On Using the Roofline Model with Lower Bounds on Data Movement

VENMUGIL ELANGO and NASER SEDAGHATI, The Ohio State University
FABRICE RASTELLO, Inria
LOUIS-NOËL POUCHET, The Ohio State University
J. RAMANUJAM, Louisiana State University
RADU TEODORESCU and P. SADAYAPPAN, The Ohio State University

The roofline model is a popular approach for "bound and bottleneck" performance analysis. It focuses on the limits to the performance of processors because of limited bandwidth to off-chip memory. It models upper bounds on performance as a function of operational intensity, the ratio of computational operations per byte of data moved from/to memory. While operational intensity can be directly measured for a specific implementation of an algorithm on a particular target platform, it is of interest to obtain broader insights on bottlenecks, where various semantically equivalent implementations of an algorithm are considered, along with analysis for variations in architectural parameters. This is currently very cumbersome and requires performance modeling and analysis of many variants.

In this article, we address this problem by using the roofline model in conjunction with upper bounds on the operational intensity of computations as a function of cache capacity, derived from lower bounds on data movement. This enables bottleneck analysis that holds across all dependence-preserving semantically equivalent implementations of an algorithm. We demonstrate the utility of the approach in assessing fundamental limits to performance and energy efficiency for several benchmark algorithms across a design space of architectural variations.

Categories and Subject Descriptors: B.4.4 [**Hardware**]: Input/Output and Data Communications—*Performance analysis and design aids*; F.2 [**Analysis of Algorithms and Problem Complexity**]: General; D.2.8 [**Software**]: Metrics—*Performance measures*

General Terms: Performance, Algorithms

Additional Key Words and Phrases: Operational intensity upper bounds, I/O lower bounds, architecture design space exploration, algorithm-architecture codesign

ACM Reference Format:
Venmugil Elango, Naser Sedaghati, Fabrice Rastello, Louis-Noël Pouchet, J. Ramanujam, Radu Teodorescu, and P. Sadayappan. 2015. On using the roofline model with lower bounds on data movement. ACM Trans. Architec. Code Optim. 11, 4, Article 67 (January 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2693656

1. INTRODUCTION

Technology trends have resulted in widely different rates of improvement in computational throughput compared to data movement rates in computer systems. With future systems, the cost of data movement through the memory hierarchy is expected

Authors' addresses: V. Elango, N. Sedaghati, L.-N. Pouchet, R. Teodorescu, and P. Sadayappan, 2015 Neil Avenue, Columbus, OH 43210; emails: {elangov, sedaghat, pouchet, teodores, saday}@cse.ohio-state.edu; F. Rastello, Antenne Inria GIANT, DRT/LETI/DACLE - Batiment 51C - Bur. C424, Minatec Campus, 17 Rue des Martyrs, 38054 GRENOBLE cedex; email: fabrice.rastello@inria.fr; J. Ramanujam, 2027E Digital Media Center, Baton Rouge, LA 70803; email: jxr@cct.lsu.edu.

DOI: http://dx.doi.org/10.1145/2693656

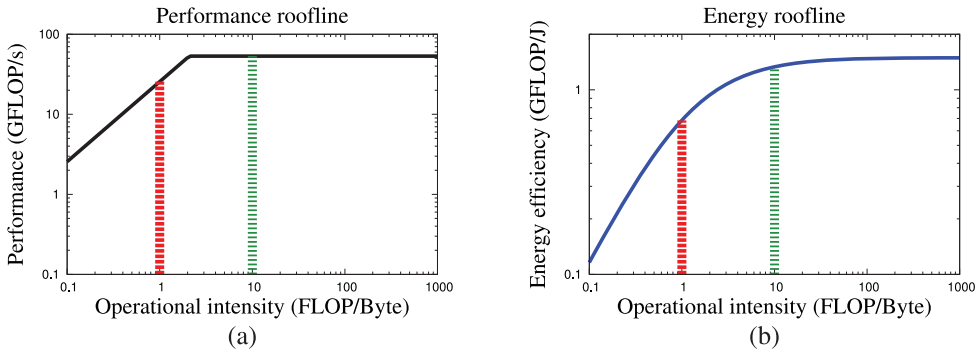


Fig. 1. Roofline model: performance and energy rooflines.

to become even more dominant relative to the cost of performing arithmetic operations [Bergman et al. 2008; Fuller and Millett 2011; Shalf et al. 2011], in terms of both time and energy. Therefore, optimizing data access costs will become ever more critical in the coming years. Given the crucial importance of optimizing data access costs in scalable parallel systems, it is of great interest to develop techniques for understanding performance limits dictated by data movement considerations.

The roofline model is an insightful visual "bound and bottleneck" model that focuses on the performance-limiting impact of off-chip memory bandwidth. In its most basic form, a performance roofline plot (explained in greater detail with Figure 1 in the next section) consists of two straight lines that represent upper bounds on performance due to the maximum computational rate of processor cores, and memory bandwidth, respectively. The horizontal axis is the "operational intensity" (*OI*) of the computation, defined as the ratio of number of computational operations performed per byte of data moved between main memory and the processor. A code will be memory-bandwidth limited unless *OI* is sufficiently high, greater than a "critical intensity" corresponding to the point of intersection of the two rooflines.

Modeled or measured data volume between main memory and last-level on-chip cache (LLC) for the execution of a code can be used to compute its *OI* and thus determine whether or not it is in the bandwidth-limited region of the roofline model for a machine. However, a significant issue is that the *OI* of an algorithm is not a fixed quantity, but depends on the on-chip LLC capacity and the problem size, and is affected by how the algorithm is implemented as well as the semantics-preserving code transformations that may be performed. Thus, it is only straightforward to use a roofline model to determine whether a particular implementation of an algorithm is bandwidth-bound on a particular machine. But we are often interested in broader insights on performance bottlenecks, where we wish to take into consideration potential transformations into semantically equivalent forms or consider architectural variations in a design space. But this is not easy. The standard approach to doing so requires performance modeling and analysis of a large number of alternative implementation scenarios.

In this article, we present a new approach to use of the roofline model by deriving upper bounds on *OI* from lower bounds on data movement. Since lower bounds on data movement are schedule-independent, its use avoids the need to explicitly model and analyze *OI* over a large number of mapping/scheduling scenarios. Lower bounds on data movement complexity of algorithms (also called I/O complexity in the literature) were first addressed by Hong and Kung [1981] using the model of the red-blue pebble game in their seminal work. Hong and Kung's model inherently allows recomputation of the same operation multiple times. This is useful in modeling algorithms that perform

recomputation of repeatedly used values rather than incurring the overhead of storing and loading the data. However, the majority of practically used algorithms do not perform any recomputation. Hence, several efforts [Ballard et al. 2011, 2012; Bilardi and Peserico 2001; Bilardi et al. 2012; Scquizzato and Silvestri 2013; Ranjan et al. 2011; Savage 1995, 1998; Savage and Zubair 2010; Cook 1974; Irony et al. 2004; Ranjan et al. 2012; Ranjan and Zubair 2012] have modeled I/O complexity under a more restricted model that disallows recomputation, primarily because it eases or enables the derivation of possibly tighter I/O lower bounds. In Elango et al. [2013], we describe an alternative approach for I/O lower bounds using graph min-cuts for the restricted model (Section 2.3). We use this min-cut-based technique to derive the I/O lower bounds (under the model that prohibits recomputation), and hence upper bounds on *OI*, for different algorithms in Section 3. We then use this upper-bound-based characterization in conjunction with the roofline model to capture the upper limits of performance of an algorithm over a range of possible alternative architectural configurations for a given VLSI technology. This article makes the following contributions:

—It derives tighter I/O lower bounds than previously known for some algorithms, using the min-cut-based techniques (Section 3).
—It presents a new approach to use the roofline model along with bounds on the maximum possible *OI* for an algorithm as a function of cache size. This bound applies to all possible valid schedules for the operations constituting the computation; that is, the bound is a fundamental upper limit for *OI*, irrespective of any optimization/transformation like loop tiling, fusion, and so forth.
—It uses the modeling of upper bounds on *OI* to enable the use of the roofline model in algorithm-architecture codesign exploration across an architectural design space:

(1) It models the maximal achievable performance of different algorithms (operations per second) for a given VLSI technology, considering different fractional chip area being allocated to cache versus cores (Section 5).
(2) It models maximal achievable energy efficiency (operations per joule) of different algorithms for a given VLSI technology, considering variation in number of cores, cache capacity, and voltage/frequency scaling (Section 6).

2. BACKGROUND

2.1. The Roofline Model

The roofline model [Williams et al. 2009] is a popular approach to bottleneck-bounds analysis that focuses on the critical importance of the memory bandwidth in limiting performance.

Performance Roofline: The performance roofline sets an upper bound on performance of a computation depending on its operational intensity. Let W be the number of operations (typically floating-point computations) that a particular implementation of an algorithm performs and let q be the total number of bytes it transfers between main memory and the processor. (Table I summarizes the parameters.)

The *operational intensity* of a computation is defined as the ratio between W and q, that is, $OI = W/q$. The performance roofline model ties together the operational intensity, peak floating-point performance, and memory performance in a two-dimensional graph. The horizontal axis (in log scale) represents the operational intensity and the vertical axis (in log scale) represents the attainable performance. The attainable performance for an architecture depends on its peak floating-point performance and the memory bandwidth and is given by

$$\text{Upper bound on attainable performance} = \text{Min}(F_{flops}, B_{mem} \times OI).$$

Table I. List of Symbols and Their Definitions

Symbol	Description
S	Size of the fast memory, e.g., cache (in words)
W	Number of arithmetic operations of an algorithm (in FLOP)
q	Number of data transfers for a particular implementation of an algorithm (in bytes)
Q	Minimum number of data transfers of an algorithm (in words)
OI	Operational intensity (in FLOP/byte)
F_{flops}	Peak computational rate (in GFLOP/s)
B_{mem}	Peak memory bandwidth to LLC (GBytes/sec)

In its most basic form, a performance roofline model contains two intersecting straight lines representing performance limits: a horizontal line at a height corresponding to the peak computational performance of the core(s) and an inclined line with slope corresponding to the peak memory-to-LLC bandwidth. This is shown in Figure 1(a).

The *OI* of a given implementation of an algorithm is typically computed using performance counters by measuring the actual number of operations and the amount of memory traffic during execution. In some cases, it may be possible to calculate *OI* through manual reasoning. If we draw a vertical line (red and green lines in Figure 1(a)) at this value of *OI*, it intersects either the inclined line, indicating that the implementation is bandwidth-bound, or the horizontal line, indicating that the implementation is compute-bound. For lower *OI*, the memory bandwidth is a fundamental limiting factor, and achievable computational throughput cannot exceed the product of *OI* and the memory bandwidth. As *OI* increases from zero, the upper bound on achievable performance steadily increases, up to the point of intersection of the two lines. To the right of the intersection point, *OI* is sufficiently high that the memory bandwidth is no longer a fundamental constraint on achievable computational throughput.

Energy Roofline: The roofline model has recently been adapted [Choi et al. 2013; Choi et al. 2014] to capture upper bounds on energy efficiency, that is, operations/joule, as a function of *OI*. For a particular *OI*, the energy cost of at least one memory access must be expended for every *OI* computational operations. The minimal energy cost for performing a set of W operations is the sum of the energy cost for actually performing the W arithmetic operations and the energy cost for W/OI memory operations. Figure 1(b) shows the energy roofline for the same architecture as the performance roofline (in Figure 1(a)). The energy roofline is smooth since the total energy is the sum of compute and data movement energies, in contrast to execution time, which is bounded by the larger of the data movement time and compute time (since data movement and computation may be overlapped).

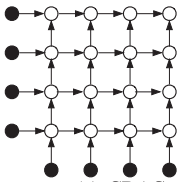
Given a particular algorithm, for example, the standard $O(N^3)$ algorithm for matrix-matrix multiplication, there exist many semantically equivalent implementations of the algorithm. For example, the standard triple-loop matrix multiplication algorithm allows for six possible permutations of the three loops—all produce exactly the same results but will generally incur a different number of cache misses and achieve different performance. A loop transformation like loop tiling produces semantically equivalent code that incurs fewer cache misses and hence achieves improved performance. In terms of the roofline model, the different semantically equivalent implementations of the standard matrix multiplication algorithm will clearly have different values of *OI*—all versions have exactly the same number of executed operations but differing amount of data transfer between main memory and cache. As elaborated in the next subsection,

```
for(i=1;i<N-1;i++)
 for(j=1;j<N-1;j++)
  A[i][j]=A[i-1][j]+A[i][j-1];
```

(a) Untiled code

```
for(it=1;it<N-1;it+=T)
 for(jt=1;jt<N-1;jt+=T)
  for(i=it;i<min(it+T,N-1);i++)
   for(j=jt;j<min(jt+T,N-1);j++)
    A[i][j]=A[i-1][j]+A[i][j-1];
```

(b) Equivalent tiled code



(c) CDAG

Fig. 2. Single-sweep two-point Gauss-Seidel code.

an algorithm can be abstracted by a computational directed acyclic graph (CDAG), and different semantically equivalent implementations of the algorithm (e.g., different loop permutations for triple-loop matrix multiplication) correspond to different valid schedules for execution of the primitive operations abstracted as vertices in the CDAG. Each valid schedule has a corresponding *OI* based on the number of data transfers between main memory and LLC. In general, finding the schedule with the highest possible *OI* would require analysis of a combinatorially explosive number of schedules. In contrast, with the approach we develop in this article, a single parametric expression can be developed for an upper bound on *OI* for a regular CDAG as a function of the size of LLC. The upper bound captures all possible valid schedules for the CDAG, including all tiled versions of the code, considering all possible tile sizes.

2.2. Upper Bounds on Operational Intensity via Lower Bounds on Data Movement

The range of possible values of *OI* for a particular algorithm has inherent upper limits for any given capacity of LLC. The upper bound on attainable *OI* for a computation is inversely related to the lower bound on the minimum number of data elements that must be moved between main memory and cache in order to execute all operations of an algorithm. If the number of operations W is invariant across different execution scenarios to be considered (including dependence-preserving program transformations and change in the cache capacity), maximizing *OI* is equivalent to minimizing the amount of data transferred between main memory and cache. The problem of finding lower bounds on the minimal required data movement in a memory hierarchy has been studied in the literature [Hong and Kung 1981; Aggarwal and Vitter 1988; Aggarwal et al. 1987; Irony et al. 2004; Bilardi et al. 2000; Bilardi and Peserico 2001; Savage 1995, 1998; Ranjan et al. 2011, 2012; Valiant 2011; Demmel et al. 2012; Ballard et al. 2011, 2012; Christ et al. 2013; Solomonik et al. 2013; Savage and Zubair 2010], where the term I/O lower bound is often used to refer to a data movement lower bound.

Consider the code shown in Figure 2(a) that performs $(N-2)^2$ arithmetic operations. Figure 2(b) shows a functionally equivalent form of the same computation, after a tiling transformation. The tiled form too has exactly the same number of arithmetic operations ($(N-2)^2$). Next, let us consider the data access cost for execution of these two code forms on a processor with a single level of cache. If the problem size N is larger than cache capacity, the number of cache misses would be higher for the untiled version (Figure 2(a)) than for the tiled version (Figure 2(b)). But if the cache size

were sufficiently large, the tiled version would not offer any benefits in reducing cache misses.

Thus, unlike the operation count of an algorithm, which stays unchanged for different valid orders of execution and is also independent of machine parameters like cache size, the data access cost depends both on the cache capacity and on the order of execution of the operations of the algorithm.

In order to model the range of valid scheduling orders for the operations of an algorithm, it is common to use the abstraction of the CDAG (Definition 2.1), with a vertex for each instance of each computational operation, and edges from producer instances to consumer instances. We use the notation of Bilardi and Peserico [2001] to describe the CDAG model:

Definition 2.1 (*CDAG*). A CDAG is a 4-tuple $C = (I, V, E, O)$ of finite sets such that (1) $I \subset V$ is the input set and all its vertices have no incoming edges, (2) $E \subseteq V \times V$ is the set of edges, (3) $G = (V, E)$ is a directed acyclic graph, (4) $V \backslash I$ is called the operation set and all its vertices have one or more incoming edges, and (5) $O \subseteq V$ is called the output set.

The vertices V of a CDAG $C = (I, V, E, O)$ are not restricted to binary operations and, hence, can have zero or more predecessors corresponding to the operands of the respective statements in the original code. Figure 2(c) shows the CDAG for the codes in Figure 2(a) and Figure 2(b), for $N = 4$; although the relative order of operations is different between the tiled and untiled versions, the set of computation instances and the producer–consumer relationships for the flow of data are exactly the same (the special "input" vertices in the CDAG represent values of elements of A that are read before they are written in the nested loop).

The Red-Blue Pebble Game: The seminal work of Hong and Kung [1981] was the first to develop an approach to bounding the minimum data movement in a two-level hierarchy, among all possible valid execution orders, of the operations of a CDAG. The inherent I/O complexity of a CDAG is modeled as the minimal number of I/O operations needed in playing the *red-blue pebble game* described next. This game uses two kinds of pebbles: a fixed number of red pebbles that represent locations in a small, fast local memory (could represent cache, registers, etc.) and an arbitrarily large number of blue pebbles that represent large, slow main memory.

Definition 2.2 (*red-blue pebble game [Hong and Kung 1981]*). Let $C = (I, V, E, O)$ be a CDAG such that any vertex with no incoming (resp. outgoing) edge is an element of I (resp. O). Given S red pebbles and an arbitrary number of blue pebbles, with an initial blue pebble on each *input* vertex, a complete calculation is any sequence of steps using the following rules that results in a final configuration with blue pebbles on all *output* vertices:

- **R1** ***(Input)***. A red pebble may be placed on any vertex that has a blue pebble (load from slow to fast memory).
- **R2** ***(Output)***. A blue pebble may be placed on any vertex that has a red pebble (store from fast to slow memory).
- **R3** ***(Compute)***. If all immediate predecessors of a vertex of $V \setminus I$ have red pebbles, a red pebble may be placed on (or moved to) that vertex (execution or "firing" of operation).
- **R4** ***(Delete)***. A red pebble may be removed from any vertex (reuse storage).

The number of I/O operations for any complete calculation is the total number of moves using rules R1 or R2, that is, the total number of data movements between the fast and slow memories. The inherent I/O complexity of a CDAG is the smallest number of such I/O operations that can be achieved among all possible complete calculations for that CDAG. An *optimal* calculation is a complete calculation achieving the minimum number of I/O operations.

While the red-blue pebble game provides an operational definition for the I/O complexity problem, it is generally not feasible to determine an optimal calculation on a CDAG. Hong and Kung [1981] developed a novel approach for deriving I/O lower bounds for CDAGs by relating the red-blue pebble game to a graph partitioning problem, called S-partitioning, and proved I/O lower bound results for several CDAGs by reasoning about the maximum number of vertices that could belong to any vertex set in a valid $2S$-partition.

Lower Bounds on Data Movement for Parallel Execution: So far, the discussion of data movement complexity has been restricted to sequential computation. But the model can be extended to reason about data movement bottlenecks for parallel execution of a program on a shared-memory multicore system, as done by Savage and Zubair [2008] and Elango et al. [2014]. Savage and Zubair [2008] proposed the Multiprocessor Memory Hierarchy Game (MMHG) that models data movement in a shared storage hierarchy with a common shared level of memory that can be accessed by all processors. MMHG can be used to model lower bounds on the volume of data movement at different levels of the cache hierarchy for a shared-memory multiprocessor. We remark that using an upper bound on *OI* using the MMHG model can only lead to lower or equal value for *OI* upper bounds, compared to using the traditional RB game and modeling only the LLC. Using just the standard RB game and the LLC capacity indeed gives valid upper bounds on *OI* for any parallel execution because the volume of data movement between main memory and LLC due to the collective operations of all cores is only dependent on LLC capacity and not the schedule of execution, whether sequential or parallel. We would simply be ignoring the data movement between higher-level caches by doing so. Thus, although in this article we limit our analysis to a single LLC shared across cores for reasons of simplicity, all performance and energy efficiency bounds we obtain are necessarily valid upper bounds to those that might be obtained using the same methodology with more detailed modeling of the different levels of the cache hierarchy and the MMHG game.

2.3. Min-Cut-Based I/O Lower Bound

In Elango et al. [2013], we developed an alternative lower bounding approach. It was motivated from the observation that the Hong and Kung 2S-partitioning approach does not account for the internal structure of a CDAG, but essentially focuses only on the boundaries of the partitions. In contrast, the min-cut-based approach captures internal space requirements using the abstraction of wavefronts. This section describes the approach. This approach is used later in Section 3 to derive *OI* upper bounds for different algorithms. The central idea is the definition of two kinds of *wavefronts* and the relation between these:

Definitions: Given a graph $G = (V, E)$, a *cut* $(\mathcal{S}, \mathcal{T})$ is defined as any partition of the set of vertices V into two disjoint subsets $\mathcal{S}$ and $\mathcal{T} = V \backslash \mathcal{S}$. An *s-t* cut is defined with respect to two distinguished vertices s and t and is any $(\mathcal{S}, \mathcal{T})$ cut satisfying the requirement that $s \in \mathcal{S}$ and $t \in \mathcal{T}$. Each cut defines a set of cut edges (the cut-set), that is, the set of edges $(u, v) \in E$, where $u \in \mathcal{S}$ and $v \in \mathcal{T}$. Given any cut edge $e = (u, v)$, the

vertex u is called a *cut vertex*. The set of cut vertices is called the *cut-vertex-set*. The capacity of a cut is the cardinality of its cut-vertex-set.

Given a DAG $G = (V, E)$ and some vertex $x \in V$, the set $\mathsf{Anc}(x)$ is the set of vertices from which there is a nonempty directed path to x in G ($x \notin \mathsf{Anc}(x)$); the set $\mathsf{Desc}(x)$ is the set of vertices to which there is a nonempty directed path from x in G ($x \notin \mathsf{Desc}(x)$). Using these two notations, a *convex cut* or a *convex partition* is defined as follows:

Definition 2.3 (*Convex cut or convex partition*). Given a graph $G = (V, E)$ and a vertex $x \in V$, a convex cut or a convex partition $(\mathcal{S}_x, \mathcal{T}_x)$ associated to x is an s-t cut with the following properties:

(1) $x \cup \mathsf{Anc}(x) \subseteq \mathcal{S}_x$.
(2) $\mathsf{Desc}(x) \subseteq \mathcal{T}_x$.
(3) $E \cap (\mathcal{T}_x \times \mathcal{S}_x) = \varnothing$.

A convex cut with minimum capacity is called a *convex min-cut*.

Schedule Wavefront: Consider a complete calculation $\mathcal{P}$ that corresponds to some scheduling (i.e., execution) of the vertices of the graph $G = (V, E)$. We view this complete calculation $\mathcal{P}$ as a string that has recorded all the transitions (applications of pebble game rules). Given $\mathcal{P}$, we define the *wavefront* $W_{\mathcal{P}}(x)$ induced by some vertex $x \in V$ at the point when x has just fired as the union of x and the set of vertices that have already fired and that have an outgoing edge to a vertex $v \in V$ that has not yet been fired. With respect to a complete calculation $\mathcal{P}$, the set $W_{\mathcal{P}}(x)$ defines the memory requirements at the timestamp just after x has fired.

Correspondence with Graph Min-Cut: Note that there is a close correspondence between the wavefront $W_{\mathcal{P}}(x)$ induced by some vertex $x \in V$ and the $(\mathcal{S}_x, \mathcal{T}_x)$ partition of the graph G. Given a convex partition $(\mathcal{S}_x, \mathcal{T}_x)$ of G, we can construct a complete calculation $\mathcal{P}$ in which at the timestamp when x has just fired, the subset of vertices of V that have already been fired exactly corresponds to $\mathcal{S}_x$; the set of fired vertices that have a successor that is not fired constitute a wavefront $W_{\mathcal{P}}(x)$ associated with x. Similarly, given a wavefront $W_{\mathcal{P}}(x)$ associated with x in a pebble game instance $\mathcal{P}$, we can construct a valid $(\mathcal{S}_x, \mathcal{T}_x)$ convex partition by placing all fired vertices in $\mathcal{S}_x$ and all the nonfired vertices in $\mathcal{T}_x$.

A minimum cardinality wavefront induced by x, denoted $W_G^{\min}(x)$, is a convex min-cut that results in an $(\mathcal{S}_x, \mathcal{T}_x)$ partition of G defined earlier. We define $w_G^{\max}$ as the maximum value over the size of all possible minimum cardinality wavefronts associated with vertices, that is, define $w_G^{\max} = \max_{x \in V}(|W_G^{\min}(x)|)$.

LEMMA 2.4. *Let $C = (\varnothing, V, E, O)$ be a CDAG with no inputs. For any $x \in V$, $Q \geq 2(|W_G^{min}(x)| - S)$. In particular, $Q \geq 2(w_G^{max} - S)$, where Q is the number of I/O operations for the optimal calculation of C.*

3. OPERATIONAL INTENSITIES OF ALGORITHMS

In this section, we develop expressions for upper bounds on *OI* for four algorithms as a function of storage capacity—matrix multiplication (MM), fast Fourier transform (FFT), conjugate gradient (CG), and 9-point 2D Jacobi computation (J2D). The *OI* upper bounds are used in the following sections for architecture-algorithm codesign exploration. The upper bounds on *OI* for MM and CG are obtained using previously known I/O lower bounds, while we derive new I/O lower bounds for FFT and J2D using the min-cut-based approach (Section 2.3). In modeling *OI* in operations per byte, we assume the data to be double precision occupying 8 bytes per word and each red/blue pebble to have a capacity of one word.

3.1. Matrix-Matrix Multiplication (MM)

The standard matrix-matrix multiplication algorithm has an operation count, $W = 2N^3$ for the product of $N \times N$ matrices. Irony et al. [2004] showed that the minimum I/O cost Q for MM satisfies $Q \geq \frac{N^3}{2\sqrt{2S}}$ for S red pebbles. Hence, the operational intensity OI satisfies $OI \leq \frac{2N^3}{N^3/(2\sqrt{2S})} = 4\sqrt{2S}$ FLOP/word, or $0.5\sqrt{2S}$ FLOP/byte.

3.2. Fast Fourier Transform (FFT)

The N-point FFT (of height $\log(N)$) has an operation count of $2N\log(N)$. The following theorem develops an I/O lower bound for FFT using the min-cut approach discussed in Section 2.3.

THEOREM 3.1 (I/O LOWER BOUND FOR FFT). *For an N-point FFT graph, the minimum I/O cost, Q, satisfies*

$$Q \geq \frac{2N\log(N)}{\log(S)},$$

where S is the number of red pebbles.

PROOF. Consider a CDAG for an FFT of size m (with no input vertices). Consider a complete calculation $\mathcal{P}$ with minimum I/O and the timestamp at which the first output vertex (i.e., vertex with no successors) o is fired in $\mathcal{P}$. Let $\mathcal{S}$ be the vertices already fired strictly before o, and $\mathcal{T}$ the others. As o is an output vertex, $\mathcal{S}$ contains all the m input vertices. By construction, $\mathcal{T}$ contains all the m output vertices, and hence, the corresponding wavefront, $|W_{\mathcal{P}}(o)| \geq m$.

Now, a CDAG for an N-point FFT (of height $\log(N)$ with the input vertices removed) can be decomposed into disjoint sub-DAGs corresponding to m-point FFTs (and of height $\log(m)$). This gives us $\lfloor N/m \rfloor \times \lfloor log(N)/log(m) \rfloor$ full sub-CDAGs. From Lemma 2.4, the I/O cost of each sub-FFT is at least $2 \times (m - S)$. If we consider $m = S\log(S)$,

$$\begin{aligned}
Q &\geq \left\lfloor \frac{N}{S\log(S)} \right\rfloor \times \left\lfloor \frac{\log(N)}{\log(S\log(S))} \right\rfloor \times 2\left(S\log(S) - S\right) \\
&\geq \left(\frac{N - S\log(S)}{S\log(S)}\right) \times \left(\frac{\log(N) - \log(S\log(S))}{\log(S\log(S))}\right) \times 2\left(S\log(S) - S\right) \\
&\geq \left(\frac{N - S\log(S)}{S\log(S)}\right) \times \left(\frac{\log(N) - \log(S\log(S))}{\log(S\log(S))}\right) \times 2\left(S\log(S)\right) - O\left(\frac{N\log(N)}{\log^2(S)}\right) \\
&\geq \frac{N\log(N)}{S\log^2(S)} \times 2(S\log(S)) - O(N) - O\left(\frac{N\log(N)}{\log^2(S)}\right) \\
&\geq \frac{2N\log(N)}{\log(S)} - O\left(\frac{N\log(N)}{\log^2(S)}\right). \quad \square
\end{aligned}$$

Finally, from the operation count and the I/O complexity for FFT, we obtain the upper bound on the operational intensity, $OI \leq \frac{2N\log(N)}{(2N\log(N))/(\log(S))} = \log(S)$ FLOP/word, or $0.125\log(S)$ FLOP/byte.

3.3. Conjugate Gradient (CG)

Many compute-intensive applications involve the numerical solution of partial differential equations (PDEs), for example, solving a heat equation on a two-dimensional plate. The first step in numerical solution of such problems involves *discretization*, where the continuous domain is reduced to discrete points at regular intervals. The problem is

```
1  x is the initial guess
2  p ← r ← b − Ax
3  do
4     v ← Ap                                          //SpMV
5     b ← (r.r)                                       //Dot product
6     a ← b/(p.v)                                     //Dot product
7     x ← x + ap                                      //AXPY
8     r ← r − av                                      //AXPY
9     g ← (r.r)/b                                     //Dot product
10    p ← r + gp                                      //AXPY
11 until ((r.r) is small)
```

Fig. 3. Conjugate gradient method.

then solved only at these discrete points, called a *computational grid*. Solution to such discretized problems involves solving a system of linear equations at each time step till convergence. Additional details regarding this process can be found in Elango et al. [2014, Section 5.1]. These linear systems, of the form $\mathbf{Ax} = \mathbf{b}$, are typically made of banded sparse matrix $\mathbf{A}$ with a repetitive pattern. Hence, in practice, the elements of $\mathbf{A}$ are not explicitly stored. Instead, their values are directly embedded in the program as constants, thus eliminating the space requirement and the associated I/O cost for the matrix.

The conjugate gradient method [Hestenes and Stiefel 1952] is one of several popular methods to iteratively solve such linear system. CG maintains three vectors at each time step—the approximate solution $\mathbf{x}$, its residual $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$, and a search direction $\mathbf{p}$. At each step, $\mathbf{x}$ is improved by searching for a better solution in the direction $\mathbf{p}$. Each iteration of CG involves one sparse matrix-vector product, three vector updates, and three vector dot-products. The complete pseudo-code is shown in Figure 3.

The I/O lower bound for CG was derived in Elango et al. [2014] using the min-cut approach. We state the theorem next. The complete proof can be found in Elango et al. [2014, Theorem 8].

THEOREM 3.2 (I/O LOWER BOUND FOR CG). *For a d-dimensional grid of size N^d, the minimum I/O cost, Q, to solve the linear system using CG satisfies $Q \geq 6N^dT$, when $N^d \gg S$, where T represents the number of outer loop iterations.*

Considering a two-dimensional grid ($d = 2$), we obtain an operation count of $20N^3T$ and hence an upper bound on OI of $OI \leq 20/6$FLOP/word, or 0.417FLOP/byte.

3.4. 9-Point Jacobi 2D (J2D)

We consider the Jacobi computation that computes at time step t the value of every point of an $N \times N$ array A in terms of its nine neighbors at time step $t-1$ as follows:

```
A[i][j] = (B[i-1][j-1] + B[i-1][j] + B[i-1][j+1]
          + B[i][j-1]
          + B[i][j] + B[i][j+1]+ B[i+1][j-1]
          + B[i+1][j] + B[i+1][j+1] )*cnst;
```

THEOREM 3.3 (I/O LOWER BOUND FOR JACOBI 2D). *For the 9-point Jacobi of size $N \times N$ with T time steps, the minimum I/O cost, Q, satisfies*

$$Q \geq 0.75\frac{N^2T}{\sqrt{S}},$$

where S is the number of red pebbles.

PROOF. Jacobi 2D has a three-dimensional iteration space of size $N^2 \times T$. Consider a partition of the iteration space with cubes of size $(2m+1)^3$. We have $\lfloor N/(2m+1) \rfloor^2 \times \lfloor T/(2m+1) \rfloor$ such cubes. For each cube, let x be the vertex at the center of the cube. The set of ancestors of x contains the lower face of the cube. The set of descendants of x contains the upper face. We have disjoint vertical paths connecting each point in the lower face with a point in the upper face of the cube. Any wavefront must cut this set of vertical paths. Hence, the wavefront just after firing x contains at least the number of points present in a horizontal face of the cube, that is, $(2m+1)^2$. From Lemma 2.4, the I/O cost of each cube is then $2((2m+1)^2 - S)$. Considering all $\lfloor N/(2m+1) \rfloor^2 \times \lfloor T/(2m+1) \rfloor$ such cubes, we get a total I/O cost of $\lfloor N/(2m+1) \rfloor^2 \times \lfloor T/(2m+1) \rfloor \times 2((2m+1)^2 - S) - |I| + |O|$.

By setting $m = \sqrt{S}$,

$$
\begin{aligned}
Q &\geq \left\lfloor \frac{N}{2\sqrt{S}+1} \right\rfloor^2 \times \left\lfloor \frac{T}{2\sqrt{S}+1} \right\rfloor \times 2((2\sqrt{S}+1)^2 - S) \\
&\geq \left(\frac{N - 2\sqrt{S}}{2\sqrt{S}+1} \right)^2 \times \left(\frac{T - 2\sqrt{S}}{2\sqrt{S}+1} \right) \times (6S + 8\sqrt{S} + 2) \\
&\geq \left(\frac{N - 2\sqrt{S}}{2\sqrt{S}+1} \right)^2 \times \left(\frac{T - 2\sqrt{S}}{2\sqrt{S}+1} \right) \times 6S \\
&\geq \left(\frac{N}{2\sqrt{S}+1} - 1 \right)^2 \times \left(\frac{T}{2\sqrt{S}+1} - 1 \right) \times 6S \\
&\geq \left(\frac{N^2 T}{(2\sqrt{S})^3} \right) \times 6S - O\left(\frac{N^2}{\sqrt{S}} + \frac{NT}{\sqrt{S}} \right) \\
&\geq 0.75 \frac{N^2 T}{\sqrt{S}} - O\left(\frac{N^2}{\sqrt{S}} + \frac{NT}{\sqrt{S}} \right). \quad \square
\end{aligned}
$$

The 9-point Jacobi has an operation count of $9N^2T$. Hence, we obtain an upper bound on operational intensity, $OI \leq 12\sqrt{S}$FLOP/word, or $1.5\sqrt{S}$FLOP/byte.

4. ARCHITECTURE DESIGN SPACE EXPLORATION

In the next two sections, we show how an analysis of upper bounds on *OI* of an algorithm as a function of cache size can be used to model the limits of achievable performance as well as energy efficiency. We perform two types of analysis:

—For a particular set of technology parameters, what are the upper bounds on achievable per-chip performance (GFLOP per second) for different algorithms, considering a range of possible ways of dividing the chip's area among processor cores versus cache memory (Section 5)?
—For a given set of technology parameters, what are the upper bounds on energy efficiency (Giga-operations per joule) for different algorithms (Section 6)?

In this section, we provide details on how various architectural parameters were chosen for the analysis.

4.1. Notation

We consider a chip with a fixed total area A, where a portion $\alpha.A$, of the total area is used for the last-level cache and the remaining area, $(1-\alpha).A$, is allocated to cores. While such an analysis could be directly extended to model constraints with respect to multiple levels of cache, we only model the impact of the LLC in this analysis. Also, we do not

Table II. Notation Used in Architecture Design Space Exploration

A,	Total chip area
$\alpha.A$,	Area occupied by LLC
$(1-\alpha).A$,	Area occupied by cores
f	Clock frequency (GHz)
P	Number of cores
S	Last-level cache size in words
π_{cpu}	Static leakage power per core
π_{cache}	Static leakage power for cache of size S
ϵ_{flop}	Energy per arithmetic operation at frequency f
ϵ_{mem}	Energy per byte of data access from/to memory

account for the area required for interconnects and other needed logic. It is important to note that since our modeling is purely an upper-bound analysis on performance and energy efficiency, the results obtained are of course valid under these simplifications: including more details of the architecture can allow tighter bounds, but all results presented later in this section are assertable upper limits on performance and energy efficiency for the modeled technology. For example, when the analysis shows that large-FFT (size too large to fit in LLC) performance cannot exceed 60 GFLOP/s aggregate performance for a multicore chip built using the modeled technology, it represents an upper bound that cannot be exceeded by any possible actual implementation with those technology parameters. A more detailed model that accounts for additional levels of cache, on-chip interconnect, control logic, and so forth may serve to tighten or reduce the upper bound to a lower value than 60GFLOP/s but cannot invalidate the conclusions drawn from the more simplified analysis.

The following valid simplifications are made to make the analysis easier: non-core and I/O clock frequencies are assumed to be fixed, interconnects are not considered in the analysis, and LLC is considered to be shared by all the cores.

In our architectural design space exploration, the number of cores, P, and the total number of locations in the LLC, S, are directly related to the total die area A, and the parameter α trades off one for the other. $\boldsymbol{F}_{flops}(f)$ varies with the clock frequency f.

Performance. The total computation time T follows the roofline model and can be expressed as

$$T = \max\left(\frac{W}{P.\boldsymbol{F}_{flops}(f)}, \frac{8 \times \boldsymbol{Q}}{\boldsymbol{B}_{mem}}\right) = \max\left(\frac{W}{P.\boldsymbol{F}_{flops}(f)}, \frac{W}{\boldsymbol{B}_{mem}.OI(S)}\right). \tag{1}$$

For a given application, W can be expressed in terms of problem size parameters, and lower bounds on $\boldsymbol{Q}$ (in words) can be expressed through our analysis in terms of S. As an example, for matrix multiplication we have $W = 2N^3$, $\boldsymbol{Q} \geq \frac{N^3}{2\sqrt{2S}} + 3N^2$.

Energy. The total energy consumption is modeled as

$$\boldsymbol{E} = W.\epsilon_{flop}(f) + 8\boldsymbol{Q}.\epsilon_{mem} + (P.\pi_{cpu} + \pi_{cache}(S)).\boldsymbol{T}, \tag{2}$$

where the quantities used are defined in Table II.

4.2. Architectural Parameters

We demonstrate the utility of modeling upper bounds on operational intensity by performing an analysis over possible architectural variations for a given technology. We use architectural parameters for an enterprise Intel Xeon processor (codename *Nehalem-EX*) [Rusu et al. 2009] and use the published and available statistics to estimate area, power and energy for different compute and memory units.

Table III. Nehalem-EX Processor Spec

Parameter	Value
Die size	684mm^2
Node technology	45nm
Num of cores	8
Num of LLC slices	8
LLC slice size	3MB
LLC size (total)	24MB
DRAM channels	4
Core voltage	0.85–1.1V
Core max clock	2.26GHz
TDP	130W
Threads per core (SMT)	2
Leakage (total)	21W
L1/L2	32KB/256KB

Table IV. Nehalem-EX Die Dimensions

Unit	Width (mm)	Height (mm)	Area (mm^2)	Area (%)
Die	31.9	21.4	684	100
Core	7.2	3.7	26.7	3.9
LLC slice	7.8	3.7	29.3	4.3

Table V. Modeled LLC Slices Using CACTI [Muralimanohar et al. 2009]

Size (KB)	H (mm)	W (mm)	Area (mm^2)
4	0.341	0.912	0.311
8	0.368	0.912	0.336
16	0.472	0.966	0.456
32	0.580	1.010	0.586
64	0.815	1.015	0.827
128	0.848	2.032	1.723
256	0.870	4.060	3.532
384	1.096	4.066	4.456
512	2.798	2.778	7.773
1,024	4.632	2.800	12.970
2,048	5.080	5.138	26.101
3,072	7.446	5.353	39.858
4,096	5.859	9.173	53.745
8,192	9.518	9.791	93.191

Table III shows the physical specifications for a testbed CPU. Using the die dimensions, and by processing the die photo, we computed the area (in mm^2) for the chip-level units of interest. For this study, we model core area, which includes private L1 and L2 caches. We also model a range of values for the shared LLC.

Table IV shows the extracted dimensions for the Nehalem-EX core and LLC. In order to explore LLC with different sizes, we used CACTI [Muralimanohar et al. 2009]. We fixed LLC design parameters based on the chip specification, varying only the size of the cache.

Table V shows the modeled cache sizes and their corresponding area (from CACTI). Using the area percentage for each unit and the reported total leakage power for the chip (21W, or 16%), we modeled the static power consumed by each core on an area-proportional basis.

Table VI. Nehalem-EX: Effect of Changing Voltage and Frequency on Core/Chip Power

Voltage (V)	Clock (MHz)	Peak Chip Dyn (W)	Peak Core Dyn (W)
1.10	2,260	111.14	102.68
1.05	2,060	95.94	87.49
1.00	1,860	81.82	73.35
0.95	1,660	62.23	60.78
0.90	1,460	58.12	49.66
0.85	1,260	48.39	39.93

In order to model dynamic power per core, we used McPAT [Li et al. 2009], an analytical tool to model the CPU pipeline and other structures. To estimate core parameters, we extended the available Xeon model to allow for a larger number of cores. All modifications were based on real parameters published by Intel [Rusu et al. 2009]. In order to model the impact of voltage/frequency scaling on energy efficiency, we extracted the maximum and minimum operating voltage for the processor and the corresponding frequencies at which the processor can operate. Using those V/F pairs, different "power states" were modeled for the processor using McPAT [Li et al. 2009]. Table VI shows how changing voltage and frequency affects total chip and core power.

Finally, we summarize the parameters for the testbed architecture used for the modeling in the next sections: $F_{flops}(f) = 9.04$GFLOP/s @ 2.26GHz, $B_{mem} = 40$GB/s, $A, = 684\text{mm}^2$, $A_{,core} = 26.738\text{mm}^2$, $\epsilon_{flop}(f) = 1.3$nJ/flop, $\epsilon_{mem} = 0.63$nJ/byte, $\pi_{cpu} = 0.819$W. Each core, which includes private L1/L2 caches, occupies around 4% of the total die area. For a very small value of α, a maximum of 25 cores would fit on the chip. At the other extreme, for α of 0.95, only one core can be put on the chip and an LLC cache of 64MB could be accommodated.

5. ALGORITHM-ARCHITECTURE CODESIGN: BOUNDS ON PERFORMANCE

In this section, for different algorithms, we consider the implications of upper bounds on *OI* on the maximal performance (operations per second) achievable on a chip. As described in the previous section, we assume that the chip area can be partitioned as desired among processor cores or cache. As shown later for four demonstration benchmarks, the upper bound on *OI* for an algorithm can be modeled as a monotonic nondecreasing function of cache size. As the fractional chip area occupied by LLC increases, the upper bound on *OI* increases. But simultaneously, the fractional area usable for cores decreases, so that fewer cores may be placed on the chip. As shown next, a collection of rooflines can be used to capture the architecture design space. Alternatively, we show that the collective data can be captured in a single multiroofline plot.

The size of the LLC was varied from a tiny 4KB size (representing a fraction of under 0.1% of the chip of approximately $700mm^2$ area) to 64MB (filling almost the entire chip area). For demonstration purposes, we analyze four algorithms: matrix multiplication (MM), fast Fourier transform (FFT), conjugate gradient (CG), and 9-point 2D Jacobi (J2D) iterative linear system solvers. In all these cases, the problem sizes were assumed to be much larger than LLC, when *OI* is essentially independent of the problem size, as per the analysis in Section 3. The following upper bounds were obtained for *OI*, in FLOP/byte, as a function of cache capacity (S words), for double-precision data occupying 8 bytes per word.:

—MM: $0.5\sqrt{2S}$
—FFT: $0.125\log(S)$
—CG: 0.417 (it is independent of cache capacity)
—J2D: $1.5\sqrt{S}$

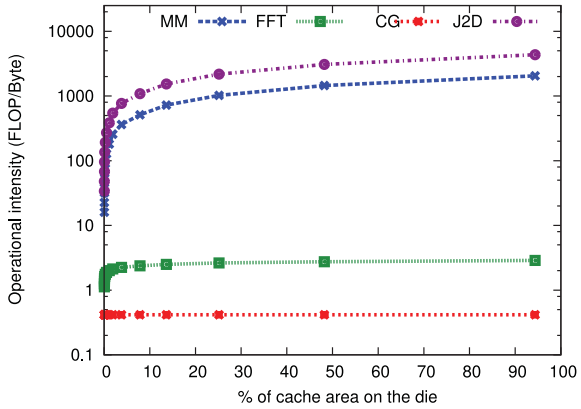


Fig. 4. Operational intensity upper bounds as a function of α.

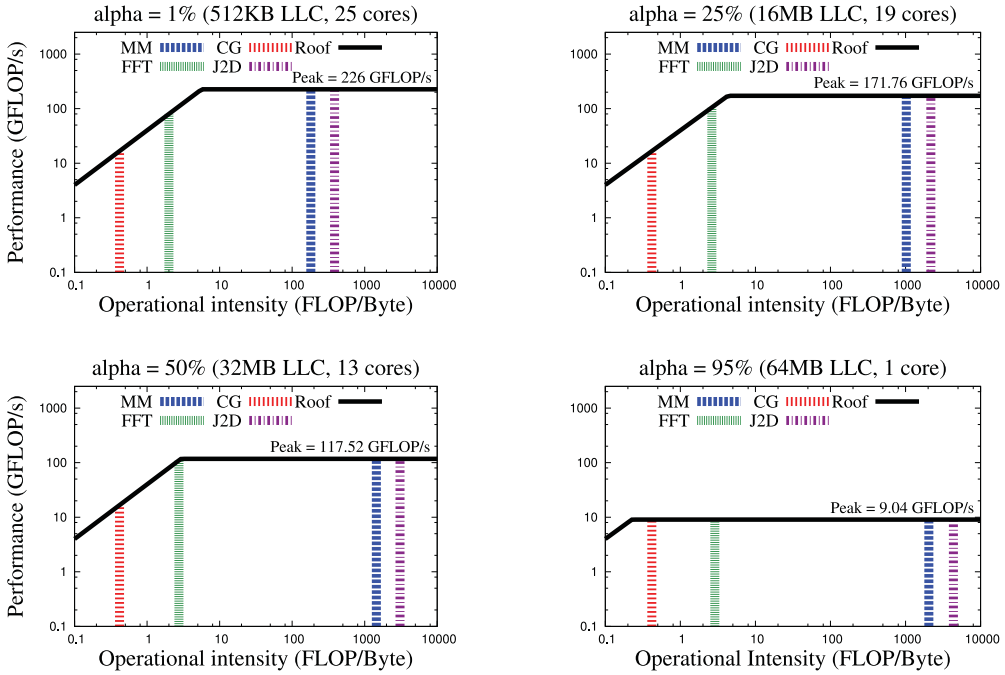


Fig. 5. Upper bounds on performance for different choices of α.

Figure 4 shows the variation of upper bounds on *OI* as a function of fractional chip real estate used for cache, assuming double-precision data occupying 8 bytes per word. It may be seen that the trends for MM and J2D are similar, while those of the remaining two are very different. MM and J2D show a significant increase in *OI* as the fraction of chip area occupied by cache is increased (the plot is logarithmic on the y-axis). FFT shows a very mild rise in *OI* as the cache fraction is increased, and the *OI* is low—between 0.5 and 2 over the entire range. CG has a flat and very low value of *OI* (0.42), irrespective of the amount of cache deployed.

Figure 5 shows four roofline plots for four specific values of α: 0.01, 0.25, 0.5, and 0.95, respectively. In each case, the vertical lines are placed at the upper bound on

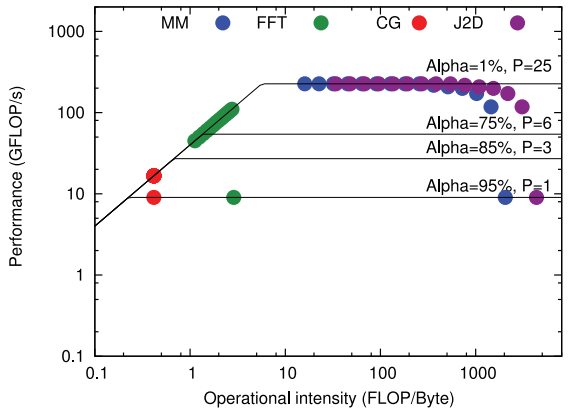


Fig. 6. Upper bounds on performance.

OI for the four algorithms and intersect either the inclined bandwidth roofline or the horizontal processor-peak roofline.

At a very small value of α of 0.01, the size of LLC is very small and a maximal number of cores (25) can be put on the die. So the horizontal roofline is at a performance of 226GFLOP/s (9.04*25). The *OI* values of the four benchmarks—CG, FFT, MM, and J2D—with this configuration are 0.41, 2.0, 181.02, and 384.0, respectively. CG and FFT are bandwidth-bound, although not to the same extent, while MM and J2D are not bandwidth-bound.

When α is increased to 0.25, the number of cores that can be placed on the chip decreases, causing a lowering of the horizontal roofline. The *OI* values increase for FFT, MM, and J2D, while they are unchanged for CG. Compared to α of 0.01, the performance upper bound for FFT increases because the intersection with the bandwidth roofline occurs further to the right. But for MM and J2D, the performance upper bounds decrease despite a higher *OI*, since the horizontal roofline has dropped due to fewer cores. It is interesting to note that the trends as a function of α are in opposite directions for FFT and MM.

Figure 6 shows a single combined roofline plot that captures the variation of upper bounds on performance for the entire design space of configurations of the chip, that is, over the range of possible values of α. The plot represents a consolidated analysis that takes into account the interdependence between the size of LLC and the maximum number of cores—the larger the LLC is, the less the remaining area on the chip for cores. The value of α determines how many cores can be placed on the chip. With the parameters of the chosen technology detailed in the previous section, each core occupies a little under 4% of the chip area. The horizontal rooflines corresponding to four values of α are shown in the figure, intersecting with the bandwidth roofline (corresponding to 40GB/sec). Four instances of horizontal rooflines are shown in the figure, tagged with the value of α and corresponding number of cores.

The results for the three benchmarks—MM, FFT, and CG—show disjoint ranges of *OI*. CG has no variation in *OI* as a function of *S*. Its *OI* of 0.417 leads to an upper bound of 16.7GFLOP/s (0.417*40GB/sec). Since each core has a peak performance of 9.04GFLOP/s for the modeled technology, CG is bandwidth-bound for any number of cores greater than one. But if α is 95%, we can only put one core on the chip, and the upper bound is 9GFLOP/s. Thus, the two red dots in the multiroofline plot capture the entire range of possibilities as alpha is varied: 9GFLOP/s when $P = 1$ (α is above 0.93) and 16.7GFLOP/s when $P \geq 2$.

For FFT, the upper bound on *OI* ranges from 1.125 to 2.875 over the range of α from 1% to 95%. At $\alpha = 1\%$, $OI = 1.125$, and the performance upper bound is 45GFLOPs, and the computation is severely bandwidth bound—the 25 cores that could be put on chip for this value of α would be heavily underutilized. As α is increased, the upper bound on performance improves and hugs the bandwidth roofline. But when α goes above 75% and the number of cores drops below six, the algorithm becomes compute-bound because the peak computational performance is lower than $40 * OI(S)$.

MM has a range (as the size of the LLC is varied) that is always in the compute-bound region of the roofline. But as the LLC size is increased, the number of cores on chip must decrease, and the performance upper bound drops at very high values of *OI*.

Jacobi 2D follows the same trend as that of MM and is always in the compute-bound region. Hence, the performance of J2D for different values of α is exactly equal to that of MM, even though J2D has higher *OI* than MM.

The analysis shows that two currently widely used algorithms, FFT and CG, will not be well suited for solution of very large problems relative to the cache capacity unless the bandwidth between main memory and cache is substantially increased relative to representative parameters of current systems.

Analysis of lower bounds on data movement and upper bounds on *OI* can be similarly carried out for any level of the storage hierarchy—including data movement between cache and registers, or disk and main memory. The bounds for any level can be obtained by appropriately setting the value of "*S*." In this article, we have focused on the data movement between the memory and on-chip LLC to aid in deriving bounds on performance for algorithm-architecture codesign because off-chip memory bandwidth is often a critical performance bottleneck. The presented methodology can, however, be similarly applied to identify potential bottlenecks at other levels of the cache hierarchy.

The lower bounds on data movement and the corresponding upper bounds on *OI* are inherent fundamental limits that cannot be overcome. We note that although the presented analysis has made several simplifying assumptions, the upper bounds on performance can be asserted to hold even when any simplifying assumptions are removed and more accurate information is used. For example, if information about area occupied by interconnects is added, the total number of cores and the total LLC cache size will go down but not increase. Hence, all observations on upper bounds on performance will still be valid. Interconnects may add latencies and also bandwidth constraints, but they will not invalidate any of the earlier conclusions on performance upper bounds for the modeled technology parameters. Similarly, although in reality full overlap of computation and communication is not feasible, and perfect scaling of performance with increase in the number of processors is also generally infeasible, those best-case assumptions do not invalidate any of the conclusions on upper limits of achievable performance over the architectural configuration space. For example, for the modeled architectural parameters, the analysis indicates that achievable performance for large CG and FFT computations (that cannot fit within LLC) will necessarily be far below system peak, irrespective of the amount of effort put into the implementation, because fundamental inherent data movement limits of those computations cannot be overcome.

6. ALGORITHM-ARCHITECTURE CODESIGN: BOUNDS ON ENERGY EFFICIENCY

An important metric is energy efficiency, defined as the ratio of number of executed operations to the energy expended in the execution. The upper bounds on *OI* can also be used to bound the maximum achievable energy efficiency. The total energy of execution is modeled as the sum of energy for performing the operations, the energy for data movement from DRAM access, and an additional component for the static leakage energy in the cores and cache. Figure 7 shows the variation in the upper bounds on

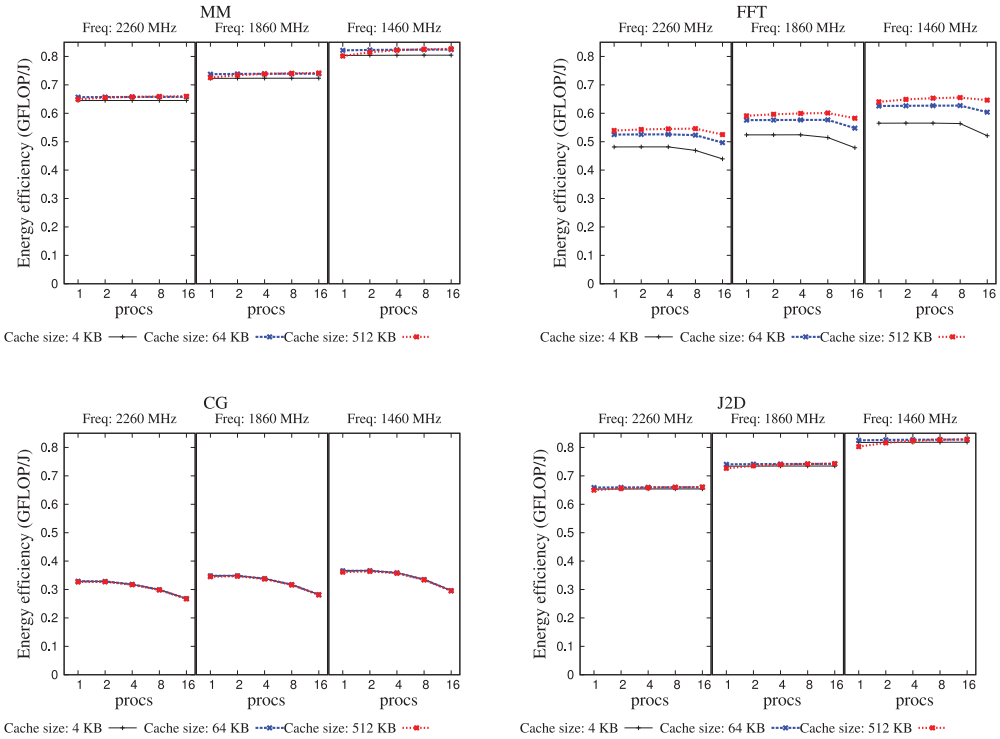


Fig. 7. Upper bounds on energy efficiency.

energy efficiency for MM, FFT, CG, and J2D as a function of (1) number of cores used, (2) the voltage and clock frequency used, and (3) the capacity of LLC.

The horizontal axis marks three different clock frequencies (2,260MHz, 1,860MHz, and 1,460MHz), representing voltage/frequency scaling, and for each of the frequencies, five choices of processor count (one, two, four, eight, and 16). Different curves correspond to different capacities of the LLC. The overall trends are as follows:

(1) For all four benchmarks, the groups of clustered lines move upward as we go from left to right on the charts, representing a decrease in the voltage/frequency. For a fixed number of cores and LLC capacity, lower frequencies lead to higher bounds on attainable energy efficiency. This is because there is a nonlinear decrease in the core's energy per operation (energy is proportional to $V^2 f$, and voltage and frequency are linearly related, so that energy is proportional to f^3) as voltage/frequency are decreased. There is also an increase in the total static leakage energy since the computation will take longer to complete, but there is an overall reduction in the lower bounds for energy, or an increase in the upper bounds on energy efficiency.

(2) Increasing the number of cores (with fixed frequency and LLC) is detrimental to energy efficiency, especially for bandwidth-bound computations. This is seen clearly for FFT and CG, with each of the lines curving downward as the number of processors is increased. This effect is mainly due to the increased static energy for the active cores. While additional cores can divide the parallel work among themselves, the computation rate is limited by the rate at which data is delivered from memory to cache, so that there is no reduction in the lower bound for execution

time. For MM and J2D, the curves are relatively flat since the computation is compute-bound. Using more cores enables the work to be done faster, and there is no increase in the total static leakage energy aggregated over all cores: using twice as many cores halves the lower bound on execution time and doubles the static leakage power of the cores.

(3) Increasing LLC capacity has two complementary effects: (i) potential for improving energy efficiency by increasing *OI* due to the larger cache, but (ii) decreased energy efficiency due to higher static leakage energy from the larger cache.

There is no useful benefit for CG since its *OI* is independent of cache capacity. At larger cache sizes, there is a detrimental effect (not seen in the charts since the cache sizes used were quite small). For FFT, the benefits from improved *OI* clearly outweigh the increased static leakage energy. For MM and J2D, although *OI* increases with cache size, it is already so high at the smallest cache size that the incremental benefits in reducing data transfer energy from main memory are very small. Hence, the curves representing different cache sizes for fixed frequency do not have much separation.

We next characterize the maximal possible energy efficiency for the four benchmarks, if we have the flexibility to choose (1) the number of cores to be turned on, (2) the amount of cache area to be used, and (3) voltage/frequency at which the cores are to be run. From Equations (1) and (2), we have the energy efficiency

$$E_{eff} = \frac{W}{E} = \left(\epsilon_{flop}(f) + \frac{\epsilon_{mem}}{OI(S)} + \frac{P.\pi_{cpu} + \pi_{cache}(S)}{\min\left(P.F_{flops}(f),\, B_{mem}.OI(S)\right)} \right)^{-1}. \tag{3}$$

Depending on the upper bound on the *OI* of the algorithms, we analyze different cases next and determine what is the best configuration for the architecture to obtain maximum energy efficiency.

Case I: *The algorithm is completely bandwidth-bound.*

This corresponds to the case where the maximal $OI(S)$ of the algorithm for a given cache size S is too low that it is bandwidth bound even on a single core at its lowest frequency. We can see from the performance roofline viewpoint that this leads to the condition $B_{mem}.OI(S) < F_{flops}(f)$. With increasing frequency, $\epsilon_{flop}(f)$ increases and thus the energy efficiency deteriorates. Hence, the highest energy efficiency is achieved when $P = 1$ and f is set at its minimum, and Equation (3) reduces to $E_{eff} = (\epsilon_{flop}(f_{min}) + \frac{\epsilon_{mem}}{OI(S)} + \frac{\pi_{cpu}+\pi_{cache}(S)}{B_{mem}.OI(S)})^{-1}$, where f_{min} is the minimum allowable frequency for the architecture.

Case II: *The algorithm is compute-bound with* p *or fewer cores and at all frequencies.*

From Equation (3), we can see that at any given frequency f, increasing P improves the E_{eff}. Hence, $P = \mathsf{p}$ is the best choice irrespective of the frequency. Also, with increasing f, $F_{flops}(f)$ increases linearly, while $\epsilon_{flop}(f)$ increases superlinearly. For a fixed value of p, the optimal energy efficiency is dependent on the machine parameters like $\pi_{cache}(S)$ and $\epsilon_{flop}(f)$. The maximal energy efficiency is obtained, $E_{eff} = \max_{f\in[f_{min},f_{max}]}(\epsilon_{flop}(f) + \frac{\epsilon_{mem}}{OI(S)} + \frac{\mathsf{p}.\pi_{cpu}+\pi_{cache}(S)}{\mathsf{p}.F_{flops}(f)})^{-1}$, where f_{min} and f_{max} are the minimum and maximum allowable frequencies for the architecture, respectively.

Case III: *The algorithm is compute-bound with* p *cores at lower frequencies and becomes bandwidth-bound with* p *cores at higher frequencies.*

Let f_{cutoff} be the frequency where the algorithm transitions from a compute-bound to a memory-bound region. For the region where $f \geq f_{cutoff}$, from case I, we know that once the algorithm becomes bandwidth-bound, increasing the frequency further

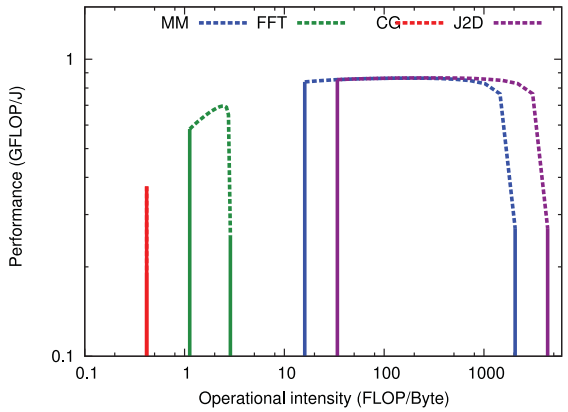


Fig. 8. Upper bounds on energy efficiency.

has a detrimental effect on the energy efficiency. Hence, the best energy efficiency is achieved when $f = f_{cutoff}$, where $\mathsf{p}.F_{flops}(f_{cutoff}) = B_{mem}.OI(S)$. When $f < f_{cutoff}$, analysis in case II showed that in the compute-bound region when the number of cores, p, is held constant, optimal frequency depends on the machine parameters. Also, we have $\mathsf{p}.F_{flops}(f) < B_{mem}.OI(S)$. Hence, $E_{eff} = \max_{f\in[f_{min},f_{cutoff}]}(\epsilon_{flop}(f) + \frac{\epsilon_{mem}}{OI(S)} + \frac{\mathsf{p}.\pi_{cpu}+\pi_{cache}(S)}{\mathsf{p}.F_{flops}(f)})^{-1}$.

Case IV: *The algorithm is compute-bound at all frequencies with* p *cores and becomes bandwidth-bound with* $\mathsf{q} = \mathsf{p} + 1$ *or a higher number of cores.*

This case gives rise to two scenarios: (1) Performance at higher frequencies ($f_p \in [f_1, f_{max}]$) with p cores overlaps with the performance at lower frequencies ($f_q \in [f_{min}, f_2]$) with q cores, and hence, the algorithm becomes bandwidth-bound at frequencies $f > f_2$ and q cores. (2) There is a gap between maximum performance achievable with p cores and minimum performance achieved with q cores, that is, $\left(\mathsf{q}.F_{flops}(f_{min}) - \mathsf{p}.F_{flops}(f_{max})\right) > 0$. In both these scenarios, the maximum achievable energy efficiency depends on the machine parameters.

Similar to the performance roofline in Figure 6, an energy multiroofline has been plotted for the four benchmarks in Figure 8 with a range of *OI* corresponding to the range of LLC sizes for our testbed architecture. Unlike the case for the performance multiroofline, the actual set of energy roofline curves are not shown in the figure because there are too many of them and they would clutter the graph; in addition to different values of α, we also consider different core frequencies, resulting in a different energy roofline for each combination.

Figure 8 shows a similar trend for the energy efficiency as that of the performance in Section 5. MM and J2D are always compute-bound and hence fall under case II for all values of S. For MM and J2D, optimal energy efficiency was generally achieved at the lowest frequency value on the testbed. But, as the number of active cores approached one (as the size of LLC increases), maximum energy efficiency was achieved at higher frequencies. On the other hand, CG becomes bandwidth bound as the number of active cores for the given LLC size exceeds three. Hence, CG starts at case IV with $P = 3$, and as the LLC size increases (and the number of allowable cores goes below three), it enters into case II. CG has the best upper bound on energy efficiency of 0.373GFLOP/J for the cache size of 4KB with three cores at 1.26GHz. The upper bound on energy efficiency for FFT initially increases with increasing cache size and then starts decreasing for similar reasons as that of the performance upper bounds. FFT achieves its maximum energy efficiency for the cache size of 8MB at $P = 10$ and $f = 1.26$GHz.

7. RELATED WORK

Williams et al. [2009] developed the roofline model that attempts to analyze bottlenecks in performance of an architecture due to memory bandwidth limits. Choi et al. [2013] and Choi et al. [2014] developed the energy version of the roofline model. Czechowski et al. [2011] developed balance principles for algorithm-architecture codesign. These models characterize algorithms using operational intensity, which, however, is not a fixed quantity for an algorithm-architecture combination. Our work complements the previous efforts. Lipshitz et al. [2013] considered the energy efficiency at the algorithmic level and proved that a region of perfect strong scaling in energy exists for matrix multiplication and the direct n-body problem. They also address the problem of finding the minimum energy required for a computation given a maximum allowed runtime and vice versa.

Several efforts have focused on developing I/O lower bounds, which is equivalent to the problem of finding upper bounds on operational intensity. Hong and Kung [1981] provided the first characterization of the I/O complexity problem using the red/blue pebble game and the equivalence to 2S-partitioning of CDAGs. Several works followed Hong and Kung's work on I/O complexity in deriving lower bounds on data accesses [Aggarwal and Vitter 1988; Aggarwal et al. 1987; Irony et al. 2004; Bilardi et al. 2000; Bilardi and Peserico 2001; Savage 1995, 1998; Ranjan et al. 2011, 2012; Valiant 2011; Demmel et al. 2012; Ballard et al. 2011, 2012; Christ et al. 2013; Solomonik et al. 2013; Savage and Zubair 2010]. Aggarwal and Vitter [1988] provided several lower bounds for sorting algorithms. Savage [1995, 1998] developed the notion of S-span to derive Hong-Kung style lower bounds, and that model has been used in several works [Ranjan et al. 2011, 2012; Savage and Zubair 2010]. Irony et al. [2004] provided a new proof of the Hong-Kung result on I/O complexity of matrix multiplication and developed lower bounds on communication for sequential and parallel matrix multiplication. Ballard et al. [2011, 2012], Demmel et al. [2012], and Solomonik et al. [2013] have developed lower bounds as well as optimal algorithms for several linear algebra computations including QR and LU decomposition and the all-pairs shortest-paths problem. Bilardi et al. [2000] and Bilardi and Peserico [2001] developed the notion of access complexity and related it to space complexity. Also, Bilardi et al. [2005] developed an analysis of the tradeoffs between bandwidth, memory, and processing for QCD computation and derived guidelines to design a machine for QCD computation.

Extending the scope of the Hong and Kung model to more complex memory hierarchies has also been the subject of research. Savage [1995] provided an extension together with results for some classes of computations that were considered by Hong and Kung, providing optimal lower bounds for I/O with memory hierarchies. Valiant [2011] proposed a hierarchical computational model that offers the possibility to reason in an arbitrarily complex parameterized memory hierarchy model. In a recent paper, we [Elango et al. 2014] extended the parallel model for shared-memory architectures by Savage and Zubair [2008] to also include the distributed-memory parallelism present in all scalable parallel architectures. The works of Irony et al. [2004] and Ballard et al. [2011] modeled communication across nodes of a distributed-memory system. Bilardi and Preparata [1999] developed lower-bound results for communication in a distributed-memory model specialized for multidimensional mesh topologies.

8. CONCLUSION

The roofline model is very useful in depicting bounds on time efficiency (operations per second) and energy efficiency (operations per joule) of a computation on a machine as a function of operation intensity, the ratio of computational operations per byte of data moved from/to memory. While operational intensity can be measured for a given

implementation of an algorithm, it is not a fixed quantity. It is a function of cache capacity and also depends on the schedule of operations; that is, it is affected by semantics-preserving code transformations. Therefore, understanding fundamental performance bottlenecks for an algorithm generally requires analysis of many alternatives.

In this article, we have proposed an approach to use upper bounds on operational intensity, derived from schedule-independent lower bounds on data movement, in order to enable effective bottleneck analysis using the roofline model. We have used the approach to model upper bounds on performance and energy efficiency across an architectural design space considering different voltage/frequency scaling and different fractions of die area being allocated to last-level cache versus cores.

ACKNOWLEDGMENTS

We thank the anonymous referees for the feedback and many suggestions that helped us in improving the presentation of the work. This work was supported in part by the U.S. National Science Foundation through awards 0811457, 0926127, 0926687, and 1059417; by the U.S. Department of Energy through award DE-SC0012489; and by Louisiana State University.

REFERENCES

Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. 1987. A model for hierarchical memory. In *Proceedings of the 19th STOC*. 305–314.

Alok Aggarwal and Jeffrey S. Vitter. 1988. The input/output complexity of sorting and related problems. *Commun. ACM* 31, 9 (1988), 1116–1127.

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2011. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.* 32, 3 (2011), 866–901.

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2012. Graph expansion and communication costs of fast matrix multiplication. *J. ACM* 59, 6 (2012), 32.

Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. 2008. *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*. Tech. Rep., Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO).

Gianfranco Bilardi and Enoch Peserico. 2001. A characterization of temporal locality and its portability across memory hierarchies. In *Proceedings of the 28th International Colloquium on Automata Languages and Programming.* 128–139.

Gianfranco Bilardi, Andrea Pietracaprina, and Paolo D'Alberto. 2000. On the space and access complexity of computation DAGs. In *Graph-Theoretic Concepts in Computer Science*. 81–92.

Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, Fabio Schifano, and Raffaele Tripiccione. 2005. The potential of on-chip multiprocessing for QCD machines. In *High Performance Computing HiPC 2005*, D. Bader, M. Parashar, V. Sridhar, and V. Prasanna (Eds.). Lecture Notes in Computer Science, Vol. 3769. Springer, Berlin, 386–397. http://dx.doi.org/10.1007/11602569_41

Gianfranco Bilardi and Franco P. Preparata. 1999. Processor - time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory Comput. Syst.* 32, 5 (1999), 531–559.

Gianfranco Bilardi, Michele Scquizzato, and Francesco Silvestri. 2012. A lower bound technique for communication on BSP with application to the FFT. In *Euro-Par*. 676–687.

Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. 2013. A roofline model of energy. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. 661–672.

Jee Whan Choi, Marat Dukhan, Xing Liu, and Richard Vuduc. 2014. Algorithmic time, energy, and power on candidate HPC compute building blocks. In *Proceedings of the 2014 IEEE 28th International Symposium on Parallel and Distributed Processing (IPDPS'14)*. 1–11.

Michael Christ, James Demmel, Nicholas Knight, Thomas Scanlon, and Katherine Yelick. 2013. *Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays Part 1*. EECS Technical Report EECS–2013-61, University of California, Berkeley.

Stephen A. Cook. 1974. An observation on time-storage trade off. *J. Comput. Syst. Sci.* 9, 3 (1974), 308–316.

Kent Czechowski, Casey Battaglino, Chris McClanahan, Aparna Chandramowlishwaran, and Richard Vuduc. 2011. Balance principles for algorithm-architecture co-design. In *Proceedings of the 3rd USENIX Conference on Hot topics in Parallelism (HotPar'11)*. 1–5.

James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2012. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comput.* 34, 1 (2012).

Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, J. Ramanujam, and P. Sadayappan. 2013. *Data Access Complexity: The Red/Blue Pebble Game Revisited*. Technical Report. OSU/INRIA/LSU/UCLA. OSU-CISRC-7/13-TR16.

Venmugil Elango, Fabrice Rastello, Louis-Noël Pouchet, J. Ramanujam, and P. Sadayappan. 2014. On characterizing the data movement complexity of computational DAGs for parallel execution. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 296–306.

Samuel H. Fuller and Lynette I. Millett. 2011. *The Future of Computing Performance: Game Over or Next Level?* National Academies Press. Retrieved from http://www.nap.edu/openbook.php?record_id=12980.

Magnus Rudolph Hestenes and Eduard Stiefel. 1952. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.* 49, 6, 2379.

Jia-Wei Hong and H. T. Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)*. ACM, 326–333.

Dror Irony, Sivan Toledo, and Alexandre Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.* 64, 9 (2004), 1017–1026.

Benjamin Lipshitz, James Demmel, Andrew Gearhart, and Oded Schwartz. 2013. Perfect strong scaling using no additional energy. In *2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*. 649–660. http://dx.doi.org/10.1109/IPDPS.2013.32

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*. 469–480.

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. *CACTI 6.0: A Tool to Model Large Caches*. Technical Report HPL-2009-85. HP Labs.

Desh Ranjan, John Savage, and Mohammad Zubair. 2011. Strong I/O lower bounds for binomial and FFT computation graphs. In *Computing and Combinatorics*. LNCS, Vol. 6842. Springer, 134–145.

Desh Ranjan, John E. Savage, and Mohammad Zubair. 2012. Upper and lower I/O bounds for pebbling r-pyramids. *J. Discrete Algorithms* 14 (2012), 2–12.

Desh Ranjan and Mohammad Zubair. 2012. Vertex isoperimetric parameter of a Computation Graph. *Int. J. Found. Comput. Sci.* 23, 4 (2012), 941–964.

Stefan Rusu, Simon Tam, Harry Muljono, Jason Stinson, David Ayers, Jonathan Chang, Raj Varada, Matt Ratta, and Sailesh Kottapalli. 2009. A 45nm 8-core enterprise Xeon processor. In *IEEE Asian Solid-State Circuits Conference, 2009. A-SSCC 2009.* 9–12.

John E. Savage. 1995. Extending the Hong-Kung model to memory hierarchies. In *Computing and Combinatorics*. LNCS, Vol. 959. 270–281.

John E. Savage. 1998. *Models of Computation*. Addison-Wesley.

John E. Savage and Mohammad Zubair. 2008. A unified model for multicore architectures. In *Proceedings of the 1st International Forum on Next-Generation Multicore/Manycore Technologies*. ACM, 9.

John E. Savage and Mohammad Zubair. 2010. Cache-optimal algorithms for option pricing. *ACM Trans. Math. Softw.* 37, 1 (2010).

Michele Scquizzato and Francesco Silvestri. 2013. Communication lower bounds for distributed-memory computations. *CoRR* abs/1307.1805 (2013).

John Shalf, Sudip Dosanjh, and John Morrison. 2011. Exascale computing technology challenges. *High Performance Computing for Computational Science–VECPAR 2010* (2011), 1–25.

Edgar Solomonik, Aydin Buluc, and James Demmel. 2013. Minimizing communication in all-pairs shortest paths. In *IPDPS*.

Leslie G. Valiant. 2011. A bridging model for multi-core computing. *J. Comput. Syst. Sci.* 77, 1 (Jan. 2011), 154–166.

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76.

Received June 2014; revised November 2014; accepted November 2014