Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 03
Lecture - 13

Lec 13: Memory hierarchy questions (1)

Hello everyone. We are doing module 2, memory hierarchy. This is lecture number 2. Today, we are going to look at four memory hierarchy questions across two small lectures. In this lecture, we are going to address the first two questions. So what are these four memory hierarchy related questions? The first one is block placement, second is identification, third replacement and write strategy.

Okay. So in the previous lecture, we have seen a thorough introduction to what a cache is. Essentially, it is a small storage and it is able to get blocks from the next level of cache or next level of the memory hierarchy. And these blocks are expected to have good enough locality so that in time as well as in spatial locality, we can keep on accessing them.

And when the block is not required, we can move it back to the next level and bring the next block. All right. So now that we have a small cache backed up by a bigger enough main memory, when I access an address in the main memory, this address will come and sit in which location of the cache is the first question. So when I bring a block from the main memory, where to place it in the upper level? Upper level is either first level cache, second level cache. So upper level is bringing it from a next level to this level.

Okay. So when I bring the block, what are the options? We have three options. One is be very strict and keep it in only one location. The other option is be very lenient, keep it anywhere or give some limited choices. Okay, so let us illustrate this.

Right. So this slide is showing a picture of a classroom. This is your if you remember your school or a college classroom, and I am saying it is exam time. So when it is an exam time, you enter the classroom, where do you sit? You are given a particular desk and table on which your roll number is written. So you will go and sit exactly at that particular roll number.

So if this is your roll number, then you will go and sit in that table because you are not permitted to sit anywhere else. So this is called only one choice. During the exam, you

have exactly one choice as to where you will sit. Okay. I will take another example.

The same classroom, but it is not exam time, it is fun time, that is you have a break period or something, so you have a free class, so it is a fun time. So if you come into the class during a fun time, where will you sit? You will not sit here, may not sit here, actually you are permitted to sit anywhere because there is nobody to monitor and there are no rules. So you can have fun in the class, so you can sit anywhere in the class. And the third one, so this I can say that suppose this is a lab assignment to be done and you have to do it in groups. Okay. So you have been given a lab assignment to be done in groups and it is expected that the group members sit together.

It does not matter where each member sits. If this is group 1, the two members of group 1, member 1 and member 2, they can sit anywhere, either this way or M2 can sit on this chair and M1 can sit on this chair. Right. So the two chairs can be occupied in any order by those two group members. So if I say this is the scenario, how many choices do you have? We have limited choices. Right? or restricted choice. We are not saying I have a very strict single choice.

So you do not have a single choice, you have multiple choices, but you do not even have a full freedom to sit anywhere. So continuing that analogy to the cache, we will see these three options in the cache. The first option is where I have a single choice, that type of a cache is called a direct map cache. How do we identify the location of a block in this? We take the block address and mod it with the number of blocks in the cache. So this way you will identify where the incoming block should sit in a direct map cache.

The second choice is we can sit anywhere in the classroom during a fun time. So this is called a fully associative cache because a block can sit anywhere throughout the whole cache. Right? It is permitted to sit anywhere. The third one is where we have a limited choice as in our lab assignment, we are permitted to sit on any chair, but along our group table. So you cannot sit in the chair of another group, but you can sit with your group members in any order.

So this is the set associative arrangement. So this restricted combination is called set associative. How do we find where to sit? Take the block number and mod it with the number of sets. Now what is this set? That is the number of groups. For example, if you were 20 students in the class, you had 5 groups, each group will have 4 students and you will modulo this with 5 that is the total number of groups.

In direct map, we mod it with the number of blocks. In set associative, we modulo it with the number of sets. So to summarize, I can definitely call a direct map cache as a

one way set associative. Set associative gives us freedom to sit anywhere and direct map cache gives me freedom to sit only in one position. So I will say direct map cache is one way set associative cache.

And similarly, a fully associative cache can also be called an M-way set associative cache and it gives you permission to sit anywhere. So this implies that my cache has only M ways, so M, sorry M blocks. So my cache has got M blocks and I can sit in any of these M blocks, hence a fully associative cache is a M-way set associative cache. Okay. So if I have a direct map cache with 4 entries, we can identify which address sits here by doing a modulo 4. So any address coming from the main memory will go through this pattern.

If I have fully associative cache, then we do not need to worry, we can sit in any position, any of these blocks. So there is no modulo to be done in this case, whereas here it is either here or it is here. So it cannot be anywhere, it is a fixed location. And in a set associative cache, I will take a little bigger one this time because 4, I will take suppose 8 entries were there and these 8 entries are to be grouped now. Right? You remember you have to form a group in your lab assignment.

So I will create this as first group. Right? This is my group 1, the next 2 blocks will form the group 2, group 3 and group 4. Right? So we have formed 4 groups, each group having 2 blocks. And therefore, even here we are going to do, in this case there are 4 groups, so we will do modulo 4, however the number of blocks in this example I have taken as 8, whereas in the first example it was 4. So here the blocks were 4.

In case we had 8 blocks in the direct map example, we would do modulo 8. Okay. So these groups which we have formed are called sets. So this is the first set, second set, third and fourth. We have 4 sets in this and using the convention we will say this is set 0, this is set 1, set 2 and this is set 3. Okay. So when you do mod 4, you will identify one of these numbers 0, 1, 2 and 3 and we can position the block in that particular set. Okay.

So quick example, cache has 8 blocks and now the question is where does address number 12 sit? We are discussing placement. So where does address number 12 sit? So you can pause the video, solve it and then check the answer. Okay. So if it is a direct map cache, we have 8 blocks in the cache. So we take the address and mod it with the number of blocks.

So you get the answer as 4. So you will be placing this block in placing address 12 in block number 4. In a fully associative cache, you can place it anywhere. There is no formula to find out where to sit and in a two-way set associative cache. Now this

two-way says that every set has got two options. If it is a four-way associative, then every set will have four options.

Because every set has got two options, how many sets are there in the cache? You have 8 blocks and 2 in every set. So we will do 8 divided by 2. So number of sets first you need to find out. So the number of sets is number of blocks divided by the associativity.

So this gives you 4. So we have total 4 sets and therefore, for this I will do 12 modulo 4 and not modulo 8. So how much does this give us? So this says it gives us the answer of 0 and this is set number 0 and not block number 0. So this answer says I have to go to set 0 and within that set there are two blocks so you can place it in any of them. Okay, moving on to the next question. Next question is how is a block found in the upper level? And definitely we will do some comparison.

We have seen an analysis in the first lecture about division of an address into various fields. So this can be considered a recap. To divide the address into an offset, it can be a byte offset or a word level offset. Then after that we use an index field to identify the block number, which block to read. So once we reach the block, within the block we know using the byte offset or the block offset what to read.

And once we reach to that index, how do I identify that the address I am looking for is the same address because multiple blocks will be mapping to that same location. So for this we have to use the remaining part of the address, which is called the tag. So I am going to use the tag to do the comparison. Okay. So here I have divided the address into a byte offset, an index and a tag.

And I am going to use this to compare. So suppose, I will again take an example with 8 blocks and with 8 blocks my index will be a 3 bit index starting from three 0s to three 1s. Okay. So using the index that is the pink portion of the address, we will come to one of these locations. Using the yellow byte offset, we will read say a particular byte which we want to read. And before we read that we should make sure that the label associated with my address is which I am interested in. So you have to compare the tag that is the remaining address. Okay.

So index and byte offset are taken care of to reach this particular location and whether to read this or not is identified by the tag portion. Okay. Now I will ask you a question. This is a direct map cache. Assuming this is a direct map cache. If this becomes a two-way set associative cache, then will the index portion increase or decrease or indirectly will the number of tag bits increase or decrease? You can pause the video and think for yourself first before seeing the answer. Okay.

So I am saying it is a two-way set associative. So these are the sets. These are four sets and now that it has become a two-way set associative, my search space reduces from eight blocks to four sets. So because the search space has reduced, what has reduced? Tag or the index? The index, because index was initially three. Now I have to look out for which of the four instead of which of the eight. Okay.

So here I was looking out for which of those eight locations. Now I have to look out for which of these four sets. So this is going to shift. Right? So my index becomes smaller and my tag becomes larger. And what would happen in the case of a fully associative cache? The pink portion completely vanishes and you get this. Right?

So as the associativity increases, the blocks per set will increase. So index size will decrease and tag size will increase and a fully associative cache has got no index field. Okay. So just for summary, I have put some formulae here and a quick example. If you are given a cache of C bytes capacity, blocks are of capital B bytes, then you need to calculate the tag, index and byte offset for an address with A bits. Okay. So using these alphabets, you can identify the formula.

So this is the formula. We also discussed this in the previous lecture as a quick summary, it is written here. You have to first find out the number of blocks in the cache by doing C divided by B. Once you identify this, you find out the size of the index using $log(C/B)$. Knowing the number of bytes, you already know the byte select or the byte offset which is $log\, B$ and that is what you do here. You take the address bits and subtract, this is index I.

So $A - I - log(B)$ where this I is log of C by B. So these are the formulae and with that I have also kept one example. Okay. So using this if I ask what all addresses will sit in block 0. Right? This is the range of addresses which will be mapped to block 0 and to find whether a block is present or not, you have to use the cache, you have to use the tag field.

If it matches, we treat it as a hit. If it does not match, we treat it as a miss. Okay. So going back to the small picture, the bottom two are set associative, this one being fully associative and this one being the set associative. Okay. Set associative caches, be it fully or n-way set associative, it gives us freedom to place a block anywhere within a given set. But this advantage comes with a problem that when we go to that set, we have to do further search.

We cannot directly go and read that block. Right? So you need to identify which of

these two blocks in this example, if it is a four-way set associative, which of the four blocks to read. So we need to handle this and see how it has to be done. Okay. So an n-way set associative cache is there, then every set will have n entries. You need to check all of these n entries to find out if your tag matches with these n entries. Imagine if we do it in a sequential manner, you will first compare the first entry, then the second entry and so on.

So you will take longer time to do the comparison. How do we do it faster? Go parallel and compare the tags of all the n entries in this set in one go. And typically the associativities are two-way, are very tiny for tiny devices, but 4, 8 or 16 way associative are normal number of blocks in a set. So you will have to do 16 comparisons for example, and it is going to take time. So we will do all of them in parallel. Now how to do it in parallel? This is an example of a two-way set associative where you can see the color coding is telling you where the blocks will go. Right?

So this and these two blocks will be sitting here and which of them is sitting here, I need to check using the tag. So I have to check whether it is block 10 or whether it is block 20. I do it sequentially or I check is it 10? is it 20? and I do this in parallel. For two-way it may be faster to do it even sequentially, but as the associativity increases, you will have a lot of delay.

So I will explain that with a four-way set associative cache. So four-way set associative, I am drawing a set here. So this is a particular set. It has got four entries and I have to compare the tags of each of these entries. We do not want to do it sequentially, we want to do it in parallel. So this is, this picture is showing you the layout where the first rectangle is showing the first way.

So when I say there are four entries, this is way 0, way 1, way 2 and way 3. Okay. Set has got ways, four-way set associative has got four ways. So way 0 is represented by this set of entries and so on and this is way 3. So we logically divide the cache way wise, that is way 0 of every set is sitting together, way 1 of every set is sitting together. Right?

This is the index which is the set ID. So when it is index 0, it means it is set 0. So set 0, this rectangle is set 0, way 0 is this entry. This entry is set set 1, way 0 and so on. This one is set 255, way 0.

And for set 255, the way 3 is sitting here. So this is set number 255's, way number 3. And the first line is set 0's, way number 3. I hope it is clear. We logically partition the cache like this. So why is this done? So that we can do a parallel comparison across any required set.

For example, I am interested in set number 4 or 3, this is set number 3. In set 3, I want to do the comparison and in parallel, you have to compare way 0, way 1, 2 and 3. So take the tag of way 0, take the tag of way 1, 2 and 3, bring them down and do the comparison in parallel. So we are doing the comparison at the same time.

So this is saving us lot of latency. Once we do this, we need to check whether it is valid. So this AND gate is used to check for validity, whether it is valid. And once it is valid, you have to declare a hit or a miss. And if it is matching with any one of this, definitely it will match with maximum one of them.

It cannot match with multiple entries. It will match with either one or none. So if it matches with one, this OR gate will be turned on to one and it will be declared as hit, otherwise it is a cache miss. Once we know that it is a cache hit, we need to then read the block from that particular column. Right? So first you need to know which of these ways has given you the hit and then go and read the data.

So the data lines are already coming here as input to a multiplexer. So I am using a MUX and the outputs of these comparators are acting as the select lines of the MUX which will identify which data to pick up and send it out. So this is how we will do the read of the data will happen. Okay. So as you increase the associativity, what is happening? Your number of comparators are increasing. Additionally, you also have to wait for the hit or miss decision before you can read the data because the data you do not know which of these n entries to read. Once you know which way to read, then you go and then the MUX selects which data to be sent out. Okay.

So if we draw a comparison between a n-way set associative and a direct map cache, the n-way set associative cache has slight disadvantage that it has got more comparators, it has got n comparators whereas a direct map cache has got only one comparator. It does not need because there is just a single way in it. We need extra MUX delay for this and the data can be read only after the hit or miss decision. So we cannot make any optimizations in associative cache.

You have to first decide a hit or miss and then go and read the data. Whereas in a direct map cache, you can start reading the data because data read is going to take time, compare the hit miss and in parallel start reading the data. What advantage do you get? If it happens to be a hit, whatever data you have read in parallel, simply start using it. If it happens to be a miss, just discard the data you have read because we are doing this optimization for saving on the time. Right? So data can be available before a hit or a miss. We can assume that it is a hit and then continue to read read the data and if it is a

miss, then simply discard the read data.

So this is direct map cache versus associative cache. Okay. So with this we have finished answering the first two questions. Thank you.