Homework 4

Md Ali A20439433
CS 546: Parallel and Distributed Processing

November 4th, 2021

Exercise 1. You are given a non-pipelined processor design which has a cycle time of 10ns. What is the best speedup you can get by pipelining it into 5 stages? If the 5 stages are 1ns, 1.5ns, 4ns, 3ns and 0.5ns, what is the best speedup you can get compared to the original processor?

Proof. This is split in two parts

I. To find the speed up we would simply see that since we have 5 stages, than we can get 5 times the speed up. We can find the new cycle time with the following equation

$$New\ Cycle\ Time = \frac{Current\ Cycle\ Time}{Speed\ up} = \frac{10ns}{5} = \boxed{2ns}$$

Hence given above we get our speed up.

II. We are now given what each stage can do, the cycle time that is limiting the system or known as the slowest stage is the stage with a cycle time of 4ns, so we will rearrange our previous equation for speed up this time.

$$Speed\ Up = \frac{Old\ Cycle\ Time}{New\ Cycle\ Time} = \frac{10ns}{4ns} = \boxed{2.5\ times\ speed\ up}$$

Hence this is as fast as we can go with the given stages.

□

Exercise 2. How does cache associativity affect the cache performance?

Proof. When dealing with cahce performance, we will essentially have to take into account three major components which are miss rate, delay, and area. Now, cache associative affects cache performance due to having more associative cache will ultimately lower miss rates but increase delay. Vice versa, with less associative cache, you will have higher miss rates but lower the delay. In terms of what would be best performance practice, I would say it depends on the scenario but in most cases it would be better to have more associative cache as it usually would be worth the minor delays, a protocol that could be used would be set-associative cache.

□

Exercise 3. Just providing deeper memory hierarchies does NOT bridge the gap between processor and memory performance. Why?

Proof. Providing a deeper memory hierarchy does not bridge the gap solely due to the fact that system processors are very fast while memory is still lacking behind. The concept of memory hierarchy is for the CPU to ultimately be able to manipulate data but it is only able to get memory from the cache, meaning that sooner or later you will run out of cache memory in this case. So the growing gap between CPU and memory speeds hinders the overall computer's performance. There is current developments such as hierarchy strategies, improvements of bus controllers, and creating smarter memories are trying to close the gap at the current moment.

□

Exercise 4. Derive the formula for calculating the current average access time (C-AMAT) for a word in the first level cache of a system. The following values for the theoretical system is in the homework document. Determine the current average access time for a memory word in the described system.

Proof. The formula would be

$C\text{-}AMAT = \dfrac{H_C}{C_H} + pMR \cdot \dfrac{pAMP}{C_M}$, where we have that

$H_C = Hit\ cycle$
$C_H = Hit\ Concurrency$
$pMR = Pure\ Miss\ Rate$
$pAMP = Average\ Pure\ Miss\ Penalty$
$C_M = Concurrency\ Miss\ Rate$

We are give the following values in our table that

$H_C = 5ns$
$C_H = 3$
$pMR = 35\%$
$pAMP = 1000ns$
$C_M = 5$

So we just need to plug all these values into our formula.

$$C\text{-}AMAT = \frac{5}{3} + \frac{35}{100} \cdot \frac{1000}{5} = 1.\overline{66} + 70 = \boxed{71.\overline{66}\ ns}$$

Hence the current average access time for memory word in this system is $71.\overline{66}\ ns$.

□

Exercise 5. Derive the recursive formula for calculating the current average access time (C-AMAT) for a word in a system with three levels of cache. The following values for the

theoretical system is in the homework document. Determine the current average access time for a memory word in the described system.

Proof. Utilizing essentially the same variables from the last problem, we can derive our formula except this time we will have to do this recursively for the different cache levels. Hence we get the equations below.

$$C\text{-}AMAT_1 = \frac{H_{C1}}{C_{H1}} + pMR_1 \cdot K_1 \cdot C\text{-}AMAT_2$$
$$C\text{-}AMAT_2 = \frac{H_{C2}}{C_{M2}} + pMR_2 \cdot \frac{pAMP_2}{C_{M2}}$$

The value K_1 is given to us in the chart. I will not label all the data here but they are found in the graph.

$$C\text{-}AMAT_1 = \frac{H_{C1}}{C_{H1}} + pMR_1 \cdot K_1 \cdot [\frac{H_{C2}}{C_{H2}} + pMR_2 \cdot K_2 \cdot C\text{-}AMAT_3]$$
$$= \frac{H_{C1}}{C_{H1}} + pMR_1 \cdot K_1 \cdot [\frac{H_{C2}}{C_{H2}} + pMR_2 \cdot K_2 \cdot [\frac{H_{C3}}{C_{H3}} + pMR_3 \cdot K_3]]$$

From here we just plug in our values from the chart in their respective variables.

$$\frac{5}{2.75} + \frac{45}{100} \cdot 1.5 \cdot [\frac{10}{3.25} + \frac{30}{100} \cdot 2.25 \cdot [\frac{35}{4.5} + \frac{15}{100} \cdot 4 \cdot \frac{100}{6}]] \approx \boxed{11.995\ ns}$$

After calculation we get an average access time for memory in the system to be approximate 11.995 ns

□

Exercise 6. How to optimize the following code to reduce the miss rate of a memory system? Please give the optimized code and explain why briefly.

Proof. To optimize the code that was given in the homework to reduce the miss rate we could merge the loops together in a single loop. The advantage of this is that two loops that have an identical index can just become one loop together. This also reducing loop overhead and gives us the opportunity to reuse. The optimized code is below

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

As you can see instead of writing 6 lines of code, we have reduced it to just 3. Hence this will also alleviate work on the programmer in addition to optimization.

□

Exercise 7. What are the hardware technologies which can increase Hit Concurrency in the formula of C-AMAT and explain why briefly?

Proof. Some hardware technologies that can increase hit concurrency in the formula of C-AMAT are non-blocking cache, multiple issue pipe-lining, and multi-core technologies. With non-blocking cache, which increases cache miss concurrency, in multiple issue pipe-lining the cache hit and cache miss concurrency both increase, and lastly the multi-core technologies can also increase the cache hit and cache miss concurrency.

□

Exercise 8. What is the Non-block cache and what are its characteristics?

Proof. Non-blocking cache is a technique essentially that hides memory latency. It does so by exploiting the overlap processor computation with data accesses. The exploitation of non-blocking cache allows execution to continue concurrently even with cache misses as long as dependencies are acknowledged. While this is happening, it successfully exploits post-miss operations, meaning we can keep giving cache hits even during a miss. This technique can also lower the the miss penalty by overlapping multiple misses, now this comes at a cost of increasing the complexity of the cache controller due to the fact that there could be multiple outstanding memory accesses. Another characteristic of non-blocking cache is that when we have a hit under miss, this further reduces the miss penalty by actually being helpful instead of ignoring the processor's requests. In essence, non-blocking cache is a helpful technique optimizes performances of caches.

□

Exercise 9. What is memory stall time? What is the advantage of the Layered Performance Matching method? Why LPM can improve memory performance (in terms of memory stall time) by more than one hundred times?

Proof. Memory stall time means that the whole processors wastes one or more cycles, so the time that is consumed in the stalling in a pipeline is a memory stall time. The advantage of layered performance matching methodology is that the performance of each layer of a memory hierarchy is optimized to closely match the request of the layer above it. This method essentially considers both data access concurrency and the locality. In essence, it showcases the effectiveness of overlapping hits and misses of layers that are higher to alleviate the performance impact on the lower layers. This ultimately leads to the layered performance matching methodology improving memory performance by more than one hundred times due to the overlapping.

□

Exercise 10. What is Layered Performance Matching (LPM) method? What is the advantage of the LPM method? Why LPM can improve memory performance (in terms of memory stall time) by more than one hundred times?

Proof. This is the essentially the same question as before, so I will just copy and paste my answer from there as I have already answer what is layered performance matching, it's

advantages, and why it improves memory performance by more than one hundred times. The advantage of layered performance matching methodology is that the performance of each layer of a memory hierarchy is optimized to closely match the request of the layer above it. This method essentially considers both data access concurrency and the locality. In essence, it showcases the effectiveness of overlapping hits and misses of layers that are higher to alleviate the performance impact on the lower layers. This ultimately leads to the layered performance matching methodology improving memory performance by more than one hundred times due to the overlapping.

□

Exercise 11. What is Pace-Matching Data Transfer? Can we achieve Pace-Matching Data Transfer?

Proof. Pace-Matching Data Transfer matches request/supply during data transfer effectively eliminating the memory wall impact. It is just like the heart bit or pace maker for blood transfer. This optimizes memory performance of the system, essentially the memory computing hierarchy is built based on the pace matching design. C-AMAT is used to calculate the ratio of data transfer from request and supply so it can layer the performance matching for each memory layer. We can absolutely achieve Pace-Matching Data Transfer, of course with assumption. The assumption we have to make though to make this work is that the application should have sufficient data concurrency and the system as a whole needs to be able to have the hardware capabilities for supporting data concurrency.

□

Exercise 12. Assume: CycleCPU = 2 ns, Cyclemem = 8 ns, fmem = 20%, IPCexe = 2.5, APC = 1; please calculate the LPM Ratio (LPMR).

Proof. We can take two approaches here one, either to increase performance or power consumption.

For both instances we will use the LPMR equation below.

$$LPMR = \frac{IPC_{exe} \cdot f_{mem}}{APC} = \frac{2.5 \cdot .2}{1} = 0.5$$

Hence the ratio is half, so knowing this we can see that in 8 ns, there is 1 memory cycle so the data supply rate for this is

$$APC \cdot N_{Cycle\ Memory} = 1$$

And then in 8 ns, there is 4 cpu cycles, so the data request rate is

$$APC \cdot N_{Cycle\ CPU} \cdot f_{mem} = 2$$

From here we can either improve memory concurrency by adding memory banks or ports to increase the data supply, hence we can increase APC from 1 to 2, so we would get this below

to match the data supply rate with the data request rate.

$APC \cdot N_{Cycle\ Memory} = 2$

Now you can also do it the other way where you decrease the CPU frequency to adjust for the data request rate. Hence we ould increase the CPU cucle from 2 to 4 so the $N_{Cycle\ CPU}$ would be 2, so we would also in this case get that the data supply rate matches with the data request rate.

$IPC_{exe} * N_{Cycle\ CPU} \cdot f_{mem} = 1$

Hence you can do it either way but the LPMR in this system is 0.5 either way without these adjustments.

□

Exercise 13. Give two non-locality-based methods which can reduce C-AMAT. Please explain why your methods works.

Proof. Two non-locality-based methods that reduce C-AMAT are Sluice Gate Theorem and Concurrency Match Theorem. In Sluice Gate Theorem, if a memory system can match an application's data access requirement for any matching parameter, then this memory system has removed the memory wall effect for this application which means it reduced C-AMAT and the memory wall problem all together in theory at least. In the concurrency Match Theorem, if an application meets the following condition at each memory later, then the LPM algorithm can find a performance matching for the application for any matching threshold. This means that we get high hit concurrency, small hit time, high pure miss concurrency, low pure miss ratio, and low pure miss penalty. Both of these theorem only work if certain assumptions are made, for instance in the sluice gate theorem the application has sufficient data concurrency, the system has sufficient hardware to support the data concurrency, and the architecture needs to elastic. In the concurrency math theorem has an issue of that LPM and C-AMAT does not tell you how to reduce C-AMAT at each layer. Either way, both of these are basically calculator's in C-AMAT. □

Exercise 14. The following Figure 1 shows a cycle-accurate C-AMAT example. In this example, there are 10 data accesses in L1 cache. Among these data accesses, data access 1, 3, 5 are hit data accesses, others are miss data accesses. As Figure 1 shows, the green cycles are hit cycles and the black cycles are miss cycles. (hint: please be careful about what is the difference of miss data access and pure miss data access). Please calculate the AMAT and C-AMAT value of this example in this Figure 1.

Proof. Using the equation we used before for C-AMAT we can also derive AMAT from the lectures. Due to time constraints I will effectively just used the formulas that were derived from before but the variables are the same as previous problems in this homework.

$$AMAT = H_C + MR + AMP = 1 + \frac{33}{53} \cdot 2 \approx \boxed{2.25}$$

$$\textit{C-AMAT} = \frac{H_C}{C_H} + pMR \cdot \frac{pAMP}{C_M} = \frac{1}{3} + \frac{16}{55} \cdot \frac{2}{4} \approx \boxed{0.48}$$ □

Exercise 15. In this following table, one set of AMAT and other parameters in L1 cache are given. Is it possible to use the given AMAT and other parameters to calculate C-AMAT in L1 cache? If it is possible, please calculate it and explain why?

Proof. Using the equation we used before from the previous problems again, we indeed can indeed use the given parameters to calculate C-AMAT in L1 cache.

$AMAT = H_C + MR \cdot AMP$
$50 = 3 + MR \cdot AMP$
$MP \cdot AMP = 47$

$$K = \frac{pMR}{MR} \cdot \frac{pAMP}{AMP} \cdot \frac{C_m}{C_M}$$

From previously we know that $MP \cdot AMP = 47$

$$0.3 = \frac{pMR \cdot \cdot 2}{47 \cdot C_M}$$
$$\frac{pMR \cdot pAMP}{C_M} = \frac{47 \cdot 0.3}{2}$$

Now we can calculate C-AMAT as shown below

$$\textit{C-AMAT} = \frac{H_C}{C_H} + \frac{pMR \cdot pAMP}{C_M}$$

Now we can just plug our values in to achieve C-AMAT.

$$\textit{C-AMAT} = \frac{3}{1.5} + \frac{47 \cdot 0.3}{2} = \boxed{9.05}$$

So we get taht C-AMAT is 9.05 hecne we were able to get C-AMAT given the parameters in L-1 cache.

□