Space-Time Tradeoffs in Memory Hierarchies

John E. Savage

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-93-08
Febuary 1993

Space-Time Tradeoffs in Memory Hierarchies [†]

John E. Savage
Brown University
Department of Computer Science, Box 1910
Providence, Rhode Island 02912
Phone: (401) 863-7642, Fax: (401) 863-7657
Email: jes@cs.brown.edu

Technical Report CS 93-08

Revised March 5, 1994

[†]This work was supported in part by the Office of Naval Research under contract N00014-91-J-4052, ARPA Order 8225.

Abstract

The speed of CPUs is accelerating rapidly, outstripping that of peripheral storage devices and making it increasingly difficult to keep CPUs busy. Multi-level memory hierarchies, scaled to simulate single-level memories, are increasing in importance. In this paper we introduce the Memory Hierarchy Game, a multi-level pebble game simulating data movement in memory hierarchies for straight-line computations. This game provides a framework for deriving upper and lower bounds on computation time and the I/O time at each level in a memory hierarchy. We apply this framework to a representative set of problems including matrix multiplication and the Fourier transform. We also discuss conditions on hierarchies under which they act as fast flat memories.

1 Introduction

A memory hierarchy for a serial computer is a connected set of memory units through which programs and data pass when moving from the highest-level unit (the slowest, largest and most remote) to the lowest-level unit (the fastest, smallest and most accessible) and back. The lowest-level unit is the local memory of a central processing unit. Memory hierarchies are used because the cost per bit of memory units is typically smaller for slower units than for faster ones. Thus, the capacity of units in a memory hierarchy typically grows with decreasing speed and increasing distance of the units from the CPU.

Memory hierarchies are increasing in importance as CPUs increase in speed relative to that of RAMs and magnetic disks, the principal form of bulk storage. CPUs have become so fast that it is increasingly difficult to keep them busy.

A key question asked about memory hierarchies is how to design them so they appear as fast as single-level memories for a large collection of important problems. We address this issue by introducing a multi-level pebble game called the **Memory Hierarchy Game** (MHG) that simulates data-independent computation and data movement in memory hierarchies. Data-independent or straight-line computations are modeled by directed acyclic graphs (dags). Not only are many important computational science algorithms described by dags, efficient prefetching in large memory hierarchies may require that all large computations be straight-line. The MHG generalizes to L levels the two-level game introduced by Hong and Kung [11].

The rules of the MHG assume that the values of input vertices of a dag $G = (V, E)$ are kept in the level-L archival memory unit. The goal of the game is to compute the values of the output vertices of G and store them in the level-L memory unit. When the value of a vertex v is resident in the level-j memory unit, we place a level-j pebble on v. A value can be in more than one unit at a time. We only allow values to move from a memory unit to an adjacent unit, one at the next higher or lower level. We model this by disallowing the placement of a level-j pebble on a vertex, $j \geq 2$, unless it carries one at level-$(j \pm 1)$. Since the value of a vertex can only be computed in a CPU, we have rules requiring that a vertex v not carrying a pebble must first be given one at level-1. To do this we require that the immediate predecessors of v carry level-1 pebbles, that is, their values must also be in the CPU. If there are insufficiently many level-1 pebbles to compute the value of a new vertex, we can replace level-1 pebbles on other vertices with level-2 pebbles, freeing up the level-1 pebbles to be used elsewhere. This corresponds to moving data up the memory hierarchy, thereby freeing up places at lower levels. This can be repeated up to the highest level, if the highest level unit can be used for intermediate storage (the "standard game"), or only to the next highest level if it cannot (the "I/O limited game").

The standard game models a memory hierarchy in which there is no large gap between the access times of the highest and next-highest level memory units whereas the I/O limited games does model such a gap. The two-level I/O limited game is actually the Paterson-Hewitt [14] one-color pebble game while the two-level standard game is the Hong-Kung Red-Blue pebble game. Thus, the MHG combines elements of both games and generalizes them to multiple levels.

A pebble strategy is "minimal" if we minimize the number of "computation steps" (placement of level-1 pebbles on a vertex containing no pebbles) and if we successively minimize the number of I/O operations at each level starting with the second. An "I/O operation at level-j" is either an "input from level-j" (placement of a level-$(j-1)$ pebble on a vertex carrying a level-j pebble) or an "output to level-j" (placement of a level-j pebble on a vertex carrying a level-$(j-1)$ pebble).

We develop methods for deriving upper and lower bounds on performance that we apply to a representative set of problems consisting of matrix multiplication, the Fast Fourier Transform and permutation and merging networks. We also generalize the model to block transfers. If the storage unit at level l holds p_l words, and the I/O time at level l, T_l, is the number of times blocks of b_l words move between the storage units at levels $l-1$ and l, then we establish a framework in which to derive lower bounds on T_l in terms of b_l and s_l, the storage capacity of all units up to and including the lth, that is, $s_l = p_1 + p_2 + \ldots + p_l$. Under weak conditions on b_l and for the problems under consideration, we show that our lower bounds on T_l can be achieved for the representative problems up to multiplicative constants for all levels simultaneously.

This approach is illustrated by the problem of multiplying matrices. We show that using any variant of the classical algorithm to multiply two $n \times n$ matrices, T_l is proportional to $\Theta(n^3/(b_l\sqrt{s_{l-1}}))$ when $s_{l-1} = O(n^2)$. If t_l is the time (relative to that of the fastest memory) to transfer a block between levels $l-1$ and l, it follows that a memory hierarchy will behave as single flat memory if $t_l \ll b_l\sqrt{s_{l-1}}$ for $2 \leq l \leq L$.

Prior Research Models for memory hierarchies have been studied by a number of authors. Aggarwal and Vitter [3] examined a two-level memory in which P blocks of contiguous items can be transferred in each step. They obtained tight bounds for sorting-related problems, including sorting, FFT, permutation networks, and matrix transposition. They did not handle the I/O-limited case or multiple levels.

Aggarwal, Alpern, Chandra and Snir [1] introduced the hierarchy memory model (HMM), which treats memory as a linear array with cost $f(x)$ to access location x in the array, and obtained tight bounds for a number of problems. They examined cost functions $\lceil \log x \rceil$, x, and x^α for $\alpha \geq 0$. They don't handle blocks, show optimal algorithms for an arbitrary spectrum of memory sizes, nor handle the I/O-limited

case or large discontinuities in the storage access time between levels.

Aggarwal, Chandra and Snir [2] introduced the BT model, an extension of the HMM model supporting block transfers. A block of size b ending at location x is allowed to move in time $f(x) + b$. They establish tight bounds on computation time for problems including matrix transpose, FFT, and sorting using the cost functions $\lceil \log x \rceil$, x, and x^α for $1 \leq \alpha \leq 1$. They allow blocks to be arbitrarily large and problem dependent but do not handle the I/O-limited case nor large discontinuities in access time.

Alpern, Carter and Feig [4] introduced the uniform memory hierarchy (UMH). The uth memory has capacity $\alpha\rho^{2u}$, block size ρ^u, and time $\rho^u/\beta(u)$ to move a block between levels; $\beta(u)$ is a bandwidth function. They allow I/O overlap between levels and determine conditions under which matrix transposition, matrix multiplication and Fourier transforms can and cannot be done efficiently.

Savage and Vitter [17] extended the one-level Paterson-Hewitt model [14] to support parallel pebbling and the Hong-Kung model to support contiguous block I/O. Vitter and Shriver [20] examine three parallel memory systems in which the memories are disks with block transfer, of the HMM type, or of the BT type. They present a randomized version of distribution sort that meets the lower bounds for these models of computation. Nodine and Vitter [13] give an optimal deterministic sorting algorithm for these memory models. The models have the limitations described above.

2 The Memory Hierarchy Game

The Memory Hierarchy Game (MHG) defined below captures the essential features of serial computers that use storage units organized into levels and in which data moves between levels from the highest to the lowest level and back. The highest level storage unit models an archival store with large access time whereas the lowest level unit models a fast memory used by the CPU for all its computations.

The L-level Memory Hierarchy Game (MHG) is played on dags with p_l pebbles at level l, $1 \leq l \leq L-1$, and an unlimited number of pebbles at level L. It has **resource vector** $\underline{p} = (p_1, p_2, \ldots, p_{L-1})$, where $p_j \geq 1$ for $1 \leq j \leq L-1$, and uses $s_l = \sum_{j=1}^{l} p_j$ pebbles at level l or less. The rules of the game are given below.

R1. (Computation Step) A first-level pebble can be placed on any vertex all of whose immediate predecessors carry first-level pebbles.

R2. (Pebble Deletion) Except for level-L pebbles on output vertices, a pebble at any level can be deleted from any vertex.

R3. (Initialization) A level-L pebble can be placed on an input vertex at any time.

R4. (Input from Level-l) For $2 \leq l \leq L-1$, a level-$(l-1)$ pebble can be placed on any vertex carrying a level-l pebble.

Standard Game

R5. (Output to Level-l) For $2 \leq l \leq L$, a level-l pebble can be placed on any vertex carrying a level-$(l-1)$ pebble.

I/O-limited Game

R5. (Output to Level-l) For $2 \leq l \leq L-1$, a level-l pebble can be placed on any vertex carrying a level-$(l-1)$ pebble.

R6. (I/O-limitation) Level-L pebbles can only be placed on input vertices and output vertices carrying level-$(L-1)$ pebbles.

The first rule permits the placement of first-level pebbles on vertices whose immediate predecessors carry first-level pebbles. This rule must be used to place a pebble on a vertex without one. Rule 2 permits the deletion of pebbles. Rule 3 allows level-L pebbles to be placed on input vertices. Since there are arbitrarily many level-L pebbles, we can assume that all input pebbles carry such pebbles initially. Rule 4 (an "input from level l") permits the movement of pebbles down the hierarchy, that is, in toward the CPU. Rule 5 (an "output to level l") in the standard game permits the movement of pebbles up the hierarchy. The modified Rule 5 of the I/O-limited game only allows movement up to level $L-1$. Rule 6 allows movement up to level L for output vertices. Thus, in the standard game level-L pebbles can be used for storage of intermediate values whereas in the I/O-limited game they cannot. Application of Rule 1 is called a **computation step** whereas application of Rule 3 is called an **initialization step**. Each application of Rules 4, 5, and 6 is called an **I/O operation**. Applications of Rule 2 are not counted.

Two restrictions of this game have been studied previously. The Red-Blue Pebble Game [11] is the standard MHG on two levels. It is used to investigate the number of I/O operations needed when the number of storage locations in the first-level memory is small. The Red Pebble Game [14] is the two-level I/O-limited MHG game. (Rule 5 does not apply.) It is generally viewed as a one-level game because second-level pebbles can be placed only on input and output vertices. The Red Pebble Game is used to explore tradeoffs between space and time for computations done completely in fast memory. Clearly, the multi-level MHG has elements of both games.

As a minor technical convenience we allow "sliding" of first-level pebbles from a predecessor to a successor vertex, that is, a pebble may be placed on a successor vertex as it is removed from a predecessor. This models the use of a register as both the source and target of an operation.

3 Computational Inequalities

In this section we derive generic lower bounds on the number of I/O and computation steps required by the MHG. We begin with definitions of these concepts and the concept of a minimal pebbling. We assume throughout that all vertices of a dag $G = (V, E)$ are reachable from input vertices.

Definition 3.1 *A pebbling $\mathcal{P}(\underline{p}, G)$ in the L-level MHG of the dag G with resource vector $\underline{p}$ is a step-by-step assignment of pebbles to vertices of G according to the rules of the MHG. Let $n_1^{(L)}(\mathcal{P})$ be the number of computations steps in $\mathcal{P}$ and let $n_l^{(L)}(\mathcal{P})$, $2 \leq l \leq L$, be the number of level-l I/O operations. A pebbling $\mathcal{P}$ is* **minimal** *if $n_1^{(L)}(\mathcal{P})$ is minimal and for $2 \leq l \leq L$, $n_l^{(L)}(\mathcal{P})$ is minimal given that $n_{l-1}^{(L)}(\mathcal{P})$ is minimal. For $1 \leq l \leq L$, let $T_l^{(L)}(\underline{p}, G)$ be the minimal value of $n_l^{(L)}(\mathcal{P})$.*

Thus, a pebbling is minimal if it has the minimal number of computation steps for the given resource vector, the minimal number of level-2 I/O operations among those pebblings with the minimal number of computation steps, and the minimal number of level-3 I/O operations given minimality at the two previous levels, etc. We use this definition of minimality because in real-world applications the time to move data between successive levels increases rapidly with level and is much larger than the time to perform a computation step. It follows that, in a minimal standard MHG computation, pebbles are never removed from a vertex once the vertex receives a pebble because there is an unlimited supply of highest-level pebbles available. However, under the I/O limitation, vertices may have to be recomputed many times.

Since there is little risk of confusion, we use the same notation, $T_l^{(L)}(\underline{p}, G)$, for the number of computation and I/O steps in the MHG with and without the I/O limitation. We use the following conventional definitions.

Definition 3.2 *For a directed acyclic graph $G = (V, E)$, let $In(G)$ and $Out(G)$ be the set of vertices of indegree zero* **(inputs)** *and outdegree zero* **(outputs)**, *respectively. All edges in G are directed from inputs to outputs via intermediate vertices. If (u, v) is an edge directed from vertex u to vertex v, v is an* **immediate descendant** *of u and u is an* **immediate predecessor** *of v. If there is a directed path from u to v, v is a* **descendant** *of u and u is a* **predecessor** *of v. Let $V^* = V - In(G)$.*

We assume that every vertex in V must be pebbled to pebble the output vertices, a condition that can always be satisfied. Since every input must be read from level L and every output written to level L, we have the following simple facts.

Lemma 3.1 *The following inequalities hold for any minimal MHG on $G = (V, E)$ for $2 \leq l \leq L$:*

$$T_l^{(L)}(\underline{p}, G) \geq |In(G)| + |Out(G)|, \quad T_1^{(L)}(\underline{p}, G) \geq |V^*|$$

The following definition abstracts ideas used by Hong and Kung [11].

Definition 3.3 *Given a dag $G = (V, E)$, let the S-***span of** *G, $\rho(S, G)$, be the the maximum number of vertices of G that can be pebbled with the MHG using only rules R1 and R2 and S level-1 pebbles without repebbling any vertex, maximized over all initial placements of S level-1 pebbles and assuming that pebbles are left on output vertices.*

The S-span of G is a measure of the maximum (over an initial placement of S level-1 pebbles) number of vertices of G that can be pebbled in the Red Pebble Game using S level-1 pebbles, assuming that pebbles placed on output vertices remain there. For example, $\rho(S, G) \leq 2S^{3/2}$ for $n \times n$ multiplication when $S \leq n^2$.

The following theorem generalizes Hong-Kung [11] lower bound on I/O time for the two-level MHG. Our proof is new and far simpler.

Theorem 3.1 *Consider a minimal pebbling of the dag $G = (V, E)$ in the standard MHG with resource vector $\underline{p}$ using s_l pebbles at level j, $1 \leq j \leq l$. Let $T_l^{(L)}(\underline{p}, G)$ be the number of I/O operations at level l, $2 \leq l \leq L$, and let $T_1^{(L)}(\underline{p}, G)$ be the number of computations steps used in this pebbling. Then, the following lower bound on $T_l^{(L)}(\underline{p}, G)$, $2 \leq l \leq L$, must be satisfied whether the MHG is I/O-limited or not:*

$$\lceil T_l^{(L)}(\underline{p}, G)/s_{l-1} \rceil \rho(2s_{l-1}, G) \geq |V^*|$$

Also, $T_1^{(L)}(\underline{p}, G) \geq |V^*|$.

PROOF Since fewer pebblings are done at each level for the standard game, the bounds are derived for this case. s_{l-1} is the number of pebbles at all levels up to and including level $l-1$. Let C be a minimal pebbling of G. It contains one computation step for each vertex in V^*. Divide C into consecutive sequential sub-pebblings $\{C_1, C_2, \ldots, C_h\}$ where each sub-pebbling has s_{l-1} level-l I/O operations except possibly the last which has no more such operations. There are $h = \lceil T_l^{(L)}(\underline{p}, G)/s_{l-1} \rceil$ such sub-pebblings.

We now develop an upper bound Q to the number of computation steps in each sub-pebbling. This number multiplied by the number h of sub-pebblings is an

upper bound to the total number of computation steps, $|V^*|$, performed by the pebbling C. It follows that

$$Qh \geq |V^*|$$

The bound on Q is developed by adding s_{l-1} new level-$(l-1)$ pebbles so that in each sub-pebbling, C_t, we may move all I/O operations at level l or higher to either the beginning or end of the sub-pebbling without changing the number of computation steps or I/O operations at level $l-1$ or less. Thus, we move without changing them all computation steps and I/O operations at level $l-1$ or lower to a "middle interval" of C_t in between the higher level I/O operations.

We now show how level-j I/O operations, $j \geq l$, can be moved to the beginning or end of C_t by adding at most s_{l-1} new level-$(l-1)$ pebbles and without changing the number of pebblings at level $l-1$ or lower. Consider a vertex v carrying a level-l pebble sometime during C_t that subsequently is given a level-$(l-1)$ pebble. Consider the first such pebble placement on v. The level-$(l-1)$ pebble assigned to v may have been in use prior to its placement on v. If a new level-$(l-1)$ pebble is used for v, the first pebbling of v with a level-$(l-1)$ pebble can be moved toward the beginning of C_t so that it precedes all placements of lower level pebbles without violating the precedence conditions of G. Attach this new level-$(l-1)$ pebble to v during C_t.

Consider a vertex v carrying a level-$(l-1)$ pebble that is pebbled with a level-l pebble during C_t. Replace this pebble with one of the new level-$(l-1)$ pebbles and attach it to v (unless such a pebble has been attached previously to v), thereby freeing up the original level-$(l-1)$ pebble. Because we attach this extra pebble to v for the entire pebbling C_t, all subsequent level-l I/O operations on v can be deleted except for the last level-l output operation which can be moved to the end of the interval. These changes do not affect any computation or level-j I/O step in C_t for $2 \leq j \leq l-1$.

We now derive an upper bound to Q. At the start of the pebbling of the middle interval there will be at most $2s_{l-1}$ pebbles of levels $l-1$ or less on G, at most s_{l-1} original pebbles plus a like number of new pebbles. Any output vertex pebbled in this interval must have a level-$(l-1)$ or lower pebble on it at the end of the interval since the pebbling is minimal. Clearly, the number of vertices pebbled in computation steps in the middle interval is largest when all $2s_{l-1}$ pebbles of levels $l-1$ or less on G are treated as level-1 pebbles. It follows that at most $\rho(2s_{l-1}, G)$ computation steps can be done on G in C_t with level-1 pebbles. This completes the proof of this theorem. □

Hong and Kung [11] derive their lower bound on the number of I/O operations in the Red-Blue Game by showing that at least $S[P(2S) - 1]$ I/O operations are necessary to pebble a graph with S red pebbles when $P(S)$ is the minimum number of vertex sets in any "S-partition" of the graph being pebbled. An **S-partition** is a partition of the vertices of a dag into sets $\{V_i\}$ such that every path from an input to a vertex in a set V_i passes through at most S vertices (a **dominator set**), the vertices in a set V_i which have no descendants in V_i (the **minimum set**) has at most S vertices, and there is no cyclic dependency between sets (V_i is dependent on V_j if there is an edge directed from a vertex in V_j to a vertex in V_i). To obtain concrete lower bounds they use the inequality $P(S) \geq |V|/\rho(S, G)$ to show that at least $S[|V|/\rho(2S, G) - 1]$ I/O operations are needed.

The following corollary to Theorem 3.1 provides an explicit lower bound to $T_l^{(L)}(\underline{p}, G)$.

Corollary 3.1 *In the standard MHG when $T_l^{(L)}(\underline{p}, G) \geq \beta(s_{l-1} - 1)$ for $\beta > 1$, the following inequality holds for $2 \leq l \leq L$.*

$$T_l^{(L)}(\underline{p}, G) \geq \frac{s_{l-1}}{\rho(2s_{l-1}, G)} \frac{\beta}{\beta + 1} |V|$$

In the standard game each vertex of G is pebbled once in a computation step. However, when the I/O limitation applies, some vertices may have to be repebbled. The following result is useful in such cases.

Theorem 3.2 *Let α be the maximum indegree of any vertex in G. Then, for any minimal computation with the MHG, whether standard or I/O-limited, and for $3 \leq l \leq L$, the following conditions hold:*

$$T_1^{(L)}(\underline{p}, G) \geq T_2^{(L)}(\underline{p}, G)/(\alpha + 1), \quad T_{l-1}^{(L)}(\underline{p}, G) \geq T_l^{(L)}(\underline{p}, G) \geq T_L^{(L)}(\underline{p}, G)$$

PROOF We have $T_1^{(L)}(\underline{p}, G) \geq T_2^{(L)}(\underline{p}, G)/(\alpha + 1)$ because for each computation step there are at most α transitions from level 2 to level 1 (otherwise some of these transitions are redundant) and at most one transition from level 1 to level 2 in any optimal computation. In an optimal pebbling a transition of a pebble from level l to level $l - 1$ on a vertex must coincide with (if $l = 2$) or be followed by (if $l > 2$) the placement of a level-1 pebble on that vertex; if not, then such an I/O operation is redundant. Similarly, any transition from $l - 1$ to l must be preceded by a transition from $l - 2$ to $l - 1$. Thus, the number of level-$(l - 1)$ I/O operations is at least equal to the number of level-l I/O operations. □

The following theorem relates the number of moves in an L-level game to the number in a two-level game and allows us to use prior results.

Theorem 3.3 *Let $S = s_{L-1} = \sum_{j=1}^{L-1} p_j$. Then, the following two inequalities hold when the graph G is pebbled in the L-level MHG, whether I/O limited or not, with the resource vector $\underline{p}$:*

$$T_1^{(L)}(\underline{p}, G) \geq T_1^{(2)}(S, G), \quad T_L^{(L)}(\underline{p}, G) \geq T_2^{(2)}(S, G)$$

Here $T_2^{(2)}(S, G)$ is the number of I/O operations in the Red-Blue Pebble Game played on G with S level-1 pebbles.

PROOF Consider a two-level MHG (the Red-Blue Game) played with $S = s_{L-1}$ level-1 (red) pebbles. These S pebbles can be classified into $L-1$ groups and labeled with the integers $\{1 \ldots L-1\}$ so that we can simulate the steps in the L-level MHG. Since the number of first level and last level operations in the simulated game are no less than the minimal number of such operations in the Red-Blue Game when the labeling restriction is dropped, the results follow. □

Combining Lemma 3.1, Theorem 3.2 and Theorem 3.3, we obtain the following lower bounds on the number of I/O operations and computation steps needed to pebble a directed acyclic graph in the I/O-limited MHG.

Corollary 3.2 *Let G be pebbled in either the standard or I/O-limited MHG with resource vector $\underline{p}$. Let $S = s_{L-1}$. If α is the maximum indegree of any vertex in the graph G, then for all $1 \leq l \leq L-1$ the following lower bound on $T_l^{(L)}(\underline{p}, G)$ holds:*

$$T_l^{(L)}(\underline{p}, G) \geq \max\left(\frac{T_2^{(2)}(S, G)}{\alpha + 1}, |V^*|\right)$$

We close this section with a result giving conditions under which lower bounds on computation and I/O time for one graph can be used for another.

Let a **reduction** of dag $G_1 = (V_1, E_1)$ be a dag $G_0 = (V_0, E_0)$, $V_0 \subseteq V_1$ and $E_0 \subseteq E_1$, obtained by deleting edges from E_1 and coalescing the non-terminal vertices on a "chain" of vertices in V_1 into the first vertex on the chain. A **chain** is a sequence $v_1, v_2, \ldots, v_r$ of vertices such that $(v_i, v_{i+1}) \in V_1$ for $1 \leq i \leq r-1$ and for all other vertices u is $(u, v_{i+1}) \notin V_1$ for $1 \leq i \leq r-2$.

Lemma 3.2 *Let G_0 be a reduction of G_1. Then, for $1 \leq l \leq L$:*

$$T_l^{(L)}(\underline{p}, G_1) \geq T_l^{(L)}(\underline{p}, G_0)$$

PROOF Any minimal pebbling strategy for G_1 can be used to pebble G_0 by simulating moves on a chain with pebble placements on the vertex to which vertices on the chain are coalesced and by honoring the edge restrictions of G_1 that are removed to create G_0. Since this strategy for G_1 may not be minimal for G_0, the inequalities follow. □

In the next section we specialize the above lower bounds to representative problems and derive corresponding upper bounds.

4 Bounds for Individual Problems

We now examine the extent to which the lower bounds derived above can be met by deriving upper and lower bounds on the number of computation and I/O steps in the standard and I/O-limited MHG for the following representative problems: matrix multiplication when only scalar additions and multiplications are used, the discrete Fourier Transform, and networks for permuting, merging and sorting sequences. These problems are at the heart of scientific computation. Carlson has designed very fast two-level FFT algorithms for CRAY supercomputers [6,7] and Dongarra *etal.* [8] cite algorithms for memory hierarchies designed for linear algebra problems.

4.1 Matrix Multiplication

We now derive a bound on the S-span of "butterfly-based level graphs," the FFT and related graphs on 2^d inputs, such as permutation networks based on three back-to-back FFT graphs $P^{(d)}$ [21], bitonic sorting networks $BS^{(d)}$ for merging sorted lists [12, pp. 632], and sorting networks $S^{(d)}$ based on bitonic sorting [12, pp. 633].

Consider a matrix multiplication algorithm conforming to the classical algorithm. All products of pairs of entries in the two matrices A and B are formed and combined in independent binary addition trees. We develop tight upper and lower bounds on the I/O and computation time for the standard MHG as well as the I/O-limited MHG. Developing good upper bounds under the I/O limitation is related to the challenging problem of deriving fast algorithms for matrix multiplication.

We begin by deriving an upper bound on the S-span of G, $\rho(S, G)$, for matrix multiplication. We simplify the proof given by Hong and Kung [11] and illustrate how such bounds are derived.

Lemma 4.1 *The S-span of any graph G associated with the classical algorithm to multiply two $n \times n$ matrices with the binary operations of addition and multiplication of component values satisfies $\rho(S, G) \leq 2S^{3/2}$ for $S \leq n^2$.*

PROOF $\rho(S, G)$ is the maximum number of pebblings possible in G from an initial placement of S pebbles under the assumption that any pebble placed on an output vertex stays there throughout the computation and the computation is minimal.

Let A, B, and C be $n \times n$ matrices with entries $\{a_{i,j}\}$, $\{b_{i,j}\}$, and $\{c_{i,j}\}$, respectively, where $1 \leq i, j \leq n$. Let $C = A \times B$. Then, $c_{i,j} = \sum_k a_{i,k} b_{k,j}$. The classical algorithm forms products of entries and adds them in binary addition trees. Each product is associated with a unique term $c_{i,j}$, as is each addition. Consider a dag for this computation which has one vertex for each addition and multiplication. We say a vertex is in an addition tree if it is either a product vertex (an input to an addition tree) or a vertex internal to such a tree.

Consider an initial placement of $S \leq n^2$ pebbles. Let r of these pebbles be in addition trees. Let the remaining $S - r$ pebbles be on input vertices. Let p be the number of products that can be formed from these pebbled inputs. Each product corresponds to one pebble placement. We show that at most $p + r - 1$ pebble placements are possible on non-input vertices in addition trees, giving a total of at most $\pi = 2p + r - 1$ pebble placements.

Given the dependencies in the classical algorithm, there is no loss in generality in assuming that product vertices are pebbled before pebbles are advanced in addition trees. It follows that at most $p + r$ addition-tree vertices carry pebbles before pebbles are advanced in addition trees. These pebbled vertices define subtrees all of whose vertices can be pebbled from the $p + r$ initial pebble placements. Since a binary with n leaves has $n - 1$ non-leaf nodes, it follows that if there are t such trees, at most $p + r - t$ pebble placements will be made. This number is maximized at $t = 1$.

We now complete the proof by deriving an upper bound on p. Let $\mathcal{A}$ be the $0 - 1$ $n \times n$ matrix whose (i, j) entry is 1 if the variable in the (i, j) position of the first matrix carries a pebble initially and 0 otherwise. Let $\mathcal{B}$ be similarly defined for the second matrix. It follows that the (i, j) entry, $\rho_{i,j}$, of the matrix product $\mathcal{C} = \mathcal{A}\mathcal{B}$, where addition and multiplication are over the integers, is equal to the number of products that can be formed that contribute to the (i, j) entry of the result matrix. Thus, $p = \sum_{i,j} \rho_{i,j}$. We now show that $p \leq \sqrt{S}(S - r)$.

Let A and B have a and b 1's, respectively, where $a + b \leq S - r$. There are at most a/α rows of A containing at least α 1's. The maximum number of products that can be formed from such rows is ab/α because each 1 in B can combine with

Figure 1: Pebbling schema for matrix multiplication.

at most a/α 1's in these rows. Now consider the product of other rows of A with columns of B. At most S such row-column inner products are formed since at most S outputs can be pebbled. Since each of them involves a row with at most α 1's, at most αS products of pairs of variables can be formed. Thus, a total of at most $p = ab/\alpha + \alpha S$ products can be formed. We are free to choose α to minimize this sum but must choose a and b to maximize it. The result is that $p \leq \sqrt{S}(S - r)$.

We complete the proof by observing that $\pi \leq 2\sqrt{S}S$ for $r \geq 0$. □

We now derive bounds on the computation and I/O times for the standard MHG. We show that the lower bounds given above on these times can be achieved up to constant multiplicative factors simultaneously at all levels in the memory hierarchy.

Theorem 4.1 *Let G be any graph consistent with the classical algorithm to multiply two $n \times n$ matrices. Let it be pebbled in the standard MHG with the resource vector $\underline{p}$. Let $s_l = \sum_{j=1}^{l} p_j$ and let k be the largest integer such that $s_k \leq 3n^2$. When $p_1 \geq 3$ there is a pebbling of G such that the following bounds hold simultaneously:*

$$T_l^{(L)}(\underline{p}, G) = \begin{cases} \Theta(n^3) & l = 1 \\ \Theta(n^3/\sqrt{s_{l-1}}) & 2 \leq l \leq k+1 \\ \Theta(n^2) & k+2 \leq l \leq L \end{cases}$$

PROOF We note that any graph G consistent with the classical matrix multiplication algorithm has $|V^*| = \Theta(n^3)$. We consider the three lower bounds in turn. The lower bound in the first case follows from the fact that we have to pebble $\Theta(n^3)$ vertices with level-1 pebbles to pebble the graph, namely, $2n^2$ input vertices, n^2 result vertices, n^3 product vertices, and $n^2(n-1)$ addition vertices.

The second lower bound follows from Corollary 3.1 and Lemma 4.1. The third lower bound follows from Corollary 3.2 and the fact that every input vertex in the Red-Blue Pebble Game must be pebbled at least once with blue pebbles.

We now give a pebbling strategy that simultaneously achieves these lower bounds up to constant multiplicative factors. The base case applies to $L = 2$. It uses the obvious algorithm suggested by Figure 1. Entries of the $n \times n$ matrix C are computed in $r_1 \times r_1$ blocks. These are formed by taking inner products of $r_1 \times r_1$ submatrices of A and B. Without loss of generality, assume that r_1 divides n. At no cost place level-2 pebbles on all $2n^2$ input vertices. Pebble an $r_1 \times r_1$ block of C by pebbling with level-1 pebbles those vertices associated with the inner product. With $s_1 = 3r_1^2$ (or $r_1 = \sqrt{s_1/3}$) level-1 pebbles, $s_1 \geq 3$, this inner product can be completed. During this pebbling level-1 pebbles are removed from submatrices of A and B and added to others. No level-2 output operations are needed except for the n^2 entries in C. The number of level-2 input operations per element of A and B is the number of times an $r_1 \times r_1$ submatrix is involved in an inner product, namely, n/r_1. Thus, a total of $2n^3/r_1$ level-2 input operations are performed, giving a combined total number of $O(n^3/\sqrt{s_1})$ I/O operations. The number of computation steps is $O(n^3)$ because each addition and multiplication is in a distinct addition tree and done once.

Now consider the multilevel case. Our strategy is to successively block C into $r_i \times r_i$ submatrices for $i = k, k-1, \ldots, 1$. To avoid memory fragmentation the integers r_i be chosen as shown below so they divide one another. They are also chosen relative to s_i so there are sufficiently many pebbles available to pebble $r_i \times r_i$ submatrices.

$$r_i = \begin{cases} \lfloor \sqrt{s_1/3} \rfloor & i = 1 \\ r_{i-1} \lfloor \sqrt{(s_i - i + 1)}/(\sqrt{3} r_{i-1}) \rfloor & i \geq 2 \end{cases}$$

Then, r_i divides r_j for $i \leq j$. Also, $\sqrt{(s_i - i + 1)/12} \leq r_i \leq \sqrt{(s_i - i + 1)/3}$ since $b/2 \leq a\lfloor b/a \rfloor \leq b$ when a and b are integers and $1 \leq a \leq b$. Thus, $s_i \geq 3r_i^2 + i - 1$. Let k be the largest integer such that $r_k^2 \leq n^2$. Without loss of generality assume that $r_k = n$, that is, we assume that A, B and C are $r_k \times r_k$ matrices. We now give an efficient pebbling strategy for G.

Our goal is to compute entries of C by pebbling output vertices of G. This is done recursively; we represent the $r_k \times r_k$ matrices A, B, and C as $m_k \times m_k$ block matrices, $m_k = r_k/r_{k-1}$, containing $r_{k-1} \times r_{k-1}$ submatrices. Each such submatrix

of C is the inner product of $r_{k-1} \times r_{k-1}$ submatrices of A with a like number of such submatrices of B, as suggested in Figure 1. The products of $r_{k-1} \times r_{k-1}$ matrices are computed recursively.

The pebbling is done using two sets of pebbles, a reserve set containing one pebble per level except the first, and the remainder. With the remaining pebbles, recursively pebble the product of two $r_j \times r_j$ submatrices A_1 and B_1 of A and B under the assumption that A_1 and B_1 have pebbles at level j or lower on their entries initially. Their product is obtained by pebbling inner products of $r_{j-1} \times r_{j-1}$ submatrices. This is done by advancing pebbles at level $j-1$ or lower to corresponding submatrices of A_1 and B_1, applying the matrix multiplication algorithm recursively, and pebbling the sum of two such matrices (except for the first step). This procedure is repeated until the lowest level is reached whereupon the base case applies.

When pebbles at levels j or lower are placed on inputs of A_1 and B_1, we exhaust pebbles at each level before using pebbles at a lower level. This guarantees that if a submatrix of A_1 or B_1 does not have all its input vertices covered with pebbles at level $j-1$ or less, there will be low-level pebbles that can be moved to them using the reserve pebbles without having to remove pebbles from other vertices. (Note that if a vertex carries a level-j pebble but the highest level remaining non-reserve pebble is a level-3 pebble, we can use reserve pebbles to put a level-3 pebble on the vertex.) Because $s_j \geq 3r_j^2 + j - 1$, there are sufficiently many non-reserve pebbles to carry out each inner product computation.

This pebbling procedure will eventually place level-1 pebbles on $r_1 \times r_1$ submatrices of A and B and they will be multiplied as in the base case. To free up pebbles to pebble other such submatrices, we may have to replace some level-1 pebbles with level-2 pebbles. By the definition of s_2, there are sufficiently many pebbles at level-2 or less to do this. Similarly, we may have to replace some level-$(j-1)$ pebbles with level-j pebbles. (Note that level-t pebbles can be used as level-j pebbles, $t < j$, if there are sufficiently many of them.)

Let's now determine the number of I/O and computation steps at each level. Since all non-input vertices of G are pebbled once, the number of computation steps is $O(n^3)$. I/O operations are done on input and output vertices, not on vertices corresponding to additions. Once an output vertex has been pebbled at the first level, reserve pebbles can be used to place a level-L pebble in it. Thus one output will be done on each the n^2 output vertices at each level.

Consider now only I/O operations on input vertices. If the product of two $r_j \times r_j$ submatrices A_1 and B_1 is to be pebbled, as many as r_j/r_{j-1} input and output operations may be necessary at level j for each of the r_j^2 entries of A_1 and B_1.

Thus, we may do as many as $2r_j^3/r_{j-1}$ I/O operations per matrix or a total of $4r_j^3/r_{j-1}$ I/O operations. No level-j I/O operations are done while pebbling submatrices. However, level-j I/O operations are done when multiplying $r_j \times r_j$ submatrices of an $r_{j+1} \times r_{j+1}$ matrix. In fact, the number of level-j I/O operations performed to multiply two $r_{j+1} \times r_{j+1}$ matrices is $(r_{j+1}/r_j)^3$ times the number to multiply two $r_j \times r_j$ submatrices. This follows because each submatrix is involved in r_{j+1}/r_j products and there are $(r_{j+1}/r_j)^2$ such submatrices in each matrix. Thus, the number of level-j I/O operations done on $r_{j+1} \times r_{j+1}$ matrices is at most $4r_{j+1}^3/r_{j-1}$. Generalizing this to the highest level, the number of level-j I/O operations is $O(n^3/r_{j-1})$ for $j \leq k$. It is $O(n^2)$ for $j > k$. □

We close this section with a lower bound for the I/O-limited version of the game. It is based on the following lower bound due to Grigoryev [10] on the number of times that pebbles are placed on input vertices in the Red Pebble Game. (See Savage and Swamy [15] for a translation of Grigoryev's proof.) As explained in Section 2, the Red Pebble Game is generally viewed as a one-level game but is actually a two-level game with the I/O limitation. The I/O time $T_2^{(2)}(S, G)$ is the number of times that inputs and outputs are pebbled.

Lemma 4.2 ([10]) *For any straight-line program for matrix multiplication, the number of times inputs must be pebbled in the Red Pebble Game when played with S red pebbles satisfies the following inequality:*

$$T_2^{(2)}(S, G) \geq \frac{n^3}{4(S+1)}$$

Theorem 4.2 *Let G be any dag associated with matrix multiplication of $n \times n$ matrices and let it be pebbled under the* **I/O limitation**. *If $S = s_{L-1} \leq n$, then the time to pebble G at the lth level, $T_l^{(L)}(\underline{p}, G)$, must satisfy the following relation for $1 \leq l \leq L$:*

$$T_l^{(L)}(\underline{p}, G) = \Omega(\frac{n^3}{S})$$

PROOF The lower bounds follow from Corollary 3.2 and Lemma 4.2. Matching upper bounds in general are not known. □

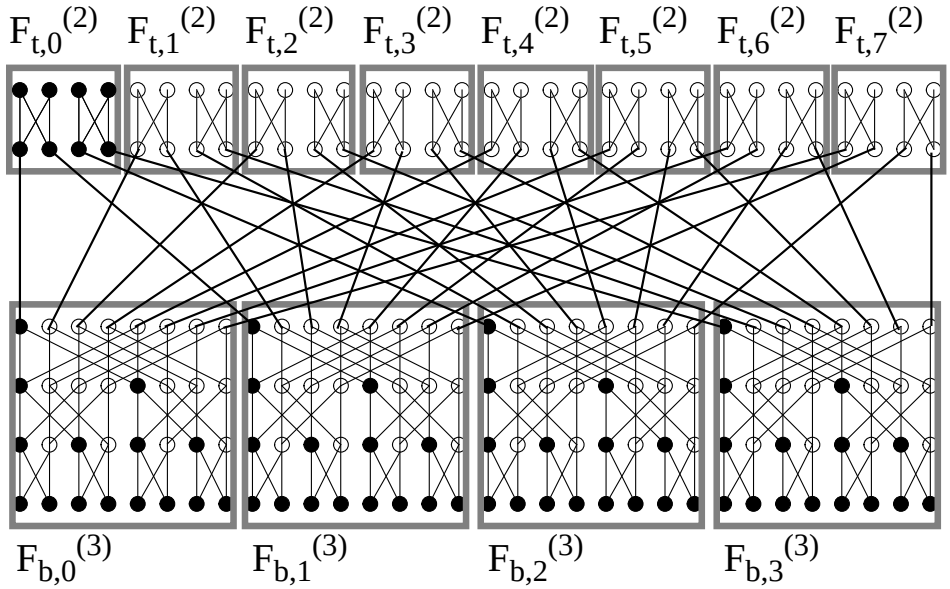

Figure 2: Decomposition of the FFT graph $F^{(5)}$.

4.2 The Fourier Transform

The (real) discrete Fourier Transform (DFT) on n inputs is defined by the matrix-vector multiplication $A\underline{x}$ where $\underline{x}$ is an n-tuple of reals and A is the Vandermonde matrix whose i,j entry is $a_{i,j} = w^{ij}$, $0 \leq i,j \leq n-1$. Here w is the nth root of unity.

The fast Fourier Transform (FFT) is a straight-line algorithm characterized by the recursively defined directed acyclic graph on $n = 2^d$ inputs, $F^{(d)}$, shown in Figure 2. The graph has n inputs in the bottom row and a total of $\log n + 1$ rows. Each FFT graph can be decomposed as a collection of FFT subgraphs, as indicated in Figure 2 and Lemma 4.3. If the outputs of the bottom FFT subgraphs are placed in rows in a matrix, the inputs to the top FFT subgraphs are obtained by a matrix transposition. We use this construction to produce good pebblings of the FFT graph.

Lemma 4.3 ([16]) *For $d \geq 2$ and each $1 \leq j \leq d-1$, the FFT graph $F^{(d)}$ on 2^d inputs can be represented as the composition of 2^{d-j} disjoint "top" FFT graphs on 2^j inputs $\{F_{t,p}^{(j)} \mid 0 \leq p \leq 2^{d-j}-1\}$ with 2^j disjoint "bottom" FFT graphs on 2^{d-j} inputs $\{F_{b,m}^{(d-j)} \mid 0 \leq m \leq 2^j-1\}$, where the mth input vertex of $F_{t,p}^{(j)}$ is the pth output vertex of $F_{b,m}^{(d-j)}$ for $0 \leq m \leq 2^j-1$ and $0 \leq p \leq 2^{d-j}-1$. The ith output vertex of $F^{(d)}$ is the rth output vertex of $F_{t,s}^{(j)}$ for the unique integers $r,s \geq 0$ such that $i = r2^{d-j} + s$ and $s < 2^{d-j}$.*

A bound on the S-span for the FFT is implicit in the work of Hong and Kung [11] and explicit in the work of Aggarwal and Vitter [3]. We state the bound and for completeness give a variant of the latter's proof.

Lemma 4.4 *The S-span of the FFT graph $F^{(d)}$ on $n = 2^d$ inputs satisfies $\rho(S, G) \leq 2S \log S$ when $S \leq n$.*

PROOF $\rho(S, G)$ is the maximum number of vertices of G that can be pebbled from some initial configuration of S pebbles on G under the assumption that any pebble placed on an output vertex stays there throughout the computation and the computation is minimal. Given that G is composed of two-input FFT graphs, the number of vertices pebbled is maximized by pebbling a vertex v_1 and its (matching) pair v_2 in a two-input FFT subgraph whenever either one of them can be pebbled. We let $\{p_i \mid 1 \leq i \leq S\}$ denote the S pebbles available to pebble G. We assign an integer cost $num(p_i)$ (initialized to zero) to the ith pebble p_i in order to derive an upper bound to the total number of pebble placements made on G.

Consider a matching pair of vertices v_1 and v_2 and their common predecessors u_1 and u_2. Suppose on the next step we can place a pebble on v_1. Then pebbles (call them p_1 and p_2) must reside on u_1 and u_2. Advance p_1 and p_2 to both v_1 and v_2. (In principle we need an additional pebble to advance two pebbles from the two inputs of a two-input butterfly to the two outputs. We allow this move without the third pebble because it only increases the number of possible moves.)

After advancing p_1 and p_2, if $num(p_1) = num(p_2)$, augment both $num(p_1)$ and $num(p_2)$ by 1; otherwise, augment the smaller by 1. Since all of the S pebbles remain on the graph during a pebbling that maximizes the number of vertices pebbled, $2\sum_{1 \leq i \leq S} num(p_i)$ is an upper bound on the total number of pebble placements.

Using the fact that all predecessors of u_1 and u_2 are disjoint, it is easy to show by induction on depth that at least $2^{\max(num(p_1), num(p_2))}$ of their predecessors carry pebbles initially. Since this number is at most S, it follows that $num(p_i) \leq \log S$ for all pebbles p_i. The lemma follows immediately. □

We now present upper and lower bounds on the number of I/O and computation steps to pebble the FFT in the standard MHG. In this case an unlimited number of highest-level pebbles is available to pebble the graph and they may be used for intermediate computations.

Theorem 4.3 *Let the FFT graph on $n = 2^d$ inputs, $F^{(d)}$, be pebbled in the* **standard MHG** *with resource vector $\underline{p}$. Let $s_l = \sum_{j=1}^{l} p_j$ and let k be the largest integer such that $s_k \leq n$. When $p_1 \geq 3$ there is a pebbling of $F^{(d)}$ such that the following relations are simultaneously satisfied:*[3]

$$T_l^{(L)}(\underline{p}, F^{(d)}) = \begin{cases} \Theta(n \log n) & l = 1 \\ \Theta(n \log n / \log s_{l-1}) & 2 \leq l \leq k+1 \\ \Theta(n) & k+2 \leq l \leq L \end{cases}$$

PROOF The first lower bound is obvious; every vertex in $F^{(d)}$ has to be pebbled a first time. The second follows from Corollaries 3.1, 3.2, Lemma 4.4 and the obvious lower bound on $|V^*|$. The third follows from the fact that pebbles at every level must be placed on each input and output vertex. We now exhibit one pebbling strategy giving upper bounds matching (up to a multiplicative factor) these lower bounds for all $1 \leq l \leq L$. Our pebbling strategy is based on Lemma 4.3.

We note that if e divides d, we can decompose $F^{(d)}$ into a collection of subgraphs $F^{(e)}$ each of which can be pebbled level by level with $2^e + 1$ level-1 pebbles without repebbling any vertex (if we slide pebbles, as discussed in Section 2). We use these observations to develop an hierarchical pebbling strategy for $F^{(d)}$.

We now define a non-decreasing sequence $\underline{d} = (d_1, d_2, \ldots, d_{L-1})$ of integers that we use to describe an efficient pebbling strategy for $F^{(d)}$. Let $d_0 = 1$ and $d_1 = \lfloor \log(s_1 - 1) \rfloor \geq 1$ where $s_1 = p_1 \geq 3$. Define m_r and d_r for $1 \leq r \leq L-1$ by

$$m_r = \left\lfloor \frac{\lfloor \log \min(s_r - 1, n) \rfloor}{d_{r-1}} \right\rfloor$$

$$d_r = m_r d_{r-1}$$

Note that d_r is divisible by d_{r-1}. Observe that $s_r \geq 2^{d_r} + 1$ when $s_r \leq n+1$. Also note that $\lfloor \log a \rfloor \geq (\log a)/2$ when $a \geq 1$ and that $a\lfloor b/a \rfloor \geq b/2$ when a and b are integers and $1 \leq a \leq b$. From this it follows that $d_r \geq (\log \min(s_r - 1, n))/4$.

These values of d_r are chosen to avoid memory fragmentation and to permit the decomposition of $F^{(d_r)}$ into $d_r/d_{r-1} = m_r$ stages with $2^{d_r - d_r - 1}$ copies of $F^{(d_r - 1)}$ on each stage. For example, $F^{(8)}$ can be decomposed into four stages with 64 copies of $F^{(2)}$ on each stage. Output vertices of a graph $F^{(2)}$ on one stage are input vertices of a graph $F^{(2)}$ on the next stage. (See Figure 2.) It simplifies the following discussion to assume that the d_r are distinct, that is, they form an increasing sequence, although the result holds even when this is not the case.

[3] All logarithms are to base 2.

We begin by describing a pebbling strategy to pebble an FFT graph $F^{(d_r)}$ on 2^{d_r} inputs using pebbles at levels 1 through $r+1$ when the number of pebbles available up to level r, s_r, satisfies $s_r \geq 2^{d_r}+1$. We show by induction on r that our $(r+1)$-level pebbling strategy pebbles $F^{(d_r)}$ using at most

$$n_l^{(r)} = 2\frac{d_r}{d_{l-1}}2^{d_r}$$

level-l I/O operations, $2 \leq l \leq r+1$, on non-input vertices. Each input vertex is pebbled once with pebbles at each level.

The basis for induction is $r = 1$ (a two-level MHG). In this case there are $s_1 \geq 2^{d_1}+1 \geq 3$ level-1 pebbles, enough to pebble $F^{(d_1)}$ without repebbling vertices or using level-2 pebbles. Level-2 pebbles are placed on inputs initially and on outputs when the pebbling is complete. Thus 2^{d_1} level-2 I/O operations are performed on non-input vertices, which establishes the basis.

Now assume that the inductive hypothesis holds for $r = R-1$ and $2 \leq l \leq R-1$. We show it holds for $r = R$ and $2 \leq l \leq R$. Per the previous discussion and Lemma 4.3, $F^{(d_R)}$ can be decomposed into $d_R/d_{R-1} = m_R$ stages with $2^{d_R-d_{R-1}}$ copies of $F^{(d_{R-1})}$ on each stage. We assume there are $s_l \geq 2^{d_l}+1$ pebbles at levels l or less for $1 \leq l \leq R$. Our pebbling strategy is to advance pebbles to the outputs of the first-stage FFT subgraphs $F^{(d_{R-1})}$ using pebbles at levels $R-1$ or less and then use additional pebbles at level R to pebble the outputs of these subgraphs. Pebbles at lower levels are left in place on the outputs of these subgraphs whenever possible. After pebbles have been advanced to the outputs of these first-stage subgraphs (and removed from preceding vertices), we pebble subgraphs at the next stage using only pebbles at levels $R-1$ or less. We then add pebbles at level R as necessary on their outputs. We repeat this until we have placed pebbles on the outputs of the graph $F^{(d_R)}$. Since there are $2^{d_R-d_{R-1}}$ FFT subgraphs $F^{(d_{R-1})}$ of $F^{(d_R)}$ in each of d_R/d_{R-1} stages, we have

$$n_l^{(R)} \leq 2^{d_R-d_{R-1}}\frac{d_R}{d_{R-1}}n_l^{(R-1)}$$

for $2 \leq l \leq R-1$. Also, $n_R^{(R)} \leq 22^{d_R}d_R/d_{R-1}$ level-R I/O operations are done since there 2^{d_R} vertices on each row of $F^{(d_R)}$ and at most two I/O operations are done on at most d_R/d_{R-1} of these rows. This pebbling strategy pebbles each vertex once with level-1 pebbles. It also pebbles input vertices with pebbles at all levels once.

Now let k be the largest integer such that $s_k \leq n = 2^d$. To pebble $F^{(d)}$, decompose it into $\lceil d/d_k \rceil$ stages such that each stage, except possibly the last, contains 2^{d-d_k}

copies of the FFT subgraph $F^{(d_k)}$. The last stage has 2^{d-d^*} copies of the FFT subgraphs $F^{(d^*)}$ of depth $d^* = d - (\lceil d/d_k \rceil - 1)d_k$.

After pebbling each subgraph $F^{(d_k)}$ in the first stage by the strategy given above, place pebbles at level $k+1$ (or less) on their outputs. Since $s_{k+1} \geq n+1$, there are enough pebbles available to pebble all n outputs of such subgraphs in the first stage. Repeat this process with subgraphs at each of the first $\lceil d/d_k \rceil - 1$ stages. To pebble the remaining vertices, those in subgraphs $F^{(d^*)}$, treat groups of $2^{d_k - d^*}$ such subgraphs as prefixes of the subgraph $F^{(d_k)}$ and pebble them in the same fashion. There will be 2^{d-d_k} such groups and the time to pebble each one using pebbles at the $k+1$ levels will be at most that to pebble $F^{(d_k)}$. From this it follows that the total number of level-l I/O operations, $2 \leq l \leq k+1$, is at most

$$(\lceil d/d_k \rceil) 2^{d-d_k} \frac{2d_k}{d_{l-1}} 2^{d_k} = \lceil d/d_k \rceil \frac{2d_k}{d_{l-1}} 2^d \leq \frac{4d2^d}{d_{l-1}}$$

The desired conclusion follows from the observation above that $d_l \geq (\log(s_l - 1))/4$ when $s_l - 1 \leq n$ and that $s_{l-1} - 1 \geq s_{l-1}/2$.

For $l \geq k+2$, level-l pebbles are used only on the input and output vertices. It follows that for these cases the number of I/O operations is at most $O(n)$. □

To derive the bounds when the I/O limitation applies we use the following result which applies to any linear straight-line program for the DFT.

Lemma 4.5 ([18]) *For any linear straight-line program graph G for the DFT, the number of times input vertices are pebbled with red pebbles in the Red Pebble Game when played with S red pebbles satisfies the following inequality:*

$$T_2^{(2)}(S, G) \geq n^2/4S + S$$

We now state lower bounds on the number of I/O and computation steps for DFTs realized by linear straight-line algorithms for the DFT. Corresponding upper bounds are stated for FFT graphs. The lower bounds are larger than the lower bounds of Theorem 4.3 when the total number of pebbles at all levels but the highest, S, satisfies $S \leq n/\log n$ where n is the number of inputs to the DFT.

Theorem 4.4 *Let $FFT(n)$ be any dag associated with the DFT on n inputs when realized by a linear straight-line program. Let $FFT(n)$ be pebbled* **under the I/O limitation** *with resource vector $\underline{p}$. Let $s_l = \sum_{j=1}^{l} p_j$. If $S = s_{L-1} \leq n$, then the*

time to pebble $FFT(n)$ at the lth level, $T_l^{(L)}(\underline{p}, FFT(n))$, must satisfy the following relation for $1 \leq l \leq L$:

$$T_l^{(L)}(\underline{p}, FFT(n)) = \Omega(\frac{n^2}{S})$$

Also, when $n = 2^d$, there is a pebbling of the FFT graph $F^{(d)}$ such that the following relations hold simultaneously:

$$T_l^{(L)}(\underline{p}, F^{(d)}) = \begin{cases} O(\frac{n^2}{S} + n \log S) & l = 1 \\ O(\frac{n^2}{S} + n \frac{\log S}{\log s_{l-1}}) & 2 \leq l \leq L \end{cases}$$

when $S \geq 2 \log n$.

PROOF The lower bound in the first equation follows directly from Corollary 3.2 and Lemma 4.5. We now show that these lower bounds can be achieved simultaneously on $F^{(d)}$ for $1 \leq l \leq L$ when the I/O limitation applies.

Recall that the two-level MHG under the I/O limitation is exactly the Red Pebble Game. We begin by describing the construction given in [16] for this game and use it as a basis for constructing a pebbling in the MHG.

Consider the decomposition of $F^{(d)}$ of Lemma 4.3 illustrated in Figure 2. In the Red Pebble Game, level-2 pebbles reside initially on input vertices. Level-1 pebbles are placed on inputs and advanced to outputs where they are replaced by level-2 pebbles. When the number of level-1 pebbles, p_1, is $2^j + d - j \leq n$, we can pebble all 2^j inputs to the "top" subgraph $F_{t,p}^{(j)}$ by pebbling the binary subtrees on 2^{d-j} inputs in the 2^j "bottom" subgraphs $F_{b,m}^{(d-j)}$, each subtree being rooted on an input to $F_{t,p}^{(j)}$. This is illustrated in Figure 2 for $p = 0$ by the black vertices. Since each subtree can be pebbled with $d - j + 1$ level-1 pebbles in minimal time, all inputs to $F_{t,p}^{(j)}$ can be pebbled with $2^j + d - j$ level-1 pebbles in time $2^j(2^{d-j+1} - 1)$.

Once inputs to a top subgraph $F_{t,p}^{(j)}$ have been pebbled, the remaining vertices in these subgraphs can be pebbled with level-1 pebbles level-by-level with $2^j + 1$ pebbles in time $j2^j$. Thus, the total time to pebble $F_{t,p}^{(j)}$ and predecessor vertices with level-1 pebbles is at most $2^j(j + 2^{d-j+1} - 1)$. Since this must be done for each of the 2^{d-j} subgraphs $F_{t,p}^{(j)}$, the total time to pebble $F^{(d)}$ with level-1 pebbles is at most $2^d(j + 2^{d-j+1} - 1)$.

Transitions of pebbles from the second to the first level occur on each input a total of 2^{d-j} times, once per subgraph $F_{t,p}^{(j)}$. A transition of pebbles from the first

to the second level occurs once on each output vertex. Thus, a total of $2^d(2^{d-j}+1)$ transitions occur between the first and second levels.

A pebbling for the L-level MHG is now given that is patterned after the above pebbling for the Red Pebble Game. Consider the above representation of $F^{(d)}$. Let q be the largest integer such that $S \geq 2^q + d - q$. Pebble the binary subtrees on 2^{d-q} inputs in the 2^q "bottom" subgraphs $F_{b,m}^{(d-q)}$ as follows: On each input vertex level-L pebbles are replaced by pebbles at all levels down to and including the first level. Then level-1 pebbles are advanced in the order which minimizes the number of level-1 pebbles in the Red Pebble Game. It may be necessary to use pebbles at all levels to make these advances; however, each vertex in a subtree (of which there are $2^{d-q+1}-1$) experiences at most two transitions at each level in the hierarchy. In addition, each vertex in a bottom tree is pebbled once with a level-1 pebble in a computation step. Therefore, the number of level-l transitions on vertices in the subtrees is at most $2^{d+1}(2^{d-q+1}-1)$ for $2 \leq l \leq L$, since this pebbling of 2^q subtrees is repeated 2^{d-q} times.

Once the inputs to a given subgraph $F_{t,p}^{(q)}$ have been pebbled, the subgraph itself is pebbled in the manner indicated in Theorem 4.3 using $O(q2^q/\log s_{l-1})$ pebbles at each level l for $2 \leq l \leq L$. Since this is done for each of the 2^{d-q} subgraphs $F_{t,p}^{(q)}$, it follows that on the top FFT subgraphs a total of $O(q2^d/\log s_{l-1})$ level-l transitions occur, $2 \leq l \leq L$. In addition, each vertex in a graph $F_{t,p}^{(q)}$ is pebbled once with a level-1 pebble in a computation step.

It follows that at most

$$T_l^{(L)}(\underline{p}, F^{(d)}) = O(2^d(2^{d-q+1}-1) + \frac{q2^d}{\log s_{l-1}})$$

level-l I/O operations occur for $2 \leq l \leq L$ as well as

$$T_1^{(L)}(\underline{p}, F^{(d)}) = O(2^d(2^{d-q+1}-1) + q2^d)$$

computation steps. It is left to the reader to verify that $2^q < 2^q + d - q \leq S < 2^{q+1} + d - q - 1 \leq 42^q$ when $q + 1 \geq \log d$ (this is implied by $S \geq 2d$), from which the result follows.□

4.3 Permutation and Merging Networks

Consider next merging networks $BS^{(d)}$ based on bitonic sorting networks [12, pp. 632] and permutation networks $P^{(d)}$ based on three back-to-back FFT graphs [21].

Replacing comparators in $BS^{(d)}$ with two-input butterfly graphs produces an FFT with edge directions reversed. It is immediate from the layered cross-product representation of FFT graphs [9] that $BS^{(d)}$ is isomorphic to the FFT graph. Thus, the bounds for the FFT apply directly to $BS^{(d)}$.

From Lemma 3.2 it follows that lower bounds for the FFT apply to $P^{(d)}$ because $P^{(d)}$ reduces to $F^{(d)}$. Since our pebbling strategy for the FFT graph in the standard MHG advances pebbles level-by-level, using higher-level pebbles sparingly to achieve the lower bounds, it follows that the pebbling strategy for the FFT graph is directly applicable to this graph.

Theorem 4.5 *Let $P^{(d)}$ be pebbled in the* **standard L-level MHG** *with resource vector $\underline{p}$. Let $s_l = \sum_{j=1}^{l} p_j$ and let k be the largest integer such that $s_k \leq n$. When $p_1 \geq 3$ there is a pebbling of $P^{(d)}$ such that the number of pebblings at each of the L levels, $\{T_l^{(L)} \mid 1 \leq l \leq L\}$, simultaneously satisfy the following relations:*

$$T_l^{(L)}(\underline{p}, P^{(d)}) = \begin{cases} \Theta(n \log n) & l = 1 \\ \Theta(n \log n)/\log s_{l-1}) & 2 \leq l \leq k+1 \\ \Theta(n) & k+2 \leq l \leq L \end{cases}$$

When the I/O limitation applies and s_{L-1} is too small, the I/O and computation time to pebble the permutation and sorting networks can be much larger than that to pebble the FFT graph. For example, Carlson and Tompa together show the following result, which implies that $P^{(d)}$, which has three FFT graphs back-to-back, requires at least as much I/O time:

Lemma 4.6 ([5,18,19]) *Let G be the graph consisting of two back-to-back copies of the FFT graph $F^{(d)}$ on $n = 2^d$ inputs. Then the number of second-level I/O operations when the* **I/O limitation** *applies in the Red Pebble Game when played with S level-1 pebbles satisfies the following inequality:*

$$T_2^{(2)}(S, G) \geq \Omega(n^3/S^2 + (n^2 \log n)/S)$$

4.4 Generalization to Block-I/O

Data is typically moved in blocks between memories in a hierarchy; data must fetched from the same block in which it was stored. Our lower bounds can be generalized to the block-I/O case by dividing the number of I/O operations by the size b_l of blocks moving between levels $l-1$ and l. This lower bound can be achieved for matrix multiplication because data is always read from the higher-level memory in the same way every time. For the FFT graph in the standard MHG instead of pebbling FFT

subgraphs on 2^{d_r} inputs, we pebble b_l FFT subgraphs on $2^{d_r}/b_l$ inputs (assuming that b_l is a power of 2). This allows all the data moving back and forth between memories in blocks to be used and accommodates the transposition mentioned at the beginning of Section 4.2. This provides an upper bound of $O(n \log n/(b_{l-1} \log(s_{l-1}/b_{l-1})))$ on the I/O time at level l. Clearly, when b_{l-1} is much smaller than s_{l-1}, say $b_{l-1} = O(\sqrt{s_{l-1}})$, the upper and lower bounds match.

5 Conclusions

We have introduced a new pebble game that captures essential features of multi-level memory hierarchies and allows us to study space-time tradeoffs in such hierarchies. Our model can handle block I/O with level-dependent block sizes. We generalize from two to many levels the Hong-Kung lower bound on the number of times data must move between adjacent levels in a memory hierarchy and give a new and much simpler method of proof. We exhibit problems for which we achieve the minimal number of I/O operations (up to a multiplicative factor) simultaneously at all levels of the hierarchy when data can be moved in blocks of size 1. With larger blocks we achieve minimal or near-minimal bounds. As a consequence, we can now answer more precise questions about the efficiency of a memory hierarchy for a class of problems.

These techniques can be extended to other problems. In the standard MHG upper bounds on the S-span of the graph of a problem provide lower bounds on the number of I/O operations in a multi-level memory hierarchy for that problem. Thus, the task of deriving lower bounds on I/O operations is reduced to that of computing the S-span of a graph.

Our lower bounds apply to straight-line computations. As suggested in the Introduction, it may not be possible to provide efficient prefetching in memory hierarchies with many levels if algorithms do a lot of branching. This may lead to recommendations that users write straight-line code, in which case our results apply.

The cost of increasing CPU speeds is growing rapidly, making parallel machines more attractive. To fully exploit parallelism it will be necessary to provide high-speed parallel memory hierarchy systems. We need a much better understanding of parallel I/O systems if we are going to meet this challenge.

Acknowledgements This work was begun while the author was on sabbatical leave in the Department of Computer Science, Warwick University, Coventry, England. The author gratefully acknowledges the hospitality and support of the faculty and staff of the Department, particularly Mike Paterson and the Chair, Graham Nudd. The author also gratefully acknowledges the comments of David Carlson.

Bibliography

[1] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir, "A Model for Hierarchical Memory," *Procs. 21st Annual ACM Symposium on Theory of Computing* (May 15-17, 1989), 305–314.

[2] A. Aggarwal, A. Chandra, and M. Snir, "Hierarchical Memory with Block Transfer," *Proc. 28th Annl. Symp. on Foundations of Computer Science* (October 1987), 204–216.

[3] A. Aggarwal and J. S. Vitter, "The Input/Output Complexity of Sorting and Related Problems," *Communications of the ACM* 31 (September 1988), 1116–1127.

[4] B. Alpern, L. Carter, and E. Feig, "Uniform Memory Hierarchies," *Proc. 31st Annual Symposium on Foundations of Computer Science* (October 22-24, 1990), 600–608.

[5] D. A. Carlson, "Time-Space Tradeoffs for Back-to-Back FFT Algorithms," *IEEE Trans. Computing* C-32 (1983), 585–589.

[6] D. A. Carlson, "Using Local Memory to Boost the Performance of FFT Algorithms on the CRAY-2 Supercomputer ," *The Journal of Supercomputing* 4 (1990), 345–356.

[7] D. A. Carlson, "Ultrahigh-Performance FFTs for the CRAY-2 and CRAY Y-MP Supercomputers," *The Journal of Supercomputing* 6 (1992), 107–116.

[8] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* 16 (March, 1990), 1–17.

[9] S. Even and A. Litman, "Layered Cross Product - A Technique to Construct Interconnection Networks," *Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures* (June 29 - July 1, 1992), 60–69.

[10] D. Y. Grigoryev, "An Application of Separability and Independence Notions for Proving Lower Bounds of Circuit Complexity," *Notes of Scientific Seminars, Steklov Math. Inst.* 60 (1976), 35–48.

[11] J. -W. Hong and H. T. Kung, "I/O Complexity: The Red-Blue Pebble Game," *Proc. 13th Ann. ACM Symp. on Theory of Computing* (May 11-13, 1981), 326–333.

[12] F. T. Leighton, in *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[13] M. H. Nodine and J. S. Vitter, "Large-Scale Sorting in Parallel Memories (Extended Abstract)," *Procs. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures* (July 21-24, 1991), 29–39.

[14] M. S. Paterson and C. E. Hewitt, "Comparative Schematology," *Proc. Proj. MAC Conf. on Concurrent Systems and Parallel Computation* (June 1970), 119–127.

[15] J. E. Savage and S. Swamy, "Space-Time Tradeoffs for Oblivious Integer Multiplication," in *Lecture Notes in Computer Science*, H. A. Maurer, ed., Springer-Verlag, Berlin, Heidelberg, New York, July, 1979, 498–504.

[16] J. E. Savage and S. Swamy, "Space-Time Tradeoffs on the FFT Algorithm," *IEEE Trans. on Info. Th.* IT-24 (Sept. 1978), 563–568.

[17] J. E. Savage and J. S. Vitter, "Parallelism in Space-Time Tradeoffs," in *Advances in Computing Research*, F. P. Preparata, ed., 1987, 117–146.

[18] M. Tompa, "Time-Space Tradeoffs for Computing Functions, Using Connectivity Properties of Their Circuits," *JCSS* 20 (1980), 118–132.

[19] M. Tompa, "Corrigendum: Time-Space Tradeoffs for Computing Functions, Using Connectivity Properties of Their Circuits," *JCSS* 23 (1981), 106.

[20] J. S. Vitter and E. A. M. Shriver, "Optimal Disk I/O with Parallel Block Transfer," *Procs. 22nd Annual ACM Symposium on Theory of Computing* (May 1990), 159–169.

[21] C. L. Wu and T. Y. Feng, "The Universality of the Shuffle-Exchange Network," *IEEE Trans. Computing* C-30 (May 1981), 324–332.