Theoretical Computer Science 402 (2008) 221–233

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# On the limits of cache-oblivious rational permutations[☆]

Francesco Silvestri

*Dipartimento di Ingegneria dell'Informazione, Università di Padova, Via Gradenigo 6/B, 35131 Padova, Italy*

A R T I C L E   I N F O

*Keywords:*
Rational permutation
Cache-oblivious algorithm
Tall cache
Lower bound
Impossibility result
Memory hierarchy

A B S T R A C T

Permuting a vector is a fundamental primitive which arises in many applications. In particular, *rational permutations*, which are defined by permutations of the bits of the binary representations of the vector indices, are widely used. Matrix transposition and bit-reversal are notable examples of rational permutations. In this paper we contribute a number of results regarding the execution of these permutations in cache hierarchies, with particular emphasis on the cache-oblivious setting. We first bound from below the work needed to execute a rational permutation with an optimal cache complexity. Then, we develop a cache-oblivious algorithm to perform any rational permutation, which exhibits optimal work and cache complexities under the tall cache assumption. We finally show that for certain families of rational permutations (including matrix transposition and bit reversal) no cache-oblivious algorithm can exhibit optimal cache complexity for all values of the cache parameters. This latter result specializes the one proved by Brodal and Fagerberg for general permutations to the case of rational permutations, and provides further evidence that the tall cache assumption is often necessary to attain cache 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 [1] 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 not be 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 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 [2], which has been extensively used in literature to develop efficient algorithms that deal with large

[☆] This work was supported in part by the EU/IST Project "AEOLUS", and by MIUR of Italy under project "MAINSTREAM". A preliminary version of this paper appeared in the *Proc. of the Second Symposium on Trustworthy Global Computing*, LNCS 4661, Springer, 2006, pp. 233–243.

*E-mail address:* francesco.silvestri@dei.unipd.it.

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.04.036

data sets, whose performance is mainly affected by the number of disk accesses (see [3] for an extensive survey on EM algorithms). In this model, operations can only be performed on data residing in RAM, and data are transferred between RAM and 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 replacement policy.

Another popular model featuring a two-level memory organization, intended to represent a RAM-cache hierarchy, is the *Ideal Cache* (IC) model, introduced by Frigo et al. [4]. As in the EM model, in the IC model operations can only be performed on data residing in the fast level, the cache, and data are moved between RAM and cache in fixed-size blocks (*cache lines*). However, unlike the EM model, block transfers are performed automatically by the hardware whenever an operand which is not in cache is referenced, and an optimal off-line replacement policy is assumed. Algorithm design on the IC aims at minimizing the number of RAM-cache transfers, called *misses* (*cache complexity*), and the number of operations performed (*work complexity*). The model has received considerable attention in the literature as the base for the design of the 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 [4]. 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 [4,5] and data structures [6] have been proposed in literature for important problems, e.g. matrix transposition, permuting, sorting and searching.

In several cases, optimality of cache-oblivious algorithms is attained under the so-called *tall cache assumption* which requires the cache size in words to be at least the square of the cache line size in words. Recently, Brodal and Fagerberg [7] have proved that a cache-oblivious algorithm for sorting cannot be optimal for every set of the values of the cache parameters; moreover, they have shown that no cache-oblivious algorithm for permuting can exhibit optimal cache complexity for all values of the cache parameters, even under the tall cache assumption. Impossibility results of a similar flavor have been proved by Bilardi and Peserico [8] in the context of DAG computations on the Hierarchical Memory Model (HMM) [9], which does not account for the spatial locality of reference.

Permuting a vector is a fundamental primitive in many problems; in particular the so-called *rational permutations* are widely used. A permutation is rational if it is defined by a permutation of the bits of the binary representations of the vector indices. Matrix transposition, bit-reversal, and some permutations implemented in the Data Encryption Standard (DES) [10] are notable examples of rational permutations. There are some works in literature which deal with the efficient implementation of specific rational permutations in a memory hierarchy: e.g., Frigo et al. [4] propose a cache-oblivious algorithm for matrix transposition which is optimal under the tall cache assumption, Carter and Kang [11] give an optimal cache-aware algorithm for the bit-reversal of a vector. To the best of our knowledge, the only works in literature which propose a general approach to rational permutations are [12–15]. The first two papers propose efficient algorithms for performing any rational permutation in the blocked HMM [12] and in the Uniform Memory Hierarchy (UMH) model [13]. In [14,15] a lower bound on the number of disk accesses and an optimal algorithm for performing rational permutations are given for the Disk Array model [16] (which is similar to the EM one).

In this paper we first bound from below the work needed to execute any family of rational permutations in the IC model with an optimal cache complexity. For achieving this bound, we prove a technical lemma which is a generalization of the argument used in [2] to bound from below the number of disk accesses of matrix transposition in the EM model. Then, we propose a cache-oblivious algorithm for performing any rational permutation, which exhibits optimal cache and work complexities under the tall cache assumption. Finally, we show that for certain families of rational permutations (including matrix transposition and bit-reversal) there is no cache-oblivious algorithm which achieves optimality for every set of the values of the cache parameters. To this purpose we follow a similar approach to the one employed in [7]. Specifically, let $\mathcal{A}$ be a cache-oblivious algorithm for a specific class of rational permutations and consider the two sequences of misses generated by the executions of $\mathcal{A}$ in two different ICs, where one model satisfies a particular assumption while the other does not. We simulate these two executions in the EM model and obtain a new EM algorithm solving the same problem of $\mathcal{A}$. By adapting the technical lemma given in the argument for bounding from below the work complexity, we conclude that $\mathcal{A}$ cannot be optimal in both ICs.

The rest of the paper is organized as follows. In Section 2, we describe the IC and EM models, and give a formal definition of rational permutation. In Section 3, we provide the aforementioned lemma and the lower bound on the work complexity. In Section 4, we describe the cache-oblivious algorithm. In Section 5, we present the simulation technique and apply it to prove the limits of any cache-oblivious algorithm performing a given set of rational permutations. In Section 6, we conclude 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* model (IC($M$, $B$)), introduced by Frigo et al. in [4], 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 $M/B > 1$ 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 copied into a line, replacing the line's previous content. The model adopts an optimal off-line replacement policy, that is it replaces the block whose next access is furthest in the future [17]. We denote as *work complexity* the number of (elementary) operations, and as *cache complexity* the number of misses.

The concept of *cache-oblivious* (resp., *cache-aware*) algorithm is also introduced in [4], as an algorithm whose specification is independent (resp., dependent) of the cache parameters $M$ and $B$. It is easy to see that both cache-oblivious and cache-aware algorithms are formulated as traditional RAM algorithms. A *cache-optimal* (resp., *work-optimal*) algorithm denotes an algorithm which reaches the best cache (resp., work) complexity when executed on an $\mathsf{IC}(M, B)$, for each value of $M$ and $B$. A number of cache-oblivious algorithms proposed in literature are cache-optimal only under the *tall cache assumption*, that is $M \geq B^2$.

The second model is the *External Memory* model ($\mathsf{EM}(M, B)$) of Aggarwal and Vitter [2]. It features two levels of memory: a (fast) RAM memory of $M$ words and an arbitrarily large (slow) disk. As the main memory in the $\mathsf{IC}$, 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 RAM. Data transfers between RAM and disk are performed as follows: an *input operation* moves a $B$-block of the disk into $B$ words of the RAM, and an *output operation* moves $B$ words of the RAM into a $B$-block of the disk. The input/output operations (I/Os) are explicitly controlled by the algorithm, and this is the main difference between the $\mathsf{IC}$ and the $\mathsf{EM}$ models. We denote as the *I/O complexity* of an $\mathsf{EM}$ algorithm the number of I/Os performed by the algorithm. We require an algorithm to store its input (resp., output) in the disk at the beginning (resp., end) of its execution. There is a natural correspondence between I/Os in the $\mathsf{EM}$ model and cache misses in the $\mathsf{IC}$ model: 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 corresponds to at most two I/Os, and for this reason we will intentionally mix the two terms.

### 2.2. Rational permutations

An *N-permutation* $\Pi_N$ is a bijective function from and to the set $\{0, \ldots, N-1\}$. This paper focuses on the so-called *rational permutations* defined as follows. Let $N = 2^n$. Denote with $\sigma$ an $n$-permutation and with $(a^i_{n-1}, \ldots, a^i_0)$ the binary representation of the value $i \in \{0, \ldots, N-1\}$, where $a^i_0$ denotes the least significant bit (LSB). The rational $N$-permutation $\Pi^\sigma_N$ maps a value $i$ to the value whose binary representation is $(a^i_{\sigma(n-1)}, \ldots, a^i_{\sigma(0)})$, for each $i \in \{0, \ldots, N-1\}$. We call $\sigma$ the bit-permutation defining $\Pi^\sigma_N$ and denote with $\sigma^{-1}$ its inverse. Note that the inverse of $\Pi^\sigma_N$ is $\Pi^{(\sigma^{-1})}_N$.

Given an $n$-permutation $\sigma$ and an index $j$, with $0 \leq j < n$, we define the following sets of bit positions:

- *j-outgoing set*: $\Psi(j, \sigma) = \{k : (k < j) \wedge (\sigma^{-1}(k) \geq j)\}$;
- *j-incoming set*: $\Upsilon(j, \sigma) = \{k : (k \geq j) \wedge (\sigma^{-1}(k) < j)\}$.

We call a bit position in $\Psi(j, \sigma)$ (resp., $\Upsilon(j, \sigma)$) *j-outgoing bit position* (resp., *j-incoming bit position*). In order to clarify the meaning of the $j$-outgoing and $j$-incoming sets, let $K$ and $H$ be two integers in $\{0, \ldots, N-1\}$ with binary representations $(a^K_{n-1}, \ldots, a^K_0)$ and $(a^H_{n-1}, \ldots, a^H_0)$, respectively, and such that $H = \Pi^\sigma_N(K)$. Then, the set $\Psi(j, \sigma)$ contains the indices of the $j$ LSBs in the binary representation of $K$ which, by virtue of permutation $\sigma$, appear among the $(n-j)$ most significant bits (MSBs) in the binary representation of $H$. Similarly, the set $\Upsilon(j, \sigma)$ contains the indices of the $n-j$ MSBs in the binary representation of $K$ which appear among the $j$ LSBs in the binary representation of $H$. Note that the cardinalities of $\Psi(j, \sigma)$ and $\Upsilon(j, \sigma)$ are equal: indeed, if $\Psi(j, \sigma)$ bits among the $j$ LSBs are permuted in the $(n-j)$ MSBs, then $\Psi(j, \sigma)$ bits among the $(n-j)$ MSBs must be permuted in the $j$ LSBs; hence, by the definition of $\Upsilon(j, \sigma)$, $|\Psi(j, \sigma)| = |\Upsilon(j, \sigma)|$. We define the *j-outgoing cardinality* $\psi(j, \sigma)$ as the cardinality of $\Psi(j, \sigma)$ (or $\Upsilon(j, \sigma)$ equivalently).

Let $V$ be a vector of $N$ entries and $V[i]$ be the $i$-th entry of $V$, with $0 \leq i < N$; we call $i$ the index of entry $V[i]$. An algorithm performs the $N$-permutation $\Pi_N$ on $V$ if it returns a vector $U$, distinct from $V$, such that $U[i] = V[\Pi_N(i)]$ for each $i$, with $0 \leq i < N$. Note that $V[i]$ is permuted into $U[\Pi^{-1}_N(i)]$, where $\Pi^{-1}_N$ is the inverse of $\Pi_N$. We suppose that a machine word is large enough to contain a vector entry or the index of a vector entry, and that the entries of any vector are stored in consecutive memory locations, sorted by indices.[1]

Let $\Sigma$ be an infinite set of permutations which contains at most one $n$-permutation for each $n \in \mathbb{N}$. Note that each $n$-permutation $\sigma \in \Sigma$ defines a rational permutation $\Pi^\sigma_N$, with $N = 2^n$. An algorithm performs the rational permutations defined by $\Sigma$ if, when given in input an $n$-permutation $\sigma \in \Sigma$ and a vector $V$ of $N = 2^n$ entries, it performs $\Pi^\sigma_N$ on $V$. For each $N = 2^n$ such that there exists an $n$-permutation $\sigma \in \Sigma$, we denote by $\psi_\Sigma(j, N)$ the $j$-outgoing cardinality $\psi(j, \sigma)$.

For example, let $V$ be a vector representing a $\sqrt{N} \times \sqrt{N}$ matrix stored in a row-major layout. An algorithm which transposes the matrix stored in $V$ for each $N = 2^n$, $n$ even, is an algorithm which performs the rational permutations defined by $\Sigma^T = \{\sigma^T_n : \forall\, n > 1 \text{ and } n \text{ even}\}$, where

$$\sigma^T_n(j) = \left(j + \frac{n}{2}\right) \bmod n. \qquad (1)$$

[1] We also suppose that the first entry of a vector stored in the memory (resp., disk) of the $\mathsf{IC}(M, B)$ (resp., $\mathsf{EM}(M, B)$) model is aligned with a $B$-block.

Since the $j$-outgoing and $j$-incoming sets of $\sigma_n^T$ are

$$\Psi(j, \sigma_n^T) = \begin{cases} \varnothing & \text{if } j = 0 \\ \{0, \dots, j-1\} & \text{if } 0 < j \leq \frac{n}{2} \\ \{j - \frac{n}{2}, \dots \frac{n}{2} - 1\} & \text{if } \frac{n}{2} < j < n, \end{cases}$$

$$\Upsilon(j, \sigma_n^T) = \begin{cases} \varnothing & \text{if } j = 0 \\ \{\frac{n}{2}, \dots, \frac{n}{2} + j - 1\} & \text{if } 0 < j \leq \frac{n}{2} \\ \{j, \dots n-1\} & \text{if } \frac{n}{2} < j < n, \end{cases} \tag{2}$$

the $j$-outgoing cardinality of $\Sigma^T$ is

$$\psi_{\Sigma^T}(j, 2^n) = \min\{j, n-j\}. \tag{3}$$

In the same fashion, an algorithm for the bit-reversal of a vector is an algorithm which performs the rational permutations defined by $\Sigma^R = \{\sigma_n^R, \forall\, n \geq 1\}$, where

$$\sigma_n^R(j) = n - j - 1$$

The $j$-outgoing and $j$-incoming sets of a bit-permutation $\sigma_n^R$ are

$$\Psi(j, \sigma_n^R) = \begin{cases} \varnothing & \text{if } j = 0 \\ \{0, \dots, j-1\} & \text{if } 0 < j \leq \lfloor \frac{n}{2} \rfloor \\ \{0, \dots, n-j-1\} & \text{if } \lfloor \frac{n}{2} \rfloor < j < n, \end{cases}$$

$$\Upsilon(j, \sigma_n^R) = \begin{cases} \varnothing & \text{if } j = 0 \\ \{n-j, \dots n-1\} & \text{if } 0 < j \leq \lfloor \frac{n}{2} \rfloor \\ \{j, \dots, n-1\} & \text{if } \lfloor \frac{n}{2} \rfloor < j < n, \end{cases}$$

from which follows that the $j$-outgoing cardinality of $\Sigma^R$ is

$$\psi_{\Sigma^R}(j, 2^n) = \min\{j, n-j\}. \tag{4}$$

## 3. Lower bounds

In this section we derive a lower bound on the work complexity of any algorithm which performs a given family of rational permutations with optimal cache complexity. To this purpose, we prove a technical lemma which generalizes a technical result given in [2] for bounding from below the number of disk accesses of matrix transposition in the EM model. The lower bound on the cache complexity given in [14] can be proved as a corollary of this technical lemma. Finally, we prove that the cache-aware algorithm obtained by the one given in [15] exhibits optimal cache and work complexities when executed in an IC($M$, $B$), for each $M$ and $B$.

Let $\Sigma$ be the set of permutations defined in Section 2.2, and consider an algorithm which is able to perform any rational $N$-permutation defined by $\Sigma$ on a vector of $N$ entries. We denote with $Q_\Sigma(N, M, B)$ the cache complexity of this algorithm, and with $V$ and $U$ the input and output vectors, respectively (recall that the two vectors $V$ and $U$ are distinct and each one is stored in $N$ consecutive memory locations).

Let the $i$-th *target group*, $1 \leq i \leq N/B$, be the set of $V$'s entries that will ultimately be in the $i$-th constituent $B$-block of $U$. We define the following convex function[2]:

$$f(x) = \begin{cases} x \log x & \text{if } x > 0 \\ 0 & \text{if } x = 0. \end{cases} \tag{5}$$

Let $\gamma$ be a $B$-block of the memory (if there is a copy of the block in cache, we refer to that copy). The *togetherness rating of* $\gamma$ ($C_\gamma(q)$) and the *potential function* ($POT(q)$) after $q$ misses are defined as:

$$C_\gamma(q) = \sum_{i=1}^{N/B} f(x_{\gamma,i}), \qquad POT(q) = \sum_{\forall B\text{-block } \gamma} C_\gamma(q),$$

where $x_{\gamma,i}$ denotes the number of entries in $\gamma$ belonging to the $i$-th target group just before the $(q+1)$-st miss.[3] As proved in [14], the values of the potential function at the beginning and at the end of the algorithm are given by the following equations:

$$POT(0) = N \log\left(\frac{B}{2^{\psi_\Sigma(\log B, N)}}\right), \qquad POT(Q_\Sigma(N, M, B)) = N \log B. \tag{6}$$

[2] We denote with log the binary logarithm and with $e$ the Napier's constant.

[3] If there is no $(q+1)$-st miss, we consider the end of the algorithm; we use this convention whenever the $(q+1)$-st miss is not defined.

Let $\Delta POT(q)$ denote the increase in potential due to the $q$-th miss, with $1 \leq q \leq Q_\Sigma(N, M, B)$, that is $\Delta POT(q) = POT(q) - POT(q-1)$. The following lemma provides an upper bound on $\Delta POT(q)$, that is the maximum increase due to a rearrangement of the entries in cache after the $q$-th miss.

**Lemma 1.** *Let $\gamma$ be the block fetched into the cache as a consequence of the $q$-th miss, and $C$ be the set of at most $M/B - 1$ blocks residing in cache with $\gamma$. Denote with $W$ the number of entries that are in $\gamma$ just before the $q$-th miss and are in a block belonging to $C$ just before the $(q+1)$-st miss, or vice versa. Then,*

$$\Delta POT(q) \leq B + W \log \frac{2eM}{W} \tag{7}$$

*for each $q$, with $1 \leq q \leq Q_\Sigma(N, M, B)$.*

**Proof.** If there is no empty cache line when $\gamma$ is fetched, then a block is evicted from the cache, but this operation does not affect the potential function. Block $\gamma$ exchanges entries with blocks in $C$ without incurring any miss; then, at most $M/B$ blocks increase their togetherness ratings before the next miss. We focus on data exchanged between $\gamma$ and blocks in $C$, since the increase in the potential function due to rearrangements between two blocks $\alpha$ and $\beta$ in $C$ was considered when $\beta$ was fetched in cache (if we suppose that $\beta$ was fetched after $\alpha$). We use the following notation:

- $m_{\alpha,i}$: number of entries in block $\alpha$ belonging to the $i$-th target group just before the $q$-th miss, with $\alpha \in C \cup \{\gamma\}$ and $1 \leq i \leq N/B$;
- $w_{\alpha,\beta,i}$: number of entries belonging to the $i$-th target group, which are in block $\alpha$ just before the $q$-th miss, and are in block $\beta$ just before the $(q+1)$-st miss, with $\alpha, \beta \in C \cup \{\gamma\}$, $\alpha \neq \beta$ and $1 \leq i \leq N/B$. (Actually, we are interested only in $w_{\gamma,\alpha,i}$ and $w_{\alpha,\gamma,i}$, with $\alpha \in C$.)

We partition the target groups into two sets $P$ and $R$: the $i$-th target group belongs to $P$ if and only if $\sum_{\alpha \in C}(w_{\alpha,\gamma,i} - w_{\gamma,\alpha,i}) \geq 0$, while it belongs to $R$ otherwise. Let:

$$W_P = \sum_{i \in P} \sum_{\alpha \in C} w_{\alpha,i} \qquad W_R = \sum_{i \in R} \sum_{\alpha \in C} \tilde{w}_{\alpha,i},$$

where $w_{\alpha,i} = w_{\alpha,\gamma,i} - w_{\gamma,\alpha,i}$ and $\tilde{w}_{\gamma,i} = w_{\gamma,\alpha,i} - w_{\alpha,\gamma,i}$. Note that $W_R + W_P \leq W$. The $m_{\alpha,i}$ values are limited by the constraints below:

$$\sum_{i \in P} m_{\gamma,i} \leq B - W_P, \qquad \sum_{\alpha \in C} \sum_{i \in R} m_{\alpha,i} \leq M - W_R. \tag{8}$$

By the definition of the convex function $f$ (Eq. (5)), the increase in potential is

$$\Delta POT(q) = \sum_{\alpha \in C \cup \{\alpha\}} (C_\alpha(q) - C_\gamma(q-1)) \leq \Delta POT_R(q) + \Delta POT_P(q),$$

where

$$\Delta POT_R(q) = \sum_{i \in R} \sum_{\alpha \in C} \left[ f\left(m_{\alpha,i} + \tilde{w}_{\alpha,i}\right) - f\left(m_{\alpha,i}\right) - f\left(\tilde{w}_{\alpha,i}\right) \right], \tag{9}$$

$$\Delta POT_P(q) = \sum_{i \in P} \left[ f\left(m_{\gamma,i} + \sum_{\alpha \in C} w_{\alpha,i}\right) - f\left(m_{\gamma,i}\right) - \sum_{\alpha \in C} f\left(w_{\alpha,i}\right) \right]. \tag{10}$$

By Inequalities (8) and the properties of concave functions, an upper bound on $\Delta POT_R(q)$ is obtained by setting $m_{\alpha,i} = (M - W_R)/(|R||C|)$ and $\tilde{w}_{\alpha,i} = W_R/(|R||C|)$. Then:

$$\Delta POT_R(q) \leq (M - W_R) \log \left(1 + \frac{W_R}{M - W_R}\right) + W_R \log \frac{M}{W_R} \leq W_R \log \frac{eM}{W_R}, \tag{11}$$

since $(1 + 1/x)^x \leq e$ if $x \geq 1$. In the same fashion, an upper bound on $\Delta POT_P(q)$ is obtained by plugging $m_{\gamma,i} = (B - W_P)/|P|$ and $w_{\alpha,i} = W_P/(|P||C|)$ into Eq. (10):

$$\Delta POT_P(q) \leq (B - W_P) \log \frac{B}{B - W_P} + W_P \log \frac{B}{W_P} + W_P \log |C| \leq B + W_P \log \frac{M}{W_P}. \tag{12}$$

By Eqs. (11) and (12), and the fact that $W_R + W_P \leq W$, we derive the following upper bound:

$$\Delta POT(q) \leq B + W_P \log \frac{M}{W_P} + W_R \log \frac{eM}{W_R} \leq B + W \log \frac{2eM}{W}. \quad \square$$

**Corollary 2.** *The increase in the potential function due to the $q$-th miss, with $1 \leq q \leq N/B$, is upper bounded by $2B \log \frac{2eM}{B}$.*

**Proof.** When a block $\gamma$ is fetched into the cache, at most $2B$ entries are exchanged between $\gamma$ and the other blocks residing in cache before the $(q+1)$-st miss. The corollary follows Lemma 1 by setting $W = 2B$. $\square$

Lemma 1 allows us to derive an alternative proof of the lower bound proved in [14], as reported below.

**Theorem 3** ([14]). *Let $\Sigma$ be an infinite set of permutations which contains at most one $n$-permutation for each $n \in \mathbb{N}$, and let $N = 2^n$. An algorithm which performs any rational $N$-permutation defined by an $n$-permutation in $\Sigma$ requires*

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

*misses in an* IC$(M, B)$, *for each value of* $M$ *and* $B$.[4]

**Proof.** The theorem follows by Corollary 2 and Eq. (6) because

$$\sum_{q=1}^{Q_\Sigma(N,M,B)} \Delta POT(q) \geq POT(Q_\Sigma(N, M, B)) - POT(0). \quad \square$$

An obvious lower bound on the work complexity of an algorithm performing rational permutations is $\Omega(N)$ since $N$ entries have to be moved from the input vector $V$ to the output vector $U$. Moreover, this work complexity is yielded by the naïve algorithm which moves each entry $V[\Pi_\sigma(i)]$ directly into $U[i]$, but this algorithm is not cache-optimal. We wonder whether there is a cache-optimal IC algorithm whose work complexity is $\Theta(N)$ for each value of IC parameters. The following theorem states that such an algorithm cannot exist.

**Theorem 4.** *Let $\Sigma$ be an infinite set of permutations which contains at most one $n$-permutation for each $n \in \mathbb{N}$, and let $N = 2^n$. Consider an algorithm which performs any rational $N$-permutation defined by an $n$-permutation in $\Sigma$ and whose cache complexity is*

$$Q_\Sigma(N, M, B) \in \Theta\left(\frac{N\psi_\Sigma(\log B, N)}{B\log(M/B)} + \frac{N}{B}\right),$$

*in an* IC$(M, B)$, *for each value of $M$ and $B$. Then its work complexity is*

$$W_\Sigma(N, M, B) \in \Omega\left(\frac{N\psi_\Sigma(\log B, N)}{\log(M/B)} + N\right) \tag{13}$$

*when $M/B > b$, for a suitable constant $b > 1$.*

**Proof.** If $\psi_\Sigma(\log B, N) \leq \log(M/B)$, Eq. (13) becomes the $\Omega(N)$ lower bound. Suppose $\psi_\Sigma(\log B, N) > \log(M/B)$, and let $c$ and $d$ be two suitable constants such that

$$c\frac{N\psi_\Sigma(\log B, N)}{B\log(M/B)} \leq Q_\Sigma(N, M, B) \leq d\frac{N\psi_\Sigma(\log B, N)}{B\log(M/B)}. \tag{14}$$

Denote with $Q'_\Sigma(N, M, B)$ the number of misses, each of which increases the potential by at least $(B/2d)\log(M/B)$. We claim that $Q'_\Sigma(N, M, B) = \Theta(Q_\Sigma(N, M, B))$. Let $\Delta POT$ be the upper bound given in Corollary 2 on the increase in the potential function due to a miss, and let $\Delta POT_1 = (B/2d)\log(M/B)$. Then,

$$\begin{aligned} POT(Q) - POT(0) &\leq \left(Q_\Sigma(N, M, B) - Q'_\Sigma(N, M, B)\right)\Delta POT_1 + Q'_\Sigma(N, M, B)\Delta POT \\ &\leq Q_\Sigma(N, M, B)\Delta POT_1 + Q'_\Sigma(N, M, B)\Delta POT. \end{aligned}$$

From Eq. (6) and Inequality (14), we derive

$$Q'_\Sigma(N, M, B)\left(2B\log\left(\frac{2eM}{B}\right)\right) \geq N\psi_\Sigma(\log B, N) - dN\psi_\Sigma(\log B, N)/(2d),$$

which implies

$$Q'_\Sigma(N, M, B) \in \Omega\left(\frac{N\psi_\Sigma(\log B, N)}{B\log(M/B)}\right).$$

Let $q$ be a miss which increases the potential by at least $\Delta POT_1$, and let $\gamma$ be the block fetched into the cache in the $q$-th miss. By Lemma 1, if at most $W$ entries are exchanged between $\gamma$ and the other blocks resident in cache with $\gamma$, the potential increases by at most $(B + W\log(2eM/W))$. If $M/B \geq b$ for a suitable constant $b > 1$ and $W < B/4d$, then $(B + W\log(2eM/W)) < \Delta POT_1$, which is a contradiction. Then $W = \Theta(B)$. Since an IC operation moves only a constant number of words between blocks, there are at least $\Omega(B)$ operations per miss. The theorem follows. $\square$

**Corollary 5.** *Each rational permutation $\Pi_N^\sigma$, defined by a bit-permutation $\sigma$, can be performed by an optimal cache-aware algorithm with work complexity*

$$W(N, M, B) \in \Theta\left(\frac{N\psi(\log B, \sigma)}{\log(M/B)} + N\right),$$

*and cache complexity*

$$Q(N, M, B) \in \Theta\left(\frac{N\psi(\log B, \sigma)}{B\log(M/B)} + \frac{N}{B}\right),$$

*in an* IC$(M, B)$, *for each value of $M$ and $B$.*

[4] Note that the lower bound given in [14] is $\Omega\left(\frac{N\max\{\psi_\Sigma(\log M, N), \psi_\Sigma(\log B, N)\}}{B\log(M/B)} + \frac{N}{B}\right)$, but it easy to see that it is asymptotically equivalent to the one given in Theorem 3.

**Proof.** An optimal algorithm for performing a rational permutation $\Pi_N^\sigma$ in a Disk Arrays model [16] with $p$ disks is given in [15]. By setting $p = 1$, this algorithm translates automatically into an EM algorithm; then, by removing all I/O operations, the algorithm becomes a cache-aware IC algorithm. Clearly, the number of block transfers performed by the optimal off-line policy of the IC model cannot be bigger than the number of disk accesses performed in the Disk Arrays model. This cache-aware algorithm is composed of $\psi(\log B, \sigma)/(B \log(M/B))$ phases, each of which requires $\Theta(N)$ work and $\Theta(N/B)$ misses. By Theorems 3 and 4, this algorithm exhibits optimal cache and work complexities in an IC$(M, B)$, for each value of $M$ and $B$. □

## 4. Cache-oblivious algorithm

In this section we propose an efficient cache-oblivious algorithm which performs each rational permutation $\Pi_N^\sigma$ on a vector $V$ of $N = 2^n$ entries, where $\sigma$ is an $n$-permutation. This cache-oblivious algorithm exhibits optimal cache and work complexities under the tall cache assumption. Moreover, we describe an efficient cache-oblivious algorithm for computing all the values $\Pi_N^\sigma(i)$, with $0 \leq i < N$: indeed, the computation of a generic bit-permutation $\sigma$ cannot be considered an elementary operation.

### *4.1. Computing the values of a rational permutation*

Let $N = 2^n$, and let $\Pi_N^\sigma$ be the rational permutation defined by an $n$-permutation $\sigma$. In this subsection we describe an algorithm which computes a vector $P$ of $N$ entries, where $P[i] = \Pi_N^\sigma(i)$ for each $i$, with $0 \leq i < N$. The algorithm derives $\Pi_N^\sigma(i)$ from $\Pi_N^\sigma(i-1)$ (note that $\Pi_N^\sigma(0) = 0$ for each $\sigma$), comparing the binary representations of $i$ and $i-1$.

Specifically, the algorithm uses four vectors:

- $S$ where $S[j] = \sigma(j)$ for each $j$, with $0 \leq j < n$;
- $I$ where $I[j] = \sigma^{-1}(j)$ for each $j$, with $0 \leq j < n$;
- $P$ where, at the end of the algorithm, $P[i] = \Pi_N^\sigma(i)$ for each $i$, with $0 \leq i < N$;
- $A$ where the $j$-th entry stores the $j$-th bit of the binary representation of the current index $i$, with $0 \leq j < n$ and $0 \leq i < N$ ($A[0]$ is the LSB).

More succinct data structures can be adopted, but we prefer the naïve ones for the sake of simplicity. The input of the algorithm is $S$ (i.e. the bit-permutation $\sigma$), while the output is $P$. Note that $I$ can be computed from $S$ by means of any sorting algorithm. The algorithm for computing $P$ is divided into $N-1$ stages: in the $i$-th stage, with $0 < i < N$, the algorithm adds 1 (modulo $N$) to the binary representation of $i-1$ stored in $A$, and derives $P[i]$ from $P[i-1]$ according to the differences between the binary representations of $i$ and $i-1$. The algorithm's pseudocode gives a more formal, and simpler, description:

**Algorithm 1** Computes the values of a bit-permutation $\sigma$; let $N = 2^n$.

```
INPUT: a vector $S$ of $n$ entries which represents the $n$-permutation $\sigma$;
OUTPUT: a vector $P$ of $N$ entries, where $P[i] = \Pi_N^\sigma(i)$ for each $i$, with $0 \leq i < N$;
1:  Compute $I$ from $S$ through Mergesort;
2:  Set all entries of $A$ to 0;
3:  $P[0] \leftarrow 0$;
4:  for $i = 1$ to $N-1$ do
5:      $P[i] \leftarrow P[i-1]$;
6:      $j \leftarrow 0$;
7:      while $A[j] = 1$ do
8:          $A[j] \leftarrow 0$; // The j-th bit of i is set to 0
9:          $P[i] \leftarrow P[i] - 2^{I[j]}$; // The I[j]-th bit of P[i] is set to 0
10:         $j \leftarrow j+1$;
11:     end while;
12:     $A[j] \leftarrow 1$; // The j-th bit of i is set to 1
13:     $P[i] \leftarrow P[i] + 2^{I[j]}$; // The I[j]-th bit of P[i] is set to 1
14: end for;
```

Note that Algorithm 1 is cache-oblivious and it is based on the binary counter [18].

**Theorem 6.** *The work and cache complexities of Algorithm 1 are:*

$$W(N, M, B) \in \Theta(N), \tag{15}$$

$$Q(N, M, B) \in \Theta\left(\frac{N}{B}\right) \tag{16}$$

*in an IC$(M, B)$, for each value of $M$ and $B$ such that $M/B \geq 4$.*

**Proof.** The inverse of $\sigma$, that is $I$, can be efficiently computed through Mergesort with work complexity $o(N)$ and cache complexity $o(N/B)$ [7].

In order to bound the cache complexity of the **for** loop (Steps 4–14), we describe a particular replacement policy for the cache and compute the cache complexity using this policy; since the IC model adopts an optimal off-line replacement policy, the actual cache complexity cannot be greater. We suppose the cache to have at least four lines, and we associate the vectors $I$, $P$ and $A$ with three distinct cache lines: that is, there is exactly one cache line for all the constituent blocks of each vector. The fourth line is used for support variables. Since the entries of $P$ are required in sequential order, each constituent block of $P$ is fetched only once into the line associated with $P$. Therefore, the number of misses due to $P$ is $\Theta(N/B)$.

Let $\alpha_i$ be the memory block which contains the entries $A[iB], \dots, A[(i+1)B-1]$, with $0 \le i < \lceil n/B \rceil$. When an entry $A[iB+k]$, with $0 \le i < \lceil n/B \rceil$ and $1 \le k < B$, is required, the corresponding block $\alpha_i$ is in cache since the previous required $A$'s entry was $A[iB+k-1]$, which also belongs to $\alpha_i$. On the other hand, when $A[iB]$ is referenced, block $\alpha_i$ is not in cache and a miss occurs. Since $A[j]$ flips $N/2^j$ times, with $0 \le j < n$, during the course of the algorithm [18], each block $\alpha_i$ is fetched into the cache $N/2^{iB}$ times. Therefore, the number of misses due to $A$ is $\Theta(N/2^B)$. Since $I[i]$ is read only after $A[i]$ for each $i$, with $0 \le i < n$, an upper bound for $A$ translates into a bound for $I$; then, the number of misses due to $I$ is $\Theta(N/2^B)$. Eq. (16) follows. Since there are $\Theta(B)$ operations for each block, Eq. (15) also follows. □

### 4.2. *Cache-oblivious algorithm*

In this subsection we present a cache-oblivious algorithm which performs any rational permutation $\Pi_N^\sigma$ on a vector $V$ of $N = 2^n$ entries, where $\sigma$ and $V$ are given as input. As usual, $U$ denotes the output vector. Before describing the algorithm, note that the recursive cache-oblivious algorithm for matrix transposition described in [4] moves each entry of the input matrix to the corresponding entry of the output matrix in an order based on the Z-Morton layout [19]. This particular *pattern of access* to $V$ minimizes the cache complexity of the algorithm under the tall cache assumption. In the same fashion, our algorithm first derives an efficient pattern of access to $V$ from $\sigma$, and then it moves each $V$'s entry, in an order given by the pattern, into the right $U$'s entry.

The pattern of access to $V$ is defined by the $n$-permutation $\tau_\sigma$ given by the following algorithm.[5]

**Algorithm 2** Computes the bit-permutation $\tau_\sigma$

```
INPUT: an $n$-permutation $\sigma$;
OUTPUT: the $n$-permutation $\tau_\sigma$;
1: Compute $\sigma^{-1}$ from $\sigma$ through Mergesort;
2: $i = 0; j = 0;$
3: while $j < n$ do
4:     if $\sigma^{-1}(i) \ge i$ then $\{\tau_\sigma^{-1}(j) = i; j = j + 1;\}$
5:     if $\sigma(i) > i$ then $\{\tau_\sigma^{-1}(j) = \sigma(i); j = j + 1;\}$
6:     $i = i + 1;$
7: end while
8: Compute $\tau_\sigma$ from $\tau_\sigma^{-1}$ through Mergesort;
```

The algorithm for performing $\Pi_N^\sigma$ on $V$ is divided into $N$ steps: in the $i$-th step, the entry $V[\Pi_N^{\tau_\sigma}(i)]$ is moved into $U[\Pi_N^{(\sigma^{-1})}(\Pi_N^{\tau_\sigma}(i))]$, with $0 \le i < N$. The pseudocode of the algorithm is the following:

**Algorithm 3** Performs the rational permutation $\Pi_N^\sigma$

```
INPUT: an $n$-permutation $\sigma$, and a vector $V$ of $N = 2^n$ entries;
OUTPUT: a vector $U$ of $N$ entries, where $U[\Pi_N^\sigma(i)] = V[i]$ for each $i$, with $0 \le i < N$;
1: Compute $\sigma^{-1}$ from $\sigma$ through Mergesort;
2: Compute $\tau_\sigma$ through Algorithm 2;
3: Compute the values of the bit-permutations $\Pi_N^{\tau_\sigma}$ and $\Pi_N^{(\sigma^{-1})}$ through Algorithm 1;
4: for $i = 0$ to $N - 1$ do
5:     $U[\Pi_N^{(\sigma^{-1})}(\Pi_N^{\tau_\sigma}(i))] = V[\Pi_N^{\tau_\sigma}(i)];$
6: end for
```

In order to prove the correctness and to evaluate the cache and work complexities of Algorithm 3, we introduce the following two lemmas.

**Lemma 7.** *Let $\sigma$ be an $n$-permutation. Then the function $\tau_\sigma$ defined by Algorithm 2 is an $n$-permutation.*

**Proof.** We claim that $\tau_\sigma^{-1}$ (hence $\tau_\sigma$) is a permutation. Suppose, for the sake of contradiction, that there are two values $j'$ and $j''$, $0 \le j' < j'' < n$ such that $\tau_\sigma^{-1}(j') = \tau_\sigma^{-1}(j'') = p$. Clearly, $p$ cannot be assigned to both $\tau_\sigma^{-1}(j')$ and $\tau_\sigma^{-1}(j'')$ by two steps of

[5] For simplifying Algorithms 2 and 3, we use the functions $\tau_\sigma$, $\tau_\sigma^{-1}$, $\sigma$, $\sigma^{-1}$ instead of their vector representations.

the same kind. Then, suppose that $p$ is assigned to $\tau_\sigma^{-1}(j')$ in Step 4 and to $\tau_\sigma^{-1}(j'')$ in Step 5: by the **if** statements in Steps 4 and 5, there exists a value $q \geq p$ such that $\sigma(q) = p$ and $\sigma(q) > q$, but this is a contradiction. In the same fashion, it can be proved that $p$ cannot be assigned to $\tau_\sigma^{-1}(j')$ in Step 5 and to $\tau_\sigma^{-1}(j'')$ in Step 4. Therefore, $\tau_\sigma^{-1}$ is a permutation since there are $n$ values and no duplicates. □

**Lemma 8.** *Let $\sigma$ be an $n$-permutation, and let $\tau_\sigma$ be the bit-permutation defined by Algorithm 2. Consider the set $\rho_i = \{\tau_\sigma^{-1}(k) : 0 \leq k \leq i + \psi(i+1, \sigma)\}$ for each $i$, with $0 \leq i < n-1$, then*

$$\rho_i = \{0, \ldots, i\} \cup \Upsilon(i+1, \sigma).$$

**Proof.** In order to prove the lemma, we show by induction on $i$, with $0 \leq i < n-1$, that at the end of the $i$-th iteration of Algorithm 2, we have $j = i + \psi(i+1, \sigma) + 1$ and $\{\tau_\sigma^{-1}(k) : 0 \leq k \leq i + \psi(i+1, \sigma)\}$, which defines $\rho_i$, is equal to $\{0, \ldots, i\} \cup \Upsilon(i+1, \sigma)$. If $i = 0$ the claim is clearly true. Let $i > 0$. Denote with $\tilde{j}$ the value of $j$ at the beginning of the $i$-th iteration, that is $\tilde{j} = (i-1) + \psi(i, \sigma) + 1$ by the inductive hypothesis. If $i$ is assigned to $\tau_\sigma^{-1}(\tilde{j})$ in Step 4, then $i \notin \rho_{i-1}$; otherwise $i \in \rho_{i-1}$ and, in particular, $i \in \Upsilon(i, \sigma)$. If $\sigma(i)$ is assigned to $\tau_\sigma^{-1}(\tilde{j})$ or $\tau_\sigma^{-1}(\tilde{j}+1)$ in Step 5, then either $\sigma(i) \in \Upsilon(i+1, \sigma) - \Upsilon(i, \sigma)$, or $\sigma(i) \in \Upsilon(i, \sigma)$. A simple case analysis shows that at the end of the $i$-th iteration $j = i + \psi(i+1, \sigma) + 1$ and $\rho_i = \{0, \ldots, i\} \cup \Upsilon(i+1, \sigma)$ □

As an example, suppose $\sigma = \sigma_n^T$, where $\sigma_n^T$ is the bit-permutation associated with the transposition of a $2^{n/2} \times 2^{n/2}$ matrix (Eq. (1)). Then $\tau_\sigma^{-1}$ is so defined:

$$\tau_\sigma^{-1}(i) = \begin{cases} \frac{i}{2} & \text{if } i \text{ even and } 0 \leq i < n \\ \frac{n}{2} + \frac{i-1}{2} & \text{if } i \text{ odd and } 0 \leq i < n. \end{cases}$$

According with Lemma 8, it is easy to see that by Eq. (2)

$$\rho_i = \begin{cases} \{0, \ldots, i\} \cup \{\frac{n}{2} \ldots \frac{n}{2} + i\} & \text{if } 0 \leq i < \frac{n}{2} \\ \{0, \ldots, n-1\} & \text{if } \frac{n}{2} \leq i < n-1. \end{cases}$$

**Theorem 9.** *Let $\sigma$ be an $n$-permutation, and let $N = 2^n$. Then, the cache-oblivious Algorithm 3 performs the rational permutation $\Pi_N^\sigma$ and requires*

$$W(N, M, B) \in \Theta(N) \tag{17}$$

*work and*

$$Q(N, M, B) \in \begin{cases} O\left(\frac{N}{B}\right) & \text{if } \frac{M}{B} \geq 2^{1+\psi(\log B, \sigma)} \\ O\left(\frac{NB}{M}\right) & \text{if } \frac{M}{B} < 2^{1+\psi(\log B, \sigma)} \end{cases} \tag{18}$$

*misses in an* IC$(M, B)$*, for each value of $M$ and $B$ such that $M/B > 4$.*

**Proof.** The correctness of Algorithm 3 follows from the fact that $\tau_\sigma$ and $\Pi_N^{\tau_\sigma}$ are permutations.

We now analyze the work and cache complexities of Algorithm 3. Recall that $\psi(\log B, \sigma)$ is the cardinality of $\Psi(j, \sigma)$ (or $\Upsilon(j, \sigma)$ equivalently). For simplifying the notation, we denote $\psi(\log B, \sigma)$ with $\psi$. As argued in the proof of Theorem 6, the computation of the inverse of an $n$-permutation requires $o(N)$ work and $o(N/B)$ misses. Hence, the computation of $\sigma^{-1}$ and Algorithm 2 (Steps 1–2) can be performed in $o(N)$ operations and $o(N/B)$ misses. The computation of the values of $\Pi_N^{\tau_\sigma}$ and $\Pi_N^{(\sigma^{-1})}$ (Step 3) requires linear work and $\Theta(N/B)$ misses (Theorem 6). Note that the values $\Pi_N^{(\sigma^{-1})}(\Pi_N^{\tau_\sigma}(i))$ can be computed by an adaptation of Algorithm 1, without affecting the complexities: as $\Pi_N^{\tau_\sigma}(i)$ is derived from $\Pi_N^{\tau_\sigma}(i-1)$, $\Pi_N^{(\sigma^{-1})}(\Pi_N^{\tau_\sigma}(i))$ is obtained by $\Pi_N^{(\sigma^{-1})}(\Pi_N^{\tau_\sigma}(i-1))$ comparing the binary representations of $i$ and $i-1$.

We now upper bound the cache complexity of Steps 4–6, in which all the entries of $V$ are permuted into $U$. Suppose $\frac{M}{B} \geq 2^{1+\psi}$ and partition the sequence of $N$ accesses to $V$ into $N/(B2^\psi)$ segments. Let $\mathcal{F}^i = \{\Pi_N^{\tau_\sigma}(iB2^\psi), \ldots, \Pi_N^{\tau_\sigma}((i+1)B2^\psi - 1)\}$, with $0 \leq i < N/B2^\psi$, be the set of the indices of the entries accessed in the $i$-th segment. By Lemma 8, the binary representations of the values in $\mathcal{F}^i$ differ on $(\log B + \psi)$ bit positions, and $\psi$ of these are the $(\log B)$-incoming bit positions of $\sigma$, which are among the $\log(N/B)$ MSBs by definition. Then, the $B2^\psi$ entries of $V$ with indices in $\mathcal{F}^i$ are distributed among $2^\psi$ blocks. Moreover, in the $(\log B + \psi)$ bit positions there are also $\psi$ $(\log B)$-outgoing bit positions of $\sigma$; then, by the definition of outgoing bit position, the $B2^\psi$ entries are permuted into $2^\psi$ blocks of the output vector $U$. Since there are at least $2^{1+\psi}$ cache lines, the permutation of entries indexed by the values in $\mathcal{F}_i$ requires $\Theta\left(2^\psi\right)$ misses, and the permutation of the whole vector $V$ requires $\Theta(N/B)$ misses.

Let $\frac{M}{B} < 2^{1+\psi}$, and let $\varphi$ be the maximum integer in $[0, \log B[$ such that $|\Psi(\log B, \sigma) \cap \Psi(\varphi, \sigma)| = \log(M/2B)$, that is $\varphi$ denotes the bigger bit position such that exactly $\log(M/2B)$ $(\log B)$-incoming bit positions are permuted into positions smaller than $\varphi$. Note that $\varphi$ is well defined since $|\Psi(\log B, \sigma)| = \psi > \log(M/(2B))$. We use the previous argument, except for the segment length. Specifically, partition the sequence of $N$ accesses to $V$ into $N/(2^\varphi M/(2B))$ segments and let $\mathcal{F}^i = \{\Pi_N^{\tau_\sigma}(i2^\varphi M/(2B)), \ldots, \Pi_N^{\tau_\sigma}((i+1)2^\varphi M/(2B) - 1\}$, with $0 \leq i < N/(2^\varphi M/(2B))$, be the set of the indices of the entries required in the $i$-th segment. The binary representations of the values in $\mathcal{F}^i$ differ on $\varphi + \log(M/(2B))$ bit positions, and $(\log(M/2B))$ of these are $(\log B)$-incoming bit positions of $\sigma$. Then the $2^\varphi M/(2B)$ entries of $V$ with indices in $\mathcal{F}^i$ are distributed among $M/(2B)$ blocks. An argument similar to the one used above proves that these $2^\varphi M/(2B)$ entries are permuted into at most $M/(2B)$ blocks of the output vector $U$. Therefore, the permutation steps requires $O(N/2^{\varphi}) = O(NB/M)$ misses, since $\varphi \geq \log(M/(2B))$, and Eq. (18) follows. The proof of Eq. (17) is straightforward. □

By Theorem 9 and the lower bounds on the work and cache complexities given in Section 3, the cache-oblivious Algorithm 3 is optimal when $M/B \geq 2^{1+\psi(\log B,\sigma)}$. Since $\psi(\log B, \sigma) \leq \log B$, the tall cache assumption (i.e. $M \geq B^2$) is sufficient to guarantee cache and work optimality of the cache-oblivious algorithm for each rational permutation. Remember that by Corollary 5, there exists a cache-aware algorithm for performing rational permutations which exhibits optimal cache and work complexities for all values of the IC parameters. In the next section, we will show that a similar cache-oblivious algorithm cannot exist.

## 5. Limits of cache-oblivious rational permutations

Theorem 4 proves that the work complexity of a cache-optimal algorithm is $\omega(N)$ when $M/B \in o(2^{\psi_\Sigma(\log B,N)})$, and $\Theta(N)$ otherwise. Clearly, the work complexity of a cache-oblivious algorithm is independent of the cache parameters (this is not the case, in general, for cache complexity); hence, a cache-oblivious algorithm cannot have optimal work complexity for each value of $M$ and $B$. One can wonder whether there exists a cache-oblivious algorithm which is cache-optimal for each $M$ and $B$, regardless of the work complexity. In this section we will prove that such an algorithm cannot exist. To this purpose we follow a similar approach to the one employed in [7].

Let $\Sigma$ be an infinite set of permutations which contains at most one $n$-permutation for each $n \in \mathbb{N}$, let $N = 2^n$, and let $\mathcal{A}$ be a cache-oblivious algorithm which performs any rational $N$-permutation defined by an $n$-permutation in $\Sigma$ on a vector of $N$ entries. Consider the two sequences of misses generated by the executions of $\mathcal{A}$ in two different ICs, where one model satisfies a particular assumption we will define, while the other does not. We simulate these two executions in the EM model and obtain a new EM algorithm solving the same problem of $\mathcal{A}$. By adapting the argument described in Section 3 to bound from below the number of disk accesses, we conclude that $\mathcal{A}$ cannot be optimal in both ICs.

### 5.1. The simulation technique

In this subsection we describe a technique for obtaining an EM algorithm from two executions of a cache-oblivious algorithm in two different IC models. The technique is presented in a general form and is a formalization of the *ad hoc* one employed in [7] for proving the impossibility result for general permutations.

Consider two models $\mathcal{C}_1$=IC$(M, B_1)$ and $\mathcal{C}_2$ = IC$(M, B_2)$, where $B_1 < B_2$. For convenience, we assume $B_2$ to be a multiple of $B_1$. Let $\mathcal{A}$ be a cache-oblivious algorithm for an *arbitrary* problem and let $Q_1$ and $Q_2$ be its cache complexities in the two models, respectively. We define an algorithm $\mathcal{A}'$ for EM$(2M, B_2)$ which emulates in parallel the executions of $\mathcal{A}$ in both $\mathcal{C}_1$ and $\mathcal{C}_2$ and solves the same problem of $\mathcal{A}$.

Let us regard the RAM in 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 coincide (except for some slight reordering) with the I/Os performed by $\mathcal{A}$ in $\mathcal{C}_2$, and occur exclusively between the disk and $\mathcal{M}_2$. On the other hand, $\mathcal{A}'$ executes all operations prescribed by $\mathcal{A}$ on data in $\mathcal{M}_1$.[6] Since there are no I/Os between $\mathcal{M}_1$ and the disk, data are inserted into $\mathcal{M}_1$ by means of transfers of $B_1$-rows between $\mathcal{M}_1$ and $\mathcal{M}_2$, which coincide with the I/Os performed by $\mathcal{A}$ in $\mathcal{C}_1$.

Let us now see in detail how the execution of $\mathcal{A}'$ in the EM$(2M, B_2)$ develops. Initially all the words in $\mathcal{M}_1$ and $\mathcal{M}_2$ are empty, that is filled with NIL values, and the EM disk contains the same data of the memory of $\mathcal{C}_2$ (or $\mathcal{C}_1$ indistinguishably) with the same layout (a one-to-one relation between the $B_2$-blocks of $\mathcal{C}_2$ and the $B_2$-blocks of the disk can be simply realized). Let $o_i$ be the $i$-th operation of $\mathcal{A}$, $i = 1 \dots h$. The execution of $\mathcal{A}$ in $C_i$, $1 \leq i \leq 2$, 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 $\mathcal{L}=\Gamma_1^2\Gamma_1^1 o_1 \dots \Gamma_j^2\Gamma_j^1 o_j \dots \Gamma_h^2\Gamma_h^1 o_h \Gamma_{h+1}^2\Gamma_{h+1}^1$. Each $\Gamma_j^i$, with $1 \leq j \leq h+1$ and $1 \leq i \leq 2$, is defined as follows:

- $\Gamma_1^i$ is the sequence of I/Os that precede $o_1$ in $\mathcal{L}_i$.
- $\Gamma_j^i$, $1 < j \leq h$, is the sequence of I/Os which are enclosed between $o_{j-1}$ and $o_j$ in $\mathcal{L}_i$.
- $\Gamma_{h+1}^i$ is the sequence of I/Os performed after $o_h$ in $\mathcal{L}_i$.

Note that a $\Gamma_j^i$ 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 $\Gamma_j^i$, with $1 \leq j \leq h+1$ and $1 \leq i \leq 2$. Let $\mathcal{A}'$ be divided into $|\mathcal{L}|$ phases. The behavior of the $j$-th phase is determined by the $j$-th entry $l_j$ of $\mathcal{L}$:

(1) $l_j$ *is an operation:* $\mathcal{A}'$ executes the same operation in $\mathcal{M}_1$.

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

(2) *$l_j$ 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 into the $B_2$-row of $\mathcal{M}_2$ associated with the line used in $\mathcal{C}_2$.
(3) *$l_j$ is an input of a $B_1$-block (i.e. an input of $\mathcal{L}_1$):* let $\gamma$ be such a $B_1$-block and $\gamma'$ be the $B_2$-block containing $\gamma$. Since there is no prefetch in the IC model, the next operation of $\mathcal{A}$ requires an entry in $\gamma$; thus $\gamma'$ must be in the cache of $\mathcal{C}_2$, too. For this reason, we can assume that $\gamma'$ was, or has just been, fetched into a $B_2$-row of $\mathcal{M}_2$. $\mathcal{A}'$ copies $\gamma$ in the right $B_1$-row of $\mathcal{M}_1$ and replaces the copy of $\gamma$ in $\mathcal{M}_2$ with $B_1$ NIL values.
(4) *$l_j$ 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.
(5) *$l_j$ is an output of a $B_1$-block (i.e. an output of $\mathcal{L}_1$):* let $\gamma$ be such a $B_1$-block and $\gamma'$ be the $B_2$-block containing $\gamma$. If $\gamma'$ is still in $\mathcal{M}_2$, then $\mathcal{A}'$ copies $\gamma$ from $\mathcal{M}_1$ into $\gamma'$ and replaces $\gamma$'s row with $B_1$ NIL values. The second possibility (i.e. $\gamma'$ is not in $\mathcal{M}_2$) can be avoided since no operations are executed between the evictions of $\gamma'$ and $\gamma$. If some operations were executed, both blocks $\gamma$ and $\gamma'$ would be kept in cache (and so in $\mathcal{M}_1$ and $\mathcal{M}_2$). Therefore, we can suppose $\gamma$ was removed just prior to the eviction of $\gamma'$.

It is easy to see that every operation of $\mathcal{A}$ can be executed by $\mathcal{A}'$ in $\mathcal{M}_1$, since there is a one to one relation between the cache lines of $C_1$ and the rows of $\mathcal{M}_1$ (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 at most $2Q_2$ (recall that a miss in the IC model is equivalent to at most two I/Os in the EM model).

Let $K = Q_1B_1/Q_2$; it is easy to see that $K \leq B_2$. Indeed, if $K$ were greater than $B_2$, a replacement policy for $\mathcal{C}_1$ which requires $Q_2B_2/B_1 < Q_1$ misses would be built from the execution of $\mathcal{A}$ in $\mathcal{C}_2$; but this is a contradiction since the replacement policy of the IC model is optimal. $\mathcal{A}'$ can be adjusted 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 the memory: it is sufficient to insert some dummy I/Os. This increases the I/O complexity of $\mathcal{A}'$ from $2Q_2$ to at most $2Q_2 + 2Q_1B_1/K = 4Q_2$ I/Os. In particular, there are at most $2Q_2$ inputs and $2Q_2$ outputs.

We define the *working set* $\mathcal{W}(q)$ after $q$ 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 $B_2$-blocks are evicted. When $\mathcal{A}'$ fetches a $B_2$-block from the disk, we can suppose that the at most $K$ entries which will be moved between $\mathcal{M}_1$ and the block are immediately included in the working set.

## 5.2. *Impossibility result for rational permutations*

In this subsection we prove that an optimal cache-oblivious algorithm which performs the rational permutations defined by $\Sigma$ cannot exist for each value of the cache parameters.

**Theorem 10.** *Let $\Sigma$ be an infinite set of permutations which contains at most one $n$-permutation for each $n \in \mathbb{N}$, and let $N = 2^n$. Consider a cache-oblivious algorithm $\mathcal{A}$ which performs any rational $N$-permutation defined by an $n$-permutation in $\Sigma$. If $\psi_\Sigma(i, N)$ is not decreasing in $N$ for each fixed $i$, and it is non decreasing in $i$ for each fixed $N$ and for each $i < (\log N)/2$, then $\mathcal{A}$ cannot be cache-optimal for each value of the $M$ and $B$ parameters.*

**Proof.** We begin by asserting that a lower bound on the cache complexity in the IC model translates into a lower bound on the I/O complexity in the EM model, and *vice versa*, since the IC model adopts an optimal off-line replacement policy [20]. Moreover, the lower bound provided in Theorem 3 is tight since it can be matched by the upper bound given in Corollary 5. Assume, for the sake of contradiction, that $\mathcal{A}$ attains optimal cache complexity for each value of $M$ and $B$. In particular, consider two models $\mathcal{C}_1$=IC$(M, B_1)$ and $\mathcal{C}_2$ = IC$(M, B_2)$ where $B_2$ is a multiple of $B_1$, and let $Q_1$ and $Q_2$ be the cache complexities of $\mathcal{A}$ in the two models, respectively. We will show that $B_1$ and $B_2$ can be suitably chosen so that $Q_1$ and $Q_2$ cannot be both optimal, thus reaching a contradiction. To achieve this goal, we apply the simulation technique described in the previous subsection to $\mathcal{A}$, and obtain an algorithm $\mathcal{A}'$ for the EM$(2M, B_2)$ solving the same problem of $\mathcal{A}$. We then apply an adaptation of Lemma 1 (which is based on a technical result given in [2] for bounding from below the number of disk accesses of matrix transposition in the EM model) to $\mathcal{A}'$, and we prove the impossibility of the simultaneous optimality of $\mathcal{A}$ in the two IC models. We denote with $Q$ and $Q_I$ the I/O complexity and the number of inputs, respectively, of $\mathcal{A}'$; remember that $Q \leq 4Q_2$ and $Q_I \leq 2Q_2$.

Let the $i$-th target group, $1 \leq i \leq N/B_2$, be the set of entries that will ultimately be in the $i$-th $B_2$-block of the output vector (remember that it must be entirely in the disk at the end of $\mathcal{A}'$). Let $\gamma$ be a $B_2$-block of the disk or a $B_2$-row of $\mathcal{M}_2$; the *togetherness rating of $\gamma$ after $q$ I/Os* is defined as:

$$C_\gamma(q) = \sum_{i=1}^{N/B_2} f(x_{\gamma,i}),$$

where $x_{\gamma,i}$ denotes the number of entries in $\gamma$ belonging to the $i$-th target group just before the $(q+1)$-st I/O. These entries are not included in the working set $\mathcal{W}(q)$ and are not NIL symbol. We also define the *togetherness rating for the working set* $\mathcal{W}(q)$ as:

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

where $s_i$ is the number of entries in the working set $\mathcal{W}(q)$ which belong to the $i$-th target group just before the $(q+1)$-st I/O. The *potential function* of $\mathcal{A}'$ after $q$ I/Os is defined as:

$$POT(q) = C_{\mathcal{W}}(q) + \sum_{\gamma \in disk} C_\gamma(q) + \sum_{\gamma \in \mathcal{M}_2} C_\gamma(q).$$

At the beginning and at the end of the algorithm the above definition is equivalent to the one given in Section 3; then, by Eq. (6), $POT(0) = N \log(B_2/2^{\psi_\Sigma(\log B_2, N)})$, and $POT(Q) = N \log B_2$. Hence, $POT(Q) - POT(0) = N\psi_\Sigma(\log B_2, N)$.

We now bound the increase in the potential function due to an input; the eviction of a block from the memory does not affect the potential. Suppose that the $q$-th I/O is an input and a $B_2$-block $\gamma$ is fetched into a $B_2$-row of $\mathcal{M}_2$. Before the $q$-th input, the intersection between $\gamma$ and the working set $\mathcal{W}(q-1)$ was empty; after the input, at most $K = Q_1 B_1 / Q_2$ entries of $\gamma$ are inserted into $\mathcal{W}(q-1)$. We use the following notation:

- $s_i$: number of entries in the working set $\mathcal{W}(q-1)$ belonging to the $i$-th target group;
- $k_i$: number of entries in $\gamma$ belonging to the $i$-th target group just before the $q$-th miss;
- $w_i$: number of entries in the (at most) $K$ words, inserted in $\mathcal{W}(q-1)$, belonging to the $i$-th target group.

The $s_i$, $k_i$ and $w_i$ values are limited by the following constraints:

$$\sum_{i=1}^{N/B_2} s_i \le 2M - K \qquad \sum_{i=1}^{N/B_2} k_i \le B_2 \qquad \sum_{i=1}^{N/B_2} w_i \le K.$$

The increase in the potential function due to the $q$-th miss ($\Delta POT(q)$) is:

$$\begin{aligned}\Delta POT(q) &= \sum_{i=1}^{N/B_2} \left[f(s_i + w_i) + f(k_i - w_i) - f(s_i) - f(k_i)\right] \\ &\le \sum_{i=1}^{N/B_2} \left[f(s_i + w_i) - f(s_i) - f(w_i)\right]. \end{aligned} \tag{19}$$

By the definition of the convex function $f$ (Eq. (5)), an upper bound on $\Delta POT(q)$ is obtained by setting $s_i = (2M-K)/(N/B_2)$ and $w_i = K/(N/B_2)$ in Inequality (19):

$$\begin{aligned}\Delta POT(q) &\le \sum_{i=1}^{N/B_2} s_i \log \frac{s_i + w_i}{s_i} + w_i \log \frac{s_i + w_i}{w_i} \\ &\le K \log e + K \log \frac{2M}{K} = K \log \frac{2eM}{K},\end{aligned}$$

since $(1+1/x)^x \le e$ if $x \ge 1$. Let $\mathcal{C}_1$ be a cache with more than $2^{\psi_\Sigma(\log B_1, N)}$ lines, while $\mathcal{C}_2$ be a cache with less than $2^{\psi_\Sigma(\log B_2, N)}$ lines. By Theorem 3, $cN/B_1 \le Q_1 \le dN/B_1$ for two suitable positive constants $c$ and $d$. Since the number of input operations is $Q_I \le 2Q_2$ (remember that an output does not increase the potential and that $K = Q_1 B_1/Q_2$),

$$POT(Q) - POT(0) \le \sum_{q=1}^{Q_I} \Delta POT(q) \le 2Q_2 K \log \frac{2eM}{K} \le 2dN \log \frac{2eMQ_2}{cN}.$$

By recalling that $POT(Q) - POT(0) = N \log 2^{\psi_\Sigma(\log B_2, N)}$,

$$N \log 2^{\psi_\Sigma(\log B_2, N)} \le 2dN \log \frac{2eMQ_2}{cN}.$$

Hence,

$$Q_2 \in \Omega\left(N \frac{2^{\frac{\psi_\Sigma(\log B_2, N)}{2d}}}{M}\right). \tag{20}$$

Since $\psi_\Sigma(i, N)$ is not decreasing in $i$ for each $i < (\log N)/2$, for $N$ and $M$ large enough, we can choose $B_2 = \epsilon M$ for a suitable constant $0 < \epsilon < 1$ such that the number $1/\epsilon$ of cache lines in $\mathcal{C}_2$ is less than $2^{\psi_\Sigma(\log B_2, N)}$. Thus,

$$Q_2 \in \Omega\left(\frac{N 2^{\frac{\psi_\Sigma(\log(\epsilon M), N)}{2d}}}{M}\right) \in \omega\left(N \frac{\psi_\Sigma(\log(\epsilon M), N)}{M}\right).$$

However, by optimality of $\mathcal{A}$ and Theorem 3, $Q_2$ must be $\Theta\left(N \frac{\psi_\Sigma(\log(\epsilon M), N)}{M}\right)$ when $B_2 = \epsilon M$, which yields a contradiction. □

The above theorem proves that any cache-oblivious algorithm which performs the rational permutations defined by $\Sigma$ cannot be cache-optimal for each value of $M$ and $B$. Matrix transposition and bit-reversal are examples of rational permutations which, by Eqs. (3) and (4), satisfy the hypothesis of Theorem 10. Thus, Theorem 10 implies that cache-oblivious algorithms for matrix transposition or the bit-reversal of a vector cannot exhibit optimal cache complexity for all values of the cache parameters. Note that Theorem 10 does not rule out the existence of an optimal cache-oblivious algorithm for some particular ranges of the cache parameters. Indeed by Theorem 9, there exists an optimal cache-oblivious algorithm under the tall cache assumption.

## 6. Conclusions

In this paper we studied various aspects concerning the execution of rational permutations in a cache-RAM hierarchy and, more generally, through the adoption of the cache-oblivious setting, in multi-level cache-hierarchies. We first proved a lower bound on the work complexity of any algorithm that executes rational permutations with optimal cache complexity. By virtue of this bound we were able to show the work optimality of the cache-aware algorithm derivable from the one in [15], which exhibits optimal cache complexity. Then, we developed a cache-oblivious algorithm for performing any rational permutation which exhibits optimal cache and work complexities under the tall cache assumption. When the rational permutation is a matrix transposition, our cache-oblivious algorithm represents an iterative version of the recursive cache-oblivious algorithm given in [4]. Finally, we proved that for certain families of rational permutations, including matrix transposition and bit-reversal, a cache-oblivious algorithm which achieves optimal cache complexity for all values of the IC parameters cannot exist. This result specializes to the case of rational permutations the result proved in [7] for general permutations, and it is achieved by means of a simulation technique which formalizes the approach used in [7].

To the best of our knowledge, the only impossibility results of the kind of those proved in this paper and in [7], were proved in [8]. An interesting avenue for further research would be to assess the limits of the cache-oblivious approach for other fundamental computational problems. Moreover, deeper investigations are required to understand why, in certain cases, the tall cache assumption is so crucial to obtain optimal cache-oblivious algorithms.

## Acknowledgments

I am extremely grateful to Andrea Pietracaprina and Geppino Pucci for the useful discussions, and to the anonymous reviewers for their interesting comments.

## References

[1] I. Foster, C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[2] A. Aggarwal, J.S. Vitter, The input/output complexity of sorting and related problems, Communications of the ACM 31 (9) (1988) 1116–1127.
[3] J.S. Vitter, External memory algorithms and data structures, ACM Computing Surveys 33 (2) (2001) 209–271.
[4] M. Frigo, C.E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: Proc. of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 285–298.
[5] E.D. Demaine, Cache-oblivious algorithms and data structures, in: Lecture Notes from the EEF Summer School on Massive Data Sets, 2002.
[6] L. Arge, G.S. Brodal, R. Fagerberg, Cache-oblivious data structures, in: D. Mehta, S. Sahni (Eds.), Handbook of Data Structures and Applications, CRC Press, 2005, p. 27 (Chapter 34).
[7] G.S. Brodal, R. Fagerberg, On the limits of cache-obliviousness, in: Proc. of the 35th ACM Symposium on Theory of Computing, 2003, pp. 307–315.
[8] G. Bilardi, E. Peserico, A characterization of temporal locality and its portability across memory hierarchies, in: F. Orejas, P.G. Spirakis, J. van Leeuwen (Eds.), Proc. of 28th International Colloquium on Automata, Languages and Programming, in: Lecture Notes In Computer Science, vol. 2076, Springer, 2001, pp. 128–139.
[9] A. Aggarwal, B. Alpern, A.K. Chandra, M. Snir, A model for hierarchical memory, in: Proc. of the 19th ACM Symposium on Theory of Computing, 1987, pp. 305–314.
[10] FIPS PUB 46-3, Data Encryption Standard (DES), National Institute for Standards and Technology, Gaithersburg, MD, USA, 1999.
[11] L. Carter, K.S. Gatlin, Towards an optimal bit-reversal permutation program, in: Proc. of the 39th IEEE Symposium on Foundations of Computer Science, 1998, pp. 544–555.
[12] A. Aggarwal, A.K. Chandra, M. Snir, Hierarchical memory with block transfer, in: Proc. of the 28th IEEE Symposium on Foundations of Computer Science, 1987, pp. 204–216.
[13] B. Alpern, L. Carter, E. Feig, T. Selker, The uniform memory hierarchy model of computation, Algorithmica 12 (2–3) (1994) 72–109.
[14] T.H. Cormen, Virtual memory for data-parallel computing, Ph.D. Thesis, Massachussetts Institute of Technology, 1992.
[15] T.H. Cormen, Fast permuting on disk arrays, Journal of Parallel and Distributed Computing 17 (1–2) (1993) 41–57.
[16] J.S. Vitter, E.A.M. Shriver, Optimal disk I/O with parallel block transfer, in: Proc. of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 159–169.
[17] L.A. Belady, A study of replacement algorithms for a virtual storage computer, IBM Systems Journal 5 (2) (1966) 78–101.
[18] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, Cambridge, MA, USA, 2001.
[19] S. Chatterjee, A.R. Lebeck, P.K. Patnala, M. Thottethodi, Recursive array layouts and fast parallel matrix multiplication, in: Proc. of the 11th ACM Symposium on Parallel Algorithms and Architectures, 1999, pp. 222–231.
[20] S. Sen, S. Chatterjee, N. Dumir, Towards a theory of cache-efficient algorithms, Journal of the ACM 49 (6) (2002) 828–858.