Dynamic Programming for Improving Cache Efficiency

korde P.S., Khanale P.B, *Member, IAENG*

***Abstract*—A dynamic programming is a solution for special type of application such as context free language recognition matrix chain multiplication and optimal binary trees. Unfortunately these implementation exhibit poor cache performances. The purpose of this paper is to design, implement empirically evaluates divide and conquer algorithm that exhibit cache performance problem.**

Keywords: **Dynamic Programming, Cache Oblivious, Cache efficient, Cache hit, Cache miss;**

I. INTRODUCTION

In modern hardware configuration of system, the main memory is divided into two levels, low and high respectively, which are L1 and L2 cache, now days it is extended up to L3 cache. Optimizing cache performance to achieve better overall performance is a difficult problem. Modern computers are including deeper and deeper memory hierarchies to hide the cost of cache misses. Dynamic Programming is widely used algorithmic technique [7]. However, standard implementation of these algorithms often fails to exploit the temporal locality of data which leads to poor I/O performance [10].

1.1 Cache Efficiency:

The I/O model [1] is simple abstraction of memory hierarchy which consists of cache size m and its partitioned number of blocks B. A cache complexity of an algorithm is measured in number of cache misses and number of blocks transfers or I/O operations it causes. The cache oblivious model [10] with additional feature of this model that do not use knowledge of M and B. it is flexible and portable and adapts all levels of memory hierarchy. There are some aspects which lead to it having good cache performance. The first, it must process data in place if it needed extra memory for input and output. For example in quick sort, merge sort process. Second it executes in divide-conquer approach for large size data for example matrix multiplication [3]. It assumes that the parameters M, B are unknown to the algorithms. In present algorithm that has been dealt with yet, need the programmer to specify M, B.

Korde P.S. is with Department of Computer Science, Shri Shivaji College,Parbhani (M.S.) India
Telephone no : 02452-224197
Mobile no:9420530657,9422878547
Email:korde.parmeshwar@rediffmail.com, korde.parmeshwar@yahoo.com

Khanale P.B.is with Department of Computer Science, Dnyopasak College Parbhani (M.S.) India
Email prakashkhanale@gmail.com

1.2 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 $\mathcal{O}$ ($\log_B N$) per operations.

Cache Oblivious Model [5] arose in particular model multilevel memory hierarchies. The process is: analysis hold for an arbitrary M and B at each level memory hierarchy. The algorithm could not worry about block replacement strategy. The development of these algorithms which give solutions to problems finding the best ways to enforce data locality.

R. Chowdary[8] present an efficient cache oblivious algorithms for several dynamic programs. These includes new algorithm with improved cache performance for largest common subsequence(LCS) and Least Weight Subsequence. His present new cache oblivious framework called Gaussian Elimination Paradigm (GEP) which gives cache algorithm for Floydwarshall all pairs shortest paths in graph among other problems.

Dynamic programming is a powerful algorithmic tool [8] because most dynamic programming which have high complicity (n^2,n^3 and worse). There are various techniques significant efforts has been put to reduce complexity several algorithm [4], Monge propery[2] has been carryout.

The cache oblivious approach invented by M. Frigo[5] does not use tuning parameters but relies on structure of an algorithm to achieve good cache performance

2. Cache Oblivious Dynamic Programs

Dynamic Programming is a method for solving complex problems by breaking them into simpler sub problems. It is applicable to problems exhibiting properties of overlapping sub problems which are only slightly smaller and optimal substructure. It takes less time than native methods.

We present a cache-oblivious implementation of the classic dynamic programming algorithms using Transferring values and storing value methods. Our algorithms continues to run in $\mathcal{O}$(m+n) space and perform $\mathcal{O}$(mn/BM) block transfers. Experimental Algorithms shows the faster than widely used algorithms. We consider three variants algorithms as

1) Diagonal Algorithms for simple Dynamic programming
2) Horizontal Algorithm for simple Dynamic programming
3) Vertical Algorithm for simple Dynamic programming

Diagonal Algorithms for simple Dynamic programming	Horizontal Algorithm for simple Dynamic programming	Vertical Algorithm for simple Dynamic programming
For d=2 to n For i= 1 to n-d+1 For j=d+i-1 For k =1 to j-1 D[i,j]=D[i,j] +D[i,k].D[k+1, j]	For i = n-1 to 1 For j = i+1 to n For k =1 to j-1 D[i,j]=D[i,j]+D[i,k].D[k+1,j]	For j = 2 to n For i = j-1 to 1 For k =1 to j-1 D[i,j]= D[i,j]+D[i,k].D[k+1,j]

Fig 1: Cache oblivious Dynamic Programming Algorithms

In simple Dynamic programming we must define storage D[i,j], 1< = i < = j < = n. These algorithms have poor cache performance.

3. Cache allocation and organization

The cache memory is organized in blocks or lines. When data is transferred between caches or between cache and main memory the entire block containing the desired address is transferred.

There are three ways in which the cache memory is organized depending on how a block is placed and found in the upper level of the memory hierarchy.

The first cache organization is called direct mapping. In a direct mapped cache a given address can only appear in one possible location in cache, and hence it is easy to map an address to its location in cache, and it is easy to determine if an address is in cache. The mapping is done using the formula: (Block address) MOD (Number of blocks in the cache) .

The disadvantage of a direct mapped cache is that an address might cause a miss while the cache is not full. If a block can be placed anywhere in the cache the cache is said to be fully associative. A fully associative cache must be searched to determine if the desired address is in cache. However misses only occur when the cache is full.

The last cache organization called a set associative cache is a compromise between these two extremes. In this case the cache is split into sets, with each set containing the same number of blocks. A block that maps into a set can be placed in any block in the set.

The cache is characterized by the cache access time, which represents the cache latency in case of a hit. Let's call this the Hit Time or $t_{hit.}$ The probability of having a cache miss is P_{miss}. The transfer time between the main memory and cache is T_{miss} or the miss penalty time This means that the effective cache latency time is equal to

$$t_{avg} = (1- P_{miss})\, t_{hit} + P_{miss}\,(t_{hit} + T_{miss}) =$$
$$t_{hit} - P_{miss}\, t_{hit} + P_{miss}\, t_{hit} + P_{miss}\, t_{miss} =$$
$$t_{hit} + P_{miss}\, t_{miss}$$

Fig 2. Mihai Alexandru Furis et al [6]

4. The Gap Problem

The gap problem [8] is a generalization of the edit distance problem that arises in execution speed. When transferring a values of D[i,j] into compiling time. When we implement two methods as Transferring values, Storing values in array.

$$D[i,j\} = \begin{cases} 0 \text{ if } i=j=0 \\ W(0,j) \text{ if } i=0; 1<j<n \\ W'(0,1) \text{ if } j=0;\ 1<i<m \\ G[i,j] \text{ if } i,j>0 \end{cases}$$

Fig 3: Gap Equation

Assuming m=n, this problem can be solved in internal memory [11] in $\mathcal{O}(n^3)$ time using $\mathcal{O}(n^2)$ space: this algorithms incurs $\theta(n^3/B)$ I/Os[11]

.

5. Experimental Results

We implemented three variants of our algorithms 1) Diagonal Algorithms for simple Dynamic programming 2) Horizontal Algorithm for simple Dynamic programming 3) Vertical Algorithm for simple Dynamic programming. There are several things to note down about Diagonal algorithms. Horizontal and Vertical Algorithms solve problems in different orders. The given algorithms have better performance as cache efficiency.

When we implement Transferring values and Storing values through these methods cache hit and cache miss ratio is n=n/2. We also measure the execution time with different array size for cache performance.

Transferring Values Vertical Algorithms

```
void main( )
{
 int a[2][2],b[2][2];
 int i,j,k,l;
 cout<<"\n  Enter the elements of matrix ";
 for(i=0;i<=2;i++)
 {  for(j=0;j<=2;j++)
 {   cin>>a[i][j];
  }   }  for(j=2;j<=2;j++)
    {  for(i=j-1;i>=0;i--)
    {    for(k=i;k<=j-1;k++)
      {      a[i][j]=a[i][j]+a[i][k]*a[k+1][j];
        }  }            }
```

Storing Values in array Vertical Algorithms

```
void main( )
{
int a[3][3]={1,2,3,4,5,6,7,8,9};
 int b[2][2];
 int i,j,k,l;
 clrscr();
   for(j=2;j<=2;j++)
    {     for(i=j-1;i>=0;i--)
    {    for(k=i;k<=j-1;k++)
      {      a[i][j]=a[i][j]+a[i][k]*a[k+1][j];
        }      }      }
```

Fig 4: Dynamic Vertical Algorithms

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 of two methods shown in table.

S r.no .	Matrices	Method	Cache hit	Cache miss
1	3 × 3	Diagonal Algorithms	6	4
2	3 × 3	Horizontal Algorithms	5	4
3	3 × 3	Vertical Algorithms	5	4
4	4 × 4	Diagonal Algorithms	10	6
5	4 × 4	Horizontal Algorithms	10	6
6	4 × 4	Vertical Algorithms	11	5
7	5 × 5	Diagonal Algorithms	6	18
8	5 × 5	Horizontal Algorithms	6	19
9	5 × 5	Vertical Algorithms	7	18

Fig 5: Ratio of Cache hit and Cache miss

In 2×2 Matrices, 3×3 Matrices the cache miss ratio n as n+1 but when size of array increase and length of cache miss ratio also goes to rises as shown in fig.

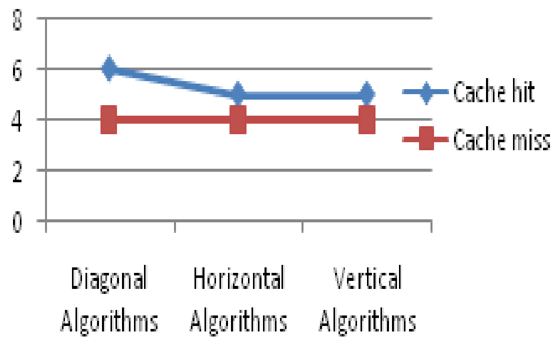

Fig 6: 3×3 Matrices cache miss ratio

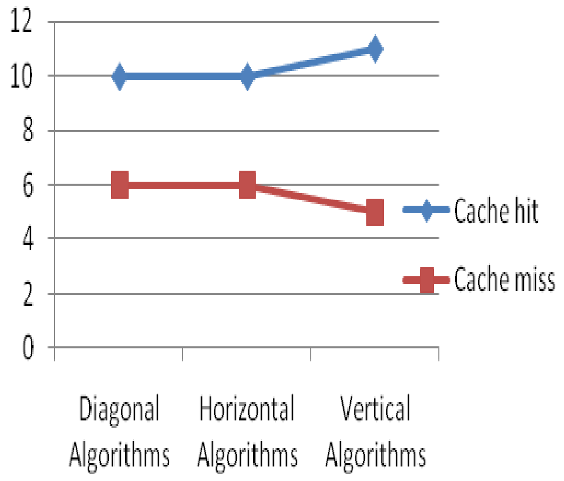

Fig 7: 4×4 Matrices cache miss ratio

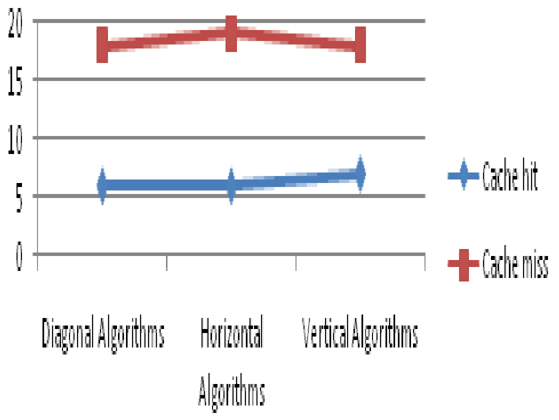

Fig 8: 5×5 Matrices cache miss ratio

We tested three algorithms Diagonal Algorithms for simple Dynamic programming, Horizontal Algorithm for simple Dynamic programming, Vertical Algorithm for simple Dynamic programming for time execution as shown in Fig 8.

6 Time Slots of Dynamic Programming Vertical Algorithms

Sr.no.	Matrices	Time in nanoseconds
01	3×3 Matrices	0.164835
02	4×4 Matrices	0.164835
03	5×5 Matrices	0.164835
04	6×6 Matrices	0.21978
05	7×7 Matrices	0.274725
06	8×8 Matrices	0.274728
07	9×9 Matrices	0.32967

Fig 9: Time slots for dynamic Programming

All three methods which required execution time as shown in tables graphically how it process shown in fig 10.

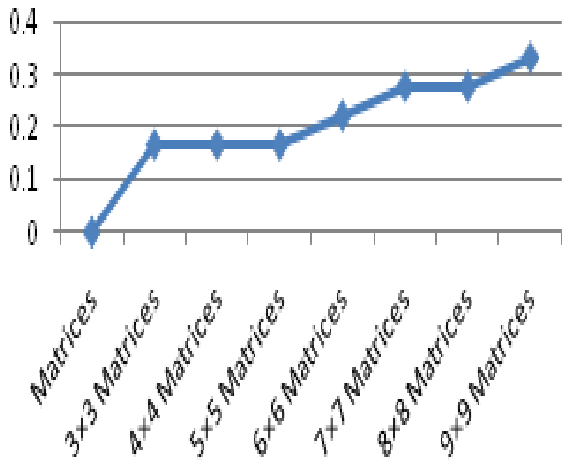

Fig 10: Slots of Dynamic Programming

All figures shows cache performance of all algorithms in L1 and L2 cache. The normalized cache miss are shown on difference between three standard algorithms have a more cache misses on largest problem size. It is interesting to calculate cache misses and cache simulations.

7. Conclusions

We have demonstrated Diagonal Algorithms, Horizontal Algorithms, and Vertical Algorithms for solving simple dynamic problem. These have good cache performance and cache aware sense.

Our Framework can be applied for several other dynamic programming problems including local alignments, multiple sequence alignments, sorting functions, sum of pair functions, objective functions.

8. Future Work

We presented practically Dynamic Algorithms with different methods, the methods which shows execution process and cache behavior.

Fig 10 gives cache performance Dynamic Algorithms, the standard algorithms have a more cache misses on largest problem size. We were not able to efficiently implement it.

References

[1] A. Aggarwal and J. Vitter, "The Input/Output Complexity of Sorting and Related Problems," Comm. ACM, vol. 31, pp. 1116-1127, 1988.
[2] Bingsheng He, Qiong Luo Cache Oblivous Nested Loop Joins CIKM06 2006 ACM
[3] Joon Sang Park Optimizing Graph Alogorithms for Improved Cache Performance IEEE Transactions on Parallel and Distributed system Vol 15 No 9 Sept 2004.
[4] M. Sniedovich. Dynamic Programming. The Marcel Dekker, Inc., New York, NY, USA, 1992.B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.
[5] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In Proc. of the 40th Annual Symposium on Foundations of Computer Science , pages 285–297, 1999.
[6] Mihai Alexandru Furis Cache Miss Analysis of Walsh-Hadamard Transform Algorithms Master thesis
[7] R. Bellman. Dynamic Programming. The Princeton University Press, Princeton, New Jersey, 1957.
[8] (a) R. Chowdhury V. Ramachandaran Cache Oblivious Dynamic Programming NSF Grant CISE Research Infrastructure NSF Grant CCF0514871
(b) R. Chowdhury V. Ramachandaran Cache Oblivious Dynamic Programming for Bioinformatics IEEE Bioinformatics Vol7No3 July Sept 2010
[9] Steven Huss-Lederman, Elaine M. Jacobson, Jeremy R. Johnson Implementation of Streassen's Algorithms 089719-854-1/1996IEEE Explore
[10] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. MIT Press, 2nd ed., 2001.
[11] Tobias Johnson Cache-Oblivious of an Array's Pair May 7 2007