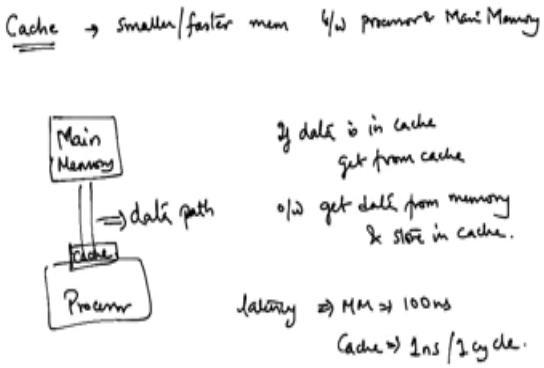
Introduction to Parallel Programming in OpenMp
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 06
Cache and Temporal Locality

So, how do we address this issue? So, we know this is a major issue, the gap between the rate at which the professor works and the; you know time it takes to fetch data from the memory.

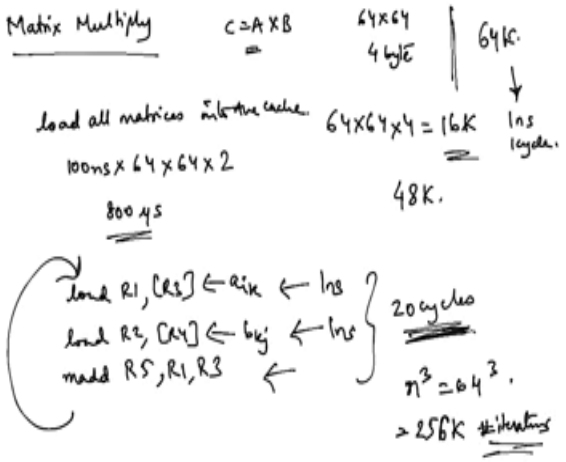
(Refer Slide Time: 00:12)



So, what we use is something called a cache it is essentially a smaller, but faster memory that sits between the processor and the main memory, it turns out to be much costlier. So, you cannot you know have your entire memory being replaced by this, but it turns out that you know you can actually gain a lot by using a cache, and even small caches help you a lot. So, how does the cache work? So, this is your main memory this is your data path, and this is your processor and what we do is that, we put a fast a memory right next to the processor, then whenever the processor access; processor accesses some data, if that data are resides in the cache it is simply returned from the cache. I am giving a very simplistic view of what a cache does, right. So, if data is in cache get from cache otherwise get data from main memory and store it in the cache and now what happens is that ah.

So, the latency to main memory was something of the order of 100 nanoseconds or 100 cycles, but the latency to the cache is just a few cycles right. It can be as small as one cycle. Does cache always give you benefits no depends on the code if I am doing something like let us say something that that is just accessing data randomly in the memory, it is not going to give me a lot of benefit, but if I am accessing data in a in a structured way in the memory it gives me a lot of. So, let us again look at the same example that we were looking at earlier.

(Refer Slide Time: 02:48)



So, we will look at matrix multiply again like. So, we wanted to compute C is equal to A times B, we are assuming that all the matrices are of size 64 cross 64, and each element is a 4 byte floating point number right. So, let me assume that my cache size is let us say 64 K, what is the total size of matrix a in bytes? There are 64 times 64 elements and each of them is 4 bytes, right.

So, the size of A is.

Student: (Refer Time: 03:26).

16384 let us compute in terms of K right. So, this is 16 K.

Student: (Refer Time: 03:35).

Similarly matrices C and B are also of the same size each of them is 16K and so, the total data size is 48K. So, all of it fit into the cache, I can fit all of this in the cache if I want to. So, how would my matrix multiply perform now? Let us assume that the cache gives me a latency of one nanosecond or one cycle, and the main memory as before is giving me a latency of 100 cycles or 100 nanoseconds . So, now, what happens is that first time that I access an element it is going to have to go through the memory and get it right which is going to take 100 cycles or 100 nanoseconds, but every subsequent time that I access the data it is going to come from the cache right. So, let me just simplify it let us just say that I just load all the data into the cache first right, although that is not the way you do it, but let me just assume that just to do a simple analysis right. So, we load all matrices into the cache how long is that going to take? 100 nanoseconds into;

Student: (Refer Time: 04:50).

Into 64; into 64.

Student: (Refer Time: 04:53).

I do not need to load see.

Student: (Refer Time: 04:59).

So, I will just stick with 2, right. So, that is about 800 microseconds, 32 times 32 is about a1000, it is 1024 right. So, I am going to approximate 1024 with 1000 for now. So, 800 microseconds is what it takes to load the matrices. Let us see now I start the execution of the code right what did I have I had the instructions load R 1 from address in R 3, load R 2 from address in our 4, this was actually fetching aik this was fetching bkj and finally, I was doing an madd into R 5 from R 1 comma R 3 right that is the instructions and then I was repeatedly executing this way.

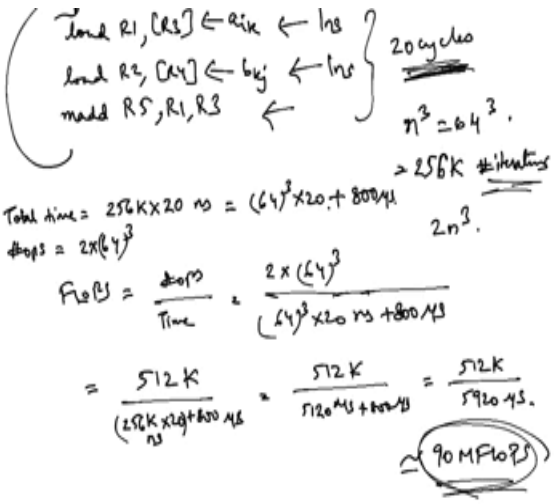
Now how long is it going to take to execute each one of these instructions one nanoseconds well I am talking about throughput right I am assuming that the we are pipelining right, and the latency the memory latency is going to be one nanosecond right and if I am able to get perfect pipelining then each of these instructions will complete in one nanosecond right and then I do this a madd I increment the addresses I go back and so on.

So, actually it is going to be a few cycles that are going to execute right. So, let me just approximate this and say that maybe takes 20 cycles, I am just taking your lose approximation right suppose it takes about 20 cycles to execute one iteration. So, this is much better than 200 nanoseconds, it used to take earlier side because the data is coming from the cache. So, your memory latency is just one nanosecond. So, you are able to do the operant fetch within a single cycle, what is the total number of iterations I have to do n cube.

Student: (Refer Time: 07:09).

So, that is about 256, 256000 around 256000 right that s the number of iterations I will have to do. And I have just taken a bound of 20 cycles that roughly the time it is going to take.

(Refer Slide Time: 07:33)



So, what is the total time 256 k into 20 nanoseconds and what is the number of operations are performed? 2 n cube right number of multiply adds I am counting multiply and add separately 2 into 64 cube right and this is 64 cube into 20 right. So, what is my flops? So, that is the number of aops divided by time which is going to.

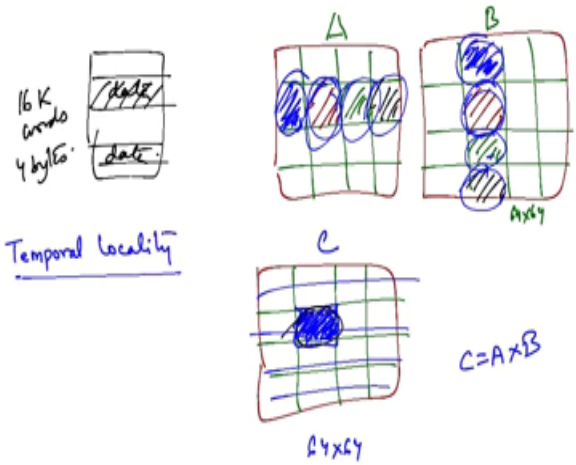
So, this time is actually not just 64 cube into 20, but 64 cube into 20 plus 800 microseconds right. So, this is going to be plus 800 microseconds. So, what is 64 cube? 64 cube is 256 k. So, this is 512 K, 256 K into 20 plus 800 microseconds this is

nanoseconds which is 512 K divided by 5120 microseconds plus 800 microseconds these about.

Student: (Refer Time: 09:39).

Similar 90 megaflops what kind of performance was I getting without the cache? You were getting about 10 mega flops right and with the cache you are getting about 90 mega flops. Well there is a lot more you can do. So, the idea here is how we can utilize the cache right to overcome the memory latency issue. So, what happens if you have very very large matrices to multiply which do not fit in the cache? So, what is the important part here right the important part is that once you have fetched the data, you want to reuse it as much as possible.

(Refer Slide Time: 10:18)



So, how does the cache work right? So, you have some data loaded over here there are different locations right. So, when we say 64 k cache and let us say the word sizes 4 byte, then it is essentially it storing 16 K words each of 4 bytes right. And what happens after you access 16 key element, when you access the next element what is going to happen? It is going to replace one of these entries, it has to replace one of these entries and there are lots of different algorithms for a doing cache replacement right how the replacement is to be done you simply use and so on, but you are not going to get into back, but the important part to understand is that you have to make sure that if you loaded some data you are going to reuse it, before you know enough accesses happen that this data gets

thrown out. Eventually it will get thrown out, when because the cache size is limited and if you are working on data which is of size more than 16 k elements in this case eventually your data will start getting thrown out.

So, now if you have large matrices and you are working on large matrices, you cannot assume that all the matrices are going to be sitting in the cache all the time. So, how do you write code that makes efficient use of the cache? So, I will just give you a high-level idea of how that is done right. So, if you want to multiply 2 matrices, let us say these are the two matrices what you do is you divide them into blocks. This is something that we are going to get deeper into later in the course we are going to focus a lot on linear algebra problems. So, what you do is you divide your matrix into blocks and each of these blocks is of size 64 cross 64, because we know that as we are working on a block of size 64 cross 64 then I can fit it into the cache right I can fit a block of a and a block of b and a block of c if each of them is each one of them is of size 60 cross 4 64.

So, now, suppose that this is my matrix A this is my matrix **B** and finally, I need the output in matrix C. So, I will take a block view of all of them what is this block? It is a sub matrix of size 64 cross 64, when you do C is equal to A times B, what do you get in this block? If I correct index these blocks is the 2 comma two th block of C. So, it is actually nothing, but the product of second block row of matrix A with the second block row of matrix B.

So, it is the product of this with this, plus the product of this with this, plus the product of this with this plus the product of this with this that is what this 2 comma 2 th block is. So, if I want to do this what will I do how will I make efficient use of the cache, first I am going to load this block of A and this block of B into the cache, do the multiplication store it in C which is going to be in the cache just this block of C. So, that can be done quickly at the rate of 90 mega flops, then I load these two blocks I would again do the matrix multiplication, now all the data is in the cache again I am able to do it at the rate of 90 mega flops and so on while computing C 2 comma 2.

Student: (Refer Time: 14:03).

Right I am doing all these operations these two blocks, and then these two blocks and then these two blocks, and then these two blocks, when I am done computing c two comma two block number 2 comma 2, then I need to basically write it back right and I

do this for all the blocks of C. So, this is called temporal locality. So, essentially what it is saying is that.