A I/O Lower Bounds for Arbitrary CDAGs

In this section we shortly introduce a general mathematical machinery we use to proof the I/O optimality of COMM. We extend Lemma **??** by Hong and Kung [3], which provides a method to find an I/O lower bound for a given CDAG. This lemma, however, does not give a tight bound, as it overestimates a *reuse set* size (cf. Lemma 2). Our key result here, Lemma 2 allows us to derive a constructive proof of a tighter I/O lower bound for a sequential execution of MMM CDAG (appendix B). We use this result in (**??**) to prove the parallel I/O optimality of COMM.

Our method heavily relies on a red-blue pebble game (Definition 1) and an S-partition abstractions (Definition 2) introduced by Hong and Kung.

A.1 Red-Blue Pebble Game

A red-blue pebble game is an abstraction modeling an execution of an algorithm in a two-level memory structure with a small-and-fast as well as large-and-slow memory. A red (or a blue) pebble placed on a vertex of a CDAG denotes that this data is inside a fast (or slow) memory. It is a powerful tool, extensible to arbitrarily many memory levels [6], that was used to derive lower bounds for algorithms such as sorting or FFT [3].

Intuition In the red-blue pebble game, a red (or blue) pebble placed on a vertex denotes that its value is inside the fast (or slow) memory. The actual computation (referred to as *pebbling*) is a series of allowed moves (e.g., moving a pebble from one vertex to another) that correspond to load, store, compute, or freeing-memory operations. The *I/O cost of a computation* is the number of pebble moves that correspond to loads and stores between the slow and the fast memory; finding the pebbling that minimizes this cost is PSPACE-complete [1, 4].

Definition 1 (Red-Blue Pebble Game [3]). *Let $G = (V, E)$ be a CDAG. In the initial/terminal configuration, only inputs/outputs of the CDAG have blue pebbles. There can be at most S red pebbles used. A complete CDAG computation is a sequence of moves that lead from the initial to the terminal pebble configuration. The allowed moves are as follows: ① placing a red pebble on any vertex with a blue pebble (load), ② placing a blue pebble on any vertex with a red pebble (store), ③ placing a red pebble on a vertex whose parents have all red pebbles (compute), ④ removing any pebble (red or blue) from any vertex (freeing memory).*

An I/O optimal execution of a CDAG corresponds to a sequence of moves (called *pebbling* of a graph) which minimizes load ① and store ② moves.

Connections to MMM For a CDAG of MMM, an example pebbling moves include placing red pebbles on some vertices corresponding to elements of matrices A and B (load operations), placing red pebbles on the corresponding elements of C (compute), removing red pebbles from the loaded inputs (freeing memory) and placing blue pebbles on the computed elements of C (store).

A.2 S-Partitions

The notion of an *S-partition* facilitates deriving lower bounds on the I/O computation cost [3]. Here, one divides a given CDAG into consecutive *subcomputations*, each of which requires at least S load and store operations. The key element in more straightforward lower bound proofs is to analytically bound the size (vertex count) of the largest subcomputation, given its input and output size (i.e., the number of vertices outside (inside) the subcomputation that have a child inside (outside) of it). Formally:

Definition 2 (S-partition of a CDAG [3]). *Let $G = (V, E)$ be a CDAG. An S-partition of G is a collection $\{V_1, ..., V_h\}$ of h subcomputations of V such that: ① $V_i \cap V_j = \emptyset$ and $\bigcup_{i=1}^{h} V_i = V$ for any $1 \le i, j \le h$, ② $\forall i \quad |Dom(V_i)| \le S$, ③ $\forall i \quad |Min(V_i)| \le S$, and ④ there is no cyclic dependence between subcomputations.*

$Dom(V_i) \not\subset V_i$ is the *dominator set*: a set of vertices such that every path from an input of the CDAG to a vertex in V_i contains some vertex in in $Dom(V_i)$. $Min(V_i) \subset V_i$ is the *minimum set* of V_i: it contains vertices that do not have any children in V_i. Finally, $H(S)$ is the cardinality of the smallest valid S-partition of a given CDAG.

We use a symbol $\mathcal{S}(S) = \{V_1, \dots, V_h\}$ to denote an S-partition.

Connections to MMM In MMM, a subcomputation V_i is a calculation of partial sums of C that can be computed with at most S elements of A and B, and that contributes to at most S outputs. Those elements from A and B, as well as previous values of C being updated, form $Dom(V_i)$. Then, $Min(V_i)$ corresponds to the result of this subcomputation (cf. Figure **??**). $H(S)$ denotes the number of such subsets required to calculate the final result. Assuming that each subcomputation computes the same number of partial results $\forall_{i,j} |V_i| = |V_j|$, and observing the total number of partial results $|V| = mnk$, we have $H(S) = \frac{mnk}{V_i}$.

A.3 Existing General I/O Lower Bound

We now cite a *general* lower bound on the cost of *any* I/O computation [3] and sketch the proof, which is the basis for our *tighter general* bound on the I/O cost (Lemma 2).

Intuition The key notion in the existing bound is to use $2S$-partitions for a given fast memory size S. For some subcomputation V_i, if $|Dom(V_i)| = 2S$ vertices, then at most S of them could contain a red pebble before V_i begins. Thus, at least S additional pebbles need to be loaded from the memory. The similar argument goes for $Min(V_i)$. Therefore, knowing the lower bound on the number of sets V_i in a valid $2S$-*partition*, together with the observation that each V_i performs at least S I/O operations, we have:

Lemma 1 (Lower bound on the number of I/Os [3])**.** *The minimal number Q of I/O operations for any valid execution of a CDAG of any I/O computation is bounded by*

$$Q \geq S \cdot (H(2S) - 1) \tag{1}$$

Details Assume that we know the optimal schedule of the CDAG. Divide the computation into h consecutive subcomputations $V_1, V_2, ..., V_h$, such that during the execution of V_i, $i < h$, there are exactly S I/O operations, and in V_h there are at most S operations. Now, for each V_i, we define two subsets of V, $V_{R,i}$ and $V_{BR,i}$. $V_{R,i}$ contains vertices that have red pebbles placed on them just before V_i begins. $V_{BR,i}$ contains vertices that have blue pebbles placed on them just before V_i begins, and have red pebbles placed on them during V_i. Using these definitions, we have: ❶ $V_{R,i} \cup V_{BR,i} = Dom(V_i)$, ❷ $|V_{R,i}| \leq S$, ❸ $|V_{BR,i}| \leq S$, and ❹ $|V_{R,i} \cup V_{BR,i}| \leq |V_{R,i}| + |V_{BR,i}| \leq 2S$. We define similar subsets $W_{BR,i}$ and $W_{R,i}$ for the minimum set $Min(V_i)$. $W_{BR,i}$ contains all vertices in V_i that have a blue pebble placed on them during V_i, and $W_{R,i}$ contains all vertices in V_i that have a red pebble at the end of V_i. By the definition of V_i, $W_{BR,i} \leq S$, by the constraint on the red pebbles, we have $W_{R,i} \leq S$, and by te definition of the minimum set,$Min(V_i) \subset W_{R,i} \cup W_{BR,i}$. Finally, by Definition 2, $V_1, V_2, ..., V_h$ form a valid $2S$-partition of the CDAG.

A.4 Tighter General I/O Lower Bounds

A.4.1 Data Reuse

A more careful look at the sets $V_{R,i}, V_{BR,i}, W_{R,i}$ and $W_{RB,i}$ allows us to refine the bound on the number of I/O operations on a CDAG. By its definition, $V_{BR,i}$ is a set of vertices on which we perform move ① (load). On the other hand, $V_{R,i}$ is the set of vertices that were already computed (red-pebbled) during some $V_j, j < i$, and are reused during V_i (the move ① is not needed). We call $V_{R,i}$ a *reuse set* of V_i. Similarly, by its definition, $W_{BR,i}$ contains all the vertices on which we perform move ② (store) during V_i. Therefore, if Q_i is the number of I/O operations during the subcomputation V_i, then we have $Q_i \geq |V_{BR,i}| + |W_{BR,i}|$. The trivial bounds are $V_{R,i} leS$ and $W_{BR,i} \geq 0$, but if one can show that $\exists_{R(S) \in \mathbb{Z}} : \forall_i : V_{R,i} \leq R(S) < S$ or $\exists_{T(S) \in \mathbb{Z}} : \forall_i : W_{BR,i} \geq T(S)$, we can use $R(S)$ and $T(S)$ to tighten a bound on Q. We call $R(S)$ a *maximum reuse* and $T(S)$ a *minimum store size* of a CDAG.

A.4.2 Reuse-based Lemma

We now enhance the general I/O lower bound *by tightening the bound on the set* $V_{R,i}$, that is, the *data reuse between the computations* bounds. We later show the schedule attaining this bound (appendix B) and we use this schedule to minimize horizontal communication between processes in a distributed MMM computation (**??**). Our main result in this section is as follows:

Lemma 2. *The minimal number Q of I/O operations for any valid execution of a CDAG $G = (V, E)$ is bounded by*

$$Q \geq (S - R(S) + T(S)) \cdot (H(S) - 1) \tag{2}$$

$R(S)$ is the maximum reuse and $T(S)$ is the minimum store size (appendix A.4.1). Moreover:

$$H(S) \geq \frac{|V|}{|V_{max}|} \tag{3}$$

where $V_{max} = \arg\max_{V_i \in \mathcal{S}} |V_i|$ is the largest subset of vertices in $\mathcal{S}(S)$.

Proof. To prove Eq. (2), we use analogous reasoning as in Lemma **??**. We associate the optimal pebbling with h consecutive subcomputations $V_1, \ldots V_h$ with the difference that each subcomputation V_i performs $Q_{i,s}$ store and $Q_{i,l}$ load operations, such that $S - R(S) \leq Q_{i,s} \leq S$ and $T(S) \leq Q_{i,l} \leq S$ for each V_i. We now define sets $V_{R,i}, V_{BR,i}, W_{R,i}$ and $W_{RB,i}$ analogously as in the previous proof. we first observe:

$$\forall_i : (S - R(S)) \leq |V_{BR,i}| \leq S$$
$$\forall_i |V_{R,i}| \leq R(S)$$
$$R(S) \leq S$$

Using the fact that $V_{R,i} \cup V_{BR,i} = Dom(V_i)$ (❶ from appendix A.3):

$$\begin{gathered} |Dom(V_i)| = |V_{R,i}| + |V_{BR,i}| \\ |Dom(V_i)| \leq R(S) + (S - R(S)) \\ |Dom(V_i)| \leq S \end{gathered} \tag{4}$$

By making analogous construction for the store operations, we show that $V_1 \dots V_h$ form a valid S-partition $\mathcal{S}(S)$. Therefore, a schedule performing $Q \geq (S - R(S) + T(S))h$ operations has an associated $\mathcal{S}(S)$, such that $|\mathcal{S}(S)| = h$. By the definition $H(S) = \min_{\mathcal{S}(S)} |\mathcal{S}(S)|$, we have $Q \geq (S - R(S) + T(S)) \cdot (H(S) - 1)$.

To prove Eq. (3), observe that V_{max} by definition is the largest subset in the optimal S-partition. As the subsets are disjoint, any other subset covers fewer remaining vertices to be pebbled than P_{max}. Because there are no cyclic dependencies between subsets, we can order them topologically as $V_1, V_2, \dots V_{H(S)}$. To ensure correct indices, we also define $V_0 \equiv \emptyset$. Now, define W_i to be the set of vertices not included in any subset from 1 to i, that is $W_i = V - \bigcup_{j=1}^{i} V_j$. Clearly, $W_0 = V$ and $W_{H(S)} = \emptyset$. Then, we have

$$\forall_i \quad |V_i| \leq |V_{max}|$$
$$|W_i| = |W_{i-1}| - |V_i| \geq |W_{i-1}| - |V_{max}| \geq i|V_{max}|$$
$$|W_{H(S)}| = 0 \geq H(S) \cdot |V_{max}|$$

that is, after $H(S)$ steps, we have $H(S)|V_{max}| \geq |V|$. □

A.4.3 Computational Intensity

For graphs of parametric sizes (e.g., MMM graph has $mnk + mk + kn$ vertices), we need a tool to allow us to bound the I/O complexity of the whole graph by bounding the maximum size of a single subcomputation. We provide an observation that connects the minimal number Q of I/O operations (cf. Eq. (2)) and a notion of *computational intensity*. Define computational intensity of the subcomputation V_i as $\rho_i = \frac{|V_i|}{V_{BR,i} + W_{BR,i}}$. Intuitively, computational intensity is the ratio of the number of computed elements ($|V_i|$) and the number of loaded and stored vertices ($V_{BR,i} + W_{BR,i}$). Having the bounds $S - R(S)$ and $T(S)$ on $V_{BR,i}$ and $W_{BR,i}$, respectively, and inserting Inequality 3 to 2, we derive the following corollary:

Corollary (Computational intensity). *Denote $\rho = \max_i(\rho_i) \leq \left(\frac{|V_{max}|}{S - R(S) + T(S)}\right)$. Then, the number of I/O operations Q is bounded by:*

$$Q \geq \frac{|V|}{\rho} \tag{5}$$

A.4.4 $2S$-Partition vs S-Partition: An MMM Example

To show why Lemma 2 gives a tighter bound than Lemma 1, consider a CDAG of MMM. We can draw it in a 3D iteration space $\mathcal{V}$. (??, ??). Then, an S-partition is a decomposition of this space into subcomputations V_i whose number of inputs ($|\alpha_i| + |\beta_i| + |\gamma_i|$) is smaller than S. When a subcomputation is viewed as a subspace in the iteration space $V_i \in \mathcal{V}$, this input constraint may be viewed as a generalization of the Loomis-Whitney inequality [5], which relates a cardinality of a finite set in $\mathbb{Z}^t$ with cardinalities of its projections to all t dimensions. This technique was used by Irony et al. [2] to derive first parallel I/O MMM lower bounds.

Assume that each subcomputation forms a cuboid $[a \times b \times c]$ in this iteration space (we actually prove it in **??**). Its faces form the dominator set. Now based on Lemma 1, we construct a $2S$-partition with a minimal cardinality, deriving subcomputations of a cubic ($a = b = c$) shape (Figure **??** b), with a cube side $a = \sqrt{2S/3}$. Such schedule will perform $2a^2 \frac{n^3}{a^3} = \sqrt{\frac{3}{2}} \frac{2N^3}{\sqrt{S}}$ I/O operations.

Now, the question is: what is the size of a maximum reuse $R(S)$? Observe that only one of three faces of this cuboid can be reused (used by a subsequent subcomputation while still being in fast memory). This observation will help us derive in (**??**) a "flat" shape (Figure **??** c) which performs $\frac{2N^3}{\sqrt{S}}$ I/O operations - an $\sqrt{3/2}$ improvement.

B Optimal Sequential Matrix Multiplication

We now proceed to establish a tight sequential I/O lower bound of MMM. Using the mathematical machinery of Lemma 2, we show a constructive proof of $Q \geq 2MNK/\sqrt{S} + MN$. This result has three main contributions (1) it is an additive improvement over the bound $2mnk/\sqrt{S} - 2S$ by Smith and van de Geijn [7] by an additive factor of $2S + mn$, (2) the proof is constructive - what immediately follows is a corresponding schedule, (3) generality of a used technique is later used to derive I/O optimal parallel schedule (**??**) and may be used in other linear algebra problems.

Theorem 1 (Sequential Matrix Multiplication I/O lower bound)**.** *Multiplying matrices of sizes $m \times k$ and $k \times m$ requires a minimum number of $\frac{mnk}{\sqrt{S}} + mn$ I/O operations.*

Proof. We first discuss a short intuition behind the proof. Next, we provide all details.

Proof Intuition Using Lemma 2, we first establish a valid subset V_i of an S-partition that achieves maximum computational intensity ρ and find its size (the number of vertices). Knowing ρ and that there are mnk vertices in the whole MMM CDAG, we find the total I/O cost of the complete execution. Throughout the proof, in addition to full details, we include intuitive interpretations of key steps. To help keep up with the intuition, assume that V_i is a cuboid (**??**), and that we want to find its optimal size under certain conditions.

Full Proof In this proof, we will use the iteration space $\mathcal{V}$ (**??**) to represent the CDAG of MMM. By Definition 2, we divide the whole computation into $H(S)$ subcomputations $V_i, i \in \{1, \dots H(S)\}$, such that $Dom(V_i), Min(V_i) \leq S$. As stated in (**??**), the dominator set of V_i is $Dom(V_i) = \phi_{ik}(V_i) \cup \phi_{kj}(V_i) \cup \phi_{ij}(V_i) = \alpha_i \cup \beta_i \cup \gamma_i$. Because all the read-after-write (RAW) dependencies are parallel to the vector $\mathbf{k}$ (Line 5 in Listing 1), the projection of the minimum set $Min(V_i)$ onto the plane $\mathbf{ij}$ is also equal to γ_i : $(\phi_{ij}(Min(V_i)) = \gamma_i)$.

Intuition: α_i, β_i and γ_i are the faces of our cuboid V_i. The "bottom" face γ_i (Figure **??**) is formed by partial results of C from previous subcomputations, and the "top" face is formed by partial results evaluated by V_i, also forming input for the next subcomputation. Because all the dependencies are facing "upwards", the minimum set (the "top" face) is positioned directly above the required partial results of C, that is $\phi_{ij}(Dom(V_i)) = \phi_{ij}(Min(V_i))$.

By the definition of S-partition, we have:

$$|Dom(V_i)| = |\alpha_i| + |\beta_i| + |\gamma_i| \leq S \tag{6}$$
$$|Min(V_i)| = |\gamma_i| \leq S$$

On the other hand, the Loomis-Whitney inequality [5] bounds the volume of V_i:

$$V_i \leq \sqrt{|\alpha_i||\beta_i||\gamma_i|} \tag{7}$$

Now we proceed to find the upper bound on the maximum reuse size $R(S) = \max_{j=1 \dots H(S)}(|V_{R,i}|)$. Consider two subsequent computations, V_i and V_{i+1}. Right after V_i, α_i, β_i and γ_i may have red pebbles on them (they are in the fast memory). On the other hand the dominator set of V_{i+1} is $Dom(V_{i+1}) = \alpha_{i+1} \sum \beta_{i+1} \sum \gamma_{i+1}$. Then, the reuse set $V_{R,i+1}$ is an intersection of those two sets:

$$\begin{aligned} V_{R,i+1} &\subset (\alpha_i \cup \beta_i \cup \gamma_i) \cap (\alpha_{i+1} \cup \beta_{i+1} \cup \gamma_{i+1}) \\ &= (\alpha_i \cap \alpha_{i+1}) \cup (\beta_i \cap \beta_{i+1}) \cup (\gamma_i \cap \gamma_{i+1}) \end{aligned} \tag{8}$$

Intuition: Visualizing V_i and V_{i+1} as two cuboids, they can only be positioned in the 3D space so that they share at most one of their sides (either $\alpha_i \cap \alpha_{i+1} \neq \emptyset$ or $\beta_i \cap \beta_{i+1} \neq \emptyset$ or $\gamma_i \cap \gamma_{i+1} \neq \emptyset$). If we allow other shapes than cuboids, their shared surface is a sum of its projections into the three planes, $\alpha_i \cap \alpha_{i+1}$, $\beta_i \cap \beta_{i+1}$, and $\gamma_i \cap \gamma_{i+1}$.

Note that $\alpha_i, \beta_i \subset \mathcal{I}$ are inputs of the computation: therefore, by the Definition 1, they start in the slow memory (they already have blue pebbles). γ_i, on the other hand, is the projection of the minimum set $\phi_{ij}(Min(V_i)) = \gamma_i$. Therefore, at the end of V_i, vertices in $Min(V_i)$ may have only red pebbles placed on them (they may have not been stored back to the slow memory yet). Furthermore, by the Definition 2, these vertices have children that have not been pebbled yet. They either have to be reused forming the reuse set $V_{R,i+1}$, or stored back, forming $W_{BR,i}$ and requiring the placement of the blue pebbles. Because by their definition, $\gamma_i \cap \alpha_i = \gamma_i \cap \beta_i = \emptyset$, we have $\gamma_i \cap V_{R,i+1} = \gamma_i \cap \gamma_{i+1}$ and $W_{BR,i} = \gamma_i \setminus \gamma_{i+1}$. Then, the number of I/O operations Q_i performed by the subcomputation V_i (cf. Lemma 2):

$$Q_i \geq |Dom(V_i)| - |V_{R,i}| + |W_{BR,i}| \tag{9}$$

Intuition: The dominator set of each subcomputation consists of certain elements from input matrices A and B, as well as the partial results of C. The number of loads required by V_i is the size of the dominator set $Dom(V_i)$ reduced by the elements already loaded (the reuse set $V_{R,i}$). The number of stores is equal to the number of outputs $|\gamma_i|$ that are not reused in subcomputation V_{i+1}, that is, not contained in $|\gamma_{i+1}|$.

We now proceed to find α_i, β_i, and γ_i that maximizes the computational intensity $\rho_i = \rho$ (appendix A.4.3). We can bound ρ_i by inserting Equations 6, 7, 8 and 9:

$$\begin{aligned}\rho_i &= \frac{|V_i|}{|Dom(V_i)| - |V_{R,i}| + |W_{BR,i}|} \\ &\leq \frac{\sqrt{|\alpha_i||\beta_i||\gamma_i|}}{|\alpha_i| + |\beta_i| + |\gamma_i| - |\gamma_i \cap \gamma_{i+1}| + |\gamma_i \setminus \gamma_{i+1}|}\end{aligned}$$

We immediately see that to maximize ρ_i, we have $\gamma_{i+1} = \gamma_i$, as it both maximizes $|\gamma_i \cap \gamma_{i+1}|$ and minimizes $|\gamma_i \setminus \gamma_{i+1}|$.

We then formulate it as an optimization problem:

$$\begin{gathered}\text{maximize } \sqrt{\gamma_i}\frac{\sqrt{|\alpha_i||\beta_i|}}{|\alpha_i| + |\beta_i|} \\ \text{subject to:} \\ |Dom(V_i)| = |\alpha_i| + |\beta_i| + |\gamma_i| \leq S \\ |\alpha_i|, |\beta_i|, |\gamma_i| \in \mathcal{Z}\end{gathered} \tag{10}$$

Observe that to maximize $\frac{\sqrt{|\alpha_i||\beta_i|}}{|\alpha_i|+|\beta_i|}$ we have $|\alpha_i| = |\beta_i|$. Then the solution to this maximization problem gives $|\alpha_i| = |\beta_i| \to 0$, $|\gamma_i| \to S$. But because α_i, β_i and γ_i are sets of vertices, therefore $|\alpha_i|, |\beta_i|, |\gamma_i| \in \mathcal{Z}$. Furthermore, we have $\phi_{kj}(\alpha_i) = \phi_{ik}(\beta_i)$, $\phi_{ij}(\alpha_i) = \phi_{ik}(\gamma_i)$ and $\phi_{ij}(\beta) = \phi_{kj}(\gamma)$. Finally, we get:

$$\begin{gathered}|\alpha_i| = |\beta_i| = \left\lfloor \sqrt{S+1} - 1 \right\rfloor \\ |\phi_{kj}(\alpha)| = |\phi_{ik}(\beta)| = 1 \\ |\gamma| = |\alpha||\beta| = \left\lfloor (\sqrt{S+1} - 1)^2 \right\rfloor\end{gathered} \tag{11}$$

From now on, to keep the calculations simpler, we use the following approximation:

$$\left\lfloor \sqrt{S+1} - 1 \right\rfloor \approx \sqrt{S} \tag{12}$$

All the following results may be easily rewritten using the precise formula and are correct up to this approximation factor.

Using the approximation 12, we have $|V_i| = S, R(S) = V_{R,i} = |\gamma_i| = S, \rho = \sqrt{S}/2$. Finally, by Corollary A.4.3:

$$Q \geq \frac{|V|}{\rho} = \frac{2mnk}{\sqrt{S}}$$

This is the I/O cost of putting a red pebble at least once on every vertex in $\mathcal{V}$. Note however, that we did not put any blue pebbles on the outputs yet (all vertices in $\mathcal{V}$ had only red pebbles placed on them during the execution). By Definition 1, we need to place blue pebbles on mn output vertices, corresponding to the output matrix C, resulting in additional mn I/O operations, yielding final bound

$$Q \geq \frac{2mnk}{\sqrt{S}} + mn$$

□

B.1 I/O Optimal Schedule

The proof of Theorem 1 is constructive: that is, from it we immediately obtain a schedule achieving it. The optimal subcomputation V_i corresponds to S calculated partial products of C. Its inputs α and β are subsets of a single column of A and a single row of B, both of length $\sqrt{S}$. This may be viewed as an outer product formulation of MMM: each subcomputation is an outer product of two vectors of length $\sqrt{S}$, with subsequent subcomputations updating this result (Listing 1).

```
T = $\sqrt{S}$
for i_1 = 1:m/T
  for j_1 = 1:n/T
    for r = 1:k    % k is the outer loop
      %elementary subcomputation V
      for i_2 = i_1*T : (i_1+1)*T
        for j_2 = j_1*T : (j_1+1)*T
          C(i_2,j_2) = C(i_2,j_2) + A(i_2,r)*B(r,j_2)
        end for
      end for
    end for
  end for
end for
```

Listing 1. Pseudocode of I/O optimal sequential MMM

References

[1] John R. Gilbert, Thomas Lengauer, and Robert Endre Tarjan. 1979. The Pebbling Problem is Complete in Polynomial Space. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing (STOC '79)*. ACM.

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

[3] Hong Jia-Wei and Hsiang-Tsung Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 326–333.

[4] Quanquan Liu. 2017. Red-Blue and Standard Pebble Games: Complexity and Applications in the Sequential and Parallel Models.

[5] L. H. Loomis and H. Whitney. 1949. An inequality related to the isoperimetric inequality. *Bull. Amer. Math. Soc.* 55, 10 (10 1949), 961–962. https://projecteuclid.org:443/euclid.bams/1183514163

[6] John E Savage. 1995. Extending the Hong-Kung model to memory hierarchies. In *International Computing and Combinatorics Conference*. Springer, 270–281.

[7] Tyler Michael Smith and Robert A. van de Geijn. 2017. Pushing the Bounds for Matrix-Matrix Multiplication. *CoRR* abs/1702.02017 (2017). arXiv:1702.02017 http://arxiv.org/abs/1702.02017