Balance Principles for Algorithm-Architecture Co-design

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

May 31, 2011
Position: Principles (i.e., “theory”) informing practice (co-design)
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:

For some algorithm: \( T_{\text{mem}} \leq T_{\text{comp}} \)

1

For principled analysis, we need theoretical models for \( T_{\text{mem}}, T_{\text{comp}} \). 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 $T_{\text{comp}}$ \quad (T_{\text{mem}} \leq T_{\text{comp}})$

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

Desired: work optimality, maximum parallelism

\(^2\)Source: Blelloch: Parallel Algorithms
Parallel DAG Model for $T_{comp}$ \quad ($T_{mem} \leq T_{comp}$)

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

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

$W(n) = $ work (total ops)

$D(n) = $ depth
Parallel DAG Model for $T_{comp}$ \quad (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.

Balance Principles for Algorithm-Architecture Co-design
I/O Model for $T_{mem}$  \( (T_{mem} \leq T_{comp}) \)

\[ Q(n; Z, L): \text{Number of cache misses.} \]
Thus, the volume of data transferred is $Q(n; Z, L) \times L$
I/O Model for $T_{\text{mem}}$ \hspace{1cm} (\hspace{1cm} T_{\text{mem}} \leq T_{\text{comp}} \hspace{1cm})

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

\[ \frac{W(n)}{Q(n; Z, L) \cdot L} = \Theta(1) \]

Intensity

\[ C \leftarrow C + A \ast B \]
I/O Model: Matrix Multiply

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

Assumes contiguous layout. Result is optimal.

\[ \frac{W}{Q \cdot L} = \mathcal{O} \left( \sqrt{Z} \right) \]

Intensity

\[ C \leftarrow C + A \ast B \]
I/O Model for $T_{mem}$ \quad ($T_{mem} \leq T_{comp}$)

We model $T_{mem}$ with:

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

- $Q$ \ldots \# of cache misses
- $C_0$ \ldots \# of cycles per second
- $p$ \ldots \# of cores
- $Z$ \ldots cache size (bytes)
- $L$ \ldots line size (bytes)
- $\alpha$ \ldots latency (s)
- $\beta$ \ldots bandwidth (bytes/s)
I/O Model for $T_{mem}$ \hspace{1cm} ($T_{mem} \leq T_{comp}$)

We model $T_{mem}$ with:

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

$Q_1$, 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 $T_{\text{mem}} \quad (T_{\text{mem}} \leq T_{\text{comp}})$

We model $T_{\text{mem}}$ with:

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

**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.

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

---

$T_{\text{comp}}, T_{\text{mem}}$

$T_{\text{mem}} \leq T_{\text{comp}}$
$T_{\text{comp}}, T_{\text{mem}}$: 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  \hspace{3cm} \text{Little's Law}  \hspace{3cm} \text{Intensity}  \hspace{3cm} \text{Amdahl's Law}

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

Ironic, et. al: Parallel Matrix Multiply Bound:

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

\( \therefore \)

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

Example: Matrix-multiply + work-stealing
Projections

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

Example: Matrix-multiply + work-stealing

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

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

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

---

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

Balance Principles for Algorithm-Architecture Co-design
“Punchline”: Projections (Matrix Multiply)

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

Cautionary prediction 2: Even matrix multiply might be memory bandwidth bound at exascale.
Consequences (Stacked Memory)

Scaling the number of PINs from memory to the processor with the *surface area* of the chip rather than the perimeter: $\beta$ 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

*Authors: Kent Czechowski, Casey Battaglino, Chris McClanahan, Aparna Chandramowlishwaran, Richard Vuduc (Georgia Tech)
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.

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

We believe that balance is an ideal frame from which to focus this principled analysis: \[ T_{\text{mem}} \leq T_{\text{comp}} \]
Limitations

How can we bring other metrics into play?

1. Power: $\text{Power}_{\text{alg}}(n; Z, L, p) \propto Q(n; Z, L, p)$?
   Power efficiency necessary for exascale

2. A more general cost metric
   (i.e. a cluster of iPads would probably be balanced)
Bounds

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bandwidth</td>
<td>Latency</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>Matrix-Multiplication</td>
<td>(\Omega\left(\frac{n^3}{P\sqrt{M}}\right))</td>
<td>(\Omega\left(\frac{n^3}{PM^{3/2}}\right))</td>
</tr>
<tr>
<td>Cholesky</td>
<td>(\Omega\left(\frac{n^2}{P}\right))</td>
<td>(\Omega\left(\sqrt{P}\right))</td>
</tr>
<tr>
<td>LU</td>
<td>(=\Omega\left(\frac{n^2}{\sqrt{P}}\right))</td>
<td>(=\Omega\left(\sqrt{P}\right))</td>
</tr>
<tr>
<td>QR</td>
<td>(O\left(\frac{n^2 \log P}{\sqrt{P}}\right))</td>
<td>(O\left(\sqrt{P \log^3 P}\right))</td>
</tr>
<tr>
<td>Symmetric Eigenvalues</td>
<td>(O\left(\frac{n^2 \log P}{\sqrt{P}}\right))</td>
<td>(O\left(\sqrt{P \log^3 P}\right))</td>
</tr>
<tr>
<td>SVD</td>
<td>(O\left(\frac{n^2 \log P}{\sqrt{P}}\right))</td>
<td>(O\left(\sqrt{P \log^3 P}\right))</td>
</tr>
<tr>
<td>(Generalized) Nonsymmetric</td>
<td>(O\left(\frac{n^2 \log P}{\sqrt{P}}\right))</td>
<td>(O\left(\sqrt{P \log^3 P}\right))</td>
</tr>
<tr>
<td>Eigenvalues</td>
<td>(O\left(\frac{n^2 \log P}{\sqrt{P}}\right))</td>
<td>(O\left(\sqrt{P \log^3 P}\right))</td>
</tr>
</tbody>
</table>

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

Tracking Balance on Various Devices

Unbalanced

Balanced

FLOPS

Bandwidth

CPU

Mobile

GPU

AMD GPU

NVIDIA GPU

Intel CPU

AMD CPU

iPad

Tegra

Class

GPU

CPU

Mobile

Bal

5

10

15

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

Balance Principles for Algorithm-Architecture Co-design
Machine Balance

Tracking Balance on Various Devices

Unbalanced

Balanced

FLOPS

27,000x

Bandwidth

CPU

GPU

Mobil

Kent Czechowski, Casey Battaglino, Chris McClanahan, Aparna Chandramowlishwaran, Richard Vuduc (Georgia Tech)
Balance Principles for Algorithm-Architecture Co-design
## Projections (CPU vs GPU)

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

**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?