Journal of Information Systems and Communication, ISSN: 0976-8742, Volume 1, Issue 2, 2010, pp-07-09

# Cache oblivious algorithm: Sequential processing

**Korde P.S.[1*] and Khanale P.B.[2]**
[1]Department of Computer Science, Shri Shivaji College, Parbhani (M.S.) India, korde.parmeshwar@rediffmail.com
[2]Department of Computer Science, Dnyopasak College, Parbhani (M.S.) India, prakash_khanale@hotmail.com

**Abstract-** Hardware implements cache as a block of memory for temporary storage of data likely to be used again. Cache Oblivious Algorithm are typically analyzed using an idealized model of cache, but it is much easier to analyze than a real cache's characteristics. But in many cases a constant factor is provably with in a more realistic cache's performance. In this paper an efficient technique is proposed to manage cache memory by using sequential processing. The processed technique introduces some modifications on well-known cache oblivious algorithm process. The proposed processing technique accomplished improved results.
Keyword: Cache Memory, Cache oblivious, Cache hit, Cache miss, Tag etc

**Introduction:**
A Cache Oblivious Algorithm is designed to exploit the CPU cache without having the size for cache or the length of the cache lines etc as n explicit parameter where n is the number of cache lines. Cache Oblivious was conceived by Charles E. Leiserson as early as 1996 and first published by Harald Prokkop in his master thesis at Massachusetts Institute of Technology in 1999. The Cache Oblivious Algorithm is a simple and elegant model to design algorithm that perform well in hierarchical memory models ubiquitous on present Hardware platforms. This model was first formulated in [M. Frigo Charles E. Leiserson. H Prokop and S. Ramachandaran Oct 1999] and since this is the topic of research.
A Cache memory is divided into blocks of words referred to usually as lines. A part of the addresses of requested word called tag, is stored together with the line. This facilitates and accelerates the process at accessing cache to find out whether a line already exist in or not. This process is termed as cache interrogation [A.Seznec, July 1997]. In addition , each line or set in the cache is given an identifier called index; the CPU address is accordingly divided into index and Tag. Tags are used to distinguish between lines[Lindia Null and Julia Lobour, 2006].

**Previous work**
Optimal Cache Oblivious Algorithms are known for the Cooley-Tukey FFT algorithm, matrix multiplication, sorting, matrix transposition and several other problems. Typically a Cache Oblivious Algorithm works by a recursive Divide and Conquer algorithms, where the problems are divided into smaller and smaller sub-problems. The Cache Oblivious Algorithm is given in algorithm Fig 1.1
The Complexity of algorithm is:

Complexity= Q(n,L,Z)
Where Z- total words, L are number of Lines, n number of cache misses that executes. The algorithm gives us Hit and Miss ratio. We have modified the above algorithm via sequential access of processing to maximize the Hit ratio.

```
Cache Oblivious Algorithm
T:Tag
I: Index Address
L: Cache Memory Line
L.T : Tag of line
Comp.: Complement of
LRU: Least Recently used Replacement Algorithm
Input : A Set of CPU-generated address c(Cache)
Output: Hit and Miss percentage
Hit=0
Miss=0
for each address
                found='true'
                calculate I(Index address)
                calculate T(Tag)
for cache Line in the Set referenced by (L)
                if L.T and L.S =01 then
                                Hit = Hit+1
                        found = 'true'
        Endif
    Endfor
                        ( Algorithm fig 1.1)
```

```
Cache Oblivious Algorithm
for each address
                found='true'
                calculate I(Index address)
                calculate T(Tag)
      for cache Line in the Set referenced by (L)
                if L.T and L.S =01 then
                                Hit = Hit+1
                            Add array[T]
                        found = 'true'
        Endif
      Endfor
                        ( Algorithm fig 1.2)
```

In Algorithm fig1.2 modified technique we just transfer the number of Tags into single array.

**Optimization Technique**

The Object Oriented Programming (OOP) tool C++ and C are used to implement the proposed Cache Oblivious Model. Matrix Multiplication is one of the most studied computational problem. We are giving two matrices of m x n and n x p and we want to compute the matrix m x p size. The output is

$$Z_{i,j} = \sum_{K=1}^{n} x_{ik}\, y_{kj}$$

We construct a suite of simple test to decrease the cache miss ratio. Let us consider proposed modified algorithm for two matrices of size m x n and n x p. Each element of the matrix is stored in another single array. The different matrix elements execute in proper manner and gives exact or perfect results. In fig 1.3 we required counter to find the value of last location of first matrix element.

```
counter=0;

for (i = 0; i < n; i++)

    for (j = 0; j < n; j++)

        cin>>a[i,j];
```

This modification technique converts the two dimensions into single dimension. It is easy to manipulate different matrix transformation operations. At the time execution the above matrix will be algorithm fig 1.4

```
for (i = 0;  i <= c1; i++)

  {

    c4 = d[i] + d[c2];

     cout<<c4;
```

In fig 1.4 The second matrix of first element it is started as value of c2. It is easy to operate different operations.

Fig 1.5 Output of Algorithm 1.1

| Total items | Hit | | Miss | |
|---|---|---|---|---|
| | Xi | Xj | Xi | Xj |
| 09 | 03 | 01 | 08 | 08 |
| 16 | 04 | 01 | 15 | 15 |
| 25 | 05 | 01 | 24 | 24 |
| 36 | 06 | 01 | 35 | 35 |

Fig 1.5 Output of Algorithm 1.2

| Total items | Hit | | Miss | |
|---|---|---|---|---|
| | Xi | Xj | Xi | Xj |
| 09 | 09 | 09 | 0 | 0 |
| 16 | 16 | 16 | 0 | 0 |
| 25 | 25 | 25 | 0 | 0 |
| 36 | 36 | 36 | 0 | 0 |

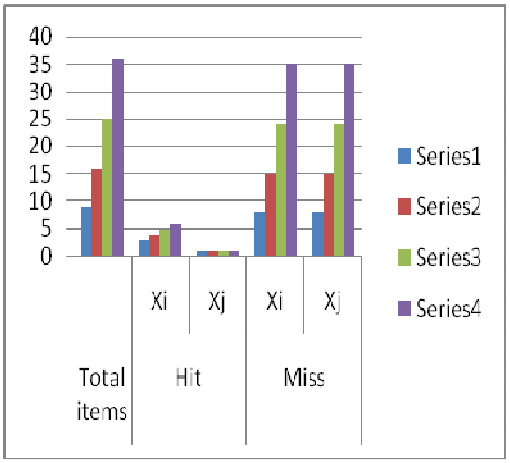


Fig. 1.6- Graphical Representation of Algorithm 1.1

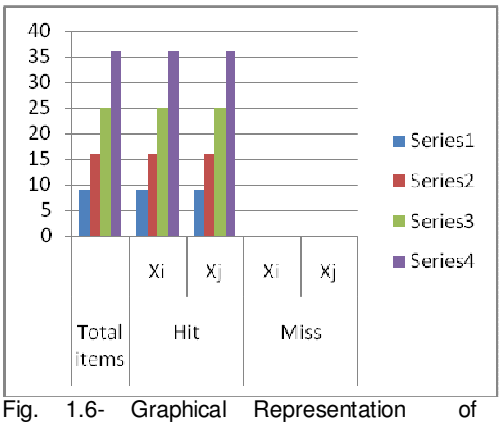


Fig. 1.6- Graphical Representation of Algorithm 1.2

**Conclusion**

We present here problem of number of cache misses done by an implementation optimal oblivious algorithm implementation. In Algorithm fig 1.1 the ratio of misses n: $n^2$ but in proposed technique fig 1.2 it gives us perfect results. The limitation of proposed technique is it is used only for limited data or information; it is not used for varity of wide and huge data or information.

**Future Work**

Sorting String in Cache Oblivious Algorithm Model is still open. Optimal shortest path and minimum spanning forest still need to be explored in the model.

**Reference**

[1] Aggarwal A., Alpern B., Chandra A. K. and Snir M. (1987) *In Proc. 19th Annu. ACM Sympos. Theory Comput., pages* 305-313.

[2] Aggarwal A., Chandra A. K. (1988) *In Proc. 20th Annu.ACM Sympos. Theory Comput., pages* 173-185.

[3] Aggarwal A., Chandra A. K. and Snir M. (1987) *In Proc. 28rd Annu. IEEE*

*Sympos. Found. Comput. Sci.,* pages 204-216.

[4] Aggarwal A. and Vitter J. S. (1988) Commun. *ACM*, 31:1116-1127.

[5] Seznec A. (1997) *A technical Report No. 1114, IRISA-INRIA, Campus de Beaulieu*.

[6] Barry Wilkinson (1996) *Computer Architecture, second edition, Prentice Hall*.

[7] Alpern B., Carter L. and Feig E. (1990) *Uniform memory hierarchies. In focs,* pages600-608.

[8] Amato N. M. and Edgar A. Ramos. (1996) *In Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 166-175.

[9] Lindia Null and Julia Lobour. (2006) *The essentials of Computer Organization and Architecture second edition, Jones and Bartlett*.

[10] Bender M. A., Duan Z., Iacono J. and Wu J. (2002) *In Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* pages 29-38.

[11] Frigo M., Charles Leiserson E., Prokop H. and Ramachandaran S. (1999) I*n Proc 40th Annual Symposium on Foundation of Computer Science*.

[12] Brodal G. S. and Fagerberg R. (2002) *In Proc. 13th Annual International Symposium on Algorithms and Computation, Lecture Notes in Computer Science*.

[13] Brodal G. S., Fagerberg R. and Fagerberg R. (2001) *Technical Report BRICS-RS-01-36, BRICS, Department of Computer Science, University of Aarhus*.

[14] Chatterjee S. and Sen S. (2000) *Cache-e_cient matrix transposition. In HPCA*, pages 195-205.

[15] Chiang Y.-J., Goodrich M. T., Grove E. F., Tamassia R., Vengro D. E. and Vitter J. S. (1995) *In Proc. 6th ACM-SIAM Sympos.Discrete Algorithms*, pages 139-149.

[16] Coppersmith D. and Winograd S. (1990) *Journal of Symbolic Computation*, 9:251-280.

[17] Cormen T. H., Leiserson C. E. and Rivest R. L. (1990) *Introduction to Algorithms. MIT Press, Cambridge, MA*.

[18] Eiron N., Rodeh M. and Steinwarts I. (1998) *In 2nd Workshop on Algorithm Engineering, volume* 16,pages 98-109.

[19] Frigo M. (1999) *In PLDI'99 | Conference on Pro-gramming Language Design and Implementation, Atlanta, GA*.

[20] Frigo M. (1999) *Portable high-performance programs. Technical Report MIT/LCS/TR-785*.

[21] Frigo M., Charles E. Leiserson, Prokop H. and Ramachandran S. (1999) *In Proc. 40th Annual Symposium on Foundations of Computer Science*.

[22] Mathias Spjuth, Martin Karlsson and Erik (2003) *Licentiate Thesis 2003-009, Department of Information Technology, Uppsala University*.

[23] Hill M.D. (!987) *A PhD thesis presented to the University of California,Berkeley*.

[24] Graham R. L., Knuth D. E. and Patashnik O. (1989) *Concrete Mathematics. Addison-Wesley, Reading, MA*.

[25] Hennessy J. L. and Patterson D. A. (1990) *Computer Architecture: A Quantitative Ap-proach. Morgan Kaufmann Publishers, Inc*.

[26] William Stallings (2006 *Computer Organization and Architecture, seventh edition,Prentice Hall*.