On the Limits of Cache-Oblivious Matrix Transposition [⋆]

Francesco Silvestri

Dipartimento di Ingegneria dell'Informazione, Università di Padova
Via Gradenigo 6/B, I-35131 Padova, Italy
francesco.silvestri@dei.unipd.it

Abstract Intuitively, a cache-oblivious algorithm implements an adaptive strategy which runs efficiently on any memory hierarchy without requiring previous knowledge of the parameters of the hierarchy. For this reason, cache-obliviousness is an attractive feature of an algorithm meant for a global computing environment, where software may be run on a variety of different platforms for load management purposes. In this paper we present a negative result on cache-obliviousness, namely, we show that an optimal cache-oblivious algorithm for the fundamental primitive of matrix transposition cannot exist without the *tall cache* assumption, which forces the (unknown) parameters of the memory hierarchy to satisfy a certain technical relation. Our contribution specializes the result of Brodal and Fagerberg for general permutations to matrix transposition, and provides further evidence that the tall cache assumption is often necessary to attain optimality in the context of cache-oblivious algorithms.

1 Introduction

A global computer infrastructure may be employed to provide dependable and cost-effective access to a number of platforms of varying computational capabilities, irrespective of their physical location or access point. This is, for example, the case of grid environments which enable sharing, selection, and aggregation of a variety of geographically distributed resources. In such a scenario, many different platforms can be available to run applications. For load management reasons, the actual platform(s) onto which an application is ultimately run, may be not known at the time when the application is designed. Hence, it is useful to design applications which adapt automatically to the actual platform they run on.

A typical modern platform features a hierarchical cascade of memories whose capacities and access times increase as they grow farther from the CPU. In order to amortize the larger cost incurred when referencing data in distant levels of the hierarchy, blocks of contiguous data are replicated across the faster levels, either

[⋆] This work was supported in part by the EU/IST Project "AEOLUS", and by MIUR of Italy under project "ALGONEXT".

automatically by the hardware (e.g., in the case of RAM-cache interaction) or by software (e.g., in the case of disk-RAM interaction). The rationale behind such a hierarchical organization is that the memory access costs of a computation can be reduced when the same data are frequently reused within a short time interval, and data stored at consecutive addresses are involved in consecutive operations, two properties known as *temporal* and *spatial locality of reference*, respectively.

Many models have been proposed to explicitly account for the hierarchical nature of the memory system. A two-level memory organization, intended to represent a disk-RAM hierarchy, is featured by the *External Memory* (EM) model of Aggarwal and Vitter [1], which has been extensively used in the literature to develop efficient algorithms that deal with large data sets, whose performance is mainly affected by the number of disk accesses (see [2] for an extensive survey on external memory algorithms). In this model, operations can only be performed on data residing in RAM, and data are transfered between the RAM and the disk in blocks of fixed size, under the explicit control of the program which decides where the blocks loaded from disk are placed in RAM and chooses the eviction policy.

Another popular model featuring a two-level memory organization, intended to represent a RAM-cache hierarchy, is the *Ideal Cache* (IC) model, introduced in [3]. As in the EM model, in the IC operations can only be performed on data residing in the fast level, the cache, and data are moved in fixed-size blocks (*cache lines*) between the RAM and the cache. However, unlike the EM model, block transfers are performed automatically by the hardware whenever an operand is referenced which is not in cache, and an optimal eviction policy is assumed. Algorithm design on the IC aims at minimizing the number of RAM-cache transfers, called *misses*, and the number of operations performed. The model has received considerable attention in the literature as the base for the design of so called *cache-oblivious* algorithms, which run efficiently without knowledge of the cache parameters, namely the cache size and the cache line size. Most importantly, cache-oblivious algorithms attaining an optimal number of misses on the IC can be shown, under certain circumstances, to attain optimal number of misses at all levels of any multi-level cache hierarchy [3].

For these reasons, efficient cache-oblivious algorithms are attractive in a global computing environment since they run efficiently on platforms featuring different memory hierarchies without requiring previous knowledge of the hierarchy parameters. A number of optimal cache-oblivious algorithms [3,4] and data structures [5] have been proposed in literature for important problems, e.g. sorting, matrix transposition and searching.

In several cases, optimality of cache-oblivious algorithms is attained under the so-called *tall cache assumption* which requires that the cache size in words be at least the square of the cache line size in words. In [3] the authors raised the natural question of whether there is a gap in asymptotic complexity between cache-oblivious algorithms and algorithms which know the parameters of the memory hierarchy. Only few works in the literature have investigated this issue.

Recently, Brodal and Fagerberg [6] have proved that an optimal cache-oblivious algorithm for sorting cannot exist without the tall cache assumption, and that an optimal cache-oblivious algorithm for general permutations does not exist regardless of the tall cache assumption. Impossibility results of a similar flavor have been proved by Bilardi and Peserico [7] in the context of DAG computations on a model of memory hierarchy which does not account for the spatial locality of reference, namely the HMM [8].

In this work, we specialize the results of [6] by showing that an optimal cache-oblivious algorithm for matrix transposition cannot exist without the tall cache assumption. To this purpose we follow a similar approach as the one employed in [6]. Specifically, let $\mathcal{A}$ be a cache-oblivious algorithm for matrix transposition and consider the two sequences of misses generated by the executions of $\mathcal{A}$ on two different ICs, where one model satisfies the tall cache assumption while the other does not. We simulate these two executions on the EM model and obtain a new EM algorithm for the matrix transposition problem. By adapting the argument used in [1] to bound from below the number of disk accesses of transposition in the EM model, we conclude that $\mathcal{A}$ cannot be optimal in both ICs.

The rest of the paper is organized as follows. In Section 2 we give a formal definition of the IC and EM models, and of the matrix transposition problem. Next, Section 3 describes the simulation technique while Section 4 applies this technique to prove the limits of cache-oblivious transposition. Section 5 concludes with some final remarks.

2 Preliminaries

2.1 The models

Two models of memory hierarchy are used in this work. The first one is the *Ideal Cache* $\mathsf{IC}(M, B)$, introduced by Frigo *et al.* in [3], which consists of an arbitrarily large main memory and a (data) cache of M words. The memory is split into blocks of B adjacent words called *B-blocks*, or simply blocks if B is clear from the context. The cache is fully associative and organized into M/B lines of B words: each line is empty or contains a B-block of the memory. The processor can only reference words that reside in cache: if a referenced word belongs to a block in a cache line, a *cache hit* occurs; otherwise there is a *cache miss* and the block has to be fetched into a line, replacing the line's previous content. The model adopts an optimal off-line replacement policy, i.e. it replaces the block whose next access is furthest in the future. We denote as *work complexity* the running time of an algorithm in the conventional RAM model, and as *cache (miss) complexity* the number of misses.

The concept of *cache-oblivious* algorithm is also introduced in [3], as an algorithm whose specifications are independent of cache parameters (M and B); it is easy to see that a cache-oblivious algorithm is formulated as a traditional RAM algorithm. We restrict our attention to optimal cache-oblivious algorithms, which reach the best cache complexity when executed on each IC model. Most of the cache-oblivious algorithms proposed in literature are optimal only under the *tall cache* assumption, i.e. $M \geq B^2$. On the contrary, we denote a cache where $M < B^2$ as *short cache*.

The second model is the *External Memory* $\mathsf{EM}(M, B)$ of Aggarwal and Vitter [1]; it features two levels of memory: a (fast) RAM memory of M words and a (slow) disk of unbounded size. As the memory in the IC, the disk storage is partitioned into blocks of B adjacent words called *B-blocks*, or simply blocks if B is clear from the context. The processor can only reference words that reside in memory. The movements between the memory and the disk are performed as follows: an input operation moves a B-block of the disk into B words of the memory, and an output operation moves B words of the memory into a B-block of the disk. The input/output operations (*I/Os*) are directly controlled by the algorithm through fetch and eviction operations, which is the main difference between IC and EM. We denote as *I/O complexity* of an algorithm the number of I/Os performed by the algorithm. We require an algorithm to store its output in the slow memory at the end of its execution; this will increase the I/O complexity of $O(M/B)$ I/Os, which is negligible when the input size is sufficiently large. It is easy to see that there is a relation between an I/O and a miss: a miss requires the fetching of a B-block from memory and the eviction of a B-block from cache if there is no empty line; hence a miss is equivalent to at most two I/Os in the EM, and for these reasons we will intentionally mix the two terms.

2.2 The matrix transposition problem

Let G be a $p \times q$ matrix and $H = G^T$ its transpose; specifically, H is a $q \times p$ matrix where $H(j,i) = G(i,j)$, $0 \leq i < p$ and $0 \leq j < q$. Without loss of generality, we suppose the size of each entry be one machine word; therefore the overall sizes of G and H are $N = pq$ words each. Since we are only interested to the limits of cache-oblivious matrix transposition, we may safely assume that p and q are much greater than M.

Lemma 1. *Any algorithm for matrix transposition requires at least*[1]

$$\Omega\left(\frac{N \log M}{B \log(1 + M/B)}\right)$$

I/Os (resp., misses) on $\mathsf{EM}(M, B)$ *(resp.,* $\mathsf{IC}(M, B)$*) if* $\min\{p, q\} \geq M$.

Proof. (Sketch) The proof is presented in [1] and the same argument carries through on the $\mathsf{IC}(M, B)$ model.

In [1] an optimal algorithm for matrix transposition is described which is parametric in B and M. Moreover, in [3] a cache-oblivious algorithm is presented, but its optimality is guaranteed only under the tall cache assumption. The interesting question arises of whether there exists an optimal cache-oblivious algorithm without the tall cache assumption, i.e. for each value of M and B. In the following sections, we will prove that such an algorithm does not exist.

[1] We use the following notation: log for base 2 logarithms and ln for natural logarithms.

3 The simulation technique

In this section we describe a technique for obtaining an EM algorithm from two executions of a cache-oblivious algorithm on two different IC models. The technique is presented in a general form and is a formalization of the *ad-hoc* one employed in [6] for proving the impossibility result for general permutations.

More precisely, consider two models $\mathcal{C}_1$=$\mathsf{IC}(M, B_1)$ and $\mathcal{C}_2 = \mathsf{IC}(M, B_2)$ where $B_1 < B_2$. Let $\mathcal{A}$ be a cache-oblivious algorithm for an arbitrary problem and let t_1 and t_2 be its cache complexities on the two models, respectively. We define an algorithm $\mathcal{A}'$ for $\mathsf{EM}(2M, B_2)$ which emulates in parallel the executions of $\mathcal{A}$ on both $\mathcal{C}_1$ and $\mathcal{C}_2$ and solves the same problem of $\mathcal{A}$.

Let us regard the RAM in $\mathsf{EM}(2M, B_2)$ as partitioned into two contiguous portions of size M each, which we refer to as $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. In turn, portion $\mathcal{M}_1$ is subdivided into blocks of B_1 words (which we call B_1*-rows*), and portion $\mathcal{M}_2$ is subdivided into blocks of B_2 words (which we call B_2*-rows*), so that we can establish a one-to-one mapping between the cache lines of $\mathcal{C}_1$ and the B_1-rows of $\mathcal{M}_1$, and a one-to-one mapping between the cache lines of $\mathcal{C}_2$ and the B_2-rows of $\mathcal{M}_2$. Algorithm $\mathcal{A}'$ is organized so that its I/Os occur exclusively between disk and $\mathcal{M}_2$ and coincide (except for some slight reordering) with the movements of cache lines between RAM and cache performed by $\mathcal{A}$ in $\mathcal{C}_2$; on the other hand, all operations prescribed by $\mathcal{A}$ are executed by $\mathcal{A}'$ on data in $\mathcal{M}_1$ [2]. The movements of cache lines between RAM and cache performed by $\mathcal{A}$ in $\mathcal{C}_1$ will be emulated by movements of B_1-rows between $\mathcal{M}_1$ and $\mathcal{M}_2$. (For convenience, we assume that B_2 is a multiple of B_1.)

Let us now see in detail how the execution of $\mathcal{A}'$ on $\mathsf{EM}(2M, B_2)$ develops. Initially all words in both $\mathcal{M}_1$ and $\mathcal{M}_2$ are empty, that is, filled with NIL values, and the EM disk contains the same data of $\mathcal{C}_2$ memory (or $\mathcal{C}_1$ indistinguishably) with the same layout (a one-to-one relation between B_2-blocks of $\mathcal{C}_2$ and B_2-blocks of the disk can be simply realized). Let α_i be the ith operation of $\mathcal{A}$, $i = 1 \dots h$. The execution of $\mathcal{A}$ on C_i, $1 \le i \le 2$, can be seen as a sequence $\mathcal{L}_i$ of operations interleaved with I/Os. Since operations in $\mathcal{L}_1$ and $\mathcal{L}_2$ are the same, we build a new sequence $\mathcal{L}$=$\Gamma_1^2 \Gamma_1^1 \alpha_1 \dots \Gamma_j^2 \Gamma_j^1 \alpha_j \dots \Gamma_h^2 \Gamma_h^1 \alpha_h \Gamma_{h+1}^2 \Gamma_{h+1}^1$. Each Γ_j^i, $1 \le j \le h+1$ and $1 \le i \le 2$, is defined as follows:

- Γ_1^i is the sequence of I/Os that precedes α_1 in $\mathcal{L}_i$.
- Γ_j^i, $1 < j \le h$, is the sequence of I/Os which are enclosed between α_{j-1} and α_j in $\mathcal{L}_i$.
- Γ_{h+1}^i is the sequence of I/Os performed after α_h in $\mathcal{L}_i$.

Note that a Γ_j^i can be empty. The length of $\mathcal{L}$, denoted as $|\mathcal{L}|$, is the sum of the number h of operations and the size of all Γ_j^i with $1 \le j \le h$ and $1 \le i \le 2$. Let $\mathcal{A}'$ be divided into $|\mathcal{L}|$ phases. The behaviour of the jth phase is determined by the jth element l_j of $\mathcal{L}$:

[2] Note that the operations of $\mathcal{A}$ do not include I/Os since block movements are automatically controlled by the machine. Moreover, $\mathcal{A}$'s operations are the same no matter whether execution is on $\mathcal{C}_1$ or $\mathcal{C}_2$.

- l_j *is an operation:* $\mathcal{A}'$ executes the same operation on $\mathcal{M}_1$.
- l_j *is an input of a B_2-block (i.e. an input of $\mathcal{L}_2$):* $\mathcal{A}'$ fetches the same B_2-block from the disk and moves it into the B_2-row of $\mathcal{M}_2$ associated with the line used in $\mathcal{C}_2$.
- l_j *is an input of a B_1-block (i.e. an input of $\mathcal{L}_1$):* let x be such B_1-block and x' be the B_2-block containing x. Since there is no prefetch in the IC model, the following operation of $\mathcal{A}$ requires an element in x; thus x' must be in $\mathcal{C}_2$ cache too. For this reason, we can assume that x' was, or has just been, fetched into a B_2-row of $\mathcal{M}_2$. $\mathcal{A}'$ copies the block x in the right B_1-row of $\mathcal{M}_1$ and replaces the copy of x in $\mathcal{M}_2$ with B_1 NIL values.
- l_j *is an output of a B_2-block (i.e. an output of $\mathcal{L}_2$):* $\mathcal{A}'$ moves the respective B_2-row of $\mathcal{M}_2$ to the disk, replacing it with B_2 NIL values.
- l_j *is an output of a B_1-block (i.e. an output of $\mathcal{L}_1$):* let x be such B_1-block and x' be the B_2-block containing x. If x' is still in $\mathcal{M}_2$, then $\mathcal{A}'$ copies x from $\mathcal{M}_1$ into x' and replaces x's row with B_1 NIL values. The second possibility (i.e. x' is not in $\mathcal{M}_2$) can be avoided since no operations are executed between the evictions of x' and x. If some operations had been executed, both blocks x and x' must be kept in cache (and so in $\mathcal{M}_1$ and $\mathcal{M}_2$). Therefore, we can suppose x was removed just prior to the eviction of x'; exactly, x is moved into x', x's row is filled with B_1 NIL values, and x' is evicted from $\mathcal{M}_2$ (see previous point).

It is easy to see that every operation of $\mathcal{A}$ can be executed by $\mathcal{A}'$ on $\mathcal{M}_1$, since there is a one to one relation between cache lines and matrix rows (excluding the B_1-blocks whose evictions from cache were anticipated, see fifth point). $\mathcal{M}_2$ is a "semimirror" of C_2, in the sense that it contains the same B_2-blocks of C_2 while $\mathcal{A}$ is being executed, except for those sub B_1-blocks which are also in $\mathcal{M}_1$. By rules 2 and 4, the I/O complexity of $\mathcal{A}'$ is $\Theta(t_2)$.

Let $K = t_1 B_1 / t_2$; it is easy to see that $K \leq B_2$. Indeed, if K was greater than B_2, an algorithm for $\mathcal{C}_1$ which requires $t_2 B_2 / B_1 < t_1$ IOs would be built from the execution of $\mathcal{A}$ on $\mathcal{C}_2$; but this is a contradiction since t_1 is optimal.

$\mathcal{A}'$ can be changed so that there are at most K words exchanged between $\mathcal{M}_1$ and a B_2-block in $\mathcal{M}_2$ before this block is removed from cache. It is sufficient to insert a dummy eviction/insertion of the B_2-block: in this way the I/O complexity is increased by a constant factor: $T = \Theta(t_2) + 2t_1 B_1 / K = \Theta(t_2)$.

We define the *working set* $W(t)$ after t I/Os as the content of $\mathcal{M}_1$ plus the words in the B_2-blocks of $\mathcal{M}_2$ that will be used by $\mathcal{A}'$ (moved to $\mathcal{M}_1$) before the large blocks are evicted. When $\mathcal{A}'$ fetches a B_2-block from the disk, we can suppose that at most K elements, which will be moved between $\mathcal{M}_1$ and the block, are immediately included in the working set.

4 Matrix transposition

In this section we prove that an optimal cache-oblivious algorithm for the matrix transposition problem does not exist without the tall cache assumption.

Let $\mathcal{A}$ be a cache-oblivious algorithm for matrix transposition and assume, for the sake of contradiction, that it attains optimal cache complexity without requiring the tall cache assumption. In particular, consider two models $\mathcal{C}_1$=$\mathsf{IC}(M, B_1)$ and $\mathcal{C}_2 = \mathsf{IC}(M, B_2)$ where $B_1 < B_2$ and let t_1 and t_2 be the cache complexities of $\mathcal{A}$ on the two models, respectively. We will show that B_1 and B_2 can be suitably chosen so that the tall cache assumption holds for $\mathcal{C}_1$ but not for $\mathcal{C}_2$, and that t_1 and t_2 cannot be both optimal, thus reaching a contradiction. To achieve this goal, we apply the simulation technique described in the previous section to $\mathcal{A}$, and we obtain an algorithm $\mathcal{A}'$ for $\mathsf{EM}(2M, B_2)$ which solves the matrix transposition problem. We then apply an adaptation of the lower bound argument by [1] to $\mathcal{A}'$, and we prove the impossibility of the simultaneous optimality of $\mathcal{A}$ on the two IC models.

More precisely, let the ith target group t_i, $1 \leq i \leq N/B_2$, be the records that will ultimately be in the ith B_2-block of the output matrix H (remember that H must be in the disk at the end of $\mathcal{A}'$). We define the following convex function:

$$f(x) = \begin{cases} x \log x & \text{if } x > 0; \\ 0 & \text{if } x = 0. \end{cases}$$

Let y be a B_2-block of the disk or a B_2-row of $\mathcal{M}_2$; the *togetherness rating* of y after t I/Os is defined as:

$$C_y(t) = \sum_{i=1}^{N/B_2} f(x_{i,y}),$$

where $x_{i,y}$ denotes the number of elements in y belonging to the ith target group. These elements are not included in the working set $W(t)$ and are not NIL symbol. We also define the togetherness rating for the working set $W(t)$:

$$C_W(t) = \sum_{i=1}^{N/B_2} f(m_i),$$

where m_i is the number of elements in the working set $W(t)$ which belong to the ith target group and are not NIL symbol. The *potential function* of $\mathcal{A}'$ after t I/Os is defined as:

$$POT(t) = C_W(t) + \sum_{y \in disk} C_y(t) + \sum_{y \in \mathcal{M}_2} C_y(t).$$

At the beginning of the algorithm, $POT(0) = 0$ since $N > \min\{p, q\} > B_2$; at the end of $\mathcal{A}'$, $POT(T) = N \log B_2$, where T is the I/O complexity of $\mathcal{A}'$.

Note that the above potential function is slightly different from the one defined in [1]: there, the potential function is given by the sum of disk and memory's togetherness ratings. We cannot use such definition because the real transposition is realized only in the working set (precisely in $\mathcal{M}_1$). Actually, if a block of the disk is moved to $\mathcal{M}_2$ and then brought back to the disk without its elements have been exchanged with $\mathcal{M}_1$'s elements, the potential function does not change.

We now analyze how an I/O made by $\mathcal{A}'$ improves the potential function. Suppose the tth I/O is an input and the B_2-block x is fetched into a B_2-row of $\mathcal{M}_2$. Before the tth input, the intersection between the block x and the working set $W(t-1)$ is empty; after the input, at most K elements of x are inserted into $W(t-1)$. We use the following notation:

– m_i: number of elements in the working set $W(t-1)$ belonging to the ith target group at time $t-1$;
– x_i: number of elements in block x belonging to the ith target group at time $t-1$;
– w_i: number of elements in the (at most) K words, inserted in $W(t-1)$, belonging to the ith target group.

The m_i, x_i and w_i values are limited by the constraints below:

$$\sum_{i=1}^{N/B_2} m_i \leq 2M - K \qquad \sum_{i=1}^{N/B_2} x_i \leq B_2 \qquad \sum_{i=1}^{N/B_2} w_i \leq K.$$

The potential increases of $\nabla POT(t)$ compared to $POT(t-1)$:

$$\begin{aligned}\nabla POT(t) = POT(t) - POT(t-1) &= C_W(t) + C_x(t) - C_W(t-1) - C_x(t-1)\\ &= \sum_{i=1}^{N/B_2} f(m_i + w_i) + f(x_i - w_i) - f(m_i) - f(x_i).\end{aligned}$$

Since $f(x_i - w_i) + f(w_i) = (x_i - w_i)\log(x_i - w_i) + w_i \log w_i \leq (x_i - w_i + w_i)\log x_i = f(x_i)$,

$$\begin{aligned}\nabla POT(t) &\leq \sum_{i=1}^{N/B_2} f(m_i + w_i) - f(m_i) - f(w_i)\\ &\leq \sum_{i=1}^{N/B_2} (m_i + w_i)\log(m_i + w_i) - m_i \log m_i - w_i \log w_i\\ &\leq \sum_{i=1}^{N/B_2} m_i \log \frac{m_i + w_i}{m_i} + w_i \log \frac{m_i + w_i}{w_i}.\end{aligned}$$

By a convexity argument, the increase in potential function is maximized when $m_i = (2M-K)/(N/B_2)$ and $w_i = K/(N/B_2)$, hence:

$$\begin{aligned}\nabla POT(t) &\leq (2M-K)\log \frac{2M - K + K}{2M - K} + K \log \frac{2M - K + K}{K}\\ &\leq K \log \left(1 + \frac{K}{2M-K}\right)^{\frac{2M-K}{K}} + K \log \left(\frac{2M}{K}\right)\\ &\leq \frac{K}{\ln 2} + K \log \frac{2M}{K} \in O\left(K \log \frac{M}{K}\right).\end{aligned}$$

Suppose now that the tth I/O is an output and the B_2-block x is evicted from a B_2-row of $\mathcal{M}_2$. Before the tth output, the intersection between the block x and the working set $W(t-1)$ contains at most K elements; after the output, at most K elements are removed from $W(t-1)$. As above, we use the following notation:

- m_i: number of elements in the working set $W(t-1)$ belonging to the ith target group at time $t-1$;
- x_i: number of elements in block x belonging to the ith target group at time $t-1$;
- w_i: number of elements in the (at most) K words, removed from $W(t-1)$, belonging to the ith target group.

The m_i, x_i and w_i values are limited by the constraints below:

$$\sum_{i=1}^{N/B_2} m_i \leq 2M \qquad \sum_{i=1}^{N/B_2} x_i \leq B_2 - K \qquad \sum_{i=1}^{N/B_2} w_i \leq K.$$

The potential increases of $\nabla POT(t)$ compared to $POT(t-1)$:

$$\begin{aligned}\nabla POT(t) = POT(t) - POT(t-1) &= C_W(t) + C_x(t) - C_W(t-1) - C_x(t-1)\\ &= \sum_{i=1}^{N/B_2} f(m_i - w_i) + f(x_i + w_i) - f(m_i) - f(x_i)\\ &\leq \sum_{i=1}^{N/B_2} (x_i + w_i)\log(x_i + w_i) - x_i \log x_i - w_i \log w_i,\end{aligned}$$

since $f(m_i - w_i) + f(w_i) \leq f(m_i)$. The increase in potential function is maximized when $x_i = (B_2 - K)/(N/B_2)$ and $w_i = K/(N/B_2)$, hence:

$$\nabla POT(t) \leq \frac{K}{\ln 2} + K \log \frac{B_2}{K} \in O\left(K \log \frac{B_2}{K}\right) = O\left(K \log \frac{M}{K}\right).$$

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be a tall and a short cache respectively, and let d, f, g be suitable positive constants. Since the I/Os of $\mathcal{A}'$ are $T = \Theta(t_2)$ and $t_1 = \Theta(N/B_1)$,

$$\begin{gathered} T\ \nabla POT \geq POT(T) \Longrightarrow dt_2 K \log \frac{M}{K} \geq N \log B_2\\ dt_2 \frac{t_1 B_1}{t_2} \log \frac{M t_2}{t_1 B_1} \geq N \log B_2 \Longrightarrow fN \log \frac{gMt_2}{N} \geq N \log B_2\\ \frac{gMt_2}{N} \geq B_2^{1/f} \Longrightarrow t_2 \in \Omega\left(N \frac{B_2^{1/f}}{M}\right).\end{gathered}$$

If $B_2 = \alpha M$ for a suitable constant $0 < \alpha < 1$, the above inequality becomes

$$t_2 \in \Omega\left(\frac{N}{M^{1-1/f}}\right) \in \omega\left(N \frac{\log M}{M}\right).$$

The miss complexity in $\mathcal{C}_2$ of an optimal algorithm for matrix transposition is $\Theta\left(N\frac{\log M}{M}\right)$, thus t_2 is not optimal. However, by the initial hypothesis on $\mathcal{A}$, we can deduce that t_2 is optimal and so we have a contradiction. We can conclude that a cache-oblivious algorithm for matrix transposition does not exist without the tall cache assumption.

5 Conclusions

In this work we have presented a simulation technique to yield an EM algorithm from two executions of the same cache-oblivious algorithm on different instantiations of the IC model. Our technique can be envisaged as a formalization and a generalization of the *ad-hoc* approach employed in [6] to prove negative results on cache-oblivious permuting. Successively, we have applied the simulation technique to matrix transposition. By further using an adaptation of the potential function employed in [1] to bound the I/O complexity of matrix transposition on EM, we were able to conclude that an optimal cache-oblivious algorithm for matrix transposition cannot exist without the tall cache assumption.

Apart from the result presented in this paper, to the best of the author's knowledge the only other impossibility results in the literature on optimal cache-obliviousness concern sorting and general permuting [6]. An interesting avenue for further research would be to address other fundamental problems, such as matrix multiplication, the Discrete Fourier Transform, or the realization of rational permutations, to expose any limitation intrinsic in their cache-oblivious realization. Moreover, a more profound understanding is still required of why the tall cache assumption is so crucial to obtain optimal cache-oblivious algorithms.

Acknowledgments

This paper benefited from useful discussions with Andrea Pietracaprina, Geppino Pucci and Gianfranco Bilardi.

References

1. Aggarwal, A., Vitter, J.: The input/output complexity of sorting and related problems. Communications of the ACM **31**(9) (1988) 1116–1127
2. Vitter, J.S.: External memory algorithms and data structures. ACM Comput. Surv. **33**(2) (2001) 209–271
3. Frigo, M., Leiserson, C., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. of 40th IEEE Symp. on Foundations of Computer Science. (1999) 285–298
4. Demaine, E.D.: Cache-oblivious algorithms and data structures. In: Lecture Notes from the EEF Summer School on Massive Data Sets. Lecture Notes in Computer Science, BRICS, University of Aarhus, Denmark (2002) to appear
5. Arge, L., Brodal, G.S., Fagerberg, R.: Cache-oblivious data structures. In Mehta, D., Sahni, S., eds.: Handbook of Data Structures and Applications. CRC Press (2005) 27

6. Brodal, G.S., Fagerberg, R.: On the limits of cache-obliviousness. In: Proc. of the 35th ACM Symp. on Theory of Computing. (2003) 307–315
7. Bilardi, G., Peserico, E.: A characterization of temporal locality and its portability across memory hierarchies. In: Proc. of 28th Int. Colloquium on Automata, Languages and Programming. LNCS 2076 (2001) 128–139
8. Aggarwal, A., Alpern, B., Chandra, A., Snir, M.: A model for hierarchical memory. In: Proc. of the 19th ACM Symp. on Theory of Computing. (1987) 305–314