Follow along:
hpcgarage.org/ppam13

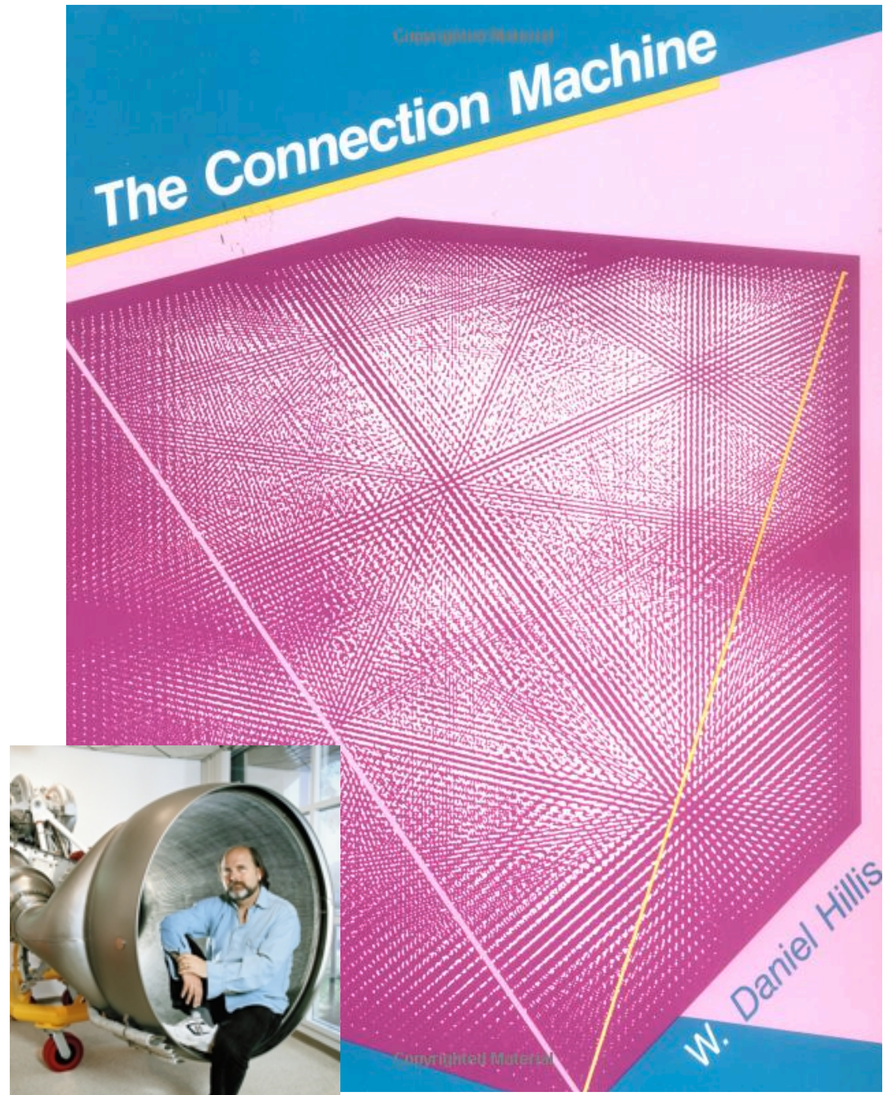
How much (execution) time, energy, and power will my algorithm cost?

Aparna Chandramowlishwaran [MIT] · **Jee Whan Choi** · **Kenneth (Kent) Czechowski** · Marat Dukhan · **Richard (Rich) Vuduc** · Casey Battaglino · Danny Browne [GTRI] · Cong Hou [Google] · David (Dave) S. Noble, Jr. · Piyush Kumar Sao

September 11, 2013 – Int'l. Conf. Parallel Processing & Applied Mathematics – PPAM

ACM Doctoral Dissertation Award Winner (1985)

Follow along:
hpcgarage.org/ppam13



http://www.amazon.com/Connection-Machine-Artificial-Intelligence/dp/0262580977/ref=sr_1_1?ie=UTF8&qid=1319571691&sr=8-1
http://www.technologyreview.com/files/1158/1106_Q-A.tif.jpg

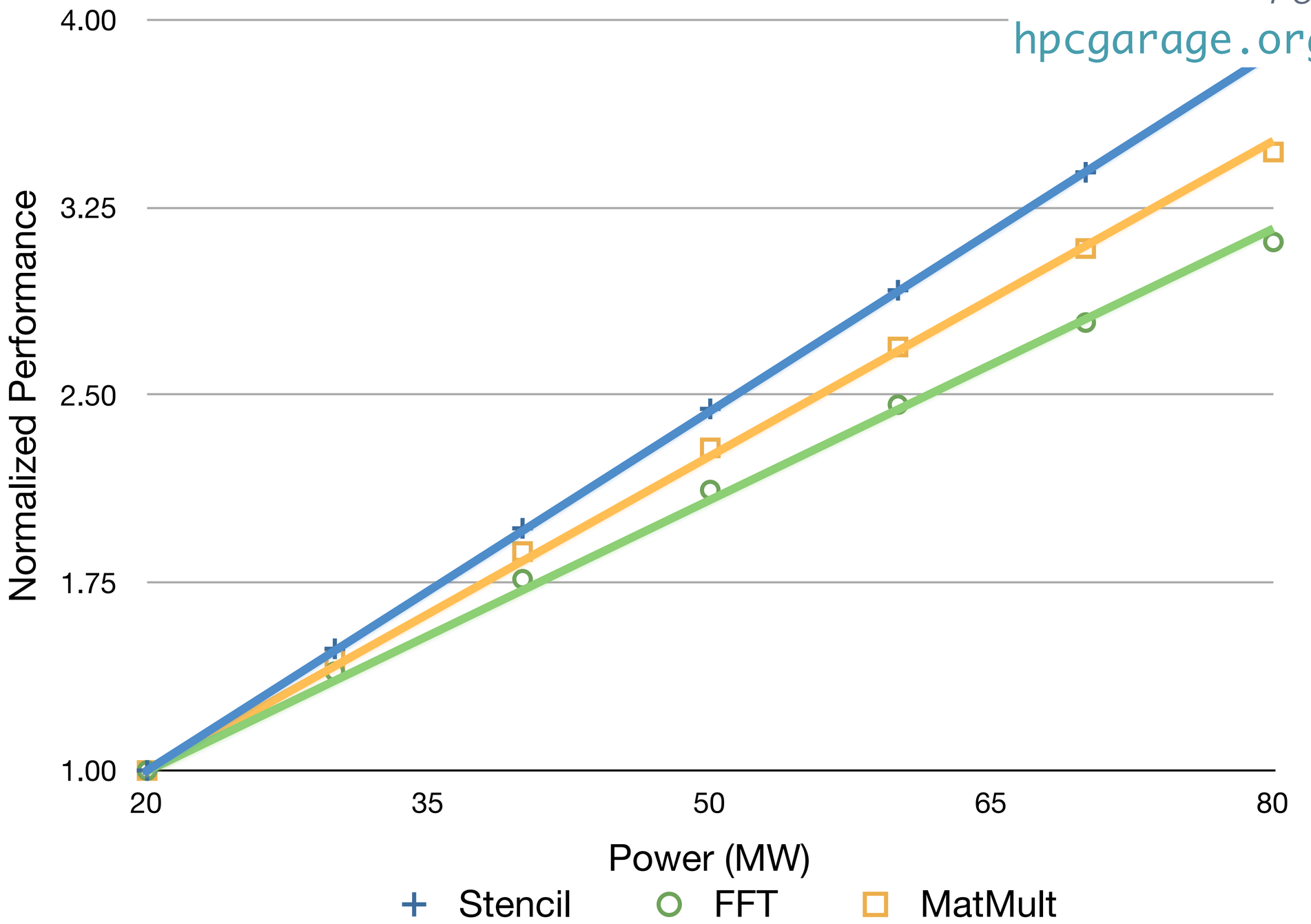
ACM Doctoral Dissertation Award Winner (1985)

Follow along:
hpcgarage.org/ppam13

http://www.amazon.com/Connection-Machine-Artificial-Intelligence/dp/0262580977/ref=sr_1_1?ie=UTF8&qid=1319571691&sr=8-1
http://www.technologyreview.com/files/1158/1106_Q-A.tif.jpg

[*Preview*] Redressing Hillis? Algorithmic power scaling

Follow along:
hpcgarage.org/ppam13



Aparna Chandramowlishwaran
[Now R.S. @ MIT]
Fast multipole method

Jee Whan Choi
Autotuning for power and energy

Kent Czechowski
Co-design

Marat Dukhan
Math libraries
& machine learning

2010 Gordon Bell Prize (+ G. Biros)
2010 IPDPS Best Paper (+ K. Knobe, Intel CnC lead)
2012 SIAM Data Mining Best Paper (D. Lee [GE Research] + A. Gray)

See our recent 2013 IPDPS papers, posted at: hpcgarage.org/ppam13

* Jee Choi — "A roofline model of energy."
* Kent Czechowski —"A theoretical framework for algorithm-architecture co-design."

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

Follow along:
hpcgarage.org/ppam13

Rebutting Hillis, Part 1:

First principles:

Rooflines in **time**, arch lines in **energy**, and "**power** lines"

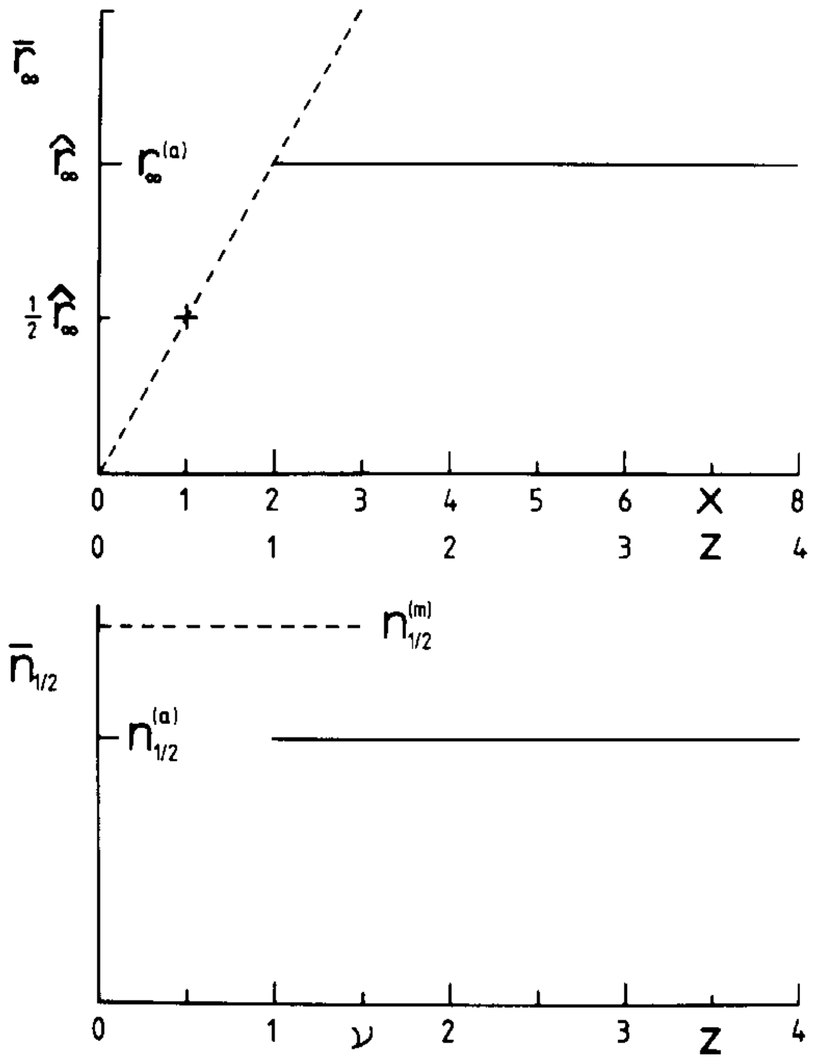
Jee

Marat

Energy and power analogues of the time-based "roofline model" of Hockney & Curington (1989) and Williams et al. (2009)

See also Choi & Vuduc in **IPDPS'13** [http://vuduc.org/pubs/choi2013-archline-ipdps.pdf] + "extended remix" technical report [http://hdl.handle.net/1853/45737]

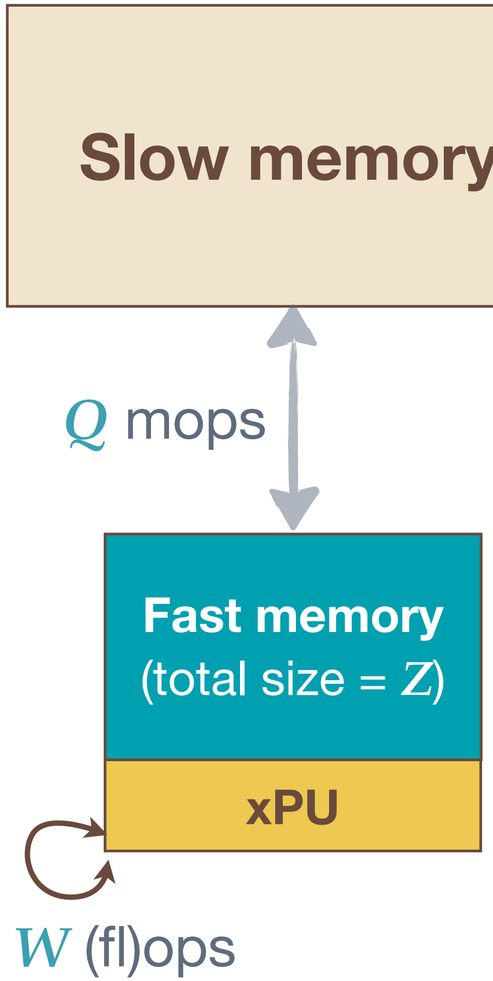
Follow along:
hpcgarage.org/ppam13



R.W. Hockney and I.J. Curington (1989).
"$f_{½}$: A parameter to characterize memory and communication bottlenecks."
Parallel Computing, **10**(3), 277–286.
doi: 10.1016/0167-8191(89)90100-2

Fig. 2. The variation of $(\bar{r}_\infty, \bar{n}_{1/2})$ with f for the case of a combined memory I/O and arithmetic pipeline, when the I/O and arithmetic can be overlapped. Full lines: parameters when arithmetic dominates, equations (14b); dotted lines: parameters when I/O dominates, equations (13b). Notation as Fig. 1, and $\nu = 1.5$.

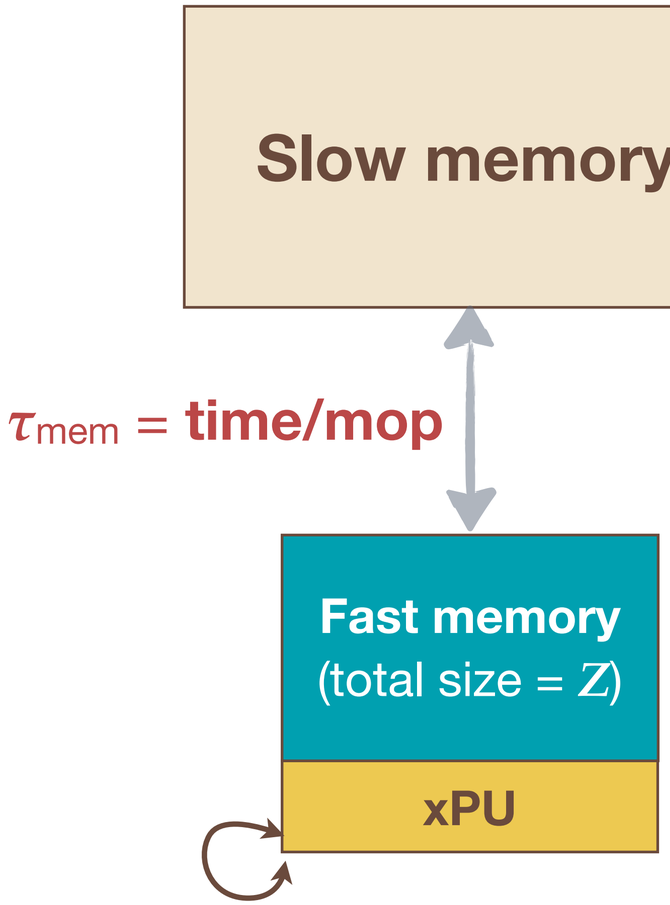
Follow along:
hpcgarage.org/ppam13



$$
\begin{aligned}
W &\equiv \text{\# (fl)ops} \\
Q &\equiv \text{\# mem. ops (mops)} \\
&= Q(Z)
\end{aligned}
$$

von Neumann bottleneck

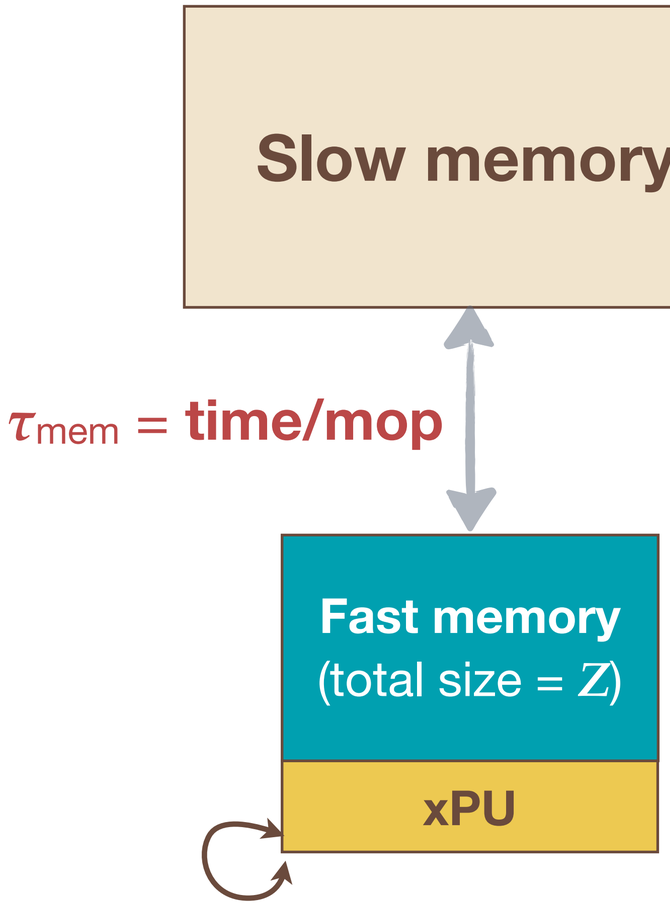
Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



$$T = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right)$$

von Neumann bottleneck

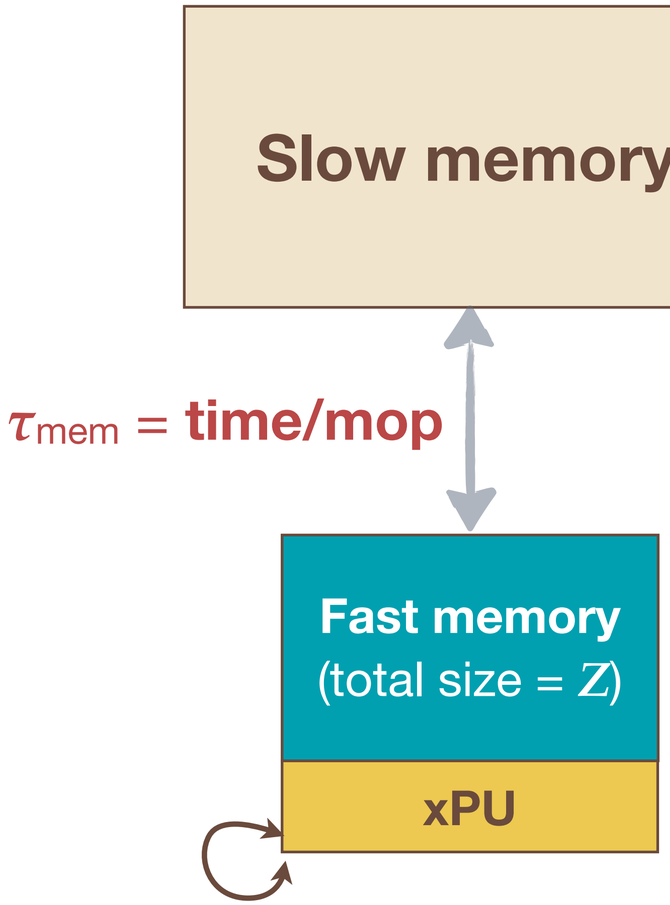
Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\,\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right)
\end{aligned}
$$

von Neumann bottleneck

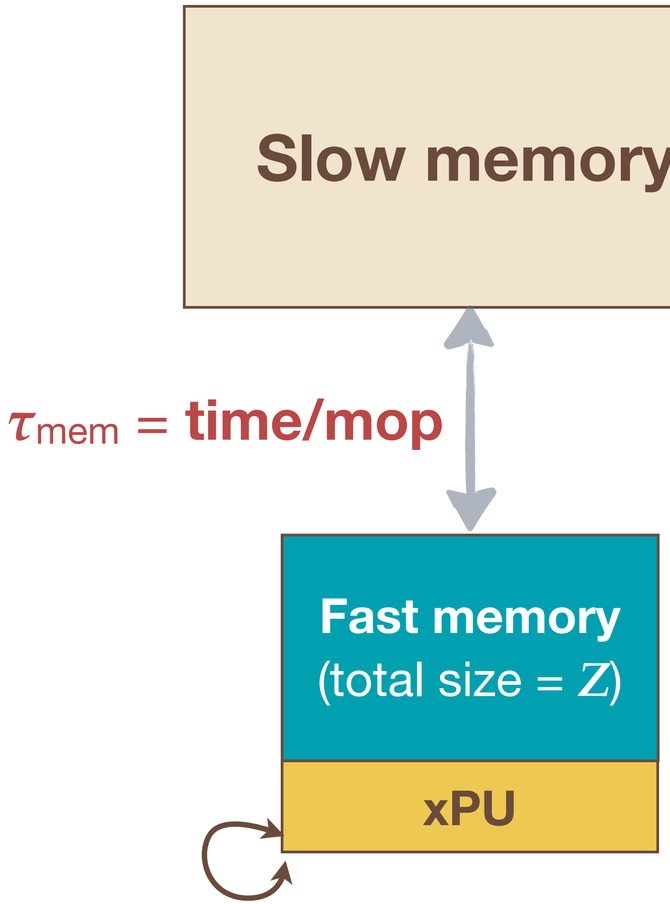
Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

von Neumann bottleneck

Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …

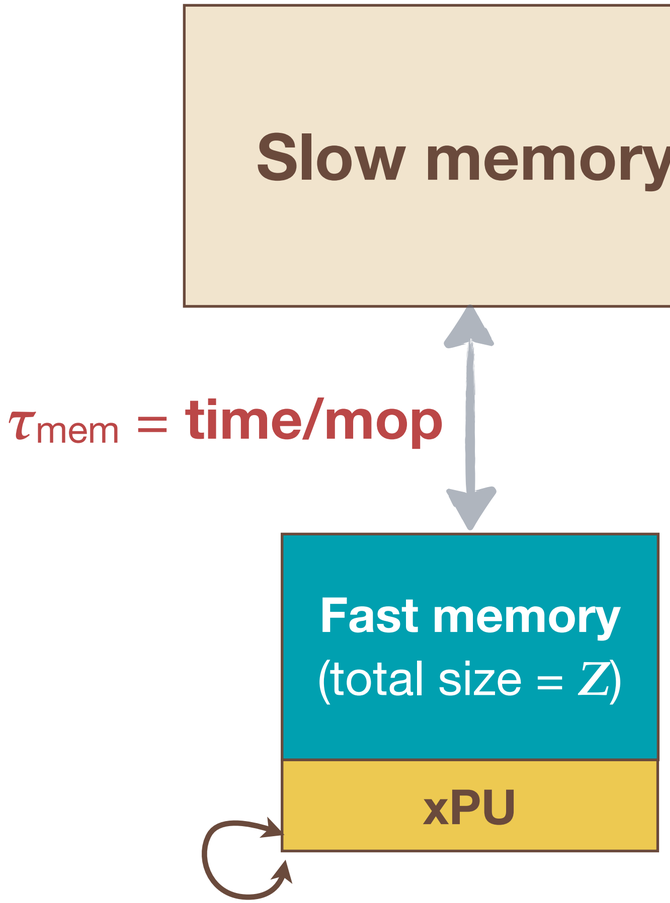


$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

Minimum time

von Neumann bottleneck

Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



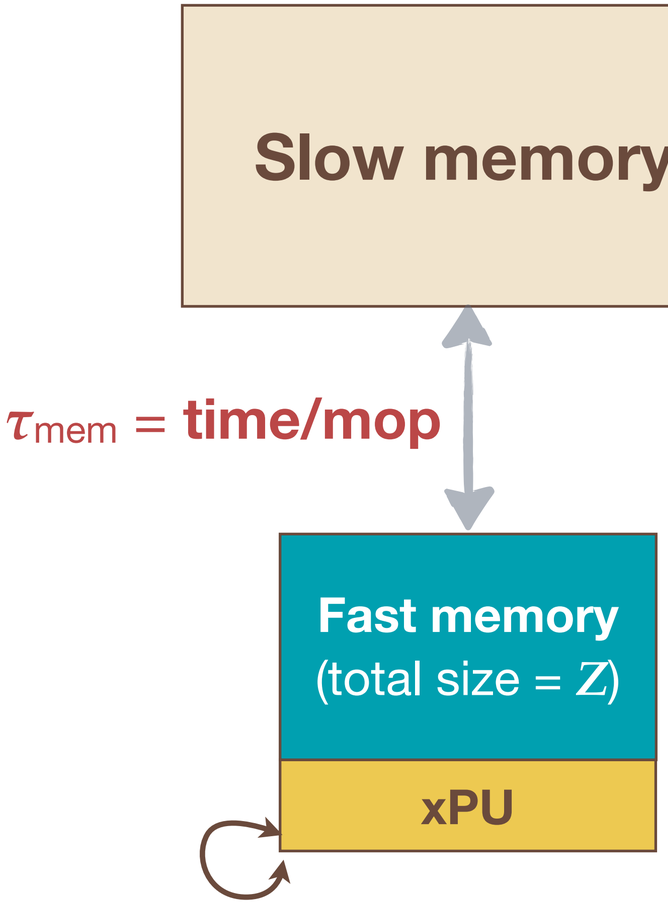
$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

Minimum time

Intensity
(flop : mop)

von Neumann bottleneck

Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

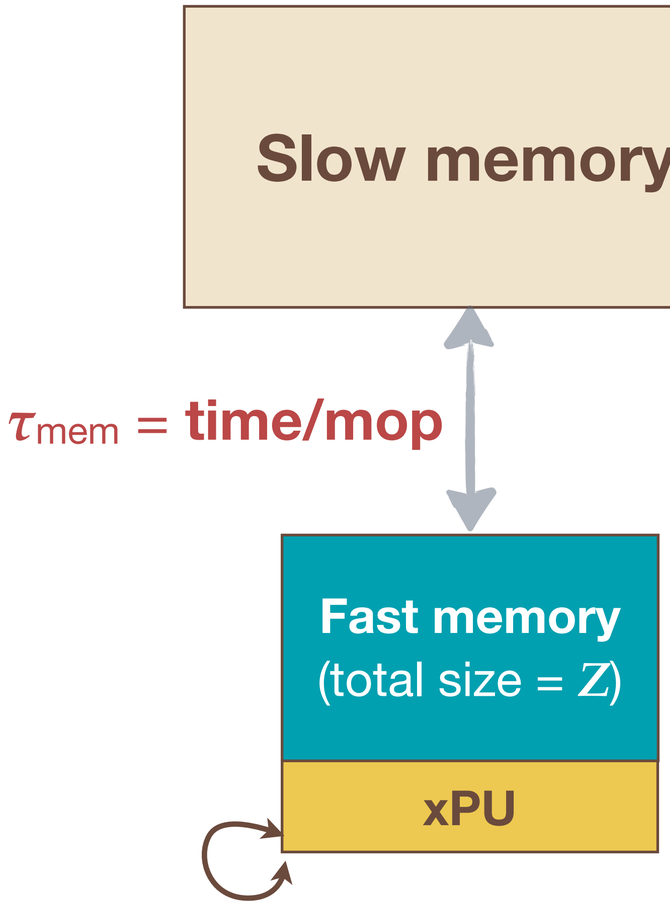
Minimum time

Intensity
(flop : mop)

Balance
(flop : mop)

von Neumann bottleneck

Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

Intensity (flop : mop)

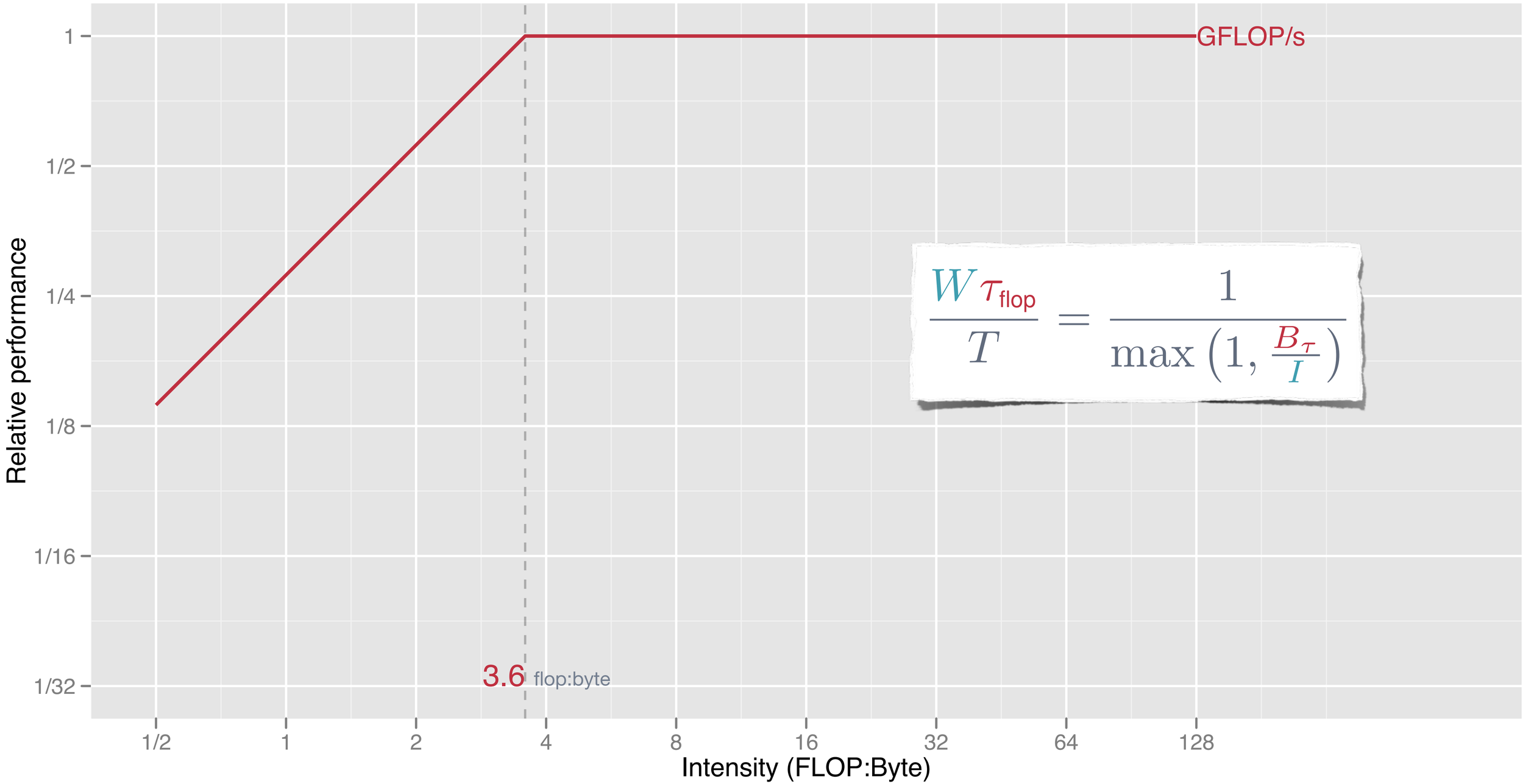
Balance (flop : mop)

von Neumann bottleneck

Consider: $\dfrac{W\tau_{\text{flop}}}{T}$

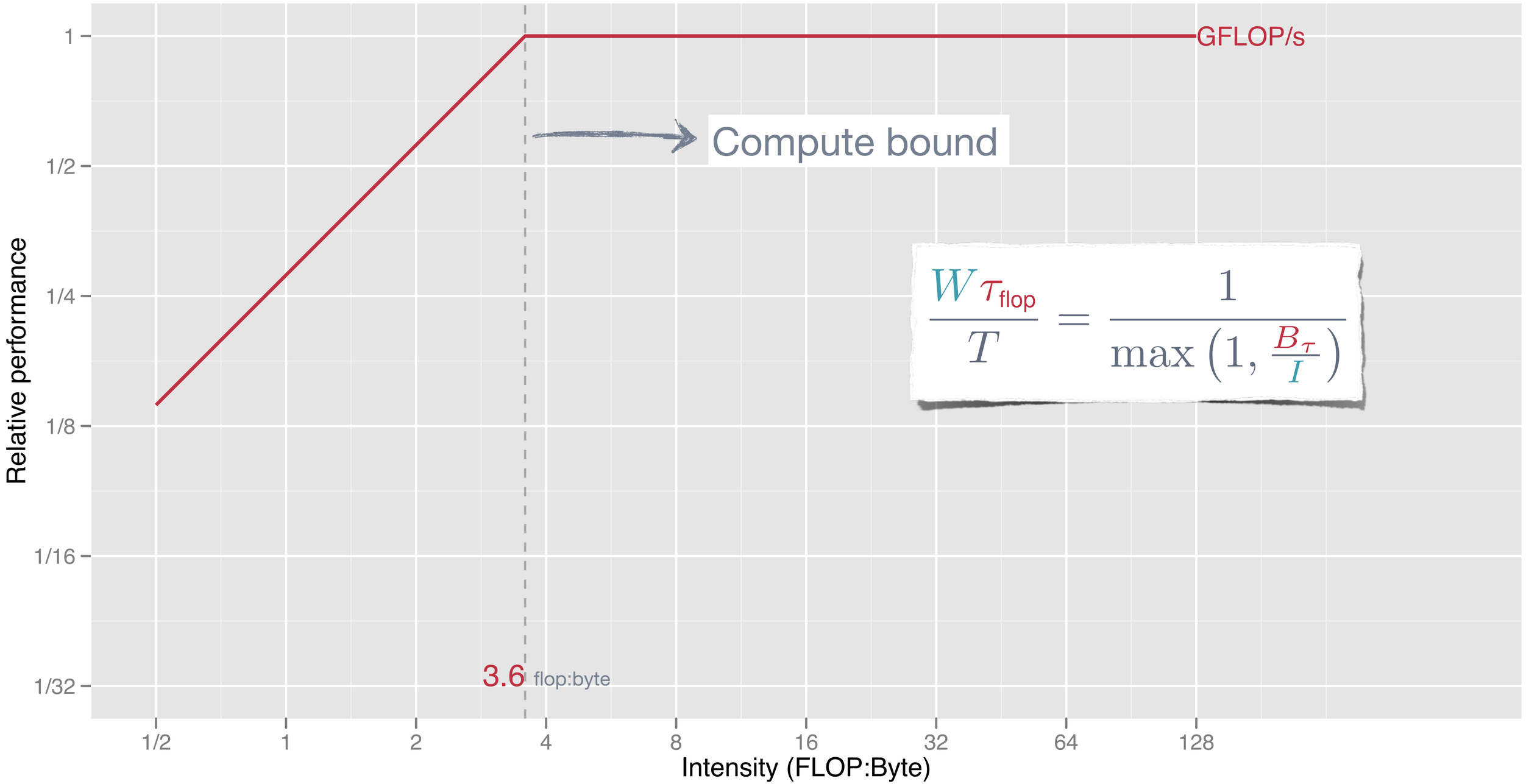
Balance analysis — Kung (1986); Hockney & Curington (1989); Blelloch (1994); McCalpin (1995); Williams et al. (2009); Czechowski et al. (2011); …

“Roofline” — Williams et al. (*Comm. ACM* ’09)



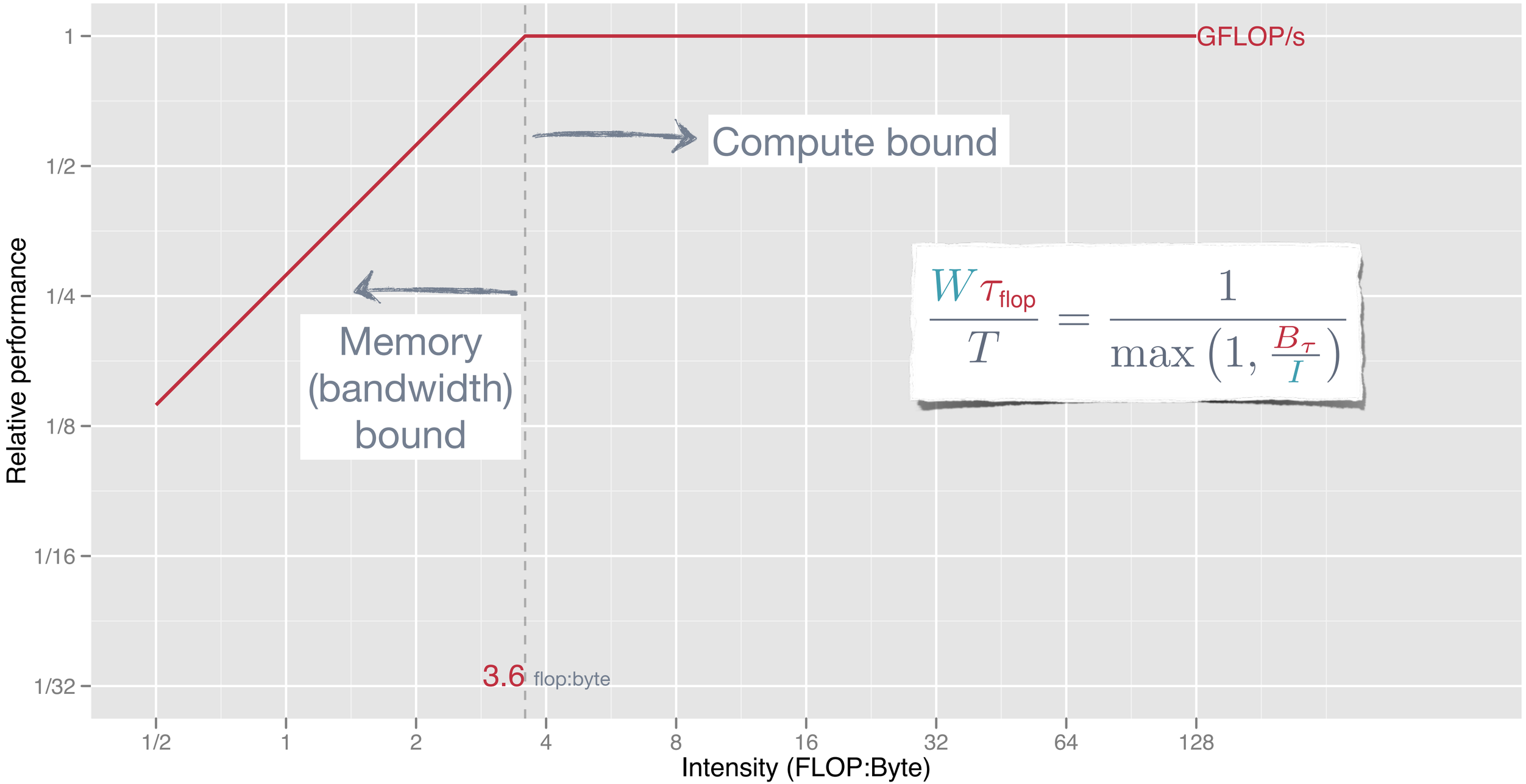
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

"Roofline" — Williams et al. (*Comm. ACM* '09)

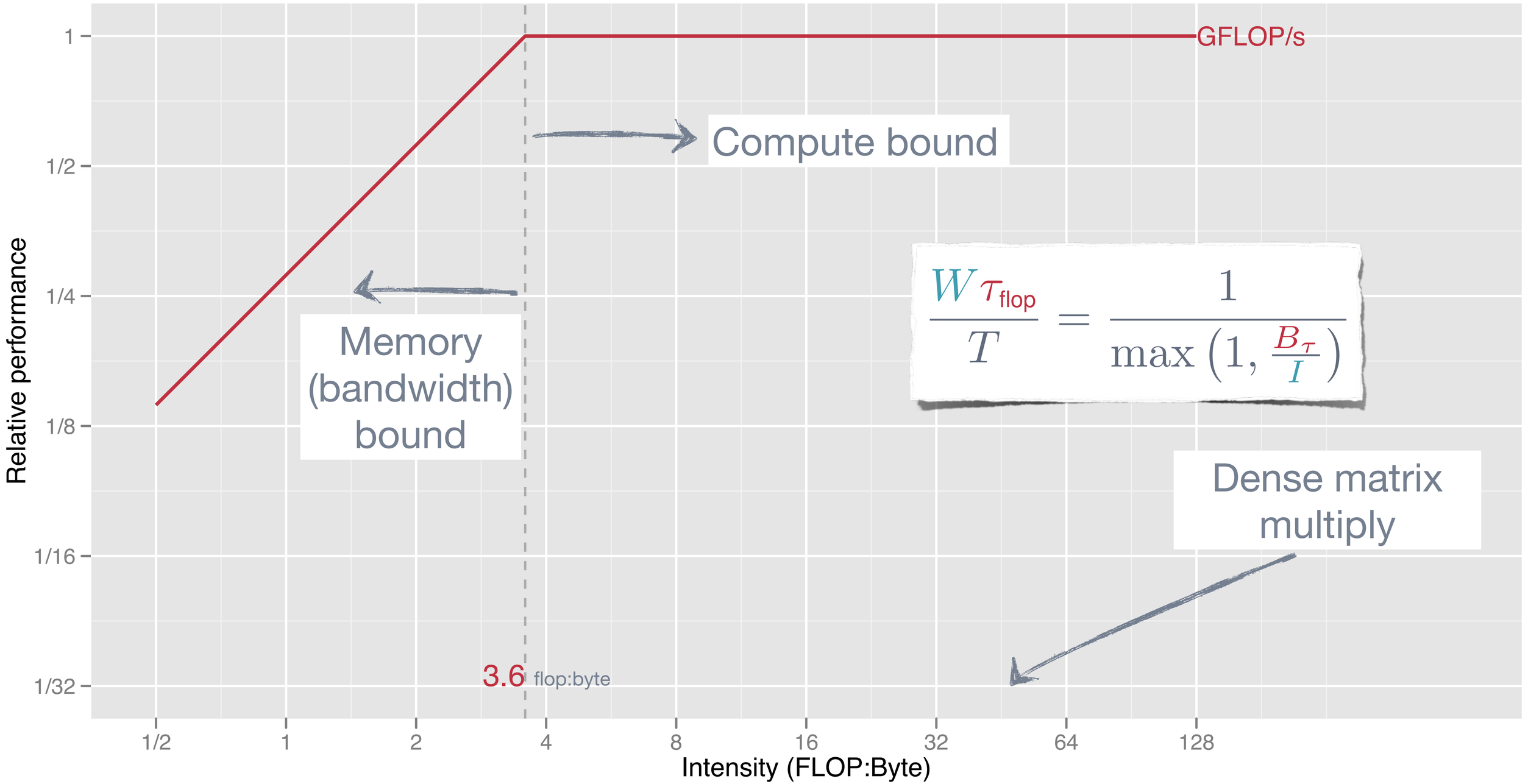


Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

"Roofline" — Williams et al. (*Comm. ACM* '09)

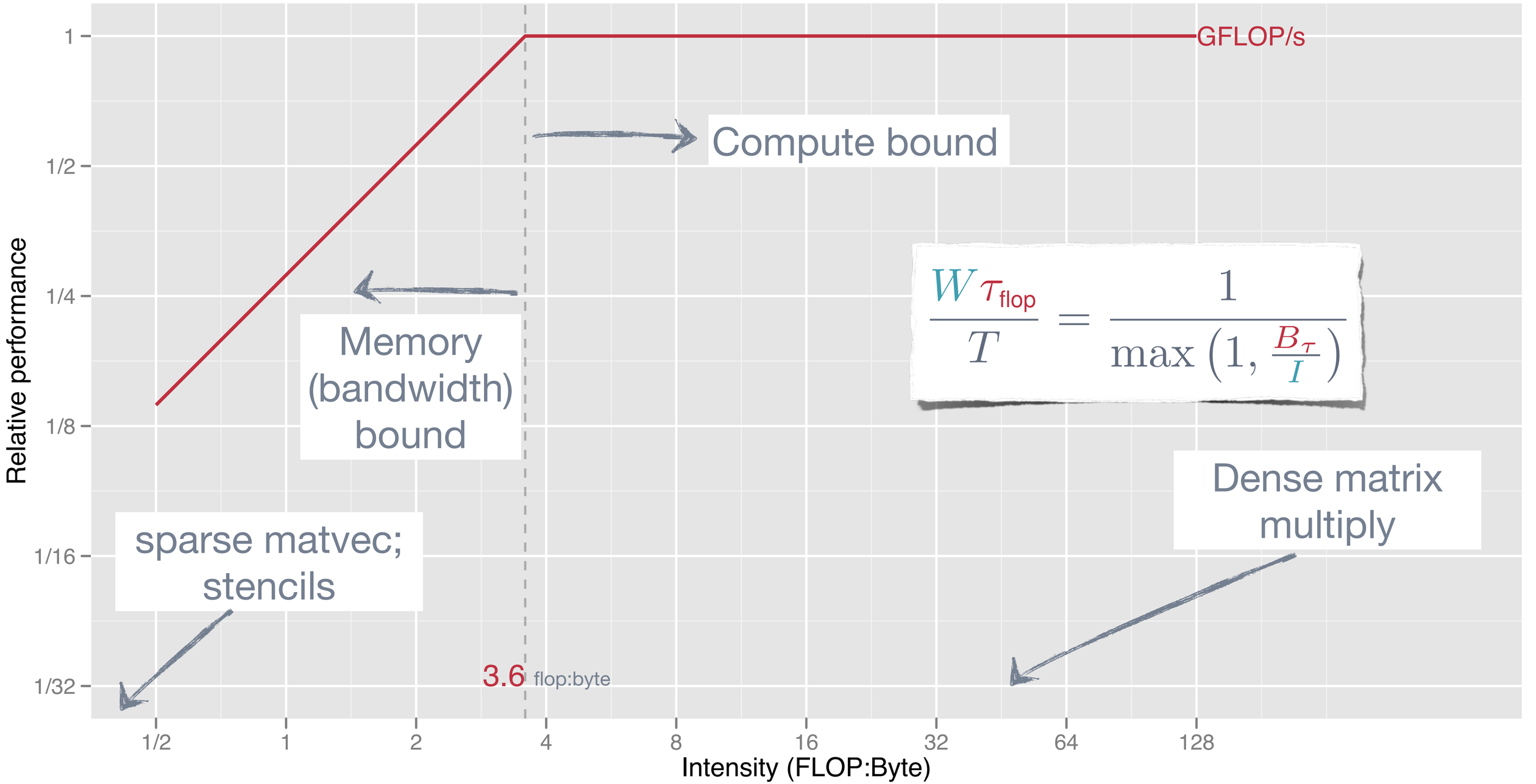


Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



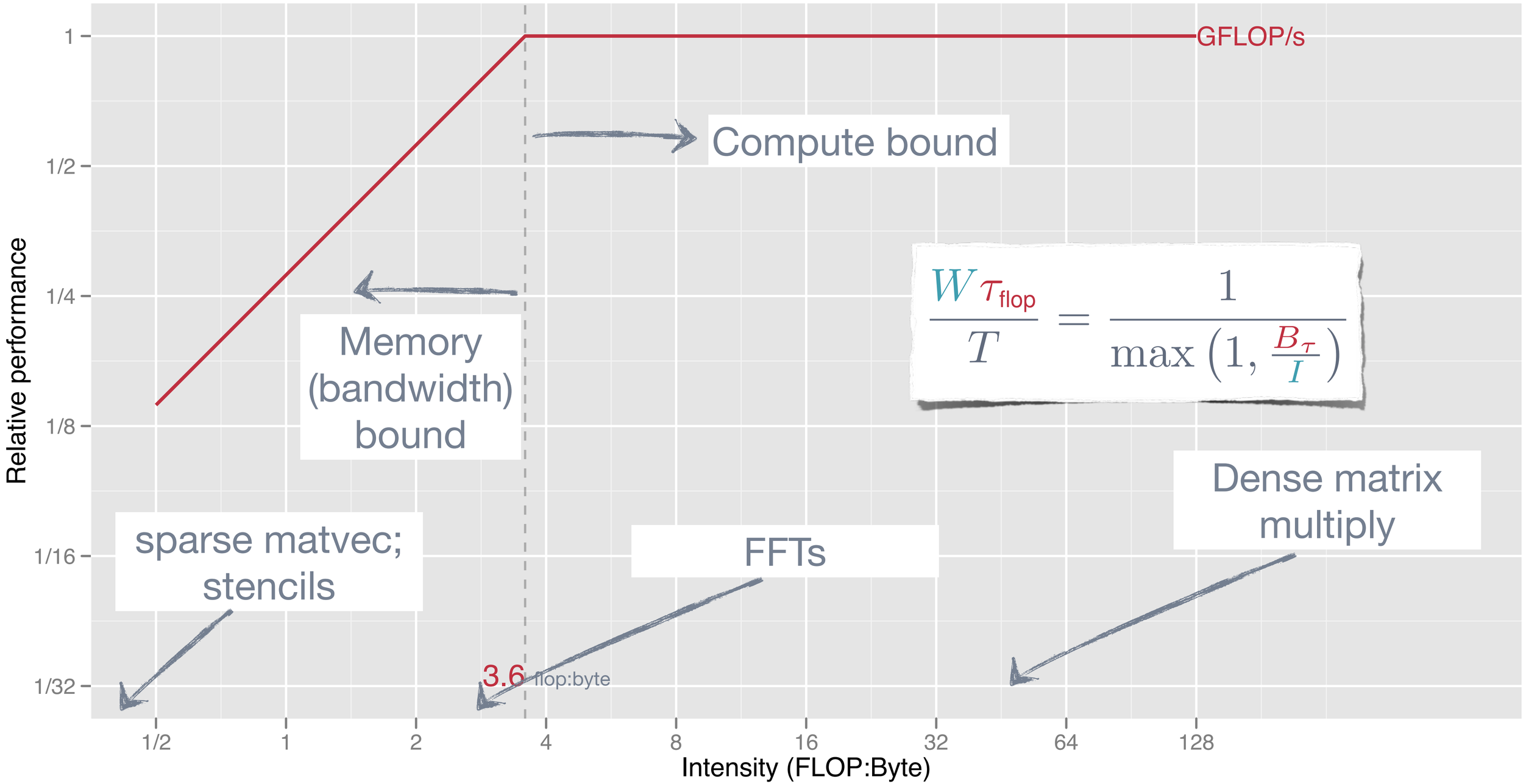
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

"Roofline" — Williams et al. (*Comm. ACM* '09)

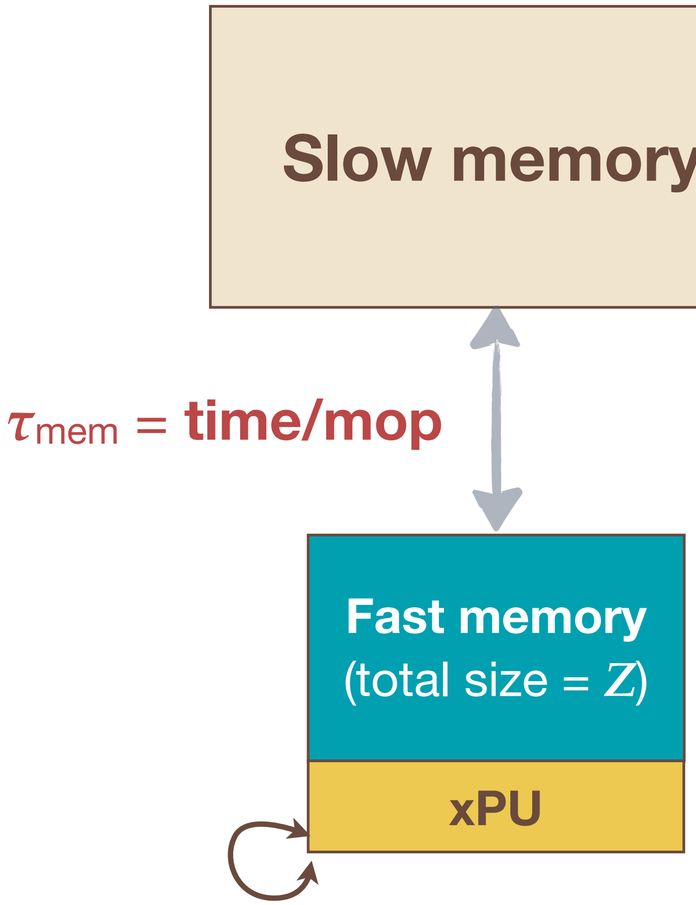


Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

"Roofline" — Williams et al. (*Comm. ACM* '09)

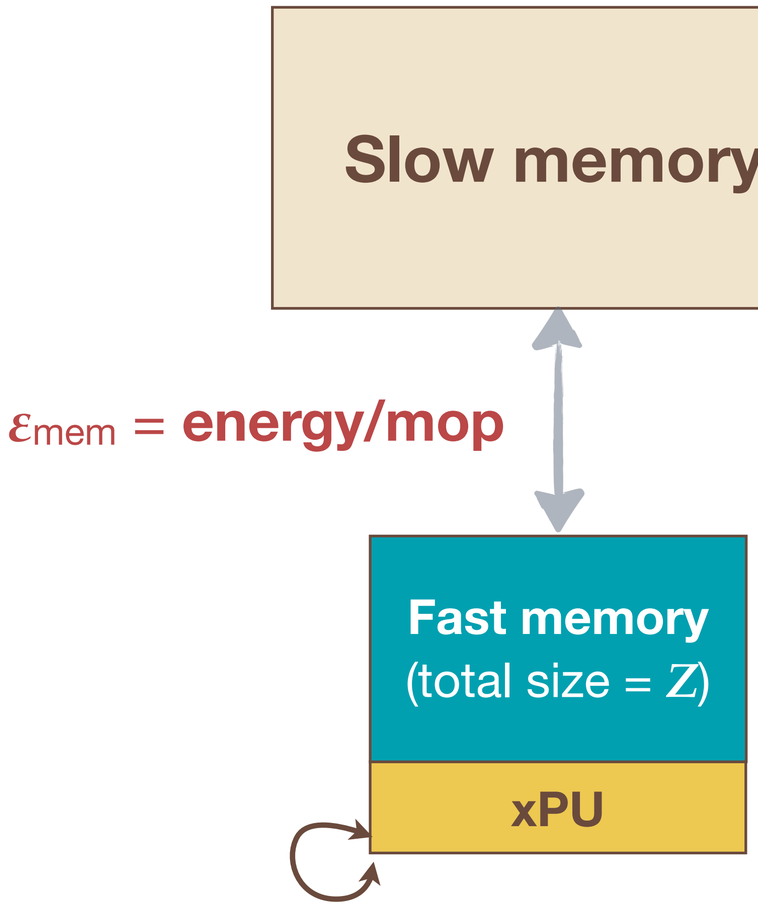


Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

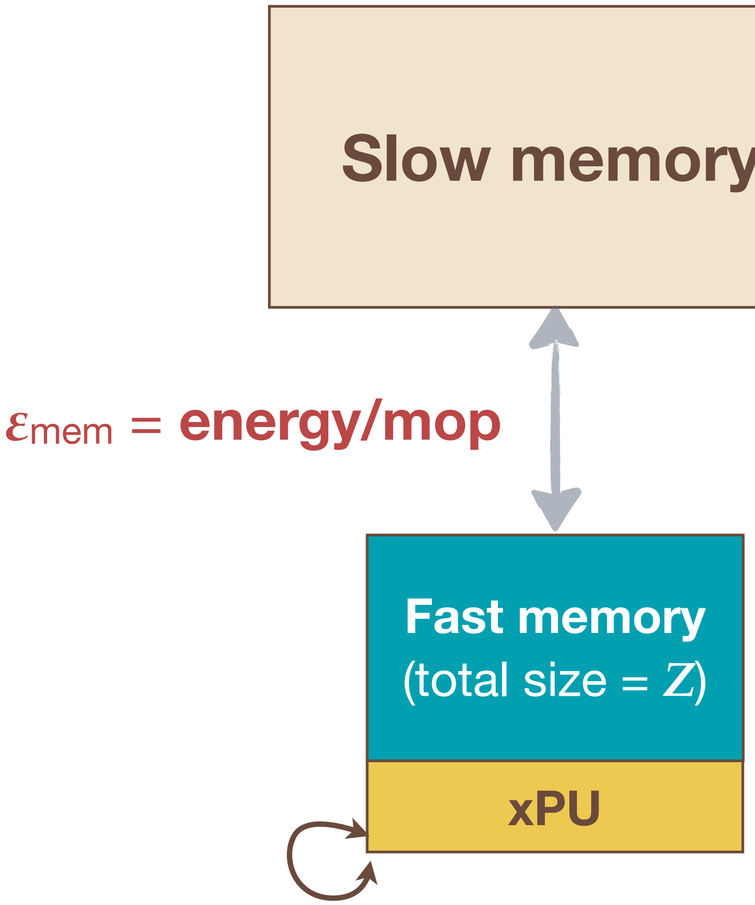
An energy analogue



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

$$
\begin{aligned}
E &= W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}} \\
&= W\epsilon_{\text{flop}}\left(1 + \frac{B_\epsilon}{I}\right)
\end{aligned}
$$

An energy analogue

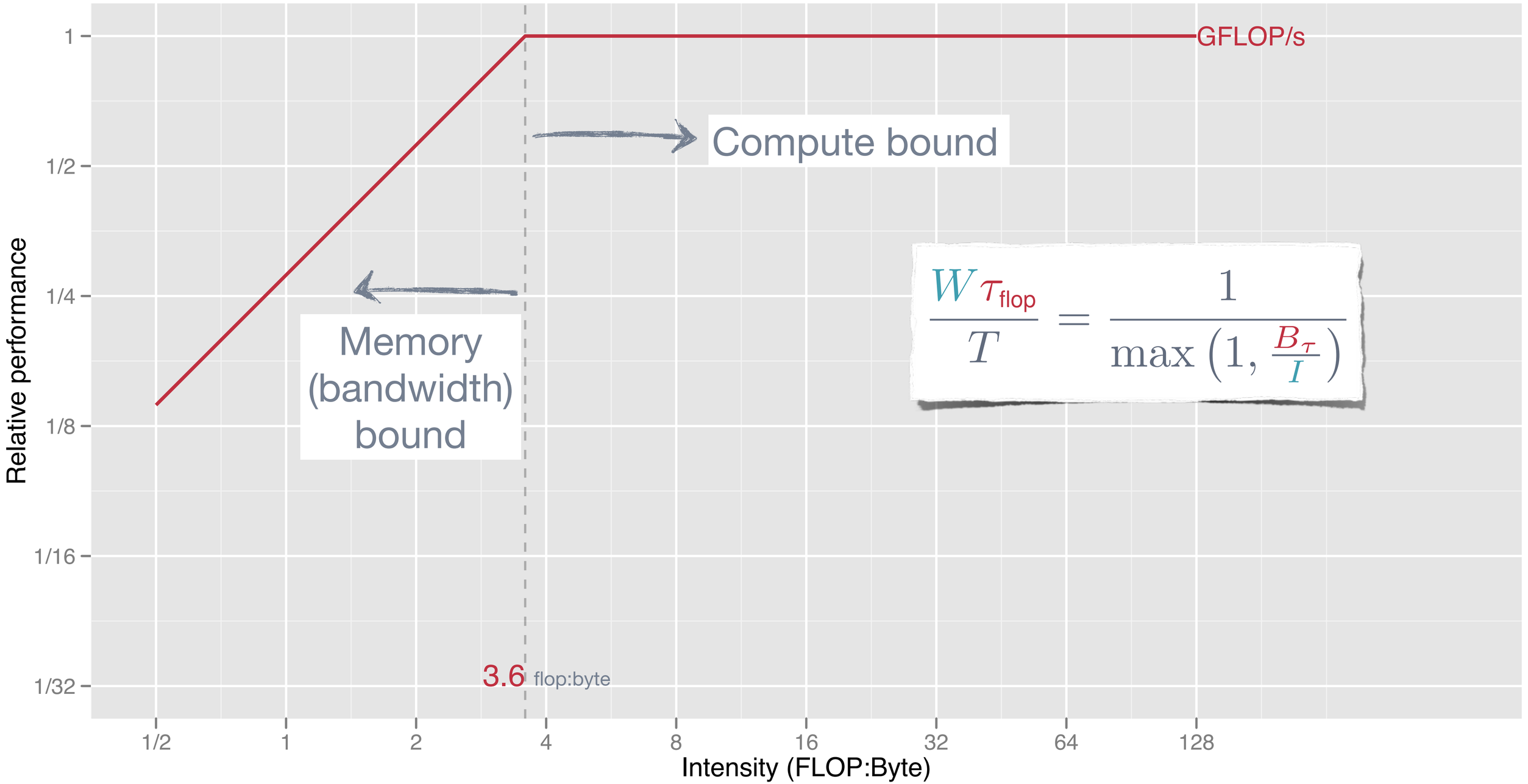


$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

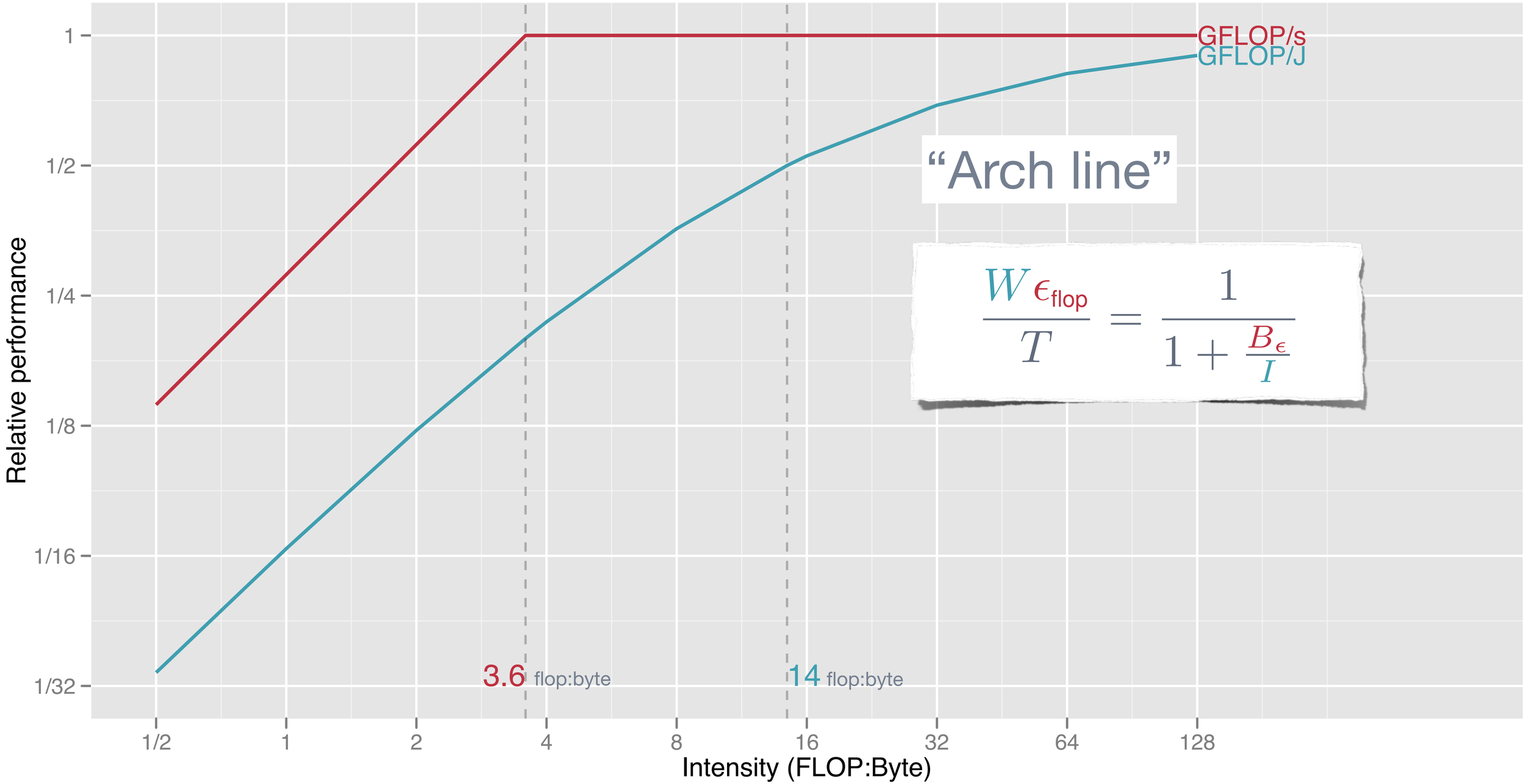
$$
\begin{aligned}
E &= W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}} \\
&= W\epsilon_{\text{flop}}\left(1 + \frac{B_\epsilon}{I}\right)
\end{aligned}
$$

An energy analogue

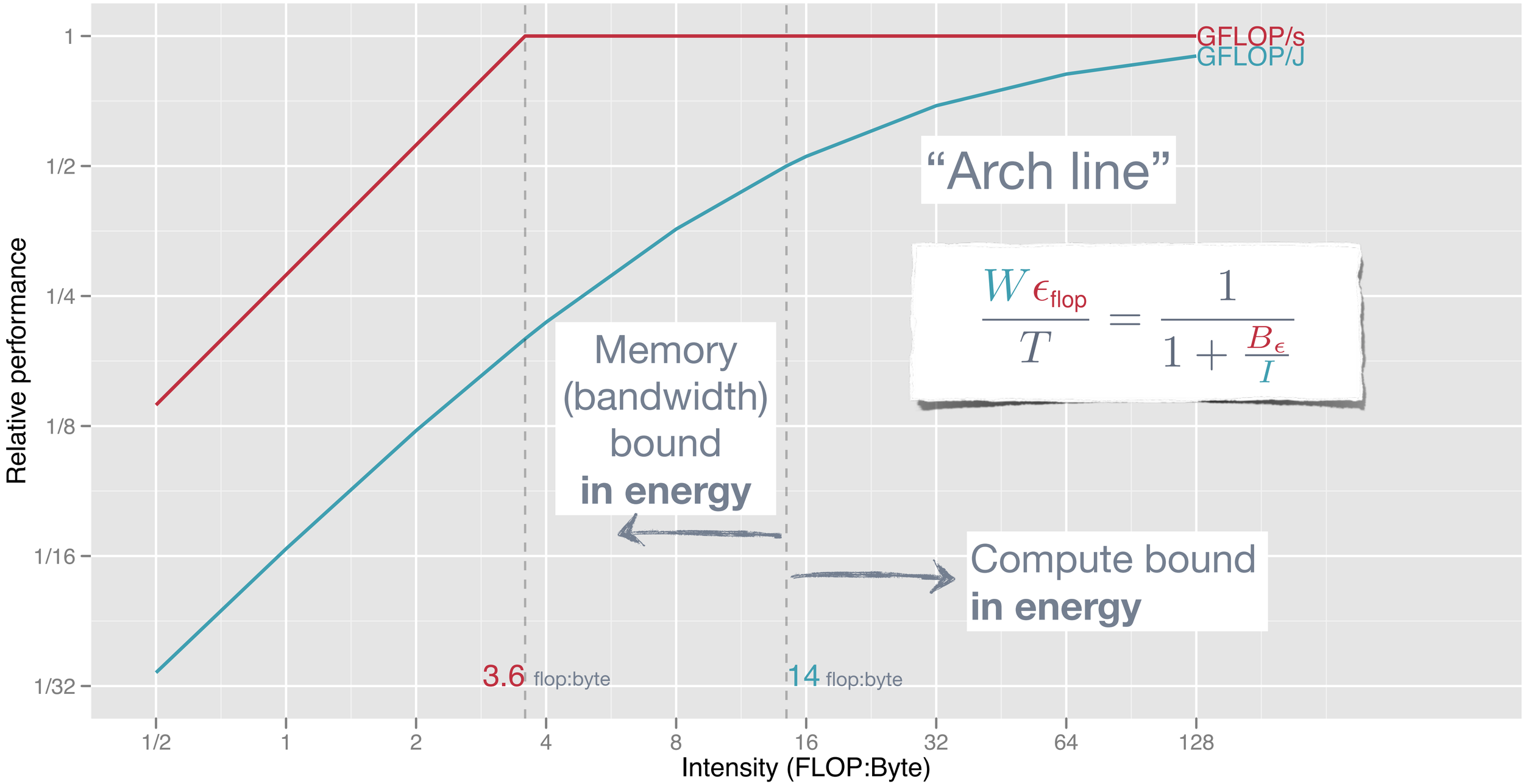
Consider: $\dfrac{W\tau_{\text{flop}}}{T}$ and $\dfrac{W\epsilon_{\text{flop}}}{E}$



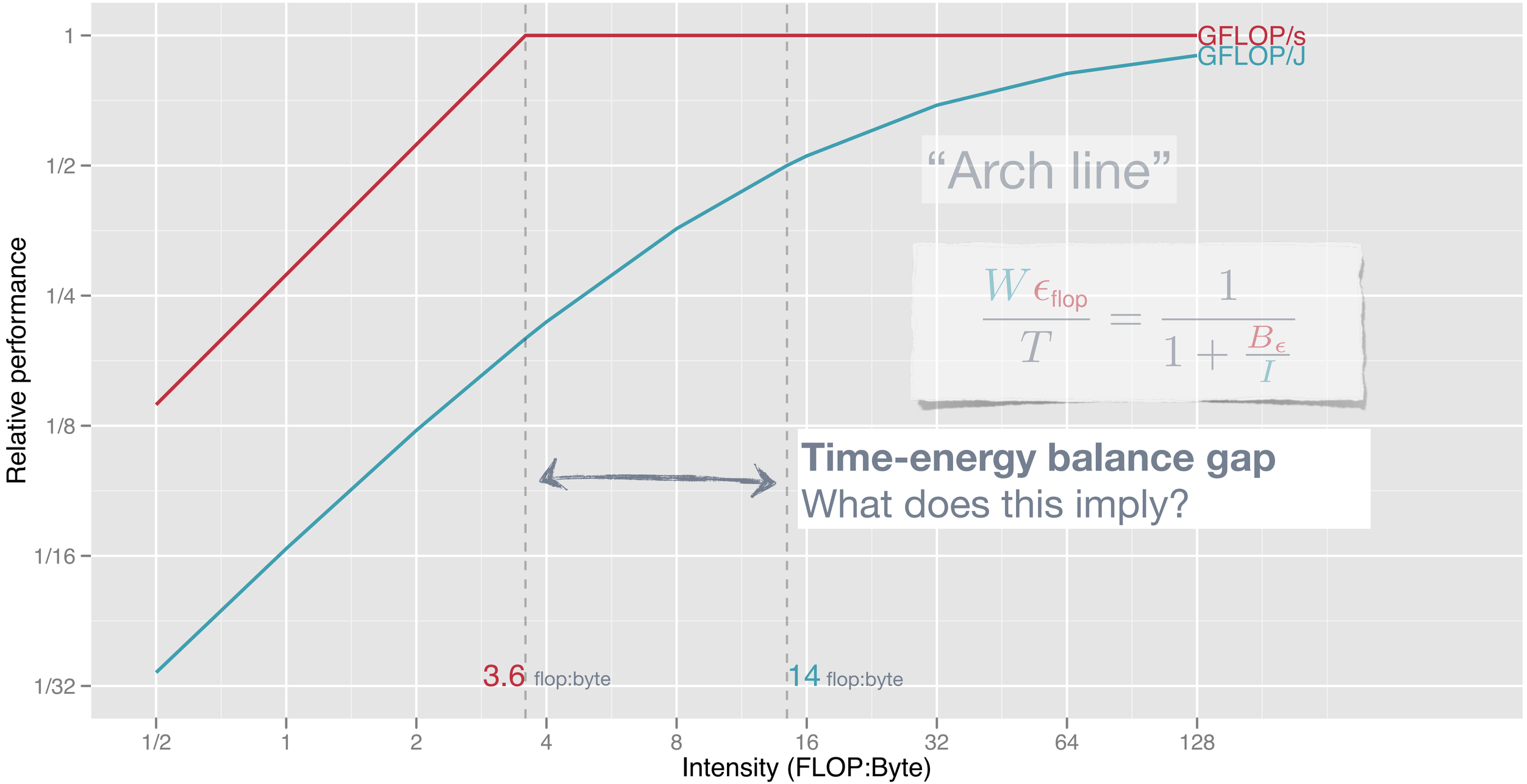
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



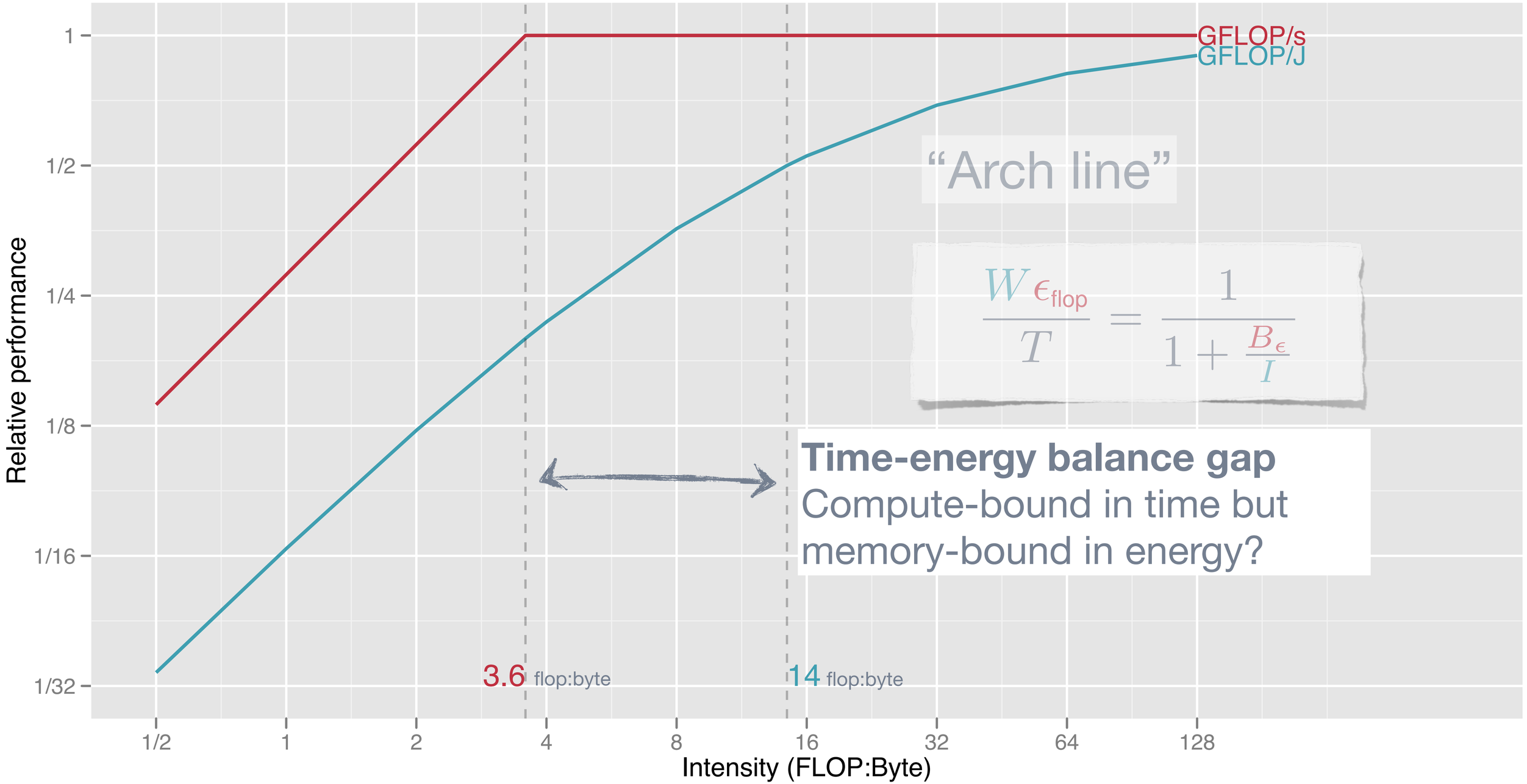
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



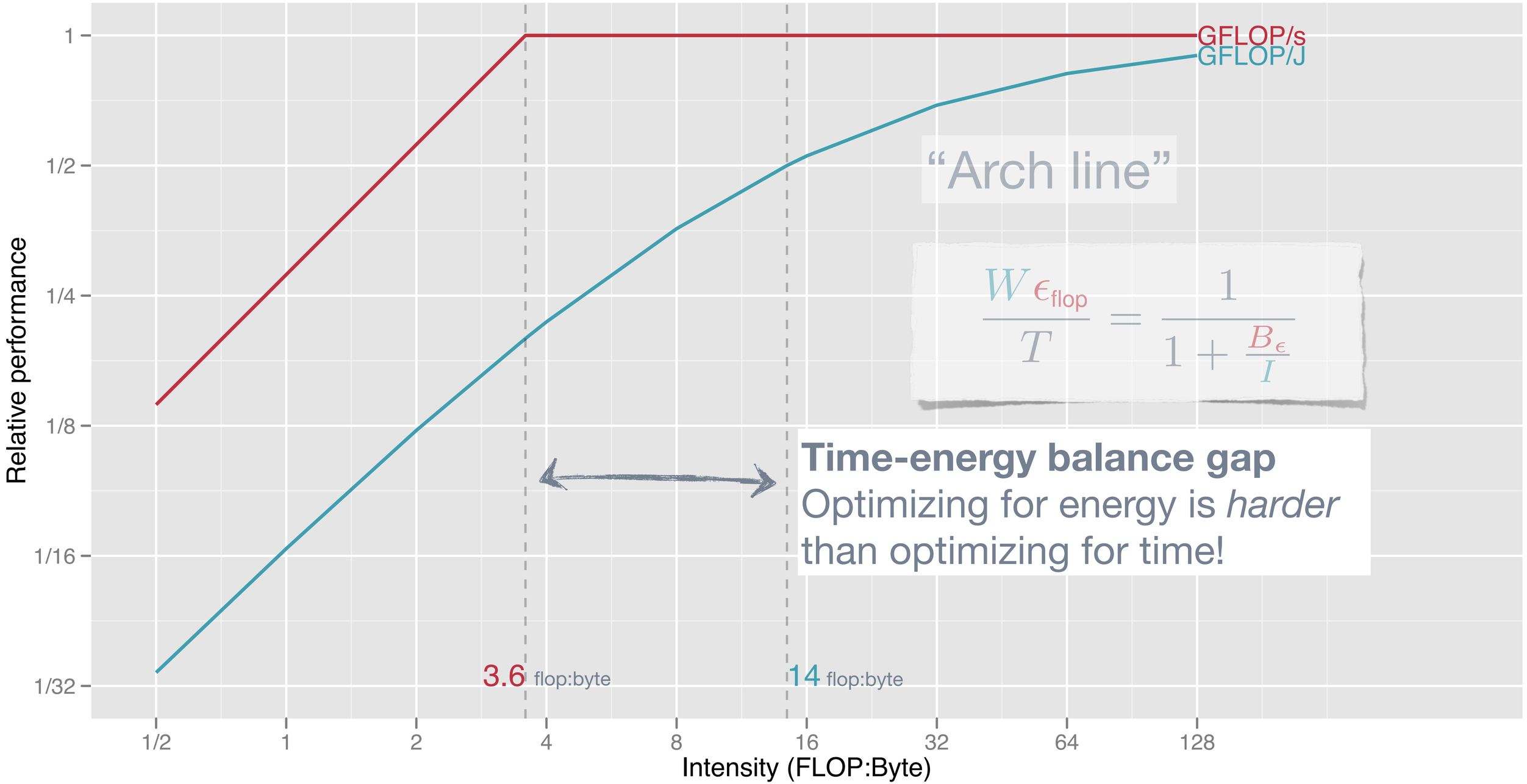
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



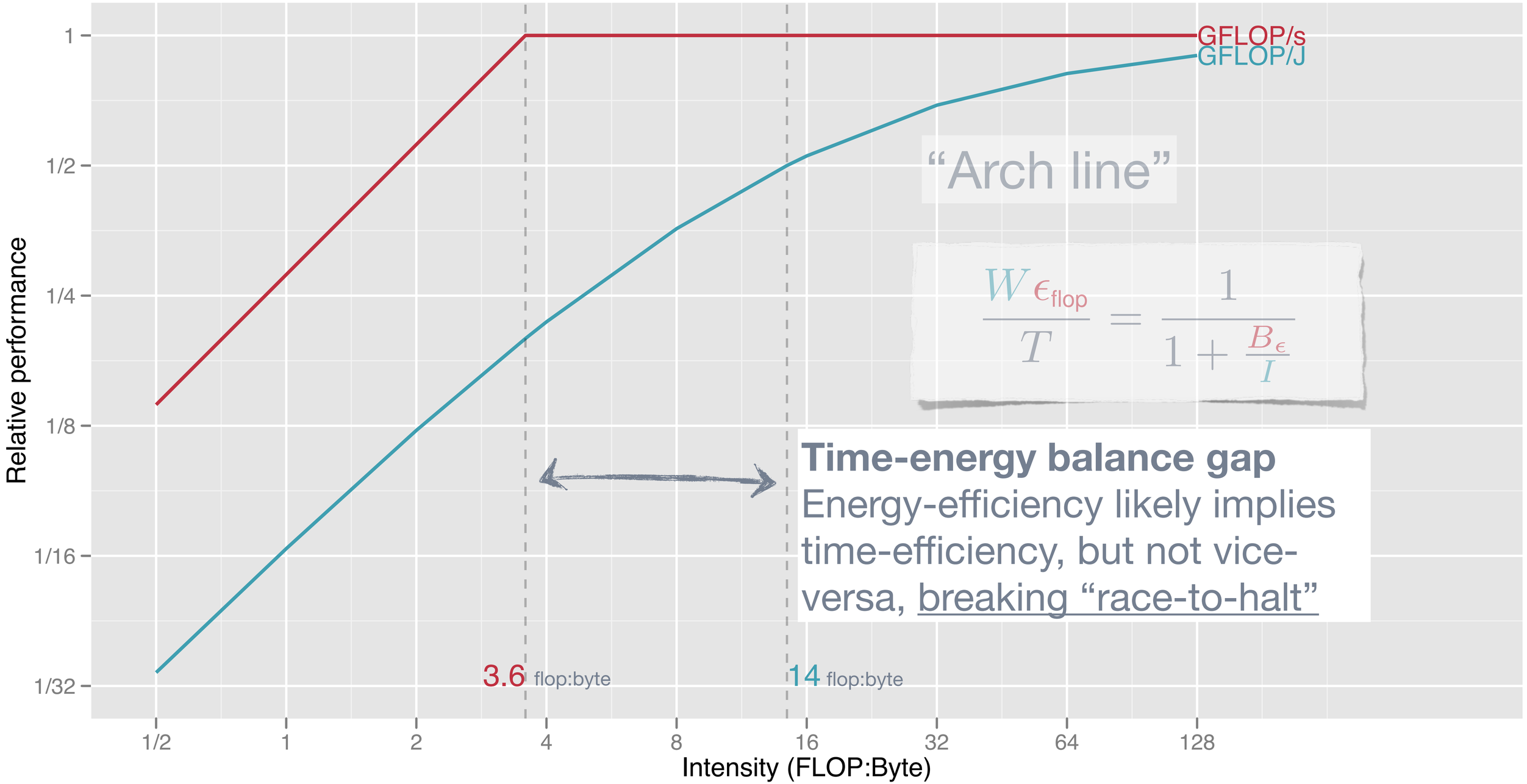
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



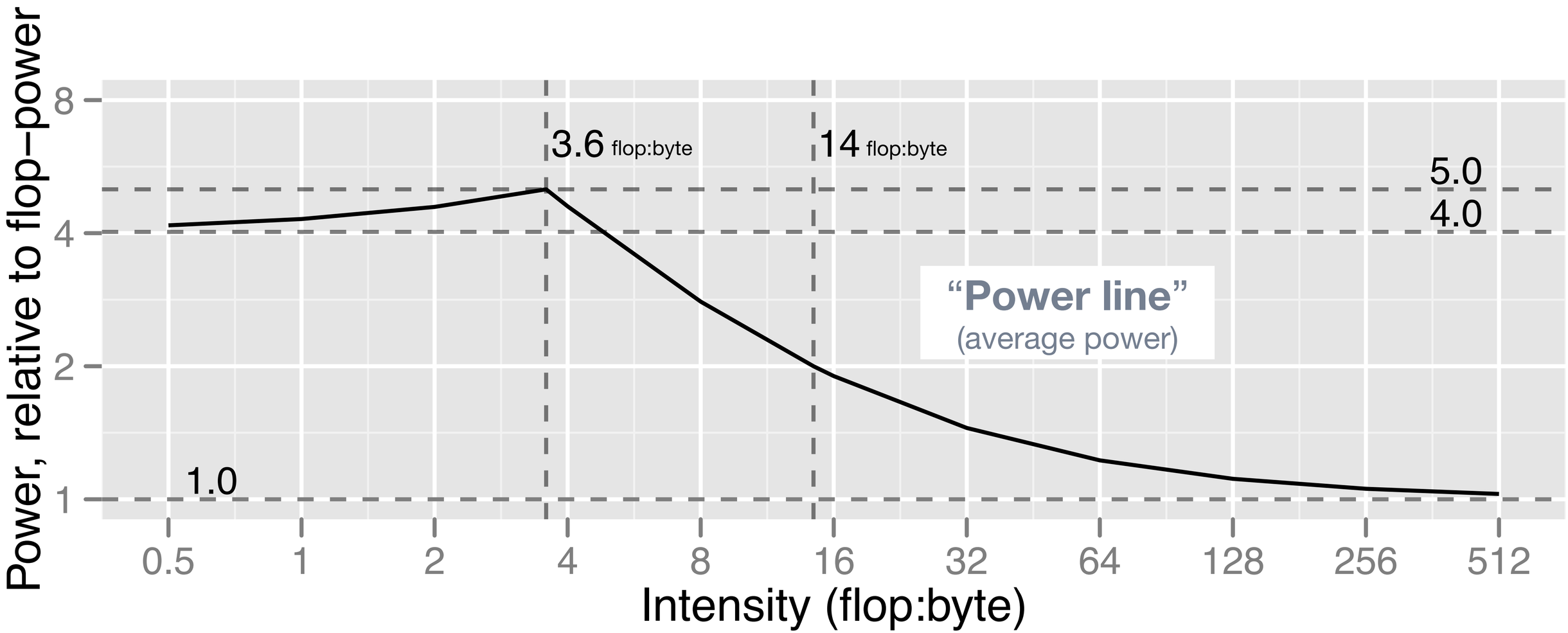
Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)



Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

Idea: Work-communication trade-offs

$$\text{Algorithm 1} = (W, Q) \quad \text{versus} \quad \text{Algorithm 2} = (fW, \frac{Q}{m})$$

$$I \equiv \frac{W}{Q}$$

Idea: Work-communication trade-offs

$$\text{Algorithm 1} = (W, Q) \quad \text{versus} \quad \text{Algorithm 2} = (fW, \frac{Q}{m})$$

$$I \equiv \frac{W}{Q}$$

$$\text{Speedup } \Delta T = \frac{T_{1,1}}{T_{f,m}}$$

$$\text{“Greenup” } \Delta E = \frac{E_{1,1}}{E_{f,m}}$$

Idea: Work-communication trade-offs

$$\text{Algorithm 1} = (W, Q) \quad \text{versus} \quad \text{Algorithm 2} = (fW, \frac{Q}{m})$$

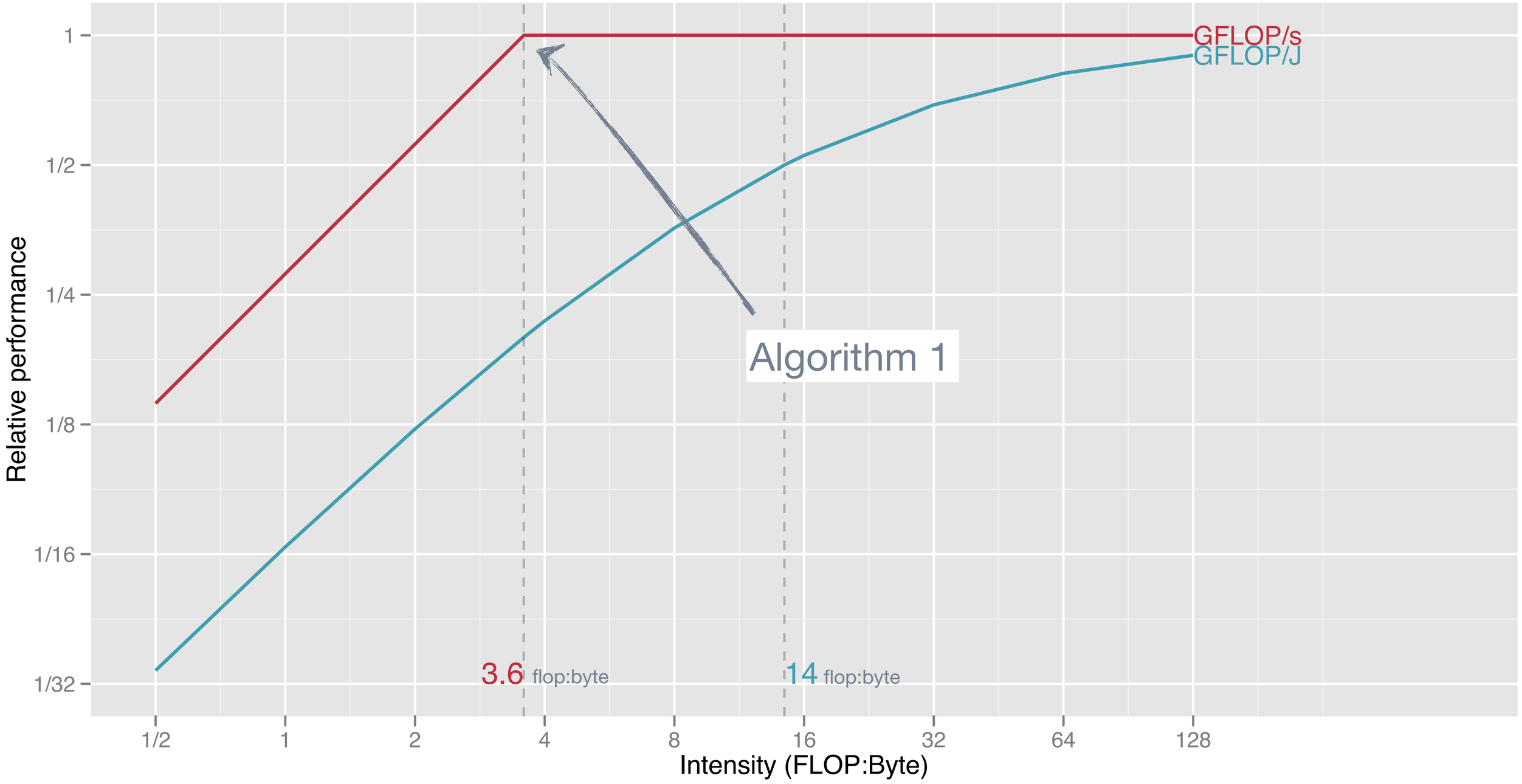
$$I \equiv \frac{W}{Q}$$

$$\text{Speedup } \Delta T = \frac{T_{1,1}}{T_{f,m}}$$

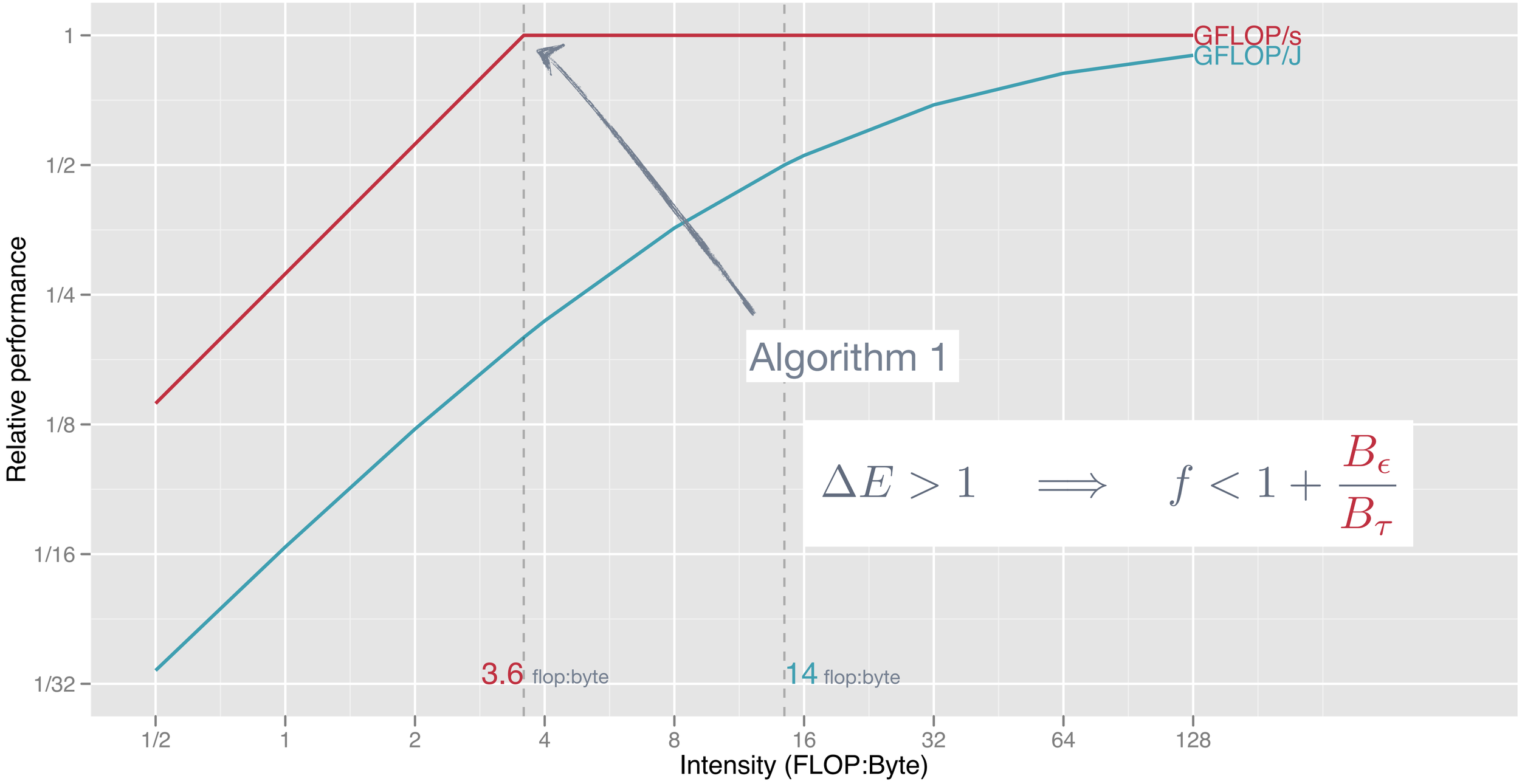
$$\text{“Greenup” } \Delta E = \frac{E_{1,1}}{E_{f,m}}$$

$$\Delta E > 1 \implies f < 1 + \frac{m-1}{m} \frac{B_\epsilon}{I}$$

A general “greenup” condition



Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

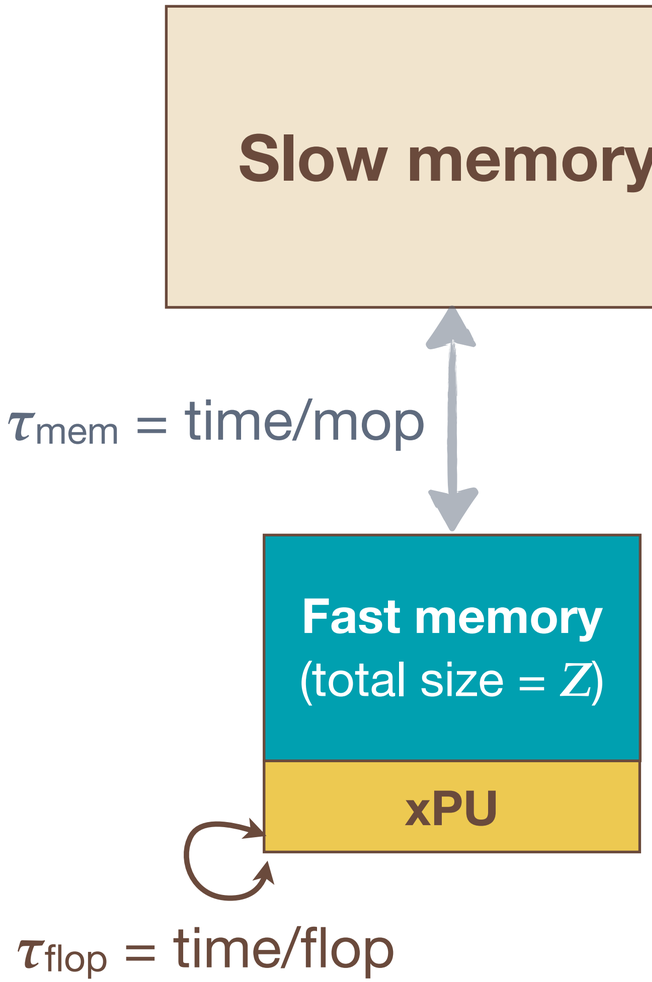


Balance estimates for a high-end NVIDIA Fermi in *double-precision*, according to Keckler et al. *IEEE Micro* (2011)

That was theory. What happens in practice?

⇒ Cannot ignore **constant power**.

Let’s add it to our model and measure.

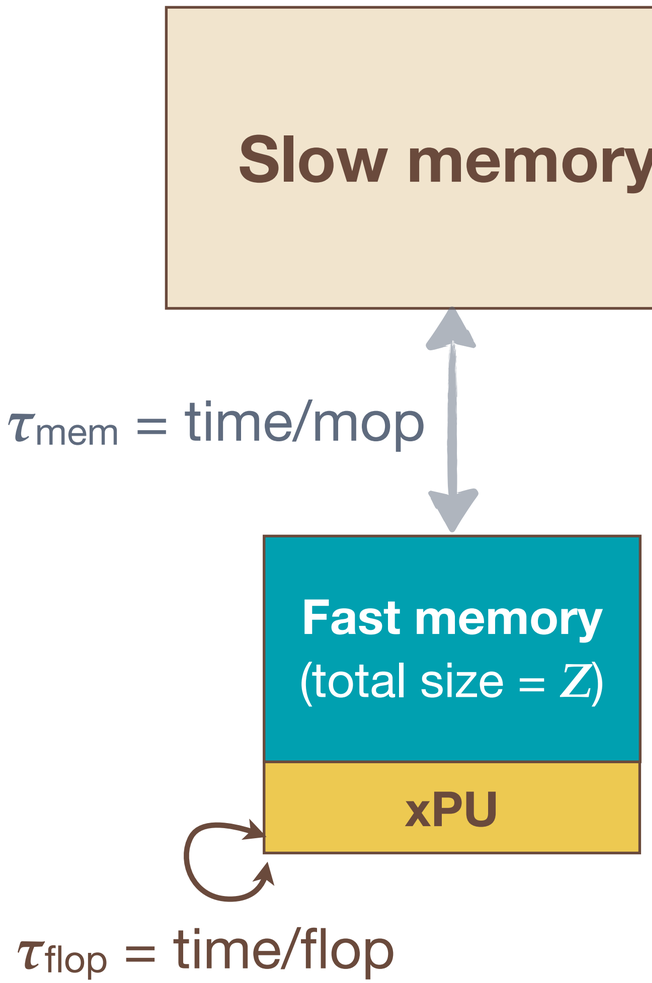


$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

$$
\begin{aligned}
E &= W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}} + T\pi_0 \\
&= W\epsilon_{\text{flop}}\left(1 + \frac{B_\epsilon}{I} + \frac{\pi_0}{\epsilon_{\text{flop}}}\frac{T}{W}\right)
\end{aligned}
$$

Constant power

Consider: $\dfrac{W\tau_{\text{flop}}}{T}$ and $\dfrac{W\epsilon_{\text{flop}}}{E}$



$$
\begin{aligned}
T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{Q}{W}\frac{\tau_{\text{mem}}}{\tau_{\text{flop}}}\right) \\
&= W\tau_{\text{flop}} \max\left(1, \frac{B_\tau}{I}\right)
\end{aligned}
$$

Constant power

$$
\begin{aligned}
E &= W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}} + T\pi_0 \\
&= W\epsilon_{\text{flop}}\left(1 + \frac{B_\epsilon}{I} + \frac{\pi_0}{\epsilon_{\text{flop}}}\frac{T}{W}\right)
\end{aligned}
$$

Constant power

Consider: $\dfrac{W\tau_{\text{flop}}}{T}$ and $\dfrac{W\epsilon_{\text{flop}}}{E}$

A microbenchmark study:

“GPU energy” includes all GPU card components (card, memory, fan) but *excludes* the host.

Same for CPU but without GPU card.

NVIDIA GTX 580 – double–precision

Intel i7–950 – double–precision

Normalized performance

1

1/2

1/4

1/8

= 197 GFLOP/s

1.03

= 1.23 GFLOP/J

0.890

1.72 (static=0)

= 53.3 GFLOP/s

2.08

= 0.316 GFLOP/J

0.896 (static=0)

1.01

Time

Energy

1/4 1/2 1 2 4 8 16

Intensity (FLOP : Byte)

NVIDIA GTX 580 – double–precision

Intel i7–950 – double–precision

Normalized performance

Time

Energy

= 197 GFLOP/s

= 53.3 GFLOP/s

1.03

2.08

= 1.23 GFLOP/J

= 0.316 GFLOP/J

0.890

1.72 (static=0)

0.896 (static=0)

1.01

1

1/2

1/4

1/8

1/4

1/2

1

2

4

8

16

Intensity (FLOP : Byte)

Constant power can shift energy-balance

NVIDIA GTX 580 – double–precision

Intel i7–950 – double–precision

Normalized performance

Time

Energy

1

1/2

1/4

1/8

= 197 GFLOP/s

1.03

= 53.3 GFLOP/s

2.08

= 1.23 GFLOP/J

0.890

1.72 (static=0)

= 0.316 GFLOP/J

0.896 (static=0)

1.01

… and
“race-to-halt”
is an artifact of
this shift

1/4

1/2

1

2

4

8

16

Intensity (FLOP : Byte)

NVIDIA GTX 580 – double–precision

Intel i7–950 – double–precision

Time

Energy

Normalized performance

= 197 GFLOP/s

= 53.3 GFLOP/s

= 1.23 GFLOP/J

= 0.316 GFLOP/J

1.03

2.08

0.890

1.72 (static=0)

0.896 (static=0)

1.01

1

1/2

1/4

1/8

1/4

1/2

1

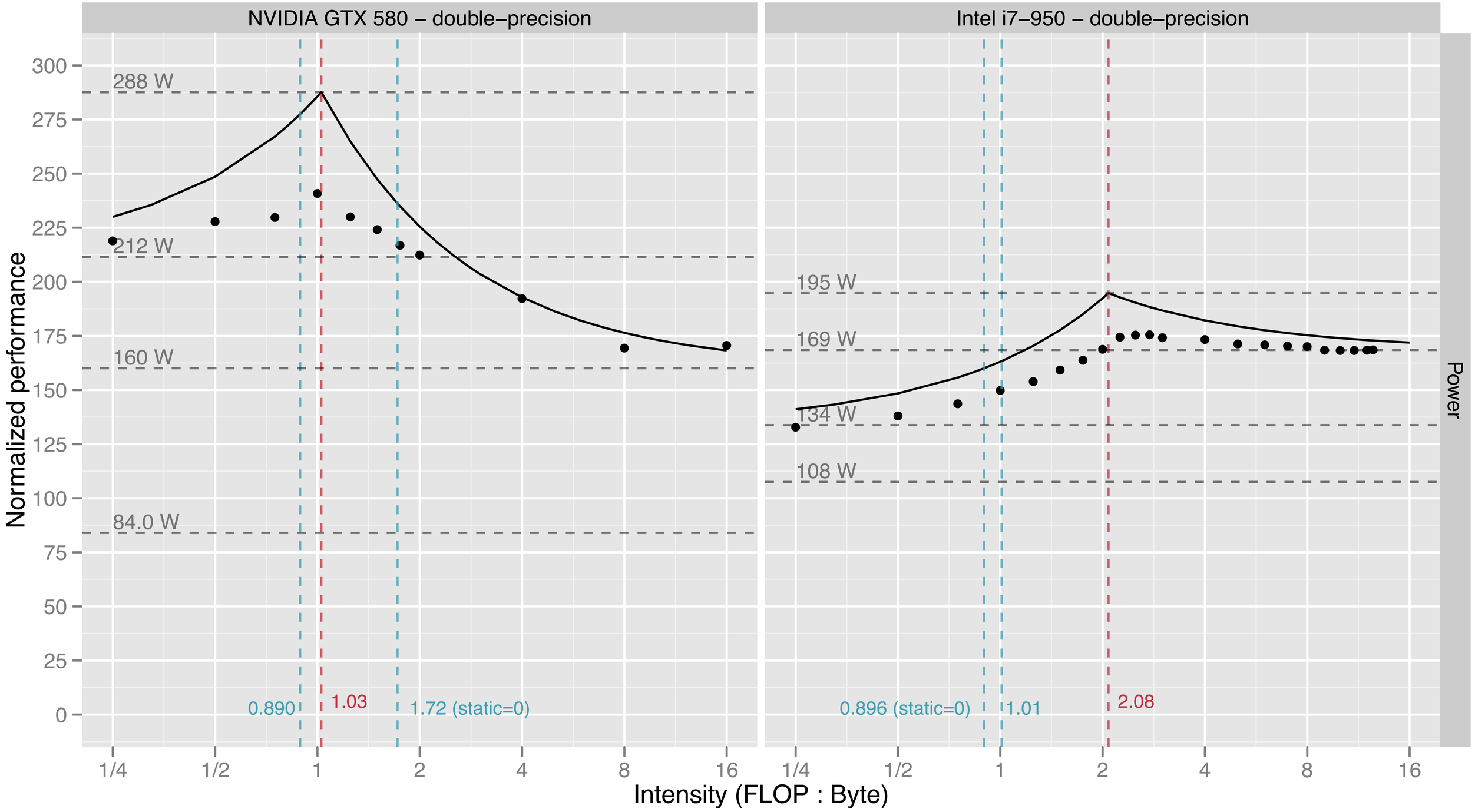
2

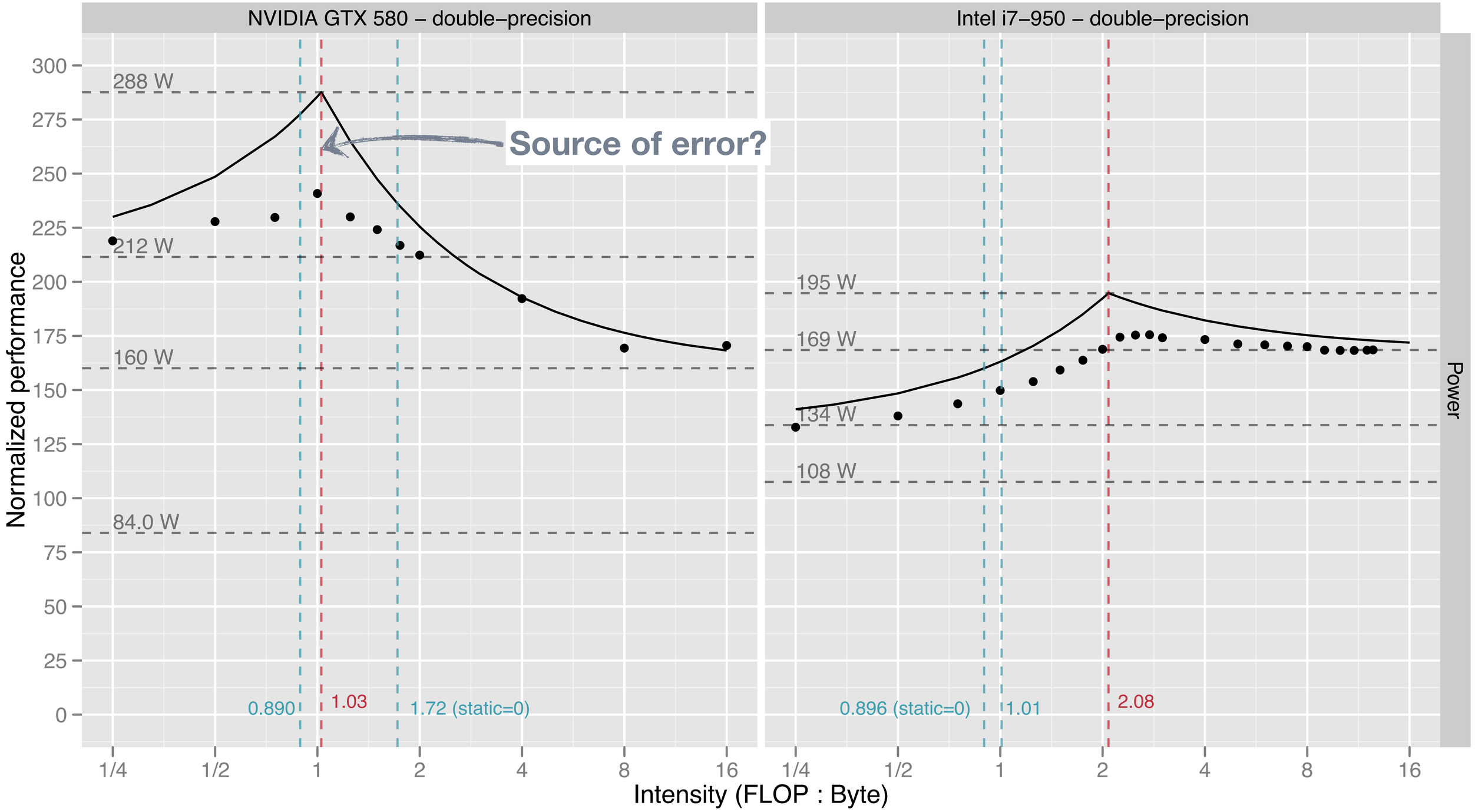
4

8

16

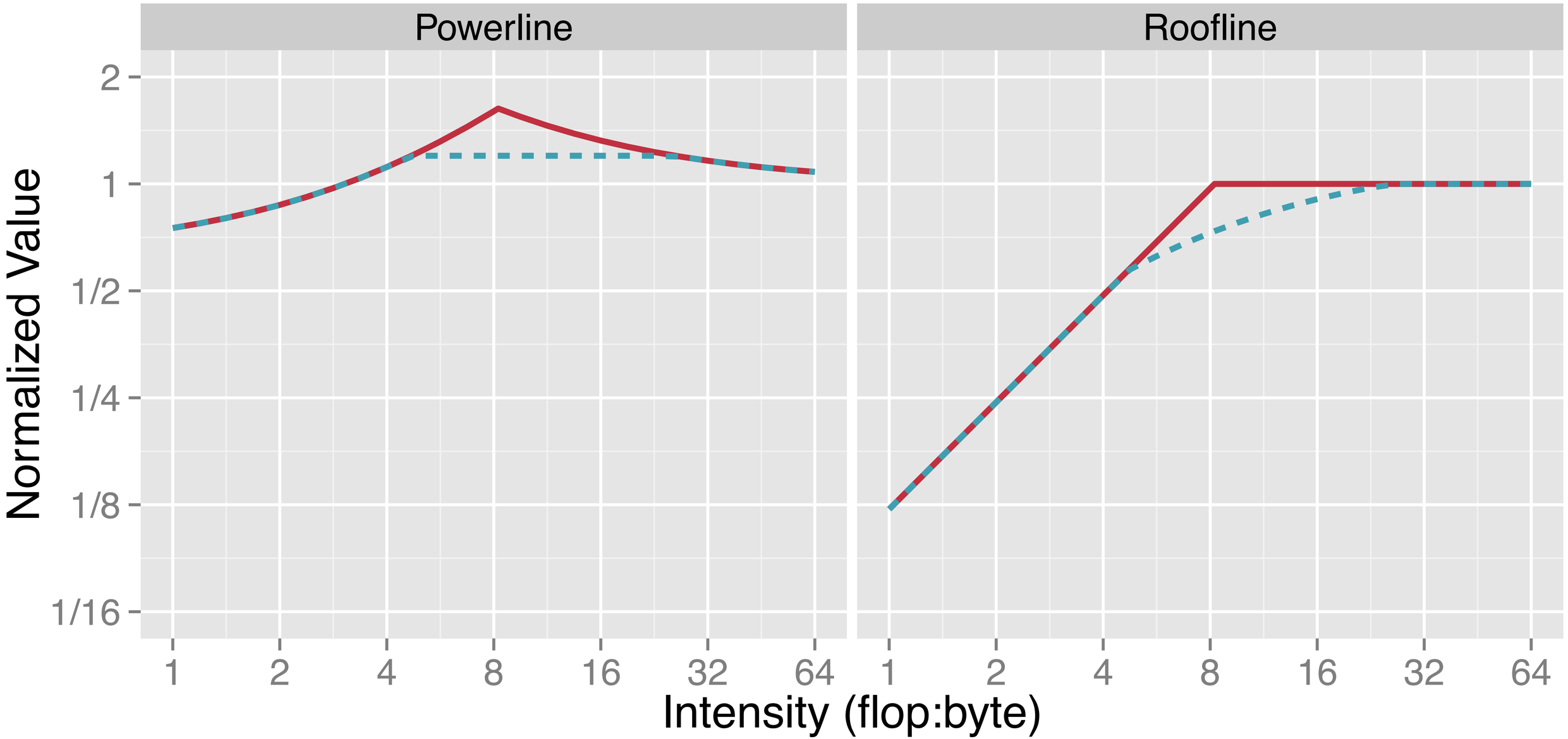
Intensity (FLOP : Byte)

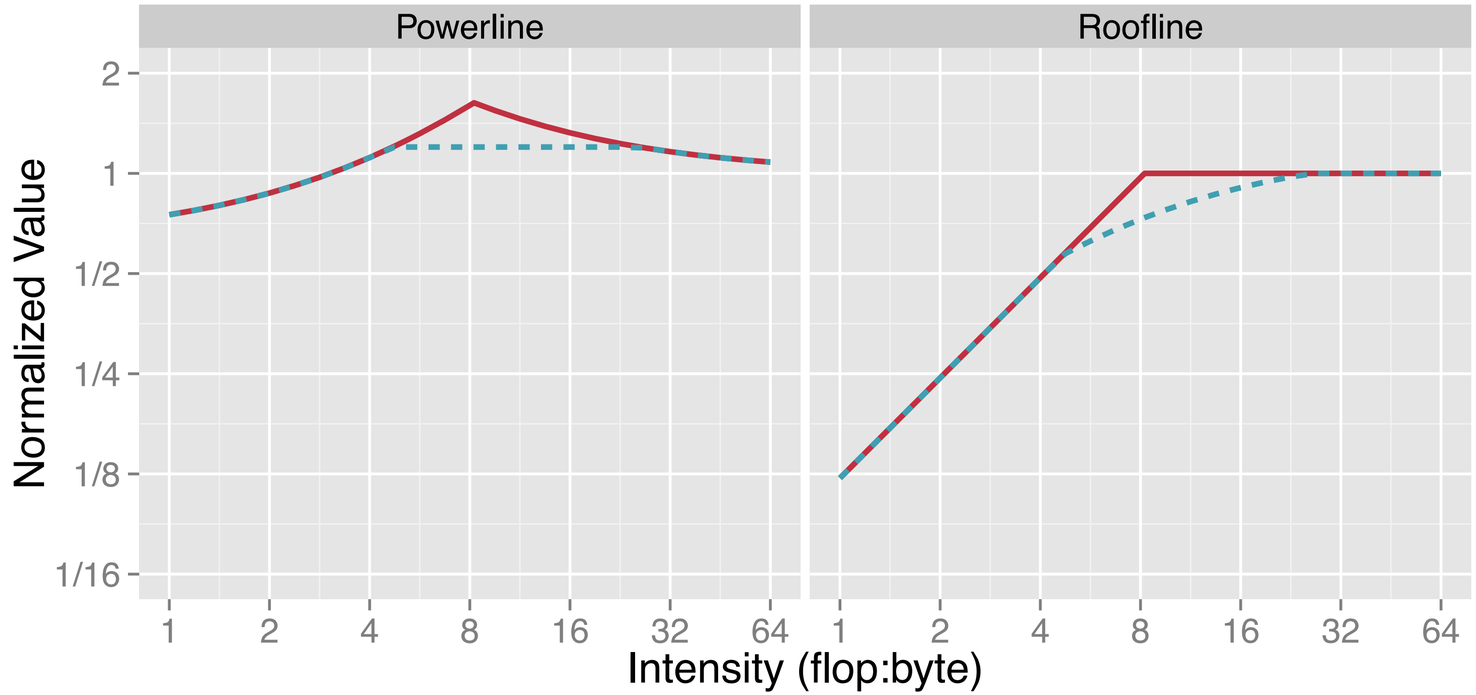

NVIDIA GTX 580 – double–precision
Intel i7–950 – double–precision
Normalized performance
Intensity (FLOP : Byte)
Power
300
275
250
225
200
175
150
125
100
75
50
25
0
288 W
212 W
160 W
84.0 W
195 W
169 W
134 W
108 W
0.890
1.03
1.72 (static=0)
0.896 (static=0)
1.01
2.08
1/4
1/2
1
2
4
8
16


NVIDIA GTX 580 – double–precision
Intel i7–950 – double–precision
Source of error?
288 W
212 W
160 W
84.0 W
195 W
169 W
134 W
108 W
0.890
1.03
1.72 (static=0)
0.896 (static=0)
1.01
2.08
Normalized performance
Power
Intensity (FLOP : Byte)
300
275
250
225
200
175
150
125
100
75
50
25
0
1/4
1/2
1
2
4
8
16

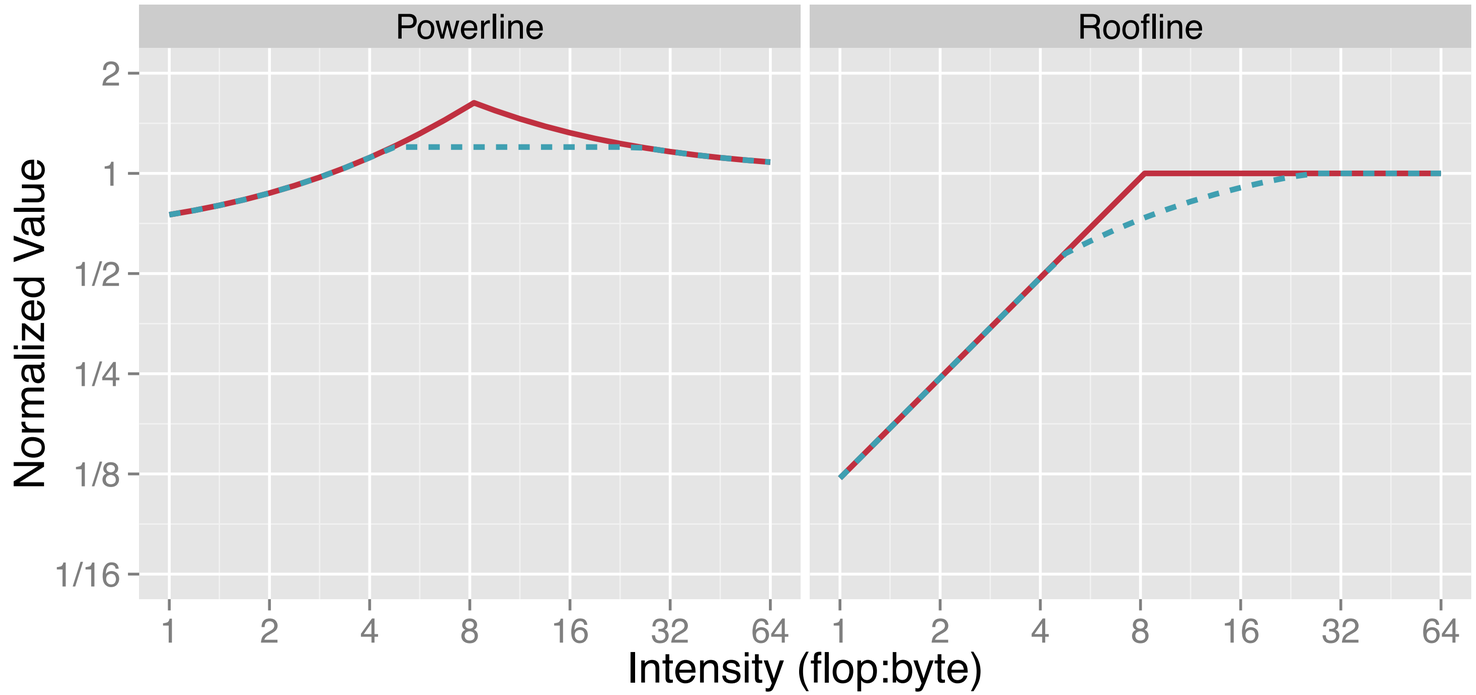
Power capping is critical.
It's also easy to add.

What might a power cap look like?



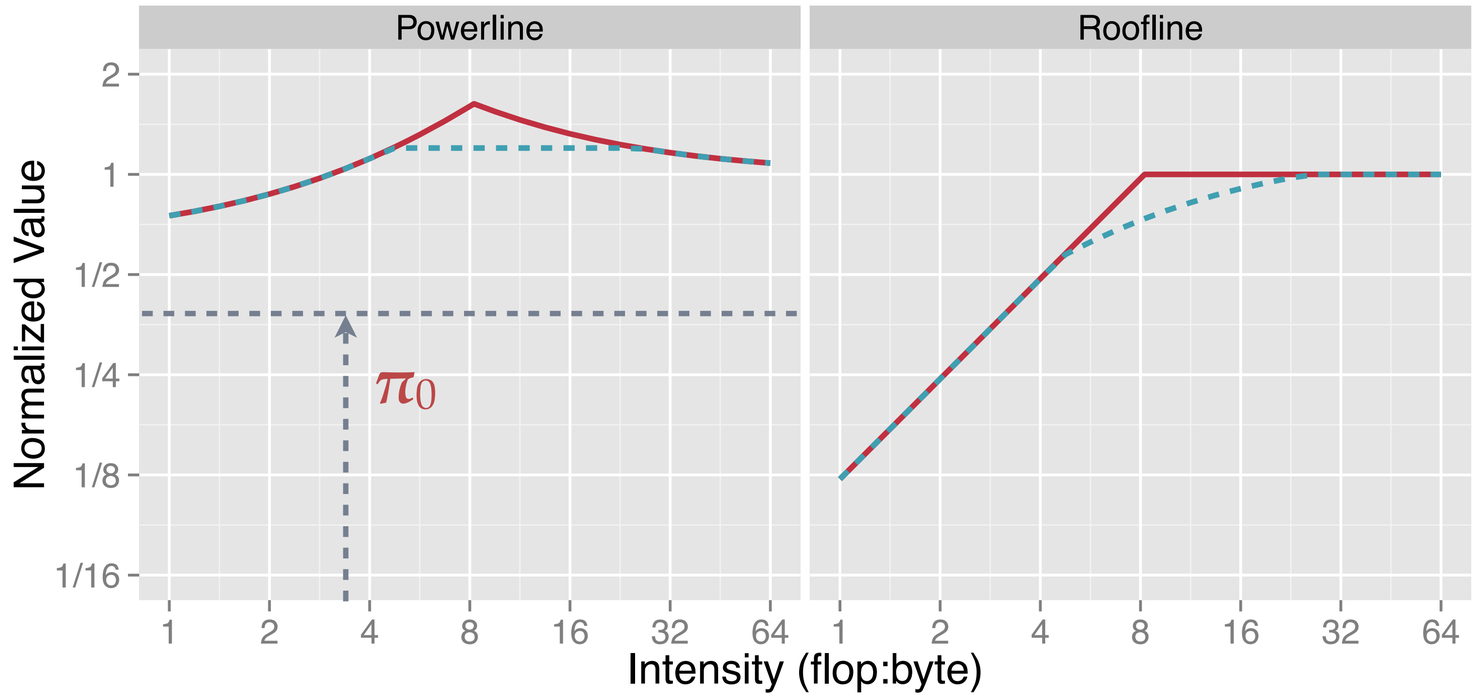


Adding a power cap



π_0 : constant

Adding a power cap



π_0 : constant

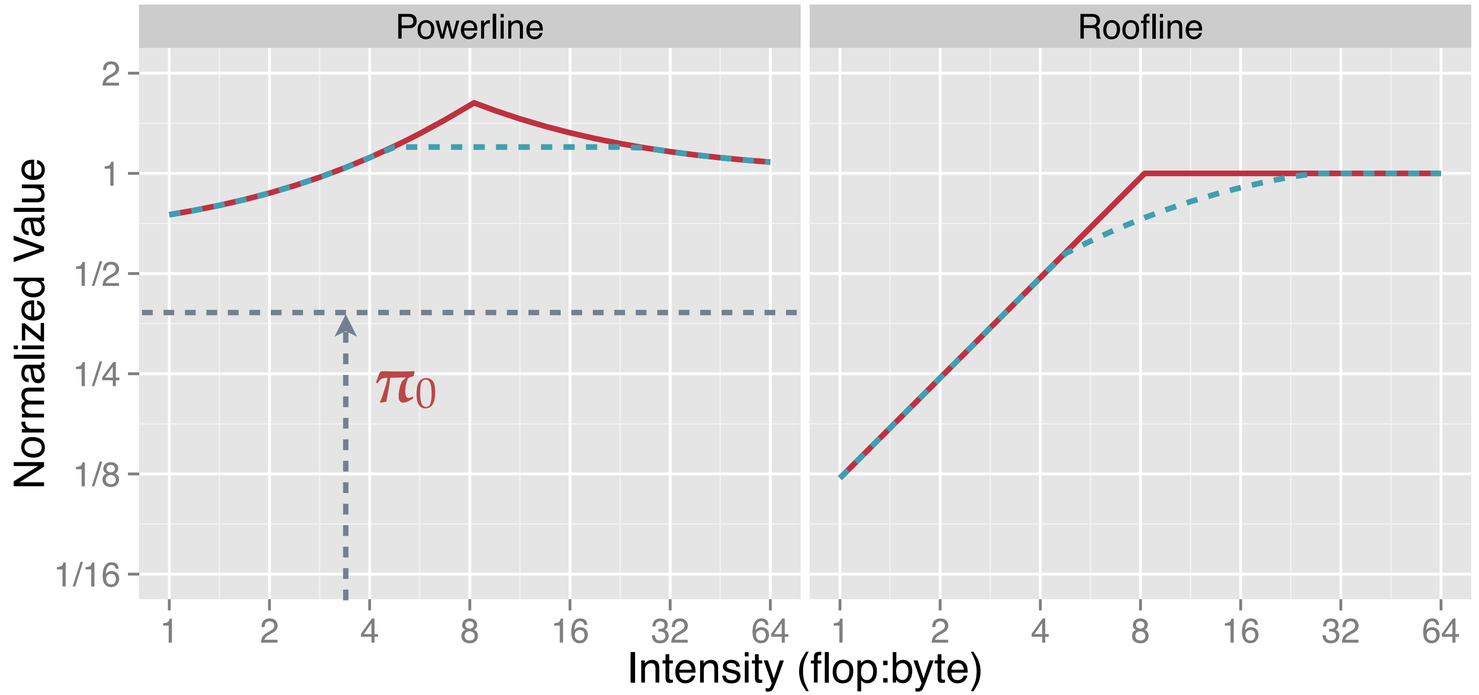
Adding a power cap

$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Adding a power cap

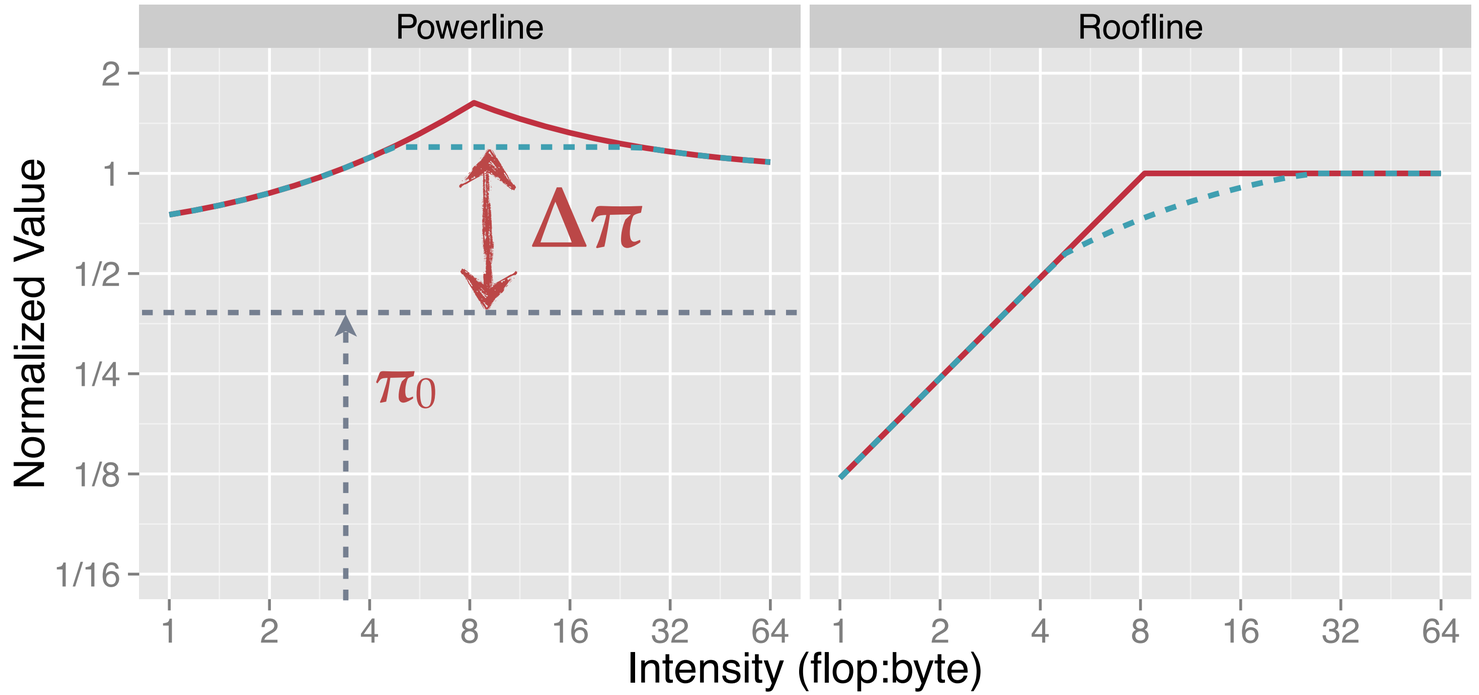


$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Adding a power cap

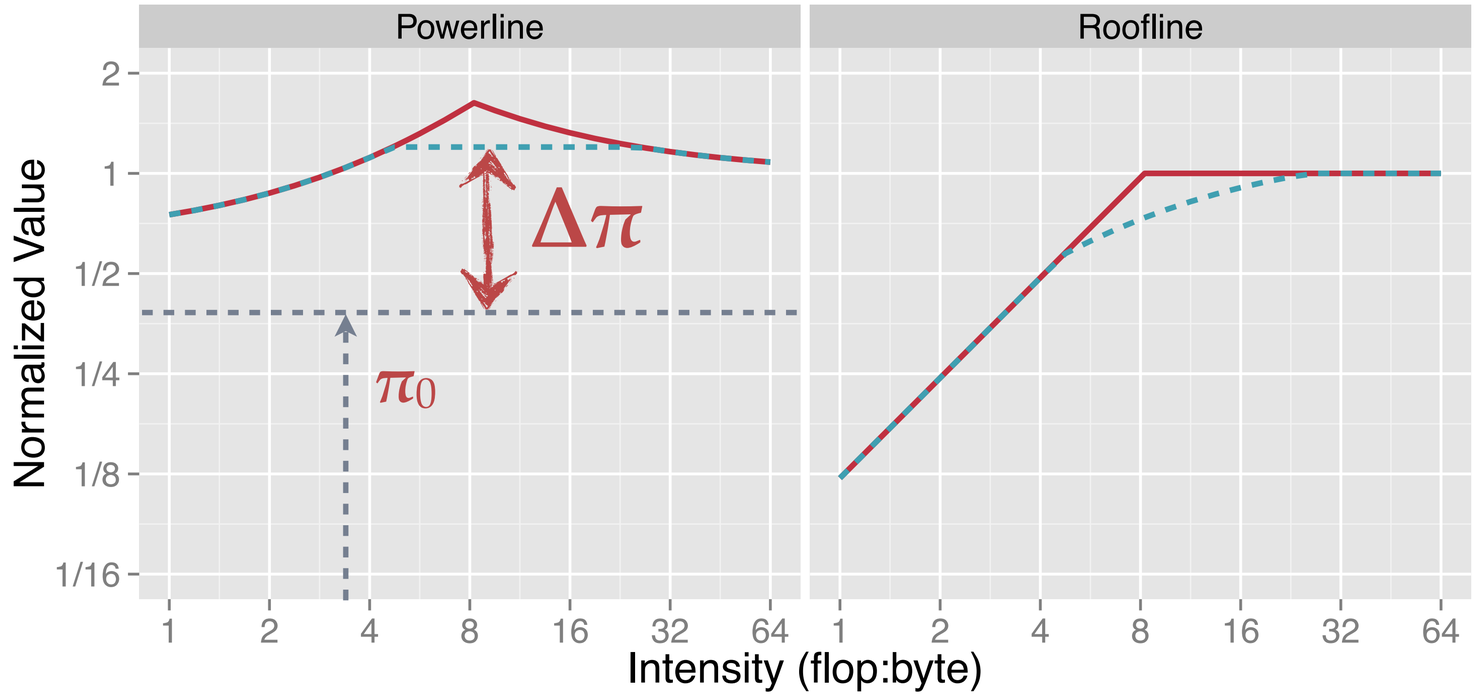


$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Adding a power cap



$$T_{\text{free}} = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right)$$

$$\Downarrow$$

$$T = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}, \frac{W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}}}{\Delta\pi}\right)$$

Time
1.6 TFLOP/s at y=1

Energy
5.7 GFLOP/J at y=1

Power
280 Watts at y=1

Normalized performance

2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512

1/4 1/2 1 2 4 8 16 32 64 128

Intensity (single–precision FLOP : Byte)

NVIDIA
GTX 580
GF100
Fermi

Time
1.6 TFLOP/s at y=1

Energy
5.7 GFLOP/J at y=1

Power
280 Watts at y=1

Normalized performance

2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512

1/4 1/2 1 2 4 8 16 32 64 128 1/4 1/2 1 2 4 8 16 32 64 128 1/4 1/2 1 2 4 8 16 32 64 128

Intensity (single–precision FLOP : Byte)

A better fit!

NVIDIA
GTX 580
GF100
Fermi

The model suggests a **structure** in the time, energy, and power relationships. It also facilitates **analysis**.

GTX Titan
GTX 680
Xeon Phi
NUC GPU
Arndale GPU
APU GPU
GTX 580
NUC CPU
PandaBoard ES
Arndale CPU
APU CPU
Desktop CPU

Power [normalized]

1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6

Cap
Memory
Compute

16 Gflop/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]

15 Gflop/J
3.0 Tflop/s [86%], 160 GB/s [82%]
66 W (const) + 140 W (cap) [100%]

11 Gflop/J
2.0 Tflop/s [100%], 180 GB/s [57%]
180 W (const) + 36 W (cap) [100%]

8.8 Gflop/J
270 Gflop/s [100%], 15 GB/s [60%]
10 W (const) + 18 W (cap) [91%]

8.1 Gflop/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]

6.4 Gflop/J
100 Gflop/s [95%], 8.7 GB/s [81%]
16 W (const) + 3.2 W (cap) [100%]

5.4 Gflop/J
1.4 Tflop/s [88%], 170 GB/s [89%]
120 W (const) + 150 W (cap) [94%]

3.2 Gflop/J
56 Gflop/s [97%], 18 GB/s [70%]
17 W (const) + 7.4 W (cap) [98%]

2.5 Gflop/J
9.5 Gflop/s [99%], 1.3 GB/s [40%]
3.5 W (const) + 1.2 W (cap) [95%]

2.2 Gflop/J
16 Gflop/s [58%], 3.9 GB/s [31%]
5.5 W (const) + 2.0 W (cap) [97%]

650 Mflop/J
13 Gflop/s [98%], 3.3 GB/s [31%]
20 W (const) + 1.4 W (cap) [98%]

620 Mflop/J
99 Gflop/s [93%], 19 GB/s [74%]
120 W (const) + 44 W (cap) [99%]

1/8 1/2 2 8 32 128 512

Intensity (single−precision flop:Byte)

Tuesday, September 10, 13

$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

$$T = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}, \frac{W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}}}{\Delta\pi}\right)$$

$$\Downarrow$$

$$\tilde{\tau}_{\text{flop}} \equiv \frac{T}{W} \equiv \tau_{\text{flop}} s_{\text{flop}}$$

$$\tilde{\tau}_{\text{mem}} \equiv \frac{T}{Q} \equiv \tau_{\text{mem}} s_{\text{mem}}$$

Throttling factors
i.e., allowable slowdown

$$\Downarrow$$

$$s_{\text{flop}} \equiv \max\left\{1, \frac{B_\tau}{I}, \frac{\epsilon_{\text{flop}}/\tau_{\text{flop}}}{\Delta\pi}\left(1 + \frac{B_\epsilon}{I}\right)\right\}$$

$$s_{\text{mem}} \equiv s_{\text{flop}} \frac{I}{B_\tau}$$

Example:
Caps imply throttling!

$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Example:

Caps imply throttling!

$$T = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}, \frac{W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}}}{\Delta\pi}\right)$$

$$\Downarrow$$

$$\tilde{\tau}_{\text{flop}} \equiv \frac{T}{W} \equiv \tau_{\text{flop}} s_{\text{flop}}$$

$$\tilde{\tau}_{\text{mem}} \equiv \frac{T}{Q} \equiv \tau_{\text{mem}} s_{\text{mem}}$$

Throttling factors
i.e., allowable slowdown

$$\Downarrow$$

$$s_{\text{flop}} \equiv \max\left\{1, \frac{B_\tau}{I}, \frac{\epsilon_{\text{flop}}/\tau_{\text{flop}}}{\Delta\pi}\left(1 + \frac{B_\epsilon}{I}\right)\right\}$$

$$s_{\text{mem}} \equiv s_{\text{flop}} \frac{I}{B_\tau}$$

$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Example:
Caps imply throttling!

$$T = \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}, \frac{W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}}}{\Delta\pi}\right)$$

$$\Downarrow$$

$$\tilde{\tau}_{\text{flop}} \equiv \frac{T}{W} \equiv \tau_{\text{flop}} s_{\text{flop}}$$

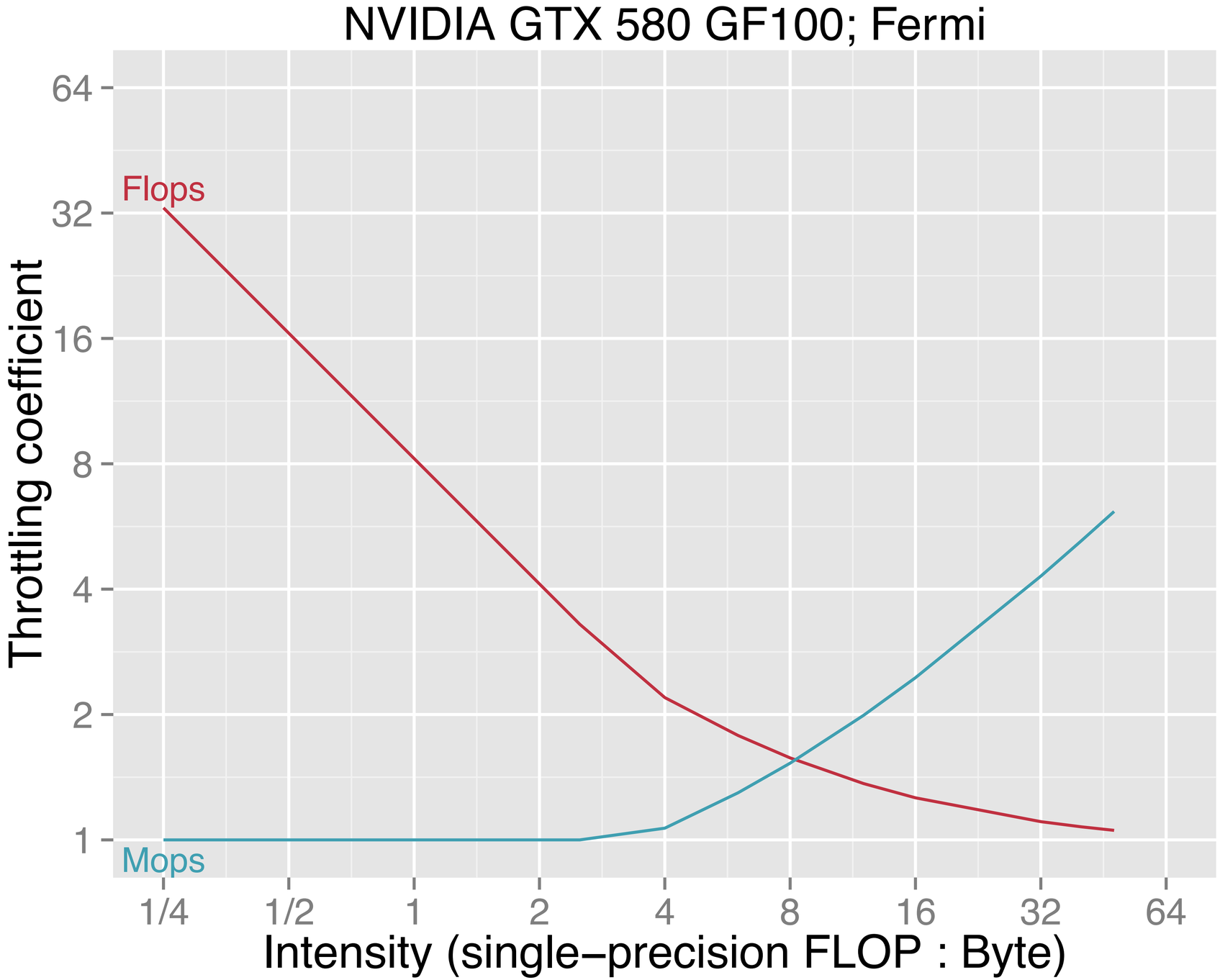
$$\tilde{\tau}_{\text{mem}} \equiv \frac{T}{Q} \equiv \tau_{\text{mem}} s_{\text{mem}}$$

Throttling factors
i.e., allowable slowdown

$$\Downarrow$$

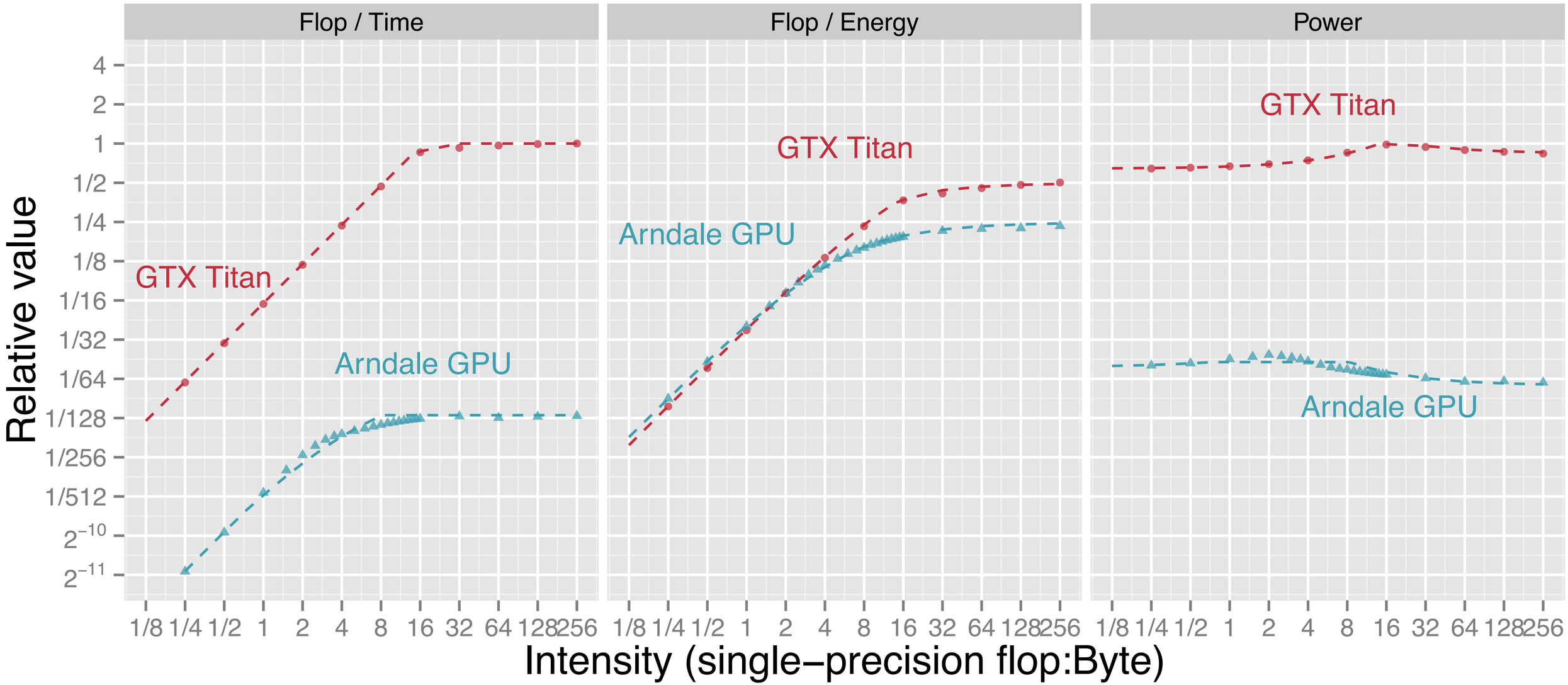
$$s_{\text{flop}} \equiv \max\left\{1, \frac{B_\tau}{I}, \frac{\epsilon_{\text{flop}}/\tau_{\text{flop}}}{\Delta\pi}\left(1 + \frac{B_\epsilon}{I}\right)\right\}$$

$$s_{\text{mem}} \equiv s_{\text{flop}} \frac{I}{B_\tau}$$


NVIDIA GTX 580 GF100; Fermi
Throttling coefficient
64
32
16
8
4
2
1
Flops
Mops
1/4
1/2
1
2
4
8
16
32
64
Intensity (single–precision FLOP : Byte)

A “what if …” comparison of system building blocks:

→ **GTX Titan** vs. **Samsung Exynos 5** (GPU only; Arndale)


Flop / Time
Flop / Energy
Power
Relative value
4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}
GTX Titan
Arndale GPU
GTX Titan
Arndale GPU
GTX Titan
Arndale GPU
1/8 1/4 1/2 1 2 4 8 16 32 64 128 256
1/8 1/4 1/2 1 2 4 8 16 32 64 128 256
1/8 1/4 1/2 1 2 4 8 16 32 64 128 256
Intensity (single−precision flop:Byte)

Flop / Time

Flop / Energy

Power

Relative value

4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}

GTX Titan

Arndale GPU

~ 122×

GTX Titan

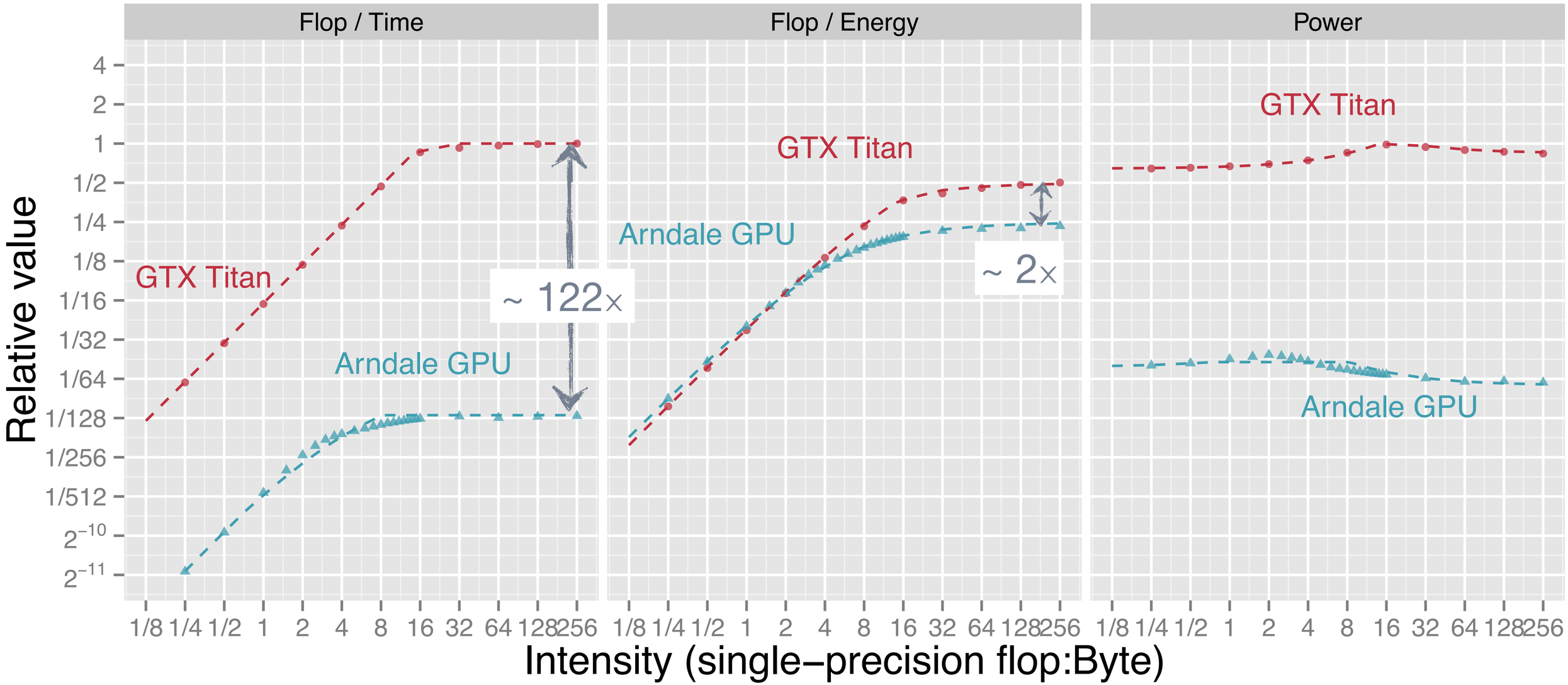
Arndale GPU

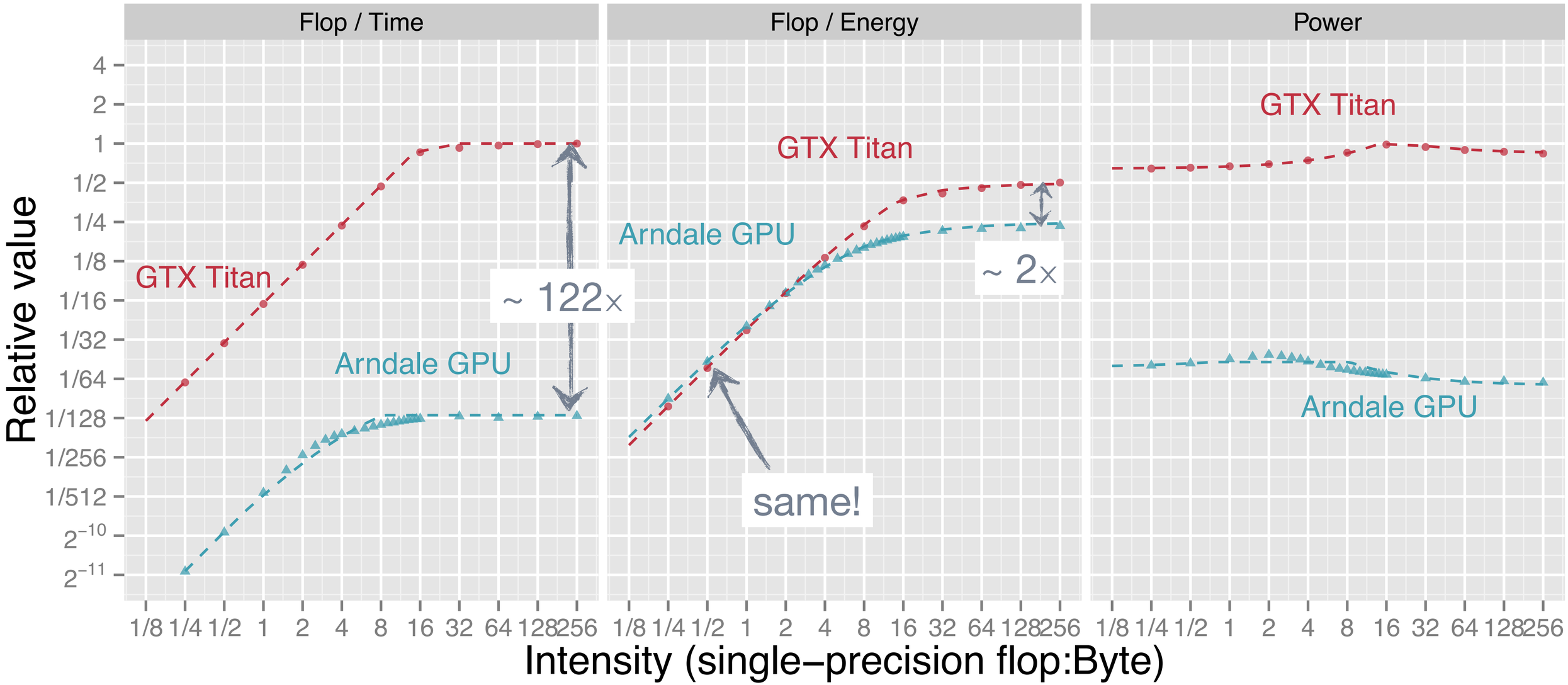
GTX Titan

Arndale GPU

1/8 1/4 1/2 1 2 4 8 16 32 64 128 256

Intensity (single–precision flop:Byte)


Flop / Time
Flop / Energy
Power
Relative value
4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}
GTX Titan
Arndale GPU
~ 122×
GTX Titan
Arndale GPU
~ 2×
GTX Titan
Arndale GPU
1/8 1/4 1/2 1 2 4 8 16 32 64 128 256
Intensity (single−precision flop:Byte)


Flop / Time
Flop / Energy
Power
Relative value
4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}
GTX Titan
Arndale GPU
~ 122×
GTX Titan
Arndale GPU
~ 2×
same!
GTX Titan
Arndale GPU
1/8 1/4 1/2 1 2 4 8 16 32 64 128 256
Intensity (single–precision flop:Byte)

Flop / Time

Flop / Energy

Power

122 x Arndale GPU

GTX Titan

Arndale GPU

GTX Titan

Arndale GPU

GTX Titan

122 x Arndale GPU

Arndale GPU

Relative value

4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}

1/8 1/4 1/2 1 2 4 8 16 32 64 128 256

Intensity (single−precision flop:Byte)

Flop / Time

Flop / Energy

Power

Relative value

4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}

122 x Arndale GPU

~ 4x

GTX Titan

Arndale GPU

GTX Titan

Arndale GPU

GTX Titan

122 x Arndale GPU

Arndale GPU

1/8 1/4 1/2 1 2 4 8 16 32 64 128 256

Intensity (single–precision flop:Byte)

Flop / Time

Flop / Energy

Power

~ 4× – so not free!

122 x Arndale GPU

~ 4×

GTX Titan

Arndale GPU

GTX Titan

Arndale GPU

GTX Titan

122 x Arndale GPU

Arndale GPU

Relative value

4
2
1
1/2
1/4
1/8
1/16
1/32
1/64
1/128
1/256
1/512
2^{-10}
2^{-11}

1/8 1/4 1/2 1 2 4 8 16 32 64 128 256

Intensity (single–precision flop:Byte)

Follow along:
`hpcgarage.org/ppam13`

Rebutting Hillis, Part 2:

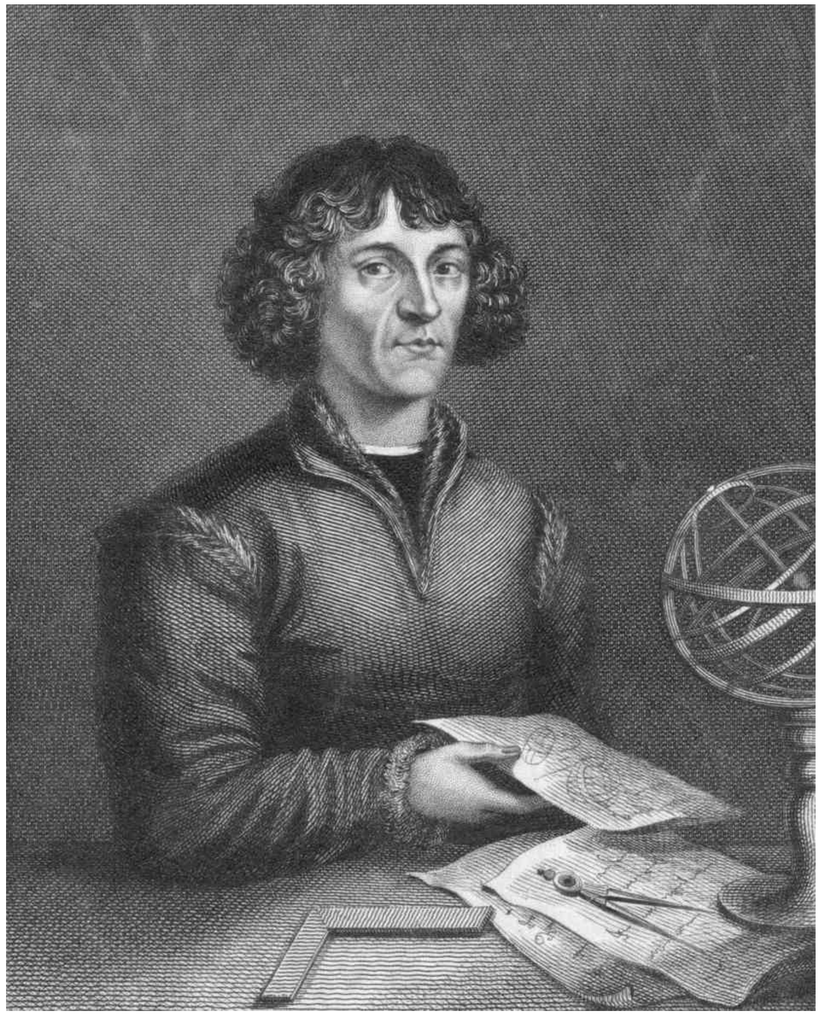
Kent Czechowski

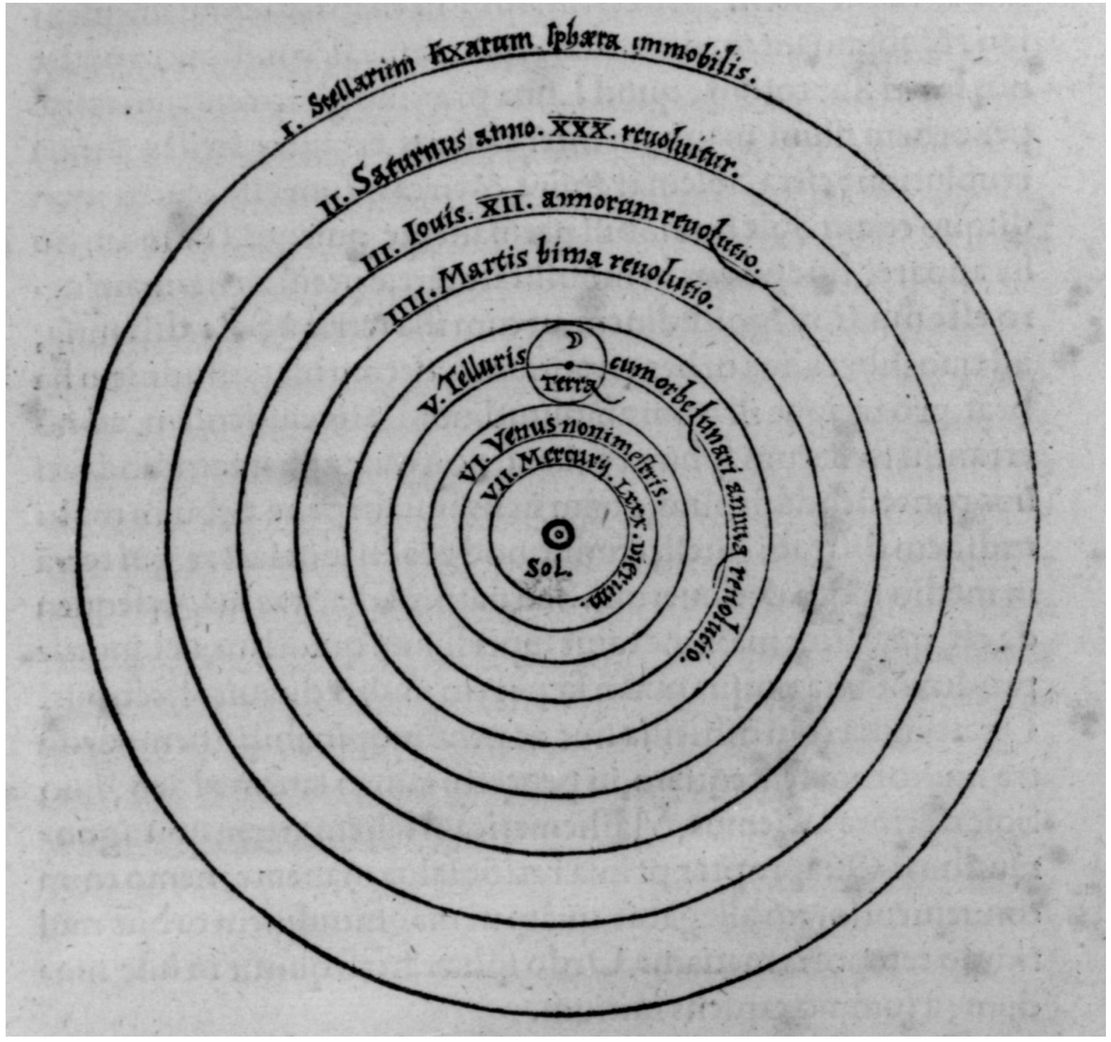
"Algorithmic" power and die-area constraints

Can first principles of algorithms direct architectures toward bigger wins?

See also Czechowski & Vuduc in **IPDPS'13** [`http://vuduc.org/pubs/czechowski2013-codesign-ipdps.pdf`]

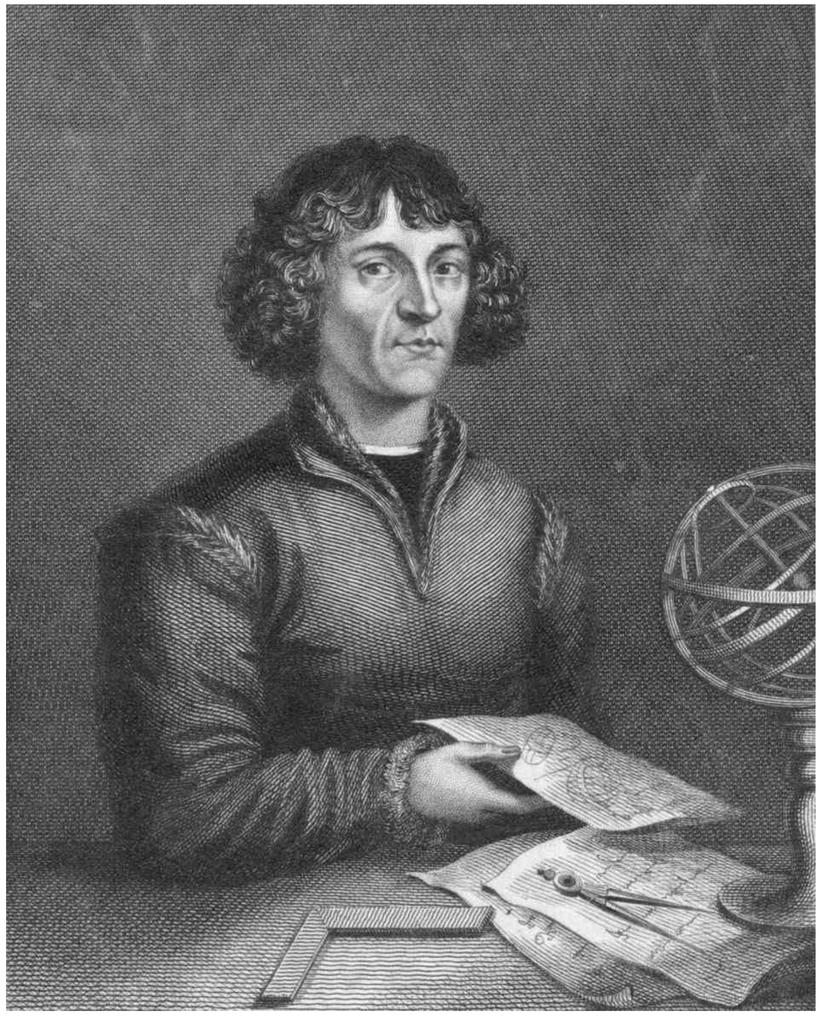
Kent: “What problem are we really trying to solve?”





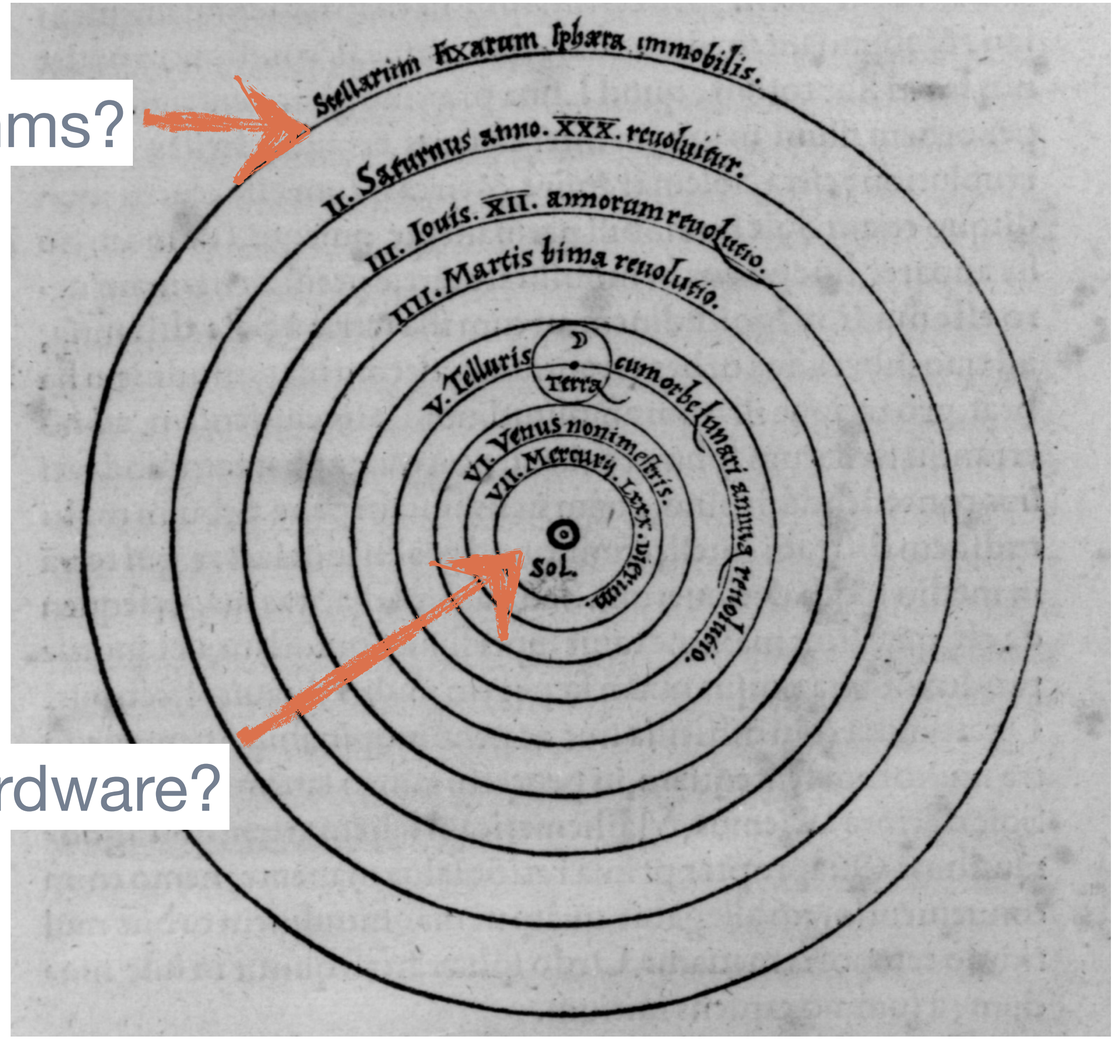
http://media.npr.org/assets/img/2011/11/02/copernicus_custom-54ef1ed5bcadf2be8a9c44d36dc7b273696946b2-s6-c30.jpg
http://www.hps.cam.ac.uk/starry/copernicuslrg.jpg

Kent: “What problem are we really trying to solve?”



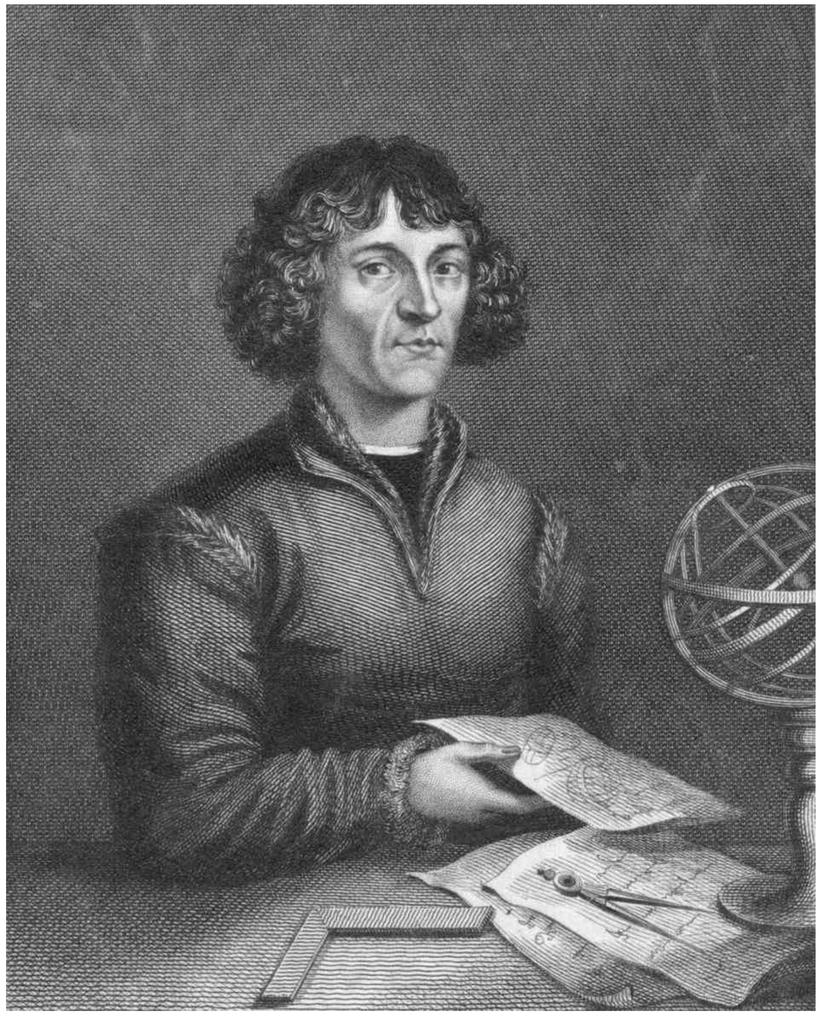
Algorithms?

Hardware?



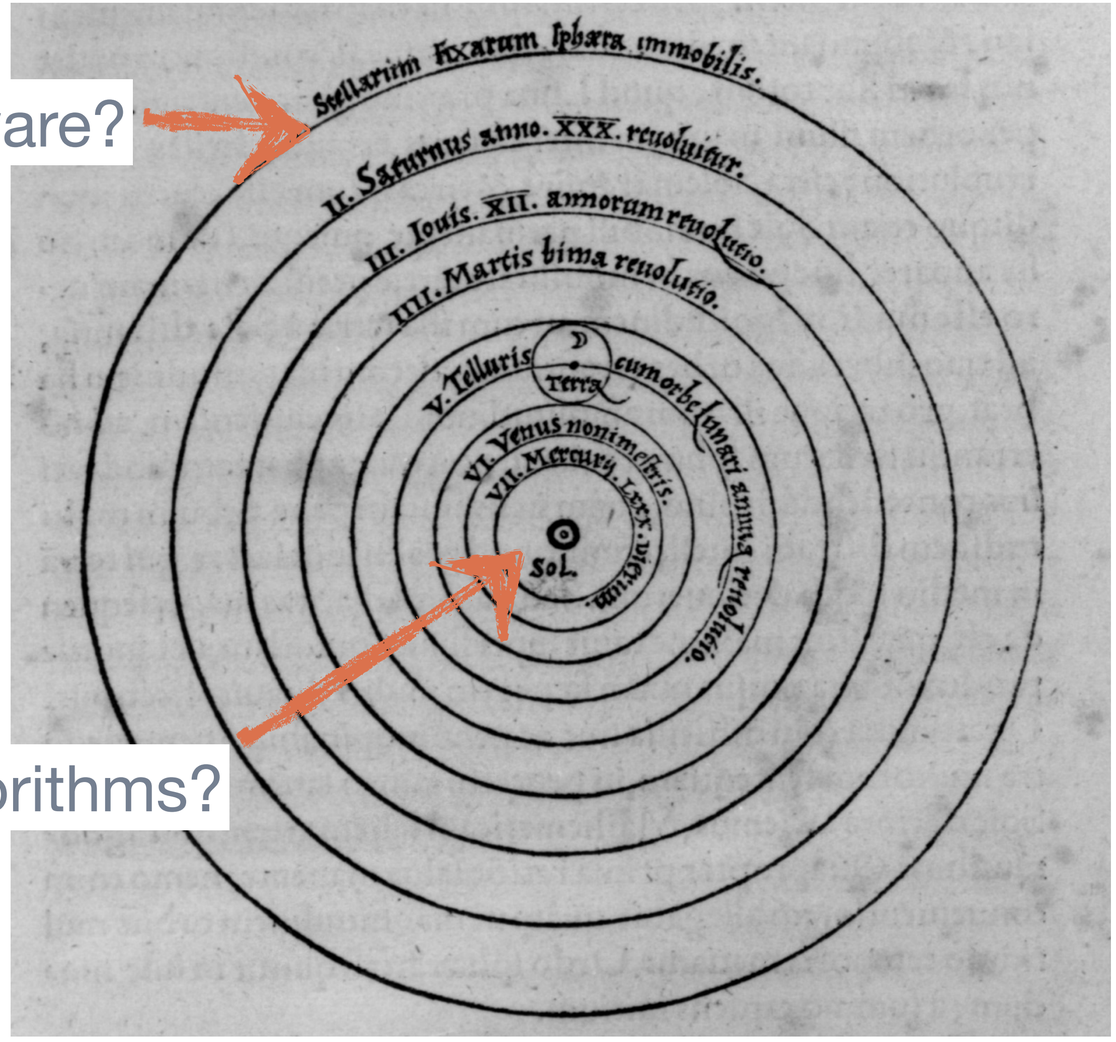
http://media.npr.org/assets/img/2011/11/02/copernicus_custom-54ef1ed5bcadf2be8a9c44d36dc7b273696946b2-s6-c30.jpg
http://www.hps.cam.ac.uk/starry/copernicuslrg.jpg

Kent: “What problem are we really trying to solve?”



Hardware?

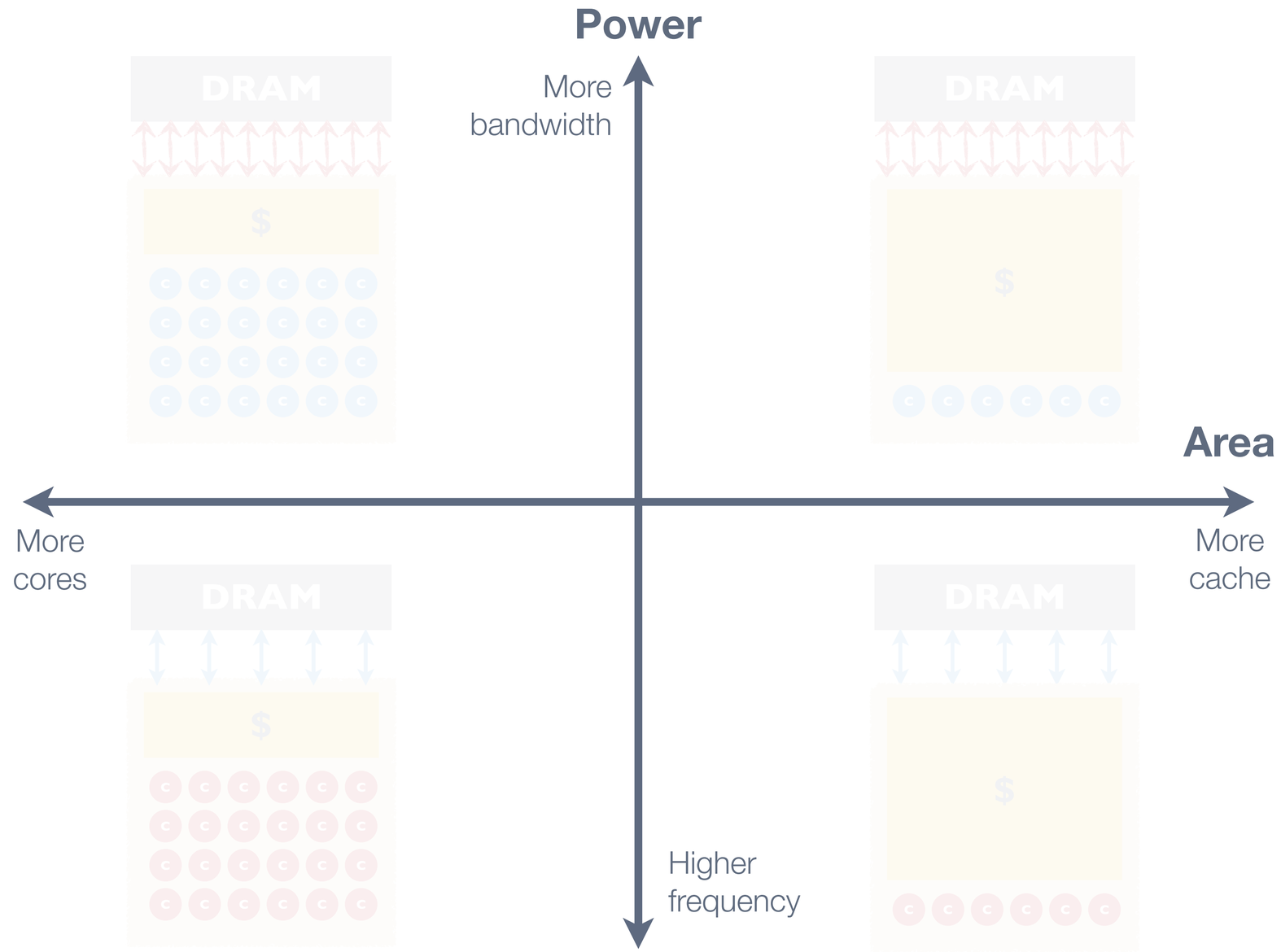
Algorithms?



http://media.npr.org/assets/img/2011/11/02/copernicus_custom-54ef1ed5bcadf2be8a9c44d36dc7b273696946b2-s6-c30.jpg
http://www.hps.cam.ac.uk/starry/copernicuslrg.jpg

A Notional Design Problem (1 processor-node example)

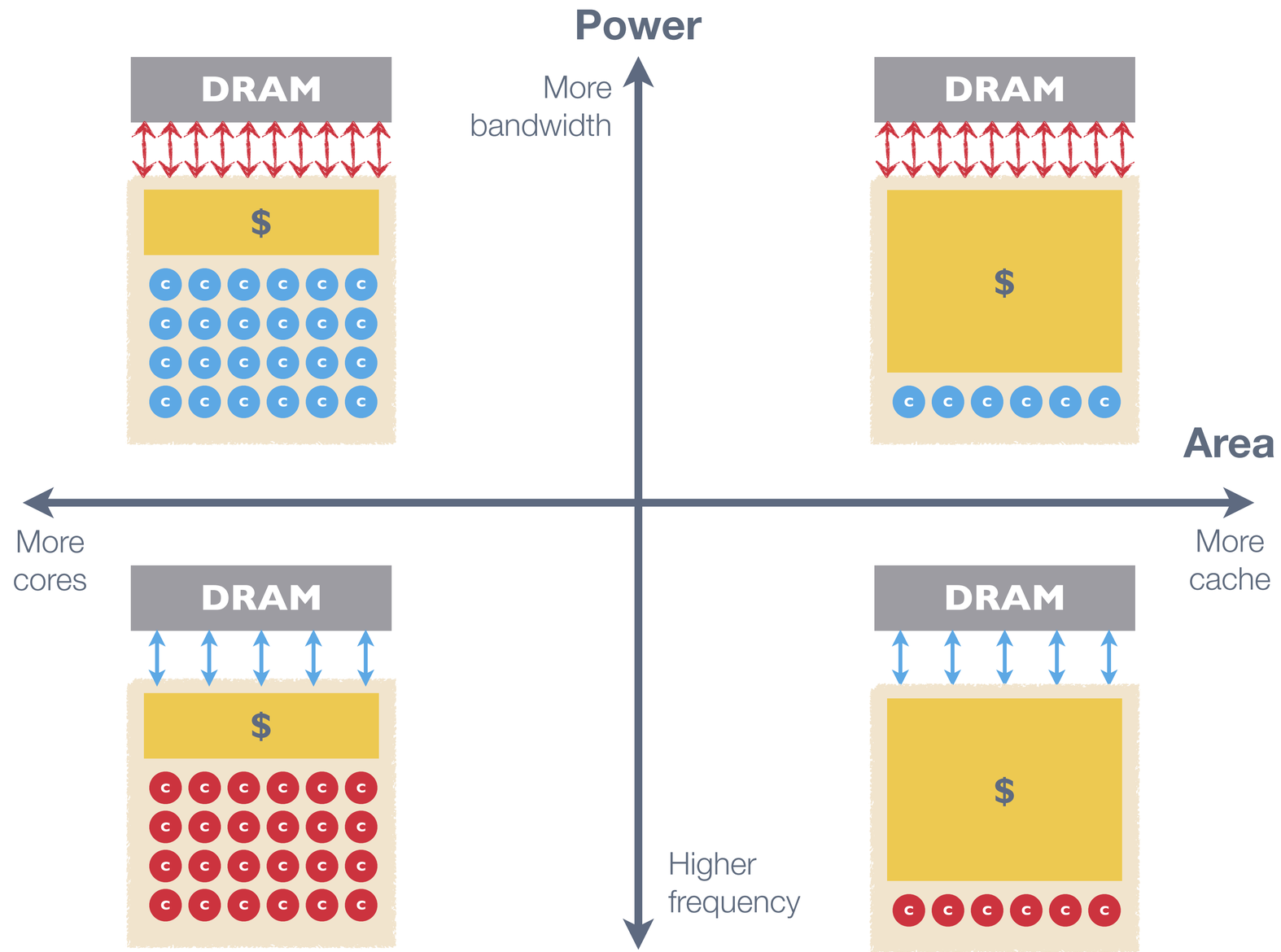
For fixed:
* **die area** (transistors)
* **power** budget
* computation



A Notional Design Problem (1 processor-node example)

For fixed:

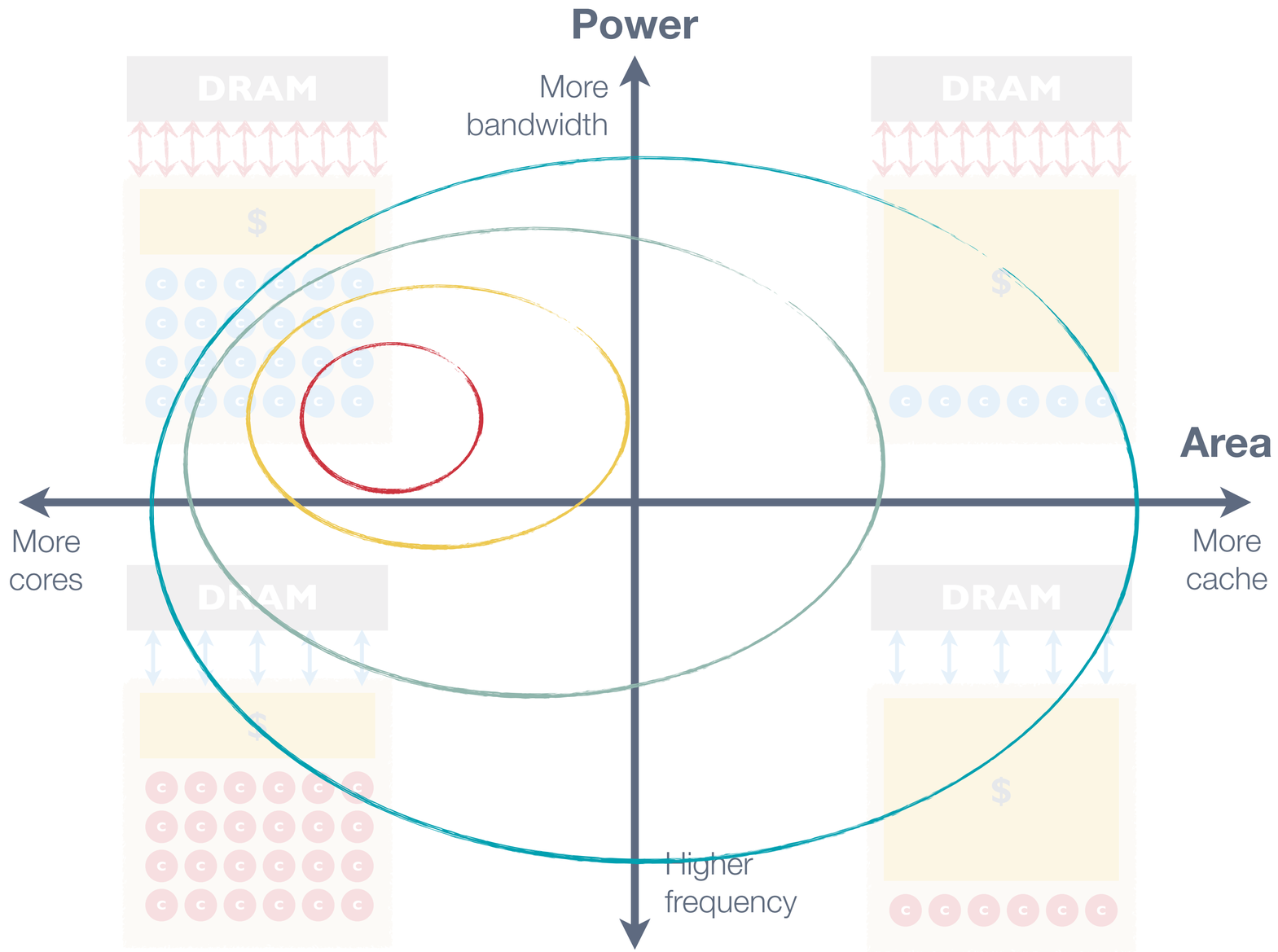
* **die area** (transistors)
* **power** budget
* computation



A Notional Design Problem (1 processor-node example)

For fixed:

* die area (transistors)
* power budget
* **computation**

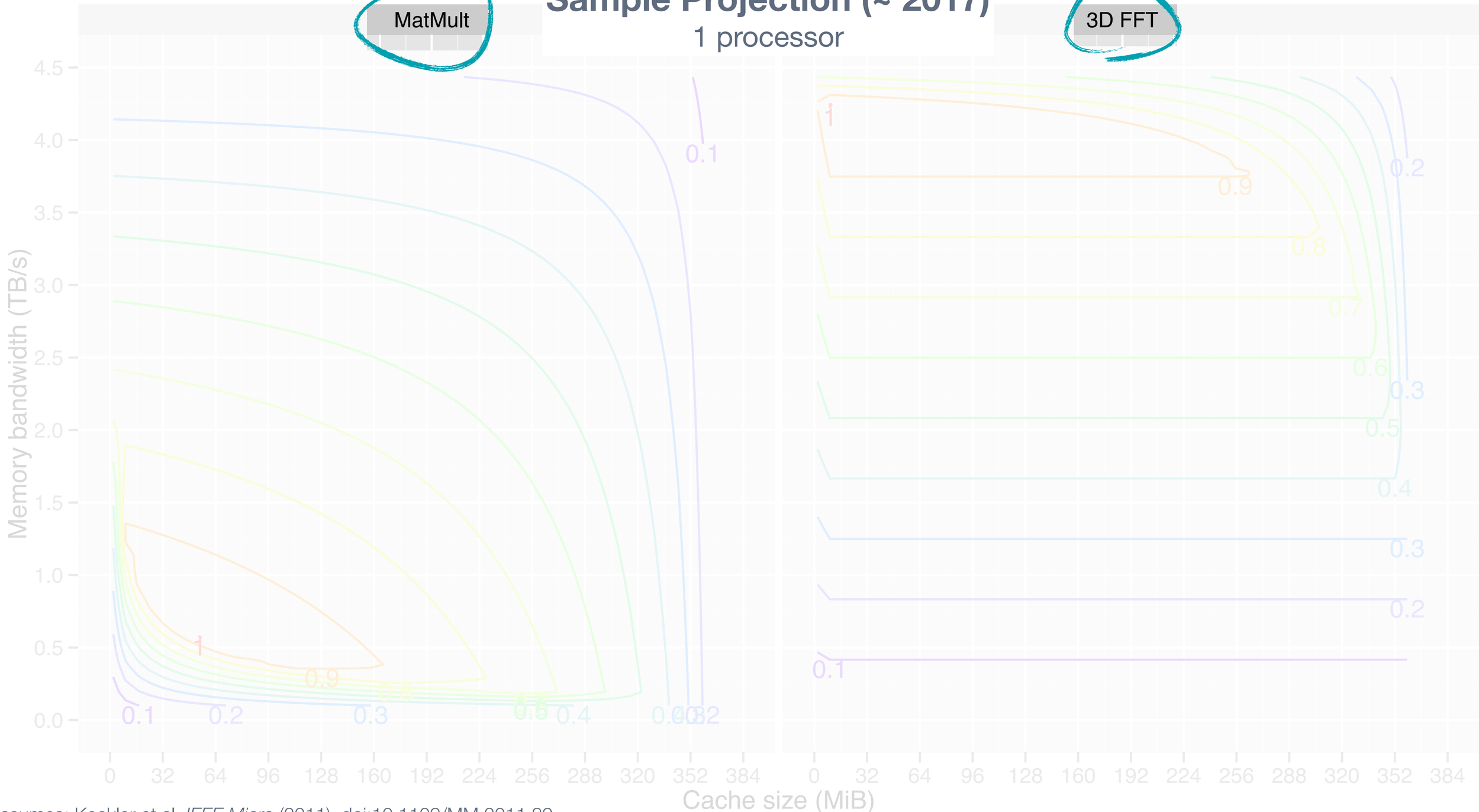


Sample Projection (~ 2017)

1 processor

MatMult

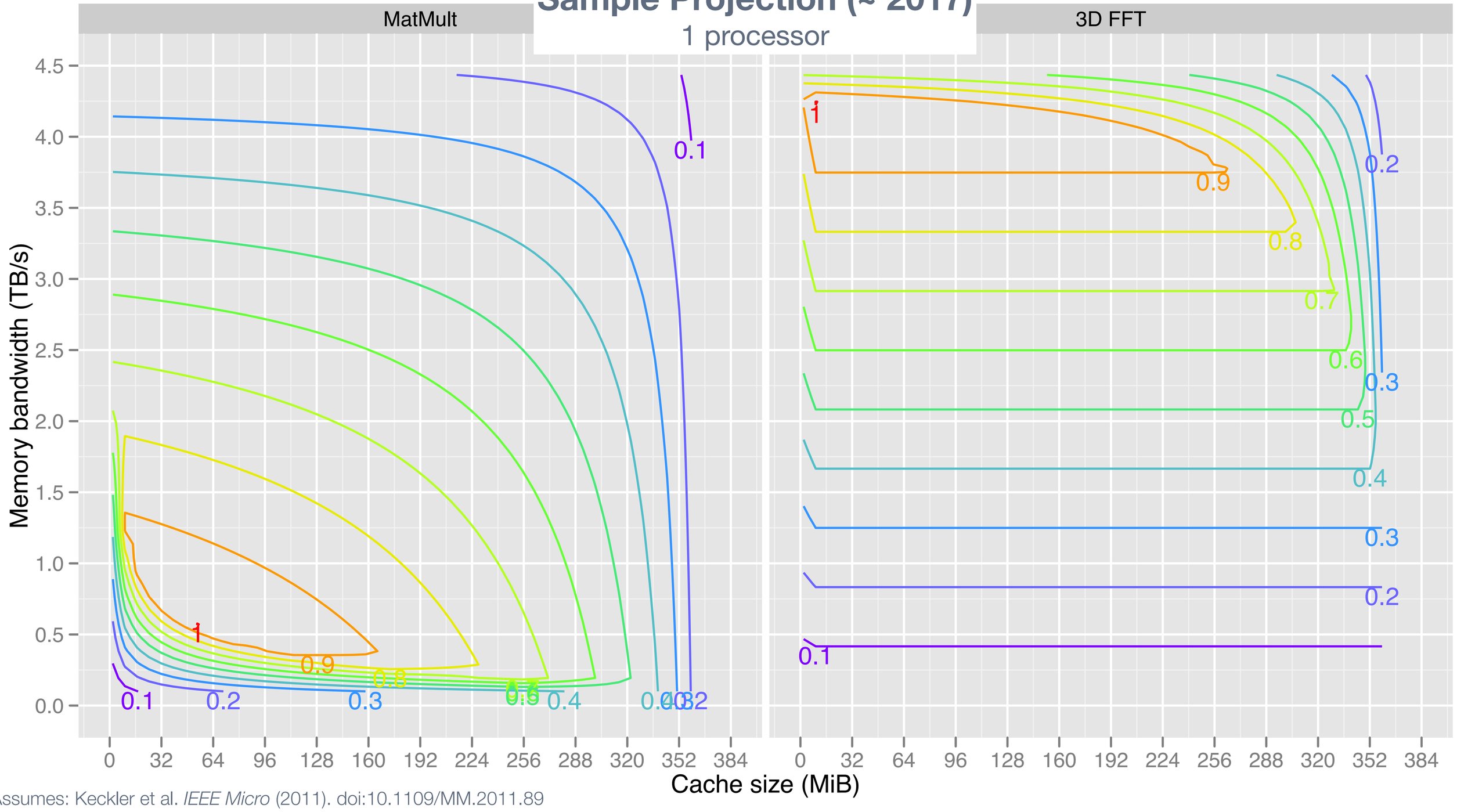
3D FFT



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

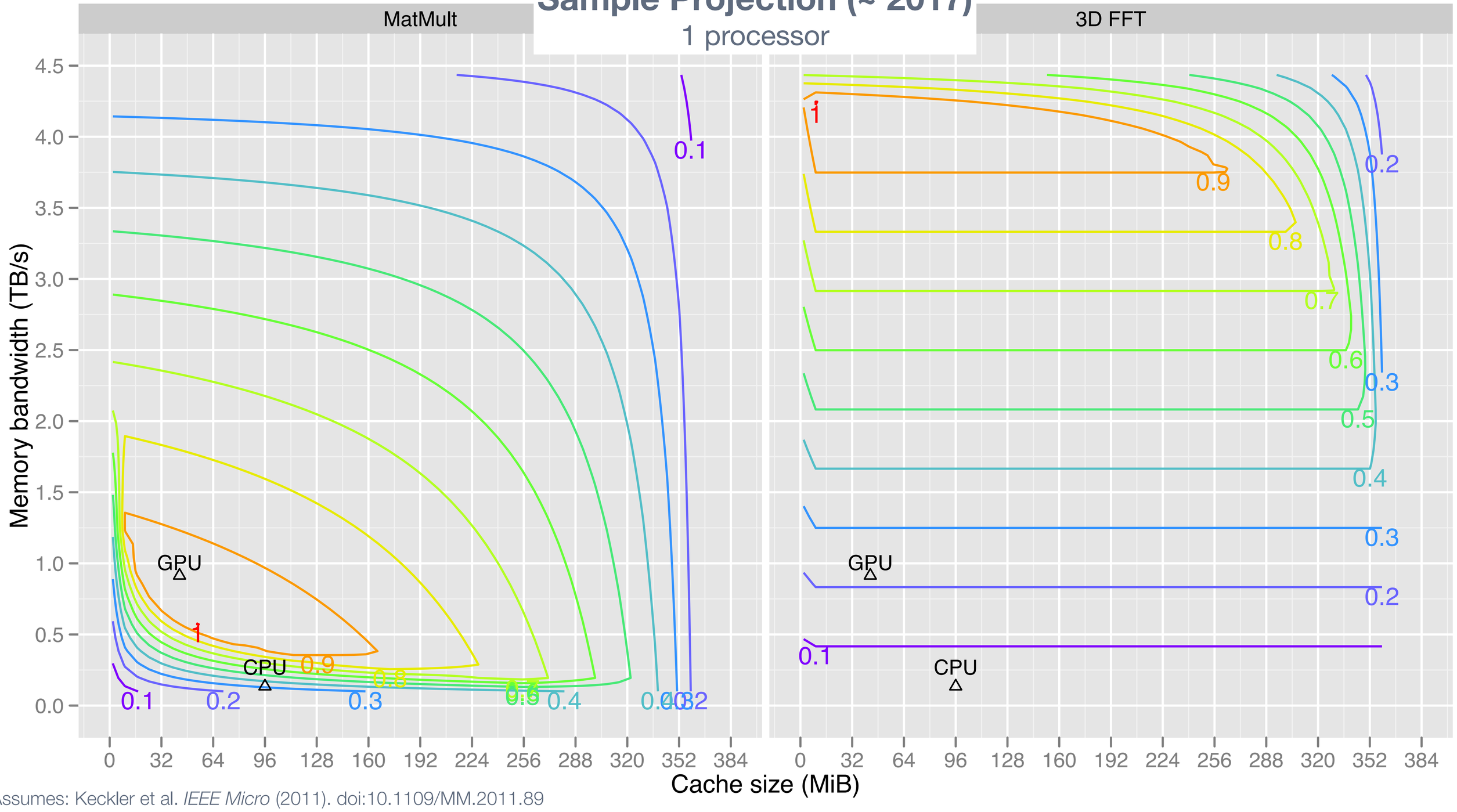
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

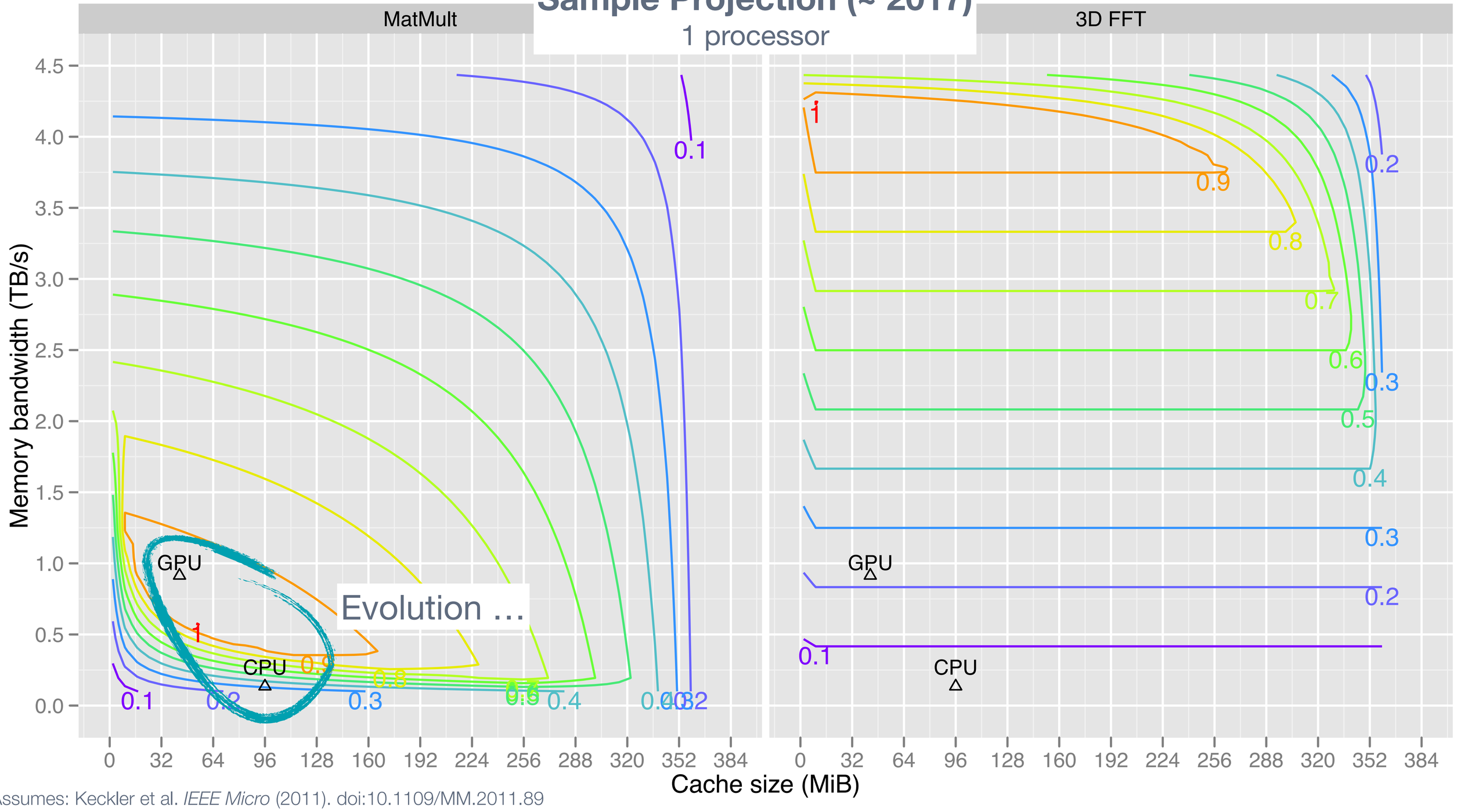
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

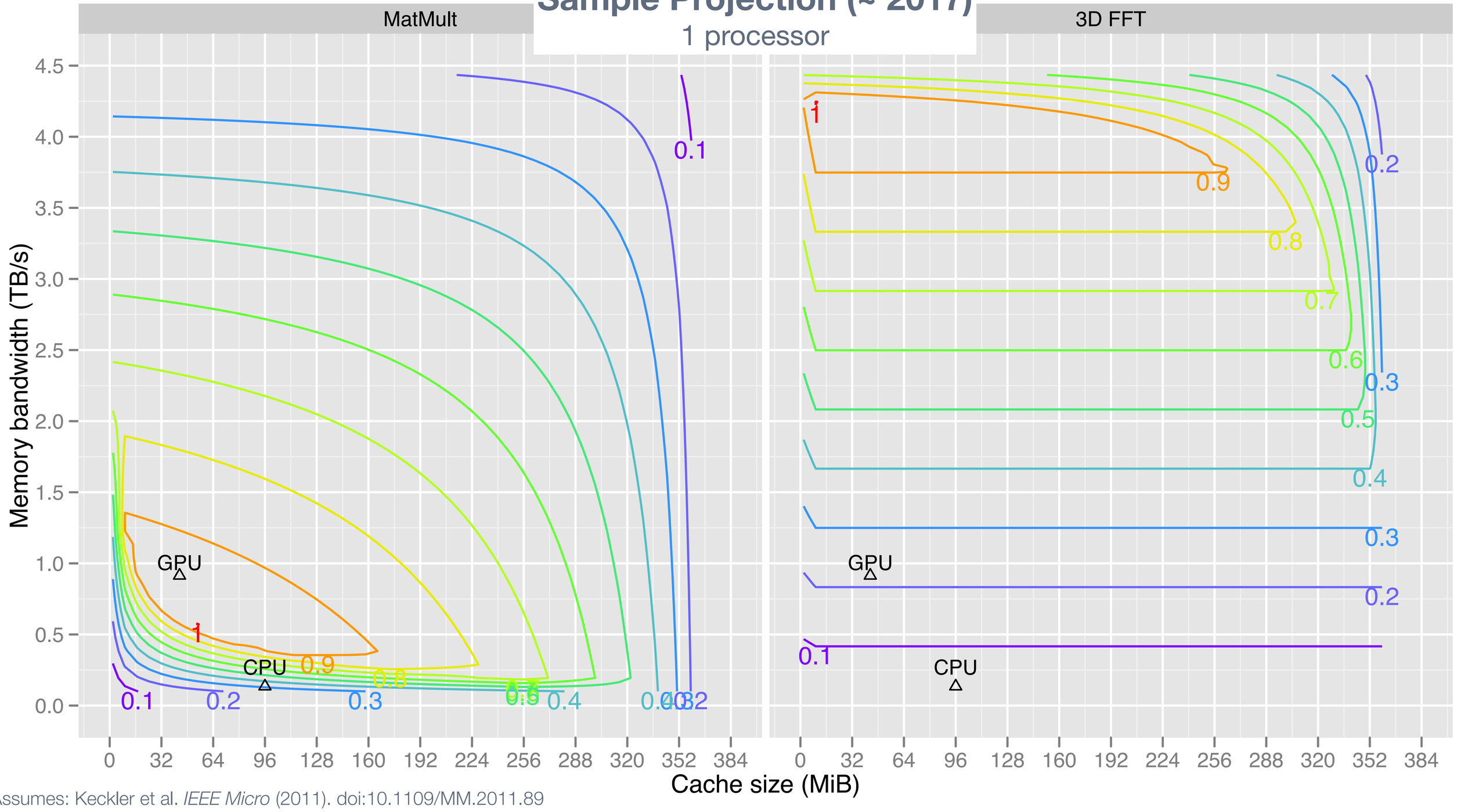
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

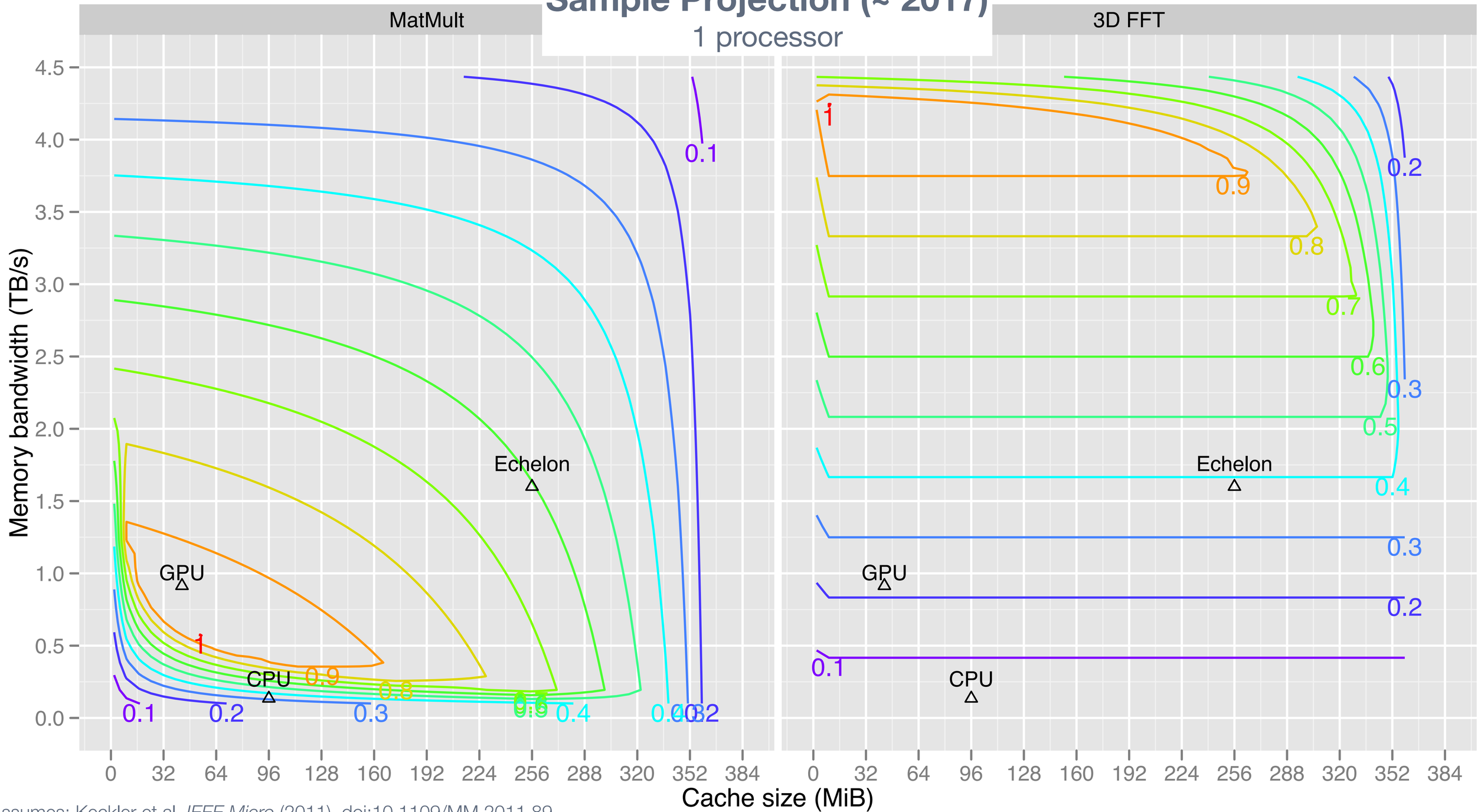
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

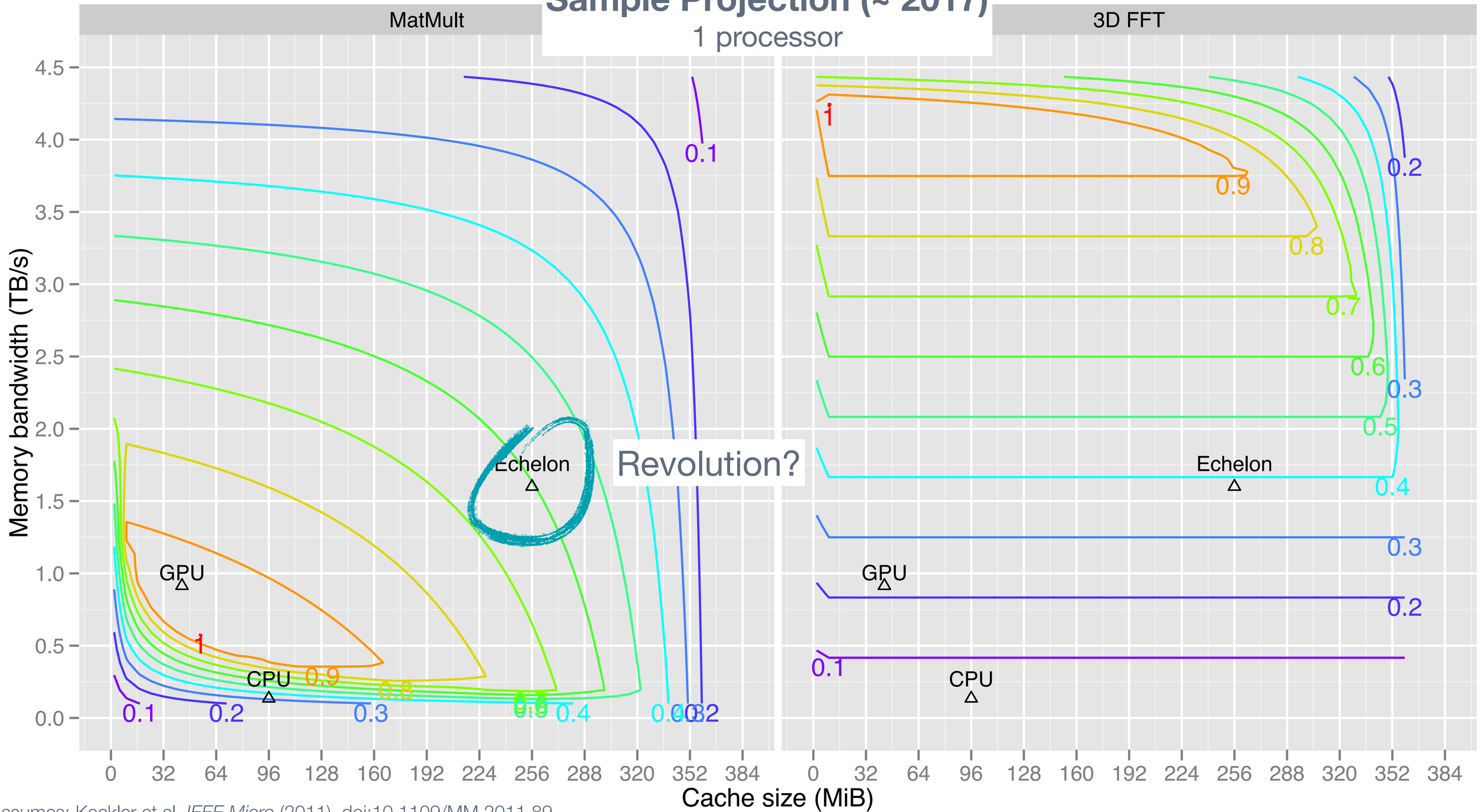
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor

MatMult

3D FFT

Memory bandwidth (TB/s)

Cache size (MiB)

Revolution?

> 2x

> 7x

Echelon

GPU

CPU

Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor

MatMult

3D FFT

Memory bandwidth (TB/s)

Cache size (MiB)

< ⅔x

~ 2x

Revolution?

GPU

CPU

Echelon

Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor

MatMult

3D FFT

Is this region interesting?

< ⅔x

~ 2x

Echelon

GPU

CPU

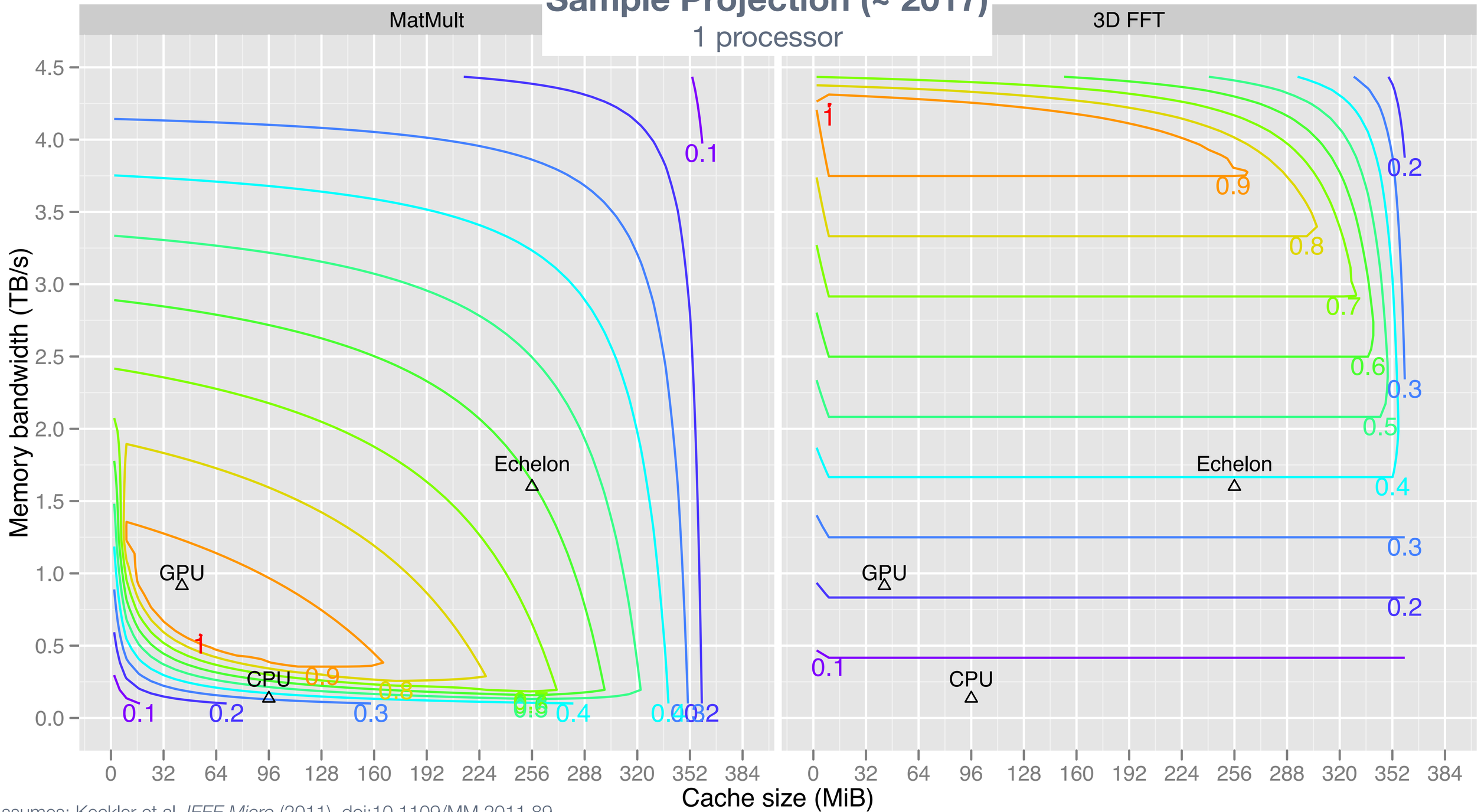
Memory bandwidth (TB/s)

Cache size (MiB)

Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

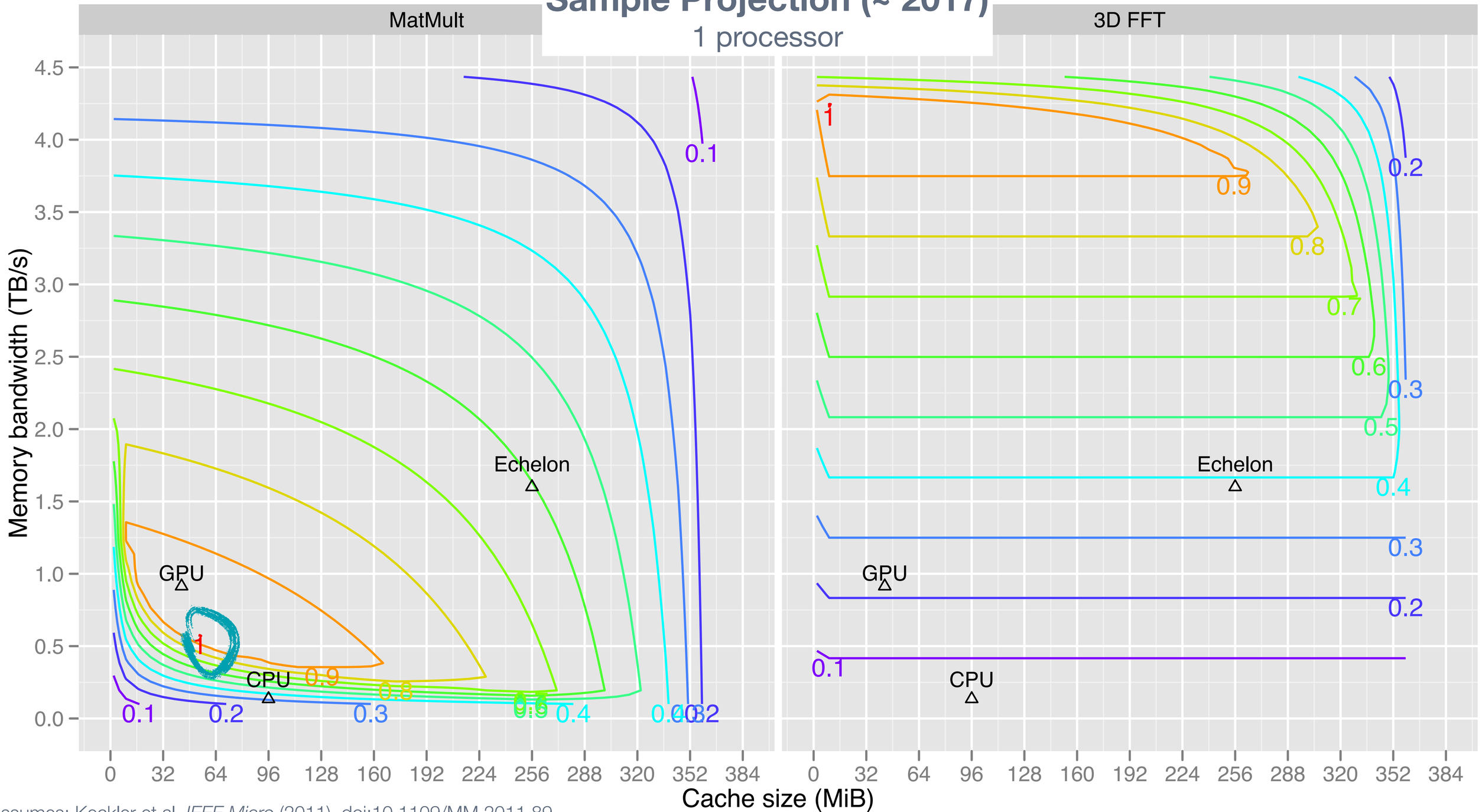
1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor



Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor

MatMult

3D FFT

Memory bandwidth (TB/s)

Cache size (MiB)

A single die configuration (cores v. LLC) might be fine …

Echelon

GPU

CPU

Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

Sample Projection (~ 2017)

1 processor

MatMult

3D FFT

Memory bandwidth (TB/s)

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

… *if* we could get “extreme” reconfigurability of **power**.

~ 8x

Echelon

GPU

CPU

1

0.9

0.8

0.7

0.6

0.5

0.4

0.3

0.2

0.1

0 32 64 96 128 160 192 224 256 288 320 352 384

Cache size (MiB)

Assumes: Keckler et al. *IEEE Micro* (2011). doi:10.1109/MM.2011.89

We can extend this analysis to a “full” system,
in which we consider, on-chip networks,
multiple nodes, network topology, …

Solutions to the constrained optimization problem

Matrix multiply — 8 EF/s peak (1M “nodes”)

3D FFT — 230 PF/s peak (4k “nodes”)

Solutions to the constrained optimization problem

Matrix multiply — 8 EF/s peak (1M “nodes”)

Relative to notional Echelon:
~ **5x faster** for MM
~ **0.9x as fast** for 3D FFT

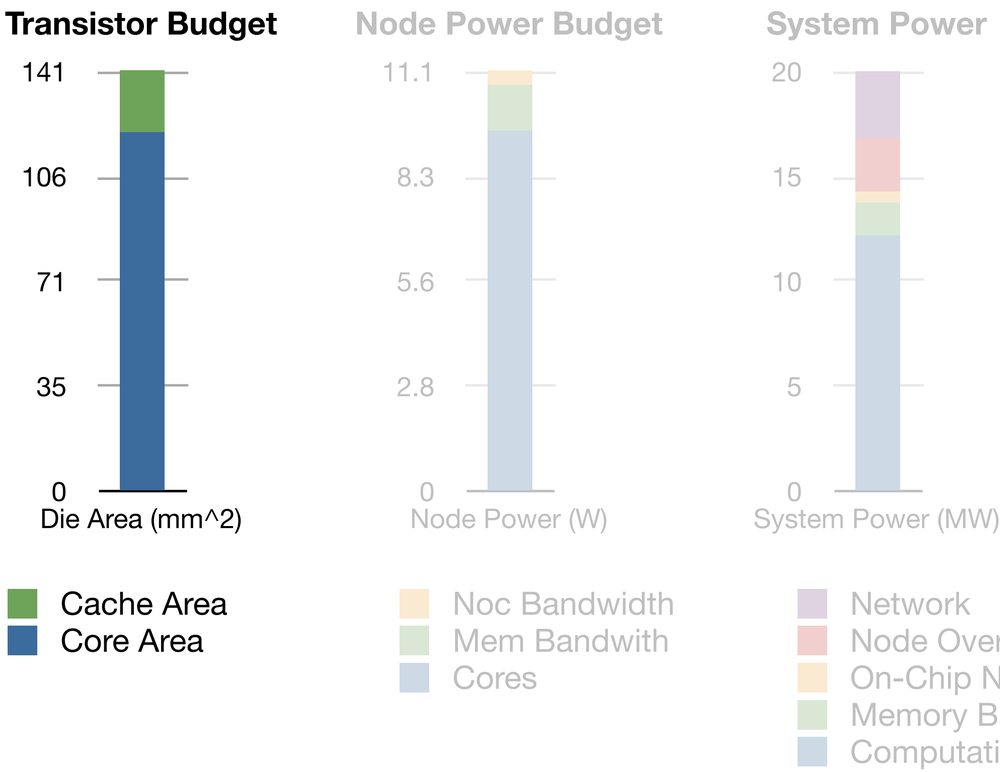
3D FFT — 230 PF/s peak (4k “nodes”)

Relative to notional Echelon:
~ **28x faster** for 3D FFTs
~ **0.14x as fast** for MM

Solutions to the constrained optimization problem

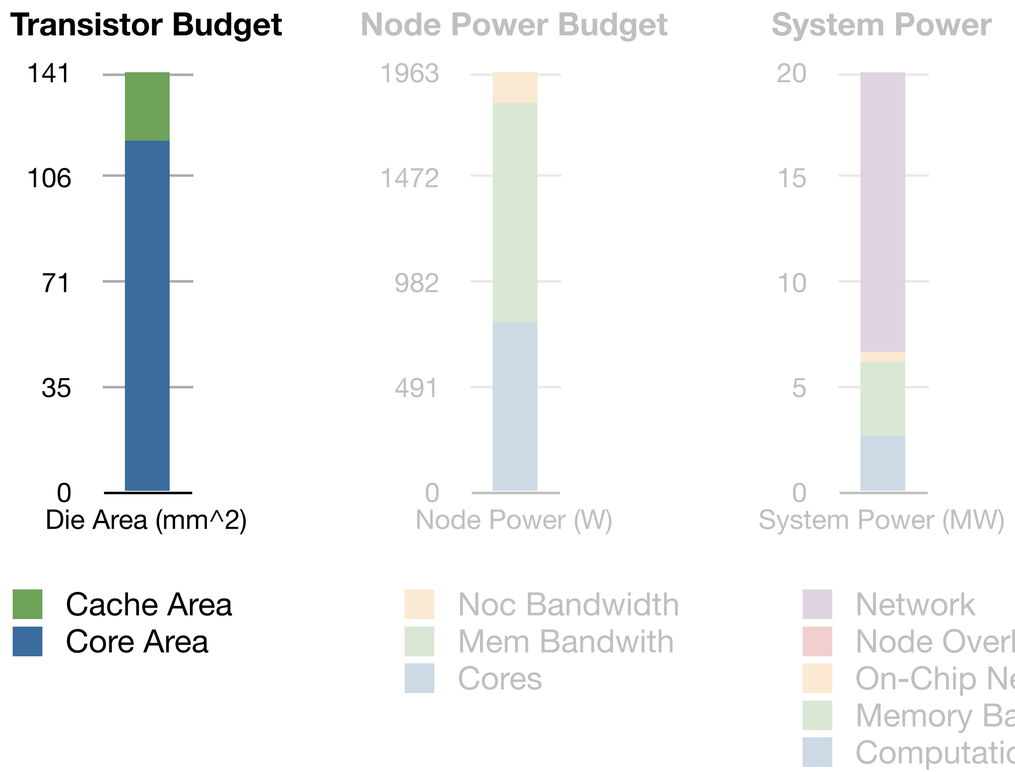
Matrix multiply — 8 EF/s peak (1M “nodes”)

Relative to notional Echelon:
~ **5x faster** for MM
~ **0.9x as fast** for 3D FFT



3D FFT — 230 PF/s peak (4k “nodes”)

Relative to notional Echelon:
~ **28x faster** for 3D FFTs
~ **0.14x as fast** for MM

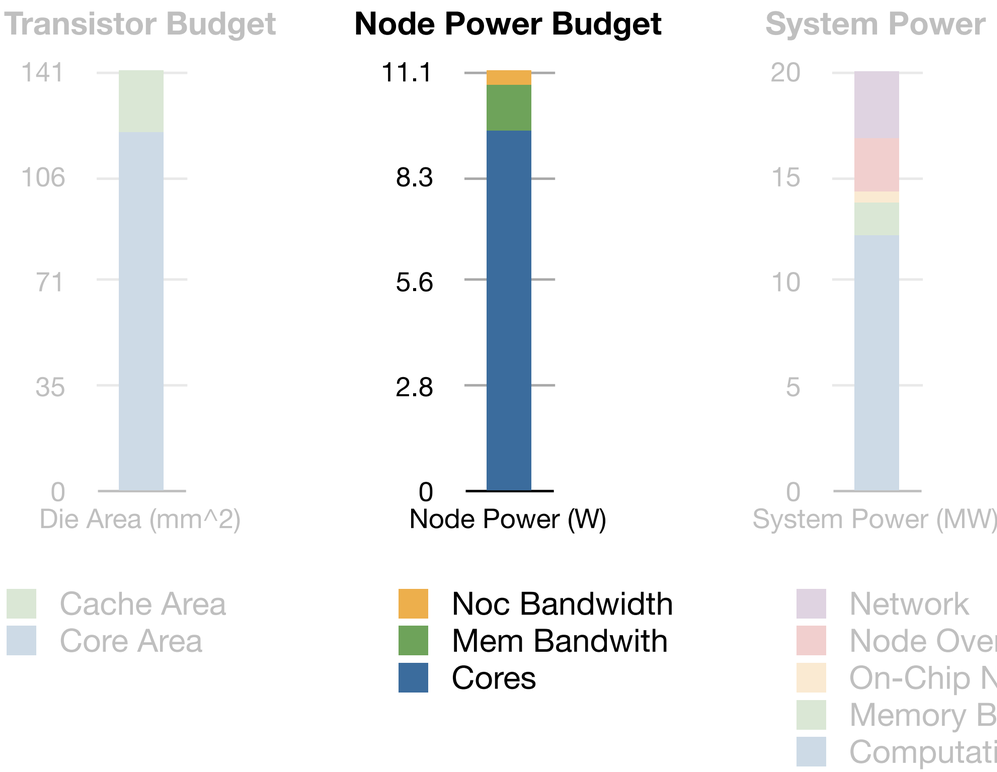


Plausibility issue: Unconstrained power density

Solutions to the constrained optimization problem

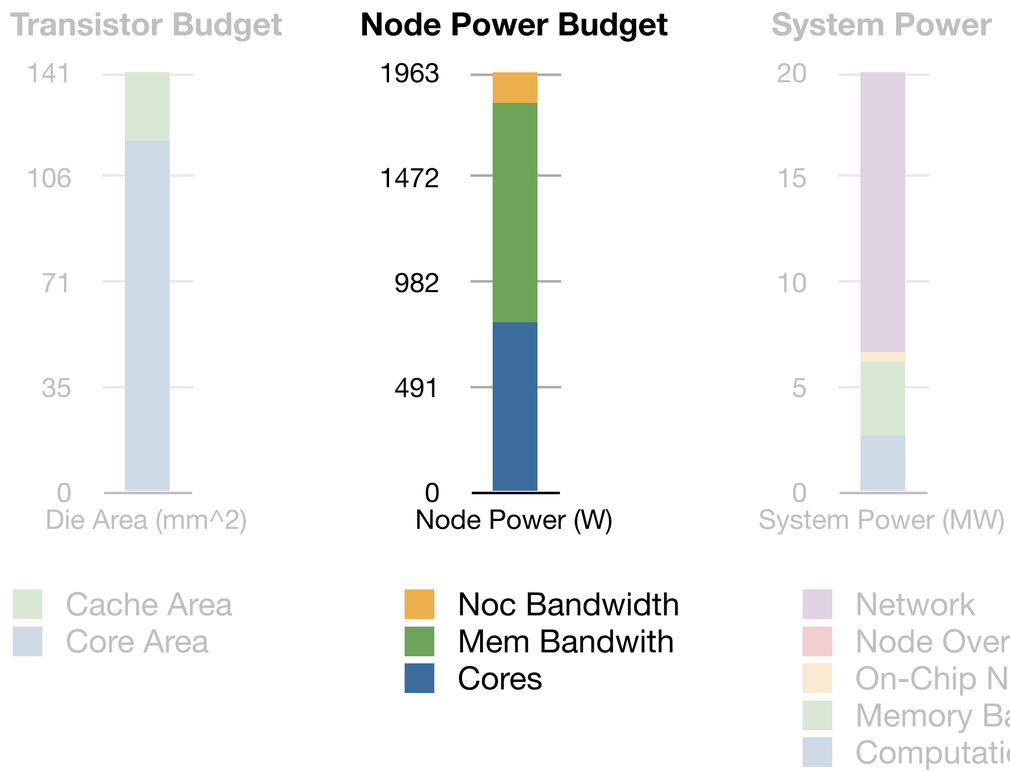
Matrix multiply — 8 EF/s peak (1M “nodes”)

Relative to notional Echelon:
~ **5x faster** for MM
~ **0.9x as fast** for 3D FFT



3D FFT — 230 PF/s peak (4k “nodes”)

Relative to notional Echelon:
~ **28x faster** for 3D FFTs
~ **0.14x as fast** for MM

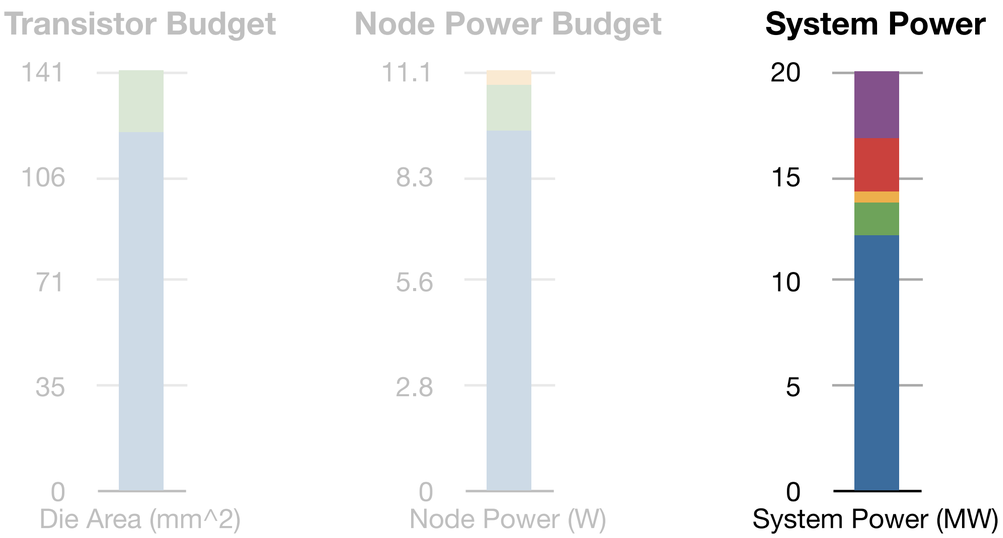


Plausibility issue: Unconstrained power density

Solutions to the constrained optimization problem

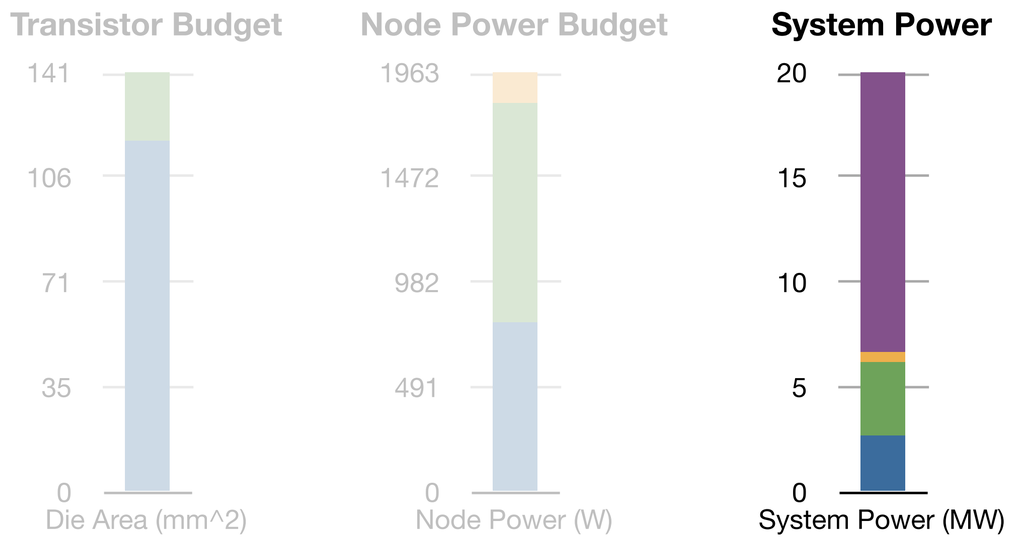
Matrix multiply — 8 EF/s peak (1M “nodes”)

Relative to notional Echelon:
~ **5x faster** for MM
~ **0.9x as fast** for 3D FFT



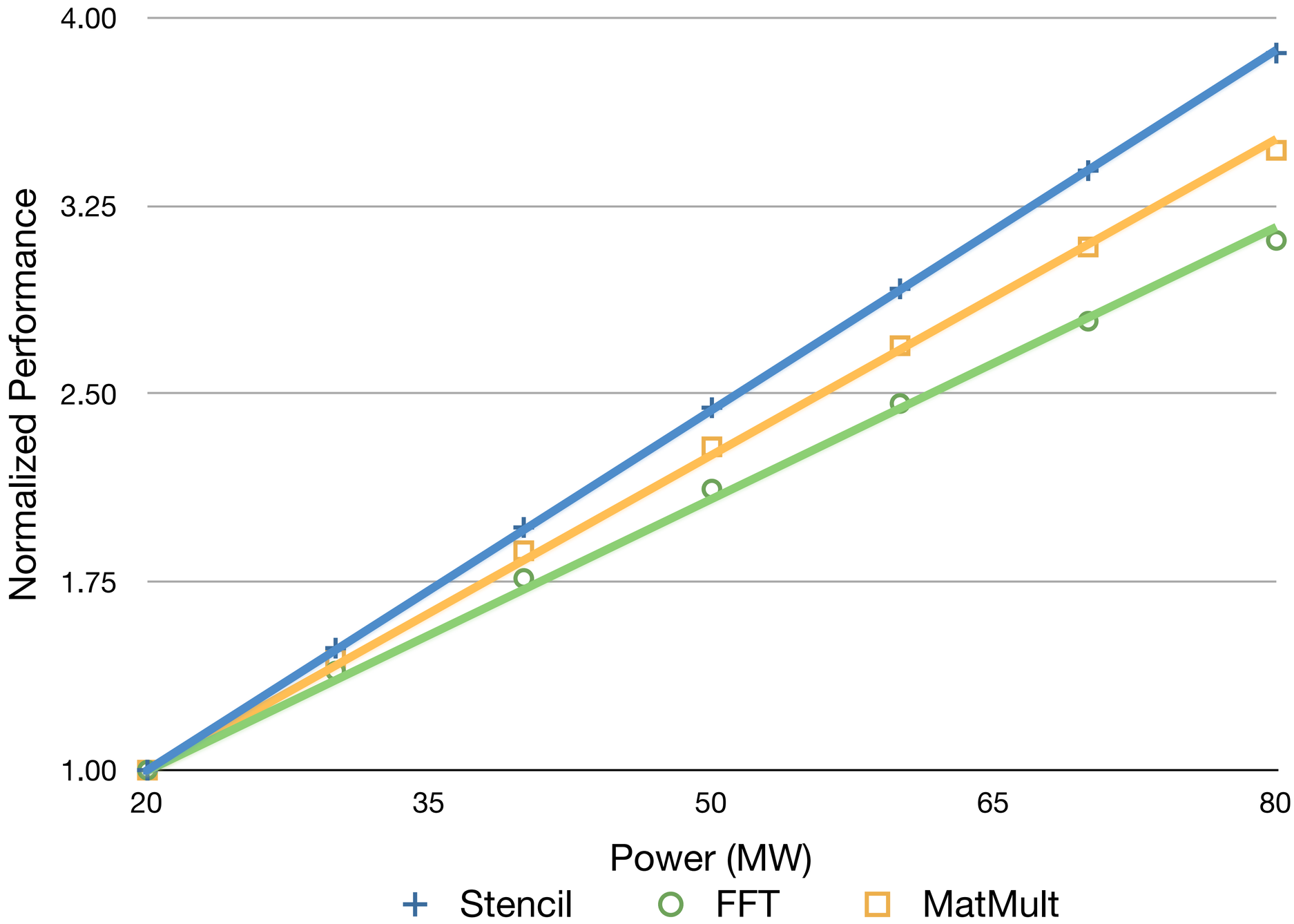
3D FFT — 230 PF/s peak (4k “nodes”)

Relative to notional Echelon:
~ **28x faster** for 3D FFTs
~ **0.14x as fast** for MM



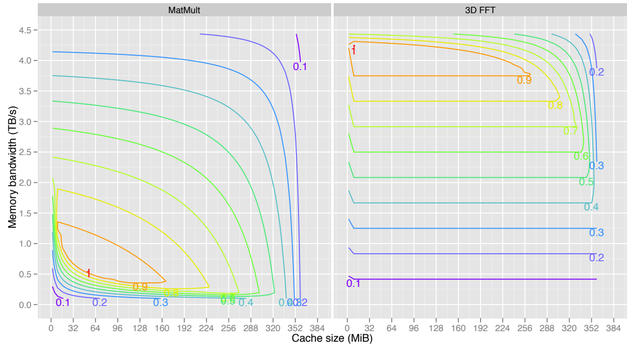
Plausibility issue: Unconstrained power density

Performance as a Function of System Power



Summary:

Are we looking in the right place in the space of "exascale-able" designs?



Can starting from algorithmic first-principles and minimizing time, with power and area as the principal constraints, identify other regions of interest?

Questions:

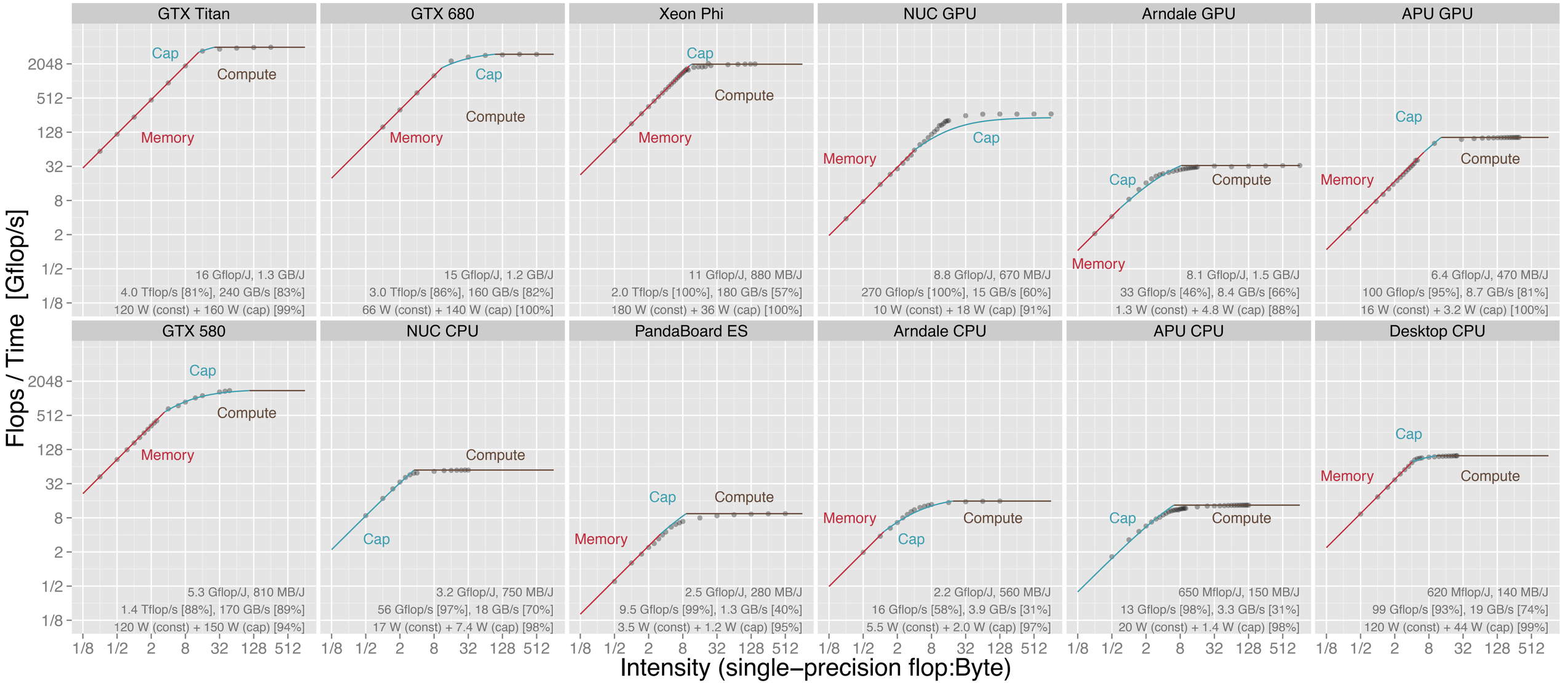
What workloads and algorithms matter most?

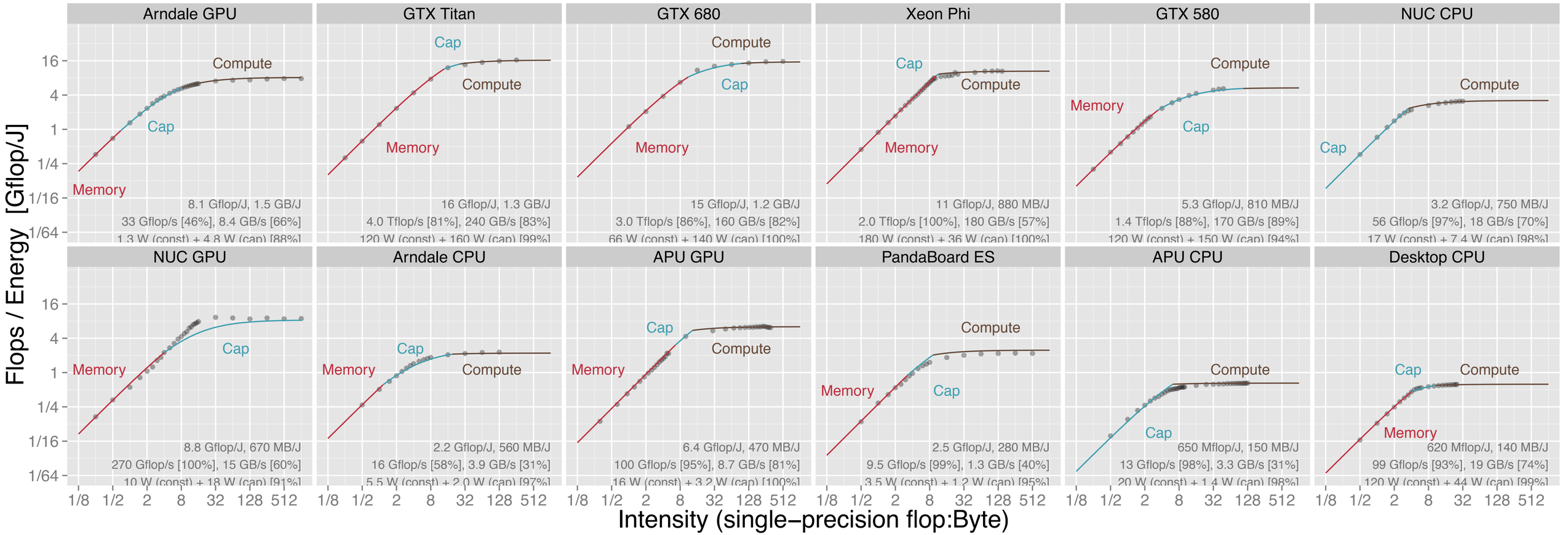
What algorithmic trade-offs can we analyze?

What architectural designs are feasible?

Can we estimate the impact of disruptive technologies?

Other junk


GTX Titan
GTX 680
Xeon Phi
NUC GPU
Arndale GPU
APU GPU
GTX 580
NUC CPU
PandaBoard ES
Arndale CPU
APU CPU
Desktop CPU
Cap
Compute
Memory
Flops / Time [Gflop/s]
2048
512
128
32
8
2
1/2
1/8
1/8 1/2 2 8 32 128 512
Intensity (single−precision flop:Byte)
16 Gflop/J, 1.3 GB/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]
15 Gflop/J, 1.2 GB/J
3.0 Tflop/s [86%], 160 GB/s [82%]
66 W (const) + 140 W (cap) [100%]
11 Gflop/J, 880 MB/J
2.0 Tflop/s [100%], 180 GB/s [57%]
180 W (const) + 36 W (cap) [100%]
8.8 Gflop/J, 670 MB/J
270 Gflop/s [100%], 15 GB/s [60%]
10 W (const) + 18 W (cap) [91%]
8.1 Gflop/J, 1.5 GB/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]
6.4 Gflop/J, 470 MB/J
100 Gflop/s [95%], 8.7 GB/s [81%]
16 W (const) + 3.2 W (cap) [100%]
5.3 Gflop/J, 810 MB/J
1.4 Tflop/s [88%], 170 GB/s [89%]
120 W (const) + 150 W (cap) [94%]
3.2 Gflop/J, 750 MB/J
56 Gflop/s [97%], 18 GB/s [70%]
17 W (const) + 7.4 W (cap) [98%]
2.5 Gflop/J, 280 MB/J
9.5 Gflop/s [99%], 1.3 GB/s [40%]
3.5 W (const) + 1.2 W (cap) [95%]
2.2 Gflop/J, 560 MB/J
16 Gflop/s [58%], 3.9 GB/s [31%]
5.5 W (const) + 2.0 W (cap) [97%]
650 Mflop/J, 150 MB/J
13 Gflop/s [98%], 3.3 GB/s [31%]
20 W (const) + 1.4 W (cap) [98%]
620 Mflop/J, 140 MB/J
99 Gflop/s [93%], 19 GB/s [74%]
120 W (const) + 44 W (cap) [99%]


Arndale GPU
Memory
Cap
Compute
8.1 Gflop/J, 1.5 GB/J
33 Gflop/s [46%], 8.4 GB/s [66%]
1.3 W (const) + 4.8 W (cap) [88%]
GTX Titan
Memory
Cap
Compute
16 Gflop/J, 1.3 GB/J
4.0 Tflop/s [81%], 240 GB/s [83%]
120 W (const) + 160 W (cap) [99%]
GTX 680
Memory
Cap
Compute
15 Gflop/J, 1.2 GB/J
3.0 Tflop/s [86%], 160 GB/s [82%]
66 W (const) + 140 W (cap) [100%]
Xeon Phi
Memory
Cap
Compute
11 Gflop/J, 880 MB/J
2.0 Tflop/s [100%], 180 GB/s [57%]
180 W (const) + 36 W (cap) [100%]
GTX 580
Memory
Cap
Compute
5.3 Gflop/J, 810 MB/J
1.4 Tflop/s [88%], 170 GB/s [89%]
120 W (const) + 150 W (cap) [94%]
NUC CPU
Cap
Compute
3.2 Gflop/J, 750 MB/J
56 Gflop/s [97%], 18 GB/s [70%]
17 W (const) + 7.4 W (cap) [98%]
NUC GPU
Memory
Cap
8.8 Gflop/J, 670 MB/J
270 Gflop/s [100%], 15 GB/s [60%]
10 W (const) + 18 W (cap) [91%]
Arndale CPU
Memory
Cap
Compute
2.2 Gflop/J, 560 MB/J
16 Gflop/s [58%], 3.9 GB/s [31%]
5.5 W (const) + 2.0 W (cap) [97%]
APU GPU
Memory
Cap
Compute
6.4 Gflop/J, 470 MB/J
100 Gflop/s [95%], 8.7 GB/s [81%]
16 W (const) + 3.2 W (cap) [100%]
PandaBoard ES
Memory
Cap
Compute
2.5 Gflop/J, 280 MB/J
9.5 Gflop/s [99%], 1.3 GB/s [40%]
3.5 W (const) + 1.2 W (cap) [95%]
APU CPU
Cap
Compute
650 Mflop/J, 150 MB/J
13 Gflop/s [98%], 3.3 GB/s [31%]
20 W (const) + 1.4 W (cap) [98%]
Desktop CPU
Cap
Memory
Compute
620 Mflop/J, 140 MB/J
99 Gflop/s [93%], 19 GB/s [74%]
120 W (const) + 44 W (cap) [99%]
Flops / Energy [Gflop/J]
16
4
1
1/4
1/16
1/64
1/8
1/2
2
8
32
128
512
Intensity (single−precision flop:Byte)

$\pi_0 + \Delta\pi$: max power

$\Delta\pi$: **usable**

π_0 : constant

Caps imply throttling!

Peak power per flop (or **mop**)

$$\pi_{\text{flop}} \equiv \frac{\epsilon_{\text{flop}}}{\tau_{\text{flop}}}$$

$$\pi_{\text{mem}} \equiv \frac{\epsilon_{\text{mem}}}{\tau_{\text{mem}}}$$

$$\Downarrow$$

$$\Delta\pi \geq \pi_{\text{flop}} + \pi_{\text{mem}} = \pi_{\text{flop}}\left(1 + \frac{B_\epsilon}{B_\tau}\right)$$

What power cap will obviate throttling?
The *balance gap* dictates a sufficient (algorithm-independent) condition.

July 2008 US DOE CScADS Meeting on Autotuning

http://cscads.rice.edu/workshops

July 2008 US DOE CScADS Meeting on Autotuning

http://cscads.rice.edu/workshops

July 2008 US DOE CScADS Meeting on Autotuning

http://cscads.rice.edu/workshops

http://goo.gl/maps/ICPt