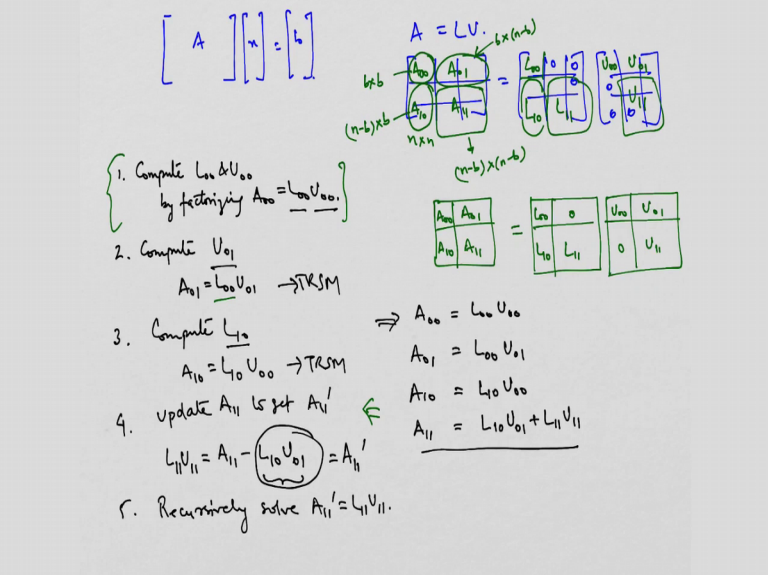
Introduction to Parallel Programming in OpenMP
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture - 33
Parallel LU Factorization

So, now, how do you actually do this when you are given a large matrix, right.

(Refer Slide Time: 00:07)



So, you are given a large matrix A x equal to b. So, remember what we want is blocked algorithms because we want our algorithm to be cache efficient. So, we want to use blocking, right because this is going to involve order n cube operations,

The data size is only n square, but the number of operations is n cube. So, I want to use a blocked algorithm in order to make efficient use of the cache. What I am going to do is I am going to rewrite this as follows. What am I trying to do? I am trying to decompose a equal to LU; what is A? A is a full matrix and if I just write all these matrices in the block form; this is what they will look like and of course, these entries will be 0 and here these entries will be 0, right.

So, what I am going to do is I am going to write this matrix in the following form, I am going to look at this block I am going to call it A 0 0, all right. I am going to call this sub matrix this is not a b by b block, right. So, if this is an n cross n matrix this block is b

cross b; what is the size of this sub matrix its b cross n minus b; right I am going to call this A 0 1 and I am going to call this sub block as A 1 0, right. So, what is the size of this n minus b cross b and this remaining sub matrix? I am going to call A 1 1; this is of size n minus b cross n minus b, all right.

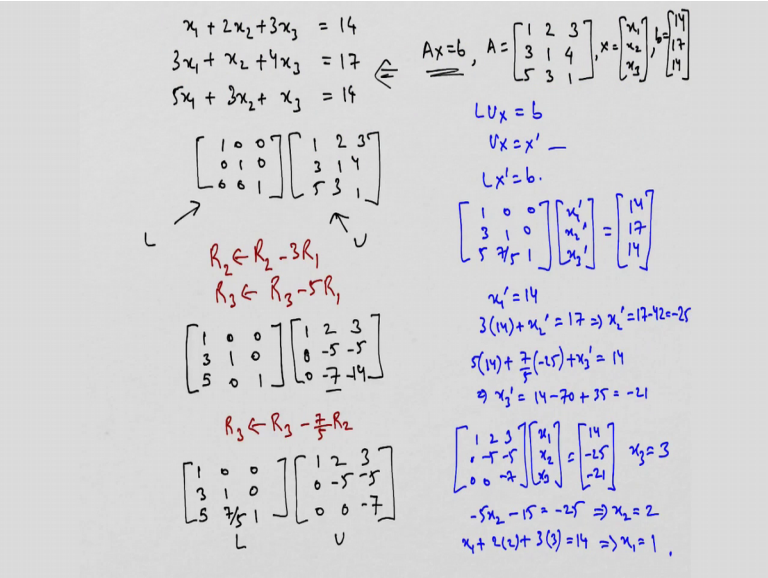
So, these are not sub matrices of equal size, right, it is just I am carving out b cross b block at the corner. Similarly, I am going to break this matrix into A 0 0; I do not need to have a name for this one because this is going to be 0s; this part I am going to call L 0 1 and this is going to be L 1 1, similarly this is U 0 0, U 0 1, sorry, this is L 1 0; L 1 1 this is U 0 0; U 0 1 and this part I am going to call U 1 1.

So, let me just rewrite this for you. So, this is how I am viewing the matrices A 0 0; A 0 1, A 1 0; A 1 1; L 0 0, L 1 0; L 1 1 this is 0; U 0 0, U 0 1, U 1 1 and this is 0. So, let me just write out the equations that we get. So, what is A 0 0. So, multiply the first row of L with the first column of U, right. So, what do I get? L 0 0 multiplied by U 0 0 gives me A 0 0; what happens if I multiply the top part of L with the right part of U? I get L 0 0 times U 0 1 is equal to A 0 1.

Now, let me multiply the lower part of L with the left part of U. So, what do I get? L 1 0 times U 0 0 gives me A 1 0 and finally, L 1 0, U 0 1 plus L 1 1, U 1 1 gives me A 1 1; all right. So, now, we are going to talk about the algorithm; the blocked algorithm to do LU factorization. So, here is how I do it? So, the first thing I do is I do this step compute L 0 0 and U 0 0 by factorizing A 0 0 equal to L 0 0, U 0 0, how do I do this? I can do this sequentially.

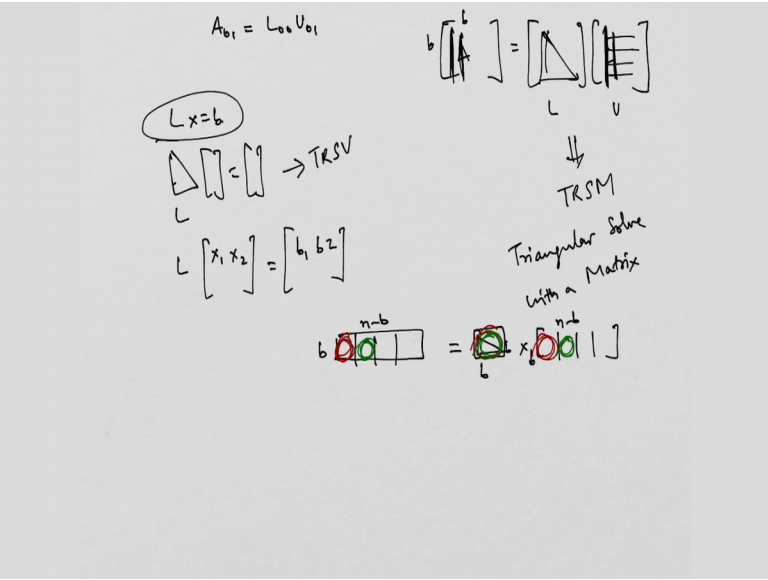
So, what are the exact steps we saw that on the previous slide, right.

(Refer Slide Time: 04:58)



What we saw over here; this entire thing was how you do the LU factorization, right, the only thing is that we are not doing it for an entire matrix; we are just doing it for a block. So, A 0 0; I will I pick up A b by b block and I factorize it I do the LU factorization of that right. I get L 0 0, U 0 0. Now that I have L 0 0 and U 0 0, I can compute U 0 1, I know that A 0 1 is equal to L 0 0, U 0 1 and U is upper triangular. So, let us look at this in a little more detail; what is happening over here.

(Refer Slide Time: 05:39)



So, A 0 1 is equal to. So, I have a matrix block A, this is b cross b, this is A, this is equal to L which is lower triangular times U which is a full matrix, right. U 0 1 is not upper triangular, right. U 0 0 is upper triangular, U 0 1 is full right because the entire U matrix is upper triangular. So, the U 0 1 part is full, all right. So, this is what I have to solve; how do I solve this? A little while ago, we learned how to solve lx equal to b where L was lower triangular. So, a lower triangular matrix times a vector is equal to another vector we just learned how to do that.

So, what happens if I have another set of vectors? So, the same lower triangular matrix, but if I have another set of vectors to solve, I can just append it to this column, right. So, L times x 1 x 2; these are 2 different vectors and I get b 1 b 2 and I can solve for them, right because what is the product of this lower triangular matrix times the first column that will give you the first column of A, right. This lower triangular matrix times the second column will give you the second column of A and so on we have already seen how to do lx equal to b, right that is just back substitution. So, here you are just doing back substitution on a set of vectors together what was this operation this operation was we called it TRSV triangular solve with a vector. This operation is called a called TRSM; triangular solve with the matrix on the right hand side.

So, these are standard operations; you will find them in all linear algebra libraries. So, what is the point; the point is I know how to solve this A 0 1 is equal to L 0 0, U 0 1, what is the size of A 0 1, here it is not b cross b instead it is of the form b cross n minus b this is equal to b cross b matrix. This is b cross n minus b times A. So, how do we do this I know how to do PRSM; this is lower triangular, right, but this is just a loads of vectors, I can just pick this into blocks of b cross b and this also into blocks of b cross b and this is nothing, but this lower triangular matrix times this first block and the second block is nothing, but this lower triangular matrix time the second block and so on, right. So, I can do this block by block, all right.

So, let us come back to the algorithm. So, I computed U 0 1 by doing a triangular solve matrix, right A 0 1 is equal to L 0 0, U 0 1, all right and now I can also compute L 1 0, right because A 1 0 is equal to L 1 0, U 0 0, U 0 0 is upper triangular. So, this is again the same operation triangular matrix solve just that instead of a lower triangular matrix, I have an upper triangular matrix. So, what have I computed so far? I computed L 0 0 and U 0 0 in the first step, I computed U 0 1 in the second step L 1 0 in the third step.

Now, what am I left to compute L 1 1 and U 1 1, right. So, let me use this last equation. So, what do I have I have A 1 1 is equal to L 1 0, U 0 1 plus L 1 1, U 1 1, I can rewrite L 1 1, U 1 1 is equal to A 1 1 minus L 1 0; U 0 1. So, I have already computed L 1 0 and U 0 1, right. So, I can multiply them and subtract that from A 1 1. So, what is this operation? This is matrix multiply I am simply multiplying to we have already seen how to paralyze that right how to do that in a blocked manner and how to paralyze it.

So, I subtract from A 1 1; L 1 0 times U 0 1, what do I get? I get a new matrix. Let us call it A 1 1 prime and A 1 1 prime is nothing, but L 1 1, U 1 1 in my original notation. So, how do I get L 1 1, U 1 1 well I have got a matrix A 1 1 prime which is supposed to be L 1 1, U 1 1. Now how am I going to get L 1 1 U 1 1?

Student: recursively

Recursively; do the same thing over a time at this step update A 1 1 after updating I am calling it A 1 1 prime and finally, recursively solve A 1 1 prime equal to L 1 1, U 1 1 and that is going to give me the next set of blocks and so on until I get the entire LU, all right. So, this is the blocked algorithm to do LU factorization and how can we paralyze it. So, let us just quickly see I am not going to write any openMP programs over here, but let us just quickly see how we would paralyze this.

So, the first operation let me just do it sequentially this is not the bottleneck of the code where is the bottleneck of the code this update A 1 1 is the most expensive part. So, let me assume that I have very large matrices right because there are small matrices it is not. So, interesting, right, you are not so interested in speeding them up using parallelism and so on, right, you are only interested when you are huge matrices and in most simulation of most scientific problem is you are dealing with huge matrices.

So, this is definitely a bottleneck, this is the biggest bottleneck why because this is a full matrix multiply and this involves n minus b cube operations, right the next level you would want to optimize steps 2 and 3 step one is just A b by b and b by v decomposition over b by b. So, it is too small in the whole scheme of things. So, I do not mind writing this part sequentially. Let me write it sequentially, I explained all the steps over here, right, how to decompose A equal to LU using Gaussian elimination? How can I paralyze steps 2 and 3 can steps 2 and 3 happen in parallel yeah both of them are dependent on what I need L 0 0 here and I need U 0 0 here and both of them are computed in step 1.

So, I have both of them. So, there is no the dependency between step 2 and step 3. So, I can just do them in parallel, right and even when I do them in parallel as I said, right, we can divide each of these into smaller blocks and just work on one block at a time just distribute this to the threads right. So, in openMP, I would divide this b cross n minus b block into b by b blocks lots of b by b blocks and just in an array use a hash pragma omp for and divide them to threads or maybe use tasks and just divide them distribute this work to different threads.

And finally, for step 4 that is matrix multiply we already know how to do that right you divide into lots of b by b blocks and you do block matrix multiply and then you recursively go and solve it. So, that is a LU factorization and solving a system of linear equations in parallel, all right that that is how it is actually implemented in most scientific or high performance computing codes that you will see on the net this is how it is implemented.