IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719
Vol. 3, Issue 11 (November. 2013), ||V4|| PP 50-55

Cache Oblivious Matrix Transpositions using Sequential Processing

korde P.S., and Khanale P.B
[1]*Department of Computer Science Shri Shivaji College,Parbhani (M.S.) India*
[2]*Department of Computer Science Dnyopasak College Parbhani (M.S.) India*

Abstract: - Matrix transpositions is a fundamental operation in linear algebra and in Fast Fourier transforms and applications in numerical analysis, image processing and graphics. The optimal cache oblivious matrix transpositions makes O $(1+N^2 / B)$ cache misses. In this paper we implement divide and conquer based algorithm for matrix transposition through recursive process.

Keywords*: - Cache Memory; Cache oblivious; Cache hit; Cache miss; Tag.*

I. INTRODUCTION

The most important data structure algorithm is vector, matrices, arrays. A typical modern platform features a hierarchical cascade of memories whose capacities and access times increase as they go further from the CPU. In order to amortize the larger cost incurred when referencing data in distinct levels of the hierarchy. A block of contiguous data is replicated across the aster levels either automatically by hardware or by software. The rationale such a hierarchical organization is that the memory access cost of computation can be reduced when the same data are frequently reused with in 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.

To improve cache performance the temporal and spatial locality of the access to the linearised matrix elements has to be improved most linear algebra libraries like BLAS[?]. Therefore use techniques like loop blocking and loop unrolling [?]. A lot of fine tuning required to reach optimal cache efficiency on given hardware.

The cache oblivious model was propose in [?] and then has been used in hundreds of research papers. It is becoming popular among researchers in external memory algorithms, parallel algorithms, data structure and other related fields. The model was designed to capture the hierarchical nature of memory organization.
This paper used matrix transposition problem to evaluate the algorithms designed to be "Optimal" under memory model on real machine. The purpose this exercise is to understand how well the asymptomatic predications of the theoretical memory models match behavior observed in real memory hierarchies.

II. OTHER RELATED WORK

The memory model [1] is the original model of two level memory hierarchies. It consists of size M and data. The algorithms can transfer contiguous block of data size B to or form disk. The Textbook of data structure in this model is the B-tree, a dynamic directory that support inserts, deletes and predecessor quarries in $O(\log_B N)$ per operations.

Although a number of cache oblivious algorithms have been proposed , to date most of the analyses have been theoretical with few studies on actual machines. An exception th this is a paper by Chatterjee and Sen[2,1]. Their work is of interest in two large dimensions it was actually the worst. Second their timing runs showed that in transposition algorithms. It was suggested that the poor performance of the cache oblivious matrix transposition algorithm was related to the associatively fo the cache, although this relationship was not fully explored.

Recently a new model that combines the simplicity of the two levels model with the realism of more complicated hierarchical models was introduced by Frigo, Leiserson, Prokopand Ramchandran[8]. This model called cache oblivious model, enable us to reason about simple two level meld, but prove results about an unknown , multi level memory model. This idea is to avoid any memory-specific parameterization that is to design algorithms that do not use any information about memory access times or block sizes.

III. MATRIX TRANSPOSITION

Matrix Transposition is a linear algebra operation in Fourier transforms and it has application in numerical analysis, image processing and graphics. The simplest of matrix algorithms.

Algorithm 1 : Cache Oblivious Matrix Transposition

Matrix transposition is one of most common operation on matrices often algorithm.
The following simple transpose the matrix essentially base on the definition of transpose and it does in place

```
For  ( i=1; i<n;  i++ )
  {
    For  ( j=1; j<n;  j++ )
     {
       Swap A [ i ] [ j ] =  A[ j ] [ i ];
     }
}
```

A matrices are stored in "row-major" storage that is the right most dimension varies the fastest. In above the number of cache misses is O (N^2). The optimal cache oblivious transposition makes O ($1+N^2$ / B)cache misses.

Algorithm 2 : Cache Ignorant Matrix Transposition

Given cache oblivious techniques, it simple loop implementation inside the block. It takes input sub matrix as (x) in the input matrix I and transposed it to the output matrix.

```
For  ( i=1; i<n;  i++ )
  {
    For  ( j=1; j<n;  j++ )
     {
        Temp=A[ j ] + [ i ];
          A[ i ] [ j ] = A[ j ] [ i ];
                A[ j ] [ i ] = temp;
       }
     }
```

In this implementation the inner loop are executed in n(n-1)/2 times and no special cache is made to use the cache efficiently.

Algorithm 3 : Cache Blocked Matrix Transposition

In Block transposition algorithm the matrix is effectively divided into a small blocks. Two blocks are distributed with respect to the identified data copied into resident variables. The variables are then coped back into the matrix, but in transposed form. The source code are

```
For  ( i=1; i<n;  i+=size )
  {
    For  ( j=1; j<n;  j+=size )
     {
        Copy of A[ i ] [ j ] to  x
          Copy of A[ j ] [ij ] to  y
          Copy of  x to A[ j ] [ i ]
          Copy of  y to A[ i ] [ j ]

       }
     }
```

In this implementation small blocks is given by size with $2=size^2$ is less than cache size, the size is assumed perfectly divides the matrix dimension n. in this method each element of matrix is now loaded into registers twice.

Depending upon the programming language the algorithm 1,2,3 us used, the elements of the matrices will be stored in row major or column-major order or even using a pointer based scheme like in C/C++ or JAVA. As we know the resulting program will be show rather disappointing performance on most current computers due to the bad use of cache memory. We reformulate the above algorithm as Sequential Processing Matrix Transposition.

Algorithm 4 :Sequential Processing for Matrix Transposition Algorithms:

First we reformulate the Algorithm 1,2,3 into the following form

```
Set counter =0
  For  i =   0 to n
         For j   =    0  to   n
             b [ counter ]= a [ j ] [ i ]
                Counter= counter+1
Set counter =0
  For  i =   0 to n
         For j   =    0  to   n
             a [ i ] [ j ] =  b[ counter]
                Counter= counter+1
```

In algorithms we have used one dimension array on the execution order of the main loop. It may be executed in same order of another loop with coping values of matrix an in column-major order. In second time it back transfer the values of matrix a respectively.

Let us consider 3 by 3 Transposition of matrices. The elements of matrices are in fig 1

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix}$$

fig 1: Matrix A

The elements of matrix A are computed as fig 2

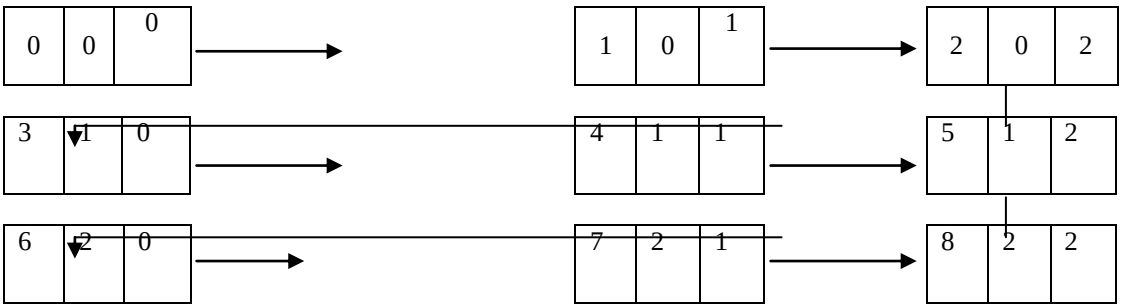

Fig 2 Graph representation of operation matrix A

The elements of matrix A process in format of fig 3

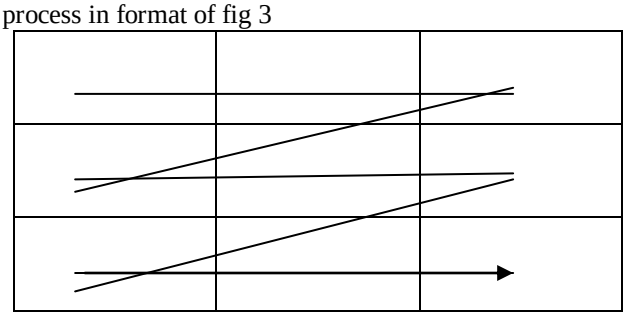

Fig 3 The process of matrix A elements

Sequential Processing Matrix Transposition

First we compute matrix A, we have perform the following two steps

1) Initialize single dimension array and transfer element of matrix A as column-major order in single dimension array.
2) The single dimension array again back copy to Matrix A.

When individual operation can be executed in arbitrary order. Our goal will be to find an optimally "localized" execution order of the operations in matrix A

$$\begin{pmatrix} a_0 & a_3 & a_6 \\ a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \end{pmatrix}$$

fig 4: Sequential Processing Matrix A

The elements of matrix A are computed as fig 2

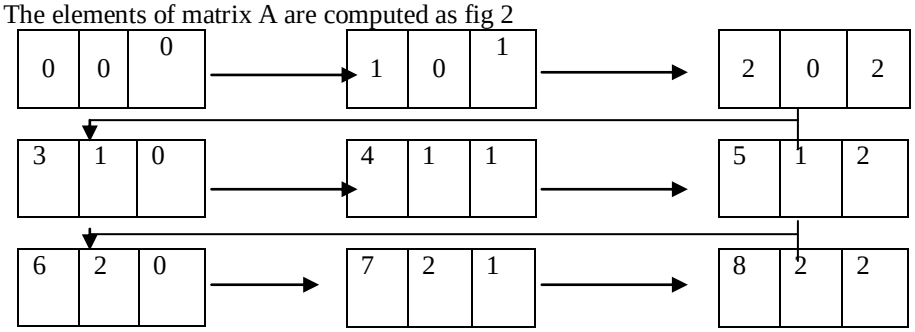

Fig 2 Graph representation of operation matrix A

Experimental Results:

We implemented Four variants of our algorithms 1) Cache Oblivious Matrix Transposition 2) Cache Ignorant Matrix Transposition 3) Cache Blocked Matrix Transposition4) Sequential Processing for Matrix Transposition Algorithms. There are several things to note down about Matrix Transposition algorithms. Cache Oblivious, Ignorant and Blocked Matrix Transposition Algorithms solve problems in different orders. Sequential Processing for Matrix Transposition Algorithm which give better performance as cache efficiency.

When we implement new method it reduces the cache miss ratio. We also measure the execution time with different array sizes.

We present here general free cache oblivious of performance, we consider here 3×3, 4×4, 5×5 matrices then find out cache miss and cache hit ratio shown in tables.

Table 1 : 3×3 Cache miss ratio

Sr.no.	Algorithms	Cache miss
01	Cache Oblivious Matrix Transposition	6
02	Cache Ignorant Matrix	6
03	Cache Blocked Matrix Transposition	6
04	Sequential Processing for Matrix Transposition Algorithms	0

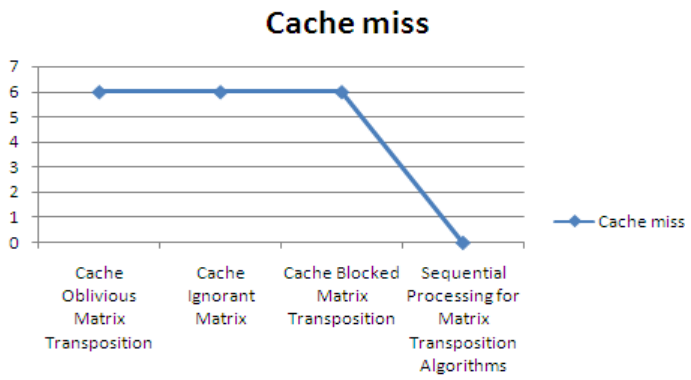

Table 2 : 4×4 Cache miss ratio 4×4

Sr.no.	Algorithms	Cache miss
01	Cache Oblivious Matrix Transposition	10
02	Cache Ignorant Matrix	10
03	Cache Blocked Matrix Transposition	10
04	Sequential Processing for Matrix Transposition Algorithms	0

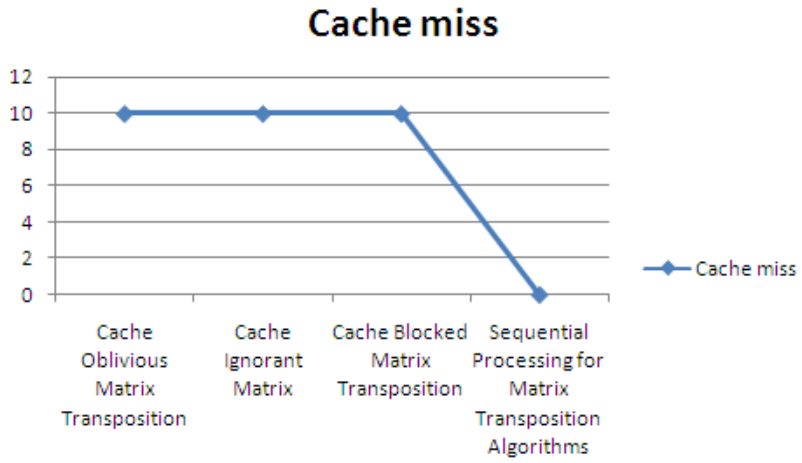

Table 3 : Cache miss ratio 5×5

Sr.no.	Algorithms	Cache miss
01	Cache Oblivious Matrix transposition	10
02	Cache Ignorant Matrix	10
03	Cache Blocked Matrix Transposition	10
04	Sequential Processing for Matrix Transposition Algorithms	0

Table 1: Analysis of Cache hit and Cache miss ratio.

Sr.no.	Matrices	Algorithms	Cache hit	Cache miss
01	3 × 3	Cache Oblivious Matrix Transposition	3	6
02	3 × 3	Cache Ignorant Matrix	3	6
03	3 × 3	Cache Blocked Matrix Transposition	3	6
04	3 × 3	Sequential Processing for Matrix Transposition Algorithms	9	0
01	4 ×4	Cache Oblivious Matrix Transposition	6	10
02	4 × 4	Cache Ignorant Matrix	6	10
03	4 ×4	Cache Blocked Matrix Transposition	6	10
04	4 ×4	Sequential Processing for Matrix Transposition Algorithms	16	0
01	5 × 5	Cache Oblivious Matrix Transposition	10	15
02	5 × 5	Cache Ignorant Matrix	10	15
03	5 × 5	Cache Blocked Matrix Transposition	10	15
04	5 × 5	Sequential Processing for Matrix Transposition Algorithms	25	0

Table 1: Analysis of Cache hit and Cache miss ratio.

Sr.no.	Algorithms	Cache miss
01	Cache Oblivious Matrix Transposition	6
02	Cache Ignorant Matrix	6
03	Cache Blocked Matrix Transposition	6
04	Sequential Processing for Matrix Transposition Algorithms	0
01	Cache Oblivious Matrix Transposition	10
02	Cache Ignorant Matrix	10
03	Cache Blocked Matrix Transposition	10
04	Sequential Processing for Matrix Transposition Algorithms	0
01	Cache Oblivious Matrix Transposition	15
02	Cache Ignorant Matrix	15
03	Cache Blocked Matrix Transposition	15
04	Sequential Processing for Matrix Transposition Algorithms	0

A Locality Preserving Cache Memory Matrix Multiplication