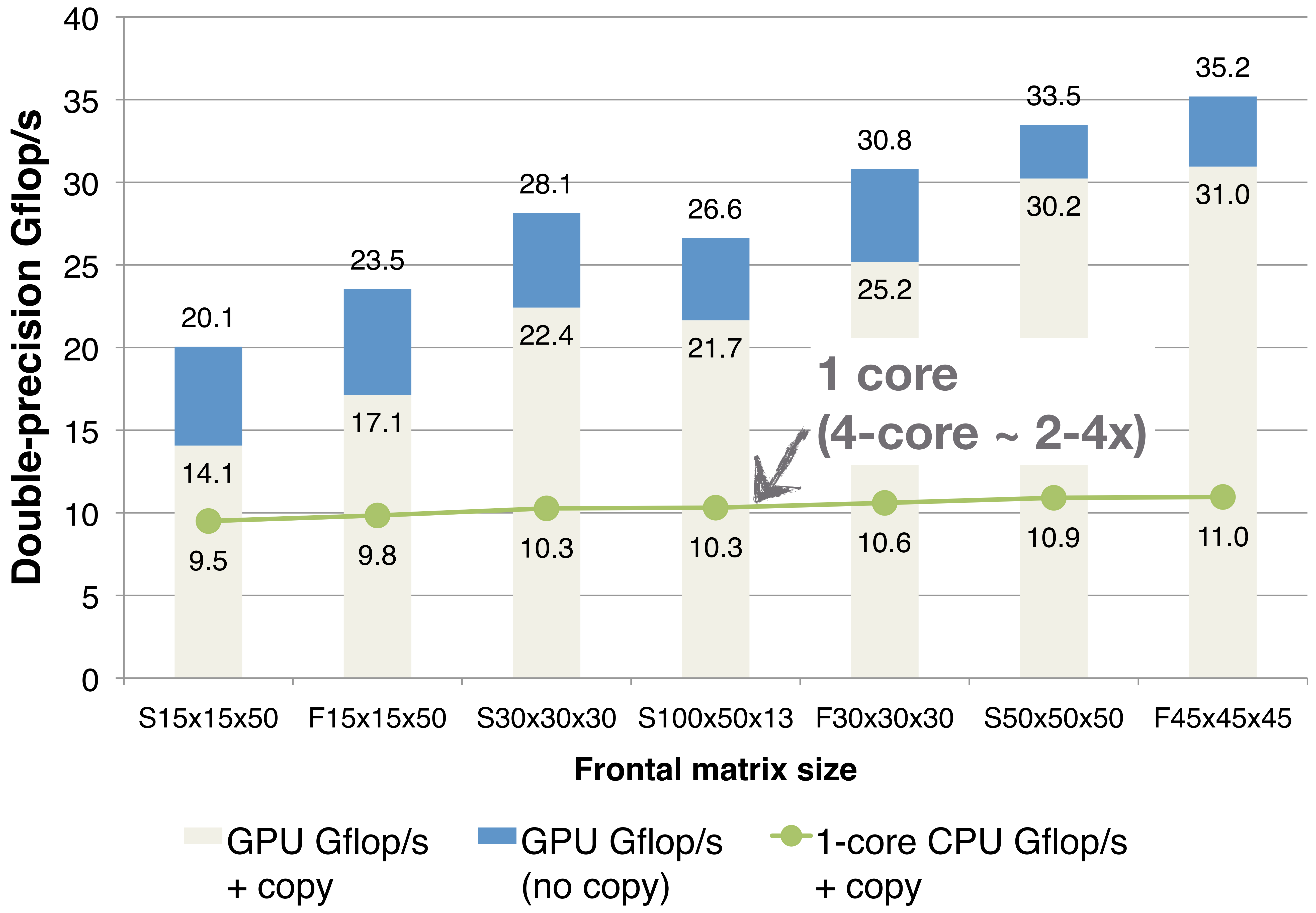

Fraction of time
in BLAS calls
% ops
0.5
0.4
0.3
0.2
0.1
0.0
dgemm
dpotrf
dsyrk
dtrsm


Intensity mix
Fraction of flops
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
flop : byte
1/8
1/4
1/2
1
2
4
8
16
32
64
128
256
512
BLAS
dgemm
dpotrf
dsyrk
dtrsm



Depth
0
1
2
3
4
5
6
7
8
9
10
11
12
13
0.00
0.05
0.10
0.15
0.20
0.25
0.30
% ops
BLAS
dgemm
dpotrf
dsyrk
dtrsm
Depth
height= 15


Depth
0
1
2
% ops
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
1/8
1/4
1/2
1
2
4
8
16
32
64
128
256
512
flop : byte
BLAS
dgemm
dpotrf
dsyrk
dtrsm


Double-precision Gflop/s
0
5
10
15
20
25
30
35
40
20.1
14.1
9.5
23.5
17.1
9.8
28.1
22.4
10.3
26.6
21.7
10.3
30.8
25.2
10.6
33.5
30.2
10.9
35.2
31.0
11.0
1 core
(4-core ~ 2-4x)
S15x15x50
F15x15x50
S30x30x30
S100x50x13
F30x30x30
S50x50x50
F45x45x45
Frontal matrix size
GPU Gflop/s + copy
GPU Gflop/s (no copy)
1-core CPU Gflop/s + copy

Summary: Definite potential, but no "magic"

- For a set of semi-irregular computations, **1 GPU ≅ 1 to 2 CPUs**.
  ⇒ Study heterogeneous computing, but scale-back expectations.
- Barriers?
  - Bandwidth-bound: Aggregate GPU : CPU bandwidth < 3x
  - PCIe: Will we really replicate GPU memory system in on-die CPU/GPU?
  - Compute-bound: 5 to 10x potential, but how? E.g., FMM granularity mismatch
  - Productivity: To 0th order, tuning required on all platforms (Bailey's Rule #6)