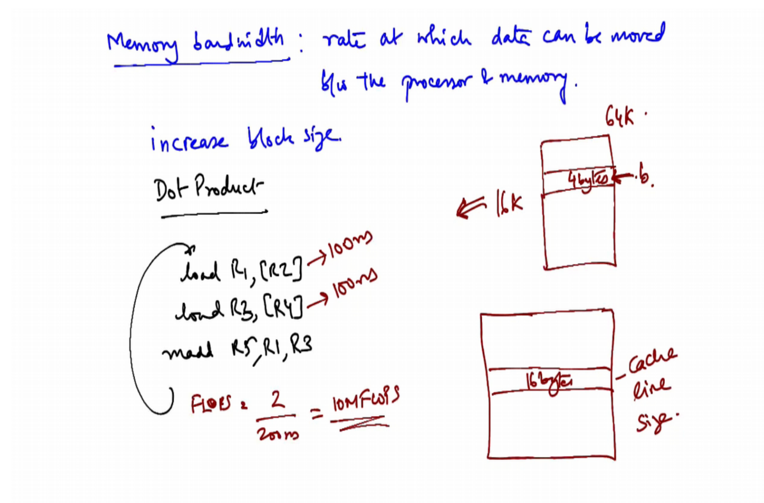
Introduction to Parallel Programming in OpenMP
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 07
Cache, Memory bandwidth and Spatial Locality

Now, let us move to another issue with the memory right. One issue is that when I want to access some memory location, from the time that I request that access to the time that the memory comes back to me with that data that there is a lot of time that has elapsed that is the latency, right that that is a problem.

(Refer Slide Time: 00:26)



But again you could do something similar to what we did with pipelining right. So, even though the latency is high, I could have more data coming back at periodic intervals. So, what is memory bandwidth? So, this is essentially the rate at which data can be moved between the processor and the memory.

So, note that this is not talking about the time it takes for the there to come back, this is talking about the total overall rate at which things can be done. So, for instance how long does it take to execute one instruction it might take 5 nanoseconds or 10 nanoseconds or 15 nanoseconds, but the through put that I get, if I am doing pipelining is one instruction being executed or completed every nanosecond every cycle right because I am pipelining. So, this is also similar. So, let us look at some aspects of memory bandwidth.

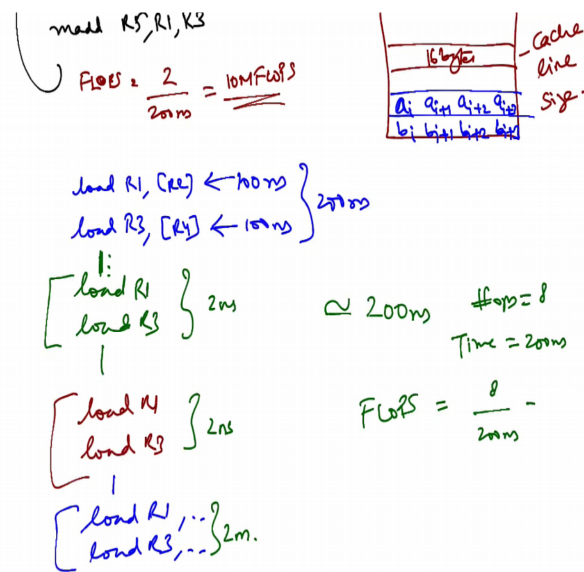
So, how do you increase the memory bandwidth? So, one simple way of increasing the memory bandwidth is you increase the block size.

So, earlier we were talking about a cache, which was storing 4 bytes. So, 64 k cache was actually 16 k entries each of 4 bytes right and this was essentially the b which was the word size of the size being written by the memory. Now what I could do is that I could increase the cache size. So, let us say that instead of 4 bytes I start maintaining 16 byte cache entries. So, this is something called the cache line size how big my cache line is right. So, let us take a simple example we will we will take the example of a dot product code not matrix multiply, but let us look at dot product right.

And let us see how it would perform under these two scenarios. So, what would the dot product code look like?? It would essentially look the same as that innermost loop of matrix multiplies, right that essentially nothing, but dot product right. So, it would again look exactly the same load R 1 from an address load R 2 from sorry R 3 from another address and then finally, add into R 5 the product of R 1 and R 3 right and then I repeat this again. Now I keep on repeating this first. Let us look at this case where I have 4 byte cache line size, what is the performance I am going to get in terms of flops? There is no cache reuse this is dot product this is not matrix multiply. So, every time I get the data is going to have to come from memory, right.

So, this is going to take 100 nanoseconds, this is going to take another 100 nanoseconds and then a few mores nanoseconds for the multiply add and some address increments and branching and so on. So, what is the flops? We have already computed this actually I am performing two operations in 200 nanoseconds. So, I get a rate of 10 mega flops. Let us see how this changes if I have 16 byte cache lines. So, now what happens is that when I do a load R 1 R 2 right what comes into the cache.

(Refer Slide Time: 04:12)



So, what is being multiplied aik and dkj well this is dot product.

So, let me just assume ai times bi right. So, I am loading ai this is going to take 100 nanoseconds to execute, but what is coming into the cache at this time not just ai, but ai ai plus 1, ai plus 2 and ai plus 3 all of them are in the cache and when I execute this load R 3, this is again I am going to take 100 nanoseconds, but what do I get in the cache I get bi, bi plus 1, bi plus 2, bi plus 3.

So, this is roughly taken me about 200 nanoseconds, but what about the next set of loads in the next iterations after a couple of more statements, I am going to have the next iteration right which is again going to have two loads, and then I will have another iteration and then another iteration right how long is it going to take for these instructions to execute? Operant fetches just going to be a single cycle fetch right it is going to a come from the cache.

So, this is just going to take 2 nanoseconds, 2 nanosecond is just for the memory fetch part of it and then there are some other overheads, which I am going to ignore for the time being, but roughly what am I getting in about 200 nanoseconds plus a bit more which I am ignoring for the time being, I am performing how many operations.

Student: 8.

Eight operations 8 multiplied, right. So, number of operations. So, if I look at four iterations the number of operations performed is 8 and the time taken is about 200 nanoseconds.

So, what is the flops rate I am going to get?

Student: (Refer Time: 06:19).

Eight by 200 nanoseconds its 40 mega flops, when we do this performance analysis right instead of doing this performance analysis for like for instance I was looking at one iteration and then I had to look at 4 iterations to do this analysis, we typically do this analysis slightly differently.

(Refer Slide Time: 06:41)

We use something called a cache hit ratio. So, what is cache hit ratio? This is the percentage of memory accesses found in the cache hit, in the cache or served by the cache.

So, if I want to compute the cache hit ratio for this example that we just saw right, what is the cache hit ratio, how many exercise, what I making and what percentage of those were found in the cache.

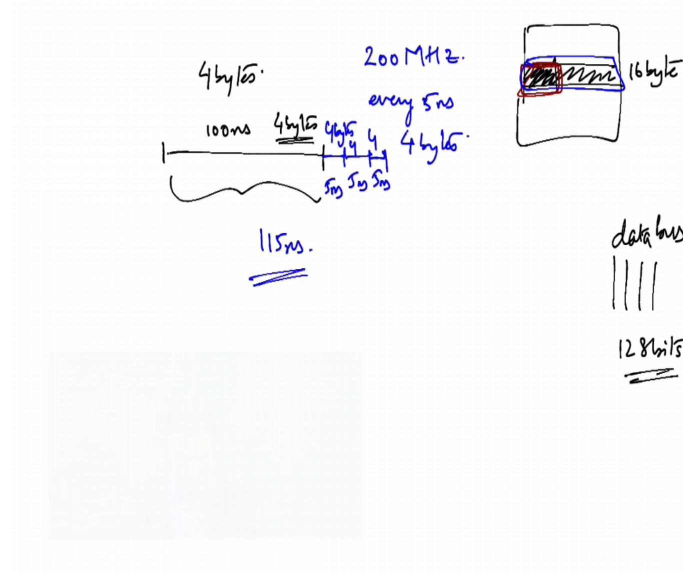
Student: (Refer Time: 07:21).

75 percent right the first iteration I had to get it from the main memory, but the next three iterations I was able to service from the cache. So, cache hit ratio is 75 percent. So, what; that means, is for all the memory access time instead of taking different memory access times, one nanosecond for 3 iteration 100 nanoseconds for one iteration so on. I can just work with the single quantity, the average memory access time and what is the average memory access time? That is nothing, but the cache hit ratio 0.75 into the time it takes to access the cache which is one nanosecond plus 1 minus the cache hit ratio that is the time it has to go to the main memory, and what is going to be a latency for that time 100 nanoseconds.

And so, this turns out to be 25.75 nanoseconds. So, if I just work with the fact that each memory access is taking 25.75 nanoseconds I will get the same rate. So, let us says quickly see that. So, what was happening over here? So, now, we only need to look at one iteration, I do not need to go across four iteration because I am taking the average time right. So, if I look at one iteration; what is the number of operations being performed to one multiplied right and what is the time taken? There are two accesses to the memory. So, the time taken is 51.5 nanoseconds, let me just approximate that by 50.

So, what is the flops 2 by 50 nanoseconds, which is what 40 mega flops its same as 8 by 200 right it same as 8 by 200 again I mean we were using a lot of approximation, but (Refer Time: 09:11) yeah because the kind of improvements, we get are you know orders of magnitudes. So, it is to take some approximations here. So, instead of 4 byte cache line size now we said we will have 16 byte cache line size.

(Refer Slide Time: 09:23)



So, how do you get a 16 byte cache line size? How do you load the data into the cache? How do you load 16 bytes into the cache?

So, one option is that you increase the size of your data bus right make your data bus broader. So, earlier the word size of 4 bytes I was fetching 32 bits at a time, now what is it going to be if I want to fill cache line sizes of 16 bytes, what is it going to being one 28 bits, but that is not practical it is not practical to build such large data buses or they are costly right and the take up space, but you want a cache line size of 16 bytes right below we just saw how it helped us. So, we want large cache line sizes, but at the same time I do not want to increase the size of the data bus what do you do in the architecture?

So, you essentially do some kind of pipeline right for data bus size will still remain to be 4 bytes. So, what happens is when you make a memory access, it takes about 100 nanoseconds to get the first 4 bytes into the cache right? So, that is just a part of the cache. So, we filled up this part of the cache in the first 100 nanoseconds. Let us assume that we are working with a memory unit that has an operating frequency of 200 megahertz. So, the operating frequency of memory is not the same as the processes it is considerably slower.

So, we will assume that were working with the 200 megahertz memory unit. So, how much data can this provides. So, it is going to provide you a word, the word size is going

to fill up the data bus. So, every 5 nanoseconds right 200 megahertz 5 nanoseconds. So, every 5 nanoseconds it can fill up the data bus again it will give you 4 bytes.

So, what will happen over here is that the first 4 bytes will take 100 nanoseconds to come, then in this next 5 nanoseconds you will get another 4 bytes, in the next 5 nanoseconds you will get another 4 bytes and in the next 5 nanoseconds you will get another 4 bytes. For a total of 16 bytes which fills up the cache line right. So, this will take about 115 nanoseconds to get the cache lines right; that is the way typically this works the larger cache line sizes are handled.