Hw9

1. Given,
Consider a memory system with a level 1 cache of 32 KB and DRAM of 512 MB with the processor operating at 1 GHz. The latency to L1 cache is one cycle and the latency to DRAM is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is four words).

We need to calculate the peak achievable performance of a dot product of two vectors.
We know that the processor can fetch a four-word cache line every 100 cycles.
The vectors are laid out linearly in memory, computation performs eight FLOPs (four multiply-adds) on 2 cache lines can be performed in 200 cycles. This is because a single memory access fetches four consecutive words in the vector. Therefore, two accesses can fetch four elements of each of the vectors.
This corresponds to a FLOP every 25 ns, for a peak speed of 40 MFLOPS.
Therefore, the peak achievable performance is 40 MFLOPS.

2. Given,
Let us consider the problem of multiplying two dense matrices of dimension 4K x 4K.
We need to calculate the peak achievable performance using a three-loop dot-product based formulation.
Here, the computation performs eight FLOPs (four multiply-adds) on 5 cache lines (one for a matrix and four for column major access to matrix b), and the corresponds to a speed of 16 MFLOPS.

Therefore, the peak achievable performance is 16 MFLOPS.

3. How does cache associativity affect the cache performance?

A: In order to achieve good cache performance, we need to maximize both Hit Ratio and search speed. Let's look into each of the cache associativity, how it affects the cache performance.

1. Direct Mapped Cache: The direct mapped cache is the simplest form of cache and the easiest to check for a hit. Since we know that there is only one possible place that any memory location can be cached, there is nothing to search. The line either contains the memory information we are looking for, or it doesn't. But, direct mapped cache also has the worst performance, because again there is only one place that any address can be stored.

Let's look at 512 KB level 2 cache and 64 MB of system memory. And cache has 16,384 lines and each one is shared by 4,096 memory addresses. In the worst case, imagine that the processor needs 2 different addresses (call them X and Y) that both map to the same cache line, in alternating sequence (X, Y, X, Y). This could happen in a small loop. The processor will load X from memory and store it in cache. Then it will look in the cache for Y, but Y uses the same cache line as X, so it won't be there. So Y is loaded from memory, and stored in the cache for future use. But then

the processor requests X, and looks in the cache only to find Y. This conflict repeats over and over. The net result is that the hit ratio here is 0%. This is a worst case scenario, but in general the performance is worst for this type of mapping.

2. Fully Associative Cache: The fully associative cache has the best hit ratio because any line in the cache can hold any address that needs to be cached. This means the problem seen in the direct mapped cache disappears, because there is no dedicated single line that an address must use. However, cache suffers from problems involving searching the cache. If a given address can be stored in any of 16,384 lines, it is difficult to know where it is. Even with specialized hardware to do the searching, a performance penalty is incurred. And this penalty occurs for all accesses to memory, whether a cache hit occurs or not, because it is part of searching the cache to determine a hit. In addition, more logic must be added to determine which of the various lines to use when a new entry must be added. All this overhead adds cost, complexity and execution time.

3. **N-Way Set Associative Cache:** The set associative cache is a good compromise between the direct mapped and set associative caches. Let's consider the 4-way set associative cache. Here, each address can be cached in any of 4 places. From the example in the direct mapped cache, where we accessed alternately two addresses that map to the same cache line, they would now map to the same cache set instead. This set has 4 lines in it, so one could hold X and another could hold Y. This raises the hit ratio from 0% to near 100% . As for searching, since the set only has 4 lines to examine this is not very complicated to deal with, although it does have to do this small search, and it also requires additional circuitry to decide which cache line to use when saving a fresh read from memory.

4. Just providing deeper memory hierarchies does NOT bridge the gap between processor And memory performance. Why?

A: we know that**,** over the past twenty years, processor speed increases dramatically outstripped performance increases for standard memory chips, in both latency and bandwidth. As a result there is a substantive performance gap between processors and memory.

To bridge this gap, architects have included deeper levels of fast cache between the processor and main memory, thus creating a memory hierarchy. The data access latency between the processor and a higher level of the memory hierarchy is often orders of magnitude lower than the latency to main memory. To achieve high performance on such machines, compliers must optimize applications to reuse, to the maximum extent possible, data elements that are fetched into cache. This optimization, which shortens the distance between two accesses to the same datum, is often referred to as locality enhancement.

Although blocking mechanism is for compiler locality enhancement, both loop interchange And loop fusion can be important tools for improving memory performance. Loop interchange improves locality by getting the loop with the most locality to the innermost position. Loop fusion, on the other hand, improves performance by merging loops that access similar sets of memory locations. After fusion, the accesses to these locations are brought closer together and thus can be reused inside the

fused loop. Traditionally, loop fusion can only be applied to a sequence of loops at the outermost position of a code segment. Although loop interchange can be applied before fusion to bring the desired loops outermost, loop interchange can also place the wrong loops at the outermost position for fusion, because it cannot foresee the overall effect of fusion transformations.

5. The Concurrent Average Memory Access Time (C-AMAT) extends the traditional AMAT with concurrency factors.

C-AMAT is defined as the average memory access time considering the overlap between multiple hit and miss accesses. C-AMAT is equal to the total memory access cycles divided by the total number of memory accesses. Let TMemCycle represent the total number of cycles executed in which there is at least one outstanding memory reference, CMemAcc represent the total number of Memory accesses.

C-AMAT = TMemCycle/ CMemAcc

However, due to the advanced structures of the modern cache and memory systems, such as pipelined cache, multi-ported cache, non-blocking cache, etc., multiple hit accesses and miss accesses could be overlapped with each other. When counting the memory access cycles, the overlapping mode is adopted. The overlapped mode means when there are several memory Accesses co-existing during the same cycle, but TMemCycle only increases by one. Another important feature of TMemCycle is that TMemCycle only includes the clock cycles with cache access activities; the cycles without memory references are excluded.

⇨ C-AMAT is the reciprocal of APC.

C-AMAT =1/APC= TMemCycle/ CMemAcc

The direct relation of C-AMAT and APC translates the correctness and accuracy of APC to C-AMAT. However, like AMAT, C-AMAT is designed to signify component-wide performance analysis. A component-based, parameterized formula of C-AMAT

Needs to be derived in order to put C-AMAT in use.

Now, we will derive traditional AMAT, the traditional AMAT is calculated as HitCycle+MR×AMP. Hit Cycle is the hit time of memory accesses, MR is the miss rate of cache accesses and AMP is the average miss penalty. AMP is calculated as the sum of all single miss access latency divided by the total number of miss accesses. The deficiency of AMAT is that there are no factors to describe the concurrency of memory Accesses, in either the hit part or the miss part of the formula. AMAT assumes the memory accesses are sequential, one after another.

And AMAT ignores the relation between hits and misses existing at the same cycle. we extend the AMAT formula by Introducing concurrency parameters for the hit and miss accesses, and propose a new counting method for MR and AMP which considers the relation between concurrent hits and misses.

The extended formula is then theoretically proven to be equal to (H/Ch)+MR*AMP/Cm ----→1

Given,

Hit time =H= 5ns
Hit concurrency=Ch =3
Pure miss rate =MR=35%
Pure miss concurrency=Cm=5
Pure miss penalty=AMP=1000ns

Concurrent average access time for a memory word = (5/3)+.35*1000/5
=1.6+70
=71.6 ns

Therefore, Concurrent average access time for a memory word is 71.6ns.

6. Now, we need to calculate recursive formula for calculating the current average access time (CAMAT) For a word in a system with three levels of cache.
From the equation 1 from above C-AMAT, we will get as follows,

$C\text{-}AMAT_{1=} (H_1/Ch_1)+pMR_1 * \eta_{1*} C\text{-}AMAT_2$

$C\text{-}AMAT_{1=} (H_1/Ch_1)+pMR_1 * pAMP_1/Cm_1$

$C\text{-}AMAT_{2=} (H_2/Ch_2)+pMR_2 * pAMP_2/Cm_2$
$\eta_{1 =(}pAMP_1/AMP1)*(Cm_1/CM_{1)}$

The physical meaning of η is R1/R2,Where R1 is pure miss cycles /miss cycles and R2 is pure miss /miss .
The actual penalty of L_2 is $C\text{-}AMAT_2$.Therefore, the actual delay impact is $\eta_{1*} C\text{-}AMAT_2$. η_1 is the L1(concurrency) data delay order.

Now, determine the concurrent average access time for a memory word in the described system as follows:
$C\text{-}AMAT_{1=} (H_1/Ch_1)+pMR_1 * \eta_{1*} C\text{-}AMAT_2$
$C\text{-}AMAT_{2=} (H_2/Ch_2)+pMR_2 * \eta_{2*} C\text{-}AMAT_3$
$C\text{-}AMAT_{3=} (H_3/Ch_3)+pMR_3 * \eta_{3*} C\text{-}AMAT_4$
$C\text{-}AMAT_{4=} (H_4/Ch_4)+pMR_4 * \eta_{4*} C\text{-}AMAT_5$

We will calculate concurrent average access time by recursively.
$C\text{-}AMAT_4$ =(100/6)+0 =16.6
$C\text{-}AMAT_3$ =(35/4.5)+.15*4* $C\text{-}AMAT_4$ =7.7+0.6*16.6=7.7+9.96=17.66
$C\text{-}AMAT_2$ =(10/3.25)+.3*2.25* $C\text{-}AMAT_3$ =3.07+0.675*17.66=3.07+11.92=14.99
$C\text{-}AMAT_2$ =(5/2.75)+.45*1.5* $C\text{-}AMAT_2$ =1.81 +0.675*14.99=1.81+10.11=11.92

Therefore, Concurrent average access time for a memory word is 11.92ns.

7. We need to optimize the following code to reduce the miss rate of a memory system.

Given Code:
```
for (i = 0; i < N; i++) {
 A[i] = B[i] + C[i];
 }
 For (i = 0; i < N; i+) {
 D[i] = B[i] + A[i];
 }
```

We can reduce the miss rate by doing compiler optimizations. In Loop fusion, many programs have separate loops that operate on the same data. We can optimize by Combining these loops allows a program to take advantage of temporal locality by grouping operations on the same (cached) data together.

In the above code, there are 2 misses per access to A and B .where we should combine the two loops by grouping operations on the same data .In optimization code, we have only one miss per access.

Optimized code:
```
for (i = 0; i < N; i++) {
   A[i] = B[i] + C[i];
   D[i] = B[i] + A[i];
}
```

8. What are the hardware technologies which can increase Hit Concurrency in the Formula of C-AMAT and explain why briefly?

A: There are few hardware technologies which can increase Hit Concurrency in the Formula of C-AMAT. They are:

1. Pipelined cache access
2. Multi-banked caches

1. Pipelined cache access: By increasing the pipeline's stages, we can increase Hit Concurrency by accessing the data. As the number of pipeline stages is larger, the number of cache accesses is larger in a fixed period of time. Most methods exists, need to use them for efficient memory concurrency. Gpu uses more pipelining effectively. In, Three Stages Pipeline and Four Stages Pipeline. In, Three Stages Pipeline, tags and read data compressed in same set, where as in Four Stages Pipeline increases with different sets (larger), therefore cache accesses is larger.

2. Multi-banked caches: Memory banks could be cache 2, by increase the number of banks. It divides cache into independent blocks to support simultaneous access. Each bank could have pipeline (subarray), therefore it is two level concurrency , and also lot of data computing, intend to data concurrency such as multi core (concurrent data access), multi thread etc. also increases hit concurrency.

9. What is the Non-block cache and what are its characteristics?

A: we know that, caches can only handle one request for data at a time. If data is not found within the cache during a request for information ("miss"), then the data is fetched from main memory. During this data fetch, the cache is usually "blocked", meaning that no other cache requests are allowed to occur.

In non-blocking cache, however, this is not the case. Rather than sit idly by twiddling its thumbs waiting for the miss to be filled from DRAM, the cache takes more requests from the processor and attempts to fill them.

- It enables multiple outstanding cache misses, like pipelining memory system.
- Extremely important for hiding memory latency.
- It is more complicated than blocking caches.
- Completion buffer controls the entries of requests and ensured that departures takes place in order even if loads complete out-of-order.
- Do not block cache references that do not need the miss data and Service multiple miss requests to memory concurrently.