Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

A roofline model of energy

Jee Whan Choi* <jee@gatech.edu>
Richard Vuduc† <richie@gatech.edu>

Technical Report # GT-CSE-2012-01

Georgia Institute of Technology
† College of Computing, School of Computational Science and Engineering
* College of Engineering, School of Electrical and Computer Engineering
hpcgarage.org

ABSTRACT

We describe an energy-based analogue of the time-based roofline model of Williams, Waterman, and Patterson (*Comm. ACM*, 2009). Our goal is to explain—in simple, analytic terms accessible to algorithm designers and performance tuners—how the time, energy, and power to execute an algorithm relate. The model considers an algorithm in terms of operations, concurrency, and memory traffic; and a machine in terms of 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.

Subject area: High-Performance Computing

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

API Application Programmer Interface

DRAM dynamic random access memory

DVFS dynamic frequency and voltage scaling

flops floating-point operations

FMA fused multiply-add

FMM fast multipole method

FMM_U fast multipole method U-list phase

GPU graphics co-processor

mops memory operations

PCIe Peripheral Component Interconnect Express

PSU power supply unit

SIMD single-instruction multiple data

SIMT single-instruction multiple thread

SM streaming multiprocessor

SRAM static random access memory

TDP Thermal design power

1 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 [12, 13, 48], 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 [24, 50, 52]. (We review this work and numerous other related studies in § 7.) 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 [9, 25, 26, 35, 39, 49]. 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 "useful" compute operations and bytes *per unit-energy* (e.g., Joules). We compare balancing computations in time against balancing in energy. [§ 2]

Secondly, we use energy-balance to develop an energy-based analogue of the time-based roofline model [49]. 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. [§ 2, § 3]

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. [§ 2, § 4]

Fourthly, we test the basic form of the model using experiments on real CPU and graphics co-processor (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 [4]. This raises the question for architects and hardware designers about what the fundamental *trends* in the balance gap will be: if energy-balance will eventually overtake time-balance, race-to-halt could break. We further use the experiments to highlight both the strengths and the limitations of our model and analysis. [§ 4, § 5, § 8]

Lastly, we ask under what general conditions we should expect an *algorithmic* time-energy trade-off. "Algorithmic" here stands in contrast to "architectural." Architecturally, for instance, increasing frequency reduces time but increases energy, due to the non-linear relationship between frequency and power. What is the story for algorithms? We consider one scenario. Suppose it is possible algorithmically to trade more compute operations for less communication. One may derive a general necessary condition under which such a trade-off will improve energy-efficiency. Furthermore, we show what improvements in energy-efficiency may or may not require a slowdown, and by how much. Again, these conditions depend fundamentally on how time-balance compares to energy-balance. [§ 6]

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.

2 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.

2.1 *Algorithm characterization*

Let W be the total number of "useful" compute operations that the algorithm performs and let Q be the total amount of data it transfers between the slow and fast memories. (Table 1 summarizes all

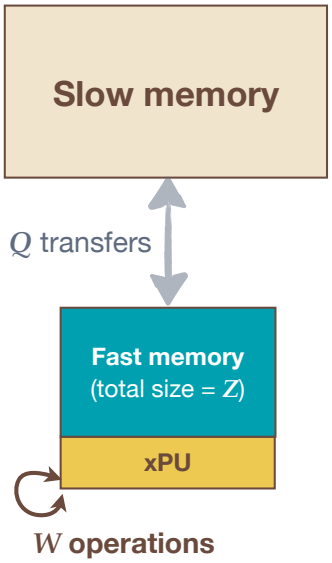


Figure 1: A simple von Neumann architecture with a two-level memory hierarchy. In our first analysis, suppose that an algorithm performs W arithmetic operations and Q memory operations, or "mops," between slow and fast memories.

of the parameters of our model.) By *useful*, we mean in an algorithmic sense; for example, we might only count floating-point operations (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 scalar flops. (That is, a 4-way single-instruction multiple data (SIMD) add is 4 scalar flops; a fused multiply-add (FMA) is two scalar 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 memory operations (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.

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.

Variable	Description
W	# of useful compute 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
$\hat{\epsilon}_{\text{flop}} \equiv \epsilon_{\text{flop}} + \epsilon_0$	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 ($\pi_0 \geqslant 0$)
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 1: 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 $I = \mathcal{O}\left(\sqrt{Z}\right)$ [30]. 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, the concept of intensity measures the inherent locality of an algorithm.

2.2 *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*, which is an additional minimum baseline energy on top of operation and data movement costs. These distinctions are critical, and together determine whether or not one should expect an algorithmic time-energy trade-off (see § 6).

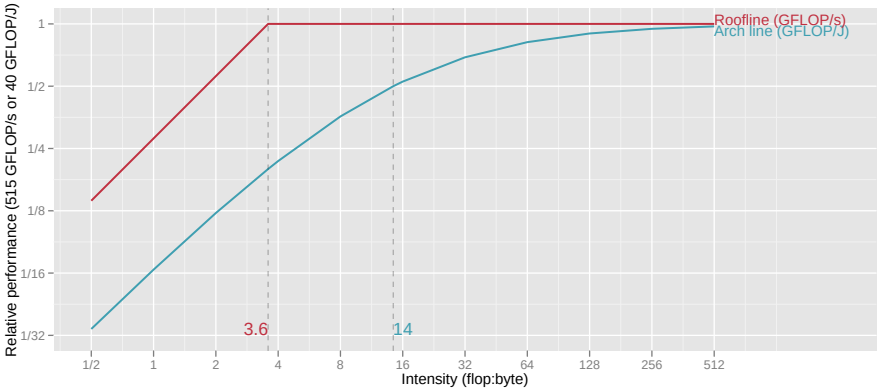
More formally, suppose T_{flops} and T_{mem} are the total time (seconds) to execute all work operations and all mops, respectively. Further assume, *optimistically*, that overlap is possible. Then, the total time T is

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

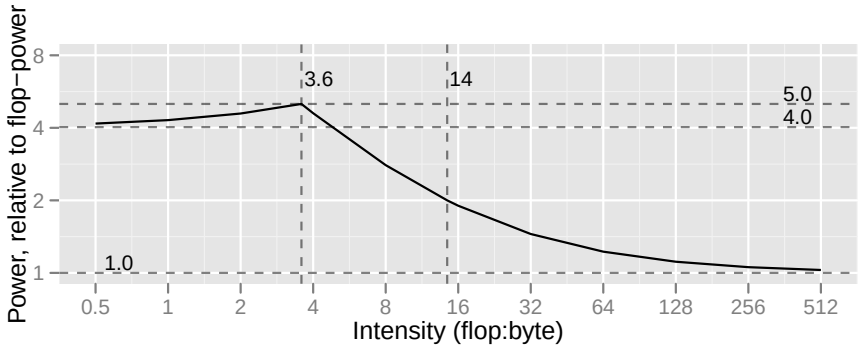
Similarly, suppose that E_{flops} and E_{mem} are the total energy (Joules) for work and mops. In addition, let $E_0(T)$ be the *constant energy* of the computation. Constant energy is the energy that must be expended for the duration of the computation, which we will further assume is a fixed cost independent of the type of operations being performed. (see § 3 for a more detailed description of the constant term). 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



(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 2, for an NVIDIA Fermi-class GPU assuming constant power is 0. Dashed vertical lines show time- and energy-balance points.

Table 2: Sample values for model parameters, based on best case (peak) capabilities of currently available systems. See table 1 for a summary of the definitions of these parameters.

Variable	Representative values NVIDIA "Fermi" GPU [48]
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.

latency-based values. (See table 2 for sample parameters.) This assumption will yield 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 [12]. 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* [9, 25, 26, 35, 39, 49]. 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* [12]. Our algorithmic design goal is to create algorithms that minimize time and have high intensity relative to machine's time-balance.

We might hypothesize that a reasonable cost model of energy is similar to that of time. Suppose each work operation has a fixed energy cost, ϵ_{flop}, and for each mop a fixed cost, ϵ_{mem}. Additionally, suppose

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

the constant energy cost is linear in T, with a fixed *constant power* of π_0 units of energy per unit time. (This notion of constant power differs from leakage power; see § 2.5.) 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. When π_0 is zero, B_ϵ is the intensity value at which flops and mops consume equal amounts of energy.

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*,

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

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.

2.3 *Rooflines in time and energy*

We can visualize the balance principles of eqs. (3) and (5) using a roofline diagram [26, 49]. 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 2. Keckler et al. presented these values for an NVIDIA Fermi-class GPU [33]; but since they did not provide estimates for constant power, for now assume $\pi_0 = 0$. The x-axis shows

intensity I. The y-axis shows performance, normalized either by the maximum possible speed (flops per unit time) or by the maximum possible energy-efficiency (flops per unit energy). 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 $W\hat{\epsilon}_{\text{flop}}/E = 1/(1 + \hat{B}_\epsilon(I)/I)$. In both cases, the best possible performance is the time or energy required by the flops alone.

The roofline for speed 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 $I \geqslant B_\tau$ means the computation is compute-bound in time. Assuming that all possible implementations have the same number of operations, the algorithm designer or code tuner should minimize time by maximizing intensity according to the balance principle.

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." The energy-balance point $I = B_\epsilon$ is the intensity 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 point at which mops no longer dominate the total energy. In this sense, an algorithm may be compute-bound or memory-bound in *energy*, which will differ from time when $B_\tau \neq B_\epsilon$.

2.4 *The balance gap*

The aim of rooflines and arches is to guide optimization. Roughly speaking, an algorithm or code designer starts with some baseline 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 $B_\tau < B_\epsilon$. The reason is that wire length is not expected to scale with feature size [33]. An algorithm with $B_\tau < I < B_\epsilon$ is *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.

3 This relationship follows from $a + b \leqslant 2\max(a, b)$.

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—would not in general hold. 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. Under this condition, so-called *race-to-halt* strategies for saving energy will tend to be effective [4].[4]

Lastly, 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.

2.5 *Interpreting constant power*

Constant power in our model differs from conventional notions of static (or leakage) power and dynamic power [3]. Static power is power dissipated when current leaks through transistors, even when the transistors are switched off; dynamic power is power due to switching (charging and discharging gate capacitance). These terms refer primarily to the device physics behind processor hardware, whereas constant power in our model represents a more abstract concept that depends on hardware *and* the algorithm or software that runs atop it.

With respect to hardware, constant power includes everything that is required to operate the device on top of leakage power. For example, constant power on a GPU will also include chips and circuitry on the printed circuit board, cooling fan, and other parts of the microarchitecture. These components may or may not be directly involved in computing or fetching data but would need to be on for the GPU to run at all.

Our constant power model does not explictly express the concept of dynamic power, which may lead to measurable inaccuracy. However, hardware designers are aggressively implementing techniques that can, for instance, turn off unused cores or aggressively gate clocks. These and other techniques tend to significantly reduce the impact of dynamic power.

With respect to software, our model of constant power can also capture inefficiencies in the algorithm or code. If a program is running inefficiently due to not having enough threads to saturate the processors or the memory units, there will be unused cores and/or longer instruction latencies due to stalls. The model includes such inefficien-

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.

cies through by charging an constant energy cost that depends on constant power and the total running time T.

3 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* of a computation, $P \equiv E/T$.

Let $\pi_{\text{flop}} \equiv \epsilon_{\text{flop}}/\tau_{\text{flop}}$ be the *power per flop*. This definition excludes constant power. Then, dividing eq. (5) by eq. (3) yields

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

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

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

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 AN EXPERIMENT

The model of § 2 is an hypothesis about the relationships among intensity, time, and energy. This section tests our model on real systems.

4.1 *Experimental Setup*

HARDWARE Table 3 shows our experimental platforms, which include an Intel quad-core Nehalem CPU and two high-end consumer-class GPUs (Fermi and Kepler). We use two tools to measure power. The first is PowerMon 2, a fine-grained integrated power measurement device for measuring CPU and host component power [6]. The second is a custom in-house PCIe interposer for measuring GPU power. At the time of this writing, our consumer-grade NVIDIA GPU hardware did not support fine-grained power measurement via NVML, NVIDIA's Application Programmer Interface (API) [40].

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 measures direct current and voltage on up

Device	Model	Peak performance Single (Double) GFLOP/s	Peak memory bandwidth GB/s	TDP Watts
CPU	Intel Core i7-950 (Nehalem)	106.56 (53.28)	25.6	130
GPU 1	NVIDIA GeForce GTX 580 (Fermi)	1581.06 (197.63)	192.4	244
GPU 2	NVIDIA GeForce GTX 680 (Kepler)	3532.8 (147.2)	192.2	190

Table 3: Platforms – TDP is Thermal design power.

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.

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 benchmarking the GPU, 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

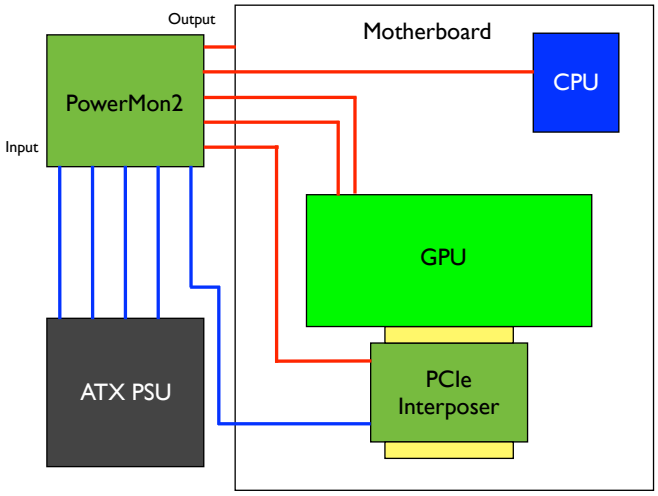


Figure 3: Placement of the measurement probes, PowerMon 2 [6] and our custom PCIe interposer

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.

4.2 *Intensity microbenchmarks*

We created microbenchmarks that allow us to vary intensity, and tuned them to achieve very high fractions of the peak FLOP/s or bandwidth that the roofline predicts. We then compared measured time and power against our model. The results for double-precision and single-precision appear in fig. 4 and fig. 5 respectively, 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 streams a multi-gigabyte array and, for each element, executes a mix of independent FMA operations. We autotuned 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. To verify the implementation, we inspected the PTX code and compared 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

5 Pending review of this paper, we will release the benchmarks publicly.

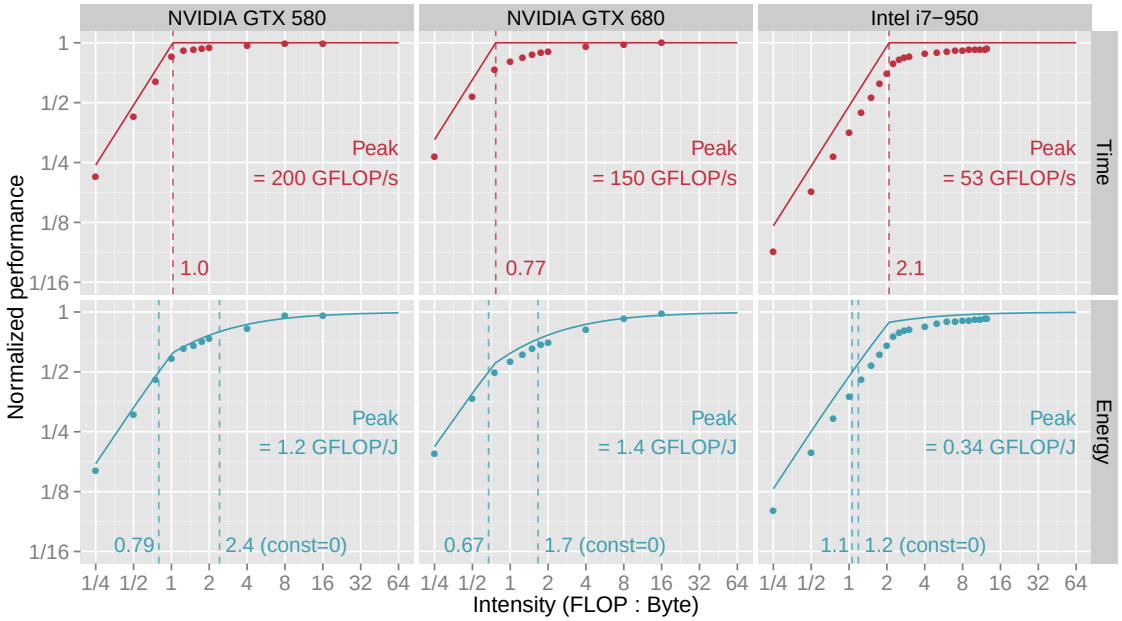


Figure 4: Measured time and energy for a double-precision synthetic benchmark corroborates eqs. (3) and (5). The impact of constant energy can be profound: GPUs have $\hat{B}_\epsilon < B_\tau < B_\epsilon$ (see vertical dashed lines). In other words, time-efficiency implies energy-efficiency *because 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.

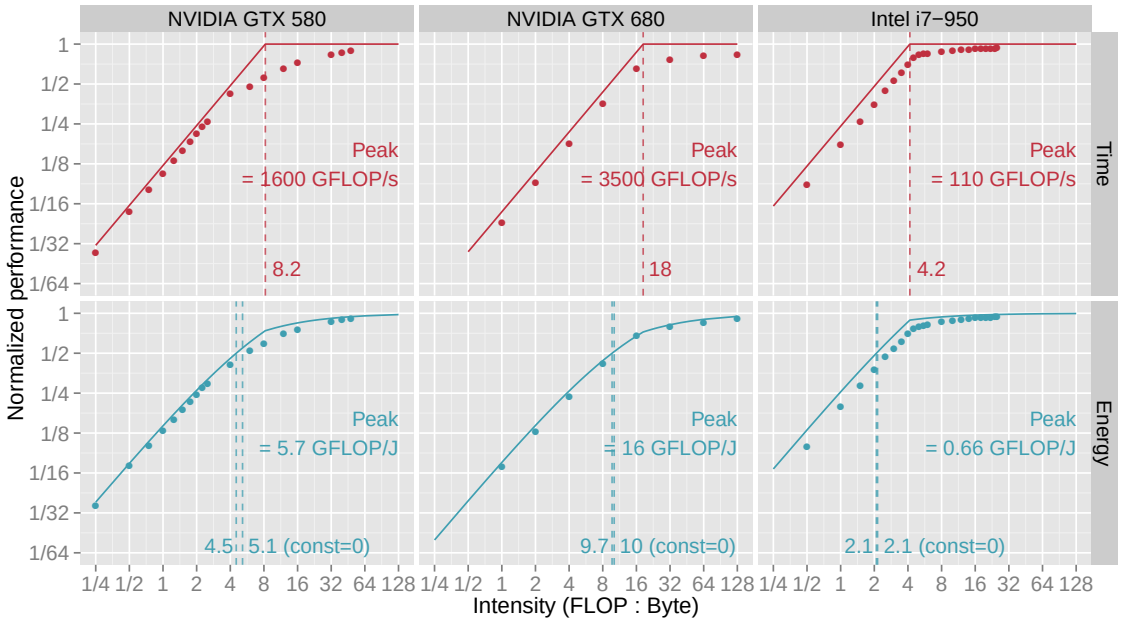


Figure 5: Same as fig. 4, but for single-precision. In this case, all platforms have $\hat{B}_\epsilon \leqslant B_\epsilon < B_\tau$.

varies the computation's intensity. 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.

Figures 4 and 5 show that both microbenchmarks perform 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 3. For instance, on the GTX 580 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 1.4 TFLOP/s respectively. However, performance does not always match the roofline. Again on the NVIDIA GTX 580, we see the largest gap near the time-balance point; this gap is much smaller on the GTX 680. We revisit this phenomenon in light of our model in § 5.2.

Percentage of system peak performance observed on GTX 680 was somewhat lower than that of GTX 580 and it took slightly more effort and tuning to achieve it. The maximum observed bandwidth was 147 GB/s, or 76.6% of system peak in both single-precision and double-precision. The maximum observed performance when the benchmark was compute bound was 148 GFLOP/s, or 101% of system peak in double-precision and 3 TFLOP/s, or 85.6% of system peak in single-precision. These percentages are based on the 1150 MHz "Boost Clock" specification. We speculate that these variations (even exceeding 100%) are due to the aggressive DVFS employed on Kepler GPUs which allows over-clocking from the "Base Clock" of 1006 MHz as long as the GPU remains under its Thermal design power (TDP), even exceeding the Boost Clock.[6]

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 similar to that of the STREAM benchmark[7] 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 3. For the energy costs in eq. (5), such specifications do not exist. Therefore, we esti-

6 http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf

7 streambench.org

	NVIDIA GTX 680	NVIDIA GTX 580	Intel Core i7-950
ϵ_s	43.2 pJ /FLOP	99.7 pJ / FLOP	371 pJ / FLOP
ϵ_d	262.9 pJ / FLOP	212 pJ / FLOP	670 pJ / FLOP
ϵ_{mem}	437.5 pJ / Byte	513 pJ / Byte	795 pJ / Byte
π_0	66.37 Watts	122 Watts	122 Watts

Table 4: Fitted energy coefficients. Note that ϵ_{mem} is given in units of pico-Joules per *Byte*. As it happens, the π_0 coefficients turned out to be identical to three digits on GTX 580 and i7-950 which are built on 40 nm and 45 nm technologies respectively, whereas GTX 680 is built on a significantly lower technology of 28 nm.

mated them using linear regression on our experimental data.[8] 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 $\Delta\epsilon_d$, which is the *additional* energy required for a double-precision flop over a single-precision flop.[9] That is, the energy per double-precision flop is $\epsilon_d \equiv \epsilon_s + \Delta\epsilon_d$. We summarize the fitted parameters in table 4. We then plug these coefficients into eq. (5) to produce the model energy curves shown in fig. 5 and fig. 4. These curves visually confirm that the fitted model captures the general trend in the data. We analyze these curves in § 5.

4.3 *Cache microbenchmarks*

Exploiting data locality is the main algorithmic tool for controlling I, though so far we have ignored the cost of explicit cache access. Let Q_{cache} be the number of cache accesses (measured in words) that our computation incurs, which are distinct from the Q that we may assume all go to main memory. We may modify eq. (4) to account for Q_{cache} as follows, assuming a per-cache access energy cost of ϵ_{cache}:

$$E = W\epsilon_{flop} + Q\epsilon_{mem} + Q_{cache}\epsilon_{cache} + \pi_0 T. \tag{10}$$

This equation assumes the two-level memory hierarchy of fig. 1. It would be straightforward to add terms for a memory hierarchy with more than two levels.

8 We use the standard regression routine in R, `r-project.org`.

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

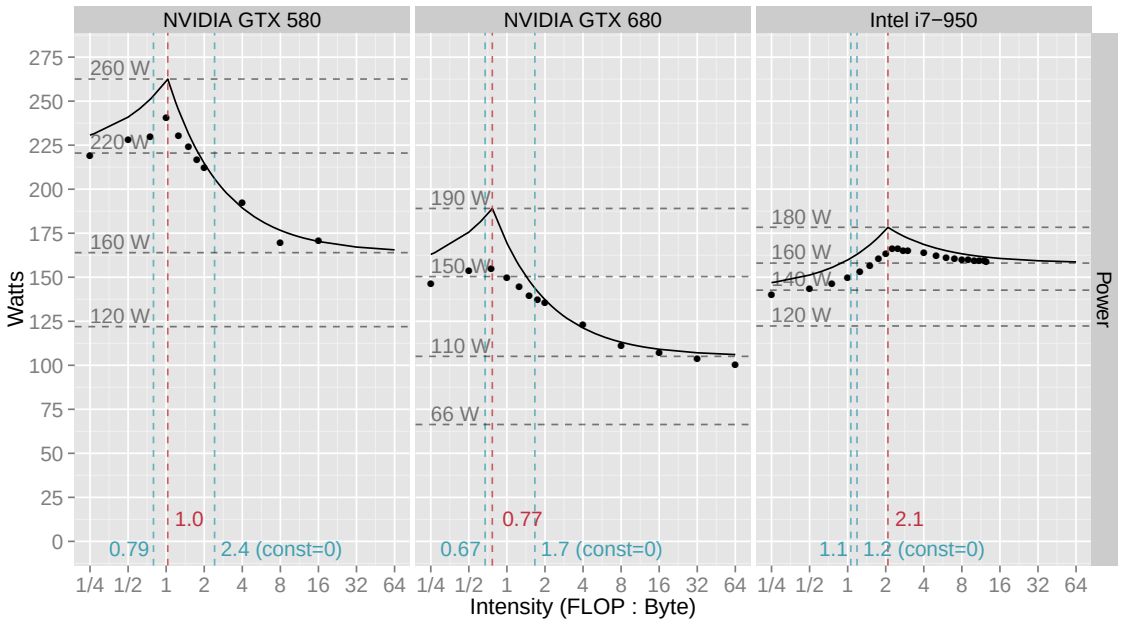


Figure 6: Measured power for the double-precision microbenchmark corroborates the "powerline" model. On the GTX 580 platform, NVIDIA reports a limit of 244 Watts, which explains the discrepancy between the observed data and the predicted powerline in the single-precision GTX 580 case.

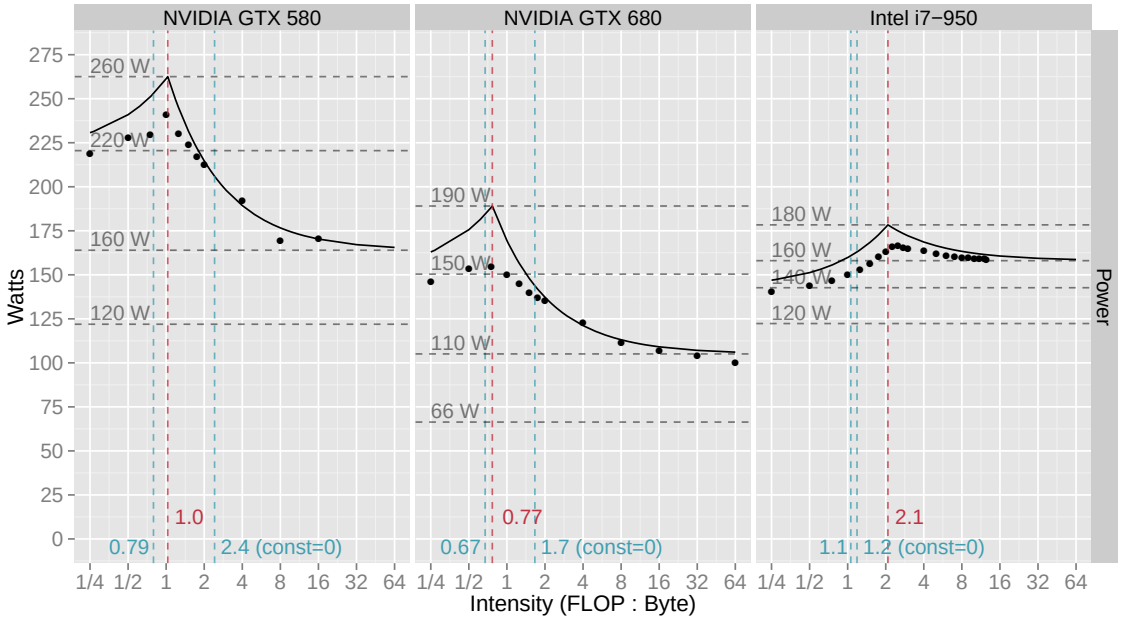


Figure 7: Measured power for the single-precision microbenchmark .

We created a pointer-chasing microbenchmark for the GPU to help measure cache access energy. First, consider the last-level L2 cache on Fermi- and Kepler-class GPUs. Our benchmark computes `k = A[k]`, where initially `k == A[k]` == the local thread ID. As such, the first thread block to access the array will fetch data from main memory into cache; threads from other thread blocks with the same local thread ID effectively fetch that same element over and over from cache. The number of threads is limited so that the entire array fits in cache. We compile the microbenchmark with `-Xptxas -dlcm=cg`, which forces the compiler to generate code that caches data in the L2 cache only. Let ϵ_{L_2} be the L2 cache energy access cost; using our previously fitted values of ϵ_{mem} and π_0, we can compute ϵ_{L_2} via

$$\epsilon_{L_2} = \frac{E - W\epsilon_{\text{flop}} - Q\epsilon_{\text{mem}} - \pi_0 T}{Q_{L_2}} \tag{11}$$

where we use performance counters to estimate the number of L2 accesses, Q_{L_2}.

We can adjust then reuse this microbenchmark to estimate L1 cache energy cost. First, recall that the GPU L1 caches are private to each multiprocessor. When L1 caching is enabled, the very first thread block to fetch a particular array element will load the corresponding cache line into both the L2 cache and the L1 cache of the multiprocessor in which it resides. Then, all *subsequent and first* thread blocks in each multiprocessor will fetch the array from the L2 cache into their respective L1 caches. After that, subsequent thread blocks will then access this array from their own L1 caches. Since we have already calculated the cost of fetching data from the L2 cache (ϵ_{L_2}), the energy cost ϵ_{L_1} of fetching data from the L1 cache can be computed using

$$\epsilon_{L_1} = \frac{E - W\epsilon_{\text{flop}} - Q\epsilon_{\text{mem}} - \pi_0 T - Q_{L_2}\epsilon_{L_2}}{Q_{L_1}}. \tag{12}$$

This scheme works for NVIDIA Fermi-class GPUs, but we must modify it for those based on Kepler. There, the L1 cache is reserved exclusively for spilling local data.[10] Therefore, we instead estimate the energy cost of fetching data from the explicitly-programmed scratch-pad memory, or "shared memory" in NVIDIA CUDA parlance. However, doing so will have two consequences. First, we may observe more L2 cache accesses than we would if we used L1. The reason is that a copy of the array is required *per thread block*, rather than *per multiprocessor*). Secondly, we may observe different effective energy costs, since the shared memory and L1 cache hardware implementations differ. The reader should, therefore, interpret "L1" energy cost estimates on Kepler accordingly.

The estimated costs of fetching data from L1 and L2 caches on Fermi and Kepler-class GPUs appear in table 5, where ϵ_{L_1} is the L1 (or shared

10 See: http://docs.nvidia.com/cuda/kepler-tuning-guide/index.html

	NVIDIA GTX 680	NVIDIA GTX 580
ϵ_{L_1}	51 pJ /Byte (shared memory)	149 pJ / Byte
ϵ_{L_2}	195 pJ / Byte	257 pJ / Byte

Table 5: Estimated energy costs of explicitly fetching data from L1 and L2 caches on GTX 580, and from shared memory and L2 on the GTX 680.

memory) access cost, and ϵ_{L_2} is the L2 cost. The median relative residuals between the fitted model and the measured microbenchmark data were less than 4.5% on the GTX 680, and less than 4% on the GTX 580.

As expected, the costs of fetching data from L1 and L2 caches is much smaller on the Kepler-class GPU. Note that Kepler-based GPUs use a better process technology than Fermi (28 nm vs. 40 nm on Fermi). We will analyze these estimates in § 5.1.

5 DISCUSSION, APPLICATION, AND REFINEMENT

5.1 *The fitted parameters*

We may compare the fitted parameters of table 4 against those that Keckler et al. provide [33]. Since they only discuss Fermi-class GPUs, giving estimates for only 45 nm and 10 nm process technologies, we will limit the following discussions of our energy cost estimates to the GTX 580.

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 4 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 Keckler et al.'s paper, 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.

Another point of note is the cost of fetching data from the L1 and L2 caches. Since the cost of reading the data from SRAM is only 1.75 pJ/Byte and the cost of transferring the data over the wire is roughly 10–20 pJ/Byte, instruction overhead accounts for most of the cost. Streamlining the microarchitecture to reduce this overhead is an opportunity for future work.

There is no information provided to check our constant power estimate. For reference, we measured true GPU idle power—when the GPU is on but not running anything—to be approximately 40 Watts. Thus, what we estimate as constant power is not the same as idle power.

The estimates of CPU energy costs for both flops and memory are higher than their GPU counterparts. This observation is not surprising since a CPU processing core is widely regarded as being more complex than its GPU counterpart. Similarly, memory energy costs are higher in the CPU system than the GPU system. A likely explanation is that GPU memory sits closer to the GPU processor than CPU memory does to the CPU processor. All of these characteristics have a profound impact on the balance gap, discussed next.

5.2 *Balance gaps and power caps*

Consider the rooflines and arch lines of fig. 4 and fig. 5. 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 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 in practice on today's systems [4].

If instead it were possible to drive $\pi_0 \to 0$, then the situation could reverse. In the two bottom-left subplots of fig. 4, we show this scenario using the hypothetical energy-balance lines labeled, "const=0." However, also observe that having $\pi_0 = 0$ does not invert the balance gap on the Intel platform. As table 4 suggests, ϵ_{flop} and ϵ_{mem} on the Intel platform are much closer than they are on the NVIDIA platform. Reflecting on these two types of systems, one question is to what extent π_0 will go toward 0 and to what extent microarchitectural inefficiences will reduce.

As noted previously, the single-precision NVIDIA GTX 580 performance in fig. 5 does not track the roofline closely in the neighbor-

hood of B_τ. The reason is that our model does not include explicit power caps. To see this effect, refer to the powerlines of fig. 6 and fig. 7 and the theory of fig. 2b. Our model demands that power increase sharply as I approaches B_τ (see § 3). For instance, on the GPU in single-precision, our model says we will need 387 Watts on the GPU as shown in fig. 7. This demand would in reality cause excessive thermal issues. 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.

5.3 *Applying and refining the model: FMM_U on the GPU*

To see how well we can estimate time and energy using our model, we apply it to the fast multipole method (FMM). The FMM is an $O(n)$ approximation method for n-body computations that would otherwise scale as $O(n^2)$ [23]. We specifically consider the most expensive phase of the FMM, called the *U-list* phase (FMM_U). For this exercise, we consider just a GPU version of FMM_U.

Algorithm 5.1 The FMM_U algorithm

```
1: for each target leaf node, B do
2:   for each target point t ∈ B do
3:     for each neighboring source node, S ∈ U(B) do
4:       for each source point s ∈ S do
5:         (δx, δy, δz) = (tx − sx, ty − sy, tz − sz)
6:         r = δx² + δy² + δz²
7:         w = rsqrtf(r) {Reciprocal square-root}
8:         φt += ds ∗ w {ds and φt are scalars}
```

ALGORITHM SKETCH The FMM_U phase appears as pseudocode in Algorithm 5.1. 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 compute-bound.

In prior work, we generated approximately 390 different code implementations of this benchmark [11]. These variants use a variety of performance optimization techniques and tuning parameter values.

FITTING We use the values in table 4 and table 5 to estimate the total energy cost of each FMM_U implementation on the NVIDIA GTX 680. All implementations are in single-precision. We derive the number of flops from the input data and the number of bytes read from the DRAM and caches using hardware counters provided by NVIDIA's Compute Visual Profiler. The average error for estimated energy consumption as compared to measured was 32%, which is much worse than our cache microbenchmark.

There are two culprints. First, we count only flops, thereby ignoring the overhead of integer instructions, branches, and other non-flop operations. Secondly, our method of estimating flop energy does not distinguish between types of flops. Recall that our flop energy estimate is half the cost of a FMA instruction. Our simple estimate will usually underestimate the true energy consumption of multiplies, divisions, and (reciprocal) square roots, for instance. For the 390 FMM_U implementations, we always underestimated total energy.

5.4 *Refining the performance estimate*

Though compute-bound, the FMM_U instruction mix is more complex than a series of FMAs. As such, we should evaluate time and energy with respect to a refined notion of peak that accounts for the instruction mix. Below, we show how to do so for the GTX 580 (Fermi). A similar line of reasoning is possible for the GTX 680 (Kepler).

The intrinsic operations that the FMM_U must perform are as follows. There are 11 scalar flops, where reciprocal square root counts as 1 flop. These flops occur over 8 operations: 1 transcendental (reciprocal square root), 3 FMAs, and 4 individual floating-point add and subtract operations.

To achieve peak on the GTX 580 (Fermi), consider its microarchitecture. The GPU processor has 512 clusters of functional units, called "CUDA cores," spread among 16 streaming multiprocessor (SM) units. Each of these cores runs at 1.54 GHz. Thus, the processor can perform (512 scalar instructions) times (1.54 GHz) = 788 billion instructions per second. If each instruction is a scalar FMA, the peak performance is 1.58 TFLOP/s. However, there is a restriction owing to the dual-warp schedulers in each SM. A warp is a group of 32 threads that execute instructions in lock-step.[11] An SM may select 1 instruction each from 2 independent warps in a cycle. That is, to achieve peak there must be

11 This style of execution is sometimes called single-instruction multiple thread (SIMT) execution.

at least 2 independent warps each with a ready FMA in every cycle. Otherwise, achieving peak is impossible.

For transcendental instructions like reciprocal square root, there are more limits. Only 1 of the two warp schedulers in an SM may issue a transcendental operation each cycle. Furthermore, there are only 4 special function units (SFUs) capable of executing such operations on each SM. Therefore, to issue a transcendental operation for a warp requires (32 threads) divided by (4 SFUs per cycle) = 8 cycles.

The key to deriving a refined estimate of peak, given the architecture and instruction mix, lies with the instruction issue rate and the dual-warp scheduler. The FMM_U instruction mix requires at least 3 cycles (for the FMAs) + 4 cycles (adds and subtracts) + 8 cycles (reciprocal square root) = 15 cycles. However, the issue rate varies during these 15 cycles: when issuing FMAs and other floating point instructions, if there is at least 1 other warp available to issue the same type of instructions, the SM will issue a total of 32 instructions. However, when issuing reciprocal square roots, no more than 4+16=20 instructions may issue if there is at least one other warp available that can issue FMAs or floating point instructions. Therefore, the average issue rate is $(7 \times 32 + 8 \times 20)/15 = 25.6$ scalar instructions per cycle. However, this does not differentiate FMAs and floating point instructions. As such, we can redefine issue rate in terms of flops/cycle. In this case, the average issue rate is $(3 \times 64 + 4 \times 32 + 8 \times 20)/15 = 32$ flops/cycle. Then the maximum performance is (32 flops/cycle/SM) $\times$ (16 SMs) $\times$ (1.54 GHz) = 788 GFLOP/s.

To verify this estimate, we wrote a synthetic benchmark where each thread executes many independent copies of this particular floating-point instruction mix in a fully unrolled loop. The performance levels off at approximately 785 GFLOP/s as we increase the size of the loop. Unfortunately, we cannot completely unroll our loops in FMM_U since we do not have any prior knowledge on the number of neighbors in the U-list, or the number of points in each neighbor. When we put our instruction mix in a loop (i.e., *not* unrolled), the performance drops to approximately 512 GFLOP/s. We believe this performance is the true peak that we should expect from the ideal FMM_U kernel. In fact, the best performance observed among our 390 kernels was 467 GFLOP/s, or 91% of this refined peak.

5.5 *Constant power*

The constant power and energy terms exhibit a large impact on energy consumption. When we ran our microbenchmark in compute-only mode, it spent 55% of its energy on flops on the GTX 580 and 69% on the GTX 680. The flop energy for FMM_U, as a result, was even worse. The FMM_U spent just 23% to 50% of its energy on flops when

running on the GTX 580, and 40% to 70.27% on the GTX 680. These relatively low fractions suggest that significant energy savings are still possible.

Additionally, the wide range of energy for flops observed for the FMM_U stress the importance of algorithmic and software-level techniques. Otherwise, we could end up *wasting* as much as 80% of the total energy judging by the 23% of energy for flops observed on the GTX 580. Beyond algorithms and software, for FMM_U specifically there is also a strong case for even more specialized function units.

6 ALGORITHMIC TRADE-OFFS

With the background of § 2, we may now ask what the consequences for algorithm design may be. One interesting case is that of a family of algorithms that exhibit a *work-communication trade-off*. Such a trade-off is one in which we can reduce memory communication at the cost of increasing computation. Below, we state when a work-communication trade-off improves time or energy, or both or neither.

The most interesting scenario will be when there exists a balance gap. As such, we will in this section assume $\pi_0 = 0$. The same analysis for $\pi_0 > 0$ appears in appendix A.

6.1 *Notation and key definitions*

Denote an abstract algorithm that executes W flops and Q mops by the pair, (W, Q). A "new" algorithm $(fW, \frac{Q}{m})$ exhibits a *work-communication trade-off* with respect to the baseline (W, Q) if $f > 1$ and $m > 1$. From § 2, the time and energy of this algorithm are

$$T_{f,m} = W\tau_{\text{flop}} \max\left(f, \frac{1}{m}\frac{B_\tau}{I}\right) \tag{13}$$

$$\text{and} \qquad E_{f,m} = W\epsilon_{\text{flop}}\left(f + \frac{1}{m}\frac{B_\epsilon}{I}\right), \tag{14}$$

respectively. The intensity $I \equiv W/Q$ is that of the *baseline* algorithm.

Higher intensity does *not* mean less time. The new algorithm's intensity is fmI by definition. Though this value is higher than that of the baseline, the new algorithm takes less time only if $1 < f < \frac{B_\tau}{I}$, i.e., the baseline was initially memory-bound.

Instead, the best way to compare times is to do so directly. As such, our analysis will also use the usual measure of *speedup*, denoted by ΔT and measured relative to the baseline:

$$\Delta T \equiv \frac{T_{1,1}}{T_{f,m}} = \frac{\max\left(1, \frac{B_\tau}{I}\right)}{\max\left(f, \frac{1}{m}\frac{B_\tau}{I}\right)}. \tag{15}$$

The algorithm exhibits a speedup if $\Delta T > 1$ and a slowdown if $\Delta T < 1$.

Similarly, we may ask whether the algorithm uses less energy than the baseline. In the same way that eq. (15) measures time-efficiency, we may measure energy-efficiency, or "greenup" ΔE, as

$$\Delta E \equiv \frac{E_{1,1}}{E_{f,m}} = \frac{1 + \frac{B_\epsilon}{I}}{f + \frac{1}{m}\frac{B_\epsilon}{I}}. \tag{16}$$

The new algorithm is more energy-efficient than the baseline if $\Delta E > 1$.

6.2 *A general "greenup" condition*

Equation (16) implies that a $(fW, \frac{Q}{m})$ algorithm decreases energy relative to the baseline when $\Delta E > 1$, or (after algebraic rearrangement)

$$f < 1 + \frac{m-1}{m}\frac{B_\epsilon}{I}. \tag{17}$$

Equation (17) is a general necessary condition for a work-communication trade-off to reduce energy.

Acccording to eq. (17), a work-communication trade-off may increase intensity but only up to a limit. Reducing communication (increasing m) increases the slack that permits extra work (higher f) to still pay-off. But the energy communication penalty imposes a hard upper-limit. In particular, even if we can eliminate communication entirely ($m \to \infty$), the amount of extra work is bounded by $f < 1 + \frac{B_\epsilon}{I}$.

To get a feeling for what this might mean, suppose we have a *baseline* computation that is compute-bound in time, and furthermore is maximally tuned. In the language of the roofline of fig. 2a, "compute-bound in time" means $I \geqslant B_\tau$, lying at or to the right of the sharp inflection; and "maximally tuned" means minimum time (or, equivalently, maximum performance), which occurs at y-values that are on the roofline itself. Equation (17) says that a new algorithm exhibiting a work-communication trade-off may reduce energy even if it requires increasing flops. However, there is a limit: even if we eliminate communication, we must have $f < 1 + \frac{B_\epsilon}{I} \leqslant 1 + \frac{B_\epsilon}{B_\tau}$. For the NVIDIA Fermi GPU architecture of table 2, this upper-bound is about five, meaning we should not increase the flops by more than a factor of five.[12] The relative gap between B_τ and B_ϵ determines the slack for extra work.

6.3 *Time and energy relationships*

Equation (15) suggests that we consider three cases in relating speedup ΔT and greenup ΔE:

1. Both the baseline algorithm and the new algorithm are memory-bound in time.

12 This factor does *not* depend asymptotically on the input size, n.

2. The baseline algorithm is memory-bound in time but the new algorithm is compute-bound in time.
3. Both the baseline algorithm and the new algorithm are compute-bound in time.

(The fourth logical case, that the baseline is compute-bound in time while the new algorithm is memory-bound in time, is impossible since we only consider $f, m > 1$.) In each of these cases, we will calculate ΔT and then ask what the resulting lower- and upper-bounds on ΔE may be. From this analysis, we will see when a speedup, a greenup, or both may be possible.

CASE 1: BASELINE AND NEW ALGORITHMS ARE MEMORY-BOUND IN TIME If the baseline and new algorithms are memory-bound in time, then $I < B_\tau$ and $f < \frac{1}{m}\frac{B_\tau}{I}$. We may conclude from eq. (15) that $\Delta T = m > 1$, where the inequality follows from assumption. In this case, we should always expect a speedup.

We can get a lower-bound on ΔE using eq. (16) and upper-bounds on f and $\frac{1}{m}$. The only upper-bound on f is $\frac{1}{m}\frac{B_\tau}{I}$. The only upper-bound on $\frac{1}{m}$ is 1. Substituting these inequalities into eq. (16), we have

$$\Delta E > \frac{1 + \frac{B_\epsilon}{I}}{\frac{B_\tau}{I} + \frac{B_\epsilon}{I}} = \frac{1 + \frac{I}{B_\epsilon}}{1 + \frac{B_\tau}{B_\epsilon}}. \tag{18}$$

Since both the numerator and denominator are greater than one but $I < B_\tau$, this lower-bound will generally be slightly less than 1, meaning a small loss in energy-efficiency can occur.

To get an upper-bound on ΔE, we need lower-bounds on f or $\frac{1}{m}$ or both. By definition, we have $f > 1$. By memory boundedness, we also have $\frac{1}{m} > f\frac{I}{B_\tau}$. Thus,

$$\Delta E < \frac{1 + \frac{B_\epsilon}{I}}{1 + \frac{B_\epsilon}{B_\tau}}. \tag{19}$$

By memory-boundedness of the baseline algorithm, $I < B_\tau$ and so the value of the upper-bound of eq. (19) will be greater than 1, but limited by the balance gap.

Equations (18) and (19) exhibit a natural symmetry and, furthermore, are independent of m and f.

CASE 2: BASELINE IS MEMORY-BOUND AND NEW ALGORITHM IS COMPUTE-BOUND If the baseline is memory-bound in time but the new algorithm is compute-bound, then $I < B_\tau$, $\frac{1}{m}\frac{B_\tau}{I} < f$, and $\Delta T = \frac{1}{f}\frac{B_\tau}{I}$. The expression for speedup says that we only expect a speedup if the increase in flops is not too large relative to the communication time penalty.

To get a lower-bound on greenup using eq. (16), we need upper-bounds on f, $\frac{1}{m}$, or both. The only such bounds are $\frac{1}{m} < 1$, by definition, and $\frac{1}{m} < f\frac{I}{B_\tau}$, by compute-boundedness of the new algorithm. Which of these is smaller depends on specific values of the various parameters; however, since both must hold we may conservatively assume either. Choosing the latter, we find

$$\Delta E \quad > \quad \frac{1}{f} \cdot \frac{1 + \frac{B_\epsilon}{I}}{1 + \frac{B_\epsilon}{B_\tau}} = \Delta T \cdot \frac{1 + \frac{I}{B_\epsilon}}{1 + \frac{B_\tau}{B_\epsilon}}. \tag{20}$$

The rightmost factor is less than 1 since $I < B_\tau$, by assumption. Thus, eq. (20) makes it clear that whatever speedup (or slowdown) we achieve by exploiting the work-communication trade-off, the energy-efficiency can be slightly lower.

To get an upper-bound on ΔE, we need lower-bounds on f, $\frac{1}{m}$, or both. For f, we have two: $f > 1$ and $f > \frac{1}{m}\frac{B_\tau}{I}$. The latter is more general, though it will be pessimistic when $m \gg \frac{B_\tau}{I}$. With that caveat, we obtain

$$\Delta E \quad < \quad m \cdot \frac{1 + \frac{I}{B_\epsilon}}{1 + \frac{B_\tau}{B_\epsilon}}. \tag{21}$$

Equation (21) emphasizes the critical role that reducing communication plays in increasing energy-efficiency.

CASE 3: BASELINE AND NEW ALGORITHMS ARE COMPUTE-BOUND IN TIME If both the baseline and new algorithms are compute-bound in time, then we may deduce that $I > B_\tau$ and $\Delta T = \frac{1}{f} < 1$. That is, we should generally expect a slowdown because flops already dominate; increasing flops only increases work without the possibility of saving any time.

We can obtain a lower-bound on ΔE by invoking upper-bounds on $\frac{1}{m}$ or on f. Our choices are $\frac{1}{m} < 1$, by definition; and $\frac{1}{m} < f\frac{I}{B_\tau}$, which follows from the new algorithm being compute-bound. The tightest of these is first; thus,

$$\Delta E \quad > \quad \frac{1 + \frac{B_\epsilon}{I}}{f + \frac{B_\epsilon}{I}} \quad = \quad \Delta T \frac{1 + \frac{B_\epsilon}{I}}{1 + \frac{1}{f}\frac{B_\epsilon}{I}} \quad > \quad \Delta T. \tag{22}$$

Substituting ΔT for $\frac{1}{f}$ leads to the final equality and inequality. Equation (22) says that the greenup will be no worse than the speedup. However, in this case $\Delta T < 1$, which means that a loss in energy-efficiency is possible.

We can get an upper-bound on ΔE using eq. (16) with a lower-bound on f. The two choices are $f > 1$ and $f > \frac{1}{m}\frac{B_\tau}{I}$. Since the baseline was also compute-bound in time, meaning $\frac{B_\tau}{I} \leqslant 1$, the latter lower-bound

is strictly less than the trivial lower-bound. Thus, the tighter upper-bound is

$$\Delta E = \frac{1+\frac{B_\epsilon}{I}}{f+\frac{1}{m}\frac{B_\epsilon}{I}} \quad < \quad \frac{1+\frac{B_\epsilon}{I}}{1+\frac{1}{m}\frac{B_\epsilon}{I}} \quad < \quad 1+\frac{B_\epsilon}{I}, \tag{23}$$

where the last inequality shows the limit of the new algorithm having no communication ($m \to \infty$). We may conclude that even though in this case we will always slowdown in time, a greenup may be possible, albeit bounded by roughly the energy communication penalty.

6.4 *Summary of the three cases*

We summarize the 3 cases as follows.

CASE 1 Baseline and new algorithms are memory-bound in time:

$$\begin{gathered} I < B_\tau \\ 1 < \Delta T = m \\ \frac{1+\frac{I}{B_\epsilon}}{1+\frac{B_\tau}{B_\epsilon}} < \Delta E < \frac{1+\frac{B_\epsilon}{I}}{1+\frac{B_\epsilon}{B_\tau}}. \end{gathered}$$

CASE 2 Baseline is memory-bound in time while the new algorithm is compute-bound:

$$\begin{gathered} I < B_\tau \\ \Delta T = \frac{1}{f}\frac{B_\tau}{I} \\ \Delta T \cdot \frac{1+\frac{I}{B_\epsilon}}{1+\frac{B_\tau}{B_\epsilon}} < \Delta E < m \cdot \frac{1+\frac{I}{B_\epsilon}}{1+\frac{B_\tau}{B_\epsilon}}. \end{gathered}$$

CASE 3 Baseline and the new algorithm are compute-bound in time:

$$\begin{gathered} B_\tau < I \\ \frac{1}{f} = \Delta T < 1 \\ \Delta T \cdot \frac{1+\frac{B_\epsilon}{I}}{1+\frac{1}{f}\frac{B_\epsilon}{I}} < \Delta E < \frac{1+\frac{B_\epsilon}{I}}{1+\frac{1}{m}\frac{B_\epsilon}{I}}. \end{gathered}$$

ILLUSTRATION To illustrate the preceding bounds, consider fig. 8. There, we compute the speedups, greenups, and their lower and upper bounds, for a variety of intensity values $I \in [B_\tau/4, B_\epsilon]$. Figure 8 shows modeled speedups and greenups as points; the minimum and maximum bounds are shown as horizontal and vertical lines. Figure 8 clarifies how it is unlikely that one will improve time ($\Delta T > 1$) while incurring a loss in energy-efficiency ($\Delta E < 1$). However, there are many opportunities for the converse, particularly in either Case 2 or Case 3.

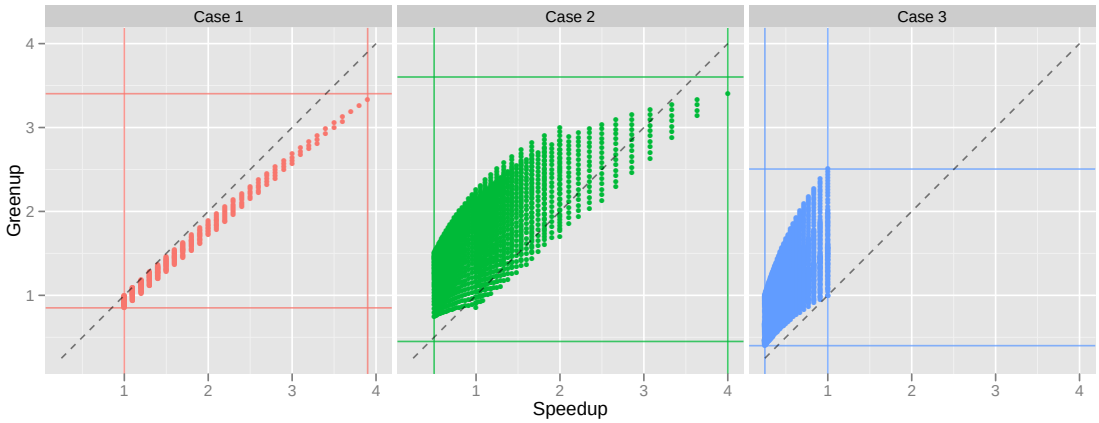


Figure 8: Illustration of the speedup and greenup bounds summarized in §6.4. Points correspond to $(\Delta T, \Delta E)$ pairs at particular values of I, f, and m; horizontal and vertical lines indicate the corresponding minimum lower bounds and maximum upper bounds on speedup and greenup, taken over all values of I, f, and m in each case.

7 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. [32].) Our model is perhaps most similar to a recent technical report by Demmel et al. [14]. 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 and there have 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 [8, 29, 38, 47]; 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 [1, 2, 34]. 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 us-

age through dynamic frequency and voltage 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 [18–20, 31, 43]. In contrast to our paper, the work on DVFS looks at a different time-energy trade-off that comes from the superlinear scaling of power and energy with frequency.

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) [43]. 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 is not intended for direct use by end-user programmers or algorithm designers.

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 [10, 15–17, 21, 37]. One notable example is the work of Dongarra et al., which observes the energy benefits of mixed-precision [15]. Our work tries to generalize beyond specific domains through the abstraction of intensity and balance.

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 [16]. They create abstract profiles to capture the differing characteristics of these applications. However, because of the diverse array of full applications they consider, they do not ascribe specific analytic properties of a computation in a way that programmers or algorithm designers can directly use to understand and re-shape 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 [21], 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 [41] and NVIDIA's Management Library for power measurement [40]. However, unlike Intel's tool, NVML only provides power consumption for the entire GPU, in-

cluding the memory. NVML provides readings in milliwatts and is accurate to within ± 5 watts.

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 [9, 12, 24–26, 35, 39, 49, 50, 52].

However, there are numerous other approaches. For instance, numerous recent studies have developed detailed GPU-specific models [5, 27, 51]; though these models are capable of directly predicting time, they require very detailed characterizations of the input program or intimate knowledge of the GPU microarchitecture. As such, it is nontrivial 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 [19, 28, 36, 43–45]. 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 its generalizations [7, 22], FLOP/s per Watt (i.e., flops per Joule) [42], and The Green Index [46].

8 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 have shown in § 6. Today, $B_\tau > B_\epsilon$, due largely to idle power and other microarchitectural inefficiencies; consequently, race-to-halt strategies will be the most reasonable first-order technique to save energy. Will this conclusion change significantly in the future?

Our study reinforces three relevant trends that suggest the balance gap will shift. First, we observed that the newer Kepler-class GPU, based on a 28 nm process technology, had a significantly lower constant power compared to the 40 nm Fermi-class GPU: 66 W for the Kepler-based GTX 680 vs. 122 W for the Fermi-based GTX 580. Secondly, GPUs in general exemplify simpler core microarchitecture design when compared to general-purpose CPU platforms. The balance gap did not appear possible on the CPU system in our study, even with no constant power; however, it did emerge on the GPU platforms in an hypothetical $\pi_0 = 0$ scenario. Thirdly, although the cost ratio between reading data from memory and computing on data is ex-

pected to remain constant [33], wire capacitance will not scale. Consequently, the cost of moving data will stay the same. Unless the distance between off-chip memory and the processing cores decreases significantly (e.g., via memory-die stacking), the balance gap will increase.

LIMITATIONS Our 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 done so in prior work [12] and plan to extend it for energy.

Secondly, we consider the best-case scenario of flops-centric (and on GPUs, FMA-centric) computation. In order for our model to estimate energy consumption more accurately, a more nuanced accounting of the instruction mix is necessary.

Thirdly, we ignored power caps, which can cause our analysis to overestimate power consumption and performance (§ 5). 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 our basic model an interesting starting point for identifying potential new directions lying at the intersection of algorithms and architecture.

Acknowledgements

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 Dan Bedard and Rob Fowler for providing a PowerMon 2 unit and technical support, and Hyesoon Kim for the PCIe interposer. This work was supported in part by the National Science Foundation (NSF) under NSF CAREER award number 0953100; the U.S. Dept. of Energy (DOE) under award DE-FC02-10ER26006/DE-SC0004915; 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.

A TIME AND ENERGY TRADE-OFFS UNDER CONSTANT POWER

This appendix generalizes the work-communication trade-off of § 6 to the case of $\pi_0 > 0$.

A.1 *Total energy,* $E_{f,m}$

Equation (5) expresses total energy $E(W, I)$ as a function of W and I. Therefore, we may write $E_{f,m}$ as

$$E_{f,m} = E(fW, fmI) \tag{24}$$

$$= fW\hat{\epsilon}_{\text{flop}} \cdot \left(1 + \frac{\hat{B}_\epsilon(fmI)}{fmI}\right) = W\hat{\epsilon}_{\text{flop}} \cdot \left(f + \frac{1}{m}\frac{\hat{B}_\epsilon(fmI)}{I}\right). \tag{25}$$

A.2 *Effective energy balance,* $\hat{B}_\epsilon(I)$

When analyzing work-communication trade-offs, we will make use of the following relationship between $\hat{B}_\epsilon(I)$ and $\hat{B}_\epsilon(fmI)$.

Lemma A.1. *Let* $f, m \geqslant 1$. *Then,* $\hat{B}_\epsilon(I) \geqslant \hat{B}_\epsilon(fmI)$.

Proof. This fact follows simply from the definition of $\hat{B}_\epsilon(I)$.

$$\begin{aligned}\hat{B}_\epsilon(I) &\equiv \eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \max(0, B_\tau - I) \\ &\geqslant \eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \max(0, B_\tau - fmI) \\ &= \hat{B}_\epsilon(fmI).\end{aligned}$$

□

A.3 *Greenup,* ΔE

By the definition of greenup, eq. (16),

$$\Delta E = \frac{1 + \frac{\hat{B}_\epsilon(I)}{I}}{f\left(1 + \frac{\hat{B}_\epsilon(fmI)}{fmI}\right)} = \frac{1 + \frac{\hat{B}_\epsilon(I)}{I}}{f + \frac{\hat{B}_\epsilon(fmI)}{mI}}. \tag{26}$$

A.4 *General necessary condition for greenup*

We derive a general necessary condition for an actual greenup, or $\Delta E > 1$. Let I be the intensity of the baseline algorithm. Suppose the new algorithm increases flops by $f > 1$ while reducing memory operations by a factor of $m > 1$. By eq. (26), the condition $\Delta E > 1$ implies

$$f < 1 + \frac{\hat{B}_\epsilon(I)}{I} - \frac{1}{m}\frac{\hat{B}_\epsilon(fmI)}{I} < 1 + \frac{\hat{B}_\epsilon(I)}{I}, \tag{27}$$

where the last inequality follows from lemma A.1. This upper-bound on f also holds in the limit that we eliminate communication entirely ($m \to \infty$). From here, there are two interesting cases to consider.

COMPUTE-BOUND LIMIT. Suppose the baseline algorithm is compute-bound, so that $I \geqslant B_\tau$. Then,

$$f < 1 + \frac{\hat{B}_\epsilon(I)}{I} = 1 + \eta_{\text{flop}} \frac{B_\epsilon}{I} \leqslant 1 + \eta_{\text{flop}} \frac{B_\epsilon}{B_\tau}. \tag{28}$$

In the limit that $\pi_0 \to 0$, $\eta_{\text{flop}} \to 1$ and the bound matches that of § 6.2.

MEMORY-BOUND LIMIT. Suppose instead that the baseline is memory-bound, so that $I < B_\tau$. Then,

$$\begin{aligned} f &< 1 + \frac{\hat{B}_\epsilon(I)}{I} \\ &= 1 + \eta_{\text{flop}} \frac{B_\epsilon}{I} + (1 - \eta_{\text{flop}}) \left(\frac{B_\tau}{I} - 1 \right) \\ &= \eta_{\text{flop}} + \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) B_\tau}{I}. \end{aligned} \tag{29}$$

That is, the upper-limit on the extra flop factor f is related to a weighted average of the time- and energy-balance constants.

A.5 ΔE *bounds for case 1: New algorithm is memory-bound in time*

LOWER BOUND Suppose $fmI < B_\tau$ and $\frac{1}{m} < 1$. Then,

$$\begin{aligned} \Delta E &= \frac{1 + \frac{\hat{B}_\epsilon(I)}{I}}{f + \frac{1}{m} \frac{\hat{B}_\epsilon(fmI)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot \max(0, B_\tau - I)}{I}}{f + \frac{1}{m} \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot \max(0, B_\tau - fmI)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - I)}{I}}{f + \frac{1}{m} \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - fmI)}{I}}. \end{aligned} \tag{30}$$

Applying $f < \frac{1}{m} \frac{B_\tau}{I}$ and $\frac{1}{m} < 1$ yields

$$\begin{aligned} \Delta E &> \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - I)}{I}}{\frac{B_\tau}{I} + \frac{\eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - B_\tau)}{I}} \\ &= \frac{\frac{I + \eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - I)}{I}}{\frac{B_\tau + \eta_{\text{flop}} B_\epsilon}{I}} \\ &= \frac{I + \eta_{\text{flop}} B_\epsilon + (1 - \eta_{\text{flop}}) \cdot (B_\tau - I)}{B_\tau + \eta_{\text{flop}} B_\epsilon} \\ &= \frac{B_\tau \cdot (1 - \eta_{\text{flop}}) + \eta_{\text{flop}} \cdot (B_\epsilon + I)}{B_\tau + \eta_{\text{flop}} B_\epsilon}. \end{aligned} \tag{31}$$

When $\pi_0 = 0$, we recover eq. (18). Otherwise, whether there is a greenup depends on η_{flop}, since $B_\tau \cdot (1 - \eta_{\text{flop}}) < B_\tau$ but $\eta_{\text{flop}} \cdot (B_\epsilon + I) > \eta_{\text{flop}} B_\epsilon$.

UPPER BOUND To derive an upper-bound on ΔE, consider the conditions $1 < m < \frac{B_\tau}{fI}$ or $f\frac{I}{B_\tau} < \frac{1}{m} < 1$ and $f > 1$. From these,

$$\Delta E = \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - fmI)}{I}}.$$

Applying the lower bounds on f, $\frac{1}{m}$, m, we get

$$\begin{aligned}\Delta E &< \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{1 + \frac{I}{B_\tau}\frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}, \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{B_\tau}}.\end{aligned}$$

Again, $\pi_0 = 0$ recovers eq. (19). Since $B_\tau > I$, the right hand side will always be greater than 1, meaning that a greenup is always possible. However, the balance gap and the intensity of the original algorithm will limit it.

A.6 ΔE *bounds for case 2: baseline is memory-bound but new algorithm is compute-bound in time*

LOWER BOUND To derive a lower bound, consider the conditions $I < B_\tau$, $fmI > B_\tau$, and $\Delta T = \frac{1}{f}\frac{B_\tau}{I}$. From these,

$$\begin{aligned}\Delta E &= \frac{1 + \frac{\hat{B}_\epsilon(I)}{I}}{f + \frac{1}{m}\frac{\hat{B}_\epsilon(fmI)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot\max(0, B_\tau - I)}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot\max(0, B_\tau - fmI)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(0)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}}.\end{aligned}$$

Applying the upper bound $\frac{1}{m} < f\frac{I}{B_\tau}$ yields,

$$\begin{aligned}
\Delta E &> \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + f\frac{I}{B_\tau}\frac{\eta_{\text{flop}} B_\epsilon}{I}} \\
&= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + f\frac{\eta_{\text{flop}} B_\epsilon}{B_\tau}} \\
&= \frac{1}{f}\cdot\frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{1 + \frac{\eta_{\text{flop}} B_\epsilon}{B_\tau}} \\
&= \Delta T \cdot \frac{I + \eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}\frac{B_\tau}{B_\tau + \eta_{\text{flop}} B_\epsilon}\frac{I}{B_\tau} \\
&= \Delta T \cdot \frac{I + \eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{B_\tau + \eta_{\text{flop}} B_\epsilon} \\
&= \Delta T \cdot \frac{\eta_{\text{flop}} B_\epsilon + B_\tau - \eta_{\text{flop}} B_\tau + \eta_{\text{flop}} I}{B_\tau + \eta_{\text{flop}} B_\epsilon} \\
&= \Delta T \cdot \frac{B_\tau (1-\eta_{\text{flop}}) + \eta_{\text{flop}} (B_\epsilon + I)}{B_\tau + \eta_{\text{flop}} B_\epsilon}.
\end{aligned}$$

If $\pi_0 = 0$, this equation yields eq. (20).

Since $B_\tau (1-\eta_{\text{flop}}) < B_\tau$ and $\eta_{\text{flop}} (B_\epsilon + I) > \eta_{\text{flop}} B_\epsilon$, we cannot determine if this lower-bound on ΔE will be greater or less than the speedup.

UPPER BOUND To derive an upper-bound, consider the conditions $I < B_\tau$ and $fmI > B_\tau$. From these,

$$\Delta E = \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}}.$$

Applying the lower bound $f > \frac{1}{m}\frac{B_\tau}{I}$,

$$\begin{aligned}
\Delta E &< \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{\frac{1}{m}\frac{B_\tau}{I} + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}}, \\
&= m \cdot \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{\frac{B_\tau}{I} + \frac{\eta_{\text{flop}} B_\epsilon}{I}}, \\
&= m \cdot \frac{\frac{I + \eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{I}}{\frac{B_\tau + \eta_{\text{flop}} B_\epsilon}{I}}, \\
&= m \cdot \frac{I + \eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot(B_\tau - I)}{B_\tau + \eta_{\text{flop}} B_\epsilon}, \\
&= m \cdot \frac{B_\tau - \eta_{\text{flop}} B_\tau + \eta_{\text{flop}} B_\epsilon + \eta_{\text{flop}} I}{B_\tau + \eta_{\text{flop}} B_\epsilon}, \\
&= m \cdot \frac{B_\tau (1-\eta_{\text{flop}}) + \eta_{\text{flop}} (B_\epsilon + I)}{B_\tau + \eta_{\text{flop}} B_\epsilon}.
\end{aligned}$$

When $\pi_0 = 0$, this inequality recovers eq. (21). As with the lower bound, the equation cannot tell us if the right hand side will be greater than or less than m, but does stresses its importance.

A.7 ΔE *bounds for case 3: baseline is compute-bound in time*

LOWER BOUND To derive a lower bound on ΔE, consider the conditions $I > B_\tau$, $fmI > B_\tau$, $\frac{1}{m} < 1$, and $\Delta T = \frac{1}{f}$. From these,

$$\begin{aligned}\Delta E &= \frac{1 + \frac{\hat{B}_\epsilon(I)}{I}}{f + \frac{1}{m}\frac{\hat{B}_\epsilon(fmI)}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot 0}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon + (1-\eta_{\text{flop}})\cdot 0}{I}} \\ &= \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}}.\end{aligned}$$

Applying the tighter upper bound of $\frac{1}{m} < 1$ (rather than $fmI > B_\tau$) yields,

$$\begin{aligned}\Delta E &> \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{f + \frac{\eta_{\text{flop}} B_\epsilon}{I}} \\ &= \frac{1}{f} \cdot \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{1 + \frac{1}{f}\frac{\eta_{\text{flop}} B_\epsilon}{I}} \\ &= \Delta T \cdot \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{1 + \frac{1}{f}\frac{\eta_{\text{flop}} B_\epsilon}{I}} \\ &> \Delta T.\end{aligned}$$

In this case, greenup will be at least as good as the speedup, at least in theory. In reality, performance will tend to decrease so that a loss in energy-efficiency is also likely.

UPPER BOUND To derive an upper bound on ΔE, consider the conditions $I > B_\tau$, $fmI > B_\tau$, $\frac{1}{m} < 1$, and $\Delta T = \frac{1}{f}$. From these,

$$\Delta E = \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{f + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}}.$$

Using the trivial lower-bound $f > 1$ (rather than $f > \frac{1}{m}\frac{B_\tau}{I}$),

$$\Delta E < \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{1 + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}},$$

In the limit of eliminating communication, ($m \to \infty$),

$$\Delta E < \frac{1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}}{1 + \frac{1}{m}\frac{\eta_{\text{flop}} B_\epsilon}{I}} < 1 + \frac{\eta_{\text{flop}} B_\epsilon}{I}.$$

Recall that the condition $\Delta T = \frac{1}{f}$ implies there will always be a slowdown in time. However, eq. (32) implies that a greenup may nevertheless be possible. The upper limit on any such greenup will be roughly the energy communication penalty.

A.8 *Summary*

Relative to § 6.2, the fact of constant power ($\pi_0 > 0$) removes any guarantee on whether we will see a greenup. However, greenups are nevertheless possible, albeit at a reduced amount related to the efficiency factor η_{flop}. It then becomes critical to understand the architectural and hardware trends most likely to effect π_0 (and therefore also η_{flop}).

REFERENCES

[1] Ismail Assayad, Alain Girault, and Hamoudi Kalla. Tradeoff exploration between reliability, power consumption, and execution time. In Francesco Flammini, Sandro Bologna, and Valeria Vittorini, editors, *Proceedings of the 30th International Conference on Computer Safety, Security, and Reliability (SAFECOMP)*, volume LNCS 6894, pages 437–451, Naples, Italy, 2011. Springer. URL http://link.springer.com/book/10.1007/978-3-642-24270-0/page/1. (Cited on page 29.)

[2] Guillaume Aupy, Anne Benoit, and Yves Robert. Energy-aware scheduling under reliability and makespan constraints. Technical Report October, INRIA, Grenoble, France, 2011. URL http://graal.ens-lyon.fr/~abenoit/papers/RR-7757.pdf. (Cited on page 29.)

[3] T. Austin, D. Blaauw, T. Mudge, K. Flautner, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, December 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1250885. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1250885. (Cited on page 10.)

[4] Muhammad Ali Awan and Stefan 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*, pages 92–101. IEEE, July 2011. ISBN 978-1-4577-0643-1. doi: 10.1109/ECRTS.2011.17. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6001772. (Cited on pages 2, 10, and 20.)

[5] Sara S. Baghsorkhi, Matthieu Delahaye, Sanjay J. Patel, William D. Gropp, and Wen mei W. Hwu. An adaptive performance modeling tool for GPU architectures. *SIGPLAN Not.*, 45(5):105–114, January 2010. ISSN 0362-1340. doi: 10.1145/1837853.1693470. URL http://doi.acm.org/10.1145/1837853.1693470. (Cited on page 31.)

[6] Daniel Bedard, Min Yeol Lim, Robert Fowler, and Allan Porterfield. PowerMon 2: Fine-grained, integrated measurement. Technical report, RENaissance Computing Institute, University of North Caroliina, Chapel Hill, NC, USA, 2009. URL http://www.renci.org/technical-reports/tr-09-04. (Cited on pages iv, 11, and 13.)

[7] Costas Bekas and Alessandro Curioni. A new energy-aware performance metric. In *Proceedings of the International Conference on*

Energy-Aware High-Performance Computing (EnA-HPC), Hamburg, Germany, September 2010. URL http://www.ena-hpc.org. (Cited on page 31.)

[8] Brad D. Bingham and Mark R. Greenstreet. Computation with energy-time trade-offs: Models, algorithms and lower-bounds. *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 143–152, December 2008. doi: 10.1109/ISPA.2008.127. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4725144. (Cited on page 29.)

[9] Guy E. Blelloch, Bruce M. Maggs, and Gary L. Miller. The hidden cost of low bandwidth communication. In Uzi Vishkin, editor, *Developing a Computer Science Agenda for High-Performance Computing*, pages 22–25. ACM, New York, NY, USA, 1994. doi: 10.1145/197912.197923. (Cited on pages 1, 7, and 31.)

[10] George Bosilca, Hatem Ltaief, and Jack Dongarra. Power profiling of Cholesky and QR factorizations on distributed memory systems. *Computer Science - Research and Development*, pages 1–9, 2012. ISSN 1865-2034. doi: 10.1007/s00450-012-0224-2. URL http://dx.doi.org/10.1007/s00450-012-0224-2. (Cited on page 30.)

[11] Aparna Chandramowlishwaran, Jee Whan Choi, Kamesh Madduri, and Richard 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. doi: 10.1145/2312005.2312039. Brief announcement. (Cited on page 22.)

[12] Kenneth Czechowski, Casey Battaglino, Chris McClanahan, Aparna Chandramowlishwaran, and Richard Vuduc. Balance principles for algorithm-architecture co-design. In *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar)*, Berkeley, CA, USA, May 2011. URL http://www.usenix.org/events/hotpar11/tech/final_files/Czechowski.pdf. (Cited on pages 1, 7, 31, and 32.)

[13] Kenneth Czechowski, Chris McClanahan, Casey Battaglino, Kartik Iyer, P.-K. Yeung, and Richard 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. doi: 10.1145/2304576.2304604. (Cited on page 1.)

[14] James Demmel, Andrew Gearhart, Oded Schwartz, and Benjamin Lipschitz. Perfect strong scaling using no additional energy. Technical report, University of California, Berkeley,

CA, USA, 2012. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-126.html. (Cited on page 29.)

[15] Jack Dongarra, Hatem Ltaief, Piotr Luszczek, and Vince 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. (Cited on page 30.)

[16] Hadi Esmaeilzadeh, Ting Cao, Xi Yang, Stephen M. Blackburn, and Kathryn S. McKinley. Looking back and looking forward: power, performance, and upheaval. *Commun. ACM*, 55(7):105–114, July 2012. ISSN 0001-0782. doi: 10.1145/2209249.2209272. URL http://doi.acm.org/10.1145/2209249.2209272. (Cited on page 30.)

[17] Xizhou Feng, Rong Ge, and K.W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, page 34, april 2005. doi: 10.1109/IPDPS.2005.346. (Cited on page 30.)

[18] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry L. Rountree, and Mark E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):835–848, June 2007. ISSN 1045-9219. doi: 10.1109/TPDS.2007.1026. URL http://dx.doi.org/10.1109/TPDS.2007.1026. (Cited on page 30.)

[19] Rong Ge and Kirk Cameron. Power-aware speedup. In *In Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 07)*, 2007. (Cited on page 31.)

[20] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 1-59593-061-2. doi: 10.1109/SC.2005.57. URL http://dx.doi.org/10.1109/SC.2005.57. (Cited on page 30.)

[21] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 21(5):658–671, May 2010. ISSN 1045-9219. doi: 10.1109/TPDS.2009.76. (Cited on page 30.)

[22] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE J. Solid-State Circuits*, 31(9):1277–1284, sep 1996. ISSN 0018-9200. doi: 10.1109/4.535411. (Cited on page 31.)

[23] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, December 1987. ISSN 00219991. doi: 10.1016/0021-9991(87)90140-9. URL http://linkinghub.elsevier.com/retrieve/pii/0021999187901409. (Cited on page 21.)

[24] Mark D Hill and Michael R Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38, July 2008. ISSN 0018-9162. doi: 10.1109/MC.2008.209. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4563876. (Cited on pages 1 and 31.)

[25] W. Daniel Hillis. Balancing a Design. *IEEE Spectrum*, 1987. URL http://longnow.org/essays/balancing-design/. (Cited on pages 1 and 7.)

[26] Roger W. Hockney and Ian J. Curington. f1/2: A parameter to characterize memory and communication bottlenecks. *Parallel Computing*, 10(3):277–286, May 1989. ISSN 01678191. doi: 10.1016/0167-8191(89)90100-2. URL http://linkinghub.elsevier.com/retrieve/pii/0167819189901002. (Cited on pages 1, 7, 8, and 31.)

[27] Sunpyo Hong and Hyesoon 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)*, pages 152–163, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-526-0. doi: 10.1145/1555754.1555775. URL http://doi.acm.org/10.1145/1555754.1555775. (Cited on page 31.)

[28] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. *SIGARCH Comput. Archit. News*, 38(3):280–289, June 2010. ISSN 0163-5964. doi: 10.1145/1816038.1815998. URL http://doi.acm.org/10.1145/1816038.1815998. (Cited on page 31.)

[29] R. Jain, D. Molnar, and Z. Ramzan. Towards a model of energy complexity for algorithms. In *IEEE Wireless Communications and Networking Conference, 2005*, pages 1884–1890. IEEE, 2005. ISBN 0-7803-8966-2. doi: 10.1109/WCNC.2005.1424799. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1424799. (Cited on page 29.)

[30] Hong 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*, pages 326–333, New York, New York, USA, May 1981. ACM Press. doi: 10.1145/800076.802486. URL http://portal.acm.org/citation.cfm?doid=800076.802486. (Cited on page 5.)

[31] 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)*, pages 221 –228, dec. 2010. doi: 10.1109/GreenCom-CPSCom.2010.143. (Cited on page 30.)

[32] Stefanos Kaxiras and Margaret Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008. ISBN 1598292080, 9781598292084. (Cited on page 29.)

[33] Stephen W. Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, September 2011. ISSN 0272-1732. doi: 10.1109/MM.2011.89. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6045685. (Cited on pages 8, 9, 19, and 32.)

[34] Vijay Anand Korthikanti and Gul Agha. Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures. In *2009 International Conference on Parallel Processing*, pages 212–219, Vienna, Austria, September 2009. IEEE. ISBN 978-1-4244-4961-3. doi: 10.1109/ICPP.2009.74. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5362310. (Cited on page 29.)

[35] H T Kung. Memory requirements for balanced computer architectures. In *Proceedings of the ACM Int'l. Symp. Computer Architecture (ISCA)*, Tokyo, Japan, 1986. doi: http://doi.acm.org/10.1145/17407.17362. (Cited on pages 1, 7, and 31.)

[36] Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang, Chun-Yi Su, and Kirk Cameron. Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems. *Computer Science - Research and Development*, pages 1–9, 2011. ISSN 1865-2034. URL http://dx.doi.org/10.1007/s00450-011-0190-0. 10.1007/s00450-011-0190-0. (Cited on page 31.)

[37] Hatem Ltaief, Piotr Luszczek, and Jack Dongarra. Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency. *Computer Science - Research and Development*, pages 1–11. ISSN 1865-2034. URL http://dx.doi.org/10.1007/s00450-011-0191-z. 10.1007/s00450-011-0191-z. (Cited on page 30.)

[38] A.J. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 77(2-4):181–187, February 2001. ISSN 00200190. doi: 10.1016/S0020-0190(00)00214-3. URL http://linkinghub.elsevier.com/retrieve/pii/S0020019000002143. (Cited on page 29.)

[39] John McCalpin. Memory Bandwidth and Machine Balance in High Performance Computers. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter*, December 1995. URL http://www.cs.virginia.edu/~mccalpin/papers/balance/index.html. (Cited on pages 1, 7, and 31.)

[40] NVIDIA. NVML API Reference Manual, 2012. (Cited on pages 11 and 30.)

[41] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27, March-April 2012. ISSN 0272-1732. doi: 10.1109/MM.2012.12. (Cited on page 30.)

[42] S. Sharma, Chung-Hsing Hsu, and Wu-Chun Feng. Making a case for a Green500 list. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006. doi: 10.1109/IPDPS.2006.1639600. (Cited on page 31.)

[43] Shuaiwen Song, Matthew Grove, and Kirk W. Cameron. An iso-energy-efficient approach to scalable system power-performance optimization. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 262–271, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4516-5. doi: 10.1109/CLUSTER.2011.37. URL http://dx.doi.org/10.1109/CLUSTER.2011.37. (Cited on pages 30 and 31.)

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

[45] Balaji Subramaniam and Wu-Chun 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, GREENCOM-CPSCOM '10, pages 139–146, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4331-4. doi: 10.1109/GreenCom-CPSCom.2010.138. URL http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.138. (Cited on page 31.)

[46] Balaji Subramaniam and Wu-Chun 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. (Cited on page 31.)

[47] Akhilesh Tyagi. Energy-Time Trade-offs in VLSI Computations. In *Foundations of Software Technology and Theoretical Computer Science*, volume LNCS 405, pages 301–311, 1989. doi: 10.1007/3-540-52048-1_52. URL http://www.springerlink.com/content/781615m3w5627p1v/. (Cited on page 29.)

[48] Richard Vuduc and Kenneth Czechowski. What GPU computing means for high-end systems. *IEEE Micro*, 31(4):74–78, July/August 2011. doi: 10.1109/MM.2011.78. (Cited on pages 1 and 7.)

[49] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65, April 2009. ISSN 00010782. doi: 10.1145/1498765.1498785. URL http://portal.acm.org/citation.cfm?doid=1498765.1498785. (Cited on pages 1, 7, 8, and 31.)

[50] Dong Hyuk Woo and Hsien-Hsin S Lee. Extending Amdahl's Law for energy-efficient computing in the many-core era. *IEEE Computer*, 41(12):24–31, December 2008. doi: 10.1109/MC.2008.530. (Cited on pages 1 and 31.)

[51] Yao Zhang and John D. Owens. A quantitative performance analysis model for gpu architectures. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 382–393, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4244-9432-3. URL http://dl.acm.org/citation.cfm?id=2014698.2014875. (Cited on page 31.)

[52] Tsahee Zidenberg, Isaac Keslassy, and Uri Weiser. Multi-Amdahl: How Should I Divide My Heterogeneous Chip? *IEEE Computer Architecture Letters*, pages 1–4, 2012. ISSN 1556-6056. doi: 10.1109/L-CA.2012.3. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6175876. (Cited on pages 1 and 31.)