Preprint – To appear in the IEEE Int'l. Parallel & Distributed Processing Symp. (IPDPS), May 2013

A roofline model of energy

Jee Whan Choi
Georgia Institute of Technology
Atlanta, Georgia, USA
jee@gatech.edu

Daniel Bedard, Robert Fowler
Renaissance Computing Institute
Chapel Hill, North Carolina, USA
{*danb,rjf*}*@renci.org*

Richard Vuduc
Georgia Institute of Technology
Atlanta, Georgia, USA
richie@gatech.edu

***Abstract*—We describe an energy-based analogue of the time-based roofline model. We create this model from the perspective of algorithm designers and performance tuners, with the intent not of making exact predictions, but rather, developing high-level analytic insights into the possible relationships among the time, energy, and power costs of an algorithm. The model expresses algorithms in terms of operations, concurrency, and memory traffic; and characterizes the machine based on a small number of simple cost parameters, namely, the time and energy costs per operation or per word of communication. We confirm the basic form of the model experimentally. From this model, we suggest under what conditions we ought to expect an algorithmic time-energy trade-off, and show how algorithm properties may help inform power management.**

***Keywords*-performance analysis; power and energy modeling; computational intensity; machine balance; roofline model**

I. Introduction

The overarching goal of this paper is to develop a simple explanation, aimed at algorithm designers and performance tuners, about the relationships among time, energy, and power. For that audience, a useful model would directly connect properties of an algorithm—such as concurrency and locality—with architectural time and energy costs. It would explain whether there is any difference in optimizing an algorithm for time versus optimizing for energy, why such differences exist, and what properties of the architecture might lead to non-trivial time-energy trade-offs. We have studied similar kinds of models in some of our prior work [1]–[3], but thus far have not considered energy in a formal way.

Our analysis is inspired by a similar set of thought experiments based on "Amdahl" analysis, written by and for architects [4]–[6]. (We review this work and numerous other related studies in §VI.) Such analyses offer architectural insights, but abstract away essential properties of an algorithm. By contrast, our analysis more explicitly connects algorithmic and architectural parameters. However, for clarity we pose and study an intentionally simple—but not overly so—model, with some initial experimental tests to confirm its basic form.

Below, we summarize what our model implies. These claims both reflect familiar intuition and also yield new or alternative explanations about time, energy, and power relationships.

First, when analyzing time, the usual first-order analytic tool is to assess the *balance* of the processing system [7]–[12]. Recall that balance is the ratio of work the system can perform per unit of data transfer. To this notion of time-balance, we define an *energy-balance* analogue, which measures the ratio of flops and bytes *per unit-energy* (e.g., Joules). We compare balancing computations in time against balancing in energy. [§II]

Secondly, we use energy-balance to develop an energy-based analogue of the time-based roofline model [12]. Because time can be overlapped while energy cannot, the energy-based "roofline" is actually a smooth "arch line" (see fig. 2a). Interestingly, if time-balance differs from energy-balance, then there are distinct notions of being "compute-bound" versus "memory-bound," depending on whether the optimization goal is to minimize time or to minimize energy. We can measure this difference as a time-energy *balance gap*. We also posit an analogous "powerline" model for power. [§II, §III]

Thirdly, when a balance gap exists and energy-balance *exceeds* time-balance, the arch line predicts that optimizing for energy may be fundamentally *more difficult* than optimizing for time. It further suggests that high algorithmic energy-efficiency may *imply* time-efficiency, while the converse—that time-efficiency implies energy-efficiency—is *not* true. [§II, §IV]

Fourthly, we test the basic form of the model using experiments on real CPU and GPU platforms. Using our model and these data, we show that the hypothetical balance gap above does not yet really exist, which consequently explains why on today's platforms race-to-halt is likely to work well. This raises the question for architects and hardware designers about what the fundamental *trends* in the balance gap will be: if energy-balance will exceed time-balance in the future, race-to-halt will break. We further use the experiments to highlight both the strengths and the limitations of our model and analysis. [§IV, §V, §VII]

Taken together, we believe these analyses can improve our collective understanding of the relationship among algorithm properties and their costs in time, energy, and power.

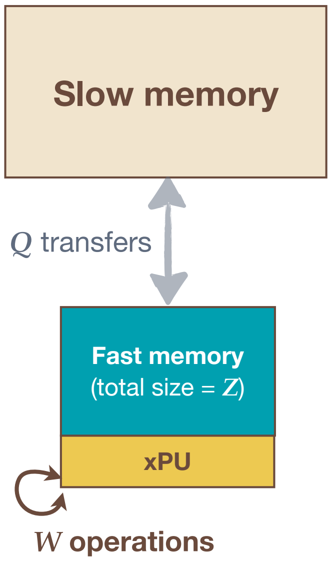


Figure 1: A simple von Neumann architecture with a two-level memory hierarchy. In our first analysis, suppose that an algorithm performs W arithmetic operations and Q memory operations, or "mops," between slow and fast memories.

II. A Basic Model and Its Interpretation

Assume the simple architecture shown in fig. 1. This architecture has a processing element, labeled "xPU", as well as two levels of memory, namely, an infinite slow memory and a fast memory of finite capacity. This system roughly captures everything from a single functional unit (xPU) attached to registers (fast memory), to a manycore processor (xPU) attached to a large shared cache (fast memory). Further assume that the xPU may only perform operations on data present in the fast memory. As such, an algorithm for this architecture must explicitly move data between slow and fast memories.

A. *Algorithm characterization*

Let W be the total number of "useful" operations that the algorithm performs and let Q be the total number of words it transfers. (Table I summarizes all of the parameters of our model.) By *useful*, we mean in an algorithmic sense; for example, we might only count flops when analyzing matrix multiply, or comparisons for sorting, or edges traversed for a graph traversal algorithm. For simplicity, we will assume W is measured in units of flops. We will also refer to W as the total *work* of the algorithm. Regarding Q, we will for simplicity not distinguish between loads and stores, though one could do so in principle. We will refer to Q as "mops" measured in some convenient storage unit, such as a word or a byte.

In a typical algorithm analysis, both W and Q will of course depend on characteristics of the input, such as its size n;[1] in addition, Q will depend on the size of the fast memory. We discuss these dependences momentarily.

For performance analysis and tuning, we may measure the algorithm's *computational intensity*, which is defined as $I \equiv W/Q$. Intensity has units of operations per unit storage, such as flops per word or flops per byte. Generally speaking, a higher value of I implies a more "scalable" algorithm. That is, it will have more work than mops; therefore, it is more likely to improve as the architecture's compute throughput increases, which happens as cores increase or SIMD lanes widen.

Variable	Description
W	# of arithmetic operations, e.g., # of flops
Q	# of main memory operations ("mops")
I	Intensity, or W/Q (e.g., flops per byte)
τ_{flop}	Time per work (arithmetic) operation, e.g., time per flop
τ_{mem}	Time per mop
B_τ	Balance in time, $\tau_{\text{mem}}/\tau_{\text{flop}}$ (e.g., flops per byte)
ϵ_{flop}	Energy per arithmetic operation
ϵ_{mem}	Energy per mop
B_ϵ	Balance in energy, $\epsilon_{\text{mem}}/\epsilon_{\text{flop}}$ (e.g., flops per Joule)
ϵ_0	Constant energy per flop
$\epsilon_{\text{flop}} + \epsilon_0$	Minimum energy to execute one flop
η_{flop}	Constant-flop energy efficiency, $\frac{\epsilon_{\text{flop}}}{\epsilon_{\text{flop}}+\epsilon_0}$
π_0	Constant power, e.g., Joule per second = Watts
π_{flop}	Baseline power per flop excluding constant power, $\frac{\epsilon_{\text{flop}}}{\tau_{\text{flop}}}$
$\hat{B}_\epsilon\ (I)$	Effective energy-balance ($\pi_0 \geq 0$)
T_{flops}	Total time to perform arithmetic
T_{mem}	Total time to perform mops
T	Total time
E_{flops}	Total energy of arithmetic
E_{mem}	Total energy of mops
E_0	Total "constant" energy
E	Total energy
P	Average power
Z	Fast memory size (e.g., words, bytes)

Table I: Summary of model parameters

What should we expect about the value of I? Recall that Q depends on fast memory capacity, which we denote by Z units of storage (words or bytes), as shown in fig. 1. Therefore, intensity will also depend on Z. A well-known result among algorithm designers is that no algorithm for $n \times n$ matrix multiply can have an intensity exceeding $I = \mathcal{O}\left(\sqrt{Z}\right)$ [13]. Consequently, if we improve an architecture by doubling Z, we will improve the inherent algorithmic intensity of a matrix multiply algorithm by no more than $\sqrt{2}$. Contrast this scenario to that of just summing all of the elements of an array. Intuitively, we expect this computation to be memory bandwidth-bound if the array is very large. Indeed, it has an intensity of $I = \mathcal{O}(1)$, that is, a constant independent of problem size or Z. Thus, increasing Z has no effect on the intensity of this kind of reduction. In short,

[1]That is, imagine a $W(n) = \mathcal{O}(n)$ style of analysis. However, unlike the traditional forms of such analysis, we will also want to characterize constants and costs much more precisely whenever possible.

Table II: Sample values for model parameters, based on best case (peak) capabilities of currently available systems. See table I for a summary of the definitions of these parameters.

Variable	Representative values NVIDIA "Fermi" GPU [2]
τ_{flop}	$(515 \text{ Gflop/s})^{-1} \approx 1.9$ ps per flop[a]
τ_{mem}	$(144 \text{ GB/s})^{-1} \approx 6.9$ ps per byte[b]
B_τ	$6.9/1.9 \approx 3.6$ flops per byte
ϵ_{flop}	≈ 25 pJ per flop[c]
ϵ_{mem}	≈ 360 pJ per byte
B_ϵ	$360/25 \approx 14.4$ flops per Joule

[a] Based on peak double-precision floating-point throughput.
[b] Based on peak memory bandwidth.
[c] Based on 50 pJ per double-precision fused multiply-add.

the concept of intensity measures the inherent locality of an algorithm.

B. Time and energy costs

Next, we translate the abstract W and Q into concrete time and energy costs. We will distinguish between the costs of performing work versus that of data transfer. Furthermore, our model of energy cost will have two significant differences from our model of time cost, namely, (i) time costs may be overlapped whereas energy may not; and (ii) we must burn *constant* energy, that is, we must burn an additional amount of baseline energy throughout the computation. These distinctions are critical, and together determine whether or not one should expect an algorithmic time-energy trade-off (see §VII).

More formally, suppose T_{flops} and T_{mem} are the total time (seconds) to execute all work operations and all mops, respectively. Further assume, *optimistically*, that overlap is possible. Then, the total time T is

$$T \equiv \max\left(T_{\text{flops}}, T_{\text{mem}}\right). \tag{1}$$

Similarly, suppose that E_{flops} and E_{mem} are the total energy (Joules) for work and mops. In addition, let $E_0(T)$ be the *constant energy* of the computation. Constant energy is the energy that must be expended for the duration of the computation, which we will further assume is a fixed cost independent of the type of operations being performed. Then, our model of energy cost is

$$E \equiv E_{\text{flops}} + E_{\text{mem}} + E_0(T). \tag{2}$$

Consider the component costs, beginning with time. Suppose each operation has a fixed time cost. That is, let τ_{flop} be the time per work operation and τ_{mem} be the time per mop. We will for the moment tacitly assume *throughput-based* values for these constants, rather than latency-based values. (See table II for sample parameters.) This assumption will tend to give a best-case analysis,[2] which is only valid when an algorithm has a sufficient degree of concurrency; we discuss a more refined model based on work-depth in prior work [1]. From these basic costs, we then define the component times as $T_{\text{flops}} \equiv W\tau_{\text{flop}}$ and $T_{\text{mem}} \equiv Q\tau_{\text{mem}}$. Then, under the optimistic assumption of perfect overlap, the algorithm's running time becomes

$$\begin{aligned} T &= \max\left(W\tau_{\text{flop}}, Q\tau_{\text{mem}}\right) \\ &= W\tau_{\text{flop}} \cdot \max\left(1, \frac{B_\tau}{I}\right), \end{aligned} \tag{3}$$

where we have defined $B_\tau \equiv \tau_{\text{mem}}/\tau_{\text{flop}}$. This quantity is the classical *time-balance point*, or simply *time-balance* [7]–[12]. Time-balance is the architectural analogue of algorithmic intensity and has the same units thereof, e.g., flops per byte. Furthermore, if we regard $W\tau_{\text{flop}}$ as the ideal running time in the absence of any communication, then we may interpret $\frac{B_\tau}{I}$ as the *communication penalty* when it exceeds 1. We refer to this condition, $I > B_\tau$, as a *balance principle* [1]. Our algorithmic design goal is to create algorithms that minimize time and have high intensity relative to machine's time-balance.

The simplest and most natural energy cost model is to define an analogous fixed energy per work operation, ϵ_{flop}, and fixed energy per mop, ϵ_{mem}. Additionally, suppose the constant energy cost is linear in T, with a fixed *constant power* of π_0 units of energy per unit time. Then,

$$\begin{aligned} E &= W\epsilon_{\text{flop}} + Q\epsilon_{\text{mem}} + \pi_0 T \\ &= W\epsilon_{\text{flop}} \cdot \left(1 + \frac{B_\epsilon}{I} + \frac{\pi_0}{\epsilon_{\text{flop}}}\frac{T}{W}\right), \end{aligned} \tag{4}$$

where $B_\epsilon \equiv \epsilon_{\text{mem}}/\epsilon_{\text{flop}}$ is the *energy-balance point*, by direct analogy to time-balance.

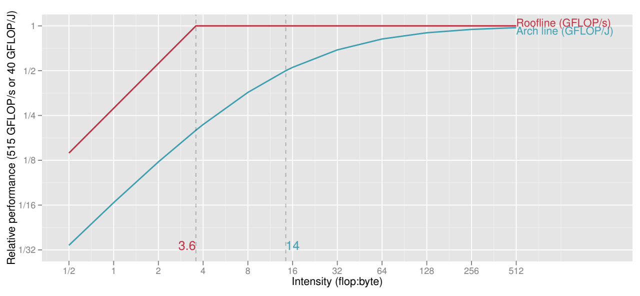
Let us refine eq. (4) so that its structure more closely parallels eq. (3), which in turn will simplify its interpretation. Let $\epsilon_0 \equiv \pi_0 \cdot \tau_{\text{flop}}$ be the *constant energy per flop*, that is, the energy due to constant power that burns in the time it takes to perform one flop. Moreover, $\hat{\epsilon}_{\text{flop}} \equiv \epsilon_{\text{flop}} + \epsilon_0$ becomes the actual amount of energy required to execute one flop, given non-zero constant power. Next, let $\eta_{\text{flop}} \equiv \epsilon_{\text{flop}}/\hat{\epsilon}_{\text{flop}}$ be the *constant flop energy-efficiency*. This machine parameter equals one in the best case, when the machine requires no constant power ($\pi_0 = 0$). Then, substituting eq. (3) into eq. (4) yields

$$E = W \cdot \hat{\epsilon}_{\text{flop}} \cdot \left(1 + \frac{\hat{B}_\epsilon(I)}{I}\right), \tag{5}$$

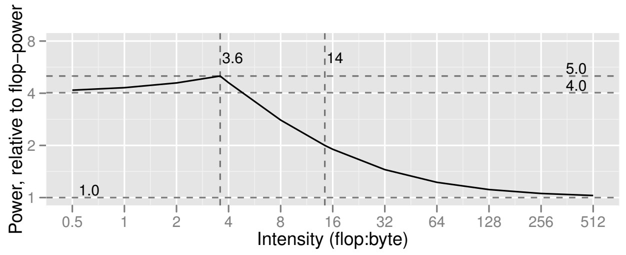
where $\hat{B}_\epsilon(I)$ is the *effective energy-balance*,

$$\hat{B}_\epsilon(I) \equiv \eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \max\left(0, B_\tau - I\right). \tag{6}$$

[2]Additionally, assuming throughput values for τ_{mem} implies that a memory-bound computation is really memory *bandwidth* bound.



(a) Rooflines versus arch lines. The red line with the sharp inflection shows the roofline for speed; the smooth blue line shows the "arch line" for energy-efficiency. The time- and energy-balance points (3.6 and 14 FLOP:Byte, respectively) appear as vertical lines and visually demarcate the balance gap.



(b) A "power-line" chart shows how average power (y-axis, normalized to π_{flop}) varies with intensity (x-axis, flop:byte). Going from bottom to top, the horizontal dashed lines indicate the flop power (y=1), the memory-bound lower limit (y=$\frac{B_\epsilon}{B_\tau}$=4.0), and the maximum power (y=$1+\frac{B_\epsilon}{B_\tau}$).

Figure 2: Rooflines in time, arch lines in energy, and power lines. Machine parameters appear in table II, for an NVIDIA Fermi-class GPU assuming constant power is 0. Dashed vertical lines show time- and energy-balance points.

The ideal energy is that of just the flops, $W\hat{\epsilon}_{\text{flop}}$. On top of this ideal, we must pay an effective energy communication penalty, which $\hat{B}_\epsilon(I)/I$ captures. Therefore, similar to the case of execution time, our algorithmic design goal with respect to energy is to create work-optimal algorithms with an intensity that is high relative to machine's effective energy-balance. That is, just like time there is a balance principle for energy, $\hat{B}_\epsilon(I) \ll I$.

When $B_\tau = \hat{B}_\epsilon(I)$, optimizing for time and energy are most likely the same process. The interesting scenario is when they are unequal.

C. Rooflines in time and energy

We can visualize the balance principles of eqs. (3) and (5) using a roofline diagram [9], [12]. A roofline diagram is a line plot that shows how performance on some system varies with intensity. Figure 2a depicts the simplest form of the roofline, using the values for τ_{flop}, τ_{mem}, ϵ_{flop}, and ϵ_{mem} shown in table II. These values were taken from Keckler et al. for a NVIDIA Fermi-class GPU [14]; but since they did not provide estimates for constant power, for now assume $\pi_0 = 0$. The x-axis (log base 2) shows intensity I. The y-axis (log base 2) shows performance, normalized either to the maximum possible speed (flops per unit time) or maximum possible energy-efficiency (flops per unit energy) on the platform. That is, the roofline with respect to time is the curve given by $W\tau_{\text{flop}}/T = \min(1, I/B_\tau)$ plotted against I. Similarly the curve for energy is given by $W\hat{\epsilon}_{\text{flop}}/E = 1/(1+\hat{B}_\epsilon(I)/I)$; since $\pi_0 = 0$ by assumption, $\hat{\epsilon}_{\text{flop}} = \epsilon_{\text{flop}}$ and $\hat{B}_\epsilon(I) = B_\epsilon$. In both cases, the best possible performance is the time or energy required by the flops alone.

The roofline for speed (inverse time) is the red line of fig. 2a. Since the component times may overlap, the roofline has a sharp inflection at the critical point of $I = B_\tau$. When $I < B_\tau$, the computation is memory-bound in time, whereas $I \geq B_\tau$ means the algorithm is compute-bound in time.

There is also a "roofline" for energy-efficiency, shown by the smooth blue curve in fig. 2a. It is smooth since we cannot hide memory energy and since $\pi_0 = 0$. As such, we may more appropriately refer to it as an "arch line." Rather than a sharp inflection, the energy-balance point $I = B_\epsilon$ is the point at which energy-efficiency is half of its best possible value.[3] Put another way, suppose W is fixed and we increase I by reducing Q. Then, B_ϵ is the intensity at which the amount of energy spent on flops equals that spent on communication. In this sense, an algorithm may be compute-bound or memory-bound in *energy*, which will differ from time when $B_\tau \neq B_\epsilon$.

D. The balance gap

The aim of rooflines and arches is to guide optimization. Roughly speaking, an algorithm designer starts with some baseline algorithm having a particular intensity (x-axis value). A roofline or arch line provides two pieces of information: (a) it suggests the target performance tuning goal, which is the corresponding y-axis value; and (b) it suggests by how much intensity must increase to improve performance by a desired amount. Furthermore, it also suggests that the optimization strategies may differ depending on whether the goal is to minimize time or minimize energy.

The balance points tell part of the story. Ignoring constant power, we expect $B_\tau < B_\epsilon$ because both the time and energy of a mop tend to be relatively large compared to that of a flop [14]. If so, an algorithm with $B_\tau < I < B_\epsilon$ is

[3]This relationship follows from the simple algebraic fact that $a + b \leq 2\max(a, b)$.

simultaneously compute-bound in time while being memory-bound in energy. Furthermore, assume that increasing intensity is the hard part about designing new algorithms or tuning code. Then, $B_\tau < B_\epsilon$ suggests that energy-efficiency is even harder to achieve than time-efficiency. The *balance gap*, or ratio B_ϵ/B_τ, measures the difficulty.

Having said that, a nice corollary is that energy-efficiency may imply time-efficiency. That is, $I > B_\epsilon$ implies that $I > B_\tau$ as well. However, the converse—that time-efficiency implies energy-efficiency—does not in general hold by similar reasoning. Of course, roofs and arches are only bounds, so these high-level claims are only guidelines, rather than guaranteed relationships. Nevertheless, it may suggest that if we were to choose one metric for optimization, energy is the nobler goal.

If, on the other hand, $B_\tau > B_\epsilon$, then time-efficiency would tend to imply energy-efficiency. This condition is one in which so-called race-to-halt strategies for saving energy will tend to be effective [15].[4]

Lastly, note that the analogous conditions hold when $\pi_0 > 0$, but with $\hat{B}_\epsilon(I)$ in place of B_ϵ. Higher constant power means lower η_{flop}; consequently, referring to eq. (6), it would cause $\hat{B}_\epsilon(I)$ to be lower than B_ϵ.

III. What the basic model implies about power

Assuming our time and energy models are reasonable, we can also make analytic statements about the *average power* that an algorithm uses when running on a machine, $P \equiv E/T$. Define the *power per flop* to be $\pi_{\text{flop}} \equiv \epsilon_{\text{flop}}/\tau_{\text{flop}}$, that is, the power to execute a flop excluding constant power. Then, dividing eq. (5) by eq. (3) and applying a little algebra leads to

$$P = \frac{\pi_{\text{flop}}}{\eta_{\text{flop}}}\left[\frac{\min(I, B_\tau)}{B_\tau} + \frac{\hat{B}_\epsilon(I)}{\max(I, B_\tau)}\right]. \quad (7)$$

The "power-line" diagram of fig. 2b depicts the most interesting features of eq. (7), using the parameters of table II with $\pi_0 = 0$ ($\eta_{\text{flop}} = 1$). If the algorithm is severely memory-bound ($I \to 0$), then $P > \pi_{\text{flop}}\frac{B_\epsilon}{B_\tau}$. If instead the algorithm is very compute-bound ($I \to \infty$), P decreases to its lower-limit of π_{flop}. The algorithm requires the maximum power when $I = B_\tau$. From its value there, we can conclude that

$$P \quad \leq \quad \pi_{\text{flop}}\left(1 + \frac{B_\epsilon}{B_\tau}\right). \quad (8)$$

That is, relative to π_{flop}, we pay an extra factor related to the balance gap. The larger this gap, the larger average power will be.

[4]The race-to-halt strategy says that the best way to save energy is to run as fast as possible and then turn everything off.

Device	Model	Peak performance Single (Double) GFLOP/s	Peak memory bandwidth GB/s	TDP (chip only) Watts
CPU	Intel Core i7-950	106.56 (53.28)	25.6	130
GPU	NVIDIA GeForce GTX 580	1581.06 (197.63)	192.4	130

Table III: Platforms

IV. An experiment

The model of §II is an hypothesis about the relationship between intensity and performance, both in time and in energy. This section explores whether we can observe this relationship on actual systems.

A. Experimental Setup

Hardware: Table III shows our experimental platforms, which include an Intel quad-core Nehalem CPU and a NVIDIA Fermi consumer-class GPU. To measure power, we use two tools. The first is PowerMon 2 [16], a fine-grained integrated power measurement device for measuring CPU and host component power. The second is a custom in-house PCIe interposer for measuring GPU power.

Figure 3 shows how the measurement equipment connects to the system. PowerMon 2 sits between the power supply unit and various devices in the system. It directly measures direct current voltage and current on up to eight individual channels using digital power monitor integrated circuits. It can sample at 1024 Hz per channel, with an aggregate frequency of up to 3072 Hz. PowerMon 2 reports formatted and time-stamped measurements without the need for additional software, and fits in a 3.5 inch internal hard drive bay.

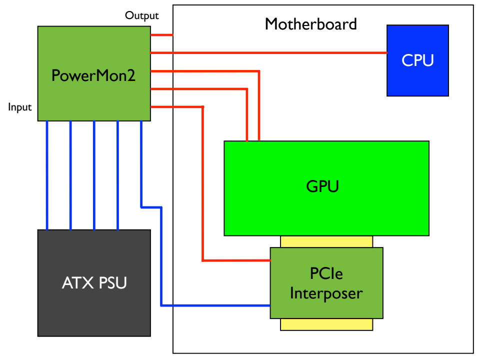


Figure 3: Placement of the measurement probes, PowerMon 2 and our custom PCIe interposer

Modern high-performance GPUs have high power requirements. Typically, they draw power from multiple sources, including the motherboard via the PCIe connector. In order to measure the power coming from the motherboard we use a PCIe interposer that sits between the GPU and the motherboard's PCIe connector. The interposer intercepts the signals coming from the pins that provide power to the GPU.

Measurement method: The GPU used in our study draws power from two 12 Volt connectors (8-pin and 6-pin) that come directly from the ATX Power Supply Unit (PSU), and from the motherboard via the PCIe interface, which supply 12 V and 3.3 V connectors. When performing GPU benchmarking, PowerMon 2 measures the current and voltage from these four sources at a regular interval. For each sample, we compute the instantaneous power by multiplying the measured current and voltage at each source and then sum over all sources. We can then compute the average power by averaging the instantaneous power over all samples. Finally, we compute the total energy by multiplying average power by total time. In this setup, we are able to largely isolate GPU power from the rest of the system (e.g., host CPU).

The PSU provides power to our CPU system using a 20-pin connector that provides 3.3 V, 5 V and 12 V sources and a 4-pin 12 V connector. As with the GPU, PowerMon 2 measures current and voltage from these four sources; we compute the average power and total energy in the same manner as above. For our CPU measurements, we physically remove the GPU and other unnecessary peripherals so as to minimize variability in power measurements.

In the experiments below, we executed the benchmarks 100 times each and took power samples every 7.8125 ms (128 Hz) on each channel.

B. Intensity microbenchmarks

We implemented microbenchmarks that allow us to vary intensity, and tuned them to achieve very high fractions of peak FLOP/s or bandwidth as predicted by the roofline. We then measured the time and power as described above to see to what extent they behave as our model predicts. The results appear in fig. 4, with measured data (shown as dots) compared to our model (shown as a solid line). We describe these benchmarks and how we instantiated the model below.[5]

The GPU microbenchmark executes a mix of independent fused multiply-add operations (FMA flops, counted as two flops each) and memory load operations. We auto-tuned this microbenchmark to maximize performance on the GPU by tuning kernel parameters such as number of threads, thread block size, and number of memory requests per thread. The GPU kernel is fully unrolled and its correctness verified by inspecting the PTX code and comparing the computed results against an equivalent CPU kernel. The CPU microbenchmark evaluates a polynomial and is written in assembly, tuned specifically to maximize instruction throughput on a Nehalem core. Changing the degree of the polynomial effectively varies the computation's intensity.

	NVIDIA GTX 580	Intel Core i7-950
ϵ_s	99.7 pJ / FLOP	371 pJ / FLOP
ϵ_d	212 pJ / FLOP	670 pJ / FLOP
ϵ_{mem}	513 pJ / Byte	795 pJ / Byte
π_0	122 Watts	122 Watts

Table IV: Fitted energy coefficients. Note that ϵ_{mem} is given in units of picoJoules per *Byte*. As it happens, the π_0 coefficients turned out to be identical to three digits on the two platforms.

The kernel is parallelized using OpenMP to run on all 4 cores. Although the CPU and GPU benchmarks differ, their intent is simply to permit varying of intensity and achieving performance as close to the roofline as possible. As such, what they compute is not as important as being highly-tuned and having controllable intensity.

Figure 4 shows that both microbenchmarks achieve performance close to the roofline in most cases. Refer specifically to the "Time" subplots, where the roofline is drawn using peak GFLOP/s and bandwidth numbers from table III. For instance, the double-precision version of the GPU benchmark achieves up to 170 GB/s, or 88.3% of system peak when it is bandwidth bound, and as much as 196 GFLOP/s, or 99.3% of system peak when it is compute bound. For single precision, the kernel performs up to 168 GB/s and 1398 GFLOP/s respectively. However, the kernel's performance departs from the roofline significantly near the time-balance point. We explain this phenomenon in terms of our model when discussing fig. 5 (see §V-B).

The CPU microbenchmark achieves up to 18.7 GB/s and 99.4 GFLOP/s, or 73.1% and 93.3% of peak in single precision performance. The achieved bandwidth is comparable to that of the STREAM benchmark[6] and the lower fraction of peak bandwidth observed is typical for CPU systems. Double-precision performance is 18.9 GB/s (73.8%) and 49.7 GFLOP/s (93.3%), respectively.

Model instantiation: To instantiate eq. (3), we estimate time per flop and time per mop using the inverse of the peak manufacturer's claimed throughput values as shown in table III. For the energy costs in eq. (5), such specifications do not exist. Therefore, we estimated them using linear regression on our experimental data.[7] In particular, the data points are a series of 4-tuples (W, Q, T, R), where we choose W and Q when running the microbenchmark, T is the *measured* execution time, and R is a binary variable set to 0 for single-precision and 1 for double-precision. We use linear regression to find the coefficients of the model,

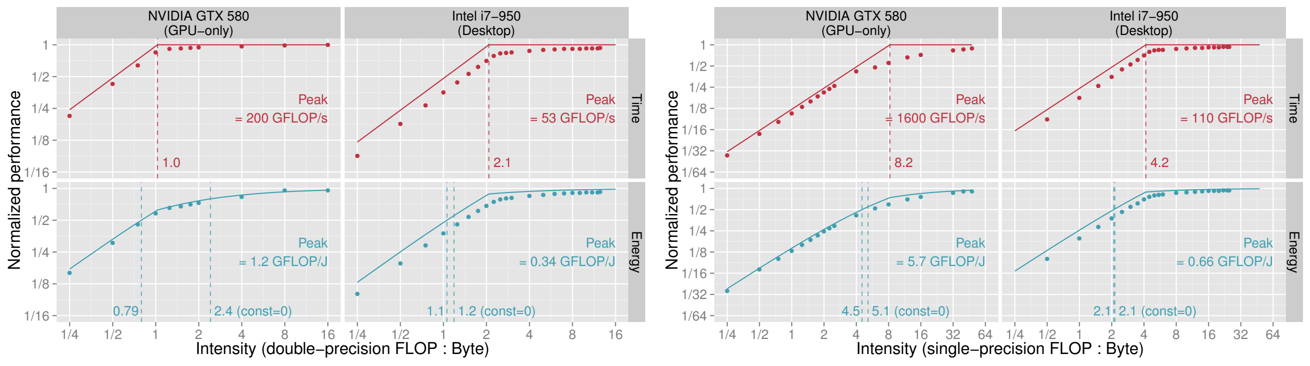
$$\frac{E}{W} = \epsilon_s + \epsilon_{mem}\frac{Q}{W} + \pi_0\frac{T}{W} + \Delta\epsilon_d R, \tag{9}$$

which yields the energy per single-precision flop, ϵ_s; energy per single-precision word, ϵ_{mem}; constant power, π_0; and

[5] http://code.google.com/p/a-roofline-model-of-energy-ubenchmarks/.

[6] streambench.org

[7] We use the standard regression routine in R, r-project.org.



(a) Double-precision (b) Single-precision

Figure 4: Measured time and energy for a synthetic benchmark corroborates the basic form of model, eqs. (3) and (5). We show the true energy-balance point as well as the energy-balance if $\pi_0 = 0$. The impact of constant energy can be profound: although the GPU double-precision case has $B_\epsilon > B_\tau$, the effective energy-balance point $\hat{B}_\epsilon$ at $y = \frac{1}{2}$ on the arch line is *less than* the time-balance. In other words, in this case time-efficiency implies energy-efficiency *because of constant power*, which further suggests that "race-to-halt" is a reasonable energy-saving strategy; were $\pi_0 \to 0$, the situation could reverse.

$\Delta\epsilon_{\mathrm{d}}$, which is the *additional* energy required for a double-precision flop over a single-precision flop.[8] That is, the energy per double-precision flop is $\epsilon_{\mathrm{d}} \equiv \epsilon_{\mathrm{s}} + \Delta\epsilon_{\mathrm{d}}$. We summarize the fitted parameters in table IV. We then plug these coefficients into eq. (5) to produce the model energy curves shown in fig. 4. These curves visually confirm that the fitted model captures the general trend in the data. We analyze these curves in more detail in the next section (§V).

V. Discussion, application, and refinement

A. The fitted parameters

Keckler et al. provide some information about GPU energy [14], providing a way to sanity-check and clarify what the fitted parameters in table IV represent.

First, Keckler et al. state that the energy cost of the floating-point unit that performs one double-precision FMA is about 50 pJ, or 25 pJ *per flop*; our estimate in table IV is about eight times larger. This discrepancy arises because the 50 pJ FMA cost excludes various instruction issue and microarchitectural overheads (e.g., registers, component interconnects), which our measurement implicitly includes. Based on our estimates, these overheads account for roughly 187 pJ/flop.

Secondly, the discussion of Keckler et al. on memory access costs suggests a baseline memory-energy cost of 253–389 pJ per Byte. This cost includes dynamic random access memory (DRAM) access costs, interface costs, and wire transfer. However, this estimate ignores instruction overheads and possible overheads due to cache. Recall that we estimated the instruction overhead for a floating point instruction to be roughly 187 pJ, or approximately 47 pJ/Byte in single–precision. Adding this number to the baseline produces an estimate of 300–436 pJ/Byte. We also have to account for the costs of storing and reading the data from the L1 and L2 caches as it travels up the memory hierarchy. From the paper by Keckler et al., we can estimate this cost to be approximately 1.75 pJ/Byte per read/write for both L1 and L2 (assuming they are both implemented using static random access memory (SRAM)), or a total of 7 pJ/Byte for both L1 and L2 read and write traffic. This brings the total cost estimate to 307–443 pJ/Byte. Our estimate of ϵ_{mem} is larger, which may reflect additional overheads for cache management, such as tag matching.

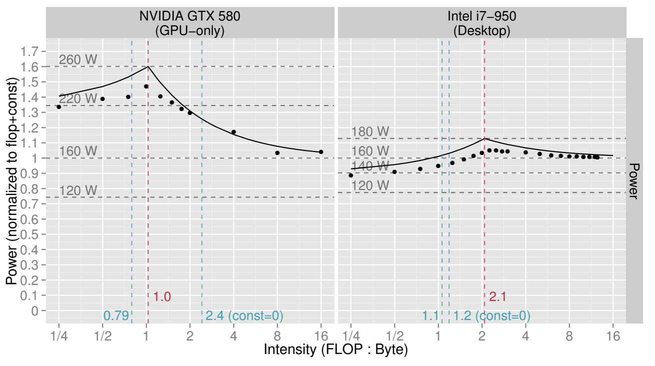
There is no information provided to check the constant power value; for reference, we measured GPU idle power (powered on but with nothing running) to be 39.6 Watts. As such, our constant power estimate accounts for much more than just idle power.

Not surprisingly, the estimates of CPU energy costs for both flops and memory are higher than their GPU counterparts. Flop energy costs are higher due to the higher complexity of a CPU processing core relative to its GPU counterpart; similarly, memory energy costs are higher due in large part to a greater distance between the processor and memory in the CPU system relative to the GPU system. Also, the CPU used in our experiments was built using an older process technology. All of these characteristics have a profound impact on the balance gap, discussed next.

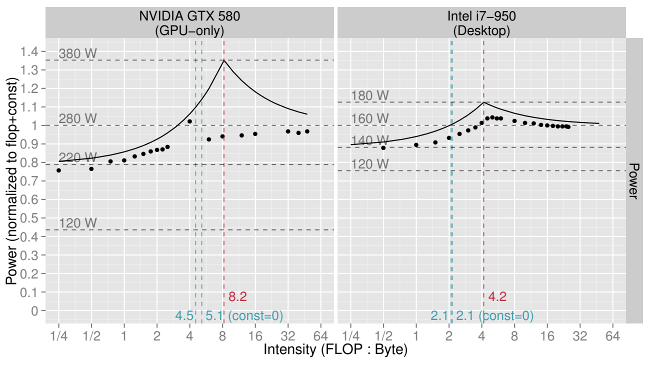
B. Balance gaps and power caps

Consider the rooflines and arch lines of fig. 4. In all cases, the time-balance point exceeds the y=1/2 energy-balance point, which means that time-efficiency will tend to imply energy-efficiency. That is, once the microbenchmark is compute-bound in time ($I > B_\tau$), it is also within a

[8]Normalizing the regressors by W produces high-quality fits, with R^2 (residual) coefficient near unity at p-values below 10^{-14}.



(a) Double-precision



(b) Single-precision

Figure 5: Measured power for the microbenchmark corroborates the "powerline" model. On the GTX 580 platform, NVIDIA reports a limit of 244 Watts, which explains the discrepancy between the observed data and the predicted powerline in the single-precision GTX 580 case.

factor of two of the optimal energy-efficiency. We believe this observation explains why race-to-halt can be such an effective energy-saving strategy [15].

If architects could drive $\pi_0 \to 0$, then the situation could reverse—see the double-precision GPU case, where the "const=0" energy-balance line in the bottom-left subplots of fig. 4a denotes this scenario. However, for the Intel platform, even having $\pi_0 = 0$ does not invert the balance gap. Instead, as table IV suggests, ϵ_{flop} and ϵ_{mem} on the Intel platform are closer than they are on the NVIDIA platform. An important question then is to what extent π_0 will go toward 0 and to what extent microarchitectural inefficiencies will reduce.

As noted previously, the single-precision GPU performance in fig. 4b does not track the roofline closely in the neighborhood of B_τ. The reason is that our model has an important limitation: we do not incorporate explicit power caps. To see this effect, refer to the powerlines of fig. 5 as well as the theoretical illustration of fig. 2b. Our model demands that power increase sharply as I approaches B_τ (see §III). For instance, on the GPU in single-precision, our model says we will need 387 Watts on the GPU as shown in fig. 5b. This scenario would cause excessive thermal issues, and indeed the GTX 580 has a maximum power rating of 244 Watts, which our microbenchmark already begins to exceed at high intensities. Thus, incorporating power caps will be an important extension for future work.

C. Applying and refining the model: FMM$_U$ on the GPU

Beyond the microbenchmark, we apply our model to the *fast multipole method* (FMM), an $O(n)$ approximation method for n-body computations, which normally scales as $O(n^2)$ [17]. We specifically consider the most expensive phase of the FMM, called the *U-list* phase (FMM$_U$), and target GPUs. Our interest here is whether we can estimate time and energy using our model.

Algorithm 1 FMM$_U$ algorithm

```
1: for each target leaf node, $B$ do
2:   for each target point $t \in B$ do
3:     for each neighboring source node, $S \in U(B)$ do
4:       for each source point $s \in S$ do
5:         $(\delta_x, \delta_y, \delta_z) = (t_x - s_x, t_y - s_y, t_z - s_z)$
6:         $r = \delta_x^2 + \delta_y^2 + \delta_z^2$
7:         $w = \text{rsqrtf}(r)$ {Reciprocal square-root}
8:         $\phi_t + = d_s * w$ {$d_s$ and $\phi_t$ are scalars}
```

The FMM$_U$ algorithm: The FMM$_U$ phase appears as pseudocode in Algorithm 1. The n points are arranged into a spatial tree, with leaf nodes of the tree containing a subset of the points. For every leaf node B, FMM$_U$ iterates over its neighboring leaf nodes. The list of neighbors is called the "U-list," denoted as $U(B)$. The node B is the *target* node, and each neighbor $S \in U(B)$ is a *source* node. For each pair (B, S), FMM$_U$ iterates over all pairs of points $(t \in B, s \in S)$ and updates ϕ_t, a value associated with the target point t. According to lines 5-8, each pair of points involves 11 scalar flops, where we count "reciprocal square-root" $(1/\sqrt{r})$ as one flop. Furthermore, each leaf contains $O(q)$ points for some user-selected q; the number of flops is therefore $O(q^2)$ for every $O(q)$ points of data, with q typically on the order of hundreds or thousands. Thus, the FMM$_U$ phase is typically compute-bound.

From prior work, we happen to have generated approximately 390 different code implementations of this benchmark [18]. These variants use a variety of performance optimization techniques and tuning parameter values.

Initial fitting: Initially, we used eq. (2) to estimate the total energy of each implementation. We derived the number of flops from the input data and the number of bytes read from the DRAM using hardware counters (L2 read misses) provided in NVIDIA's Compute Visual Profiler [19]. However, when compared to measured data, our energy estimates were lower by 33% on average.

Accounting for cache access: We attributed this difference to cache access costs. To account for such costs, we used our reference implementation of FMM_U, which relies only on L1 and L2 caches for data reuse. That is, it does not use shared or texture memory or register-level blocking. By measuring the number of bytes read from the L1 and L2 caches using counters and then dividing the *difference* between measured energy and energy estimated using eq. (2) by this number, we estimated a cache access energy cost to be 187 pJ/Byte. (This estimate does not of course distinguish between different levels of cache access.) When we used this estimate for all the other FMM_U variations that only use L1 and L2 caches (about 160 such kernels), our new energy estimates had a median error of 4.1%. For future work, we plan to study this in greater detail and derive individual costs of loading data from L1, L2, texture and shared memory.

VI. Related work

The perspective of this paper is algorithms, rather than architecture, systems, or embedded software, where time, power, and energy are traditionally studied (see the survey of Kaxiras et al. [20].) Our model is perhaps most similar to a recent technical report by Demmel et al. [21]. However, our model is more parsimonious and, as such, clarifies a number of issues such as the notion of a balance gap or why race-to-halt works on current systems. At a more technical level, we also differ in that we assume computation-communication overlap and have furthermore tried to validate the basic form of our model with experiments.

Additional algorithmic theory work: The algorithms community has also considered the impact of energy constraints, particularly with respect to exploiting scheduling slack. There have also been numerous other attempts to directly explore the impact of energy constraints on algorithms. These include new complexity models, including new energy-aware Turing machine models [22]–[25]; this body of work addresses fundamental theoretical issues but is hard to operationalize for practical algorithm design and tuning. Other algorithmic work takes up issues of frequency scaling and scheduling [26], [27]. Such models are particularly useful for exploiting slack to reduce energy by, for instance, reducing frequency of non-critical path nodes.

Systems-focused frequency scaling: In more practical software-hardware settings, the emphasis is usually on reducing energy usage through Dynamic Voltage and Frequency Scaling (DVFS). DVFS attempts to minimize energy consumption with little or no impact in performance by scaling down the frequency (and therefore the voltage) when processor speed does not limit performance [28]–[32]. This work suggests a different flavor of time-energy trade-off, which comes from the superlinear scaling of power and energy with frequency, than what we consider in this paper. Among these, Song et al. propose a particulary notable iso-energy-efficiency model for determining the problem size and clock frequency to achieve a desired level of energy-efficiency on a system of a particular size (number of processors or nodes) [32]. It is, however, not explicit about algorithmic features such as intensity. Like some of the theory work, much of this DVFS research focuses on predicting *slack* in the application which allows cores to be clocked down to save power. The perspective is systems-centric, and does not really attempt to provide programmers or algorithm designers with any insight into what they can do to make programs more energy efficient.

Profiling and observation-based studies: There are a number of empirical studies of time, power, and energy in a variety of computational contexts, such as linear algebra and signal processing [33]–[38]. One notable example is the work of Dongarra et al., which observes the energy benefits of mixed-precision [33].

Esmaeilzadeh et al. measure chip power for a variety of applications, with a key high-level finding being the highly application-dependent behavior of power consumption [38]. They create abstract profiles to capture the differing characteristics of these applications. However, they do not ascribe specific properties of a computation in a way that programmers or algorithm designers can use to understand and change time-energy behavior.

Tools: Although we adopted PowerMon 2 as our measurement infrastructure, there are numerous other possibilities. Perhaps the most sophisticated alternative is PowerPack [36], a hardware-software "kit" for power and energy profiling. However, the infrastructure is relatively elaborate and expensive to acquire, in contrast to PowerMon 2. In future studies, we expect to be able to use even simpler measurement methods based on vendor-provided hardware support. These include Intel hardware counters for power [39] and NVIDIA's Management Library for power measurement [40].

Other modeling approaches: The direct inspiration for this paper comes from studies of architecture-cognizant extensions to Amdahl's Law, balance, and the time-based roofline [1], [4]–[12].

However, there are numerous other approaches. For instance, numerous recent studies have developed detailed GPU-specific models [41]–[43]; though these models are capable of directly predicting time, they require detailed characterizations of the input program and/or intimate knowledge of the GPU microarchitecture. As such, it is non-trivial to translate the output of these models into actionable algorithmic or software changes. There are also numerous models that try to incorporate power and energy [31], [32], [44]–[47]. However, like the time-based models, much of this work is systems-centric, abstracting away algorithmic properties.

Metrics: Our models reason directly about the basic measures of time, energy, and power. When considering trade-offs and multiobjective optimization, other metrics may be better suited. These include the energy delay product (EDP) and generalizations [48], [49], FLOP/s per Watt (i.e., flops per Joule) [50], and The Green Index [51].

VII. Conclusions and Future Directions

In our view, the most interesting outcome of our analysis is the balance gap, or the difference between the classical notion of time-balance, B_τ, and its energy analogue, B_ϵ. We believe balance gaps have important consequences for algorithms, as we sketch below. Today, $B_\tau > B_\epsilon$, due largely to constant power and other microarchitectural inefficiencies; consequently, race-to-halt strategies will be the most reasonable first-order technique to save energy. Will this change significantly in the future?

Work-communication trade-offs: Suppose in the future that there is a significant balance gap, with $B_\epsilon > B_\tau$ and $\pi_0 = 0$. An interesting class of algorithms to consider are those exhibiting a *work-communication trade-off*. That is, denote an abstract algorithm by the pair (W, Q), where the algorithm performs W flops and Q mops. A "new" algorithm $(fW, \frac{Q}{m})$ exhibits a *work-communication trade-off* with respect to the baseline (W, Q) if $f > 1$ and $m > 1$. A natural question to ask in our model is the following: under what conditions on f and m should we expect a speedup, a "greenup" (improvement in energy-efficiency), both, or neither?

For example, let $E_{f,m}$ be the energy of the new algorithm so that $E_{1,1}$ is the energy of the baseline. Denote the greenup, analogous to a speedup, by $\Delta E \equiv E_{1,1}/E_{f,m}$. One can easily show that a general condition for a greenup $\Delta E > 1$ is

$$f < 1 + \frac{m-1}{m}\frac{B_\epsilon}{I}, \tag{10}$$

where I is that of baseline. This condition has a hard upper-limit. In particular, even if we can eliminate communication entirely ($m \to \infty$), the amount of extra work is bounded by $f < 1 + \frac{B_\epsilon}{I}$. When the baseline algorithm is already compute-bound in time, so that $I \geq B_\tau$, then $f < 1 + \frac{B_\epsilon}{B_\tau}$. One can imagine more analyses and experiments to determine when such a greenup will also accompany a speedup, which we are purusing as a part of our on-going work. Some of this additional analysis appears in our companion technical report [52].

Limitations: The model is just a first-cut at bridging algorithm and architecture analysis. Regarding its limitations, these are the most important in our view. First, we have suppressed latency costs, under the assumption of sufficient concurrency; we have considered such costs in prior work [1] and plan to extend it for energy. Secondly, we consider just the two-level memory hierarchy; however, our analysis in §V suggests how to incorporate such costs, thereby improving model accuracy and enabling deeper inferences about architecture. Thirdly, we ignored power caps, which can cause our analysis to overestimate power consumption and performance (§V). Having said that, at least the predictions appear empirically to give upper-bounds on power and lower-bounds on time. In spite of these limitations, we hope algorithm designers, performance tuners, and architects will find it an interesting starting point for identifying potential new directions lying at the intersection of algorithms and architecture.

Acknowledgements

This paper is an abridged version of a longer technical report [52].

We thank Marat Dukhan for his CPU polynomial code, as well as Kenneth Czechowski and Aparna Chandramowlishwaran for their comments, encouragement, and feedback. Thanks also to Hyesoon Kim for the PCIe interposer. This work was supported in part by the National Science Foundation (NSF) under NSF CAREER award number 0953100 and joint NSF 0903447 and Semiconductor Research Corporation (SRC) Award 1981; the U.S. Dept. of Energy (DOE), Office of Science, Advanced Scientific Computing Research under award DE-FC02-10ER26006/DE-SC0004915, and the Scientific Discovery through Advanced Computing (SciDAC) program under award DE-FC02-06ER25764 and DE-FG02-11ER26050/DE-SC0006925; and grants from the Defense Advanced Research Projects Agency (DARPA) Computer Science Study Group program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, DOE, or DARPA.

References

[1] K. Czechowski, C. Battaglino, C. McClanahan, A. Chandramowlishwaran, and R. Vuduc, "Balance principles for algorithm-architecture co-design," in *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar)*, Berkeley, CA, USA, May 2011.

[2] R. Vuduc and K. Czechowski, "What GPU computing means for high-end systems," *IEEE Micro*, vol. 31, no. 4, pp. 74–78, July/August 2011.

[3] K. Czechowski, C. McClanahan, C. Battaglino, K. Iyer, P.-K. Yeung, and R. Vuduc, "On the communication complexity of 3D FFTs and its implications for exascale," in *Proc. ACM Int'l. Conf. Supercomputing (ICS)*, San Servolo Island, Venice, Italy, June 2012.

[4] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[5] D. H. Woo and H.-H. S. Lee, "Extending Amdahl's Law for energy-efficient computing in the many-core era," *IEEE Computer*, vol. 41, no. 12, pp. 24–31, Dec. 2008.

[6] T. Zidenberg, I. Keslassy, and U. Weiser, "Multi-Amdahl: How Should I Divide My Heterogeneous Chip?" *IEEE Computer Architecture Letters*, pp. 1–4, 2012.

[7] H. T. Kung, "Memory requirements for balanced computer architectures," in *Proceedings of the ACM Int'l. Symp. Computer Architecture (ISCA)*, Tokyo, Japan, 1986.

[8] W. D. Hillis, "Balancing a Design," *IEEE Spectrum*, 1987.

[9] R. W. Hockney and I. J. Curington, "f1/2: A parameter to characterize memory and communication bottlenecks," *Parallel Computing*, vol. 10, no. 3, pp. 277–286, May 1989.

[10] G. E. Blelloch, B. M. Maggs, and G. L. Miller, "The hidden cost of low bandwidth communication," in *Developing a Computer Science Agenda for High-Performance Computing*, U. Vishkin, Ed. New York, NY, USA: ACM, 1994, pp. 22–25.

[11] J. McCalpin, "Memory Bandwidth and Machine Balance in High Performance Computers," *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter*, Dec. 1995.

[12] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, p. 65, Apr. 2009.

[13] H. Jia-Wei and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing - STOC '81*. New York, New York, USA: ACM Press, May 1981, pp. 326–333.

[14] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep. 2011.

[15] M. A. Awan and S. M. Petters, "Enhanced Race-To-Halt: A Leakage-Aware Energy Management Approach for Dynamic Priority Systems," in *2011 23rd Euromicro Conference on Real-Time Systems*. IEEE, Jul. 2011, pp. 92–101.

[16] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon 2: Fine-grained, integrated measurement," RENaissance Computing Institute, University of North Caroliina, Chapel Hill, NC, USA, Tech. Rep., 2009.

[17] L. GREENGARD and V. ROKHLIN, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, Dec. 1987.

[18] A. Chandramowlishwaran, J. W. Choi, K. Madduri, and R. Vuduc, "Towards a communication optimal fast multipole method and its implications for exascale," in *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA)*, Pittsburgh, PA, USA, June 2012, brief announcement.

[19] NVIDIA, "Compute Visual Profiler," 2012.

[20] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*, 1st ed. Morgan and Claypool Publishers, 2008.

[21] J. Demmel, A. Gearhart, O. Schwartz, and B. Lipschitz, "Perfect strong scaling using no additional energy," University of California, Berkeley, CA, USA, Tech. Rep., 2012.

[22] A. Tyagi, "Energy-Time Trade-offs in VLSI Computations," in *Foundations of Software Technology and Theoretical Computer Science*, vol. LNCS 405, 1989, pp. 301–311.

[23] A. J. Martin, "Towards an energy complexity of computation," *Information Processing Letters*, vol. 77, no. 2–4, pp. 181–187, Feb. 2001.

[24] R. Jain, D. Molnar, and Z. Ramzan, "Towards a model of energy complexity for algorithms," in *IEEE Wireless Communications and Networking Conference, 2005*. IEEE, 2005, pp. 1884–1890.

[25] B. D. Bingham and M. R. Greenstreet, "Computation with Energy-Time Trade-Offs: Models, Algorithms and Lower-Bounds," *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 143–152, Dec. 2008.

[26] V. A. Korthikanti and G. Agha, "Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures," in *2009 International Conference on Parallel Processing*. Vienna, Austria: IEEE, Sep. 2009, pp. 212–219.

[27] G. Aupy, A. Benoit, and Y. Robert, "Energy-aware scheduling under reliability and makespan constraints," INRIA, Grenoble, France, Tech. Rep. October, 2011.

[28] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 34–.

[29] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 835–848, Jun. 2007.

[30] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the gpu," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, dec. 2010, pp. 221 –228.

[31] R. Ge and K. Cameron, "Power-aware speedup," in *In Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 07)*, 2007.

[32] S. Song, M. Grove, and K. W. Cameron, "An iso-energy-efficient approach to scalable system power-performance optimization," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 262–271.

[33] J. Dongarra, H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architecture," in *The 2nd International Conference on Cloud and Green Computing*, Nov. 2012.

[34] X. Feng, R. Ge, and K. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, april 2005, p. 34.

[35] G. Bosilca, H. Ltaief, and J. Dongarra, "Power profiling of Cholesky and QR factorizations on distributed memory systems," *Computer Science - Research and Development*, pp. 1–9, 2012.

[36] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 21, no. 5, pp. 658–671, May 2010.

[37] H. Ltaief, P. Luszczek, and J. Dongarra, "Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency," *Computer Science - Research and Development*, pp. 1–11, 10.1007/s00450-011-0191-z.

[38] H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley, "Looking back and looking forward: power, performance, and upheaval," *Commun. ACM*, vol. 55, no. 7, pp. 105–114, Jul. 2012.

[39] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, March-April 2012.

[40] NVIDIA, "NVML API Reference Manual," 2012.

[41] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2009, pp. 152–163.

[42] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W. mei W. Hwu, "An adaptive performance modeling tool for GPU architectures," *SIGPLAN Not.*, vol. 45, no. 5, pp. 105–114, Jan. 2010.

[43] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for gpu architectures," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 382–393.

[44] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron, "Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems," *Computer Science - Research and Development*, pp. 1–9, 2011, 10.1007/s00450-011-0190-0.

[45] B. Subramaniam and W.-C. Feng, "Statistical power and performance modeling for optimizing the energy efficiency of scientific computing," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, ser. GREENCOM-CPSCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 139–146.

[46] S. Hong and H. Kim, "An integrated GPU power and performance model," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 280–289, Jun. 2010.

[47] C. Su, D. Li, D. Nikolopoulos, K. Cameron, B. de Supinski, and E. Leon, "Model-based, memory-centric performance and power optimization on numa multiprocessors," in *IEEE International Symposium on Workload Characterization*, Nov. 2012.

[48] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, sep 1996.

[49] C. Bekas and A. Curioni, "A new energy-aware performance metric," in *Proceedings of the International Conference on Energy-Aware High-Performance Computing (EnA-HPC)*, Hamburg, Germany, Sep. 2010.

[50] S. Sharma, C.-H. Hsu, and W.-C. Feng, "Making a case for a Green500 list," in *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[51] B. Subramaniam and W.-C. Feng, "The Green Index: A metric for evaluating system-wide energy efficiency in HPC systems," in *8th IEEE Workshop on High-Performance, Power-Aware Computing (HPPAC)*, Shanghai, China, May 2012.

[52] J. W. Choi and R. Vuduc, "A roofline model of energy," Georgia Institute of Technology, School of Computational Science and Engineering, Atlanta, GA, USA, Tech. Rep. GT-CSE-12-01, December 2012.