The roofline model is a popular approach to “bounds and bottleneck” performance analysis. It focuses on the limits to 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 paper, we alleviate 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 using 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.

1. INTRODUCTION

Technology trends have resulted in widely different rates of improvement in computational throughput as compared to data movement rates in computer systems. With future systems, the cost of data movement through the memory hierarchy is expected 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], both in terms of 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 “bounds 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 Fig. 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 sufficiently high OI, greater than a certain “critical intensity” corresponding to the point of intersection of the two rooflines, is essential for a code to avoid being memory-bandwidth limited.

Modeled or measured data volume between main-memory and last-level 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 LLC capacity, possibly on 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 number of alternative implementation scenarios.
In this paper, we present a different approach to use of the roofline model by using upper bounding techniques on OI and thereby avoiding the need to explicitly model and analyze OI over a large number of usage scenarios. We then use this upper-bound based characterization in a roofline model approach to capture the upper limits of performance of an algorithm over a range of possible alternative architectural configurations for a given VLSI technology. This paper makes the following contributions:

— It develops techniques to bound the maximum possible OI of a computation as a function of cache size. This bound applies to all possible valid schedules for the operations constituting the computation, i.e., the bound is a fundamental upper limit for the OI, irrespective of any optimization/ transformation like loop tiling, fusion etc.

— It uses the modeling of upper bounds on OI to enable the use of the roofline model in algorithm-architecture co-design 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.

2. It models maximal achievable energy efficiency (operations per Joule) for different algorithms on a given VLSI technology, considering variation in number of cores, cache capacity, and voltage/frequency scaling.

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. 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 a positive slope corresponding to the peak memory-to-LLC bandwidth. This is shown by the black lines in Fig. 1. The horizontal axis represents 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. At the point of intersection of the two lines, the OI corresponds to the ratio of peak computational performance of the core(s) to the peak memory bandwidth. 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

Fig. 1: Roofline model: Performance and Energy rooflines
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.

The roofline model has recently been adapted [Choi et al. 2013; Choi et al. 2014] to capture upper bounds on energy efficiency, i.e., 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 per-operation energy cost for performing a set of $N$ operations is the sum of the energy cost for actually performing the $N$ arithmetic operations and the energy cost for $\frac{D}{W}$ memory operations. Fig. 1 shows an energy roofline (in blue) for the same architecture as the performance roofline (pair of intersecting lines in black). 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, e.g., 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, an algorithm can be abstracted by a computational directed acyclic graph (CDAG), and different semantically equivalent implementations of the algorithm (for example, 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 paper, 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 minimal number of data elements that must be moved between main memory and cache in order to execute all operations of an algorithm. Given an execution of a computation that executes $W$ operations and performs $D$ bytes of data transfer to/from main memory during execution, the OI is $\frac{W}{D}$. 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 Pares 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). In this section, we provide some background on prior work on I/O lower bounds problem, followed by a presentation of a new approach we have developed to address this problem (Section 3). We then use the I/O lower bounding techniques to form upper bounds on OI, which enables multi-roofline based
Consider the code shown in Fig. 2(a). Its computational complexity can be simply stated as \((N-2)^2\) arithmetic operations. Fig. 2(b) shows a functionally equivalent form of the same computation, after a tiling transformation. The tiled form too has exactly the same computational complexity of \((N-2)^2\) arithmetic operations. 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 (Fig. 2(a)) than the tiled version (Fig. 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 computational complexity of an algorithm, which stays unchanged for different valid orders of execution of its operations and is also independent of machine parameters like cache size, the data access cost depends both on the cache capacity and 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 computational directed acyclic graph (CDAG), with a vertex for each instance of each computational operation, and edges from producer instances to consumer instances. Fig. 2(c) shows the CDAG for the codes in Fig. 2(a) and Fig. 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).

### 2.3. The Red-Blue Pebble Game

The seminal work of Hong and Kung [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. We use the notation of Bilardi & Peserico [Bilardi and Peserico 2001] to describe the CDAG model used by Hong & Kung:

**Definition 2.1 (CDAG).** A computational directed acyclic graph (CDAG) is a 4-tuple \(C = (I,V,E,O)\) of finite sets such that: (1) \(I \subseteq V\) 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) \(V \setminus I\) is
called the operation set and all its vertices have one or more incoming edges; (5) \( O \subseteq V \) is called the output set.

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 below. 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, i.e., 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 & Kung developed a novel approach for deriving I/O lower bounds for CDAGs by relating the red-blue pebble game to a graph partitioning problem defined as follows.

**Definition 2.3** (Hong & Kung \( S \)-partitioning of a CDAG [Hong and Kung 1981]). Let \( C = (I, V, E, O) \) be a CDAG. An \( S \)-partitioning of \( C \) is a collection of \( h \) subsets of \( V \) such that:

- **P1.** \( \forall i \neq j, V_i \cap V_j = \emptyset \), and \( \bigcup_{i=1}^{h} V_i = V \)
- **P2.** there is no cyclic dependence between subsets
- **P3.** \( \forall i, \exists D \in \text{Dom}(V_i) \) such that \( |D| \leq S \)
- **P4.** \( \forall i, |\text{Min}(V_i)| \leq S \)

where a dominator set of \( V_i, D \in \text{Dom}(V_i) \) is a set of vertices such that any path from \( I \) to a vertex in \( V_i \) contains some vertex in \( D \); the minimum set of \( V_i, \text{Min}(V_i) \) is the set of vertices in \( V_i \) that have all its successors outside of \( V_i \); and for a set \( A, |A| \) is the cardinality of the set \( A \).

Hong & Kung showed a construction for a \( 2S \)-partition of a CDAG, corresponding to any complete calculation on that CDAG using \( S \) red pebbles, with a tight relationship between the number of vertex sets \( h \) in the \( 2S \)-partition and the number of I/O moves \( q \) in the complete calculation, as shown in Theorem 2.4. The tight association theorem between any complete calculation and a corresponding \( 2S \)-partition provides the key Lemma 2.5 that serves as the basis for Hong & Kung’s approach to deriving lower bounds on the I/O complexity of CDAGs typically by reasoning on the maximal number of vertices that could belong to any vertex-set in a valid \( 2S \)-partition.

**Theorem 2.4** (Pebble Game, I/O and 2S-Partition [Hong and Kung 1981]). Any complete calculation of the red-blue pebble game on a CDAG using at most \( S \) red pebbles is
associated with a 2S-partition of the CDAG such that \( S \geq q \geq S (h - 1) \), where \( q \) is the number of I/O moves in the complete calculation and \( h \) is the number of subsets in the 2S-partition.

**Lemma 2.5 (Lower Bound on I/O [Hong and Kung 1981]).** Let \( H(2S) \) be the minimal number of vertex sets for any valid 2S-partition of a given CDAG (such that any vertex with no incoming – resp. outgoing – edge is an element of \( I \) – resp. \( O \)). Then the minimal number \( Q \) of I/O operations for any complete calculation on the CDAG is bounded by: \( Q \geq S \times (H(2S) - 1) \). This key lemma has been useful in proving I/O lower bounds for several CDAGs [Hong and Kung 1981] by reasoning about the maximal number of vertices that could belong to any vertex-set in a valid 2S-partition.

### 2.4. 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 collaborators [Savage and Zubair 2008] and Elango et al. [Elango et al. 2014]. We describe the Multicore Memory Hierarchy Game (MMHG) below. [Savage and Zubair 2008].

**Parallel Machine Model:** The abstraction of a parallel computer is shown in Fig. 3. The parallel computer has a number of cores and a hierarchy of storage elements: a set of private registers (at level 1) for each core, a private L1 cache per core (at level 2), and a hierarchy of zero or more additional levels of cache (through level \( L - 1 \)), and a shared main memory (at level \( L \)). The total number of storage entities at level \( l \) is denoted \( N_l \), and the capacity of each entity at level \( l \) is \( S_l \) words. The hierarchical structure means that each storage entity at level \( l \) is connected to a unique storage entity at level \( l + 1 \), and an integral multiple (usually a power of 2) of entities at level \( l - 1 \).

![Fig. 3: Model of a multicore parallel system](image)

**Parallel Red-Blue Pebble Game:** The Multiprocessor Memory Hierarchy Game (MMHG) game proposed by Savage and Zubair [Savage and Zubair 2008], models data movement in a shared storage hierarchy with a common shared level of memory that can be directly accessed by all processors. With the MMHG game, a different set of red pebbles is associated with each storage entity in the parallel system – we can consider there to be different shades of red, one per distinct storage entity. Associated with the storage entities at a level \( l \) in the hierarchy, we have \( N_l \) distinct shades of red pebbles, each associated with one of the \( N_l \) distinct storage entities in the system at that level. The rules of MMHG are stated below, and encode the constraints on movement of data.

**Definition 2.6 (Multiprocessor Memory Hierarchy Game (MMHG)).** Let \( C = (I, V, E, O) \) be a CDAG. Given for each level \( 1 \leq l \leq L \), \( N_l \times S_l \) number of red pebbles of different shades \( R^1_l, R^2_l, \ldots, R^{N_l}_l \), respectively, and unlimited blue pebbles, with a blue pebble on each input vertex, a complete calculation is any sequence of steps using the following rules that results in a final state with blue pebbles on all output vertices:

**R1 (Input)** A level-\( L \) pebble, \( R^i_l \), can be placed on any vertex that has a blue pebble.

**R2 (Output)** A blue pebble can be placed on any vertex that has a level-\( L \) pebble on it.
R3 **(Move up)** For \(1 \leq l < L\), a level-\(l\) red pebble, \(R^l_i\) can be placed on any vertex that has a level-\(l+1\) pebble \(R^l_{i+1}\) where \(R^l_{i+1}\) is in a cache that is a child of the cache that holds \(R^l_i\).

R4 **(Move down)** For \(1 < l \leq L\), a level-\(l\) red pebble, \(R^l_i\) can be placed on any vertex that has a level-\((l-1)\) pebble \(R^{l-1}_j\) where \(R^{l-1}_j\) is in a cache that is a child of the cache that holds \(R^l_i\).

R5 **(Compute)** If all the immediate predecessors of a vertex \(v\) have level-\(1\) red pebbles of shade \(p\) on them, then a level-\(1\) red pebble \(R^1_p\) may be placed on \(v\); here \(p\) is the index of the processor that computes vertex \(v\).

R6 **(Delete)** Any shade of red pebble may be removed from any vertex (reuse storage).

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 I/O using the MMHG model can only lead to lower or equal value for I/O 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 I/O 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 paper 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.

### 3. CONVEX MIN-CUT BASED I/O LOWER BOUND

In this section, we describe an alternative I/O lower bounding approach [Elango et al. 2013] that we use in developing lower bounds on data movement, and thereby upper bounds on I/O. It is motivated from the observation that the Hong & 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, we develop an approach that captures internal space requirements using the abstraction of wavefronts. The central idea is the definition of two kinds of wavefronts and a relation between them.

**Definitions:** We first present needed definitions. Given a graph \(G = (V, E)\), a cut is defined as any partition of the set of vertices \(V\) into two parts \(S\) and \(T = V - S\). An \(s - t\) cut is defined with respect to two distinguished vertices \(s\) and \(t\) and is any \((S, T)\) cut satisfying the requirement that \(s \in S\) and \(t \in T\). Each cut defines a set of cut edges (the cut-set), i.e., the set of edges \((u, v)\) where \(u \in S\) and \(v \in T\). The capacity of a cut is defined as the sum of the weights of the cut edges. The minimum cut problem (or min-cut) is one of finding a cut that minimizes the capacity of the cut. We define vertex \(u\) as a cut vertex with respect to an \((S, T)\) cut, as a vertex \(u \in S\) that has a cut edge incident on it. A related problem of interest for this paper is the vertex min-cut problem which is one of finding a cut that minimizes the number of cut vertices.

Given a DAG \(G = (V, E)\) and some vertex \(x \in V\), the set \(\text{Anc}(x)\) is the set of vertices from which there is a non-empty directed path to \(x\) in \(G\) \((x \notin \text{Desc}(x))\); the set \(\text{Desc}(x)\) is the set of vertices to which there is a non-empty directed path from \(x\) in \(G\) \((x \notin \text{Anc}(x))\). Using those two notions, we consider a convex cut \((S_x, T_x)\) associated to \(x\) as follows: \(S_x\) includes \(x \cup \text{Anc}(x)\); \(T_x\) includes \(\text{Desc}(x)\); in addition, \(S_x\) and \(T_x\) must be constructed such that there is no edge from \(T_x\) to \(S_x\). With this, the sets \(S_x\) and \(T_x\) partition the graph \(G\) into two convex partitions. We define the wavefront induced by \((S_x, T_x)\) to be the set of vertices in \(S_x\) that have at least one outgoing edge to a vertex in \(T_x\).

**Schedule wavefront:** In any complete calculation \(P\) of the red-blue pebble game on a CDAG \(C\), consider the step that applies the firing rule R3 to compute vertex \(x\). Associated with each vertex \(x\) of \(V\), we identify a schedule wavefront \(W_P(x)\) to be the set of vertices of \(V\) that have already fired before \(x\), but have at least one successor that has not yet been computed. \(W_P(x)\) thus represents the set of live values in the computation at the time that \(x\) is computed.
Graph min-cut wavefront: A convex cut of $C$ is a partition of its vertices into two sets $S$ and $T$ such that there is no edge from $T$ to $S$. The cut-vertex-set is $\text{Min}(S)$, i.e., vertices of $S$ that have a successor in $T$. We define a graph min-cut wavefront $W_G^\text{min}(x)$ to be a constrained cut-vertex-set with respect to a vertex $x \in V$, as a minimum cardinality cut-vertex-set with an additional constraint that all ancestors of $x$ fall in the partition $S$ and all descendants of $x$ fall in the partition $T$. We define $w_G^\text{max} = \max_{x \in V} (|W_G^\text{min}(x)|)$.

The key idea behind the approach is that for any vertex $x$, the schedule wavefront $W_P(x)$ must have at least as many vertices as $W_G^\text{min}(x)$ and that the size of any schedule wavefront imposes a lower bound on I/O due to the limited number of red pebbles. Consider the vertex $x$ in the Diamond DAG shown in Fig. 4. A graph min-cut wavefront $W_G^\text{min}(x)$ is shown by the dashed line that includes $x$ and five other vertices (this wavefront is not unique, but all min-cut wavefronts for $x$ will include six vertices). The relationship between schedule wavefronts and graph min-cut wavefronts means that in any complete calculation, at the time $x$ is fired, there must be at least six live values. If the number of red pebbles $S$ is less than six, $6 - S$ of these live vertices in the CDAG must necessarily have been spilled (no re-pebbling is permitted in the pebble game model we use), representing a corresponding minimal amount of I/O. A proof of the following lemma, and further details of the min-cut based lower bounding approach are provided in a technical report [Elango et al. 2013].

**Lemma 3.1.** Let $C = (\emptyset, V, E, O)$ be a CDAG with no inputs. For any $x \in V$, $2 (|W_G^\text{min}(x)| - S) \leq IO(C)$. In particular, $2 (w_G^\text{max} - S) \leq IO(C)$.

Tighter Bounds via Partitioning: If applied to the whole CDAG, Lemma 3.1 will usually lead to a very weak bound. To overcome this limitation, the idea is to decompose it into smaller sub CDAGs, and sum up their individual I/Os. The following theorem formalizes this approach and its proof is provided in a technical report [Elango et al. 2013].

**Theorem 3.2** (Min-Cut with Divide and Conquer). Let $C = (I, V, E, O)$ be a CDAG. Let $V_1, \ldots, V_p$ be a (not necessarily acyclic) disjoint partitioning of $V$, and $C_1, \ldots, C_p$ be the induced partitioning of $C$ ($I_i = V_i \cap I$, $E_i = E \cap V_i \times V_i$, $O_i = O \cap V_i$). Let for each $i$, $C'_i = (\emptyset, V'_i, E'_i, O')$ be the sub-DAG obtained from $C_i$ by deleting all input and output vertices ($V'_i = V_i - I_i - O_i$, $E'_i = E_i \cap V'_i \times V'_i$, and $G'_i = (V'_i, E'_i)$). Then the minimum I/O of $C$ can be bounded by:

\[
\sum_{i=1}^{p} 2 (w_{G'_i}^\text{max} - S) + |I| + |O| \leq IO(C)
\]

This theorem is key to enable the decomposition of a CDAG and reason on its components individually. As this article focuses on implementation-independent roofline analysis, we show for illustration purpose only in Appendix A how the I/O lower bounds are derived for the three algorithms we use in the next sections.
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 (Sec. 5)?

— For a given set of technology parameters, what are the upper bounds on energy efficiency (Giga-operations per Joule) for different algorithms (Sec. 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 last level of cache (LLC) in this analysis. Also, we do not 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 GFLOPs aggregate performance for a multi-core 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, etc., may serve to tighten or reduce the upper bound to a lower value than 60 GFLOPs, but cannot invalidate the conclusions drawn from the more simplified analysis.

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 $\alpha$ trades off one for the other. $f_{\text{flops}}(f)$ varies with the clock frequency $f$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Total chip area</td>
</tr>
<tr>
<td>$\alpha.A$</td>
<td>Area occupied by LLC</td>
</tr>
<tr>
<td>$(1 - \alpha).A$</td>
<td>Area occupied by cores</td>
</tr>
<tr>
<td>$f$</td>
<td>Clock frequency (GHz)</td>
</tr>
<tr>
<td>$P$</td>
<td>Number of cores</td>
</tr>
<tr>
<td>$f_{\text{flops}}(f)$</td>
<td>Peak performance of a single core at frequency $f$ (GFLOPs)</td>
</tr>
<tr>
<td>$B_{\text{mem}}$</td>
<td>Peak memory bandwidth to LLC (Gbytes/sec)</td>
</tr>
<tr>
<td>$S$</td>
<td>Last level cache size in words</td>
</tr>
</tbody>
</table>

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

$$T = \max \left( \frac{W}{P \cdot f_{\text{flops}}(f)} \cdot \frac{Q_{\text{mem}}}{B_{\text{mem}}} \right) = \max \left( \frac{W}{P \cdot f_{\text{flops}}(f)} \cdot \frac{W}{B_{\text{mem}} \cdot \text{OI}(S)} \right)$$

(1)

with,

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W$</td>
<td>Total number of arithmetic operations</td>
</tr>
<tr>
<td>$Q_{\text{mem}}$</td>
<td>Total number of bytes for memory accesses</td>
</tr>
<tr>
<td>$T$</td>
<td>Total computation time</td>
</tr>
<tr>
<td>$\text{OI}(S)$</td>
<td>Operational Intensity as a function of $S$</td>
</tr>
</tbody>
</table>
For a given application, $W$ can be expressed in terms of problem size parameters and lower bounds on $Q_{mem}$ can be expressed through our analysis in terms of $S$. As an example, for matrix multiplication we have $W = 2N^3$, $Q_{mem} \geq \frac{N^3}{2\sqrt{2}} + 3N^2$.

Energy. The total energy consumption is modeled as:

$$E = W \varepsilon_{flop}(f) + Q_{mem} \varepsilon_{mem} + (P_{\pi_{cpu}} + P_{\pi_{cache}}(S)) \cdot T,$$

where the quantities used are defined below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_{cpu}$</td>
<td>Static leakage power per core</td>
</tr>
<tr>
<td>$\pi_{cache}(S)$</td>
<td>Static leakage power for cache of size $S$</td>
</tr>
<tr>
<td>$\varepsilon_{flop}(f)$</td>
<td>Energy per arithmetic operation at frequency $f$</td>
</tr>
<tr>
<td>$\varepsilon_{mem}$</td>
<td>Energy per byte of data access from/to memory</td>
</tr>
</tbody>
</table>

### 4.2. Architectural Parameters

We demonstrate the utility of modeling upper bounds on operational intensity by performing an analysis over possible architectural variants 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>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die Size</td>
<td>684 mm$^2$</td>
</tr>
<tr>
<td>Technology Node</td>
<td>45 nm</td>
</tr>
<tr>
<td>Num of Cores</td>
<td>8</td>
</tr>
<tr>
<td>Num of LLC Slices</td>
<td>8</td>
</tr>
<tr>
<td>LLC Slice Size</td>
<td>3MB</td>
</tr>
<tr>
<td>LLC Size (total)</td>
<td>24 MB</td>
</tr>
<tr>
<td>DRAM Channels</td>
<td>4</td>
</tr>
<tr>
<td>Core voltage</td>
<td>0.85 V</td>
</tr>
<tr>
<td>Core Max Clock</td>
<td>2.26 GHz</td>
</tr>
<tr>
<td>TDP</td>
<td>130 W</td>
</tr>
<tr>
<td>Threads per core (SMT)</td>
<td>2</td>
</tr>
<tr>
<td>Leakage (total)</td>
<td>21 W</td>
</tr>
</tbody>
</table>

Table I: Nehalem-EX processor spec.

Table I 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 last level cache (LLC).

<table>
<thead>
<tr>
<th>Unit</th>
<th>Width (mm)</th>
<th>Height (mm)</th>
<th>Area (mm$^2$)</th>
<th>Area (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die</td>
<td>31.9</td>
<td>21.4</td>
<td>684</td>
<td>100</td>
</tr>
<tr>
<td>Core</td>
<td>7.2</td>
<td>3.7</td>
<td>26.7</td>
<td>3.0</td>
</tr>
<tr>
<td>LLC Slice</td>
<td>7.8</td>
<td>3.7</td>
<td>29.3</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table II: Nehalem-EX die dimensions.

Table II 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 III: Modeled LLC Slices using CACTI [Muralimanohar et al. 2009]

Table III shows 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.

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 greater number 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 range 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 IV shows how changing voltage and frequency effects total chip and core power.

Table IV: Nehalem-EX: effect of changing voltage and frequency on core/chip power

Finally, we summarize the parameters for the testbed architecture used for the modeling in the next sections: $F_{\text{flops}}(f) = 9.04$ GfLOPS @ 2.26 GHz, $B_{\text{mem}} = 40$ GB/s, $A = 684$ mm$^2$, $A_{\text{core}} = 26.738$ mm$^2$, $\epsilon_{\text{flop}}(f) = 1.3$ nJ/flop, $\epsilon_{\text{mem}} = 0.63$ 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 $\alpha$, a maximum of 25 cores would fit on the chip. At the other extreme, for $\alpha$ of 0.95, only one core can be put on the chip and an LLC cache of 64 MBytes could be accommodated.

5. ALGORITHM-ARCHITECTURE CODESIGN: BOUNDS ON PERFORMANCE

In this section, we consider the implications of upper bounds on OI on the maximal performance (operations per second) achievable on a chip, for different algorithms. As described in the previous section, we assume that the chip area can be partitioned as desired among processor cores or cache. As shown below for three demonstration benchmarks, the upper bound on OI for an algorithm can be modeled as a monotonic non-decreasing 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 below, 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 multi-roofline plot.

The size of the LLC was varied from a tiny 4 Kbyte size (representing a fraction of under 0.1% of the chip of approximately 700 mm² area) to 64 Mbytes (filling almost the entire chip area). For demonstration purposes, we analyze three algorithms: matrix multiplication (MM), fast Fourier transform (FFT), and a conjugate gradient (CG) iterative linear system solver. In all these cases, the problem sizes were set to be much larger than LLC. Using the analysis techniques described earlier, we can derive the following upper bounds on OI for the algorithms, as a function of cache capacity ($S$ words):

- Matrix multiplication: $4\sqrt{S}$
- FFT: $\log(S)/2$
- CG: $10/3$ (it is independent of cache capacity)

Derivations of these parametric expressions for the upper bound on OI are provided in the appendix.

Fig. 5 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 are very different for the three benchmarks. Matrix multiplication shows 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.

![Cache area vs. Operational Intensity](image)

Fig. 5: Operational intensity upper bounds as a function of $\alpha$

Fig. 6 shows four roofline plots for four specific values of $\alpha$: 0.01, 0.25, 0.5 and 0.95, respectively. In each case, the vertical lines are placed at the upper bound on OI for the three algorithms, and either intersect the curved bandwidth roofline or the horizontal processor-peak roofline. We plot the OI using a log-scale while the performance is in a linear scale for easier comparison – hence the bandwidth roofline is curved in these plots.

At a very small value of $\alpha$ 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 226 GFLOPs (9.04*25). The OI values increases from 0.41 for CG to 1.0 for FFT and 181.02 for MM. CG and FFT are bandwidth bound, although not to the same extent, while MM is not bandwidth bound.

When $\alpha$ is increased to 0.25, the number of cores that can be put on chip decreases, causing a lowering of the horizontal roofline. The OI values increase for FFT and MM, while they are...
unchanged for CG. Compared to $\alpha$ 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, the performance upper bound decreases 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 $\alpha$ are in opposite directions for FFT and matrix multiplication.
Fig. 7 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, i.e., over the range of possible values of \( \alpha \). 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, the less the remaining area on the chip for cores. The value of \( \alpha \) 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 \( \alpha \) are shown in the figure, intersecting with the bandwidth roofline (corresponding to 40 Gbytes/sec; here we use a log-log plot, and therefore the bandwidth roofline is a straight line). Four instances of horizontal rooflines are shown in the figure, tagged with the value of \( \alpha \) and the corresponding number of cores.

The results for the three benchmarks 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 16.7 GFLOPs (0.417*40Gbytes/sec). Since each core has a peak performance of 9.04 GFLOPs for the modeled technology, CG is bandwidth-bound for any number of cores greater than one. But if \( \alpha \) is 0.95, we can only put one core on the chip, and the upper bound is 9 GFLOPs. Thus, the two red dots in the multi-roofline plot capture the entire range of possibilities as alpha is varied: 9 GFLOPs when \( P = 1 \) (\( \alpha \) is above 0.93) and 16.7 GFLOPs when \( P \geq 2 \).

For FFT, the upper bound on OI ranges from 0.5625 to 1.4375 over the range of \( \alpha \) from 0.01 to 0.95. At \( \alpha = 0.01 \), OI=0.5625, and the performance upper bound is 22.5 GFLOPs, and the computation is severely bandwidth bound – the 25 cores that could be put on chip for this value of \( \alpha \) would be heavily underutilized. As \( \alpha \) is increased, the upper bound on performance improves, and hugs the bandwidth roofline. But when \( \alpha \) goes above 0.75, and the number of cores drops below 6, 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.

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.

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 maximal 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. Fig. 8, Fig. 9, and Fig. 10 show the variation in the upper bounds on energy efficiency for FFT, matrix multiplication, and CG, respectively, 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 (2260 MHz, 1860 MHz, and 1460 MHz), representing voltage/frequency scaling, and for each of the frequencies, five choices of processor count (1, 2, 4, 8, and 16). Different curves correspond to different capacities of the LLC. The overall trends are as follows.

1) For all three benchmarks, the groups of clustered lines move upwards 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 downwards 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 there is no reduction in the lower bound for execution time. For matrix multiplication, 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 matrix multiplication, 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 three benchmarks, if we have the flexibility to choose (i) the number of cores to be turned on, (ii) the amount of cache area to be used, and (iii) voltage/frequency at which the cores are to be run. From Equations (1) and (2), we
have the energy efficiency,

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

Depending on the upper bound on the OI of the algorithms, we analyze different cases below 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 \(B_{\text{mem}} \cdot OI(S) < F_{\text{flops}}(f)\). With increasing frequency, \(\epsilon_{\text{flop}}(f)\) increases and thus the energy efficiency deteriorates. Hence, highest energy efficiency is achieved when \(P = 1\) and \(f\) is set at its minimum, and, Equation (3) reduces to

\[
E_{\text{eff}} = \left( \epsilon_{\text{flop}}(f_{\text{min}}) + \frac{\epsilon_{\text{mem}}}{OI(S)} + \frac{\pi_{\text{cpu}} + \pi_{\text{cache}}(S)}{B_{\text{mem}} \cdot OI(S)} \right)^{-1},
\]

where, \(f_{\text{min}}\) is the minimum allowable frequency for the architecture.

**Case II:** The algorithm is compute-bound with \(p\) or less cores and at all frequencies.

From Equation (3), we can see that at any given frequency \(f\), increasing \(P\) improves the \(E_{\text{eff}}\). Hence, \(P = p\) is the best choice irrespective of the frequency. Also, with increasing \(f\), \(F_{\text{flops}}(f)\) increases linearly, while \(\epsilon_{\text{flop}}(f)\) increases super-linearly. For a fixed value of \(p\), the optimal energy efficiency is dependent on the machine parameters like \(\pi_{\text{cache}}(S)\) and \(\epsilon_{\text{flop}}(f)\). The maximal energy efficiency obtained,

\[
E_{\text{eff}} = \max_{f \in [f_{\text{min}}, f_{\text{max}}]} \left( \epsilon_{\text{flop}}(f) + \frac{\epsilon_{\text{mem}}}{OI(S)} + \frac{P \cdot \pi_{\text{cpu}} + \pi_{\text{cache}}(S)}{P \cdot F_{\text{flops}}(f)} \right)^{-1},
\]

where, \(f_{\text{min}}\) and \(f_{\text{max}}\) 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 \(f_{\text{cutoff}}\) be the frequency where the algorithm transitions from compute-bound to memory-bound region. For the region where \(f \geq f_{\text{cutoff}}\), from case I, we know that once the algorithm becomes bandwidth-bound, increasing the frequency further has detrimental effect on the energy efficiency. Hence, the best energy efficiency is achieved when \(f = f_{\text{cutoff}}\), where, \(P \cdot F_{\text{flops}}(f_{\text{cutoff}}) = B_{\text{mem}} \cdot OI(S)\).

When \(f < f_{\text{cutoff}}\), 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,

\[
p \cdot F_{\text{flops}}(f) < B_{\text{mem}} \cdot OI(S).\]

Hence, \(E_{\text{eff}} = \max_{f \in [f_{\text{min}}, f_{\text{cutoff}}]} \left( \epsilon_{\text{flop}}(f) + \frac{\epsilon_{\text{mem}}}{OI(S)} + \frac{P \cdot \pi_{\text{cpu}} + \pi_{\text{cache}}(S)}{P \cdot F_{\text{flops}}(f)} \right)^{-1}.
\]

**Case IV:** The algorithm is compute-bound at all frequencies with \(p\) cores, and becomes bandwidth bound with \(q = p + 1\) or higher number of cores.

This case gives rise to two scenarios: (1) Performance at higher frequencies \((f_{q} \in [f_{1}, f_{2}])\) with \(p\) cores overlaps with the performance at lower frequencies \((f_{q} \in [f_{\text{min}}, f_{2}])\) 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, i.e., \(q \cdot F_{\text{flops}}(f_{\text{min}}) - p \cdot F_{\text{flops}}(f_{\text{max}}) > 0\). In both these scenarios, the maximum achievable energy efficiency depends on the machine parameters.

Similar to the performance roofline in Fig. 7, the energy roofline has been plotted for the three benchmarks in Fig. 11 with a range of OI corresponding to the range of LLC sizes for our testbed architecture.

Fig. 11 shows a similar trend for the energy efficiency as that of the performance in Sec. 5. Mat-mul is always compute-bound and hence falls under case II for all values of \(S\). Based on
the machine parameters for the testbed, the optimal energy efficiency is achieved for the lowest frequency value, in general. But, as the number of active cores approaches one (as the size of LLC increases), the higher frequency value provides maximal energy efficiency. On the other hand, CG becomes bandwidth-bound as the number of active cores for the given LLC size exceeds 3. Hence, CG starts at case IV with $P = 3$, and as the LLC size increases (and the number of allowable cores goes below 3), it enters into case II. CG has the best upper bound on energy efficiency of 0.35 GFLOP/J for the cache size of 4 KB with 3 cores @ 1.26 GHz. The upper bound on energy efficiency for FFT initially increases with the increasing cache size and then starts decreasing for the similar reasons as that of the performance upper bounds. FFT achieves its maximal energy efficiency for the cache size of 8 MB at $P = 10$ and $f = 1.26$ GHz.

7. RELATED WORK

Williams et al. [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. [Choi et al. 2013; Choi et al. 2014] developed the energy version of the roofline model. Czechowski et al. [Czechowski et al. 2011] have developed balance principles for algorithm-architecture co-design. These models characterize algorithms using operational intensity, which however is not a fixed quantity for an algorithm-architecture combination. Our work complements the above efforts; we are not aware of any other efforts that have developed methods for characterizing upper bounds on operational intensities of algorithms.

Several efforts have focused on developing I/O lower bounds, which is equivalent to the problem of finding upper bounds on operational intensity. Hong & Kung provided the first characterization of the I/O complexity problem using the red/blue pebble game and the equivalence to 2S-partitioning of CDAGs [Hong and Kung 1981]. Several works followed Hong & Kung’s work on I/O complexity in deriving lower bounds on data accesses [Aggarwal and Vitter 1988; Ågarwal 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 et al. provided several lower bounds for sorting algorithms [Aggarwal and Vitter 1988]. Savage [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. [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. Demmel et al. [Ballard et al. 2011; 2012; Demmel et al. 2012; Solomonik et al. 2013] have developed lower bounds as well as optimal algorithms for several linear algebra computations including QR and LU decomposition.
and all-pairs shortest paths problem. Bilardi et al. [Bilardi et al. 2000; Bilardi and Peserico 2001] develop the notion of access complexity and relate it to space complexity.

Extending the scope of the Hong & Kung model to more complex memory hierarchies has also been the subject of research. Savage provided an extension together with results for some classes of computations that were considered by Hong & Kung, providing optimal lower bounds for I/O with memory hierarchies [Savage 1995]. Valiant proposed a hierarchical computational model [Valiant 2011] that offers the possibility to reason in an arbitrarily complex parameterized memory hierarchy model.

The use of Hong & Kung’s model has required manual algorithm-specific reasoning to find S-partitions, even for regular graphs. Savage’s [Savage 1995] S-span model also requires problem-specific insights. In addition to these works, other approaches [Aggarwal and Vitter 1988; Demmel et al. 2012; Valiant 2011; Ballard et al. 2012; Bilardi and Peserico 2001] also require problem-specific insights to develop bounds. In contrast, we have developed an approach that can be used to develop I/O lower bounds for an arbitrary CDAG without any problem-specific reasoning. We note here that very recent work from U.C. Berkeley [Christ et al. 2013] has developed a very novel approach to developing parametric I/O lower bounds that does not require problem-specific reasoning. The approach is applicable/effective for a class of nested loop computations but is either inapplicable or produces weak lower bounds for other computations (e.g., stencil computations, FFT, etc.).

In a recent paper, we [Elango et al. 2014] extend the parallel model for shared-memory architectures by Savage and Zubair [Savage and Zubair 2008] to also include the distributed-memory parallelism present in all scalable parallel architectures. The works of Irony et al. [Irony et al. 2004] and Ballard et al. [Ballard et al. 2011] model communication across nodes of a distributed-memory system. Bilardi and Preperata [Bilardi and Preparata 1999] develop lower bound results for communication in a distributed-memory model specialized for multi-dimensional 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 operational 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, i.e., is affected by semantics-preserving code transformations. Therefore understanding fundamental performance bottlenecks for an algorithm generally requires analysis of many alternatives.

In this paper, 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.

APPENDIX: Operation Intensity Upper Bounds of Algorithms

A.1. Matrix-Matrix multiplication

The standard matrix-matrix multiplication of size $n \times n$ has an operation count, $W = 2n^3$. Irony et al. [Irony et al. 2004] showed that the I/O lower bound $Q$ for matmul satisfies, $Q \geq \frac{n^3}{\sqrt{2S}}$, where $S$ is size of LLC in words. Hence, the operational intensity $OI$ satisfies, $OI \leq \frac{2n^3}{\frac{n^3}{(2\sqrt{2S})}} = 4\sqrt{2S}$.

A.2. FFT

The $n$-point FFT (of height $\log(2n)$) has an operation count of $n\log(2n)$. The following theorem derives the I/O lower bound for FFT using the mincut approach discussed in Sec. 3.
THEOREM A.1. For the n-point FFT graph, the minimum I/O cost, $Q$, asymptotically satisfies

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

where $S$ is the number of red pebbles

PROOF. Consider a DAG for an FFT of size $m$. Consider a pebble game instance $\mathcal{P}$ with minimum I/O and the time-stamp at which the first output vertex (i.e. vertex with no successors) $o$ is fired by this schedule. Let $S$ be the vertices already fired strictly before $o$, and $T$ the others. As $o$ is an output vertex, $S$ contains all the $m$ input vertices. By construction, $T$ contains all the $m$ output vertices. Hence, the corresponding wavefront, $|\mathcal{W}(o)| \geq m$.

Now, a DAG for an $n$-point FFT (of height $\log(2n)$) can be decomposed into disjoint sub-DAGs corresponding to $m$-points FFTs (and of height $\log(2m)$). This gives us $\lfloor n/m \rfloor \times \lfloor \log(2n)/\log(2m) \rfloor$ full sub-DAGs. From Lemma 3.1, the I/O cost of each sub-FFT (by counting a full spill for each input node) is at least $2 \times (m - S)$. If we consider $m = S \log(S)$,

$$Q \geq \lfloor n/S \log(S) \rfloor \times \left\lfloor \log(2n) \right\rfloor / \log(2S \log(S)) \times 2(S \log(S) - S)$$

$$\approx \frac{n \log(n)}{S \log^2(S)} \times 2(S \log(S))$$

$$\approx \frac{2n \log(n)}{\log(S)}$$

Finally, from the operation count and the I/O complexity for FFT, we obtain the upper bound on the operational intensity, $OI \leq \frac{n \log(2n)}{2n \log(n) \left\{ \log(n) \right\}} \approx \frac{\log(S)}{2}$.

A.3. Conjugate Gradient

Conjugate Gradient (CG) is an iterative method for solving sparse system of linear equations that generally arise from discretization of partial differential equations (PDEs). We consider the use of CG in solving a sparse linear system arising from the discretization of the PDE for the heat equation, described below.

Consider the heat flow on a long thin bar of unit length, of uniform material and insulated, so that the heat can enter and exit only at the boundaries (refer Fig. 12(a)). Let $u(x, t)$ represent the temperature at position $0 \leq x \leq 1$, and time $t \geq 0$. The objective is to determine the change in temperature over time ($u(x, t)$). The governing heat equation that describes this distribution of heat is given by the PDE:

$$\frac{du(x, t)}{dt} = \alpha \times \frac{d^2u(x, t)}{dx^2}$$

where, $\alpha$ is the thermal diffusivity of the bar. (For mathematical treatment, it is sufficient to consider $\alpha = 1$).

Since the problem is continuous, to numerically solve the heat equation, it needs to be discretized (through finite difference approximation) to reduce it to a finite problem. In the discretized problem,
the values of $u(x,t)$ are only computed at discrete points at regular intervals of the bar, called the computational grid or mesh. The state variables at these grid points are given by $u(x(i),t(m))$, where $x(i) = i \times h$, $0 \leq i \leq n+1 = 1/h$ and $t(m) = m \times k$; $h$ and $k$ are the grid spacing and timestep, respectively. Fig. 12(b) shows an example grid obtained by discretizing the one-dimensional bar.

The governing equation, after discretization, yields the following equation at grid point $i$ and timestep $m+1$.

$$\frac{-a}{2} \times U(i-1,m+1) + (1 + a) \times U(i,m+1) - \frac{a}{2} \times U(i+1,m+1) =$$

$$\frac{a}{2} \times U(i-1,m) + (1 - a) \times U(i,m) + \frac{a}{2} \times U(i+1,m)$$

where, $U(p,q) = u(x(p), t(q))$ and $a = k/h^2$. Hence, the solution to the problem involves solving a linear system of $n-1$ equations at each timestep till convergence. Each timestep $m+1$ is dependant on values of the previous timestep $m$.

This linear system can be represented in tridiagonal matrix form as follows:

$$\begin{pmatrix}
1 + a & -\frac{a}{2} & 0 & \cdots & 0 \\
-\frac{a}{2} & 1 + a & -\frac{a}{2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & \cdots & -\frac{a}{2} & 1 + a & -\frac{a}{2} \\
0 & \cdots & 0 & -\frac{a}{2} & 1 + a
\end{pmatrix}
\begin{pmatrix}
U(1,m+1) \\
U(2,m+1) \\
U(3,m+1) \\
\vdots \\
U(n-1,m+1) \\
U(n,m+1)
\end{pmatrix} =
\begin{pmatrix}
b(1,m) \\
b(2,m) \\
b(3,m) \\
\vdots \\
b(n-1,m) \\
b(n,m)
\end{pmatrix}$$

where, $b(i,m)$ represents the right-hand side of the $i$-th equation at timestep $m+1$. In general, for a $d$-dimensional problem, the coefficient matrix is of size $n^d$-by-$n^d$, while the vectors are of size $n^d$. In practice, the elements of the matrix 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.

Each iteration of CG involves one sparse matrix-vector product, three vector updates, and three vector dot-products. The complete pseudocode is shown in Fig. 13.

```
x is the initial guess
p ← r ← b - Ax
do
  v ← Ap
  b ← (r.r)  // SpMV
  a ← b/(p.v)
  x ← x + ap
  r ← r - av  // AXPY
  g ← (r.r)/b  // Dot-prod
  p ← r + gp  // AXPY
until ((r.r) is small)
```

Fig. 13: Classical Conjugate Gradient method

**Operation count:** The vector dot-product at line 6 requires $2n^d$ operations. The computation of $(r.r)$, at lines 9, 11 and 5, requires a single vector dot-product of operation count $2n^d$. The vector update operations at lines 7, 8 and 10 have operation count of $2n^d$ each. The SpMV operation at line 4 is a stencil computation that requires $2(2d+1)n^d$ operations. This provides a total operation count of $20n^2T$ for a two-dimensional problem.

Before we derive the I/O lower bound for CG, we state the following theorem that is useful to obtain tighter bound for CG.
THEOREM A.2 (VERtex SPLITTING). Let \( C = (I,V \cup dV,E,O) \) be a CDAG. Let \( C' = (I,V \cup dV',E',O) \) be the CDAG induced by splitting the vertices in \( dV \) of \( C \). Then, \( IO(C) \) can be related to \( IO(C') \) as follows: \( IO(C') = IO(C) \).

PROOF. Consider a complete calculation \( \mathcal{P} \) for \( C \), of cost \( IO(C) \). We will build a complete calculation \( \mathcal{P}' \) for \( C' \), of cost \( IO(C') \). This will prove that \( IO(C') \leq IO(C) \). We build \( \mathcal{P}' \) from \( \mathcal{P} \) as follows: (1) for any vertex \( v \in dV \), the transition \( R3 \) involving \( v \) in \( \mathcal{P} \) is suffixed by a transition \( R3 \); (2) any other transition in \( \mathcal{P} \) is reported as is in \( \mathcal{P}' \).

Now, consider the complete calculation \( \mathcal{P}' \) for \( C' \). We will build a complete calculation \( \mathcal{P} \) for \( C \) of cost \( IO(C') \). This will prove that \( IO(C) \leq IO(C') \). We build \( \mathcal{P} \) from \( \mathcal{P}' \) as follows: (1) any transition of any vertex in \( V \) in \( \mathcal{P}' \) is reported as is in \( \mathcal{P} \); (2) any transition of a vertex in \( dV' \) in \( \mathcal{P}' \) is reported as is in \( \mathcal{P} \); (3) any transition \( R3 \) in \( \mathcal{P}' \) on a vertex \( v'' \in dV'' \) is ignored; (4) any transition other than \( R3 \) on a vertex \( v'' \in dV'' \) in \( \mathcal{P}' \) is reported as is in \( \mathcal{P} \). \( \square \)

Theorem A.3 stated below (see [Elango et al. 2013] for proof) allows to compose the results of non-convex partitioning.

THEOREM A.3 (DECOMPOSITION). Let \( C = (I,V,E,O) \) be a CDAG. Let \( V_1,V_2,\ldots,V_p \) be an arbitrary (not necessarily acyclic) disjoint partitioning of \( V \) (\( i \neq j \Rightarrow V_i \cap V_j = \emptyset \) and \( \bigcup_{1 \leq i \leq p} V_i = V \)) and \( C_1,C_2,\ldots,C_p \) be the induced partitioning of \( C \) (\( I_i = I \cap V_i \), \( E_i = E \cap V_i \times V_i \), \( O_i = O \cap V_i \)). Then \( \sum_{1 \leq i \leq p} IO(C_i) \leq IO(C) \). In particular, if \( Q \) is a lower bound on the IO of \( C_i \), then \( \sum_{1 \leq i \leq p} Q_i \) is a lower bound on the I/O of \( C \).

We now provide the I/O lower bound for CG using the min-cut approach.

THEOREM A.4 (I/O LOWER BOUND FOR CG). For a d-dimensional grid of size \( n^d \), the minimum I/O cost to solve the corresponding linear system using CG, \( Q \), satisfies \( Q \geq 6n^dT \), when \( n^d \gg S \); here, \( T \) represents the number of outer loop iterations.

We provide a proof sketch here. Formal proof is provided below. Let \( C \) be the CDAG for CG. \( C \) is decomposed into \( T \) sub-CDAGs, each corresponding to a single iteration of the loop. Each of these sub-CDAGs is subdivided into two sub-CDAGs by splitting the vertices corresponding to computations of elements of \( r \) in line 8 and removing the split edges (Theorem A.2).

Consider the dot-product computation at line 6. Let \( x \) be the vertex corresponding to the computation of \( a \). Vectors \( p \) and \( v \), that are used as operands for computing \( a \) are needed later, at lines 7 and 8, along with scalar \( a \). Hence, any graph min-cut wavefront induced by \( x \) should be of size at least \( 2n^d \).

Similarly, consider line 9. Let \( y \) be the vertex corresponding to computation of \( g \). Vector \( r \), which is an operand for computing \( g \) is used later in line 10, along with \( g \). Thus, any graph min-cut wavefront induced by \( y \) should be of size at least \( n^d \).

By summing up the individual lower bounds of all the sub-CDAGs and applying theorem A.3, we have, \( Q \geq T \times 2(3n^d - 2S) \approx 6n^dT \), when \( n^d \gg S \). \( \square \)

PROOF. Let \( C \) be the CDAG for CG. Consider the vertices \( V_p \) corresponding to computation of elements of \( p \) (line 10) at all the iterations. By applying Theorem A.2 split \( v \in V_p \) into \( v' \) and \( v'' \), and remove the edge between \( v' \) and \( v'' \). This provides us \( T \) sub-CDAGs, \( C_1,C_2,\ldots,C^T \), each corresponding to a single iteration of the loop.

Subdivide \( C_i,1 \leq i \leq T \), into two sub-CDAGs, \( C_i^1 \) and \( C_i^2 \), by splitting \( v \in V_i' \) – vertices corresponding to computation of elements of \( r \) (line 8) in iteration \( i \) – into \( v' \) and \( v'' \), and removing the edges between \( v' \) and \( v'' \).

Let \( v^1_i \) be the vertex in \( C_i^1 \) corresponding to computation of scalar \( a \) at line 6. The \( 2n^d \) vertices in \( \text{Anc}(v^1_i) \), corresponding to elements of \( p \) and \( v \) computed in iterations \( i-1 \) and \( i \), respectively, have
edges to the vertices in \( \text{Desc}(v_1^{i,1}) \) (lines 7 and 8), i.e., \(|W_{\text{min}}^{i,1}(v_1^{i,1})| = 2n^d\). Hence, from lemma 3.1, we have that \( Q_{v_1} \), the minimum I/O restricted to \( C^{i,1} \), is bounded by \( 2(2n^d - S) \).

Let \( v_g^i \) be the vertex in \( C^{i,2} \) corresponding to scalar \( g \) at line 9. The \( n^d \) vertices in \( \text{Anc}(v_g^{i,2}) \), corresponding to elements of \( r \) computed in iteration \( i \), have edges to the vertices in \( \text{Desc}(v_g^{i,2}) \) (line 10), i.e., \(|W_{\text{max}}^{i,2}(v_g^{i,2})| = n^d\). Hence, \( Q_{v_g} \) is bounded by \( 2(n^d - S) \).

From theorem 3.2, by summing up the I/O cost of sub-CDAGs, we get, minimum I/O cost \( Q \geq T \times 2(3n^d - 2S) \). When \( n^d \gg S \), we have, \( Q \geq 6n^d T \). \( \square \)

Hence, from Theorem A.4 and the operation count for CG, we obtain the upper bound on the operational intensity, \( OI \leq 20/6 = 10/3 \).

REFERENCES


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