OPTIMISED MATRIX MULTIPLICATION

Name: Sravya.P
Dept name :School of Informatics, Computing, and Cyber System
Organization:Northern arizona university
City:Arizona state, flagstaff
Email: sp2379@nau.edu

***Abstract*—Comparing the Matrix multiplication by main memory, cache memory and Registers to conclude which algorithm takes less time to compute and perform the algorithm.**

Keywords—main memory, cache, Registers.

I. INTRODUCTION

In this project we are studying about optimization of an application taking matrix multiplication as reference.

The reason for this project is, for a huge application it uses a large amount of time, so we try to make other possible ways to process the project in less time then required, and understand the computer architecture.

Overview of the problem :

If we want to build a huge project the time is obviously very large So, we try to reduce the time by taking matrix multiplication as our application.

We solve this problem by the use of cache memory, and registers because of the fact that they are close to CPU and may help to compute the application in less time than the main memory.

II. BACKGROUND

Main Memory :

Main memory is where programs and data are kept when the processor is actively using them. When programs and data become active, they are copied from secondary memory into main memory where the processor can interact with them. A copy remains in secondary memory.

Main memory is sometimes called RAM. RAM stands for Random Access Memory. "Random" means that the memory cells can be accessed in any order. However, properly speaking, "RAM" means the type of silicon chip used to implement main memory.

Cache memory:

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
Simply speaking caches are the subsets of main memory.

Register Memory:

Register memory is the smallest and fastest memory in a computer. A register temporarily holds frequently used data, instructions, and memory address that are to be used by CPU.

Register holds a small amount of data around 32 bits or 64 bits. The speed of the CPU depends on the number and size of register that are build into the CPU.

The areas we should know about this project are

1. Cache memory locality:
   Temporal locality: Referred to the data items that are used frequently
   Spatial locality : Referred to address that is located near by in cache memory
2. Tags in cache memory:
   Tags are used to specify the data
   Tags along with the index value allow use to find the exact data
3. Cache Blocking:
   Since the entire matrix may not fix into the cache memory, we break the entire matrix into blocks and do the calculation
4. Hit rate/ miss rate:
   Hit rate is used to determine if the data is present in the cache memory
   Miss rate is used to determine if the data is not present in the cache memory
5. Registers
   Since the register are in cpu use this method to calculate the matrix multiplication and expect it to be fastest among all.

DESCRIPTION

Cache memory locality:

1. Temporal Locality:

This is referred to the data items that are used frequently

Advantages:

1. We can keep the rows in the first matrix row as common and change the columns in the second row, because we use the same row to compute the first column of the resultant matrix.
2. We can also use the resultant matrix as frequently used matrix since we refer that frequently to store the data.

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

2 Spatial Locality:

This is referred to the address that is located near by in the cache memory.

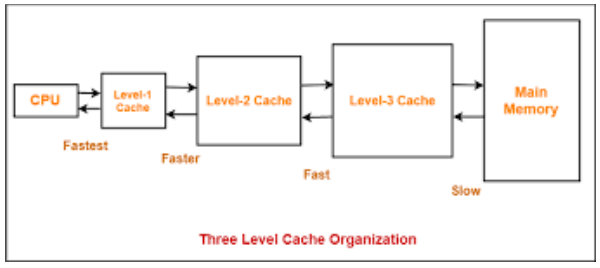
Advantages:

1. We will transfer not just one byte or word from the main memory to the cache but a series of sequence location called blocks,
2. We assume that it is necessary to refer the cache before a referencing to the main memory to fetch a word, to check if the information is held there or not which will reduce the time.

Introduction to cache memory:

There are 3 levels of cache memory

1. L1 cache
2. L2 cache
3. L3 cache



Tags and index in cache memory:

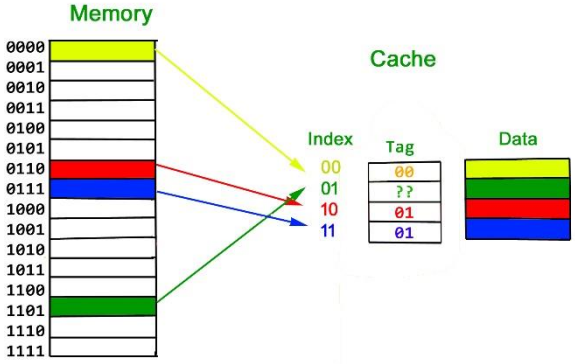
How to we know that we extracted the exact data we want?

This can be achieved by tags and index in cache memory

For example:

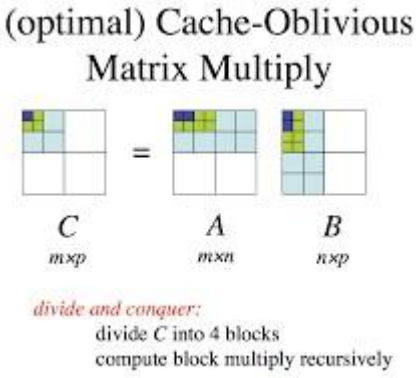
If we want green colour then

Index: 01 Tag : 11 : together gives us the exact value we want



Because of the fact that the cache memory is less in size we can only fit a certain size of data in the cache memory, so we divide the entire matrix into block to fit in the cache memory. This process is called **cache blocking**

When two loops uses the same data it is beneficial to reduce amount of data access between the loops, means cache line fetch into the cache by the first loop may still be in cache when the second loop is running, s it does not have to waste time to fetch the same data again.



How do we know that the data in the cache memory?

We can achieve this by using performance analysis which give ous the informationabout the cache miss and cache hit

Cache hit : referes that the element is in cache memory

Cache miss: referes that the element is not in the cache memory

The cache ratio can be calculated by:

Miss ratio = 1- hit ratio. (or)

Hit ratio = 1 – Miss ratio. (or)

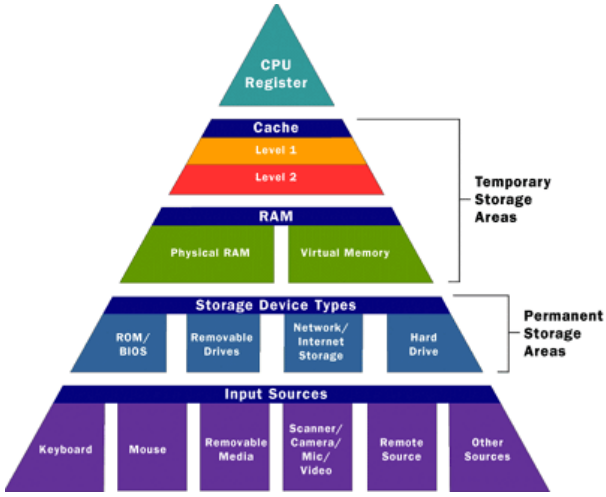
Miss – Hit = 1

Conclusion of cache memory:

We take the advatage of the fact that the cache is near to the CPU and using those cache can reduce the time used to fetching the data from the main memory, by placing the data in cache memory

REGISTERS INTRODUCTION:

We use register believing from the fact that since the register are close to CPU the computational time to process in register are faster that the cache memory.



Problem with registers:

When the processor loads the data from the main memory into the registers it loads in a scalar form, The processor is capable of loading wide range of single instruction multiple data(SIMD) means one at a time, By the time the last data is fetched and loaded we are not sure that the data of given size is still present in the location,

The problem in the registers is CPU is not fully utilized because of the memory bound.

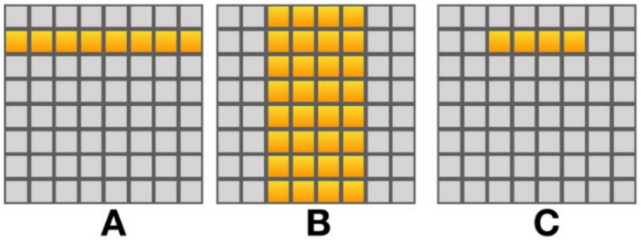
Since the register are not very large we divide the entire matrix into blocks and compute the values through blocks called register blocking.

Solution:

The problem above is that the memory is been wasted,

so the solution is to take the data of desired size and fill the memory by placing the dummy values in the first iteration, and while computing we replace the dummy data with the actual data we need and compute the result by this we have 2 advantages.

1. We are sure that the data we need is there without flushing out by placing the dummy values and replacing each while computing.
2. We can keep the colums in matrix A as constant and then change the values in matrix B by which we will get the result of n values in the resultant matrix.



Results:

Trails	N	Matrix multiplication	Cache memory	Registers	Cache miss	Cache hit
1	300	0.15	0.16	0.11	4424	4423
2	500	0.77	0.74	0.58	24842	24841
3	700	2.04	2.02	1.54	74107	74106
4	1000	5.45	5.68	4.33	258021	258020
5	1500	19.24	19.32	14.33	1619964	1619963
6	2000	59.97	52.57	45.98	84331367	84331366
7	2500	158.09	139.90	67.91	1874242480	1874242479
8	3000	209.91	203.98	123.33	3380111043	3380111042
9	3500	522.00	404.36	284.19	5380649268	5380649267
10	4000	681.65	435.14	451.85	8048920483	8048920482
Average		165.926	137.187	99.415		

Conclusions:
The time for small values of N in cache memory is equal to normal matrix multiplication, as the N values increases the time decreases in cache memory. But Irrespect of the size the time is small for register even for the small values of N.

We can say that the normal matrix multiplication has large average time value because of the fact that it is used to fetch the data from main memory.

The average time by cache memory is less than the normal matrix because of the fact that the it is close to CPU and uses less time to fetch data and compute the algorithm.

The averge time for register is much less than the other two because the register are in CPU which takes little time to fetch the data and compute the result.

We can get the cache miss and cache hit by using perf and those are listed in the result table.

By the result we come to the conclusion that register use less time than cache memory and normal matrix multiplication based on the fact that registers are close to the CPU and takes less time to fetch data and compute the result matrix.

Lesson learned:
The best approach when we want less time to compute the application is by using register if possible or the second option is by using cache memory