Balance Principles for Algorithm-Architecture Co-design

Kent Czechowski, Casey Battaglino, Chris McClanahan,
Aparna Chandramowlishwaran, Richard Vuduc
(Georgia Tech)

May 31, 2011

Position

Position: Principles (i.e, "theory") informing practice (co-design)

Position

Position: Principles (i.e, "theory") informing practice (co-design)

Hardware/Software Co-design?
Algorithm-Architecture Co-design?

Position

Position: Principles (i.e, "theory") informing practice (co-design)
For some computation to scale efficiently on a future parallel processor:

1. Allocation of cores?
2. Allocation of cache?
3. How must latency/bandwidth increase to compensate?

Or alternatively, given a particular parallel architecture, what classes of computations will perform efficiently?

Why theoretical models?

The best alternative (and perhaps the "status quo") in co-design is to put together a model of your chip and simulate your algorithm.

Very accurate, but by this point you've already invested lots of time and effort into a specific design.

Why theoretical models?

We advocate a more principled approach that can model the performance of a processor based on some of its most high-level characteristics known to be the main bottlenecks (communication, parallel scalability)...

Such a model can be refined and extended as needed, i.e based on cache characteristics, heterogeneity of the cores

Balance

We define balance as:

$$\text{For some } \textit{algorithm}\text{: } T_{mem} \leq T_{comp}$$

[1]

For principled analysis, we need theoretical models for T_{mem}, T_{comp}
To be relevant for current/future processors, these models must integrate:

1. Parallelism
2. Cache/Memory Locality

[1] Similar to classical notions of balance: [Kung 1986], [Callahan, et al 1988], [McCalpin 1995]

Why Balance?

Importance of considering balance:

1. Inevitable trend towards imbalance: peak flops outpacing memory hierarchy.
2. Imbalance may be nonintuitive (make an improvement to some aspect of a chip without realizing that other areas must also improve to compensate) — for a particular algorithm

Why Balance?

Balance is a particularly powerful lens for maintaining more *realistic* expectations for performance. Processor makers present raw figures for performance: peak flops, memory specs– very one-dimensional figures on their own. (i.e CPU vs. GPU wars)

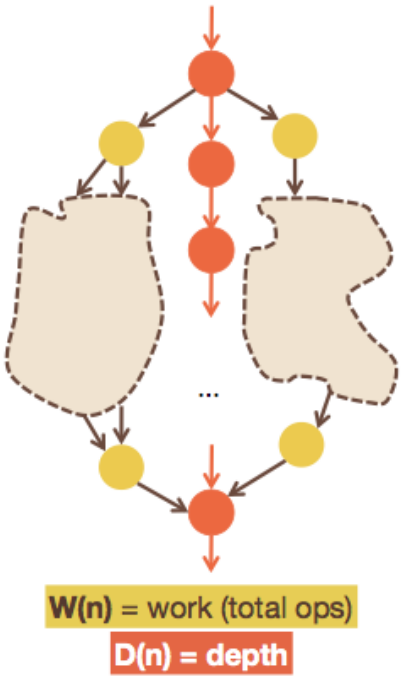
Balance marries the two in a way that allows parallel scalability to also enter the picture– and recognizes that not all architectures are suitable for all applications.

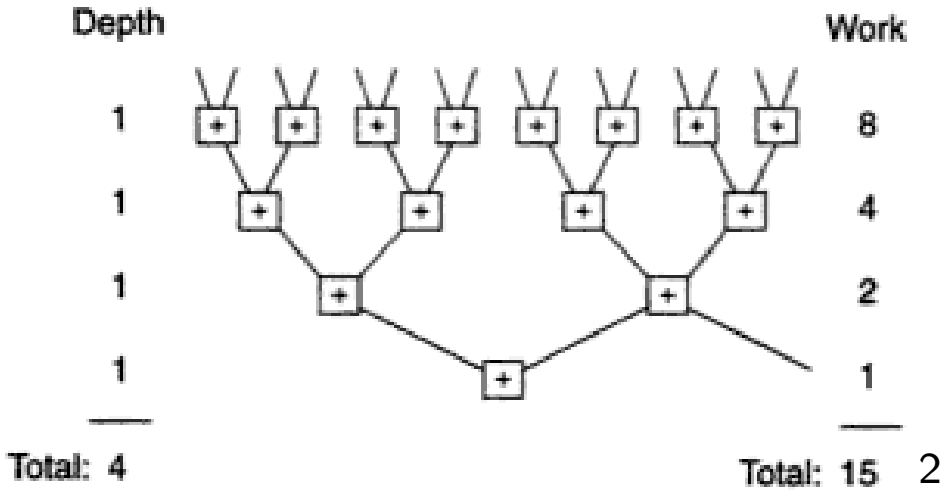
Assumptions

For our particular "principled" approach we use two models:
T_{mem}: External Memory Model (I/O Model)
T_{comp}: Parallel DAG Model / Work-Depth Model
For these models alone to be expressive we have assumptions...

1. We are modeling work on a single socket. n is large enough to not fit completely in the outer level of cache.
2. For our algorithm, we can easily deduce the structure of a dependency DAG for any n
3. The developer can overlap computation and communication arbitrarily well
4. Communication costs are dominated by misses between cache and RAM($\therefore T_{comm} \propto$ cache misses $=$ Q(n)).

Parallel DAG Model for T_{comp} $(T_{mem} \leq T_{comp})$





Inherent parallelism: $\frac{W(n)}{D(n)}$ . . . spectrum between embarrassingly parallel and inherently sequential (application: CPA)

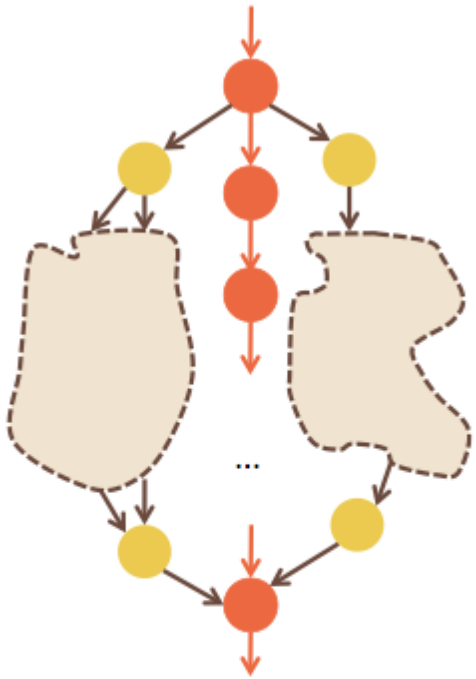
Desired: work optimality, maximum parallelism

[2]Source: Blelloch: Parallel Algorithms

Parallel DAG Model for T_{comp} $(T_{mem} \leq T_{comp})$

Brents Theorem [1974]: Maps DAG model to PRAM model

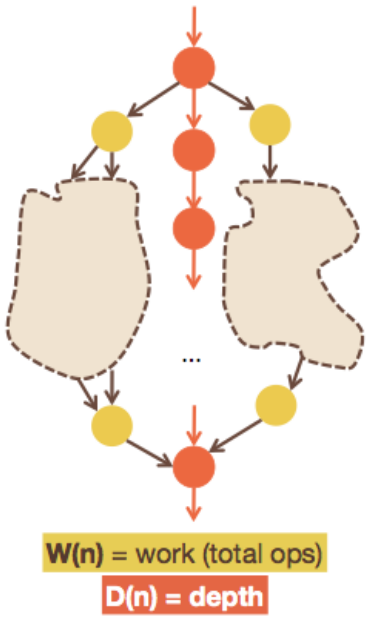
$$T_p(n) = O(D(n) + \frac{W(n)}{p})$$



Parallel DAG Model for T_{comp} $(T_{mem} \leq T_{comp})$

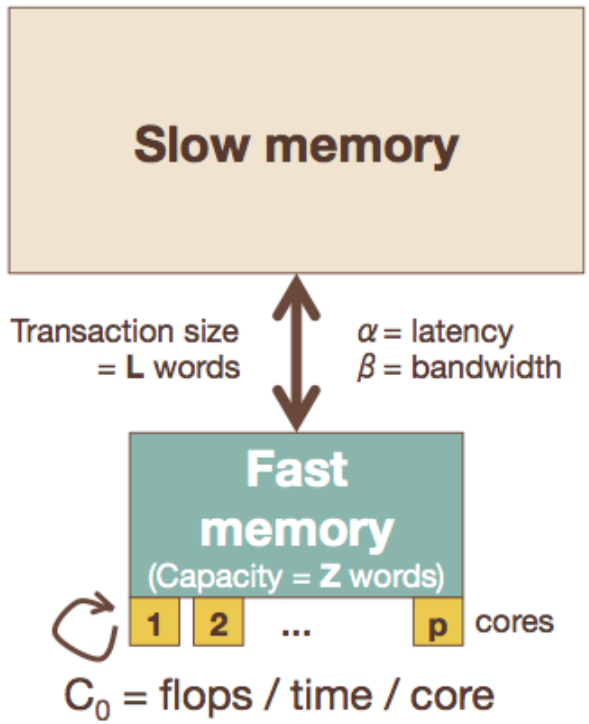
We model T_{comp} with:

$$T_{comp}(n; p, C_0) = (D(n) + \frac{W(n)}{p}) \cdot \frac{1}{C_0}$$



This gives us a lower bound that an optimally-crafted algorithm could theoretically achieve.

I/O Model for T_{mem} $(T_{mem} \leq T_{comp})$



$Q(n; Z, L)$: Number of cache misses.
Thus, the volume of data transferred is $Q(n; Z, L) \times L$

I/O Model for T_{mem} $(T_{mem} \leq T_{comp})$

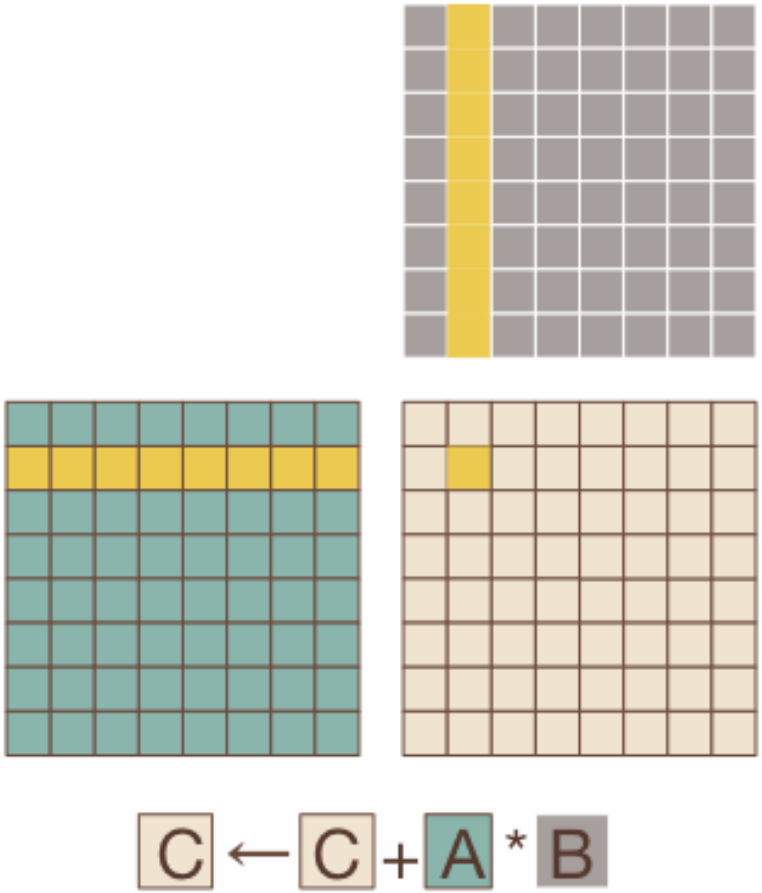
Our *intensity* is thus

$$\frac{W(n)}{Q(n; Z, L) \times L}$$

Desired: minimize work (work-optimality) while maximizing intensity (by minimizing cache complexity).

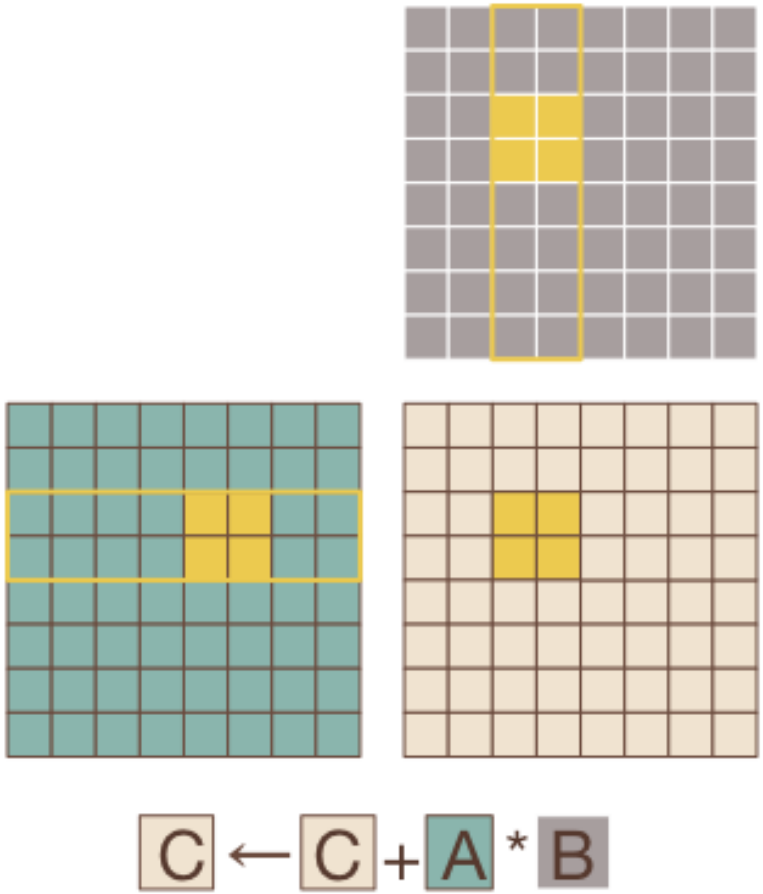
Intensity on its own is very descriptive: intuitively we know that high-intensity operations such as matrix multiply perform well on GPUs, whereas low-intensity vector operations perform poorly. "W" and "Q" underly this behavior

I/O Model: Matrix Multiply



$$\underbrace{\frac{W(n)}{Q(n;Z,L)\cdot L} = \Theta(1)}_{\textbf{Intensity}}$$

I/O Model: Matrix Multiply



$$Q(n; Z, L) = \Omega\left(\frac{n^3}{L\sqrt{Z}}\right)$$

Assumes **contiguous layout.**
Result is **optimal.**

$$\underbrace{\frac{W}{Q \cdot L} = \mathcal{O}\left(\sqrt{Z}\right)}_{\textbf{Intensity}}$$

I/O Model for T_{mem} $(T_{mem} \leq T_{comp})$

We model T_{mem} with:

$$T_{mem}(n; p, Z, L, \alpha, \beta) = \alpha \cdot D(n) + \frac{Q_{p;Z,L}(n) \cdot L}{\beta}$$

Q . . . # of cache misses

C_0 . . . # of cycles per second

p . . . # of cores

Z . . . cache size (bytes)

L . . . line size (bytes)

α . . . latency (s)

β . . . bandwidth (bytes/s)

I/O Model for T_{mem} $\quad (T_{mem} \leq T_{comp})$

We model T_{mem} with:

$$T_{mem}(n; p, Z, L, \alpha, \beta) = \underline{\alpha \cdot D(n)} + \frac{Q_{p;Z,L}(n) \cdot L}{\beta}$$

Q_1, sequential cache complexity, is well known for most algorithms. Q_p, parallel cache complexity, must be separately derived, but can be directly obtained from Q_1 if certain *scheduling* principles are followed.

I/O Model for T_{mem} $(T_{mem} \leq T_{comp})$

We model T_{mem} with:

$$T_{mem}(n; p, Z, L, \alpha, \beta) = \alpha \cdot D(n) + \frac{Q_{p;Z,L}(n) \cdot L}{\beta}$$

Example: Work-stealing + core-private caches.

$$Q_p(n; Z, L) < Q_1(n; Z, L) + \mathcal{O}\left(\frac{p \cdot Z \cdot D(n)}{L}\right)$$

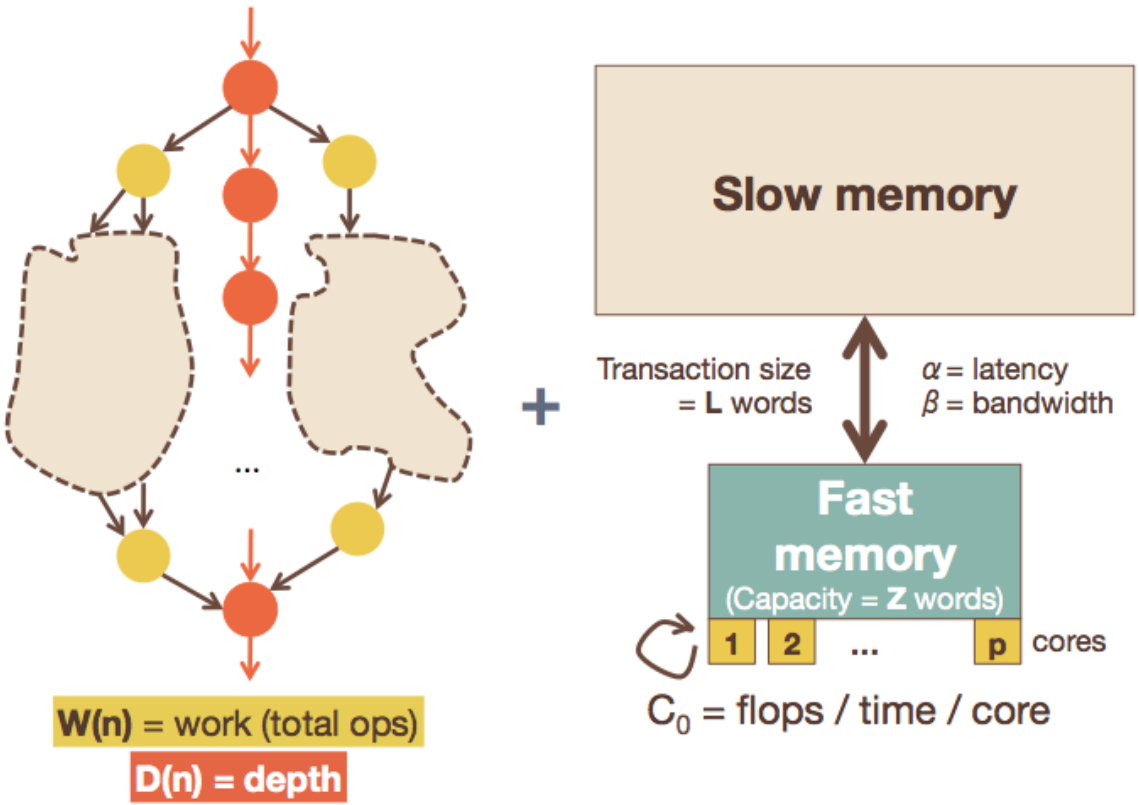
Example: Parallel depth-first + all-cores shared caches.

$$Q_p(n; Z + p \cdot L \cdot D(n), L) < Q_1(n; Z, L)$$

3

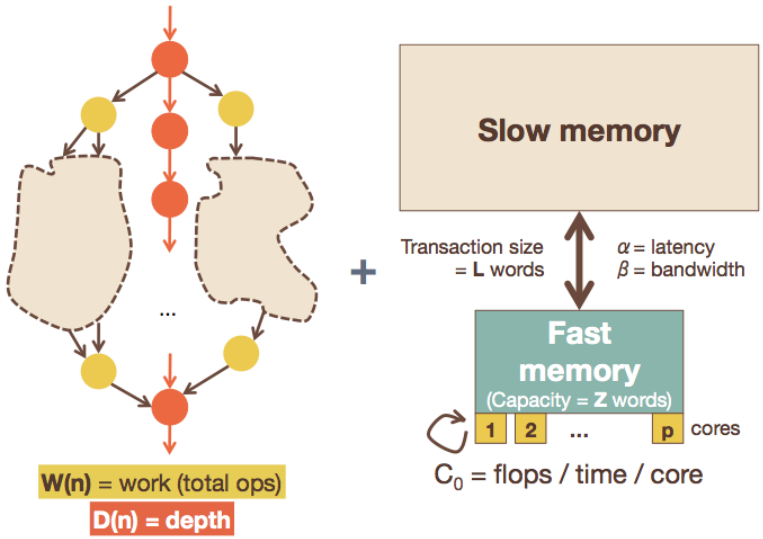
[3] Blelloch, Gibbons, Simhadri (2010). Low-depth cache-oblivious algorithms.

T_{comp}, T_{mem}



$$T_{mem} \leq T_{comp}$$

T_{comp}, T_{mem}: After some algebra



$$\frac{p \cdot C_0}{\beta/L}\left(1+\frac{\alpha\beta/L}{Q_{p;Z,L}/D}\right) \leq \frac{W}{Q_{p;Z,L}L}\left(1+\frac{p}{W/D}\right)$$

Balance | Little's Law | Intensity | Amdahl's Law

$$T_{mem} \leq T_{comp}$$

Projections

Irony, et. al: Parallel Matrix Multiply Bound:

$$Q_{p;Z,L}(n) \geq \frac{W(n)}{\sqrt{2} \cdot L\sqrt{Z/p}}$$

$$\therefore$$

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O}\left(\sqrt{\frac{Z}{p}}\right)$$

Example: **Matrix-multiply + work-stealing**

Projections

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O}\left(\sqrt{\frac{Z}{p}}\right)$$

Example: **Matrix-multiply + work-stealing**

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O}\left(\log \frac{Z}{p}\right)$$

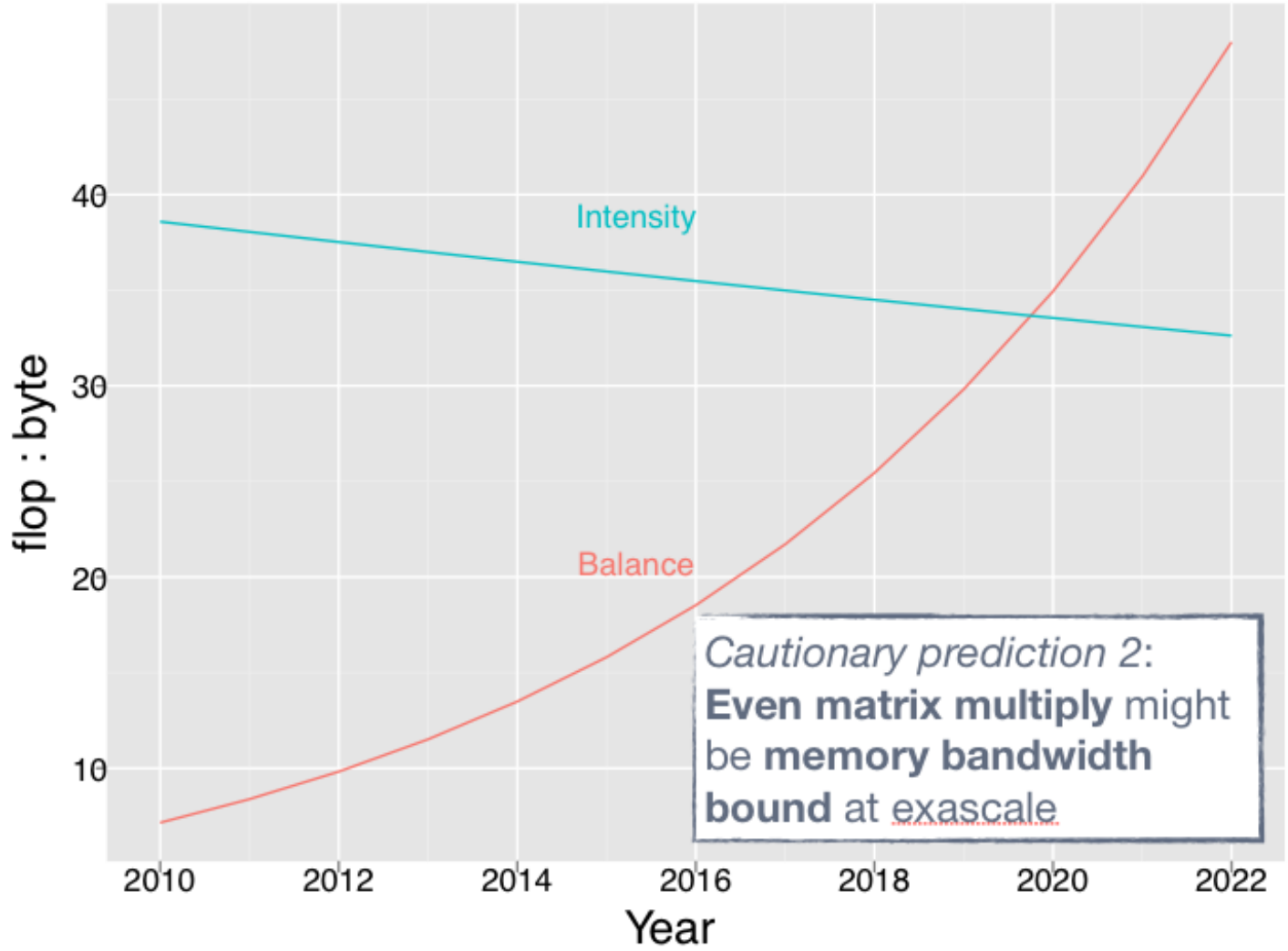
Example: **Cache-oblivious comparison-based sorting* + work-stealing**

Sort: the deterministic cache-oblivious algorithm by Blelloch (SPAA10) in which $W = n \log n, D = (\log n)^2, Q = n/L \times \log_Z(n)$.

"Punchline": Projections (Matrix Multiply)

Parameter	$t = 0$ NVIDIA Fermi C2050	CPU doubling time years	10-year projection
Peak flops, $p \cdot C_0$	1.03 Tflop/s	1.7	59 Tflop/s
Peak bandwidth, β	144 GB/s	2.8	1.7 TB/s
Latency, α	347.8 ns	10.5*	179.7 ns
Transfer size, L	128 Bytes	10.2	256 Bytes
Fast memory, Z	2.7 MB	2.0	83 MB
Cores, p	448	1.87	18k
$p \cdot C_0/\beta$	7.2	—	34.9
$\sqrt{Z/p}$	38.6	—	33.5

Projections (Matrix Multiply)



Consequences (Stacked Memory)

Scaling the number of PINs from memory to the processor with the *surface area* of the chip rather than the perimeter: β scales at a higher dimension.

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O}\left(\sqrt{\frac{Z}{p}}\right)$$

Example: **Matrix-multiply + work-stealing**

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O}\left(\log \frac{Z}{p}\right)$$

Example: **Cache-oblivious comparison-based sorting* + work-stealing**

Limitations

Big-Oh Notation

Existing analysis is often ($\approx$ always) in "Big-Oh" notation.
So W, D, Q are often in the form $O(f(n))$. For large n,

$$O(f(n)) \approx C \cdot f(n)$$

C can sometimes be determined from principles, or from static/dynamic analysis, or simply from benchmarking.
i.e, for FFT, $W(n) = \#flops = 5(n \log n)$

Limitations

Every model has limitations. We use the DAG model and External Memory model.

T_{comp} and T_{mem} can be changed to *any* model that aims to represent memory and compute time independently, i.e if there is a more suitable or predictable model on a particular architecture or algorithm. Example: increasingly heterogeneous chips (many more degrees of freedom).

We believe that *balance* is an ideal frame from which to focus this principled analysis: $T_{mem} \leq T_{comp}$

Limitations

How can we bring other metrics into play?

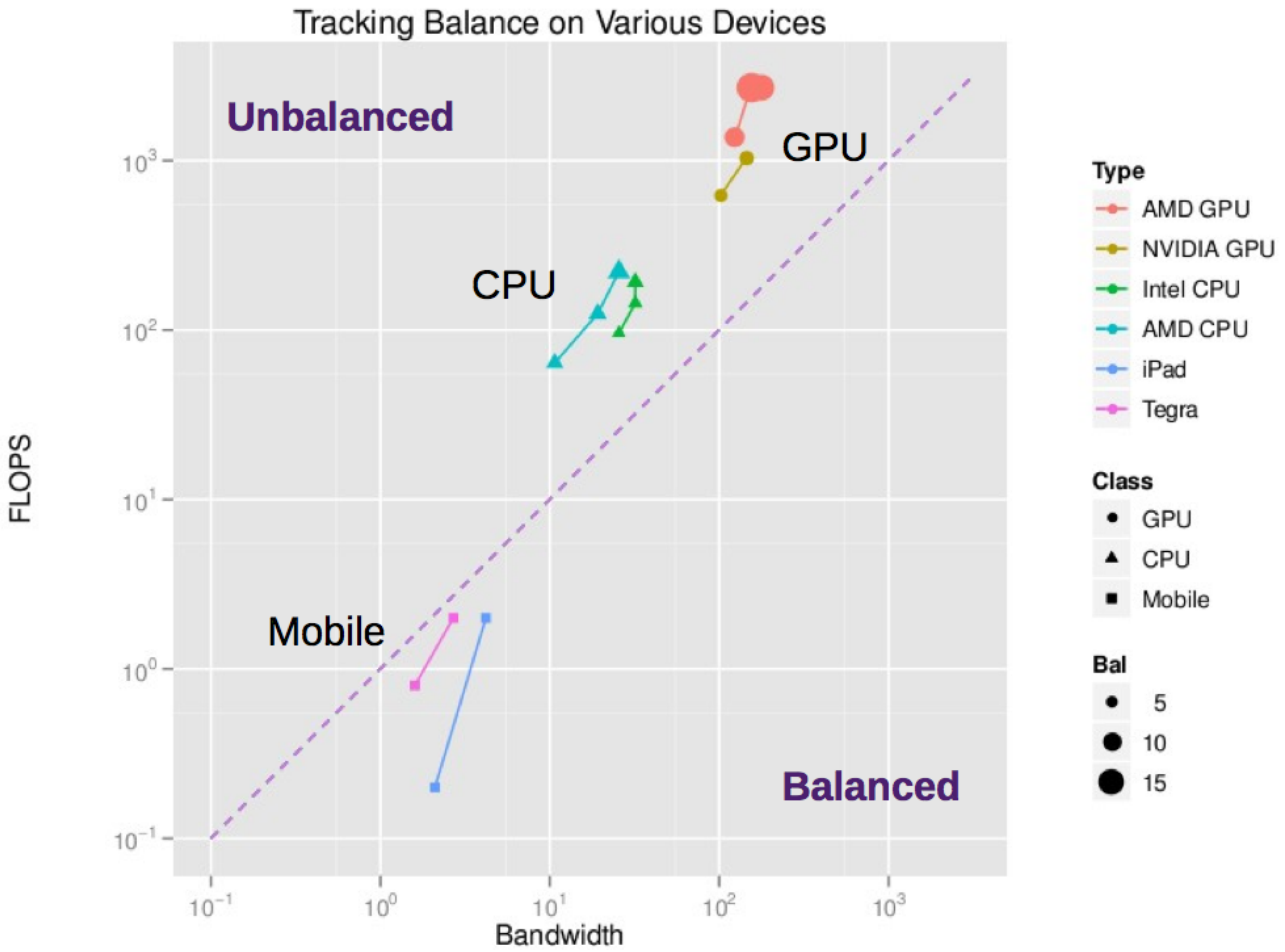
1. Power: $Power_{alg}(n; Z, L, p) \propto Q(n; Z, L, p)$?
   Power efficiency necessary for exascale
2. A more general cost metric
   (i.e a cluster of iPads would probably be balanced)

Bounds

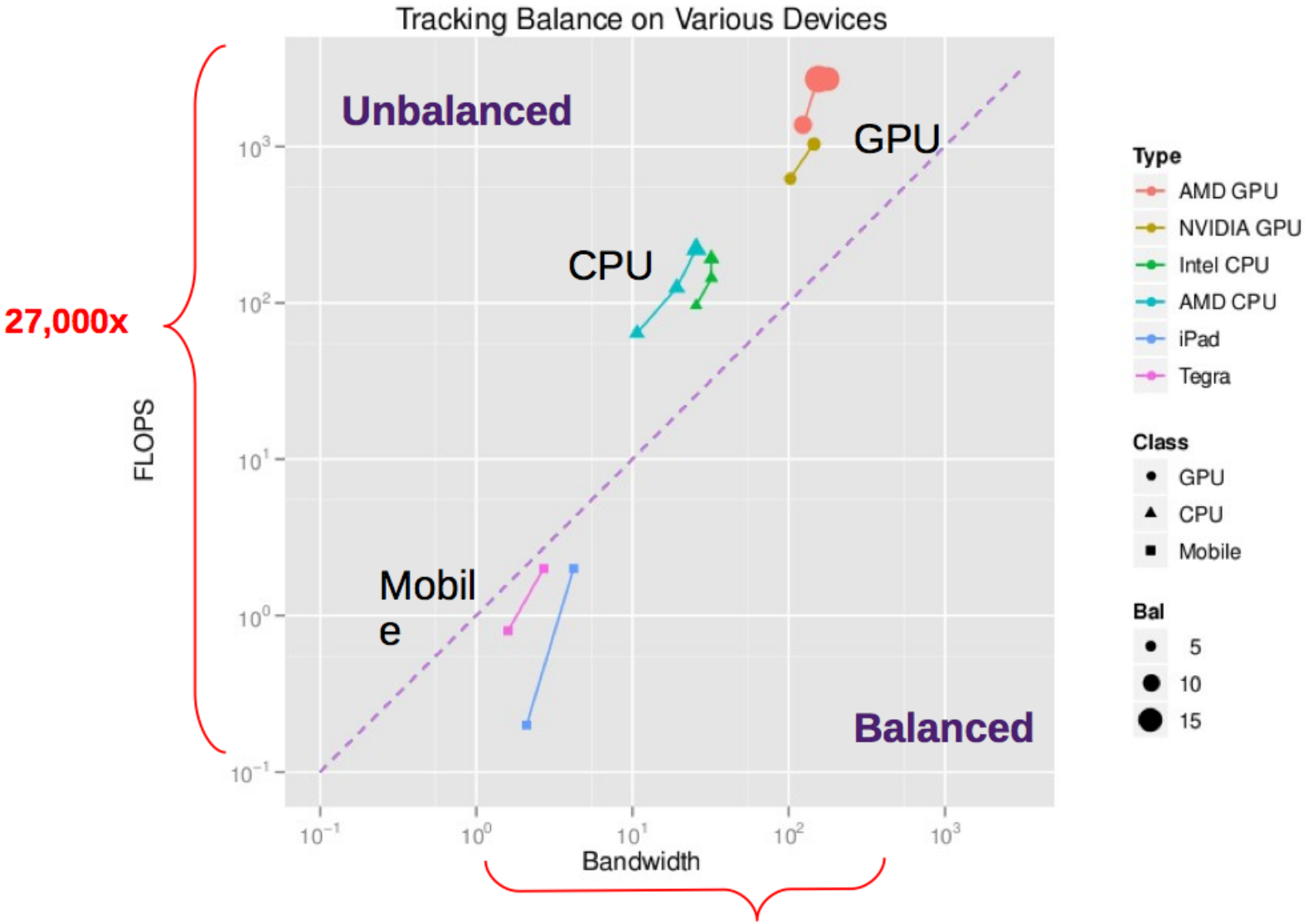
	Lower bound		Upper bound	
Algorithm	Bandwidth	Latency	Bandwidth	Latency
Matrix-Multiplication	$\Omega\left(\frac{n^3}{P\sqrt{M}}\right)$ $= \Omega\left(\frac{n^2}{\sqrt{P}}\right)$	$\Omega\left(\frac{n^3}{PM^{3/2}}\right)$ $= \Omega\left(\sqrt{P}\right)$	$O\left(\frac{n^2}{\sqrt{P}}\right)$ Can69	$O\left(\sqrt{P}\right)$ Can69
Cholesky			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ BCC+97	$O\left(\sqrt{P} \log P\right)$ BCC+97
LU			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ DGX08	$O\left(\sqrt{P} \log P\right)$ DGX08
QR			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ DGHL08a	$O\left(\sqrt{P} \log^3 P\right)$ DGHL08a
Symmetric Eigenvalues			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ BDD09	$O(\sqrt{P} \log^3 P)$ BDD09
SVD			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ BDD09	$O(\sqrt{P} \log^3 P)$ BDD09
(Generalized) Nonsymmetric Eigenvalues			$O\left(\frac{n^2 \log P}{\sqrt{P}}\right)$ BDD09	$O(\sqrt{P} \log^3 P)$ BDD09

Figure: Established bounds on communication in linear algebra.
$M = \Theta(\frac{N^2}{P})$ (Ballard, et. al, 2009)

Machine Balance



Machine Balance



Projections (CPU vs GPU)

Parameter	Keeneland values	doubling time (in years)	10-year increase factor
Cores: p_{cpu} p_{gpu}	12 448	1.87	40.7×
Peak: $p_{\text{cpu}} \cdot C_{\text{cpu}}$ $p_{\text{gpu}} \cdot C_{\text{gpu}}$	268 Gflop/s 1 Tflop/s	1.7	59.0×
Memory BW: β_{cpu} β_{gpu}	25.6 GB/s 144 GB/s	3.0	9.7×
Fast memory: Z_{cpu} Z_{gpu}	12 MB 2.7MB	2.0	32.0×
I/O device: $\beta_{\text{I/O}}$	8 GB/s	2.39	18.1×
Network BW, β_{link}	10 GB/s	2.25	21.8×

Table: Using the hardware trends we can make predictions about relative performance of future hardware. (BW = bandwidth)

Contact

Kent Czechowski
kentcz (at) gatech

Casey Battaglino
cbattaglino3 (at) gatech

Questions?