On Characterizing the Data Movement Complexity of Computational DAGs for Parallel Execution

Venmugil Elango
The Ohio State University
elango.4@osu.edu

Fabrice Rastello
Inria
Fabrice.Rastello@inria.fr

Louis-Noël Pouchet
Univ. of California–Los Angeles
pouchet@cs.ucla.edu

J. Ramanujam
Louisiana State University
ram@cct.lsu.edu

P. Sadayappan
The Ohio State University
saday@cse.ohio-state.edu

ABSTRACT

Technology trends are making the cost of data movement increasingly dominant, both in terms of energy and time, over the cost of performing arithmetic operations in computer systems. The fundamental ratio of aggregate data movement bandwidth to the total computational power (also referred to the *machine balance parameter*) in parallel computer systems is decreasing. It is therefore of considerable importance to characterize the inherent data movement requirements of parallel algorithms, so that the minimal architectural balance parameters required to support it on future systems can be well understood.

In this paper, we develop an extension of the well-known red-blue pebble game to develop lower bounds on the data movement complexity for the parallel execution of computational directed acyclic graphs (CDAGs) on parallel systems. We model multi-node multi-core parallel systems, with the total physical memory distributed across the nodes (that are connected through some interconnection network) and in a multi-level shared cache hierarchy for processors within a node. We also develop new techniques for lower bound characterization of non-homogeneous CDAGs. We demonstrate the use of the methodology by analyzing the CDAGs of several numerical algorithms, to develop lower bounds on data movement for their parallel execution.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: General; B.4.4 [**Hardware**]: Input/Output and Data Communications—*Performance Analysis and Design Aids*; D.2.8 [**Software**]: Metrics—*Complexity measures*

Keywords

I/O complexity, Red-blue pebble game, Parallel data movement complexity, Lower bounds

SPAA'14, June 23–25, 2014, Prague, Czech Republic.

ACM 978-1-4503-2821-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2612669.2612694.

1. INTRODUCTION

Recent technology trends have resulted in much greater rates of improvement in computational processing rates of processors than the bandwidths for data movement across nodes or between the memory/cache hierarchies within nodes in a parallel system. This mismatch between maximum computational rate and peak memory bandwidth means that data movement and communication costs of an algorithm will be increasingly dominant determinants of performance. Although hardware techniques for data prefetching and overlapping of computation with communication can alleviate the impact of memory access latency on performance, the mismatch between maximum computational rate and peak memory bandwidth is much more fundamental; *the only solution is to limit the total rate of data movement between components of a parallel system to rates that can be sustained by the interconnects at different components and levels of a parallel computer system.*

It is therefore of considerable importance to develop techniques to characterize lower bounds on the data movement complexity of parallel algorithms. We address this problem in this paper. We formalize the problem by developing a parallel extension of the red-blue pebble game model introduced by Hong and Kung in their seminal work [16] on characterizing the data access complexity (called *I/O complexity* by them) for sequential execution of computational directed acyclic graphs (CDAGs). Our extended pebble game abstracts data movement in scalable parallel computers today, that consist of multiple nodes interconnected by a high-bandwidth interconnection network, with each node containing a number of cores that share a hierarchy of caches and the node's physical main memory.

In contrast to some other prior efforts that have modeled lower bounds for data movement in parallel computations, we focus on relating data movement lower bounds to the critical architectural balance parameter of the ratio of peak data movement bandwidth (in GBytes/sec) to peak computational throughput (in GFLOPs) at different levels of a parallel system. We develop techniques for deriving lower bounds for data movement for CDAGs under the parallel red-blue pebble game, and use these techniques to analyze a number of numerical algorithms. Interesting insights are provided on architectural bottlenecks that limit the performance of the algorithms.

This paper makes several contributions:

- It develops an extension of the red-blue pebble game that effectively models essential characteristics of scalable parallel computers with multi-level parallelism; (i) multiple nodes

with local physical memory that are interconnected via a high-speed interconnection network like Infiniband or a custom interconnect (e.g., IBM BlueGene system [17], or Cray XE6 [10]), and (ii) many cores at each node, that share a hierarchy of caches and the node's physical main memory.
- It develops a lower bound analysis methodology that is effective for analyzing non-homogeneous CDAGs using a decomposition approach.
- It develops new parallel lower-bounds analysis for a number of numerical algorithms.
- It presents insights into implications on different architectural parameters in order to achieve scalable parallel execution of the analyzed algorithms.

2. BACKGROUND: THE RED-BLUE PEBBLE GAME

2.1 Computational Model

The model of computation we use is a computational directed acyclic graph (CDAG), where computational operations are represented as graph vertices and the flow of values between operations is captured by graph edges. Two important characteristics of this abstract form of representing a computation are that (1) there is no specification of a particular order of execution of the operations: the CDAG abstracts the schedule of operations by only specifying partial ordering constraints as edges in the graph; and (2) there is no association of memory locations with the source operands or the result of any operation. We use the notation of Bilardi & Peserico [5] to formally describe CDAGs. We begin with the model of CDAG used by Hong & Kung.

DEFINITION 1 (CDAG-HK).
A computational directed acyclic graph (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) $V \setminus I$ is called the operation set and all its vertices have one or more incoming edges; (5) $O \subseteq V$ is called the output set.

2.2 The Red-Blue Pebble Game

Hong & Kung used this computational model in their seminal work [16]. minimal number of I/O operations needed while optimally playing the The *Red-Blue pebble game*. game uses two kinds of pebbles: a fixed number of red pebbles that represent small fast local memory (could represent cache, registers, etc.), and an arbitrarily large number of blue pebbles that represent the large slow main memory. Starting with blue pebbles on all inputs nodes in the CDAG, the game involves the generation of a sequence of steps to finally produce blue pebbles on all outputs. A game is defined as follows.

DEFINITION 2 (RED-BLUE PEBBLE GAME [16]).
Given a CDAG $C = (I,V,E,O)$ such that any vertex with no incoming (resp. outgoing) edge is an element of I (resp. O), S red pebbles and arbitrary number of 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 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 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 on that CDAG. An *optimal* calculation is a complete calculation achieving the minimal number of I/O operations.

2.3 S-partitioning for Lower Bounds on I/O Complexity

This red-blue pebble game provides an operational definition for the I/O complexity problem. However, it is not practically feasible to generate all possible complete calculations for large CDAGs. 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 3 (S-PARTITIONING OF CDAG [16]).
Given a CDAG C, 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 $\forall i,\ \exists D \in \mathsf{Dom}(V_i)$ *such that* $|D| \leq S$

P4 $\forall i,\ |\mathsf{Min}(V_i)| \leq S$

where a dominator set of V_i, $D \in \mathsf{Dom}(V_i)$ 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, $\mathsf{Min}(V_i)$ is the set of vertices in V_i that have all its successors outside of V_i; and $|Set|$ is the cardinality of the set Set.

Corresponding to any complete calculation on that CDAG using S red pebbles, Hong & Kung showed a construction for a $2S$-partition of a CDAG, 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 follows.

THEOREM 1 (PEBBLE GAME, I/O AND $2S$-PARTITION [16]).
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 $S \times h \geq q \geq S \times (h-1)$, where q is the number of I/O moves in the complete calculation and h is the number of subsets in the 2S-partition.

The tight association from the above theorem between any complete calculation and a corresponding $2S$-partition provides the following key lemma that served as the basis for Hong & Kung's approach to deriving lower bounds on the I/O complexity of CDAGs.

LEMMA 1 (LOWER BOUND ON I/O [16]). *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 of 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 [16] by reasoning about the maximal number of vertices that could belong to any vertex-set in a valid $2S$-partition.

3. ENABLING BOUNDS FOR COMPOSITE CDAGS: THE RBW PEBBLE GAME

Application codes are typically constructed from a number of sub-computations using the fundamental composition mechanisms of sequencing, iteration and recursion. For instance, the conjugate gradient method, described in Sec. 5.2, consists of sequence of sparse matrix-vector product, vector dot-product and SAXPY operations, for every iteration. Applying the I/O lower bounding techniques directly on the CDAG of such composite application codes can produce very weak lower bounds. For instance, consider the following code segment.

```
1 Inputs: p, q, r, s: Vectors of size N
2 Output: sum: Scalar
3 A = p × q^T
4 B = r × s^T
5 C = AB
6 sum = Σ_{i=1}^{N} Σ_{j=1}^{N} C_ij
```

The computational complexity of this computation can be simply obtained by adding together the computational costs of the constituent steps, i.e., $N^2+N^2+2N^3+N^2$ arithmetic operations. In contrast, the data movement complexity for this computation cannot so simply be obtained by adding together the data movement lower bounds for the individual steps. Let us consider data movement costs in a two-level memory hierarchy with unbounded main memory and a limited number of words (S) in fast storage – this might represent the number of registers in the processor, or scratchpad memory or cache memory. It is known [16, 18, 3] that an asymptotic lower bound on data movement between (arbitrarily large) slow memory and fast memory for matrix multiplication of $N \times N$ matrices is $N^3/2\sqrt{2S}$. An outer-product of two vectors of size N requires $2N$ input operations from slow memory and output of the N^2 results back to slow memory, i.e., total I/O of $2N+N^2$, independent of the fast memory capacity S. Similarly, the last step has a data movement complexity of N^2+1 I/O operations between slow and fast memory. But a lower bound on the data movement complexity of the total computation cannot be obtained by simply adding together contributions for the steps. It is not even possible to assert that the maximum among them is a valid lower bound on the data movement complexity of the total computation. The reason is that data from a previous step could possibly be passed to a later step in fast storage without having to be stored in main memory. With $4N+4$ fast memory locations, it is feasible to perform the above computation with a total of only $4N+1$ I/O operations, $4N$ to bring in the four input vectors into fast memory, and repeatedly recompute elements of A and B to contribute to an element of C, and when ready, accumulate it into sum. The I/O complexity of the composite multi-step computation is thus lower than that of the matrix multiply step contained in it. This motivates us to split the CDAG based on individual sub-computations, determine the lower bound for each sub-CDAG separately, and finally compose the result to obtain the I/O lower bound of the whole computation. However, using the original red/blue pebble game model of Hong & Kung, as elaborated below, it is not feasible to analyze the I/O complexity of sub-computations and simply combine them by addition.

The Hong & Kung red/blue pebble game model places blue pebbles on all CDAG vertices without predecessors, since such vertices are considered to hold inputs to the computation, and therefore assumed to start off in slow memory. Similarly, all vertices without successors are considered to be outputs of the computation, and must have blue pebbles at the end of the game. If the vertices of a CDAG corresponding to a composite application are disjointly partitioned into sub-DAGs, the analysis of each sub-DAG under the Hong & Kung red/blue pebble game model will require the initial placement of blue pebbles on all predecessor-free vertices in the sub-DAG, and final placement of blue pebbles on all successor-free vertices in the sub-DAG. The optimal calculation for each sub-DAG will require at least one load (R1) operation for each input and a store (R2) operation for each output. But in playing the red/blue pebble game on the full composite CDAG, clearly it may be possible to pass values in a red pebble between vertices in different sub-DAGs, so that the I/O complexity is less than the sum of the I/O costs for the optimal calculations for each sub-DAG. In fact, it is not even possible to assert that the maximum among the I/O lower bounds for sub-DAGs of a CDAG is a valid lower bound for the composite CDAG.

In order to enable such decomposition, a modified game called the Red-Blue-White pebble game [14] was defined, with the following changes to the Hong & Kung pebble game model (the Red-Blue-White pebble game is formally defined in Sec. 3.1):

1. **Flexible input/output vertex labeling:** Unlike the Hong & Kung model, where all vertices without predecessors must be input vertices, and all vertices without successors must be output vertices, the RBW model allows flexibility in indicating which vertices are labeled as inputs and outputs. In the modified variant of the pebble game, predecessor-free vertices that are not designated as input vertices do not have an initial blue pebble placed on them. However, such vertices are allowed to fire using rule R3 at any time, since they do not have any predecessor nodes without red pebbles. Vertices without successors that are not labeled as output vertices do not require placement of a blue pebble at the end of the game. However, all compute vertices (i.e., vertices in the operation set) in the CDAG are required to have fired for any complete calculation.
2. **Prohibition of multiple evaluations of compute vertices:** The RBW game disallows recomputation of values on the CDAG, i.e., each non-input vertex is only allowed to evaluate once using rule R3. Several other efforts [3, 4, 5, 26, 19, 22, 23, 25, 9, 18, 20, 21] have also imposed such a restriction on the pebble game model. While such a model is indeed more restrictive than the original Hong & Kung model, the restriction in the model enables the development of techniques to form tighter lower bounds [14].

3.1 The Red-Blue-White Pebble Game

DEFINITION 4 (RED-BLUE-WHITE (RBW) PEBBLE GAME). *Given a CDAG $C=(I,V,E,O)$, S red pebbles and arbitrary number of blue and white 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 white pebbles on all vertices and blue pebbles on all output vertices:*

R1 (Input) *A red pebble may be placed on any vertex that has a blue pebble; a white pebble is also placed along with the red pebble, unless the vertex already has a white pebble on it.*

R2 (Output) *A blue pebble may be placed on any vertex that has a red pebble.*

R3 (Compute) *If a vertex v does not have a white pebble and all its immediate predecessors have red pebbles on them, a red pebble along with a white pebble may be placed on v.*

R4 (Delete) *A red pebble may be removed from any vertex (reuse storage).*

In the modified rules for the RBW game, all vertices are required to have a white pebble at the end of the game, thereby ensuring

that the entire CDAG is evaluated. Non-input vertices without predecessors do not have an initial blue pebble on them, but they are allowed to fire using rule R3 at any time – since they have no predecessors, the condition in rule R3 is trivially satisfied. But if all successors of such a node cannot be fired while maintaining a red pebble, "spilling" and reloading using R2 and R1 is forced because the vertex cannot be fired again using R3.

Definition 3 is adapted to this new game so that Theorem 1 and thus Lemma 1 can hold for the RBW pebble game.

DEFINITION 5 (*S*-PARTITIONING OF CDAG – RBW GAME). *Given a CDAG C, an S-partitioning of C is a collection of h subsets of $V \setminus I$ such that:*

P1 $\forall i \neq j,\ V_i \cap V_j = \emptyset$, *and* $\bigcup_{i=1}^{h} V_i = V \setminus I$

P2 *there is no cyclic dependence between subsets*

P3 $\forall i,\ |In(V_i)| \leq S$

P4 $\forall i,\ |Out(V_i)| \leq S$

where the input set of V_i, $In(V_i)$ is the set of vertices of $V \setminus V_i$ that have at least one successor in V_i; the output set of V_i, $Out(V_i)$ is the set of vertices of V_i also part of the output set O or that have at least one successor outside of V_i.

The proof of Theorem 1 under the RBW pebble game is provided in [14].

For (sub-)graphs without input/output sets, the application of S-partitioning will however lead to a trivial partition with all vertices in a single set (e.g., $h = 1$). A careful tagging of vertices as virtual input/output nodes will be required for better I/O complexity estimates, as described below.

3.2 Decomposition

Definition 4 allows the partitioning of a CDAG C into sub-CDAGs $C_1, C_2, \ldots, C_p$, to compute lower bounds on the I/O complexity of each sub-CDAG $IO(C_1), IO(C_2), \ldots, IO(C_p)$ independently and simply add them to bound the I/O complexity of C. This is stated in the following decomposition theorem, whose proof may be found in [14].

THEOREM 2 (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 $\bigcup_{i=1}^{p} V_i = V$) 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 $\sum_{i=1}^{p} IO(C_i) \leq IO(C)$. In particular, if LB_i is a lower bound on the I/O cost of C_i, then $\sum_{i=1}^{p} LB_i$ is a lower bound on the I/O cost of C.*

We state the following corollary and theorem, which are useful in practice for deriving tighter lower bounds. The complete proofs can be found in [14].

COROLLARY 1 (INPUT/OUTPUT DELETION). *Let C and C' be two CDAGs: $C' = (I \cup dI, V \cup dI \cup dO, E', O \cup dO)$, $C = (I, V, E' \cap V \times V, O)$. Then $IO(C')$ can be bounded below by $IO(C)$ as follows:*

$$IO(C) + |dI| + |dO| \leq IO(C') \tag{1}$$

There are cases where separating input/output vertices leads to very weak lower bounds. This happens when input vertices have high fan out such as for matrix-multiplication: if we consider the CDAG for matrix-multiplication and remove all input and output vertices, we get a set of independent chains that can each be computed with no more than 2 red pebbles. To overcome this problem, the following theorem allows us to compare the I/O of two CDAGs: a CDAG $C' = (I', V, E, O')$ and another $C = (I, V, E, O)$ built from C' by just transforming some vertices without predecessors into input vertices, and some others into output nodes so that $I' \subset I$ and $O' \subset O$. In contrast to the prior development above, instead of adding/removing input/output vertices, here we do not change the vertices of a CDAG but instead only change the labeling (tag) of some vertices as inputs/outputs in the CDAG. So the CDAG remains the same, but some input/output vertices are relabeled as standard computational vertices, or vice-versa.

THEOREM 3 (INPUT/OUTPUT (UN)TAGGING – RBW). *Let C and C' be two CDAGs of the same DAG $G = (V, E)$: $C = (I, V, E, O)$, $C' = (I \cup dI, V, E, O \cup dO)$. Then, $IO(C)$ can be bounded below by $IO(C')$ as follows (tagging):*

$$IO(C') - |dI| - |dO| \leq IO(C) \tag{2}$$

Reciprocally, $IO(C')$ can be bounded below by $IO(C)$ as follows (untagging):

$$IO(C) \leq IO(C') \tag{3}$$

Some algorithms will benefit from decomposing their CDAGs into non-disjoint vertex sets. For instance, when we have computations that are surrounded by an outer time loop, a common technique to derive their lower bound is to decompose the CDAG, where vertices computed during each outer loop iteration are placed in separate sub-CDAGs. In such cases, when the vertices, V', computed in iteration t are used as inputs for iteration $t+1$, by placing V' in the sub-DAGs corresponding to both iterations t and $t+1$, we could obtain a lower bound that is tighter by at least a constant factor.

Before we state the non-disjoint decomposition theorem, we introduce needed definitions here. Given a DAG $G = (V, E)$ and some vertex $x \in V$, the ancestor set, $\mathsf{Anc}(x)$ is the set of vertices from which there is a non-empty directed path to x in G ($x \notin \mathsf{Anc}(x)$); the descendant set, $\mathsf{Desc}(x)$ is the set of vertices to which there is a non-empty directed path from x in G ($x \notin \mathsf{Desc}(x)$). For some $V_i \subset V$, $\mathsf{InSet}(V_i)$ is the set of vertices of V_i, that have atleast one predecessor outside V_i.

THEOREM 4 (NON-DISJOINT DECOMPOSITION). *Let $C = (I, V, E, O)$ be a CDAG and $x \in V$ be some vertex. Let $V_2 = \mathsf{Desc}(x)$ and $C_2 = (I_2, V_2, E_2, O_2)$ be the induced sub-graph ($I_2 = I \cap V_2$, $E_2 = E \cap V_2 \times V_2$, $O_2 = O \cap V_2$). Let $V_a = V \setminus \mathsf{Desc}(x)$. The sub-graph $C_1 = (I_1 = I \cap V_1, V_1, E_1, O_1 = O \cap V_1)$ is built as follows: (1) start with vertices $V_1 = V_a \cup \mathsf{InSet}(\mathsf{Desc}(x))$ and edges $E_1 = E \cap V_a \times V_a$; (2) Add an edge from x to each vertex in $\mathsf{InSet}(\mathsf{Desc}(x))$, i.e., $\forall d \in \mathsf{InSet}(\mathsf{Desc}(x)), E_1 = E_1 \cup x \times d$. Then, $IO(C_1, S+1) + IO(C_2, S) \leq IO(C, S)$, where, $IO(C, S)$ represents the I/O cost for computing C with S red pebbles.*

PROOF. Consider a complete calculation $\mathcal{P}$ of C with S red pebbles. We let Q_{L1} be the number of R1 transitions (loads) in C associated to the vertices of $V \setminus [\mathsf{Desc}(x) + x]$. We let Q_{S1} be the number of R2 transitions (stores) in C associated to the vertices of $V \setminus \mathsf{Desc}(x)$. We let Q_2 be the number of R1 and R2 transitions (loads/stores) in C associated to a vertex in $\mathsf{Desc}(x)$. We have that $IO(C, S) >= Q_{L1} + Q_{S1} + Q_2$. The idea of the proof is to show that $Q_{L1} + Q_{S1} >= IO(C_1, S+1)$ and that $Q_2 >= IO(C_2, S)$.

Let us first prove that $Q_{L1} + Q_{S1} >= IO(C_1, S+1)$. We consider the restriction of $\mathcal{P}$ to V_1. This is not a complete calculation for C_1 yet, as the predecessors of the vertices in $\mathsf{InSet}(\mathsf{Desc}(x))$ need not be the same in C_1 as in C. We have one additional red pebble that we can dedicate to stay on x. Hence, all the R1 (load) and R4 (delete) transitions associated to vertex x, after the execution of x in $\mathcal{P}$, can be removed. This gives a complete calculation for C_1 as follows:

- As the sub-graph induced by V_a is a sub-graph of C, all transitions associated to vertices of $V \setminus [\mathsf{Desc}(x)+x]$ plus the transition R3 (compute) of x are valid (this part of the complete calculation from $\mathcal{P}$ has been unchanged).
- For the vertices in $\mathsf{InSet}(\mathsf{Desc}(x))$, the only transitions are R3/R2 (compute/ store) and is valid as all the associated transitions of its predecessors in V_a are unchanged (apart from x which holds a red pebble as soon as it is computed). The cost of this complete calculation (with $S+1$ red pebbles) for C_1 is $Q_{L1}+Q_{S1}$.

This proves the inequality.

Let us now prove that $Q_2 >= IO(C_2,S)$. We consider the restriction of C to the vertex of $C_2 = \mathsf{Desc}(x)$. This is a complete calculation for C_2 of cost Q_2 which proves the second inequality. □

3.3 Min-Cut for I/O Complexity Lower Bound

In [14], we developed an alternative lower bounding approach. It was 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, the min-cut based approach captures internal space requirements using the abstraction of wavefronts. This section describes the approach.

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 $\mathcal{S}$ and $\mathcal{T} = V - \mathcal{S}$. 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), i.e., the set of edges (u,v) where $u \in \mathcal{S}$ and $v \in \mathcal{T}$. 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 $(\mathcal{S},\mathcal{T})$ cut, as a vertex $u \in \mathcal{S}$ 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.

We consider a convex cut $(\mathcal{S}_x,\mathcal{T}_x)$ associated to x as follows: $\mathcal{S}_x$ includes $x \cup \mathsf{Anc}(x)$; $\mathcal{T}_x$ includes $\mathsf{Desc}(x)$; in addition, $\mathcal{S}_x$ and $\mathcal{T}_x$ must be constructed such that there is no edge from $\mathcal{T}_x$ to $\mathcal{S}_x$. With this, the sets $\mathcal{S}_x$ and $\mathcal{T}_x$ partition the graph G into two convex partitions. We define the wavefront induced by $(\mathcal{S}_x,\mathcal{T}_x)$ to be the set of vertices in $\mathcal{S}_x$ that have at least one outgoing edge to a vertex in $\mathcal{T}_x$.

Schedule Wavefront: Consider a complete calculation $\mathcal{P}$ that corresponds to some scheduling (i.e., execution) of the vertices of the graph $G = (V,E)$ that follows the rules R1–R4 of the Red-Blue-White pebble game (see Definition 4 in Sec. 3.1). 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 (i.e., a white pebble has just been placed on x) as the union of x and the set of vertices $u \in V$ that have already fired and that have an outgoing edge to a vertex $v \in V$ that have not fired yet. Viewing $\mathcal{P}$ as a string, $W_{\mathcal{P}}(x)$ is the set of vertices x and those white-pebbled vertices to the left of x in the string associated with $\mathcal{P}$ that have an outgoing edge in G to not-white-pebbled vertices that occur to the right of x in $\mathcal{P}$. With respect to a complete calculation $\mathcal{P}$, the set $W_{\mathcal{P}}(x)$ defines the memory requirements at the time-stamp 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. For a valid convex partition $(\mathcal{S}_x,\mathcal{T}_x)$ of G, we can construct a complete calculation $\mathcal{P}$ in which at the time-stamp when x has just fired, the subset of vertices of V that are white pebbled exactly corresponds to $\mathcal{S}_x$; the set of fired (white-pebbled) nodes that have a successor that is not white-pebbled constitute a wavefront $W_{\mathcal{P}}(x)$ associated with x. Similarly, given 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 white pebbled vertices in $\mathcal{S}_x$ and all the non-white-pebbled vertices in $\mathcal{T}_x$.

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

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

4. PARALLEL I/O LOWER BOUNDS

In this section, we develop an approach to model data movement complexity for parallel execution. We describe our abstraction of a parallel computer, define a pebble game adapted for characterizing lower bounds for parallel computation, and then present the methodology for developing parallel lower bounds.

4.1 Parallel Machine Model

Our abstraction of a parallel computer is shown in Fig. 1. The model seeks to capture the essential characteristics of large-scale parallel systems, which exhibit multi-level parallelism and a hierarchical storage structure. The parallel computer has a set of N_{nodes} multi-core nodes connected by an interconnection network. Each node 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. The total number of main memory modules N_L equals the number of nodes N_{nodes} in the system.

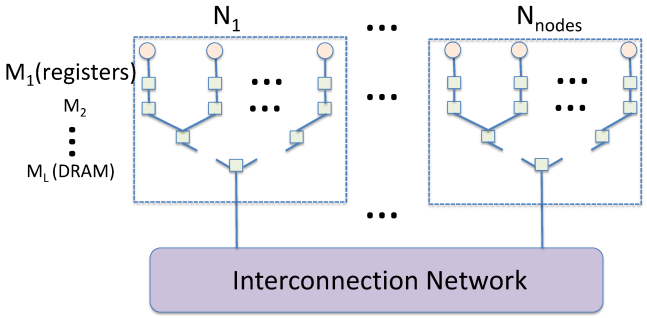

Figure 1: Model of parallel system

4.2 P-RBW: The Parallel Red-Blue-White Pebble Game

In this sub-section, we present the framework for developing lower bounds on the data movement complexity for parallel execution. In particular, we consider two types of data movement:

1. *Movement across the levels of the storage hierarchy within a node*, called **vertical data movement**;
2. *Movement between nodes*, called **horizontal data movement**.

The model used here may be viewed as an extension of the Multiprocessor Memory Hierarchy Game (MMHG) game proposed by Savage and Zubair [24], which modeled "vertical" data movement in a shared storage hierarchy but not the "horizontal" movement of data between the memories of nodes in the parallel system. Thus, whereas Savage's model assumes a common shared level of memory that can be directly accessed by all processors, we model a collection of shared-memory multiprocessor nodes coupled by an interconnection network. As demonstrated later in Sec. **??**, this distinction allows better modeling of fundamental data movement constraints of parallel algorithms on intra-node memory bandwidth (between off-chip main memory and on-chip cache(s) on a node) versus interconnection network bandwidth (e.g., Gigabit Ethernet of Infiniband used to connect nodes in a scalable parallel system).

With the Parallel RBW (P-RBW) 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 the P-RBW are stated below, and encode the constraints on movement of data in the parallel computer: i) vertical data movement can occur between physically connected entities in the storage hierarchy (Rules R4 and R5), and ii) data can be moved via the interconnection network between the memories of any pair of nodes (Rule R3).

DEFINITION 6 (PARALLEL RBW (P-RBW) PEBBLE GAME). *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 $R_l^1, R_l^2, \cdots, R_l^{N_l}$, respectively, and unlimited blue and white 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 white pebbles on all vertices and blue pebbles on all output vertices:*

- ***R1 (Input)*** *A level-L pebble, R_L^i can be placed on any vertex that has a blue pebble; a white pebble is also placed along with the shade of red pebble, unless the vertex already has a white pebble on it.*
- ***R2 (Output)*** *A blue pebble can be placed on any vertex that has a level-L pebble on it.*
- ***R3 (Remote get)*** *A level-L pebble, R_L^i can be placed on any vertex that has another level-L shade pebble R_L^j.*
- ***R4 (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+1}^j where R_l^i is in a cache that is a child of the cache that holds R_{l+1}^j.*
- ***R5 (Move down)*** *For $1 < l \leq L$, a level-l red pebble, R_l^j can be placed on any vertex that has a level-$(l-1)$ pebble R_{l-1}^i where R_{l-1}^i is in a cache that is a child of the cache that holds R_l^j.*
- ***R6 (Compute)*** *If a vertex v does not have a white pebble and all its immediate predecessors have level-1 red pebbles of shade p on them, then a level-1 red pebble R_1^p along with a white pebble may be placed on v; here p is the index of the processor that computes vertex v.*
- ***R7 (Delete)*** *Any shade of red pebble may be removed from any vertex (reuse storage).*

4.3 I/O Lower Bound for Vertical Data Movement

The hierarchical memory can enforce either the inclusion or exclusion policy. In case of inclusive hierarchical memory, when a copy of a value is present at a level-l, it is also maintained at all the levels $l+1$ and higher. These values may or may not be consistent with the values held at the lower levels. The exclusive cache, on the other hand, does not guarantee that a value present in the cache at level-l will be available at the higher levels. The following result is derived for the inclusive case. But, they also hold true for the exclusive case, where the difference lies only in the number of red pebbles that we consider in the corresponding two-level pebble game.

THEOREM 5 (VERTICAL I/O COST). *Let $C = (I,V,E,O)$ be a CDAG. Consider any complete calculation on C using the rules of P-RBW pebble game; for this complete calculation, consider the level-l storage j with the maximum number of R4/R5 transitions with its children at level-$(l-1)$. Then, the corresponding amount of data movement between the level-l storage j and its children is at least $IO_1(C, S_{l-1} \times N_{l-1})/N_l$, where $IO_1(C,S)$ is the I/O lower bound of C for a single processor with fast memory of size S.*

PROOF. Consider a complete calculation of C using the rules of P-RBW game that minimizes the overall amount of I/O between levels $k < l$ and level l storage. This amount of I/O will be bounded by $IO_1(C, S_{l-1} \times N_{l-1})$. Consider one of the N_l caches with the maximum amount of I/O. It will be bounded by $IO_1(C, S_{l-1} \times N_{l-1})/N_l$. □

It is possible to obtain tighter results for the cases that use S-partitioning techinique to derive the vertical I/O lower bounds. The following theorem extends the S-partitioning techinque to the vertical case.

THEOREM 6 (S-PARTITIONING BASED VERTICAL I/O COST). *Let $C = (I,V,E,O)$ be a CDAG. Consider any complete calculation on C using the rules of P-RBW game; for this complete calculation, consider the level-l storage j whose group of processors P_l^j perform the maximum number of R6 (compute) transitions. Then, the corresponding amount of data movement between the level-l storage j and its children is at least $\left(\frac{|V|}{U(C,2S_{l-1})\times N_l} - \frac{N_{l-1}}{N_l}\right) \times S_{l-1} \approx \frac{|V|\times S_{l-1}}{U(C,2S_{l-1})\times N_l}$, where, $|V|$ is the total number of vertices in C, $U(C,2S)$ is the largest vertex-set in any 2S-partition of C.*

PROOF. Consider a complete calculation that minimizes the overall amount of I/O between levels $k < l$ and level-l storage. Consider the group of P/N_l processors that do the maximum computation. They do at least $|V|/N_l$ amount of work. Let us consider the partition of those P/N_l processors into N_{l-1}/N_l sets of P/N_{l-1} processors that share the same level-$(l-1)$ storage unit. Each set of processors (that we denote by P_{l-1}^i, $0 < i \leq N_{l-1}/N_l$) does at least $\alpha^i \times \frac{|V|}{N_l}$ amount of work where $\sum_i \alpha^i = 1$. We let V^i be the subset of vertices of C fired by P_{l-1}^i.

Let us denote S_{l-1} by S to simplify the notations. The goal is to show that each P_{l-1}^i performs at least $\left[\left|V^i\right|/U(C,2S) - 1\right] \times S$

I/O to its level-l storage, where, $U(C,2S)$ is the largest 2S-partition (RBW pebble game) of CDAG C. Consider a complete calculation of C with RBW game with S red pebbles. Consider the partitioning of the complete calculation $\mathcal{P}$ into $\mathcal{P}_1,\cdots,\mathcal{P}_h$ used in the proof of Theorem 1 for RBW. We let V_j^i be the set of vertices of V^i fired in $\mathcal{P}_j$ ($\bigcup_{j=1}^{h} V_j^i = V^i$; $\forall_{j\neq j'} V_j^i \cap V_{j'}^i = \emptyset$). With the usual reasoning we can prove that $|\mathsf{In}(V_j^i)| \leq 2S$ and $|\mathsf{Out}(V_j^i)| \leq 2S$, i.e., each V_j^i is a $2S$-partition of C. Thus for each j, $|V_j^i| \leq U(C,2S)$. Now from a complete calculation for P-RBW game, we can build a complete calculation for RBW game. By construction, each V_j^i is associated to at least S I/O to level-l storage in the complete calculation for the P-RBW game. Thus the total amount of I/O for P_{l-1}^i is at least $\left[\left|V^i\right|/|V_j^i| - 1\right] \times S \geq \left[\left|V^i\right|/U(C,2S) - 1\right] \times S$. Finally, $\sum_{i=1}^{N_{l-1}/N_l} \left(\frac{|V^i|}{U(C,2S_{l-1})} - 1\right) \times S_{l-1} = \sum_{i=1}^{N_{l-1}/N_l} \left(\frac{\alpha_i \times (|V|/N_l)}{U(C,2S_{l-1})} - 1\right) \times S_{l-1}$ $= \left(\frac{|V|}{U(C,2S_{l-1}) \times N_l} - \frac{N_{l-1}}{N_l}\right) \times S_{l-1} \approx \frac{|V| \times S_{l-1}}{U(C,2S_{l-1}) \times N_l}$. □

4.4 I/O Lower Bound for Horizontal Data Movement

The following theorem extends the S-partitioning technique to the horizontal case.

THEOREM 7 (S-PARTITIONING BASED HORIZONTAL I/O COST). *Let $C = (I,V,E,O)$ be a CDAG. Consider any complete calculation on C using the rules of P-RBW game; for this complete calculation, consider the level-L storage i whose group of processors P_L^i perform the maximum number of R6 (compute) transitions. The corresponding amount of remote get transitions are at least $\left(\frac{|V|}{U(C,2S_L) \times N_L} - 1\right) \times S_L$, where, $|V|$ is the total number of vertices in C, $U(C,2S)$ is the largest vertex-set in any $2S$-partition of C.*

PROOF. We let V^i be the subset of vertices of C fired by the set of processors P_L^i. Let us denote S_L by S for simplicity. Consider a complete calculation $\mathcal{P}$ on C using RBW game with S red pebbles. Consider the partitioning of this complete calculation into $\mathcal{P}_1,\cdots,\mathcal{P}_h$ used in the proof of Theorem 1 for RBW. We let V_j^i be the set of vertices of V^i fired in $\mathcal{P}_j$ ($\bigcup_{j=1}^{h} V_j^i = V^i$; $\forall_{j\neq j'} V_j^i \cap V_{j'}^i = \emptyset$). With the usual reasoning we can prove that $\left|\mathsf{In}(V_j^i)\right| \leq 2S$ and $\left|\mathsf{Out}(V_j^i)\right| \leq 2S$, i.e., each V_j^i is a $2S$-partition of C. Thus for each j, $|V_j^i| \leq U(C,2S)$. Now from a complete calculation for P-RBW game, we can build a complete calculation for RBW game. By construction, each V_j^i is associated to at least S I/O operations in the complete calculation for P-RBW game. Thus the total amount of I/O for P_L^i is at least $\left[|V^i|/|V_j^i| - 1\right] \times S \geq \left[\left|V^i\right|/U(C,2S) - 1\right] \times S$.

Since the group P_L^i performs maximum number of computations, $\left|V^i\right| \geq |V|/N_L$. Hence, the total amount of remote get of processors P_L^i is at least $\left(\frac{|V|}{U(C,2S_L) \times N_L} - 1\right) \times S_L$. □

5. ILLUSTRATION OF USE

Lower and upper bound analysis of algorithms can help us identify whether an algorithm is bandwidth bound at different levels of the memory hierarchy. Lower bound results can be related to architectural parameters. Consider a multi-node/multi-core system with P processors. Let N_l be the total storage capacity (in data elements) available at level l. Consider a memory unit at level l, M_l^i, that incurs the maximum communication. M_l^i, is shared by the processor set P_l^i, such that $\left|P_l^i\right| = P/N_l$. Let $\mathcal{B}_l^i$ denote the total available memory bandwidth between M_l^i and all its children at level $l-1$.

Let $C = (I,V,E,O)$ be the CDAG of the algorithm being analyzed and $C_l^i \subset C$ be the sub-CDAG executed by the processors P_l^i. The time taken for execution of C is given by $T \geq \max(T_l^i, T_{comp})$, where, T_l^i denotes the communication time at M_l^i and T_{comp} denotes the computation time for C. For the algorithm to be not bound by memory bandwidth at level l,

$$T_l^i \leq T_{comp} \tag{4}$$

Let IO_l^i denote the amount of data transferred between M_l^i and all its children at level $l-1$ for the execution of C_l^i. Then,

$$T_l^i = \frac{IO_l^i}{\mathcal{B}_l^i} \geq \frac{LB_l^i}{\mathcal{B}_l^i} \tag{5}$$

where, LB_l^i denotes the lower bound on the amount of data transfer at memory unit M_l^i for any valid execution of C_l^i. The computation time of C is given by (F below indicates processor performance in floating-point arithmetic operations per second per core)

$$T_{comp} = \frac{W}{P} \times \frac{1}{F} \tag{6}$$

where, W is the total number of arithmetic operations. From Equations (4), (5) and (6), we have,

$$\frac{LB_l^i}{\mathcal{B}_l^i} \leq \frac{W}{P} \times \frac{1}{F} \quad \text{or} \quad \frac{LB_l^i}{W} \leq \frac{\mathcal{B}_l^i}{P} \times \frac{1}{F}$$

As $P = \left|P_l^i\right| \times N_l$,

$$\frac{LB_l^i \times N_l}{W} \leq \frac{\mathcal{B}_l^i}{\left|P_l^i\right| \times F} \tag{7}$$

The term at the right-hand side of Equation (7) is the machine balance value for the machine at level-l. Any algorithm that fails to satisfy the Equation (7), will be invariably bandwidth bound at level-l.

Through similar argument, given that UB_l^i is the upper bound on the minimum amount of data transfer required by the algorithm at memory unit M_l^i, we can show that if the algorithm is bandwidth bound, then it definitely satisfies the condition,

$$\frac{UB_l^i \times N_l}{W} \geq \frac{\mathcal{B}_l^i}{\left|P_l^i\right| \times F} \tag{8}$$

Hence, if an algorithm fails to satisfy Equation (8), we can safely conclude that there is at least one execution order of C that is not constrained by the memory bandwidth at level l.

In particular, we are interested in understanding the memory bandwidth requirements (1) between the main memory and last level cache (LLC) within each node, and, (2) between different nodes, for various algorithms. For simplicity, we assume that the LLC is shared by all the cores within a node, which is common in practice.

Considering the particular case of data movement between LLC and the main memory, Equation (7) becomes,

$$\frac{LB_{vert} \times N_{nodes}}{W} \leq \frac{\mathcal{B}_{vert}}{N_{cores} \times F} \tag{9}$$

where, LB_{vert} is the vertical data movement lower bound, $\mathcal{B}_{vert}$ is the total bandwidth between DRAM and LLC, N_{nodes} represents the number of nodes in the system, and N_{cores} represents the number of cores within each node. Similarly, considering the inter-node

communication, Equation (8) becomes,

$$\frac{UB_{horiz} \times N_{nodes}}{W} \geq \frac{\mathcal{B}_{horiz}}{N_{cores} \times F} \tag{10}$$

where, UB_{horiz} and $\mathcal{B}_{horiz}$ represent the upper bound on the horizontal data movement cost and inter-processor communication bandwidth, respectively.

Specifications for some of the computing systems are shown in table 1.

Table 1: Specifications of various computing systems

Machine	N_{nodes}	Mem. (GB)	LLC (MB)	Vertical balance (words / FLOP)	Horiz. balance (words / FLOP)
IBM BG/Q	2048	16	32	0.052	0.006
Cray XT5	9408	16	6	0.0256	0.005

5.1 Example: Solving the Heat Equation

Many compiute intensive applications involve the numerical solution of partial differential equations (PDEs). As an example, consider the heat flow on a long thin bar of unit length, of uniform material and insulated, so that heat can enter and exit only at the boundaries (Fig. 2(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, α is the thermal diffusivity of the bar. (For mathematical treatment, it is sufficient to consider $\alpha = 1$).

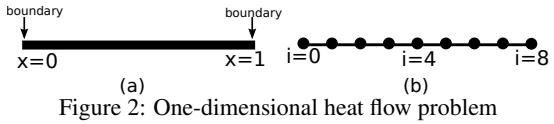

Figure 2: One-dimensional heat flow problem

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. 2(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 timestamp $m+1$.

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

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 timestamp till convergence. Each timestamp $m+1$ is dependant on values of the previous timestamp m.

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

$$\begin{pmatrix} 1+a & -\frac{a}{2} & & & & \\ -\frac{a}{2} & 1+a & -\frac{a}{2} & & & \\ & -\frac{a}{2} & 1+a & -\frac{a}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{a}{2} & 1+a & -\frac{a}{2} \\ & & & & -\frac{a}{2} & 1+a \end{pmatrix} \times \begin{pmatrix} U(1,m+1) \\ U(2,m+1) \\ U(3,m+1) \\ \vdots \\ U(n-1,m+1) \\ U(n,m+1) \end{pmatrix} = \begin{pmatrix} b(1,m) \\ b(2,m) \\ b(3,m) \\ \vdots \\ b(n-1,m) \\ b(n,m) \end{pmatrix} \tag{11}$$

where, $b(i,m)$ represents the right-hand side of the i-th equation at timestamp $m+1$. Solving this linear system of the form $\mathbf{Ax} = \mathbf{b}$ for vector $\mathbf{x}$ provides the solution to the original problem. 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.

Many iterative methods have been developed to efficiently solve such large linear systems of equations. The following section derives the vertical and horizontal data movement bounds for some of these iterative linear system solvers using the results from Sections 3 and 4 and compares it against the machine balance values.

5.2 Conjugate Gradient (CG)

The Conjugate Gradient method [15] is suitable for solving symmetric positive-definite linear systems. CG maintains 3 vectors at each timestep - 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 pseudocode is shown in Fig. 3.

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

Figure 3: Conjugate Gradient method

5.2.1 Vertical data movement cost

In this sub-section, we provide the lower bound for the amount of data movement between different levels of hierarchy for CG.

THEOREM 8 (MIN-CUT BASED I/O LOWER BOUND FOR CG). *For a d-dimensional grid of size n^d, the minimum I/O cost to solve the linear system using CG, Q, satisfies $Q \geq 6n^dT/P$, when $n^d \gg S$; where, T represents the number of outer loop iterations, and P is the number of processors.*

PROOF. Let C be the CDAG for CG. Consider the vertex υ_x, corresponding to computation of the scalar a at line 6. The $2n^d$ predecessor vertices of υ_x, corresponding to elements of vectors $\mathbf{p}$ and $\mathbf{v}$, have disjoint paths to the $\mathsf{Desc}(\upsilon_x)$ (due to computations in lines 7 and 8, respectively). This gives us a wavefront of size $\left|W_G^{min}(\upsilon_x)\right| = 2n^d$. Similarly, considering the vertex, υ_y, corresponding to the computation of scalar g, at line 9, we obtain a

wavefront of size $\left|W_G^{min}(\upsilon_y)\right| = n^d$, due to the disjoint paths from the predecessors **r** to $\mathsf{Desc}(x)$ (due to the computation at line 10).

Recursively applying Theorem 4 on the complete CDAG C, provides us T sub-CDAGs, $C_1, C_2, \ldots, C_T$, corresponding to each outer loop iteration. (Vertices corresponding to the elements of vector **p** computed at line 10 are shared between neighboring sub-CDAGs). Further, non-disjointly sub-dividing each of these sub-CDAGs, C_i, into $C_{i_{|x}} and C_{i_{|y}}$ (vertices corresponding to the computation of vector **r** from line 8 are shared between $C_{i_{|x}} and C_{i_{|y}}$), to decompose the effects of wavefronts $W_G^{min}(\upsilon_x)$ and $W_G^{min}(\upsilon_y)$, we obtain a lower bound of,

$$\begin{aligned} Q &\geq T \times (2(2n^d - S)) + T \times (2(n^d - S)) \\ &= T \times (2(3n^d - 2S)) \end{aligned}$$

which tends to $6n^dT$ when $n^d \gg S$. Finally, application of Theorem 5 provides a lower bound of $6n^dT/P$ for the parallel case. □

5.2.2 Horizontal data movement cost

Consider the block partitioning of the input grid among the processors, with block size along each dimension $B = n/N_{nodes}^{1/d}$. Each processor holds the input data corresponding to its local grid points and computes the data needed by those grid points. Computation of the sparse matrix-vector product, at line 4 in Fig. 3, requires send and receive of values at the ghost cells of size $(B+2)^d - B^d$ at each timestep with the neighboring processors. If Q is the minimum I/O cost for executing CG, then,

$$\begin{aligned} Q &\leq 2 \times ((B+2)^d - B^d) \times T \\ &= 2 \times (B^d + \binom{d}{1} B^{d-1}2^1 + \binom{d}{2} B^{d-2}2^2 \\ &\quad + \cdots + \binom{d}{d-1} B^1 2^{d-1} + \binom{d}{d} B^0 2^d - B^d) \times T \\ &= O(4dB^{d-1}T) \end{aligned}$$

5.2.3 Analysis

Equations (9) and (10) provided us conditions to determine the vertical and horizontal memory constraints of the algorithms. We will use them to show that the running time of CG is mainly constrained by the vertical data movement. We consider a three-dimensional problem ($d = 3$) for the analysis.

Operation count: The vector dot-product at line 6 requires $2n^d$ operations. The computation of $(\mathbf{r}.\mathbf{r})$, 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 $24n^3T$ for a three-dimensional problem.

The vertical I/O lower bound per node, $LB_{vert} = 6n^3T/N_{nodes}$. Hence,

$$\frac{LB_{vert} \times N_{nodes}}{W} = \frac{\left(6n^3T/N_{nodes}\right) \times N_{nodes}}{24n^3T} = \frac{6}{24} = 0.25$$

This value is higher than the vertical machine balance value of various machine (refer table 1), leaving Equation (9) unsatisfied. This shows that CG will be unavoidably bandwidth bound along the vertical direction for the problems that cannot fit into the cache. The only way to improve the performance would be to increase the main memory bandwidth.

On the other hand, let us consider the horizontal data movement cost.

$$\begin{aligned} \frac{UB_{horiz} \times N_{nodes}}{W} &= \frac{12B^2T \times N_{nodes}}{24n^3T} \\ = \quad \frac{12\left(n/N_{nodes}^{(1/3)}\right)^2 N_{nodes}}{24n^3} &= \frac{N_{nodes}^{(1/3)}}{2n} \end{aligned}$$

This value is much lower than the horizontal machine balance values of various machines for the problem size typically encountered in practice, indicating that it is the intra-node memory bandwidth that is much more of a fundamental bottleneck than inter-node communication for CG.

5.3 Jacobi Method

Jacobi's method involves stencil computation, starting with an initial guess for the unknown vector **x** and iteratively replacing the current approximate solution at each grid point by a weighted average of its nearest neighbors on the grid. Hence, the information at one grid point can only propogate to its adjacent grid points in one iteration. Thus, it takes at least n steps to propogate the information throughout the grid and reach to the solution.

5.3.1 Vertical data movement cost

In this section, we derive the I/O lower bound for Jacobi computation on a d-dimensional grid. We provide the proof for a 2D-grid below, which can be generalized to a grid of d-dimensions as shown later.

THEOREM 9 (I/O LOWER BOUND FOR JACOBI).
For the two-dimensional 5-points Jacobi computation of size $n \times n$ with $T-1$ time steps, the minimum I/O cost, Q, satisfies $Q \geq \frac{N^2T}{4P\sqrt{2S}}$, where, P is the number of processors.

PROOF. The CDAG for the Jacobi computation has the property that all inputs can reach all outputs through vertex-disjoint paths. These vertex-disjoint paths will be called *lines*, for simplicity. Let $F(d)$ denote a monotonically increasing function such that for any two vertices u and v on the same line that are at least distance d apart, $F(d)$ has the following properties: (1) none of these $F(d)$ vertices belong to the same line; (2) Each of these vertices belongs to a path connecting u and v. In [16, Theorem 5.1], Hong & Kung showed that for any CDAG C that satisfies the above mentioned properties, its largest vertex-set in any $2S$-partition of C, $U(C,2S) = 2.(F^{-1}(2S)+1)$, and the serial I/O lower bound, Q_s can be bounded by $Q_s \geq L/(2.(F^{-1}(2S)+1))$, where L is the total number of vertices on the lines. From the structure of CDAG for 2D-Jacobi computation, it can be seen that $F^{-1}(2S) = 2\sqrt{2S} - 1$. Hence, $U(C,2S) = 4\sqrt{2S}$, and from Theorem 6, the parallel I/O cost, $Q \geq n^2T/(4P\sqrt{2S})$. □

With the similar reasoning, the I/O lower bound can be extended to higher dimensions, leading to the I/O cost of $Q \geq n^dT/(4P(2S)^{1/d})$, for a d-dimensional grid. The well-known tiled Jacobi implementation (of tile size S) with scheduling for wavefront parallelism has an I/O cost of $Q \leq (4d+2)n^dT/(PS^{1/d})$. Hence, the parallel I/O lower bound derived above is asymptotically tight.

5.3.2 Horizontal data movement cost

The well-known distributed memory tiled Jacobi implementation incurs data communication cost at the boundaries due to the exchange of ghost cell values, similar to the Conjuagte Gradient method. Hence, the upper bound on the horizontal data movement

cost for the Jacobi method, $Q = O(4dB^{d-1}T)$, where, $B = n/N_{nodes}^{1/d}$ is the block size along each dimension.

5.3.3 Analysis

We consider a three-dimensional problem ($d = 3$) for the analysis in this section. The operation count for a three-dimensional 7-point Jacobi method is $7n^3T$. The vertical I/O lower bound per node, $LB_{vert} = n^3T/(4(2S)^{1/3}N_{nodes})$. Hence,

$$\frac{LB_{vert} \times N_{nodes}}{W} = \frac{\left(n^3T/4.(2S)^{1/3}.N_{nodes}\right) \times N_{nodes}}{7n^3T} = \frac{1}{35.3 \times S^{1/3}}$$

For an LLC of size $S = 6$ MB (i.e., 0.75 MWords), $LB_{vert} = 3 \times 10^{-4}$ words/FLOP. This value falls below the vertical machine balance parameter of various architectures (refer Table 1). Now, considering the vertical I/O upper bound per node, $UB_{vert} = 14n^dT/(S^{1/d}N_{nodes})$.

$$\frac{UB_{vert} \times N_{nodes}}{W} = \frac{\left(14n^3T/S^{1/3}.N_{nodes}\right) \times N_{nodes}}{7n^3T} = \frac{2}{S^{1/3}}$$

For $S = 0.75$ MWords, $UB_{vert} = 0.022$ words/FLOP. This value falls sightly below the vertical machine balance parameters shown in Table 1, showing that the Jacobi method need not be necessarily bandwidth bound along the vertical direction, as opposed to CG.

Similarly, for the horizontal case,

$$\frac{UB_{horiz} \times N_{nodes}}{W} = \frac{12B^2T \times N_{nodes}}{7n^3T} = \frac{1.714 \times N_{nodes}^{(1/3)}}{n}$$

As in the case of CG, this value is much lower than the horizontal machine balance values of different machines for the common problem sizes, indicating that the horizontal data movement is not a bottleneck.

This shows that even though Jacobi method might require more iterations to converge, its lower ratio of I/O cost to the computational cost along both vertical and horizontal directions might make methods similar to it a more attractive alternative to CG for solving large sparse systems of linear equations on future machines that have more skewed machine balance parameters than current systems.

6. RELATED WORK

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 [16]. Their $2S$-partitioning approach uses dominators of incoming edges to partitions but does not account for the internal structure of partitions. In this paper, in addition to using the $2S$-partitioning technique, we also use an alternate lower bound approach that models the internal structure of CDAGs, and uses graph mincut as the basis. In addition, Hong & Kung's original model does not lend itself easily to development of effective lower bounds for a CDAG from bounds for component sub-graphs. With a change of the pebble game model to the RBW game, we were able to use CDAG decomposition to develop tight composite lower bounds for inhomogeneous CDAGs.

Several works followed Hong & Kung's work on I/O complexity in deriving lower bounds on data accesses [2, 1, 18, 6, 5, 22, 23, 19, 20, 28, 13, 3, 4, 8, 27, 25]. Aggarwal et al. provided several lower bounds for sorting algorithms [2]. Savage [22, 23] developed the notion of S-span to derive Hong-Kung style lower bounds and that model has been used in several works [19, 20, 25]. Irony et al. [18] 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. More recently, Demmel et al. 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 [3, 4, 13, 27]. Bilardi et al. [6, 5] develop the notion of access complexity and relate it to space complexity. Bilardi and Preparata [7] developed the notion of the closed-dichotomy size of a DAG G that is used to provide a lower bound on the data access complexity in those cases where recomputation is not allowed. Our notion of schedule wavefronts is similar to the closed-dichotomy size in their work. Extending the scope of the Hong & Kung model to more complex memory hierarchies has been the subject of some 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 [22]. Valiant proposed a hierarchical computational model [28] that offers the possibility to reason in an arbitrarily complex parameterized memory hierarchy model.

Unlike Hong & Kung's original model, several models have been proposed that do not allow recomputation of values (also referred to as "no repebbling") [3, 4, 5, 26, 19, 22, 23, 25, 9, 18, 20, 21]. Savage [22] develops results for FFT using no repebbling. Bilardi and Peserico [5] explore the possibility of coding a given algorithm so that it is efficiently portable across machines with different hierarchical memory systems, without the use of recomputation. Ballard et al. [3, 4] assume no recomputation is allowed in deriving lower bounds for linear algebra computations. Ranjan et al. [19] develop better bounds than Hong & Kung for FFT using a specialized technique adapted for FFT-style computations on memory hierarchies. Ranjan et al. [20] derive lower bounds for pebbling r-pyramids under the assumption that there is no recomputation. Recently, Ranjan et al. [21] developed a technique for binomial graphs. Very recent work from U.C. Berkeley [8] has developed a very novel approach to developing parametric I/O lower bounds 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.).

The P-RBW game developed in this paper extends the parallel model for shared-memory architectures by Savage and Zubair [24] to also include the distributed-memory parallelism present in all scalable parallel architectures. The works of Irony et al. [18] and Ballard et al. [3] model communication across nodes of a distributed-memory system. Bilardi and Preperata [7] develop lower bound results for communication in a distributed-memory model specialized for multi-dimensional mesh topologies. Our model in this paper differs from the above efforts in defining a new integrated pebble game to model both horizontal communication across nodes in a parallel machine, as well as vertical data movement through a multi-level shared cache hierarchy within a multi-core node.

Czechowski et al. [11, 12] consider the relationship between the ratio of an algorithm's data movement cost to arithmetic work and the machine balance ratio of memory bandwidth to peak performance. Our analysis of algorithms in this paper involves a very similar theme as theirs, but we develop new lower bounds analyses. Further, we compare and contrast data movement demands for horizontal across-node communication versus vertical within-node data movement and observe that the latter is often the more constraining factor.

7. CONCLUSION

Characterizing the parallel data movement complexity of a program is a cornerstone problem, that is particularly important with current and emerging power-constrained architectures where the

data transfer cost will be the dominant energy and performance bottleneck. In this paper, we have presented an extension to the Hong and Kung red-blue pebble game model to enable development of lower bounds on data movement for parallel execution of CDAGs. The model distinguishes horizontal data movement between nodes of a distributed-memory parallel system from vertical data movement within the multi-level memory/cache hierarchy within a multi-core node. The utility of the model and the developed lower bounding techniques was demonstrated by analysis of several numerical algorithms and the garnering of interesting insights on the relative significance of horizontal versus vertical data movement for different algorithms.

Acknowledgments.

This work is supported in part by the U.S. National Science Foundation through awards 0811457, 0904549, 0926127, 0926687, and 0926688, by the U.S. Department of Energy through award DE-SC0008844, and by the U.S. Army through contract W911NF-10-1-000.

8. REFERENCES

[1] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir. A model for hierarchical memory. In *19th STOC*, pages 305–314, 1987.

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Analysis Applications*, 32(3):866–901, 2011.

[4] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6):32, 2012.

[5] G. Bilardi and E. Peserico. A characterization of temporal locality and its portability across memory hierarchies. *Automata, Languages and Programming*, pages 128–139, 2001.

[6] G. Bilardi, A. Pietracaprina, and P. D'Alberto. On the space and access complexity of computation dags. In *Graph-Theoretic Concepts in Computer Science*, volume 1928 of *LNCS*, pages 81–92. 2000.

[7] G. Bilardi and F. P. Preparata. Processor - Time Tradeoffs under Bounded-Speed Message Propagation: Part II, Lower Bounds. *Theory Comput. Syst.*, 32(5):531–559, 1999.

[8] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick. Communication lower bounds and optimal algorithms for programs that reference arrays - part 1. EECS Technical Report EECS–2013-61, UC Berkeley, May 2013.

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

[10] Cray XE6. http://www.cray.com/Products/Computing/XE.aspx.

[11] K. Czechowski, C. Battaglino, C. McClanahan, A. Chandramowlishwaran, and R. Vuduc. Balance principles for algorithm-architecture co-design. In *Proceedings of the 3rd USENIX Conference on Hot Topic in Parallelism*, HotPar'11, pages 9–9, 2011.

[12] K. Czechowski and R. Vuduc. A theoretical framework for algorithm-architecture co-design. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 791–802, 2013.

[13] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Scientific Computing*, 34(1), 2012.

[14] V. Elango, F. Rastello, L.-N. Pouchet, J. Ramanujam, and P. Sadayappan. Data access complexity: The red/blue pebble game revisited. Technical Report OSU-CISRC-7/13-TR16, Ohio State University, September 2013.

[15] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems, 1952.

[16] J.-W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. of the 13th annual ACM sympo. on Theory of computing (STOC'81)*, pages 326–333. ACM, 1981.

[17] IBM Blue Gene team. The ibm blue gene project. *IBM Journal of Research and Development*, 57(1/2):0:1–0:6, 2013.

[18] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[19] D. Ranjan, J. Savage, and M. Zubair. Strong I/O lower bounds for binomial and FFT computation graphs. In *Computing and Combinatorics*, volume 6842 of *LNCS*, pages 134–145. Springer, 2011.

[20] D. Ranjan, J. E. Savage, and M. Zubair. Upper and lower I/O bounds for pebbling r-pyramids. *J. Discrete Algorithms*, 14:2–12, 2012.

[21] D. Ranjan and M. Zubair. Vertex isoperimetric parameter of a computation graph. *Int. J. Found. Comput. Sci.*, 23(4):941–, 2012.

[22] J. Savage. Extending the Hong-Kung model to memory hierarchies. In *Computing and Combinatorics*, volume 959 of *LNCS*, pages 270–281. 1995.

[23] J. E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.

[24] J. E. Savage and M. Zubair. A unified model for multicore architectures. In *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, page 9. ACM, 2008.

[25] J. E. Savage and M. Zubair. Cache-optimal algorithms for option pricing. *ACM Trans. Math. Softw.*, 37(1), 2010.

[26] M. Scquizzato and F. Silvestri. Communication lower bounds for distributed-memory computations. *CoRR*, abs/1307.1805, 2013.

[27] E. Solomonik, A. Buluç, and J. Demmel. Minimizing communication in all-pairs shortest paths. In *IPDPS*, 2013.

[28] L. G. Valiant. A bridging model for multi-core computing. *J. Comput. Syst. Sci.*, 77:154–166, Jan. 2011.