Matrix Transposition Algorithm Using Cache Oblivious

Submitted: 11th January 2023; accepted: 18th March 2023

Samuel Guzmán López, Adolfo Javier San Gil Santana, Jorge Alberto Cuba Alonso del Rivero, Sonia Pérez Lovelle, Humberto Díaz Pando

DOI: 10.14313/JAMRIS/4-2023/25

Abstract:
The Parallel and Distributed Computing group belonging to the Integrated Technological Research Complex (CITI) has been engaged in the creation of general-purpose components that support the processing of large volumes of information that characterize the problems involved in parallel computing.

Using the oblivious cache model, which works independently of the computer architecture, and the divide and conquer principle, an algorithm for matrix transposition is implemented to reduce the execution time of this algebraic operation. The algorithm ensures that most of the data content is loaded to the cache for fast processing, and makes the most of its stay in the cache to minimize missed reads and achieve greater speed.

The work includes conclusions and statistical tests carried out from experiments on computers with different architectures, reflecting the superiority of the algorithm that uses oblivious cache from an order of matrix determined according to the characteristics of each PC.

Keywords: Cache oblivious, Matrix transposition, Missed readings

1. Introduction

The Integrated Technological Research Complex (CITI) was created as a coordination project between the Technological University of Havana (CUJAE) and the Ministry of Interior (MININT). This entity is designed to host the most advanced technologies being worked with worldwide [1].

CITI’s mission is to develop technologies to enhance the security and internal order of the country. Its vision is to be a creative, innovative and benchmark organization in human capital management. In addition, to be a reference in the applicability of the results obtained in the development of systems, technologies and innovative integrated applications, with impact on security and internal order, for which it will base its work on the integration of highly qualified professionals with talented students [1].

At CITI there are projects in which matrix transposition is intensively used, so this task was assigned to the Parallel and Distributed Computing group, which is dedicated to reduce the execution time of various algorithms by employing parallelism and recurrence techniques. This time, the technique selected by the group was the cache oblivious, a recurrent technique about which there is some literature and implementation tested and documented by other authors [2-4]. This method was used by the authors in a research work on matrix multiplication obtaining good results [5].

2. Caching Algorithms

2.1. Cache-aware Algorithms

Cache-aware algorithms take into account the hardware architecture on which they are running, mainly the cache architecture, i.e. the number of cache levels and the size of each level. They are specifically developed to perform as well as possible in the environment for which they were created.

This poses a problem when changing the environment, since if a cache-aware algorithm is executed outside the architecture for which it was designed, it will not perform well. To counteract this problem, cache oblivious algorithms were created, able to work efficiently on any architecture [6].

2.2. Cache Oblivious Algorithms

Cache oblivious algorithms have a design that will always be “cache-optimal”, regardless of the cache hierarchy. In 1996, the idea of realizing algorithms that do not take into account the architecture of the computer where they are executed was conceived by Charles E. Leiserson and called cache oblivious algorithms. This topic was first published in 1999 byHarald Prokop in his master’s thesis at the Massachusetts Institute of Technology [4]. The use of the cache oblivious model has a wide variety of applications such as: matrix multiplication, matrix transposition, Bioinformatics (RNA secondary structure prediction), Shortest Path Algorithm with order O(n), dynamic programming of the Gaussian solution (Numerical Mathematics).

The use of the cache oblivious model aims to decrease missed reads or cache misses since these algorithms use the divide-and-conquer principle to divide the problem into small subproblems until a cache-fitting size is reached, regardless of the size of the cache.
By reducing the number of missed reads or cache misses, execution times are significantly reduced, resulting in greater efficiency.

One of the features by which it outperforms the traditional cache is self-tuning. In typical cache algorithms, the algorithms require tuning to various cache parameters that are not always available from the manufacturer and are often difficult to extract automatically which hinders code portability whereas in cache oblivious algorithms no such tuning is required, a single algorithm should work well on all machines without any modification [3, 4, 7–9].

2.3. Matrix Transposition

Matrix transposition is a fundamental operation of linear algebra and other computational primitives such as the multidimensional Fast Fourier Transform; it is also applied in numerical analysis in economics, image and graphics processing, as well as being used in cryptographic methods [10].

This seemingly innocuous permutation problem lacks both temporal and spatial locality and is therefore difficult to implement efficiently for matrices with a large volume of data. In fact, there is no temporal locality to exploit, since each element of the matrix is accessed at most once [10].

As far as spatial locality is concerned, the matrix element swaps (i, j) and (j, i) implicit in the transpose semantics, when translated into memory addresses using canonical row-major or column-major ordering, equals the memory localities ni+j and nj+i. Depending on the values of i and j, these may be close or far apart in terms of cache sets or memory pages. Careful scheduling of these swap operations is required to gain any advantage from these multiword cache lines [10].

Explicit transposition of an array into memory can often be avoided by accessing the same data in a different order. For example, software libraries for linear algebra, such as BLAS, generally provide options to specify that certain matrices should be interpreted in transposed order to avoid the need for data movement [10].

Describing the algebraic operation as such, a transposed matrix is the action of selecting rows from the original matrix and rewriting them as columns in the new matrix (see Figures 1 and 2).

In other words, the transposed matrix is the action of selecting rows from the original matrix and rewriting them as columns in the new matrix.

Examples:

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}^t = \begin{bmatrix}
a & c \\
b & d
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix}^t = \begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix}
\]

2.4. Transposition of Block Matrices

The manipulation of matrices with a large number of rows and columns involves big problems, even when they are handled with a computer. Therefore, it is often interesting to know how to decompose a problem using large matrices into smaller problems, i.e., using smaller matrices [11].

The possibility of decomposing a matrix into smaller matrices has many applications in communications, electronics, solving systems of equations, taking advantage of the vector structure of some computers, and so on. And, especially, it gives the possibility of writing a matrix in a more compact way [11].

The blocks are obtained by drawing imaginary vertical and horizontal lines between the elements of the matrix. Their dimension depends on the size of the cache blocks and aims to store as much information as possible.

3. Algorithm Implementation

3.1. Description of Operation

The fundamental idea is to reduce the transpose of a matrix to the transpose of small submatrices. This is achieved by dividing the matrices in a half along their largest dimension until only one matrix transpose that fits in the cache needs to be carried out (in theory, one could further divide the matrices down to a base case of size 1 × 1, but in practice a larger base case is used, e.g., 16 × 16, in order to amortize the overhead of calling recursive subroutines) [12].

3.2. Description of the Implementation

In section 2, all the theoretical foundations that support the implementation of a matrix transposition algorithm using the cache oblivious model were presented. Algorithm 1, adapted from the one found in https://es.stackoverflow.com, was used.

This algorithm has four integers and a pointer as parameters, of which the first and third are fundamental to divide the original matrix into small submatrices. The second and fourth refer to the number of rows and columns respectively, while the pointer refers to the result matrix.

\[
\begin{bmatrix}
0 & 0 & 4 \\
1 & 0 & 3 \\
0 & 1 & 0 \\
0 & 3 & 2 \\
0 & 2 & 3 \\
0 & 3 & 4 \\
3 & 3 & 1
\end{bmatrix}^t
\]

Figure 2. Example of transposition of a matrix of order 3x7 (taken from [11])

Figure 1. Example of transposition of a square matrix and another of order 2x3 (taken from [11])
Table 1. Computer characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
<th>PC5</th>
<th>PC6</th>
<th>PC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel(R) Core (TM) i3-5020U CPU @ 2.2GHz</td>
<td>Intel(R) Celeron (R) CPU G3900 @ 2.8GHz</td>
<td>Intel(R) Core (TM) i3-7130U CPU @ 2.7GHz</td>
<td>Intel(R) Core (TM) i3-4130 CPU @ 3.4GHz</td>
<td>Intel (R) Pentium (R) CPU G4560 @ 3.50GHz</td>
<td>Intel (R) Pentium (R) CPU G4560 @ 3.50GHz</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>4.00GB</td>
<td>2.00GB</td>
<td>8.00GB</td>
<td>8.00 GB DDR3</td>
<td>16.2GB</td>
<td>8.00GB</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>Single-Channel DDR3 @798MHz</td>
<td>Single-Channel DDR4‐2133</td>
<td>(2.95GB utilizable) DDR4‐2400</td>
<td>(7.95GB utilizable) DDR4‐2400</td>
<td>(15.8GB utilizable) DDR4‐3200</td>
<td>(7.95GB utilizable) DDR4‐2400</td>
<td></td>
</tr>
<tr>
<td>Type of system</td>
<td>Windows 64 bits</td>
<td>Windows 64 bits</td>
<td>Windows 64 bits</td>
<td>Windows 64 bits</td>
<td>Linux 64 bits</td>
<td>Windows 64 bits</td>
<td></td>
</tr>
<tr>
<td>Motherboard</td>
<td>ASUSTeK COMPUTER INCX540LA</td>
<td>Gigabyte Technology Co., Ltd. B85M‐DS3H</td>
<td>Gigabyte Technology Co., Ltd. B85M‐DS3H</td>
<td>Dell Inc. 02D7G R U3E1 Versión A00</td>
<td>Gigabyte B85M‐DS3H</td>
<td>HP Spectre 14‐EA</td>
<td></td>
</tr>
<tr>
<td>Cache L1 (instructions)</td>
<td>64KB</td>
<td>64KB</td>
<td>64KB</td>
<td>64KB</td>
<td>128KB</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>Cache L1 (data)</td>
<td>64KB</td>
<td>64K</td>
<td>64K</td>
<td>64KB</td>
<td>192KB</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>Cache L2</td>
<td>512KB</td>
<td>512KB</td>
<td>512KB</td>
<td>512KB</td>
<td>5MB</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Cache L3</td>
<td>3MB</td>
<td>2MB</td>
<td>2MB</td>
<td>3MB</td>
<td>12MB</td>
<td>3MB</td>
<td></td>
</tr>
</tbody>
</table>

void cachetranspose(int rb, int re, int cb, int ce, Matrix* T) {
    int r = re - rb, c = ce - cb;
    if (r <= 16 && c <= 16) {
        for (int i = rb; i < re; i++) {
            for (int j = cb; j < ce; j++) {
                T->data[i*rows+i] = data[i*columns+j];
            }
        }
    } else if (r > c) {
        cachetranspose(rb, rb+(r/2), cb, ce, T);
        cachetranspose(rb+(r/2), re, cb, ce, T);
    } else {
        cachetranspose(rb, re, cb, c/2, T);
        cachetranspose(rb, re, cb+(c/2), ce, T);
    }
}

Algorithm 1. Recursive matrix transposition algorithm using cache oblivious

4. Experiments

For the development of the experiments it was necessary a previous study of several algorithms (traditional, blocks and blocks_for_squared_matrices) to establish a comparison with those using the cache oblivious model (for square and non-square orders). These experiments consist of running each algorithm 5 times on orders with different characteristics (see Table 1). From the results obtained, a statistical analysis is performed to determine whether the algorithms using the cache oblivious model are superior (in terms of execution time and missed reads) to those that do not use this model.

4.1. Results

In this section we present diagrams showing the average execution time and the missed readings (the latter only in PC5), for each of the algorithms analyzed.

In those cases, where non-squared matrices were tested, these were filled with zeros in order to use the blocks_for_squared_matrices algorithm for the corresponding comparisons.
It is evident from Figure 5 that, for the computer described as PC2, the algorithm using cache oblivious is superior in terms of execution time to the traditional and block algorithms for all orders, while the blocks_for_squared_matrices algorithm has lower or similar times to the one using cache oblivious up to order 10000, from which the cache oblivious algorithm presents lower values.

Figure 6 shows that, for the computer identified as PC2, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_squared_matrices algorithms for all the orders analyzed.

Figure 7 shows that, for the computer identified as PC3, the algorithm using cache oblivious is superior in terms of execution time to the traditional and block algorithms for all orders, while the blocks_for_squared_matrices algorithm has lower execution times than the one using cache oblivious until order 12000, when they start to have similar values.

Figure 8 shows that, for the computer identified as PC2, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_square_matrices algorithms for all the orders analyzed.

Figure 9 shows that, for the computer identified as PC3, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_squared_matrices algorithms for all the orders analyzed.

Figure 10 shows that, for the computer identified as PC4, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_square_matrices algorithms for all the orders analyzed.

Figure 11 shows that, for the computer identified as PC5, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_squared_matrices algorithms for all the orders analyzed.

Figure 12 shows that, for the computer identified as PC6, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_squared_matrices algorithms for all the orders analyzed.

Figure 13 shows that, for the computer identified as PC5, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_square_matrices algorithms for all the orders analyzed.

It is evident in Figure 13 that, for the computer identified as PC6, the algorithm using cache
oblivious is superior in terms of execution time to the traditional algorithms, by blocks and blocks_for_squared_matrices, starting from the order 10000x10000.

Figure 14 shows that, for the computer identified as PC6, the algorithm using cache oblivious is superior in terms of execution time to the traditional, block and block_for_squared_matrices algorithms for all the orders analyzed.

Figure 15 shows that, for the computer identified as PC7, the blocks_for_squared_matrices algorithm is superior to the others analyzed and it can be observed that the algorithm using cache oblivious obtains a certain parity from the order 14000x14000.
missed readings. As PC5, the algorithm that uses cache oblivious has fewer lowest number of missed readings. The library used (PAPI) has not provided new updates since the XP version of Windows. Because the library used (PAPI) has not provided new updates since the XP version of Windows, it can be used on any architecture supported by the library. PAPI provides an abstraction layer that allows developers to access PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors. PAPI is used to account for missed reads. Its main purpose is to provide access to the PMCs (Performance Application Programming Interface) library, developed at the University of Tennessee, was used to account for missed readings. Its main purpose is to provide access to the PMCs (Performance Monitoring Counter) of a diverse collection of modern processors.
The test was performed with the statistical software R. After the test it was demonstrated that the matrix transposition algorithm using the cache oblivious, depending on the architecture of the computer where it was used and from a certain order, will be better than the other algorithms analyzed.

6. Conclusion
Under the computational conditions used for the experiments:
1) On a computer with a Windows operating system, in the matrix transposition operation, for square order matrices it is not feasible to employ the algorithm using the cache oblivious model for an order less than 14000 x 14000.
2) Regardless of the computer architecture, it was shown that from order 6000 x 8000 for non-square order matrices, the matrix transposition algorithm using cache oblivious is faster than the rest of the algorithms studied.
3) The blocks_for_squared_matrices algorithm has a lower performance when used for non-square matrices since these must be completed with zeros until their order is square and therefore the algorithm increases its execution time.
4) For large volumes of information, the execution time is in direct correspondence to the missed readings.
5) Algorithms that use the cache oblivious model for large volumes of information have fewer missed readings than the rest.

AUTHORS
Samuel Guzmán López – Technological University of Havana José Antonio Echeverría, Cuba, e-mail: samuguzmanlopez97@gmail.com.
Adolfo Javier San Gil Santana – Technological University of Havana José Antonio Echeverría, Cuba, e-mail: asang@ceis.cujae.edu.cu.
Jorge Alberto Cuba Alonso del Rivero – Technological University of Havana José Antonio Echeverría, Cuba, e-mail: jcuba@ceis.cujae.edu.cu.
Sonia Pérez Lovelle – Technological University of Havana José Antonio Echeverría, Cuba, e-mail: sperezl@ceis.cujae.edu.cu.
Humberto Díaz Pando – Technological University of Havana José Antonio Echeverría, Cuba, e-mail: hdiap@ceis.cujae.edu.cu.
*Corresponding author

References