How Much (Execution) Time and Energy Does My Algorithm Cost?

Do we need to design algorithms differently if our goal is to save energy, rather than time or space? This article presents a simple and speculative thought experiment that suggests when and why the answer could be "yes."

By Jee Whan Choi and Richard W. Vuduc

DOI: 10.1145/2425676.2425691

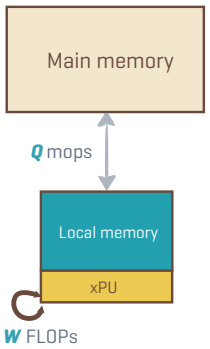
In 1985 at MIT, a brilliant young iconoclast, named Danny Hillis,[1] concluded his doctoral dissertation with a provocative claim: The design of computer algorithms had gone too far in abstracting away the physical realities of a machine; if not corrected, computer science theory would become irrelevant to designing new architectures [1].

This accusation raises a natural question. What would it mean to design an algorithm or write code in a way that incorporates physical costs? Although we learn in a typical computer science class to minimize abstract measures related to time and space, the most interesting and tangible cost on a modern computing platform is a very physical one, namely, energy [2]. When a cell phone battery runs out, we have used too much energy; when the electricity bill is high, we have used too much energy. What might an energy cost model for algorithms look like, and could it tell us anything about how to design new machines and new algorithms?

Neither of us, the authors, claims to know the answer. But perhaps we can glean some insight through a thought experiment that tries to relate time (e.g., seconds), energy (e.g., Joules), and power (e.g., Watts = Joules/second). Let's start with an abstract model of an algorithm running on a machine and translate the model into execution time (see Figure 1); we then ask what an energy model might look like and try to relate the two. (For details, see our technical report [3].)

Suppose an algorithm executes W floating-point operations (or "flops" for short) and moves Q bytes of data between the processor and memory. A natural cost model for time is to say the machine can perform a flop in t_f units of time and can move a byte in time t_m. In the best case, we may overlap flops and memory operations,

Figure 1. An algorithm running on this abstract von Neumann architecture must explicitly move data between main and local memory, and perform operations only on data in local memory.



1 By titling his concluding chapter "New Computer Architectures and Their Relationship to Physics, or, Why Computer Science is No Good," W. D. Hillis' 1985 dissertation boldly proclaimed the irrelevance of computer science theory to the design of practical machines.

such that the running time is bounded from below by:

$$T \geq \max(Wt_f, Qt_m) = Wt_f \max(1, \frac{B_T}{I}),$$

where Wt_f is the minimum time to execute just the flops, $B_T \equiv \frac{t_m}{t_f}$ is the time-balance of the machine, $I \equiv \frac{W}{Q}$ is the intensity of the algorithm [4].

To minimize time, we seek high-intensity algorithms (large I) and/or machines such that $B_T < I$. These desiderata express balance principles of algorithm design [5].

This simple model captures key first-order performance behavior, as the top subplot of Figure 2 suggests for a modern consumer-grade GPU. We show performance, or $\frac{W}{T}$ normalized to the GPU's peak flops per second (FLOP/s), as we vary I. If W is fixed, then the plot is effectively inverse time such that higher is better. The solid "roofline" [6] is the model. The points are actual measurements for a microbenchmark that streams a large array while performing I independent flops per byte. The dashed vertical line indicates the time-balance point B_T, which marks the intensity at which an algorithm transitions from being "memory-bound" in time (left of B_T) to being "compute-bound" (right of B_T).

A reasonable first hypothesis is that an energy cost model will look similar. However, we cannot overlap energy the way we can with time; we must therefore always pay the sum of energy for flops and communication. In addition, today's physical systems expend energy even when no operations are performed; we may posit a constant energy cost that is, say, linearly proportional to the total running time T. Thus, total energy becomes $E = We_f + Qe_m + p_{const}T$, where e_f is the energy cost (Joules) per flops and e_m the cost to move each byte; and p_{const} is the constant energy per unit time, which we will call "constant power." To reveal the relationship between E and T more directly, rewrite E as:

$$E = We_f(1 + (\frac{\tilde{B}_E(I)}{I}),$$

where $\tilde{e}_f = e_f + p_{const}\, t_f e_f = e_f + p_{idle}\, t_f$ is the effective energy per flops including idle energy, and $\tilde{B}_E(I)$ suitably defined is the effective energy balance.

This energy equation is easy to interpret by direct analogy to time: The product We_f is the minimum energy (including constant energy) needed to perform the flops; and $\frac{\tilde{B}_E(I)}{I}$ is the energy penalty for data movement. When $\tilde{B}_E(I) < I$, an algorithm expends less energy moving data than performing flops; this scenario implies a notion of being compute-bound versus memory-bound with respect to energy, and furthermore yields a balance principle for energy. Most importantly, this model captures real

Figure 2. Examples of time and energy cost models for algorithms. The model appears as a solid line; experimental data for double-precision flops as markers. Today, time-balance dominates energy-balance (1.0 > 0.79 and 2.1 > 1.1 double-precision flops per byte). For algorithms and software, this may explain why race-to-halt saves energy. In the future, this situation may change—if cores become leaner and constant energy goes to zero, energy-balance could instead come to dominate time-balance. The energy-balance of 2.4 flops per byte when, hypothetically, constant power is zero, actually exceeds the time-balance of 1.0 flops per byte.

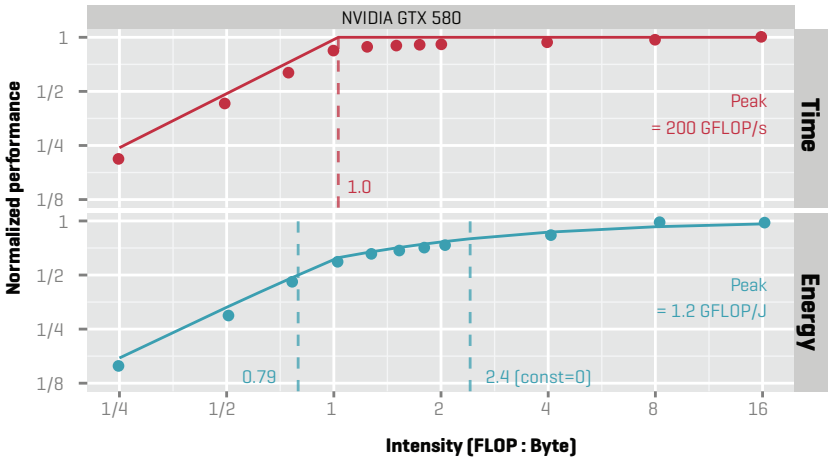
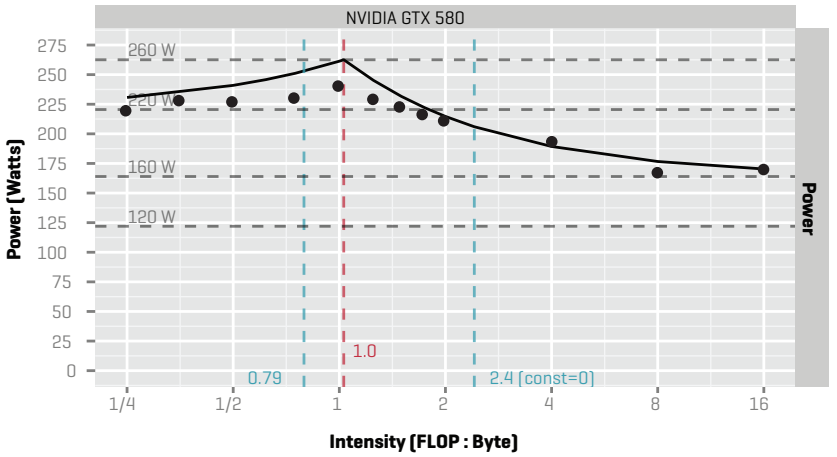


Figure 3. Modeled power (solid lines) appears against measurement (markers) for the system of Figure 2. Distinct intensity regimes show where a program might use lower or higher power.



behavior: The bottom two subplots of Figure 2 show that this simple energy model (solid line) can match measured energy data (markers).

Even simple models like this one have surprising ramifications both for algorithms and architectures. For example, consider the balance gap between time-balance B_T and energy-balance $\tilde{B}_E = \tilde{B}_E(I)$ at the value of I where energy for flops and data are equal. When these values are equal, time-efficiency and energy-efficiency are essentially the same, implying that so-called "race-to-halt" strategies for saving energy—by running as fast as possible and then shutting down—will work well [7]. Suppose instead that $B_T < \tilde{B}_E$. Then, energy-efficiency would tend to imply time-efficiency but the converse would not hold. Designers of algorithms and software might then choose to focus on optimizing energy rather than time. While the current literature suggests $B_T < \tilde{B}_E$ should hold [8], Figure 2 suggests the opposite on today's systems, $\tilde{B}_E < B_T$, appears to be true instead. The model explains why: If we hypothetically set $p_{\text{const}} = 0$, then energy-balance would exceed time-balance! (Compare the hypothetical 2.4 flops per byte to the actual 1.0 flops per byte). That is, with energy-efficient cores, constant energy and constant power mask any balance-gap effect that would cause energy-efficiency to dominate time-efficiency. This raises a hardware and architectural question, which is to identify or change the trajectory of evolution for B_T and $\tilde{B}_E$.

Let's close by considering two more examples. One example concerns systems' physics and the other involves algorithms.

Regarding system physics, consider that time and energy models make it possible to reason directly about algorithms and power. Figure 3 shows average instantaneous power for the preceding model, $\frac{E}{T}$, compared to measurements, (observed E)/(observed T). There are distinct intensity regimes in which an algorithm will require relatively more or less power. For instance, memory-bound algorithms ($I < B_T$) need more power up to the time-balance point, after which power decreases to its "ideal" value, namely, when the algorithm is dominated by flops. One question is whether direct knowledge of these regimes might enable smarter power throttling.

Although we learn in a typical computer science class to minimize abstract measures related to time and space, the most interesting and tangible cost on a modern computing platform is a very physical one, namely, energy.

Regarding algorithms, we believe families of algorithms exhibiting work-communication trade-offs will be among the most interesting to consider. That is, suppose we start with the same baseline algorithm that executes W flops and moves Q bytes. Next, suppose we can decrease communication by a factor of $m > 1$ to $\frac{Q}{m}$ at the cost of increasing flops by a factor $f > 1$ to Wf. One can derive general conditions on f and m under which we can hope for a speedup (decrease in time) and/or a "greenup" (decrease in energy) relative to the baseline. For instance, when $p_{\text{const}} = 0$ a greenup will occur if $f < 1 + \frac{m-1}{m}\frac{\tilde{B}_E}{I}$. If the baseline algorithm was already compute-bound in time (i.e., $I \geq B_T$) and $m \to \infty$ (no communication), this condition becomes $f < 1 + \frac{\tilde{B}_E}{B_T}$; meaning a work-communication trade-off will save energy only if the extra work is bounded by roughly the balance gap.

CONCLUSION

The time, energy, and power modeling exercise in this article is just one example of what may be possible in connecting algorithm attributes directly to machine parameters through a cost model that reflects the realities of physical machines. This exercise ignores numerous details of the memory hierarchy, communication costs (latency versus bandwidth), scheduling of computation, and other aspects of data movement such as internode communication. There are a number of promising approaches [9, 10] with broad implications for the design of algorithms for single processor systems, clusters, and data centers. We believe now is an excellent time to revisit algorithms and architecture research and ask whether a notion of physically constrained co-design might be fruitful.

References

[1] Hilis, W. D. *The Connection Machine.* MIT Press, Cambridge, 1989.

[2] Esmaeilzadeh, H., Cao, T., Yang, X., Blackburn, S.M., and McKinley, K.S. Looking back and looking forward: Power, performance, and upheaval. *Communications of the ACM* 55, 7 (2012), 105–114.

[3] Choi, J.W. and Vuduc, R.W. *A roofline model of energy.* Technical Report GT-CSE-2012-01, Georgia Institute of Technology, Atlanta, GA, USA, 2012.

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

[5] Czechowski, K., Battaglino, C., Mcclanahan, C., Chandramowlishwaran, A., and Vuduc, R. Balance principles for algorithm-architecture co-design. *USENIX Wkshp. Hot Topics in Parallelism (HotPar), Usenix Association* (2011), 1–5.

[6] Williams, S., Waterman, A., and Patterson, D. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM* 52, 4 (2009), 65.

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

[8] Keckler, S.W., Dally, W.J., Khailany, B., Garland, M., and Glasco, D. GPUs and the Future of Parallel Computing. *IEEE Micro* 31, 5 (2011), 7–17.

[9] Bingham, B.D. and Greenstreet, M.R. Computation with energy-time trade-offs: Models, algorithms and lower-bounds. *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, (2008), 143–152.

[10] Demmel, J., Gearhart, A., Schwartz, O., and Lipschitz, B. *Perfect strong scaling using no additional energy.* Technical Report No. UCB/EECS-2012-126, University of California, Berkeley, CA, USA, 2012.

Biographies

Jee Whan Choi is a fifth year Ph.D. student in the School of Electrical and Computer Engineering at Georgia Institute of Technology. His research interests include modeling for performance and power for multi-core, accelerators and heterogneous systems. Jee received his B.S. and M.S. from Georgia Institute of Technology.

Richard (Rich) Vuduc is an assistant professor in the School of Computational Science and Engineering at the Georgia Institute of Technology. His research lab, The HPC Garage (visit hpcgarage.org), is interested in high-performance computing, with an emphasis on parallel algorithms, performance analysis, and performance tuning.