Balance principles for algorithm-architecture co-design

Kent Czechowski[*], *Casey Battaglino*[†], *Chris McClanahan*[*],
Aparna Chandramowlishwaran[†], *Richard Vuduc*[†]
Georgia Institute of Technology
[†] *School of Computational Science and Engineering*
[*] *School of Computer Science*
{*kentcz,cbattaglino3,chris.mcclanahan,aparna,richie*}*@gatech.edu*

Abstract

We consider the problem of "co-design," by which we mean the problem of how to design computational algorithms for particular hardware architectures and vice-versa. Our position is that *balance principles* should drive the co-design process. A balance principle is a theoretical constraint equation that explicitly relates algorithm parameters to hardware parameters according to some figure of merit, such as speed, power, or cost. This notion originates in the work of Kung (1986); Callahan, Cocke, and Kennedy (1988); and McCalpin (1995); however, we reinterpret these classical notions of balance in a modern context of parallel and I/O-efficient algorithm design as well as trends in emerging architectures. From such a principle, we argue that one can better understand algorithm and hardware trends, and furthermore gain insight into how to improve both algorithms and hardware. For example, we suggest that although matrix multiply is currently compute-bound, it will in fact become memory-bound in as few as ten years—even if last-level caches grow at their current rates. Our overall aim is to suggest how to co-design rigorously and quantitatively while still yielding intuition and insight.

1 Our Position and its Limitations

We seek a formal framework that can explicitly relate characteristics of an algorithm, such as its inherent parallelism or memory behavior, with parameters of an architecture, such as the number of cores or cache sizes or memory latency. We refer to this task as one of algorithm-architecture *co-design*. Our goal is to say *precisely* and *analytically* how changes to the architecture might affect the scaling of a computation, and, conversely, identify what classes of computation might execute efficiently on a given architecture.

Our approach is inspired by the fundamental principle of *machine balance*, applied here in the context of modern architectures and algorithms. "Modern architectures" could include multicore CPUs or manycore GPU-like processors; and by modern algorithms, we mean those analyzed using recent techniques that explicitly count parallel operations and I/O operations. Given an algorithm analyzed this way and a cost model for some architecture, we can estimate compute-time and I/O-time explicitly. We then connect the algorithm and architecture simply by applying the concept of balance, which says a computation running on some machine is efficient if the compute-time dominates the I/O (memory transfer) time, as suggested originally by Kung (1986) and since applied by numerous others [13, 28, 32, 38].[1] The result is a constraint equation that binds algorithm and architecture parameters together; we refer to this constraint as a *balance principle*.

Position. Our position is that this simple and classical idea of balance is an insightful way to understand the impact of architectural design on algorithms and vice-versa. We argue this point by giving a detailed example of how one might derive a balance principle for an algorithm running on a manycore architecture, and analyze the resulting balance principle to determine the trajectory of algorithms given current architectural trends. Among numerous observations,

- we show that balance principles can yield unified ways of viewing several classical performance engineering principles, such as classical machine balance, Amdahl's Law, and Little's Law;
- we predict that matrix multiply, the prototypical compute-bound kernel, will actually become memory-bound in as few as ten years based on current architectural trends, *even* if last-level caches continue to grow at their historical rates;
- we argue that stacked memory, widely believed to be capable of largely solving the problem of how

[1] We depart slightly from traditional conventions of balance, which ask instead that compute-time equal I/O-time.

to scale bandwidth with compute capacity, still may not solve the problem of compute-bound kernels becoming increasingly memory-bound;

- we suggest how minimizing I/O-time relative to computation time will also save power and energy, thereby obviating the need for "new" principles of energy- or power-aware algorithm design.

Limitations. Our approach has several weaknesses.

First, our discussion is theoretical. That is, we assume abstract models for both the algorithms and the architectures and make numerous simplifying assumptions. However, as an initial step in formalizing intuition about how algorithms and architectures need to change, we believe that an abstract analytical model is a useful start.

Secondly, the specific balance principles we discuss in this paper still assume "time" as the main figure of merit, rather than, say, dollar cost. Nevertheless, we emphasize that time-based balance principles are just one example; one is free to redefine the figure of merit appropriately to consider balanced systems in other measures. We discuss energy based measures specifically in § 3.

Thirdly, what we suggest is a kind of re-packaging of many known principles, rather than discovering an entirely new principle for co-design. Still, we hope our synthesis proves useful in related efforts to design better algorithms and hardware through analytical techniques.

2 Deriving a Balance Principle

Deriving a balance principle consists of three steps.

1. Algorithmically analyze the parallelism.
2. Algorithmically analyze the I/O behavior, or number of transfers through the memory hierarchy.
3. Combine these two analyses with a cost model for an abstract machine, to estimate the total cost of computation vs. the cost of memory accesses.

From the last step, we may then impose the condition for balance, i.e., that compute-time dominate memory-time.

This process resembles Kung's approach [28], except that we (a) consider parallelism explicitly; (b) use richer algorithmic models that account for more detailed memory hierarchy costs; and (c) apply the model to analyze platforms of interest today, such as GPUs. Note that we use a number of parameters in our model; refer to Figure 1 and Table 1 as a guide.

Analyzing parallelism. To analyze algorithmic parallelism, we adopt the classical work-depth (work-span) model [9], which has numerous implementations in real programming models [11, 18, 24]. In this model, we represent a computation by a directed acyclic graph (DAG)

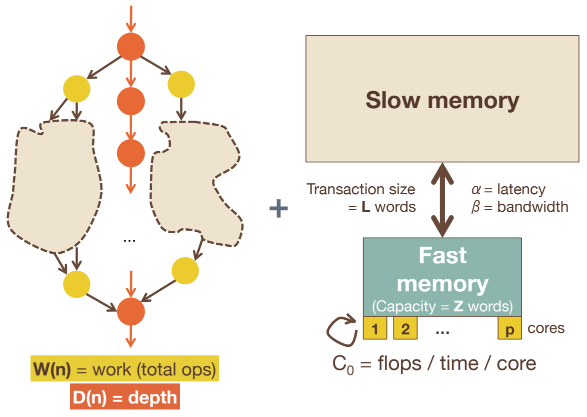

Figure 1: (*left*) A parallel computation in the work-depth model. (*right*) An abstract manycore processor with a large shared fast memory (cache or local-store).

of operations, where edges indicate dependencies, as illustrated in Fig. 1 (left). Given the DAG, we measure its *work*, $W(n)$, which is the total number of unit-cost operations for an input of size n; and its *depth* or *span*, $D(n)$, which is its critical path length measured again in unit-cost operations. Note that $D(n)$ defines the minimum execution time of the computation; $W(n)/D(n)$ measures the average available parallelism as each critical path node executes. The ratio $D(n)/W(n)$ is similar to the concept of a sequential fraction, as one might use in evaluating Amdahl's Law [4, 21, 39]. When we design an algorithm in this model, we try to ensure *work-optimality*, which says that $W(n)$ is not asymptotically worse than the best sequential algorithm, while also trying to maximize $W(n)/D(n)$.

Analyzing I/Os. To analyze the I/O behavior, we adopt the classical external memory model [3]. In this model, we assume a *sequential* processor with a two-level memory hierarchy consisting of a large but slow memory and a small fast memory of size Z words; work operations may only be performed on data that lives in fast memory. This fast memory may be an automatic cache or a software-controlled scratchpad; our analysis in this paper is agnostic to this choice. We may further consider that transfers between slow and fast memory must occur in discrete transactions (blocks) of size L words. When we design an algorithm in this model, we again measure the work, $W(n)$; however, we also measure $Q_{Z,L}(n)$, the number of L-sized transfers between slow and fast memory for an input of size n. There are several ways to design either cache-aware or cache-oblivious algorithms and then analyze $Q_{Z,L}(n)$ [3, 19, 22, 40]. In either case, when we design an algorithm in this model, we again aim for work-optimality while also trying to maximize the algorithm's *computational intensity*, which is $W(n)/(Q_{Z,L}(n) \cdot L)$. Intensity is the algorithmic ratio of operations performed to words transferred.

Architecture-specific cost model. The preceding analyses do not directly model time; for that, we need to choose an architecture model and define the costs. For this example, we use Fig. 1 (right) as our abstract machine, which is intended to represent a generic many-core system with p cores, each of which can deliver a maximum of C_0 operations per unit time; a fast memory of size Z words, shared among the cores; and a memory system whose cost to transfer $m \cdot L$ words is $\alpha + m \cdot L/\beta$, where α is the latency and β is the bandwidth in units of words per unit time. To estimate the best possible compute time, T_{comp}, we invoke Brent's theorem [12]:

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

where the operations on each core complete at a maximum rate of C_0 operations per unit time.

To model memory time, we need to convert $Q_{Z,L}(n)$, which is a measure of L-sized I/Os for the sequential case, into $Q_{p;Z,L}(n)$, or the total number of L-sized I/Os on p threads for the parallel case. One approach is to invoke recent results detailed by Blelloch et al. that bound $Q_{p;Z,L}(n)$ in terms of $Q_{Z,L}(n)$ [10]. To use these bounds, we need to select a scheduler, such as a work-stealing scheduler when the computation is nested-parallel [2], or a parallel depth-first scheduler [15]. For now, suppose we have either calculated $Q_{p;Z,L}(n)$ directly or applied an appropriate bound. To estimate the memory time cost, T_{mem}, suppose we charge the full latency α for each node on the critical path and assume that all $Q_{p;Z,L}(n)$ memory transfers will in the best case be aggregated and pipelined by the memory system and thereby be delivered at the peak bandwidth, β. Then,

$$T_{\text{mem}}(n; p, Z, L, \alpha, \beta) = \alpha \cdot D(n) + \frac{Q_{p;Z,L}(n) \cdot L}{\beta}. \quad (2)$$

The balance principle. A balance principle follows immediately from requiring that $T_{\text{mem}} \leq T_{\text{comp}}$, yielding (dropping n and subscripts for simplicity)

$$\underbrace{\frac{pC_0}{\beta}}_{\text{balance}} \left(1 + \underbrace{\frac{\alpha\beta/L}{Q/D}}_{\text{Little's}}\right) \leq \underbrace{\frac{W}{QL}}_{\text{intensity}} \left(1 + \underbrace{\frac{p}{W/D}}_{\text{Amdahl's}}\right). \quad (3)$$

Observe that Equ. (3) contains within it many of our intuitive performance engineering principles.

For example, Kung's classical balance principle [28] in our notation would have been

$$\frac{pC_0}{\beta} \quad \leq \quad \frac{W}{Q_{p;Z,L} \cdot L} \quad (4)$$

which says the machine's inherent balance point (left-hand side, peak operations to bandwidth) should be no greater than the algorithm's inherent computational intensity (right-hand side), measured in units of operations to words. Indeed, Equ. (4) also appears within our Equ. (3), but with two additional correction terms. As it happens, these correction terms correspond exactly to Little's Law and a generalization of Amdahl's Law. The Little's Law correction term says that the product of the machine's latency and bandwidth must be matched algorithmically by $Q_{p;Z,L}/D$, which we can interpret as the average degree of available memory parallelism. Similarly, the Amdahl-like correction term says that the number of cores must be matched by W/D, which is the algorithm's inherent average degree of available parallelism.

Examples. The interesting aspect of this formalism is that given a specific algorithm or class of computations characterized by W, D, and $Q_{p;Z,L}$, we can hope to see the precise relationships among algorithm and architecture characteristics.

For example, consider the multiplication of two $n \times n$ matrices. In this case, it just so happens that we can take $D(n) \ll W(n)$ and

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

due to Irony et al. [25].[2] Substituting these expressions into Equ. (3) yields a balance principle specific to matrix multiply, which when simplified becomes

$$\frac{p \cdot C_0}{\beta} \quad \leq \quad \sqrt{\frac{Z}{p}}. \quad (5)$$

What does this relation tell us? If we double cores with the intent of doubling performance, we can only do so if we also adjust Z, C_0, and/or β by, say, also doubling β *and* Z. The balance principle makes the relationship between architecture parameters for this algorithm precise.

Next, consider the problem of sorting. A deterministic cache-oblivious algorithm with a work-stealing scheduler has a balance constraint of

$$\frac{p \cdot C_0}{\beta} \quad \leq \quad \mathcal{O}\left(\log_2 \frac{Z}{p}\right) \quad (6)$$

[2]The bound on Q is a lower-bound on an entire class of so-called two-dimensional algorithms; thus, our analysis applies to the best possible algorithm in that class.

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

Table 1: Starting with the NVIDIA Fermi C2050, released in 2010, we project forward assuming growth rates that follow the last 30 years of CPU trends, expressed as the number of years required to double *except* for latency*, for which we show the halving trend. For Z/p in the last line, we use Z in 4-byte words not bytes.

assuming the $W(n) = \mathcal{O}(n \log n)$, $D(n) = \mathcal{O}\left(\log^2 n\right)$, and $Q_{Z,L}(n) = (n/L) \cdot \log_Z n$ and applying the appropriate bound to obtain $Q_{p;Z,L}(n)$ [10]. Unlike matrix multiply, this balance principle implies that increases in cache size for sorting have a relatively small effect, and is therefore a relatively less effective way to maintain balance. Fast Fourier transforms would be largely similar.

3 Examining Trends

We can also interpret a balance principle in light of architectural trends. This section considers such trends in terms of algorithm, memory, and power and energy.

Example: The shelf-life of balanced matrix multiply. What is the long-term impact of the recent manycore shift introduced by GPU systems on matrix multiplication? Past experience suggests that matrix multiply is largely compute-bound and will remain so more or less indefinitely. However, it should not be surprising that the manycore shift accelerates the pace at which even matrix multiply will become memory-bound. We estimate this transition will take place in as few as ten years.

Observe that the abstract manycore system depicted in Fig. 1 (right) is GPU-like, in the sense that there is a large shared fast memory and a main memory system designed to efficiently deliver words in batched transactions. For example, we could map an NVIDIA Fermi-based architecture [1] onto this model as follows.

- Take p to be the number of so-called NVIDIA CUDA cores, with C_0 as the per-core fused multiply-add throughput.
- The fast shared memory size Z as the total capacity of the registers and local-store memory ("shared memory" in NVIDIA parlance). We choose these rather than the shared L2 cache because the memory hierarchy size inversion actually makes the L2 cache *smaller* than even just the register file [34].
- The parameter L reflects the fact that a GPU memory system can merge multiple outstanding requests into fewer transactions. Though it is not needed for our matrix multiply bound, we could interpret L as the 128-byte minimum memory transaction size. Alternatively, we could use some other multiple of a so-called CUDA warp size, which is akin to a thread batch ("SIMT") or the concept of a SIMD vector width in a traditional architecture description.
- The α and β parameters are taken to be the main memory latency and peak bandwidth, respectively.

With this mapping, how does Equ. (5), evolve as the GPU architectures evolve?

To make this extrapolation, we need to make assumptions about how the various GPU architectural parameters will change over time. Since there are too few GPU data points on which to base an extrapolation, we instead make the following assumption. Although the GPU constitutes a jump in absolute peak performance (cores) and bandwidth, from this point forward it will still be subject to the same growth trends in numbers of transistors, decreases in latency, and increases in bandwidth. Indeed, since his 2004 paper on why latency lags bandwidth, Patterson's trends have largely held steady for CPUs [35], with Moore's Law for transistor growth expected to continue through approximately 2020. Thus, we extrapolate the GPU trends from this point forward, beginning with the parameters of an NVIDIA Fermi C2050, and projecting forward using the CPU trends as summarized in Table 1. The result for matrix multiply is shown in Fig. 2, which suggests that matrix multiply will in fact become imbalanced in the sense of Equ. (5) by 2020.

Example: Stacked memories. Equation 4 says one can balance an increase in cores by a similar increase in bandwidth. Stacked memory promises to deliver just this type of scaling, by scaling the number of pins or channels from memory to a processor with the surface area of the chip rather than its perimeter [30]. Now consider Equ. (5) and Equ. (6). Stacking will likely work for sorting, since the right-hand side of Equ. (6) depends on p through the relatively modest $\log_2 Z/p$; but *not* for matrix multiply where Z will still need to increase.

Example: Energy and power. Balancing in time is also desirable for minimizing power dissipation, because an imbalanced computation ($T_{\text{mem}} \geq T_{\text{comp}}$) spends cy-

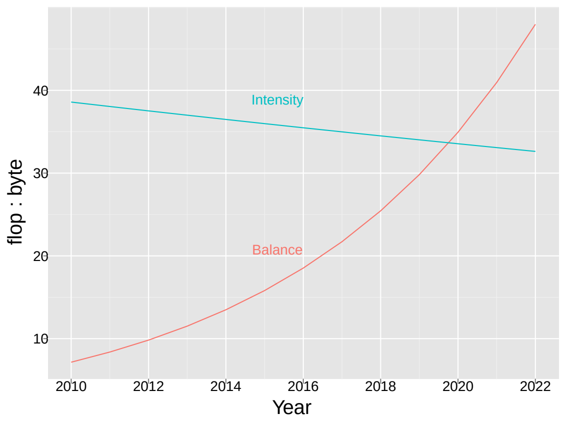

Figure 2: Extrapolation of the matrix multiply balance principle, Equ. (5), over time. The red line labeled *balance* is the left-hand side Equ. (5); the other line is the right-hand side.

cles idly waiting for transactions to arrive. Although the idle power dissipation of cores is relatively low in practice, the power spent idling begins to overcome power productively spent as $\frac{T_{\text{mem}}}{T_{\text{comp}}}$ exceeds $\frac{P_{\text{max}}}{P_{\text{idle}}}$. Minimizing $Q_{p;Z,L}$ by using an I/O-efficient algorithm is even more important, because increased communication is both a major drain of power and a source of imbalance.

4 Related Work

There is an extensive literature on analytical modeling of parallel algorithms and analyzing I/O [3, 6, 9, 14, 17, 22, 25, 29, 33, 36, 37], any of which might yield different balance principles from the ones shown here.

The most closely related of these models is the original machine balance characterizations of Kung and others [13, 28, 32, 38]. We extend the classical notion with explicit modeling of communication properties of both the architecture (latency, bandwidth, memory transfer size) and the algorithm (number of messages).

A second group of models are processor models based on reinterpretations of Amdahl's Law [21, 39]. Among other conclusions, these models make the case for asymmetric multicore processors quantitatively and intuitively; however, as it was not the primary goal, these models exclude explicit references to machine communication or algorithm detail, other than the usual sequential fraction that appears in classical Amdahl's Law.

There is increasing interest in co-design models for for future exascale systems [20]. These analyses suggest the allowable range of latency and bandwidth parameters required for scaling on an exascale system. We believe these analyses could be rewritten as balance principles to yield additional quantitative and qualitative claims.

Our example is "GPU-like," though it abstracts away many of the important details of GPU-specific models [16, 23, 41]. However, such models depend on the execution behavior of some code artifact, and so may be less useful for projections of the kind we consider here.

Lastly, we mention the recent interest in power- and energy-oriented algorithm models [7, 8, 26, 27, 31]. Based on our brief discussion of energy and power in § 3, it is not in our minds clear yet whether, from the perspective of algorithm design, there is really a fundamentally different approach required other than balancing in time as we do in this paper.

5 Conclusions and Extensions

Our balance principle analysis joins recent work on analytical models that try to sketch a path forward for multicore and manycore systems, such as Amdahl's Law-based analysis of Hill and Marty [21]. We believe the balance-based approach extends the insights of prior models with a more explicit binding of algorithm and architecture characteristics. Indeed, the balance principle of Equ. (3) yields Amdahl's Law as one component.

Looking forward, there are a variety of natural extensions, including (i) deriving balance principles for other machine models, with, say, deeper memory hierarchies, heterogeneous cores, or distributed memory; (ii) deriving balance principles for other algorithms, such as the Berkeley motifs [5]; (iii) consideration of alternative cost models beyond time; (iv) translating theory into practice by, for instance, creating balance models from actual programs, profiles, and/or execution traces.

Acknowledgements. We thank Logan Moon for his suggestions. This work was supported in part by the National Science Foundation (NSF) under award number 0833136, NSF CAREER award number 0953100, NSF I/UCRC award number 0934114, NSF TeraGrid allocations CCR-090024 and ASC-100019, joint NSF 0903447 and Semiconductor Research Corporation (SRC) Award 1981, and grants from the U.S. Dept. of Energy (DOE) and the Defense Advanced Research Projects Agency (DARPA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, SRC, DOE, or DARPA.

References

[1] NVIDIA's next generation CUDA compute architecture: Fermi, V1.1, Sept. 2009.

[2] U. A. Acar, G. E. Blelloch, and R. D. Blumofe. The data locality of work stealing. In *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures - SPAA '00*, pages 1–12, New York, New York, USA, July 2000. ACM Press.

[3] A. Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Aug. 1988.

[4] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, page 483, New York, New York, USA, 1967. ACM Press.

[5] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, S. Williams, J. Shalf, and Others. The landscape of parallel computing research: A view from berkeley, 2006.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in linear algebra. Technical Report UCB/EECS-2009-62, University of California, Berkeley, CA, USA, Feb. 2009.

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

[8] B. D. Bingham and M. R. Greenstreet. An energy aware model of computation. In *Proceedings of Exploiting Concurrency Efficiently and Correctly–EC^2*, Int'l. Conf. Computer Aided Verification (CAV), Princeton, NJ, USA, July 2008.

[9] G. E. Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, Mar. 1996.

[10] G. E. Blelloch, P. B. Gibbons, and H. V. Simhadri. Low depth cache-oblivious algorithms. In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures - SPAA '10*, page 189, New York, New York, USA, June 2010. ACM Press.

[11] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *ACM SIGPLAN Notices*, 30(8):207–216, Aug. 1995.

[12] R. P. Brent. The Parallel Evaluation of General Arithmetic Expressions. *Journal of the ACM*, 21(2):201–206, Apr. 1974.

[13] D. Callahan, J. Cocke, and K. Kennedy. Estimating interlock and improving balance for pipelined architectures. *Journal of Parallel and Distributed Computing*, 5(4):334–358, Aug. 1988.

[14] S. Chakrabarti, J. Demmel, and K. Yelick. Modeling the benefits of mixed data and task parallelism. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures - SPAA '95*, pages 74–83, New York, New York, USA, June 1995. ACM Press.

[15] S. Chen, T. C. Mowry, C. Wilkerson, P. B. Gibbons, M. Kozuch, V. Liaskovitis, A. Ailamaki, G. E. Blelloch, B. Falsafi, L. Fix, and N. Hardavellas. Scheduling threads for constructive cache sharing on CMPs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures - SPAA '07*, page 105, New York, New York, USA, 2007. ACM Press.

[16] J. W. Choi, A. Singh, and R. W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '10*, Bangalore, India, 2010. ACM Press.

[17] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1–12, July 1993.

[18] M. Frigo, P. Halpern, C. E. Leiserson, and S. Lewin-Berlin. *Reducers and other Cilk++ hyperobjects*. ACM Press, New York, New York, USA, Aug. 2009.

[19] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 285–297, New York, NY, USA, Oct. 1999. IEEE Comput. Soc.

[20] H. Gahvari and W. Gropp. An introductory exascale feasibility study for FFTs and multigrid. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–9, Atlanta, GA, USA, Apr. 2010. IEEE.

[21] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38, July 2008.

[22] J.-W. Hong 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.

[23] S. Hong and H. Kim. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. *ACM SIGARCH Computer Architecture News*, 37(3):152, June 2009.

[24] Intel Corporation. Intel Threading Building Blocks, 2009.

[25] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, Sept. 2004.

[26] R. Jain, D. Molnar, and Z. Ramzan. Towards a model of energy complexity for algorithms. In *Proc.˜IEEE Wireless Communications and Networking Conf.*, volume 3, pages 1884–1890, Mar. 2005.

[27] V. A. Korthikanti and G. Agha. Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures. In *2009 International Conference on Parallel Processing*, pages 212–219, Vienna, Austria, Sept. 2009. IEEE.

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

[29] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. Quantitative system performance: Computer system analysis using queueing network models, 1984.

[30] G. H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *2008 International Symposium on Computer Architecture*, pages 453–464. IEEE, June 2008.

[31] A. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 77(2-4):181–187, Feb. 2001.

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

[33] R. W. Numrich. A metric space for computer programs and the principle of computational least action. *The Journal of Supercomputing*, 43(3):281–298, Aug. 2007.

[34] D. Patterson. The top 10 innovations in the new NVIDIA Fermi architecture, and the top 3 next challenges, Sept. 2009.

[35] D. A. Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, 2004.

[36] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

[37] L. G. Valiant. A bridging model for multi-core computing. In *Proceedings of the European Symposium on Algorithms (ESA)*, volume LNCS 5193, pages 13–28, Universität Karlsruhe, Germany, Sept. 2008. Springer Berlin / Heidelberg.

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

[39] D. H. Woo and H.-H. S. Lee. Extending Amdahl's Law for energy-efficient computing in the many-core era. *IEEE Computer*, 41(12):24–31, Dec. 2008.

[40] K. Yotov, T. Roeder, K. Pingali, J. Gunnels, and F. Gustavson. An experimental comparison of cache-oblivious and cache-conscious programs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures - SPAA '07*, page 93, New York, New York, USA, 2007. ACM Press.

[41] Y. Zhang and J. D. Owens. A Quantitative Performance Analysis Model for GPU Architectures. In *17th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, San Antonio, TX, USA, 2011.