CR15: notes for the lecture of January 9, 2023
Pebble Game Models (2/2)

Loris Marchal
(Notes revised by Frédéric Vivien)

January 5, 2023

1 Red-Blue pebble game for data transfer minimization

In some cases, the amount of fast storage (i.e., memory) is too limited for the complete execution of a program. In that case, communication are needed to move data from/to a second level of storage (i.e., disk), which is usually larger but slower. Because of the limited bandwidth of the secondary storage, the amount of data transfers, sometimes called Input/Output (or simply I/O) volume, is a crucial parameter for performance, as seen in the introduction. Once again, we present this problem for the pair (main memory, disk), but this may well apply to any pair of storages in the usually deep memory hierarchy going from the registers and fastest caches to the slowest storage systems.

1.1 Definition and examples

While the first pebble game allows to model algorithms under a limited memory, Hong and Kung have proposed another pebble game to tackle the I/O volume minimization in their seminal article [1]. This game uses two types of pebbles (of different colors) and thus is also called the red/blue pebble game to distinguish with the original (black) pebble game. The goal is to distinguish between the main memory storage (represented by red pebbles), which is fast but limited, and the disk storage (represented by blue pebbles), which is unlimited but slow. As in the previous model, a computation is represented by a directed acyclic graph. The red and blue pebbles can be placed on vertices according to the following rules:

- (RB1) A red pebble may be placed on any vertex that has a blue pebble.
- (RB2) A blue pebble may be placed on any vertex that has a red pebble.
- (RB3) If all predecessors of a vertex v have a red pebble, a red pebble may be placed on v.
- (RB4) A pebble (red or blue) may be removed at any time.
- (RB5) A blue pebble can be placed on an input vertex at any time.
- (RB6) No more than S red pebbles may be used at any time.

The goal of the game is to put a red pebble on each output vertex at some point of the computation, and to use the minimum number of RB1/RB2 rules to reach this goal. Red vertices represents values that currently lies in the main memory, after a computation, while blue pebbles represents values that are on disk. A value on disk may be read from disk (rule RB1) and similarly a value in memory can be stored on disk (rule RB2). Finally, we may compute a value if all its inputs are already in memory (rule RB3). The volume of I/O is the total number of moves using rules RB1 or RB2, that is the total number of data movements between memory and disk.

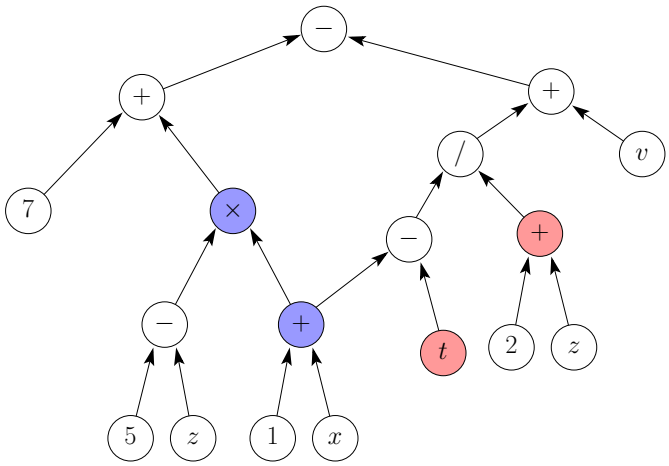

Figure 1: Playing the red/blue pebble game. The two red nodes represent values that are currently in memory, whereas the two blue nodes represent values that have been computed but then have been evicted to disk. Before pebbling node $(1 + x) - t$, a red pebble has to be put again on node $1 + x$, corresponding to reading this value from the disk.

Consider now the FFT graph presented in the slides. With $S = 3$ red pebbles, the graph can be computed by putting each intermediate vertex in slow memory, so the number of I/Os is $k\ 2^{k+1}$ (inputs only read, outputs only written, intermediate vertices both written and read). In fact the last vertex computed at one level may not need to be written (move from red to blue), but could be reused immediately to compute one of its two immediate successors. There are $k-1$ intermediate levels for which a node is not written and the read back. Furthermore, the computed target nodes need not be written back to memory: we can just remove their red pebbles (rule RB5). Hence, a total of I/Os of:

$$2^k + (k-1) * 2 * \left(2^k - 1\right) = 2^{k+1} - 2^k - 2k + 2$$

1.2 The Hong-Kung Lower-Bound method

We present here the method proposed by Hong and Kung in their seminal paper [1] to derive lower-bounds on the volume of I/O needed for some computations with limited storage, that is, limited number of red pebbles. We present a slightly simpler version of their result as revisited by Savage [2]. We first define the S-span of a DAG.

Definition 1. *Given a DAG G, its S-span, $\rho(S, G)$, is the maximum number of vertices of G that can be pebbled with S pebbles in the black pebble game, maximized over all initial placements of the S pebbles.*

Then we express the lower bound on the number of I/O steps $T_{I/O}(S, G)$, defined as the number of steps using rules RB1 or RB2 in a pebbling scheme S of the graph G.

Theorem 1. *For every pebbling scheme $\mathcal{S}$ of a DAG $G = (V, E)$ in the red-blue pebble-game using at most S red pebbles, the number of I/O steps satisfies the following lower bound:*

$$\lceil T_{I/O}(S, G)/S \rceil \rho(2S, G) \geq |V| - |Inputs(G)|$$

Proof. We first divide the pebbling scheme $\mathcal{S}$ into phases where each phase has exactly S I/O operations (except the last one which may have less). There are $h = \lceil T_{I/O}/S \rceil$ such phases.

We now establish a bound on the maximal number of vertices computed during each phase. To do this, we transform the pebbling of a phase to move all read operations to the beginning of the phase. Let V_{read} be the set of vertices that are pebbled with rule RB1 (a blue pebble is

transformed in a red pebble) during the phase. Note that $|V_{read}| \leq S$ by definition of phases. We consider a supplementary set of $|V_{read}|$ red pebbles. For each vertex v in V_{read}, we put a red pebble from the supplementary set on v either at the beginning of the phase if it is pebbled in blue at that time, or the first time a blue pebble is put on it in the phase (first use of rule RB2 on that vertex), we keep that red pebble on v until the end of the phase, and we remove the read steps from the pebbling scheme of the phase. Note that the new pebbling scheme of the phase is still valid. At the start of the new phase, there are at most $2S$ red pebbles on the graph. Clearly, the number of vertices pebbled with a red pebble is not larger than if $2S$ pebbles were available and allowed to move freely. Thus, the number of compute steps is at most $\rho(2S, G)$.

In total, the number of compute steps of the whole pebbling scheme is at most $h\rho(2S, G)$, and it should be at least $|V| - |Inputs(G)|$ to cover the whole graph, which gives the result. □

We now apply this result to our running example: the product of two $N \times N$ matrices. In order to do this, we first estimate the span of any DAG computing this product, corresponding to any standard algorithm.

We start with a small lemma.

Lemma 1. *Let T be a binary (in-)tree representing a computation, with p pebbles on some vertices. At most $p - 1$ vertices can be pebbled in the tree.*

Proof. Let us first assume that only leaves are pebbled. We first identify in the tree the s subtrees with all leaves pebbled. All the non-leaf vertices of these subtrees (and only these vertices) can be pebbled. For such a subtree with l leaves, this represent $l - 1$ nodes. Hence, exactly $p - s$ vertices can be pebbled in the whole tree, which is at most $p - 1$ in the case of a single subtree.

In the general case, some inner nodes may be pebbled. In such a case, we prune the tree by deleting any inner node and leaf which is the (transitive) predecessor of a pebbled inner node. We then fall back to the previous case. □

Theorem 2. *For every DAG G to compute the product of two $N \times N$ matrices in a regular manner (performing the N^3 products), the span $\rho(S, G)$ is such that $\rho(S, G) \leq 2S\sqrt{S}$ for $S \leq N^2$.*

Proof. We consider any initial placement of the S pebbles, and try to bound the number of vertices that can be pebbled from this placement. Since we consider a regular matrix product $C = AB$, all $C_{i,j}$ elements are obtained as inner product (or dot product) of the inputs: all products $A_{i,k}B_{k,j}$ are performed, and then summed through addition trees (each one being a binary tree). Note that inputs are shared for several products, while addition trees are not shared.

We consider that among the vertices already pebbled initially, r are nodes of addition trees (either product or addition) and $S - r$ are inputs (no outputs are pebbled to maximize the span). Let p be the number of products that can be pebbled from the $S - r$ inputs (we will bound it below). After all products are performed (it can be done in a first step without loss of generality), we end up in $p + r$ nodes in several addition trees. Thanks to the previous lemma, at most $p + r - t$ new vertices can be pebbled when t trees are used, which is at most $p + r - 1$. Thus, the span is bounded by $\rho(S, G) \leq 2p + r - 1$ (p products and $p + r - 1$ additions).

We now concentrate on bounding p. We consider matrices A' and B' whose (i, j) entry is 1 when the corresponding input in A or B is initially pebbled, and 0 otherwise. Then we consider the product $C' = A'B'$. $C'_{i,j}$ is equal to the number of products that can be performed from the initial position of the pebbles that will contribute to the $C_{i,j}$ element. Thus, $p = \sum_{i,j} C'_{i,j}$.

Let a (resp. b) be the number of 1 in A' (resp. B'), such that $a + b = S - r$ and let α be any integer smaller than N. There are at most a/α rows of A' with at least α ones ("dense" rows). The maximum number of non-null products that can be performed using these rows is ab/α, since each (full) row can contribute to at most b products. We now consider the other rows of matrix A' (or A). Given the limited number of pebbles, at most S final elements of C can be produced. For the "sparse" part of the matrix A, this corresponds to at most αS products. In total, this gives $p \leq ab/\alpha + \alpha S$, which gives $p \leq 2\sqrt{abS}$ for $\alpha = \sqrt{ab/S}$. This upper bound is maximized when the $S - r$ pebbles are balanced among A and B, that is, when with $a = b = (S - r)/2$ which gives $p \leq (S - r)\sqrt{S}$. The span is then bounded by $\rho(S, G) \leq 2(S - r)\sqrt{S} + r - 1 \leq 2S\sqrt{S}$. □

Hence, the I/O time for the matrix product is bounded by:

$$T_{I/O} = \Theta\left(\frac{N^3}{\sqrt{S}}\right)$$

Note that these results may be extended to deeper memory hierarchy, using more pebble colors (see [2]).

Material used for this course and further documentation

Refer to chapter 11 ("Memory-Hierarchy Tradeoffs") of the book "Models of Computation" by J. Savage [2] for more information on the red-blue pebble game.

References

[1] J.-W. Hong and H.T. Kung. I/O complexity: The red-blue pebble game. In *STOC'81: Proceedings of the 13th ACM symposium on Theory of Computing*, pages 326–333. ACM Press, 1981.

[2] John E. Savage. *Models of Computation: Exploring the Power of Computing.* Addison-Wesley, 1998.