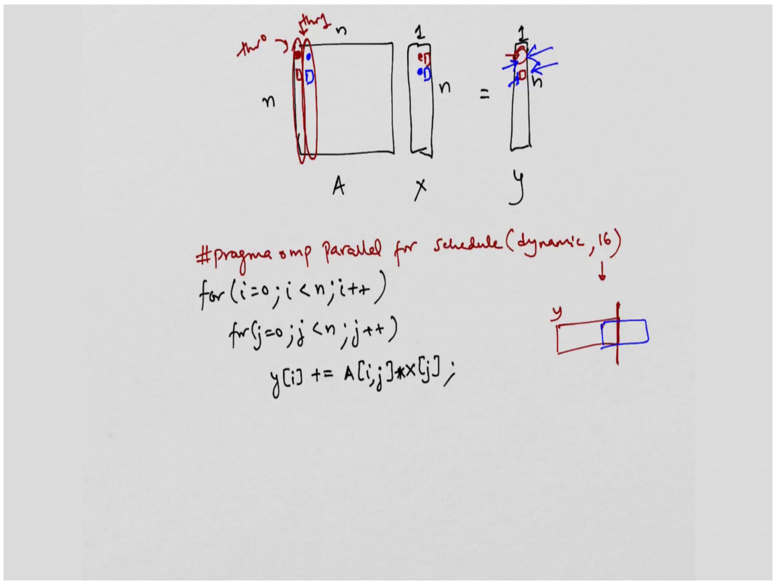
Introduction to Parallel Programming in OpenMP
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 25
Matrix-Vector operations (Matrix-Vector Multiply)

All right let us go to a matrix vector operation right. So, what is a simple matrix vector operation?

(Refer Slide Time: 00:07)



```
#pragma omp parallel for schedule(dynamic, 16)
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    y[i] += A[i;j]*x[j];
```

So, you have a matrix A for simplicity I will take it to be n cross n and let us say you have a vector x which is n cross one, and you want to compute the x equal to y by equal to ax right. So, I want to multiply a matrix with the vector how do I distribute the work what is the simple code for this. So, I will have a loop for i equal to 0, i is less than n, i plus plus, j equal to 0 j is less than n j plus plus and then I say y of i plus equal to fine I am assuming I have initialized y i equal to 0 outside right.

So, this is the operation I wanted to perform; how should I distribute this work, what is a good way of distributing it. So, do not look at the code when you want to distributed like look at the problem, how would I like to distribute this work amongst a threads. So, let me start with one very simple one right let me distribute the columns of a, what happen if I distribute the columns of a amongst the threads. So, let us say thread 0, takes care of this column thread 1 takes care of this column.

So, what will it do what will thread 0 do? It will multiply this element with this element and that result will have to be we will have to go here right, but that is not the complete element of y, that we want that is just a partial answer right and then what will thread one do it will multiply this with this and add it to this right. So, these threads have to basically combine the results right and then thread 0 would also go and pick up the second element, multiply it with this first element and store that here.

Student: (Refer Time: 03:01).

And then thread one would also do the same multiply this by this and add it over here

Student: (Refer Time: 03:15).

Right. So, what is wrong with this approach what is good about it what is bad about it.

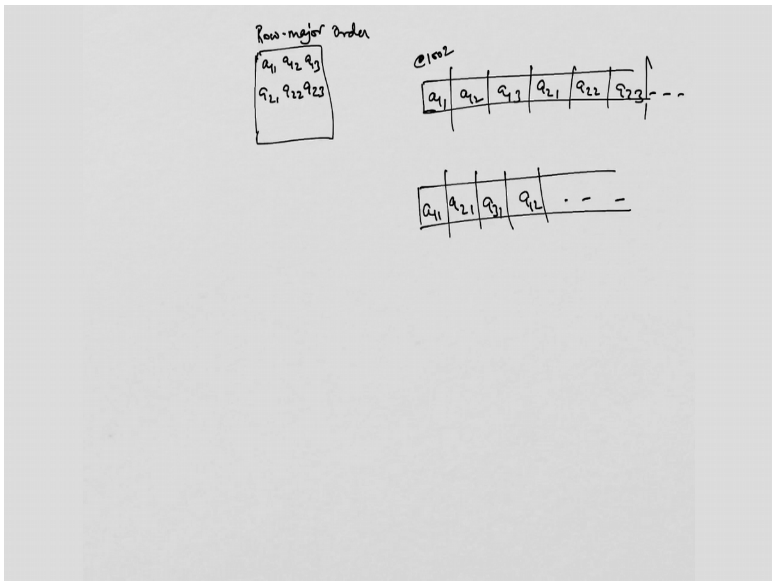
Student: We have full load one thread one (Refer Time: 03:23).

So, if all of them need to write to the same location right and that to very often, every time you have to go and update an element of I just you do one multiplication one addition multiplication and then you have to go and access one element of y, and for that you have to lock it because others are also trying to do operations over there what other issue is there with this approach. So, have you guys worked with Cor fortran.

Student: C.

C anybody worked in Fortran you worked with Fortran. So, there is a basic difference in the way matrices are stored in c and Fortran. So, you have to be very careful when you are coding up particularly I mean if you use to one then you know about it .

(Refer Slide Time: 04:13)


Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
@1002
a_{11} a_{12} a_{13} a_{21} a_{22} a_{23}
a_{11} a_{21} a_{31} a_{12}

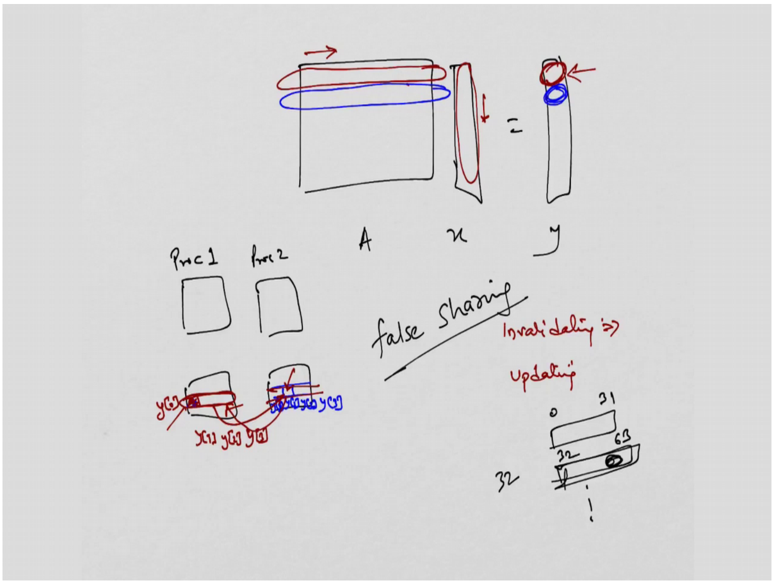
So, the way c stores the matrices is that it stores them in the row major order right, which means that if this is a 11, a 12, a 13, a 21, a 22, a 23 and so on in physical memory physical memory does not have a matrix structure right; obviously, it is just a linear you can think of it as a linear structure the addresses are arranged linearly.

So, a 11 would be stored at some location right whatever this location might be may be address 1002 and then a 12 at the next location and then a 13 and then a 21 and then a 22 then a 23 and so on right that is the way c store of the matrices and how does Fortran store the matrices the other way around right column major order.

So, Fortran would store it as a 11 followed by a 21 followed by a 31 and then a 12 and so on. So, why is this important for us? Thread 0 is working on this column in c how is this columns stored it is not contiguous a row is contiguous column is not contiguous in c in Fortran the column is contiguous right so, but we will stick to see for now.

So, the column is not contiguous and that is a problem right because these data accesses every data access is going to require a hit to the memory, it is not going to find it in the cache right. So, let us study the other approach.

(Refer Slide Time: 05:52)



Let me pick up a row of a and multiply to the column of x and what will I get? I will get the first element of y. So, this is contiguous, a vector is always contiguous I mean whether it is a column or a row does not matter a vector is a vector its always stored contiguously and what is good about this approach where you are accessing this row also contiguously. So, you are getting the benefits of streaming or this is contiguous yeah and each thread is writing to a separate location of y a separate element of y.

So, he has ownership of what is being written into. So, you do not need locks you do not need critical sections you cannot always do it depends on the algorithm, but one of the basic things you should try to do is divide the work given to different threads based on the output where they have to write because if you can separate out the regions where they are writing then you can get rid of a lot of locks and overhead of locks and critical sections, but when you run this code you will be surprised that you are not getting very good performance there is one basic problem over here what is that.

So, let us look at two threads right. So, let us say the first thread is working on row 1 the second thread is working on row 2 right well both of them are accessing x that is not an issue, when multiple processes are reading the same element right reading the same data it can be fetched into their local cache. The only problem comes when any one of them writes to that data or remember the cache coherency protocols right it has to go and either invalidate or update all the other caches right. So, problem comes with writing with reading there is no issue. If everybody is only reading data it will come into each one is cache and reside over there no issues at all ok.

So, what happens over here? So, this is processor one, this is processor two, right and he has is one cache he has is one cache. So, as for as a and x is concerned right there is no issue they are only reading these and reading them sequentially not an issue, here is the problem that will happen with y. When processor one reads y right it is going to load up in its cache somewhere. So, its updating y is 0.

Student: (Refer Time: 08:30).

Right, but unfortunately what you get is the cache line you do not get an element. So, what has come into its cache?

Student: (Refer Time: 08:37) elements.

Four elements maybe right depends on the cache line sizes and the data size, but if you have an 8 byte double precision floating point number and a 32 byte cache line you will get 4 elements.

So, what has come into the cache not just y 0, but y 1, y 2, y 3 all of them are in the cache of processor one. Now processor two starts working it is going to fetch y 1 over here and when it fetches y 1 its going to automatically fetch y 0, y 2 and y 3 also the same cache line is going to come in both of them.

Now, the moment processor one updates y 0 what is going to happen? Although it has not updated y 1, yet it has touched that cache line and whether some data has been updated or not or is dirty now, that is kept track off at the cache line not at the element because the cache does not know what is the data you are storing in it a cache only knows that I have a 32 byte line that is all it knows right.

So, you have touched some data in this cache line. So, the cache says that this is dirty now; this cache line is dirty what is going to be the implication of that? The cache coherency protocol (Refer Time: 09:59) an and it goes and let us say invalidates this cache line, which processor two was working on right and then processor two when it tries to update y 1 its going to make this dirty what is that mean to invalidate the other cache line? It means that the next time processor one wants to work on y 0, it will have to again go and fetch it from the memory because its data has been marked dirty.

So, either you are invalidating or you are updating. In case of invalidating what happens? Whenever you update your value you go and mark all those caches where the same line is loaded you go and mark them as invalid which means that the next time that processor wants to access that data in that cache line, it is going to have to go to the memory right.

In case of updating what do you have to do you have to actually send that data you have to send that data to all, the other caches where that cache line resides so that they can update the local caches. So, getting the issue in both cases there is a lot of overhead of synchronization.

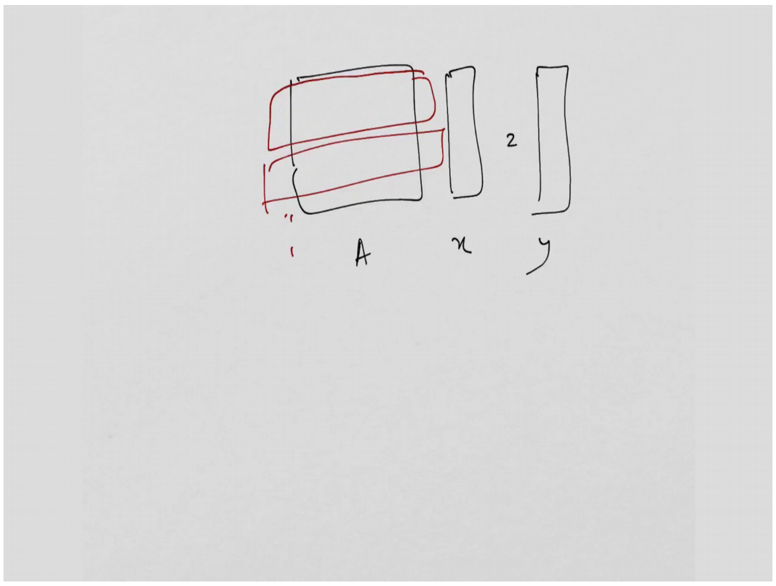
So, this is something called false sharing. Even though you are not sharing the data, but because of the cache line size being large, so the architecture it seems that you are sharing data both of them are trying to access that data and marking it dirty right. So, there are multiple ways of dealing with this problem, one is that just store it in a local variable and update it at the end right into y 0 or y 1 updated only once yeah.

Student: Why where processor to two cache will be having 1 0, it will be having y 1, y 2, y 3, y 4 because it is going to updating the memory location of the y 1.

Yeah. So, the way the caches work is if your cache line size is 32, then the only entries that you can store a address 0 to 31 address 32 to 63 and so on. Irrespective of which data in that you are accessing. So, even if you are accessing this data, the cache line it will load is 32 to 63. So, all addresses start from multiples of 32 byte offsets ok.

So, how do you deal with this problem now? So, there are multiple ways of dealing with it one is that you could store the result in some local variable private variable and then at the end you go and update y 0 right, but that is not the way you want to work right that is not very natural sure, you can do that is one solution and any other solution any anything can else you can suggest.

(Refer Slide Time: 12:43)



So, here is what you can do you can say that let me allocate the first maybe 10, 12, 16 whatever right rows to thread 0 next to thread 1and so on.

So, I am talking about large matrices over here right. So, I could just again use a dynamic schedule and divided into chunks of maybe 16 or something and work on 16 rows at the time right. And let the runtime figure out that how it is to be divided amongst the threads right how would the code look like this is what the code looks like right. So, I add a hash pragma omp parallel for with schedule as dynamic right what is this i? this i is nothing, but the rows right it is a row number. So, I can just say dynamic comma let us say 16 right that is enough to paralyze this matrix vector product that works well.

So, there is always some false sharing that is going to happen right you cannot completely avoid it, but you should try to minimize it as much as possible if you know that you know there are simple tricks to avoid false sharing then you should definitely employ it. So, here also I mean at the end of the day after you have allocated 16 elements of y right you do not know whether that is it a 32 bit by boundary or not right that this may be in the middle of a cache line, maybe this cache line starts here and goes up to here right. So, there may be some false sharing.

So, if you want to avoid that also then you have to ensure that your entire array starts at a 32 byte boundary. So, that it starts at a cache line. So, there are ways of doing that and see right you can specify those things and then ensuring that the chunk size that you have specified is again an amount of data, which is multiple of the cache line size. So, you can do all that if you want.