A Data Locality for Z-Curve Cache Oblivious Matrix Multiplication Algorithms

korde P.S.¹, Khanale P.B²

¹Department of Computer Science Shri Shivi College, Parbhani (M.S.) India
²Department of Computer Science Dnyopasak College Parbhani (M.S.) India

Abstract: The memory of modern computer is layered in a hierarchy, top to bottom[T. Axford 1992] primary cache, secondary cache, main memory, virtual memory and distributed memory with more levels to come in the future. Our goal is to automatically achieve that of fine-tuned algorithms on a multi-level memory hierarchy, this automatically is because cache oblivious algorithms have knowledge about any capacity and block size of each level of the hierarchy. We tested the famous problem of matrix multiplication using sequential access processing [Korde P.S et.al. 2011] which decreases the blocks as compared to Peano Curve Cache Oblivious Matrix Multiplication [Bader M A, et.al. 2004]. In this paper we simply change the execution flow of sequence with different blocks.

Keywords: Cache oblivious; Cache miss, Sequential process, Cache hit

1. INTRODUCTION.
Cache line is the transfer of block a cache a page of virtual memory is transfer of block of a virtual memory system into main memory. Patterson and Nennessy[D.A. Paterson et.al. 1998] use the same term block for this same concept. A memory miss is a memory access that triggers the transfer block into a level of the memory hierarchy a cache miss is memory miss in cache. The different types of memory in the hierarchy have different relative speed and sizes. The large problems of high performance computing and cost of memories in the hierarchy and efficiency the number of times that the slower memories are accessed. A memory speed fall further and further behind processor speed[W.A. Walf et.al. 1995] this demand become more and more important.
The cache oblivious model[M. Frigo et.a. 1999] assumes simplified two levels of hierarchy the cache and memory. The model has the following assumption on the cache first, the cache is tall C>=B² and is fully associative, second the cache uses a optimal replacement policy. If the cache is full, the ideal cache line will be replaced based on future accesses.
The cache complexity of an algorithms is defined to be the asymptotical number of blocks transfers between the cache and memory incurred by the algorithm. A cache oblivious has an optimal cache complexity Θ ( log b N).
Although a number of cache oblivious algorithm have been proposed to date most of the analysis have been theoretical with few studies on actual machines. Chaterjee and Sen[S. Chaterjee 2000] on “Cache Efficient Matrix Multiplication”, in this paper outline the number of matrix transposition algorithms.
To improve cache performance the temporal and spatial locally of the access to the linearized matrix elements has to be improved. Most linear algebra libraries like implementations of BLAS[C.L. Lawson et.al. 1989] therefore use techniques like loop blocking, loop unrolling [Kazushige Goto et.al.] a lot of fine tuning is required on a given hardware and very often the tuning has to popular that are based on recursive block matrix multiplication [Erik D. Demaine et. al 2002]. They automatically achieve the desired block of main loop.
In this paper, we will present an approach that used an reordering of matrix element that is based on cache oblivious sequential processing and Peano curve cache oblivious algorithms processing which results from block recursive multiplication schemes.

2. CACHE OBLIVIOUS MATRIX MULTIPLICATION
The standard algorithm that perform O(n³) operations proceeds as shown in fig-1. Performing 27 recursive matrix multiplications and nine matrix additions. Better Reuse of the local memory with in transfer block Peano curve cache oblivious matrix multiplication [Bader M A et.al.2004] uses block recursive structure and an element ordering that is based on fig-1

Fig-1 Recursive blocks of Peano curve

In this resulting code, index jump can be totally avoided which lead to asymptotically optimal spatial and temporal locality of data access as shown in fig-1.
To improve cache performance we implement the new method that is cache oblivious matrix multiplication algorithms using sequential access processing [Korde P.S. et.al. 2011] as shown in fig-2.
However in this method we change the processing of execution of sequence. In this paper execution process is vice-versa as shown in fig-3

**3 Z Curve Cache Oblivious Processing**

In this paper, we present an approach that uses a Z Curve Oblivious Recursive Storage access processing. The Recursive Storage access (see fig 3) technique that converts the two dimensions into single. It is very easy to manipulate different matrix transformation operations.

However, our presented scheme executes natural process of matrix multiplication in access to all matrices involved and reduces cache miss. It also uses only two types of recursive blocks. Now it is easy to access the matrix element levels. It gives better element access from the cache memory.

Let's consider the multiplication of 3-by-3 matrices. The elements matrices are combined as single matrices.

\[
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix}
\]

... (4)

The elements \( C_k \) of matrix \( C \) are computed as sum of products of \( a \) and \( b \).

\[
C_k = \sum_{ij} a_i b_j
\]

Where each set \( C_k \) contains multiplication of both pair elements. Figure 4 gives the graph representation of operations of 3-by-3 matrix multiplication. The nodes of graph are triples \((i, j, k)\). In figure 4, two nodes are connected if the difference between two indices of nodes is not larger than one. Observe that all the nodes are connected and there are natural processes at all. It can be traversed in forward or backward direction.

**Figure 6: Graph representation of operations of 3-by-3 matrix multiplication**

For example, the element \( C_0 = \{(a_6, b_0), (a_7, b_3), (a_8, b_6)\} \) and that of \( C_1 = \{(a_6, b_1), (a_7, b_4), (a_8, b_7)\} \). The multiplications scheme presented can be easily extended to multiplication of 5-by-5, 7-by-7 and so on. It can be used on any matrix multiplication as long as the matrix dimensions are odd numbers. It is necessary to use a block recursive approach. In case of a large matrix, the matrix can be divided into 3-by-3 recursive blocks as shown in figure 5.

**Figure 7: 3-by-3 Z Curve Cache Oblivious Blocks**

Recursive blocks for 9-by-9 matrices are shown in figure 3. Observe that we have used only two types B and D.
recursive blocks for the complete 9-by-9 matrix. This is the best possible way to divide the 9-by-9 matrix.

**Figure 8:** Z-Curve Cache Oblivious recursive blocks for 9-by-9 matrix.
Also, observe that the range of indices within a matrix block is contiguous. So, it fulfills the basic requirement of recursive block and avoids the jumping of blocks.

### 4. Z-Curve Cache Oblivious Matrix Multiplication

Now we will show the use of recursive blocks B and D in case of 9-by-9 matrix multiplication. The two 9-by-9 matrices and their resultant matrix is given by equation (9).

\[
\begin{bmatrix}
  a_0 & a_1 & a_2 \\
  a_3 & a_4 & a_5 \\
  a_6 & a_7 & a_8 \\
\end{bmatrix}
\times
\begin{bmatrix}
  b_0 & b_1 & b_2 \\
  b_3 & b_4 & b_5 \\
  b_6 & b_7 & b_8 \\
\end{bmatrix}
= 
\begin{bmatrix}
  c_0 & c_1 & c_2 \\
  c_3 & c_4 & c_5 \\
  c_6 & c_7 & c_8 \\
\end{bmatrix}
\]

\[\text{We can write the equations for elements of resultant matrix as given in equation (10).}\]

\[
B_{c_0} = A_{0} + B_{a_0} + D_{a_4} + D_{a_8} + D_{b_0} + D_{b_4} + D_{b_8}
\]

\[
D_{c_4} = A_{4} + B_{b_0} + D_{b_4} + D_{b_8}
\]

\[
D_{c_8} = A_{8} + B_{b_0} + D_{b_4} + D_{b_8}
\]

\[\text{Similarly we can write equations for } B_{c_1}, \ D_{c_2}, \ B_{c_3}, \ B_{c_5}, \ D_{c_7}, \ D_{c_9}. \text{ If we consider the ordering of the matrix blocks, there are exactly four types of block multiplications as given in equation (5).}\]

\[
\begin{align*}
B & \leftarrow B \cdot B \\
B & \leftarrow D \cdot B \\
D & \leftarrow B \cdot X \\
D & \leftarrow D \cdot D
\end{align*}
\]

\[\text{Thus we have a closed system of four multiplication schemes. Observe that we have got only four multiplication schemes which are much less than other works M. Bader and Christoph Zenger [Bader M A et.al. 2004]. We need to compute matrix operations for all four multiplication schemes. The matrix operations for } B = B + B.B \text{ can be same as that given in figure 3. The matrix operations for } B = B + B.D \text{ is given in figure 12.}\]

**Figure 12:** Matrix operations for \( B = B + B.D \)

After carefully examining all matrix operations we can prepare the table for Z Curve Cache Oblivious operations for all matrix multiplication schemes as given in table 1.

<table>
<thead>
<tr>
<th>C</th>
<th>+</th>
<th>A . B</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>+</td>
<td>B . B</td>
</tr>
<tr>
<td>B</td>
<td>+</td>
<td>D . B</td>
</tr>
<tr>
<td>D</td>
<td>+</td>
<td>D . D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>+</th>
<th>A . B</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>+</td>
<td>B . B</td>
</tr>
<tr>
<td>B</td>
<td>+</td>
<td>D . B</td>
</tr>
<tr>
<td>D</td>
<td>+</td>
<td>D . D</td>
</tr>
</tbody>
</table>

\[+ \text{ sign indicates that block scheme is executed in forward direction and } - \text{ sign indicates that the block scheme is executed in reverse direction.}\]

**Algorithm 1:** Implementation of Z-Curve Recursive storage access scheme
Mult(int BA, int BB, int BC, int dim)
{
    if (dim==1)
    {
    }
    else
    {
        Mult(BA,BB,BC,dim/3); b+=BB; c+=BC; 
        Mult(-BA,-BB,-BC,dim/3); b+=BB; c+=BC; 
        Mult(BA,BB,BC,dim/3);      a+=BA; b=b-1; b+=BB; c+=BC;

        Mult(BA,BB,BC,dim/3); b+=BB; c+=BC; 
        Mult(-BA,-BB,-BC,dim/3); b+=BB; c+=BC; 
        Mult(BA,BB,BC,dim/3); a+=a-2; a-=BA; c+=BC; b=b-1; b+=BB; c+=BC;

        Mult(-BA,BB,BC,dim/3); b+=BB; c+=BC; 
        Mult(BA,-BB,-BC,dim/3); b+=BB; c+=BC; 
        Mult(-BA,BB,BC,dim/3); a+=a-2; a-=BA; b=b-1; b+=BB; c+=BC;

        Mult(BA,BB,BC,dim/3); b+=BB; c+=BC; 
        Mult(-BA,-BB,-BC,dim/3); b+=BB; c+=BC; 
        Mult(BA,BB,BC,dim/3); a+=a-4; a+=BA; b=b-1; b+=BB; c+=BC;

        Mult(BA,BB,BC,dim/3); b+=BB; c+=BC; 
        Mult(-BA,-BB,-BC,dim/3); b+=BB; c+=BC; 
        Mult(BA,BB,BC,dim/3); b+=BB; c+=BC;
    }
}

5. Data Access Locality and Cache Efficiency

The main memory of computer is commonly indexed as a one dimensional array mapping a multi-dimensional structure like matrix into main memory is a matter of determining a total linear order of all elements. Traditionally matrices have been stored in main memory using Row-major or Column-major order. For example two 3-by-3 matrix blocks, the algorithms will perform 27 operations. So there is \( n^2 \) elements \( n^3 \) operations. In Peano Cache Oblivious Multiplication Algorithms [Bader M A et.al.2004]

\[
L_A(n) = 3 n^{2/3} \\
L_B(n) = 2 n^{2/3} \\
L_C(n) = 2 n^{2/3}
\]

-------- Peano Cache Oblivious Multiplication Algorithms [Bader M A et. al. 2004]

The data locality functions are very close to the theoretical optimal \( n^{2/3} \). However we also implement cache oblivious sequential access process [Korde. P.S. et.al. 2011] in previous process. In this Z Curve Cache Oblivious method we only change the sequence flow of execution.

6. Conclusion

We here present Z Curve Cache Oblivious Matrix Multiplication. As compare to cache oblivious matrix multiplication [Bader M A et al. 2004] which consist of eight different blocks but in this technique we decrease the number blocks into two category only. However Cache oblivious Peano Curve matrix multiplication algorithms shows that the number of cache misses is order of \( O(\sqrt{M}) \). This is asymmetrically optimal for this algorithm also that is based on recursive block multiplications.

References


