# A DETAILED DERIVATION OF I/O LOWER BOUND PROOFS

**Detailed Derivation Of I/O Lower Bound Proofs**

## A.1 Computational Intensity

We start by proving Lemma 2:

**Lemma 2.** *If $|H_{max}|$ can be expressed as a closed-form function of $X$, that is there exists some function $\chi$ such that $|H_{max}| = \chi(X)$, then the lower bound on $Q$ may be expressed as*

$$Q \geq n\frac{(X_0 - M)}{\chi(X_0)},$$

*where $X_0 = \arg\min_X \rho = \arg\min_X \frac{\chi(X)}{X-M}$.*

**Intuition.** *$\chi(X)$ expresses computation "volume", while $X$ is its input "surface". The term $X - M$ bounds the required communication and it comes from the fact that not all inputs have to be loaded (at most $M$ of them can be reused). $X_0$ corresponds to the situation where the ratio of this "volume" to the required communication is minimized (corresponding to a highest lower bound).*

Proof. Note that Lemma 1 is valid for any $X_c$ (i.e., for any $X_c$, it gives a valid lower bound). Yet, these bounds are not necessarily tight. As we want to find tight I/O lower bounds, we need to maximize the lower bound. $X_0$ by definition minimizes $\rho$; thus, it maximizes the bound. Lemma 2 then follows directly from Lemma 1 by substituting $\rho = \frac{\chi(X_0)}{X_0 - M}$. □

**Note.** If function $\chi(X)$ is differentiable and has a global minimum, we can find $X_0$ by, e.g., solving the equation $\frac{d\frac{\chi(X)}{X-M}}{dX} = 0$. The key limitation is that it is not always possible to find $\chi$, that is, to express $|H_{max}|$ solely as a function of $X$. However, for many linear algebra kernels $\chi(X)$ exists. Furthermore, one can relax this problem preserving the correctness of the lower bound, that is, by finding a function $\hat{\chi} : \forall_X \hat{\chi}(X) \geq \chi(X)$.

## A.2 Iteration vector, iteration domain, and access sizes

We now prove Lemma 3:

**Lemma 3.** *Given the ranges of all iteration variables $D^t, t = 1, \ldots, l$ during subcomputation $H$, if $|H| = \prod_{t=1}^{l} |D^t|$, then $\forall j = 1, \ldots, m$ : $|A_j(\boldsymbol{D})| = \prod_{k=1}^{dim(\boldsymbol{\phi}_j)} |D_j^k|$ and $|H|$ is maximized among all valid subcomputations which iterate over $\boldsymbol{D} = [D^1, \ldots, D^t]$.*

**Intuition.** *Lemma 3 states that if each iteration variable $\psi^t, t = 1, \ldots, l$ takes $|R_h^t|$ different values, then there are at most $\prod_{t=1}^{l} |D^t|$ different iteration vectors $\boldsymbol{\psi}$ which can be formed in $H$. So, intuitively, to maximize $|H|$, all combinations of values $\psi^t$ should be evaluated. On the other hand, this also implies maximization of all access sizes $|A_j(\boldsymbol{D})| = \prod_{k=1}^{dim(\boldsymbol{\phi}_j)} |D_j^k|$.*

To prove it, we now introduce two auxiliary lemmas:

**Lemma 9.** *For statement $S$, the size $|H|$ of subcomputation $H$ (number of vertices of $S$ computed during $H$) is bounded by the sizes of the iteration variables' sets $R_h^t, t = 1, \ldots, l$:*

$$|H| \leq \prod_{t=1}^{l} |D^t|. \tag{5}$$

Proof. Inequality 5 follows from a combinatorial argument: each computation in $H$ is uniquely defined by its iteration vector $[\psi^1, \ldots, \psi^l]$. As each iteration variable $\psi^t$ takes $|R_h^t|$ different values during $H$, we have $|R_h^1| \cdot |R_h^2| \cdot \cdots \cdot |R_h^t| = \prod_{t=1}^{l} |D^t|$ ways how to uniquely choose the iteration vector in $H$. □

Now, given $\boldsymbol{D}$, we want to assess how many different vertices are accessed for each input array $A_j$. Recall that this number is denoted as access size $|A_j(\boldsymbol{D})|$.

We will apply the same combinatorial reasoning to $A_j(\boldsymbol{D})$. For each access $A_j[\boldsymbol{\phi}_j(\boldsymbol{\psi})]$, each one of $\psi_j^k$, $k = 1, \ldots, dim(\boldsymbol{\phi}_j)$ iteration variables loops over set $R_{h,j}^k$ during subcomputation $H$. We can thus bound size of $A_j(\boldsymbol{D})$ similarly to Lemma 9:

**Lemma 10.** *The access size $|A_j(\boldsymbol{D})|$ of subcomputation $H$ (the number of vertices from the array $A_j$ required to compute $H$) is bounded by the sizes of $dim(\boldsymbol{\phi}_j)$ iteration variables' sets $R_{h,j}^k, k = 1, \ldots, dim(\boldsymbol{\phi}_j)$:*

$$\forall_{j=1,\ldots,m} : |A_j(\boldsymbol{D})| \leq \prod_{k=1}^{dim(\boldsymbol{\phi}_j)} |D_j^k| \tag{6}$$

*where $D_j^k \ni \psi_j^k$ is the set over which iteration variable $\psi_j^k$ iterates during $H$.*

Proof. We use the same combinatorial argument as in Lemma 9. Each vertex in $A_j(\boldsymbol{D})$ is uniquely defined by $[\psi_j^1, \ldots, \psi_j^{dim(\boldsymbol{\phi}_j)}]$. Knowing the number of different values each $\psi_j^k$ takes, we bound the number of different access vectors $\boldsymbol{\phi}_j(\boldsymbol{\psi}_h)$. □

**Example**: *Consider once more statement $S1$ from LU factorization in Figure 3. We have $\boldsymbol{\phi}_0$ = [i, k], $\boldsymbol{\phi}_1$ = [i, k], and $\boldsymbol{\phi}_2$ = [k, k]. Denote the iteration subdomain for subcomputation $H$ as $\boldsymbol{D} = \{[k^1, i^1], \ldots, [k^{|H|}, i^{|H|}]\}$, where each variable $k$ and $i$ iterates over its set $k^g \in \{\psi_{k,1}, \ldots, \psi_{k,K}\} = R_h^k$ and $i^g \in \{\psi_{i,1}, \ldots, \psi_{i,I}\} = R_h^i$, for $g = 1, \ldots, |H|$. Denote the sizes of these sets as $|R_h^k| = K_h$ and $|R_h^i| = I_h$, that is, during $H$, variable $k$ takes $K$ different values and $i$ takes $I_h$ different values. For $\boldsymbol{\phi}_1$, both iteration variables used are different: $k$ and $i$. Therefore, we have (Equation 6) $|A_1(\boldsymbol{D})| \leq K_h \cdot I_h$. On the other hand, for $\boldsymbol{\phi}_2$, the iteration variable $k$ is used twice. Recall that the access dimension is the minimum number of different iteration variables that uniquely address it (Section 2.2), so its dimension is $dim(A_2) = 1$ and the only iteration variable needed to uniquely determine $\boldsymbol{\phi}_2$ is $k$. Therefore, $|A_2(\boldsymbol{D})| \leq K_h$.*

**Dominator set.** Input vertices $A_1, \ldots, A_m$ form a dominator set of vertices $A_0$, because any path from graph inputs to any vertex in $A_0$ must include *at least* one vertex from $A_1, \ldots, A_m$. This is also the *minimum* dominator set, because of the disjoint access property (Section 2.2): any path from graph inputs to any vertex in $A_0$ can include *at most* one vertex from $A_1, \ldots, A_m$.

*Proof of Lemma 3.* For subcomputation $H$, we have $|\bigcup_{j=1}^{m} A_j(\boldsymbol{D})| \leq X$ (by the definition of an $X$-partition). Again, by the disjoint access property, we have $\forall j_1 \neq j_2 : A_{j_1}(\mathcal{D}) \cap A_{j_2}(\mathcal{D}) = \emptyset$. Therefore, we also have $|\bigcup_{j=1}^{m} A_j(\boldsymbol{D})| = \sum_{j=1}^{m} |A_j(\boldsymbol{D})|$. We now want to maximize $|H|$, that is to find $H_{max}$ to obtain computational intensity $\rho$ (Lemma 2).

Now we prove that to maximize $|H|$, inequalities 5 and 6 must be tight (become equalities).

From proof of Lemma 9 it follows that $|H|$ is maximized when iteration vector $\boldsymbol{\psi}$ takes all possible combinations of iteration variables $\psi_h^t \in R_h^t$ during $H$. But, as we visit each combination of all $l$ iteration variables, for each access $A_j$ every combination of its $[\psi_j^1, \ldots, \psi_j^{dim(\boldsymbol{\phi}_j)}]$ iteration variables is also visited. Therefore, for every $j = 1, \ldots, m$, each access size $|A_j(\mathcal{D})|$ is maximized (Lemma 10), as access functions are injective, which implies that for each combination of $[\psi_j^1, \ldots, \psi_j^{dim(\boldsymbol{\phi}_j)}]$, there is one access to $A_j$. $\prod_{t=1}^{l} |R_h^t|$ is then the upper bound on $|H|$, and its tightness implies that all bounds on access sizes $|A_j(\mathcal{D})| \leq \prod_{k=1}^{dim(\boldsymbol{\phi}_j)} |D_j^k|$ are also tight. □

## A.3 Computational intensity and out-degree one vertices

Here we present a short proof of Lemma 4 followed by an example:

**Lemma 4.** *If in a cDAG $G = (V, E)$ every non-input vertex has at least $u$ direct predecessors with out-degree one that are graph inputs, then the maximum computational intensity $\rho$ of this cDAG is bounded by $\rho \leq \frac{1}{u}$.*

Proof. By the definition of the red-blue pebble game, all inputs start in slow memory, and therefore, have to be loaded. By the assumption on the cDAG, to compute any non-input vertex $v \in V$, at least $u$ input vertices need to have red pebbles placed on them using a load operation. Because these vertices do not have any other direct successors (their out-degree is 1), they cannot be used to compute any other non-input vertex $w$. Therefore, each computation of a non-input vertex requires at least $u$ unique input vertices to be loaded. □

*Example:* Consider Figure 11. In a), each compute vertex $C[i, j]$ has two input vertices: $A[i, j]$ with out-degree 1, and $b[j]$ with out-degree $n$, thus $u = 1$. As both array $A$ and vector $b$ start in the slow memory (having blue pebbles on each vertex), for each computed vertex from $C$, at least one vertex from $A$ has to be loaded, therefore $\rho \leq 1$. In b), each computation needs two out-degree 1 vertices, one from vector $a$ and one from vector $b$, resulting in $u = 2$. Thus, $\rho \leq \frac{1}{2}$.

## A.4 Data Reuse Across Multiple Statements

**Lemma 5.** *The I/O cost of a program containing statements $S$ and $T$ which share the input array $A_i$ is bounded by*

$$Q_{tot} \geq Q_S + Q_T - Reuse(A_i)$$

*where $Q_S$, $Q_T$ are the I/O costs of a program containing only statement $S$ or $T$, respectively. Furthermore, we have:*

$$Reuse(A_i) = \min\{|A_i(\boldsymbol{R}_S)|, |A_i(\boldsymbol{R}_T)|\}$$

*where $|A_i(\boldsymbol{R}_S)|$ and $|A_i(\boldsymbol{R}_T)|$ are the total number of accesses to array $A_i$ during the optimal execution of statements $S$ or $T$ separately.*

Proof. Consider an optimal sequential schedule of a cDAG $G_S$ containing statement $S$ only. For any subcomputation $H_s$ and its associated iteration domain $\boldsymbol{R}_s$ its minimum dominator set is $Dom(H_s) = \bigcup_{j=1}^{m} A_j(\boldsymbol{R}_s)$. To compute $H_S$, at least $\sum_{i=1}^{m} |A_j(\boldsymbol{R}_s)| - M$ vertices have to be loaded, as only $M$ vertices can be reused from previous subcomputations.

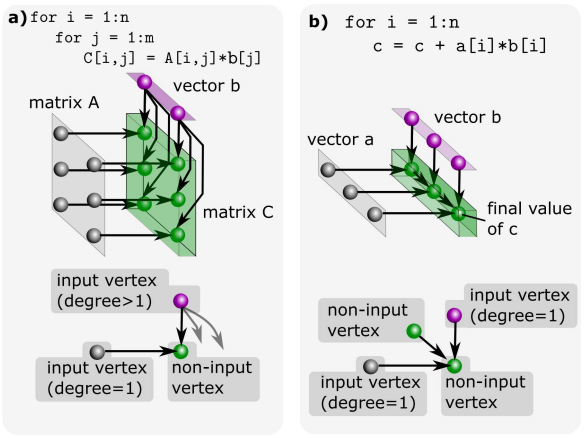


**Figure 11: cDAGs with out-degree 1 input vertices. a)** $u_a = 1, \rho_a \leq 1$. **b)** $u_b = 2$, $\rho_b \leq \frac{1}{2}$.

We seek if any loads can be avoided in the common schedule if we add statement $T$, denoting its cDAG $G_{S+T}$. Consider a subset $A_i(\boldsymbol{R}_x)$ of vertices in $A_i$.

Consider some subset of vertices in $A_i$ which potentially could be reused and denote it $\Theta_i$. Now denote all vertices in $A_0$ (statement $S$) which depend on any vertex from $\Theta_i$ as $\Theta_S$, and, analogously, set $\Theta_T$ for statement $T$. Now consider these two subsets $\Theta_S$ and $\Theta_T$ separately. If $\Theta_S$ is computed before $\Theta_T$, then it had to load all vertices from $\Theta_i$, avoiding no loads compared to the schedule of $G_S$ only. Now, computation of $\Theta_T$ may take benefit of some vertices from $\Theta_i$, which can still reside in fast memory, avoiding up to $|\Theta_i|$ loads.

The total number of avoided loads is bounded by the number of loads from $A_i$ which are shared by both $S$ and $T$. Because statement $S$ loads at most $|A_i(\boldsymbol{R}_S)|$ vertices from $A_i$ during optimal schedule of $G_S$, and $T$ loads at most $|A_i(\boldsymbol{R}_T)|$ of them for $G_T$, the upper bound of shared, and possibly avoided loads is $Reuse(A_i) = \min\{|A_i(\boldsymbol{R}_S)|, |A_i(\boldsymbol{R}_T)|\}$.

□

The **reuse size** is defined as $Reuse(A_i) = \min\{|A_i(\boldsymbol{R}_S)|, |A_i(\boldsymbol{R}_T)|\}$. Now, how to find $|A_i(\boldsymbol{R}_S)|$ and $|A_i(\boldsymbol{R}_T)|$?

Observe that $|A_i(\boldsymbol{R}_S)|$ is a property of $G_S$, that is, the cDAG containing statement $S$ only. Denote the I/O optimal schedule parameters of $G_S$: $V_{max}^S$, $X_0^S$, and $|A_i(\boldsymbol{R}_{max}^S(X_0^S))|$ (Section 3.2). Similarly, for $G_T$: $V_{max}^T$, $X_0^T$, and $|A_i(\boldsymbol{R}_{max}^T(X_0^T))|$. We now derive: 1) at least how many subcomputations does the optimal schedule have: $s \geq \frac{|V|}{|H_{max}|}$, 2) at least how many accesses to $A_i$ are performed per optimal subcomputation $|A_i(\boldsymbol{R}_{max}(X_0))|$. Then:

$$Reuse(A_i) = \min\{|A_i(\boldsymbol{R}_{max}^S(X_0^S))| \frac{|V^S|}{|V_{max}^S|}, \quad (7)$$
$$|A_i(\boldsymbol{R}_{max}^T(X_0^T))| \frac{|V^T|}{|V_{max}^T|}\}$$

We now proceed to Lemma 6

**Lemma 6.** *Any dominator set of set $B_j(\mathcal{D})$ must be of size at least $|Dom(B_j(\mathcal{D}))| \geq \frac{|B_j(\mathcal{D})|}{\rho_S}$.*

Proof. By Lemma 1, for one loaded vertex, we may compute at most $\rho_S$ vertices of $A_0$. These are also vertices of $B_j$. Thus, to

compute $|B_j(\mathcal{D})|$ vertices of $B_j$, at least $\frac{|B_j(\mathcal{D})|}{\rho S}$ loads must be performed. We just need to show that at least that many vertices have to be in any dominator set $Dom(B_j(\mathcal{D}))$. Now, consider the converse: There is a vertex set $D = Dom(B_j(\mathcal{D}))$ such that $|D| < \frac{|B_j(\mathcal{D})|}{\rho S}$. But that would mean, that we could potentially compute all $|B_j(\mathcal{D})|$ vertices by only loading $|D|$ vertices, violating Lemma 1. □

## A.5 Parallel I/O Lower Bounds

**Lemma 7.** *The minimum number of I/O operations in a parallel pebble game, played on a cDAG with $|V|$ vertices with $P$ processors each equipped with $M$ pebbles, is $Q \geq \frac{|V|}{P \cdot \rho}$, where $\rho$ is the maximum computational intensity independent of $P$ (Lemma 1).*

Proof. Following the analysis of Section 3 and the parallel machine model (Section 5), the computational intensity $\rho$ is independent of a number of parallel processors - it is solely a property of a cDAG and private fast memory size $M$. Therefore, following Lemma 1, what changes with $P$ is the volume of computation $|V|$, as now at least one processor will compute at least $|V_p| = \frac{|V|}{P}$ vertices. By the definition of the computational intensity, the minimum number of I/O operations required to pebble these $|V_p|$ vertices is $\frac{|V_p|}{\rho}$. □