Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 03
Lecture - 15

Lec 15: Six basic cache optimisations (1)

Hello everyone. We are doing module 2 on memory hierarchy. This is lecture number 4. In today's lecture, we are going to look at 6 basic cache optimizations. And if the cache is optimized, it will perform better and help the overall performance of the computer system. So, to address any optimization issue, we need to understand what are the different parameters which are in our hand to change, right?

So the performance of a memory hierarchy is defined among other factors mainly by the average memory access time. That is how long did you take to go and read the memory to bring the data to the processor. So when processor has a request, that request of a read or a write has to be satisfied by the memory hierarchy. If it is found in the cache, good enough.

If it is not found in the cache, it has to go to the next level cache or to the main memory to bring the data. Okay. So the performance or rather when the program runs, you will have several accesses, some may be found in the cache, some may not be found in the cache. So we average all of this out and then declare the average memory access time for a given program and the short form for this is called AMAT. So AMAT is defined by the given formula on the screen. It is the hit time.

So whenever we find the data into the cache, so the time to go to the cache and find out if it is there, so that is the hit time and if you do not find the data, then we have to go to the next level. So in case of misses, we use the miss rate and then followed by the miss penalty to handle these many misses. Okay, so AMAT is hit time plus miss rate into miss penalty. So that is the formula for average memory access time and this formula is giving us a framework to optimize the cache performance. So we have these three parameters, hit time, miss rate and miss penalty and very obviously if you look at the formula, if I reduce the hit time, I reduce the miss rate, I reduce the miss penalty, overall the AMAT will reduce.

Now how do you understand each of these parameters because once we understand the components of each of these three terms, we will be able to optimize on them. Okay, so

in this slide we are going to see how that is. So first we will see the hit time component. This is a cache, the blue table here. You can see the cache index, it has got eight entries, the tag and the data.

So when we come with an address to this cache and before we declare a hit or a miss, that whole time spent is called the hit time. Right? So we come from the processor with an address, it can go directly to the cache or go through some address translation and then it enters the address decoder. So it has to go to the decoder and identify which index it matches to and once it matches to a given index, then we have to bring out the tag of that entry, then do a tag comparison. After the tag comparison, if it is a set associative cache, you will also encounter the mux delay to select which of the four or five ways, four or eight ways to select from and then finally transfer the data. So this all adds up to the hit time of the cache.

So if there is a hit in the cache, this is the amount of time I am going to spend. If there is a miss, then we need to spend more time. Now how do you calculate the miss rate? We had seen a formula earlier. Miss rate is the percentage of accesses which miss in the cache. So this is the number of misses divided by the total accesses. Okay.

So miss rate is the percentage of the number of misses divided by the total accesses. Okay. Then whenever there is a miss, if we, if there is no hit in the cache, then you need to spend time going to the next level of the memory to bring the data. So miss penalty first come to the cache, search in the cache. If it is a hit, give the data. If it is a miss, then you have to go all the way to the next level.

And I would draw a longish line here because there is lot of delay between the next lower level memory and the cache. So this delays. So you first go, then you have to read the data there, then come back, write the data inside this. So that is called loading the cache. Right? So you load or allocate that block inside the cache and then send the data to the processor.

So whenever you encounter a miss, first you have to go to the next level of memory, bring the block and store the block in the cache and then transfer the required contents to the processor. Again, when you store the data into the cache, you need to see that the cache has space or it does not have the space and accordingly run a particular method to establish this. Okay. So there is lots of work happening to cater to a miss. So overall, the foundation of improving the cache performance is the AMAT and we want to, we want to concentrate on reducing the miss rate, reducing the miss penalty and also reducing the hit time of the cache. Okay. So what are the options available to us? This slide is just summarizing the information, but we will see each of these in detail.

To reduce the miss rate, we have got three options. One is you can use a larger block, you can use a larger cache or a larger associativity. Three options for the miss rate. For miss penalty, we have two options, using multilevel caches and giving priority to reads and for reducing the hit time, we wish to avoid address translation while indexing the cache. So we will see all the details during this lecture.

Okay. As our objective in the first one, we are going to see this. Here our objective is to reduce the miss rate. So let us see the first let us see what are the causes of misses. Misses are normally called the 3Cs model. Why 3Cs? Because each miss starts with the alphabet C.

So if I classify the misses, the first type of a miss is called compulsory miss. As the word says compulsory, no matter whatever is the situation, this is this miss is going to happen. So any first time access to a block is always going to miss into the cache because it is not there in the cache. You have to go to the next level and bring the data into the cache. So this type of a miss is also called all those accesses which will miss in a cache of infinite capacity. Right.

So even if you have a large enough cache, the first access is always going to miss. So these are compulsory misses. And the second type of a miss is capacity miss. Capacity miss as the word says, your cache has got limited capacity. Suppose you can store 8 blocks and you want to access 16 blocks.

So the newer blocks will obviously not have space in the cache. So because of this, you will, when you bring a new block, the old block has to move out and that causes capacity misses because you cannot contain all the needed blocks into the cache. Okay, so here if we are using a direct map cache, suppose I have a direct map cache and this is the address and a new value E wishes to come and sit where A is sitting, what would happen? This A will get removed and E can sit here, whereas there was empty location in other places. So I would not call this as a capacity miss because the cache has 3 empty locations, but still A had to go out. So capacity miss will be defined only when there is no free space left in the cache.

And therefore, we categorize or we label them as those misses which will also occur in a fully associative cache. So if this box was a fully associative cache, then the E could sit anywhere here and would not result into a miss. So capacity misses occur because of smaller cache capacity and they would also occur in a fully associative cache. The third type of a miss is called conflict miss. Now conflict misses occur in set associative or direct map cache.

As we saw here, when the new E wanted to come, A had to be removed because it was a direct map cache. The same situation can occur in a set associative cache where, suppose this is my set 0 which has got two options, so A and B is sitting here and now E wants to come and sit somewhere here inside the set. So even if there are empty locations in set 1, I am not permitted to sit anywhere in set 1 because the block is targeted to S0. So when E comes, A or B will have to move out and such type of misses are called conflict misses. They occur in a set associative or a direct map cache.

They are also termed as collision misses because the blocks kind of collide with each other and move each other out. They are misses in any n-way set associative cache, so that is conflict miss. So this is the 3 C's model, the compulsory miss, the capacity miss and the conflict miss. But recently we have a fourth C coming into picture which is called the coherence misses and in this subject we are going to encounter this more and more because these happen due to the multiple processors running multi-threaded shared programs. So when programs share data, a program would have cached a data item in its local cache and the other program wants to access the same data item.

So when the other process or other thread wants this data item, it has to be removed from this thread and given there. So we are causing more number of misses because of this coherence maintenance issue. Okay. So that is the fourth C, that is the coherence miss and cause because multiple caches need to be kept coherent in a multi-processor environment. So we have seen these four causes of misses. Now we will move to the optimizations. Okay.

So the first optimization to reduce the miss rate is use a larger block size. So very intuitive, to reduce miss rate meaning all accesses which are going, I should get as many hits as possible. So to get more hits in a cache, your block should be big enough because if your block is big because of spatial locality, we would have brought in more data already and there are high chances that the newer data items would be found in the same block. Okay. So I can have a larger block to reduce miss rate. So this is clear, but is this effective? So now you imagine you have to go to the next level and bring a larger block, what would it incur? It would increase the miss penalty because you have to go to the next level and bring more amount of data rather than only one data item which you have missed upon. Right.

So this increases the miss penalty. It also increases the conflict misses. Now how does it increase conflict misses? Conflict misses occur when you are in a direct map or a set associative cache. So given that we had a cache of a fixed capacity, so I will just proportionally draw this. So this was the cache capacity and for smaller blocks I could fit

these many blocks here, but if I increase the block size, I could fit very few blocks. So I have almost double the block size and I can only fit four blocks here whereas maybe here I was able to fit eight blocks.

So when you have four blocks or lesser blocks and associativity, so you will increase conflict misses because new blocks will remove older blocks. So it increases conflict misses. Then there is no benefit in this option if it increases the AMAT because when your miss penalty increases, the average memory access time will also go up. So what do we do? The solution for this block size depends on how much is the bandwidth and latency available at the next level because the main target is controlling the miss penalty of this large block. Okay. So as miss penalty is increasing because I am bringing bigger blocks, I need to worry about how much effort I am spending in bringing this large block. Okay.

So if your next level is giving you high bandwidth at the cost of little high latency, so if you have high latency and high bandwidth at the next level, so if I draw it here, this is the cache and that is the next level. I will just call it memory directly suppose. So this is the link I am discussing about. So this link between the cache and the memory, if it has high bandwidth and it has higher latency, so this would encourage me to bring larger blocks. Why? Because of the high bandwidth in one go, I am able to fetch lots of data. Okay.

So with little more miss penalty or a little more latency, large amount of data can be brought in. So let us be greedy and bring more amount of data because we have good bandwidth. The other option is if your bandwidth is low, that you can only bring small amount of data in one trip, then better bring small blocks. Do not increase the block size. So we will bring smaller blocks if I have lower bandwidth. Right.

So suppose I was going to bring a larger block which had both data items A and B. To bring this, I will need two trips, one trip to bring A, the second trip to bring B. So is it required to make two trips and bring both of them? May not be. Because let me first bring A and then if B is required in future, let me bring B because the miss penalty of a small block. How much are you spending? Small block has got some miss penalty and I have to make two trips. So that is approximately equal to the miss penalty of bringing a block which is twice the size. Right.

So this is the statement here. So twice the miss penalty of a small block, that is I am going to spend two trips to bring the small blocks is approximately same as bringing a bigger block which is spending double the amount of sorry, time. So the overall time is

same. So let me bring a small block if required the second block can be brought. Okay. So that is about having larger blocks. Now having a larger cache, very intuitive, have as big cache as possible because after the compulsory misses you would not encounter many misses more often.

So obvious way to reduce the miss rate is have a larger cache. Now what is the drawback with this? Drawback is larger the cache, more the hit time, then more costly because the faster memories or the cache memories are expensive and also power hungry. So it is going to increase the hit time and also have more cost and power will also go up. So we do not want to do this immediately, but we can do this for lower level caches because lower level caches can afford to have a slightly higher hit time and therefore we can use this optimization for lower level cache. The third method of reducing the miss rate is using higher associativity.

So why higher associativity? Because in a direct map cache we just have a single option to place a block. Right? It will only sit here and nowhere else. So if already A is sitting when E wants to come, E will remove A and sit in the place of A. If I increase the associativity and I say that these two blocks form the one set, so even if A is sitting here, if I increase the associativity then E can also sit here because this is the same set. So in a direct map cache this is not possible, but in a set associative cache there are more chances that the block will get alternative options to fit into the cache.

This reduces miss rate. So with experimentation people have found that you can increase associativity. So the greedy approach will be why not have a fully associative cache because in this example I said I had a two-way set associative A was sitting when E comes it sits in another place. Suppose two more blocks come, so where will they sit? Right? So they would definitely increase the miss rate by removing A and E. So why not go for a four-way associative, why not go for an eight-way associative and hence a fully associative cache. But with a fully associative cache your search time increases because you have to scan through all those entries to see if the block is there and then even the replacement policies become more complicated and time consuming. Okay.

So fully associative cache is expensive in terms of hit time. Hence people have experimented and found that an eight-way set associative cache is as effective as a fully associative cache. So if we have eight-way we are good enough. So eight-way is as effective as a fully associative cache. And the second rule of thumb is that a 2:1 cache rule of thumb which says a direct map cache of size n has the same miss rate as a two-way set associative cache of size $n/2$. Okay.

So I will illustrate it here. So the purple color is a direct map cache with eight blocks.

So this is a direct map cache with n equal to 8. I have eight blocks kept here and this is a two-way set associative cache, two-way set associative cache where number of blocks is equal to 4. So this is the set, this is block number 1, 2, block 3 and block 4 and these four blocks are divided across two sets. So this is n versus n by 2, so 8 versus 4 blocks and direct map versus two-way set associative.

So if we double the associativity then we can half the cache size and this both designs will give me the same miss rate. If you see the addresses mapping, the first set is mapping address 0 and 2, the second one is mapping 1 and 3. So when 4 and 6 comes, so 4 will also be housed here, 6 will be housed here and so on. And there are, experimentally people have found that this arrangement is giving almost similar miss rate as this arrangement.

Now is this valid forever? Definitely not. This property is known to be valid for smaller caches that is only up to 128 kilobytes of cache. So if you have 128 kilobyte cache sizes then we can exploit this property beyond that it may or may not hold. Okay. So overall for the miss rate what have we seen? We have tried to increase the block size but we increase the block size, miss rate definitely reduces but the miss penalty increases. Then we tried increasing the associativity, associativity increases the hit time. Right?

So miss rate is improved but hit time increases. And if you have a faster processor, you have more gigahertz in your processor, it wants to access data quickly and to access data quickly your hit time should be very small. Hence the cache design should be simple, it should not spend more time searching for the block. So for faster clocks you need a simple cache but at the same time to control the miss penalty you have to manage the associativity at the same time. Okay. So to summarize let us see the effect of the 3Cs. One is that the compulsory misses were reduced by having a larger block but having large blocks does not give benefit forever, it is less effective.

To reduce the capacity misses we have gone for larger cache sizes and for reducing conflict misses we can increase the associativity. Okay. So now what happens if you change the cache size? So when the cache size increases or for the same cache if you increase the block size the type of misses move from one type to the other. So changes in the capacity will also have effect on the conflict misses and so on. So a capacity miss gets translated to a conflict miss and vice versa depending on what you are doing. So maybe capacity misses reduce but conflict miss increase and something similar impact will also happen.

So these three optimizations are very much interlinked. So techniques that reduce miss

rate, increase the hit time or increase the miss penalty. So if I am saying the miss rate is reducing it is indirectly increasing the hit time. You take more time to hit and more time to fetch the block. So the miss penalty also increases.

So just optimizing one is not sufficient. We need to seek a balance among all the possibilities to achieve a best AMAT. So with this we finished three optimizations. In the next lecture we will see the remaining three optimizations. Thank you.