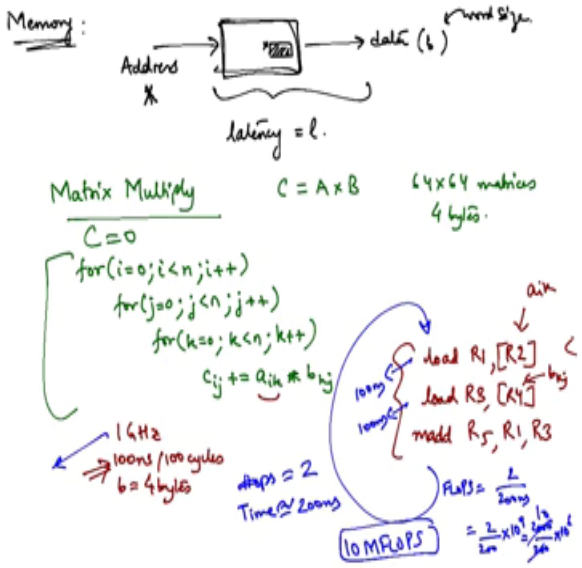
Introduction to Parallel Programming in OpenMp
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture - 05
Memory Latency

(Refer Slide Time: 00:01)



Let us look at some features of the memory which are you know commonly used to improve the performance of a single processor system. So, what is the basic function of the memory? The basic function is that we supply an address or we call it a location; I supply a an address X, it is just a location in this physical memory and what it has to do is it has to return me, the data that resides in that location.

So, for instance this could be location x. So, it has to return me this data and there is always the size to that data. So, it will return me a data of let us say b size b, typically we refer to this as the word size. So, what determines the performance of the memory? So, one important factor is the latency. So, this is the time it takes from when I give the address and ask for that data, to the time that it supplies me the data. I will just denote it by l. So, l is the latency.

The performance of a memory system basically depends on it is ability to feed data to the processor, because in a lot of applications we see that you know the processors you have fast processors, but the kind of code that we are working on the memory unit is not able

to supply data to it faster. So, let us look at a small example. So, let us take a small example of matrix multiply. So, let us say we are trying to compute C is equal to A times B, and let us say that all of these are 64 cross 64 size matrices and each element is let us say 4 bytes .

So, what would be the simplest code that you would write for this? You have three loops and you say that cij is equal to plus equal to aik into bkj right, and before you start executing this code you have to initialize C to 0, hugging that will involve a couple of loops which I am not writing down, but you initialize C to 0 and then you execute this set of instructions. So, let us figure out how long this code is going to take or what kind of performance I can get out of this code ok.

So, for that let me first assume a few things. So, let me assume that I have a one gigahertz processor, and let me assume that the cost of accessing the memory is 100 nanoseconds or 100 cycles, and I am going to assume that my word size is 4 bytes right. So, whenever I ask for some data from some location, it is going to return me a 4 byte value. So, we want to figure out how long this code is going to take the execute rights. I will I will assume a very simple processor in order issue no super pipelining.

So, let me just concentrate on the innermost loop right that is the most important part which is going to determine the performance. So, what are the statements that are going to execute? First I have to load aik. So, I will say load aik into some register R 1 and how do I specify that I want to load location aik. So, I will store that location in come register. So, I will store the address of aik in some register let us say R 2, and then I am going to say load the location specified by R 2 into R 1 into register R 1. So, that is loading aik and then I load into let us say R 3 I load dkj and.

Finally what am I going to do? I am going to multiply these and add them into cij, and let us say that I am maintaining cij in some register right in some register R 5. So, I am loading into register R 1 from the address that is kept in register R 2, I am loading into R 3 from the address that is kept in register R 4 and then finally, what I am I doing I am multiplying R1 and R 3 and adding them to R 5.

So, we are assuming there is an instruction a madd at which does a multiplication and addition together. So, if latency is 100 nanoseconds, how long is this code going to take to execute or let us just look at one iteration; how long is it going to take to execute one

iteration. So, this instruction is going to take 100 nanoseconds, this instruction is going to take 100 nanoseconds, and then a few nanoseconds for a madd and then I am going to go back here I am going to loop back and I will have to increment aik, I will have to increment dkj and then I am going to execute this again, but incrementing and doing this a madd all of these do not involve any memory operations.

So, they are not so expensive. So, let me nor them for the time being, I mean it is not to ignore them, but for the time being just for the sake of simplicity let me just ignore them right. So, how many operations am I able to perform in how much time if I look at one iteration, how many operations am I performing. So, typically we gauge the performance of any code that we have the units we use is flops

That is floating point operations for a second right how many floating point operations can I perform every second? So, we want to calculate what is the flops that I am getting for this code right. So, what is the number of floating point operation, I am performing in one iteration? So, there are two operations multiply and add, look there is a single instruction, but that is a separate issue as far as then floating point operations are concerned there are two floating point of operations right one is multiply, one is that the fact that I can encode them using the same instruction is completely a separate issue right your architecture may or may not do that.

But when I want to calculate the flops I have to do it on any architecture. So, number of operations is to and what is the time it takes to perform one iteration? So, I am going to approximate this by 200 nanoseconds, there is a bit more that is taken by madd branching and incrementing of addresses and so on, but I will I am just ignoring that for now. So, the flops or the performance I am going to get is two operations every 200 nanoseconds. So, this is 2 by 200 into 10 to the power 9. So, this will be 2000 by 200 into 10 to the power 6, 10 into 10 to the power 6. So, this is 10 mega flops. So, that is the performance I am getting.

So, what is the process of frequency? 1 gigahertz when you look at a 1 gigahertz, processor you think you are going to perform 1 Giga operations in every seconds 1 Giga a floating point operations of a second right it can actually do that under certain conditions right not when you have to go to the memory every time to fetch your data. If your data is only in registers and you are always doing your computations with data and

registers, then you can actually achieve that right. There are other conditions also under which you can achieve that certain codes like matrix multiply and you know linear algebra codes get close to that, but a generic code does not get very close to it.

So, in this particular case what we see is that we are getting a performance of 10 mega flops which is you know quite far from gigaflop or something.