Introduction to Computational Fluid Dynamics

Prof. M. Ramakrishna
Department of Aerospace Engineering
Indian Institute of Technology - Madras

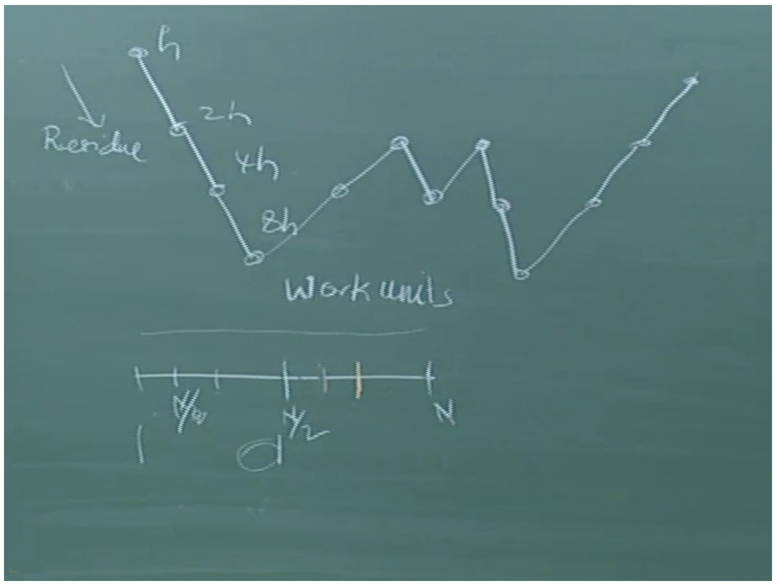
Lecture - 35
Multigrid method - Final, Parallel Computing

We have been looking at the different ways by which we can accelerate the convergence I would say of the program, I use convergence in a very generic sense, again I repeat when I say accelerate I actually mean that I want my answer back as quickly as possible. So you have to really take the whole picture into account, I want to we have looked at I used Laplace equation as an example for multigrid method, those Laplace equation is linear equation.

And as a consequence the equation that we got for the correction was identical to the original equation okay, so what you are going to do is I will show you how to apply it a possible way to apply it to 1D Euler equation, in fact it is a variation of an algorithm that was actually developed here on campus. Just a small, it is basically an application that application of the so-called correction scheme, what we have been looking so far right to the Euler equations.

We will do that, but first yesterday, after the class I got a question, so I thought maybe I will just mention it because it is of importance okay. So we are transferring the residue from the finest grid to a grid a coarse grid, and the course grid sizes is 1/2, the number of grid points is 1/2 or the actual grid size grid width is double the fine grid right double. So we were going through a process okay, so if you think about it we are going through a process.

(Refer Slide Time: 02:06)



Let me just start of this is h, 2h, 4h, 8h right, so we have a process by which we transfer the residue okay, so remember we are going down it is always the residue. I want you to remember this, because when we do the Euler equation there can be some confusion residue, so you are going to transfer the residue from the fine grid to the coarse grid okay. So what is going to happen is that this is we are not going to make corrections essentially.

We are not going to make the corrections although, the corrections will always be made on finer grid that is the idea, the residuals always passed to a coarse grid. So question the obvious question is why go from h to 2h? Why not go from h to 4h? Right, if so we talked about having V-cycle or a W-cycle or something of that sort, so yes you could have right you could have so you could have different kinds of cycles that you go through.

And earlier I would suggest that well we want to stay down here at the lower end, because the amount of computation number of work units the term we used was work units right, the number of work units being used was less, am I making sense okay. And the obvious questions that comes up is why go from h to 2h? when you can go from h to 4h or h to 8h directly right, why bother with the intermediate steps, and there is a logic to it.

The reasoning being that if you go from h to 8h, so you remember the underlying principle that we are using, if you go from h to 8h right, the grid size is much much larger, the number of grid is much less. So the highest frequency that can be represented here am I making sense, is 1/8 the frequency that can be represented there okay, we are using that fact that when I go from h to 2h, so we were basically saying that on that grid h if this represents the spectrum the height the frequency range that you can represent on the grid h.

On the grid 2h that range drops from the largest wave number that you can represent becomes N/2 okay, so on this what I want to show is on this if you have a residue let us say that is uniformly distributed initially, the error is if you want to do enough iterations here, so that you eliminate the wave numbers on the higher half, then you can transfer it to coarser grid, where it would be N/2 right.

If you wanted to go down to N/8, you would have to do a lot more work on the fine grid, we do not want to do that much work on the fine grid, you understand what I am saying, it would either have to do a lot more work on the fine grid or you have to do more residual smoothing or smoothing of that sort okay. In order to push it down right, so this hierarchy really works, in fact you could say well is not there a way by which we put get an intermediate grid, but the cost is very high.

So it turns out that h, 2h, 4h, 8h seems to be the best way to do it okay, there may be other possibilities that you could think of instead of going from maybe h to 2h maybe h to 1.58h somehow right, I mean you can think in terms of fractions like that. But basically what you do is now, what is just to continue on this more since I have drawn this figure, what is the algorithmic what are the issues when you implement?

So one way would be yes you do some iterations with the equation, so if you do some iterations with equations possibly you shrink it, you get it down to that right, you do some residual smoothing and maybe you get it down to N/2 transfer the grid that is one way to do it. If you do not do that from the demo that we did, do you remember what happens if you take frequency higher than what the grid can represent? Do you remember what happens to that? It folds over.

So any error that you get close to N will actually go close to 0, am I making sense, any error that you make close to N will actually go close to 0, if it goes close to 0 is very bad right, close to 0 decaying at the slowest rate, close to 0 is really bad for all the grid close to 0 is difficult, am I making sense close to 0 is quite bad. So what you want is you want, on the other hand close to N/2 is not bad, so now I am talking about how much effort how much residual smoothing do you want to do.

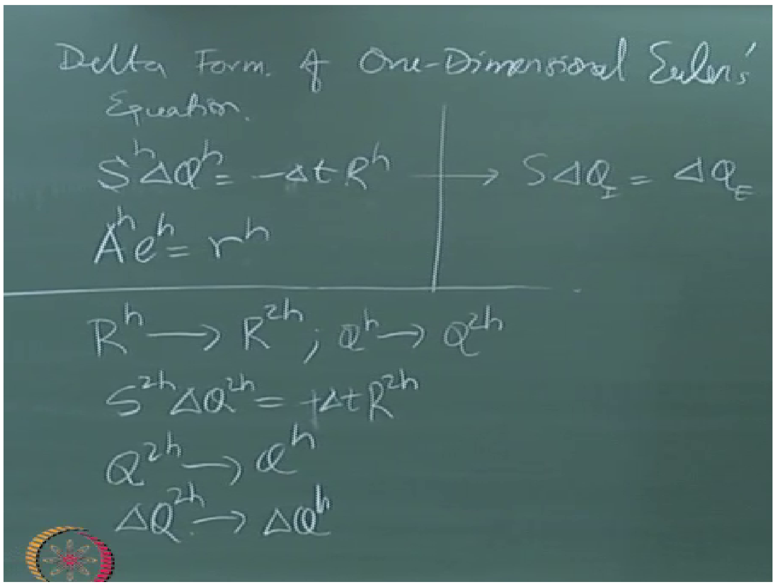
If you go from N to 3 quarters N where will 3 quarters N fold over toward N/4, am I making sense. So if you go to somewhere there this is going to fold into the higher frequencies N/2 which is going to go away quickly anyway, am I making sense. So you can see that actually you do not need to make that much effort that on the finest grid, on the finest grid you need to make enough effort that you definitely get to 3 quarters N, maybe you leave yourself a little margin okay.

Then transfer it, yes there is aliasing you are doing something bad, but it the transfers whatever happens whatever contamination that occurs, occurs there if you are willing to live with it, it is going to decay very quickly right on the 2h grid it is going to decay very quickly right you are going to put in that effort anyway, but that is coarser grid, you are doing less work there, for one sweep you do less work there, am I making sense okay right.

So of course the proper way to do it would be to do sufficient iterations, so that you push out you eliminate, so you totally reduce the residue that you have in on the top half, so that whatever you transfer down there is no sampling related issue, there is no aliasing, is that fine okay. So there may be a temptation to go from h to 4h, but that would require a lot more work on the fine grid, that is the reason why we are not going to do it okay is that fine right.

So now how do we apply this to non-linear equations, well as I said I am going to look at a very particular way by which we have done it. We just recognize the fact that we do not solve non-linear equations, we invariably end up linearizing right, we do not solve non-linear equations most certain times, we just end up linearizing unless it is some form some quadratic or something simple that we can handle, it typically ends up in linearizing.

(Refer Slide Time: 10:03)



Delta Form of One-Dimensional Euler's Equation.

$S^h \Delta Q^h = -\Delta t R^h \longrightarrow S\Delta Q_I = \Delta Q_E$

$A^h e^h = r^h$

$R^h \longrightarrow R^{2h}; \; Q^h \longrightarrow Q^{2h}$

$S^{2h} \Delta Q^{2h} = +\Delta t R^{2h}$

$Q^{2h} \longrightarrow Q^h$

$\Delta Q^{2h} \longrightarrow \Delta Q^h$

And the most familiar linear form that you have right now is the delta form okay. So if I look at the delta form of recollect we have already derived the delta form the one dimensional Euler equations. It is of the form I need some kind of a matrix now, so I will just do this I do not think I have used this anywhere, so it is sort of form S delta Q=-delta t*R, in fact if you allow me I will observe the delta t also into the R okay.

So it just gives me basically S is-R, it does not matter we can I do not know why cut corners okay leave it as it is okay. And if you think about it, if you look at this equation now this equation looks like the correction equation, what did we say? We had that is basically what we had right, so this eh is the correction to the current estimate that you have is correction right, the current candidate solution, just like delta Q is the correction to the current candidate solution.

If we are looking at if we were look only looking at steady state, we will just still focus on looking at study state, because we know there are ways by which we can apply the same algorithms to transients right. So if I am just looking at surely the steady-state delta Q is just like this correction, residue is a residue right, so there is no reason there are algorithms out there that will basically say the original equation is non-linear.

And therefore, you should there is an extra term that shows up you can look it up right, I am not going to spend time on it. What we have basically done is we have recognized the fact that anyway I am going to linearize the equation, the equation looks the same right. So what we do is we take this residue here, and transfer the residue the important thing right, the important thing is we transfer the residue R, so all of these are presumably at h we transfer the residue right okay.

It is possible that because Q is remembered this is delta Q, and you are always advancing in time, so other than this you may also have to transfer Qh to Q2h, here we basically this restriction we simply use the injection operator that basically means that we just take the value at that point, we do not really do right we do not do our transfer by averaging and so on okay, so Qh to Q2h you can transfer Rh to R2h, and solve this then on, is that fine.

And in between as I said before doing this transfer just like we did earlier, you can do whatever residual smoothing that is required right. So before you do the transfer before you actually do the transfer, so what would be the algorithm you march in time for a few times steps on the finest grid, you compute the residual anyway from the delta form you compute the residue at every time step, do some residual smoothing, transfer that residue from Rh to R2h right, use injection so we have the Q.

You take time steps now on the 2h grid, am I making sense, so you can take a few times steps to a 2h grid, if you want you find the residue, smooth that residue, transfer the residue to a coarser grid right, then on that coarser grid again go through the same process repeat that the same process, am I making sense okay. When you come back, you transfer the Q back, you transfer the Qh to Q2h and delta Q.

So there are very I am giving you a very simplistic algorithm various wrinkles that you can add to this, you transfer the correction back, so this is the critical thing that I want you to remember. There should be no confusion, you can get into a conclusion because I have pointed out to you this equation, this equation can also be looked upon as S times delta Q implicit right, I pointed out to you that you can look upon this equation as S supplied to delta Q implicit=delta Q explicit right.

If you recollect, if S was the identity matrix, S is the identity matrix and you are do using for instance central differences for R that would be FTCS right, so the right hand side in the sense delta Q in the sense represents the right hand side in a sense represents the correction that you would get from an explicit scheme right, just to recollect we have done this before.

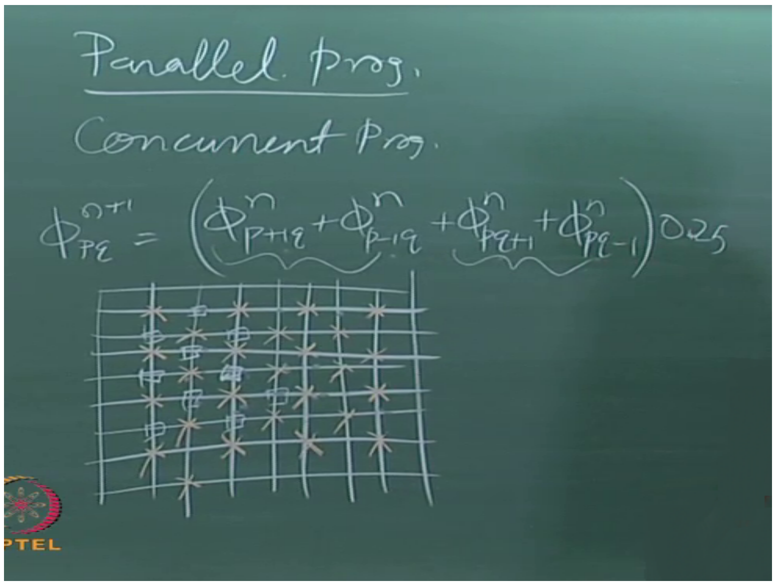
So the delta form in general can be looked upon as the operator acting on the implicit scheme giving you the explicit the increment correction from the explicit scheme.

So this if you look at from this perspective it can get a bit confusing right, so when you are doing multigrid it is better to look at it this way, and remember that it is always the residue that is transfer from the fine grid to the coarse grid, and it is always the correction that is transferred from the coarse grid back to the fine grid okay. And you are always going to evaluate the residue on the finest grid always, the residue ultimately.

When you say yes I have solved the problem, you are going to take a few time steps and you are always going to evaluate the residue on the finest grid, am I making sense, the last few time steps that you take before you declare convergence will always be on the finest grid, is that okay. So that you would have evaluated that residue on the finest grid, and the solution that you have therefore, is on the finest grid right, that is very important, are there any questions fine.

So this is as far as multigrid is concerned. What I am going to do now is I am going to quickly point out other ways by which you can get your answer back fast right, I am not talking about now directly algorithmic, since we are talked about putting things on top, I will just say a few words about parallel computing okay.

(Refer Slide Time: 17:41)



Parallel. Prog.

Concurrent Prog.

$$\phi_{pq}^{n+1} = \left(\underbrace{\phi_{p+1q}^{n} + \phi_{p-1q}^{n}} + \underbrace{\phi_{pq+1}^{n} + \phi_{pq-1}^{n}}\right)0.25$$

I am going to just say a few words about parallel computing, parallel programming okay. So all of us nowadays all of us do a parallel programming right as a routine right, I mean it is a fact parallel or concurrent programming it is called concurrent, concurrent as in the program is running bits and pieces concurrently, which is true. You just get on to the net you check your email across the net.

Or you browse a web page across the net, you are doing parallel programming your local computer is running something some part of it, the computer faraway is running some part of it. In fact, there are lots of computers in between running all sorts of things for you, all occurring simultaneously, so that if you for example or watching a video or something of that sort right, for people who may be watching this streaming, they will see it streaming am I making sense in a sequential serial fashion with nothing jumping.

Whereas there are actually multiple computers along the way that are running, am I making sense, and in your own computer there may be a graphic processor that is running the display and so on. So lots of parallel programs that are running even now right, so the minute you click a button on the browser and something is happening there, something is happening here, you are running a program in parallel.

The only thing that you have to see is how do we bring that advantage to our CFD course okay. So first just you have to get a little flavor, as I said I am only going to give you a little flavor as to what are the underlying ideas, we are not going to spend a lot of time on this. Parallel programming comes in it can be you know they can classify at different levels, so normally you can talk in terms of the coarseness or the fineness of the how are parallelisms right.

So fine grain would be for instance the Laplace equation, if you say phi pq at iterations n+1, 1 quarter of that right. So fine grain, fine grained computation would be that on the given CPU, you recognize that these 2 can be added up while those 2 are added up, that is a very fine grained, am I making sense right. Actually on the CPU there are lots of things that occur in parallel in the sense that it is possible that when this number is being actually loaded into the CPU, the address was being calculated even as the arithmetic here is being done.

I mean in the actual CPU the other level finer grains of parallelism that is going on that you are fortunately saved from you know having to know what to do look at right okay fine. But it is possible that you say that that I have 2 CPUs I have 2 core or 4 core computer, it is possible that you somehow set it up, so that these additions takes place on one, those additions takes place on one very fine grained right.

So a given phrase or a given sentence is broken up into small pieces that is one way to look at it fine grain. A slightly coarser grain right would be, or I will make it I will swing over maybe to a very coarse grain, I may regret that let me just get a colour chalk. You remember yesterday that we are talking about the chequerboard pattern okay, so we are going to use a chequerboard pattern. So what I will do is mark out points in a checkerboard pattern.

I hope that works, maybe we have chosen too many grid points it does not matter okay, it looks like a mesh but it is okay fine, I do not think I have missed any point. The idea here is if you look at the orange x, and if you look at just the where I have not marked so I have put 4 squares here right 4 quadrilaterals there, the orange x depends on the 4 quadrilaterals, if you solve Laplace equation okay.

So any one of these x-ed out values, the key critical point here is any one of these orange x-ed out values does not depend on the other orange x-ed out value, does that make any sense is that clear, you just look at p+1q, p-1q, pq+1, pq-1, any one of these orange x-ed out values does not depend on that depends on the neighbouring values, does not depend on the other orange x-ed out. So the all the grid points which are marked with an orange x are independent of each other.

They can be calculated simultaneously right, they can be calculated simultaneously, you can understand what I am saying, because they do not depend on each other one value does not depend on the other right. And all the grid points which are not x-ed out do not depend on any other grid point that is not x-ed out, the only depend on the orange x's. So if I take this square here, it depends only on the 4 neighbors which are all x-ed out, they are all orange x, am I making sense.

It does not depend on other white square anywhere, is that clear okay. So in a sense all the ones with the white square I am not going to fill it all out, because it is a lot of white squares also, all the ones with the white square can be calculated simultaneously. They are independent of each other okay, so in a sense we have partitioned the problem, we have found some element of parallelism, and they are independent of each other that means they can be done simultaneously.

So on relatively what should I call it coarser grain still fine grain, if you were to store these are vectors, the white vectors would depend on the orange vectors, and you can write a vector equation, and if you have a computer that allows for vector operations, you can actually do simultaneously you understand what I am saying, if you had a computer that would allow you to do the sum and average right, this kind of an operation would allow you to do a vector operation like this.

You could just basically compute all the white squares in one shot as a vector operation, then all the orange vector x's is another one shot, you understand what I am saying as a parallel operation. But that of course assumes that your computer will allow you to do parallel operations right, it is possible there are even on your even on the desktop that you have, it is actually possible for you to write it out.

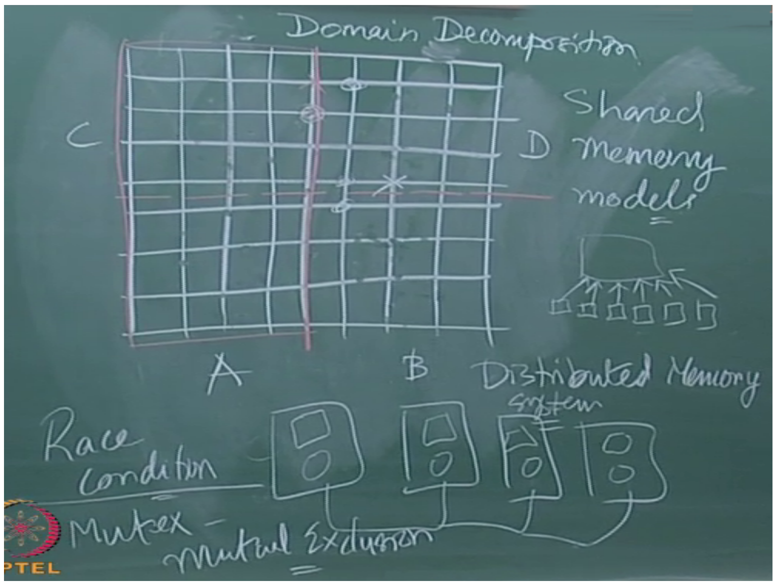
You may to choose 8 at a time or you may have to choose 4 at a time, that is you may have to choose 4 of these at a time because it supports a maximum of 4 a vector of 4 right, or you may have to choose 8 at a time because it supports the maximum a vector of 8 or whatever you understand, your graphics processor for example does mostly 4x4 transformations, all the matrices or 4x4 operations floating point operations right.

So it is possible for you to actually say that I can do 4x4 vector operations, I will map this in a parallel fashion into 4x4 vector operation, am I making sense okay right. That is the other way to do it, here what we have done is we have done what is known as domain decomposition, I have decomposed the domain on which the problem is defined, I have decomposed it into 2 sets. I could decompose it as many sets as I want, but I have decomposed it into 2 sets.

Each of the orange x is itself can be divided into 4 or 8 or whatever is you can further subdivide, knowing that they can all run in parallel right. So for example if you divided this sub into if you say that any CPU that have can do vectors of 8, but I have 10 CPUs then you can take 8, you can break it up into 10, 8, 8, 8 and they can all run in parallel. The fact is because they are independent of each other, they can run in parallel that is the important part okay, that is the important part.

And if you get what is known as linear speed up, then if we use 2 CPUs it will be twice as fast, and if you use 4 CPU it will be 4 times the fast. Sometimes, because of various reasons you can get super linear speed up but that is rare, most of the times you either get linear or something less than linear, is that fine.

(Refer Slide Time: 27:52)



Another way to partition it, right now we since we are talking about domain decomposition okay see as I said one way of looking for parallelism is that you basically see that the operator and the operations that you can do can be decomposed, here the parallelism we seek is come from the domain right. So domain decomposition, so what you could do is you could break it up along that grid line, I divide the domain into 2, along the grid line I partition it into 2.

So the grid values to the right and the grid values to the left are mostly independent of each other is not it, the grid values to the right and the grid values to the left are mostly independent of each other. So you have to decide now, there are different ways by which you can do this, the number of grids that I have drawn you could of course draw the dividing line in between right, that could make the partition easy, but you can decide how you want associate.

So let us just say for now that you have associated all of these points, you are associated all of those points in one set okay. So what we are going to do is we are going to solve this problem basically on one CPU, and you are going to solve this other problem in another CPU, that is the idea. Then what we do at the interface? What is the problem at the interface? Well, if I am going to take the average of this, I need a value that is on the adjacent CPU.

So if I say that this is going to be solved on CPU A, and this is going to be solved on CPU B, then if I go to the boundary grid and taking averages at boundary grid, I need values from the adjacent CPU, so for boundary grid B I need values from A, from boundary grid A I need values from B okay. Now one of the first observations that we make is B will take values from these kinds of responsibilities have to be very clear.

B will read values from A, but B will never write B will never change these values they are not computed by it, they are not completed by B. The values that these grid points are the responsibilities of A alone, there is no place where there is overlapping, there is no place where there is no grid point at which both will do the governing equation, I want to be very clear, there is no grid point where both would do the governing equation.

If on A you were to compute the, if you want to compute the boundary the value at the boundary grid point, you would see from the averaging process or Laplace equation that you need something that belongs to B. So when you doing this parallel computing A and B need to transfer data okay, we are back to this idea of transferring again, A and B need to transfer data right. And as far as you are concerned as seen here it looks like well it would be looks suspiciously like applying boundary conditions, but A and B could be transferring data.

I am just saying that it is suspiciously like boundary conditions, because in your mind you can in your program you can implement it as though they are boundary conditions okay there is a certain ease, is that clear okay. So along the border wherever you however, you decide to split it so you could split it for instance here I will deliberately draw it in between okay, instead of drawing it at the grid line, I will draw it in between.

So if you had 4 CPUs, you either I have a quad core machine or you actually have 4 CPUs, it is possible that you split it 4 pieces and the same thing happens. So you need to know, see you now need a little organization, you need to know where is this data. So if I split it now if this bottom 2 are A B and the top 2 are C D, I will write it on the side C, D, if these 2 are A B and those 2 are C D, then you need to know whether the data comes in your computing a D boundary whether it comes from D or weather when you computing a D boundary whether it comes from C okay.

So they will have to trade, so you can see that as I break this up into smaller and smaller pieces, what is going to happen? What is the trend that you see? The volume of computation on any CPU is going to shrink, the amount of data that you are going to trade is going to increase okay, so you have to strike a balance, it does not make sense saying I have 1024 CPUs, I have 1024 grid points, let me distribute 1 grid point on each CPU.

Then all they will be doing is trading, am I making sense? It does not make sense, so you have to be careful, you have to make sure that each one of them is doing a reasonable amount of computation before it trades okay. And again we are back to this issue of if am interested only in the steady state, maybe I will take a few time steps before I trade. Why should I trade at every time step? I am not looking for a transient right.

So maybe I take 5 or 10 time steps and then I trade, am I making sense, in each one of this I am only looking for a steady state solution. If I am only looking for a steady state solution, then I do not have to update these boundaries at every time step, if I am looking for a transient yes, but if I am not looking for a transient I just take a few times steps instead of trading everyone time step, I trade I take 10 times steps, and then I tried.

So and then you usually find that so this really depends on how expensive is the trade okay, how expensive is it to trade this? So now you come to the next 2 levels of coarseness or fineness of computation. The 2 models that you will hear about are shared memory models, so typically if you have a dual core or a quad core or an 8 core or a 6 core or whatever it is, so it is CPU given in that box sitting on the oppositely on the same memory bus.

The memory is shared all the CPUs see the same memory okay, so in which case the trade is not really that expensive, there are other issues in parallel programming that you have to be worried about right, and there may be people who get upset that I not talk about race conditions and all that but anyway right, I will just mention it as you go along. But the fact of the matter is, there is no expense in the sense of there is no expense in trading.

So you could trade in every time step if you wanted okay, there could be a potentially a problem with each trade we will talk about that, but in a sense if it is a shared memory model, which basically means that you have memory and you have lot of CPUs. And all of them are connected to same memory, they see the same memory then there is no cost right, other than the fact that you may have you know there is no cost as far as your concern.

But there could be cost I mean there could be bus contention there are lots of other issues, a lot of homework that the CPU may have to do in order to get what it wants right. The general idea being that you keep the trade down to as little as possible right okay. The other possibility is a distributed memory system or a model, as I said all of this is just to give you a flavor as to what it is all about.

So here you would have computers that have a CPU and their memory, each one would have their CPU and the memory, and they will all be connected together in some fashion, again the tightness and looseness of coupling will depend on how fast this connection is okay. So if you believe that kind of thing that we do is we just hook them up to a regular network, you have a rack of right a bunch of these and we just hook them up.

Of course our individual CPUs maybe shared memory individual boxes may have 8 cores or 4 cores in them, which means that they may have a bunch of CPUs sharing the memory, and then they could also be connected on the network, am I making sense, is that okay. In which case then the trade becomes a little more expensive right, in which case the what you have to remember is that each one of them, the amount of computation that you do before the trade has to be a little larger for the parallelism to actually be to be worthwhile okay.

So again you do the computation here each one of them would be doing the averaging or the Euler equations or Navier-Stokes equations or whatever it is in each one of the pieces, and at the boundaries you trade the data that is required to be trade, is that fine okay. So what are the possible issues that you have? One possibility of course is right so you have it is possible that there is a piece of memory especially in the shared memory system.

There is a piece of memory that you are reading from into it simultaneously something is writing, see we have already as I said as I have already mentioned, so if you take this boundary point C has the authority to write into that boundary point, none of the other CPUs have an authority to write into, that are not going to do it, it is not going to happen right. So you are not going to really see 2 writes happening to the same memory, that is what I mean.

You are not going to see 2 processes 2 threads writing to the same, 2 processes writing to the same 2 parts of your computer program writing to the same right, so I sort of inadvertently I did not mean to you do it, and I already used the word thread. So this is like you have a thread or a process which is running here right, your process executing here or thread running here, the thread running there, thread running there, they are called threads of execution, you understand.

So there is a thread of execution, when you are thinking about your program running, when you are debugging your program, what are you saying you are actually following a thread of execution you are saying wait a minute okay, here I am averaging, I am setting it here, and then I am going to subtract and find the residue right, you are following a path you are following a thread of execution.

When you are doing parallel computing there are 4 threads of execution if there are 4 CPUs okay right. So the thread of execution here, the thread of execution here especially in the shared memory model where you often use what are known as threads, I mean it actually is the technical term, but the thread execution here, a thread execution here, thread execution here, thread of execution here.

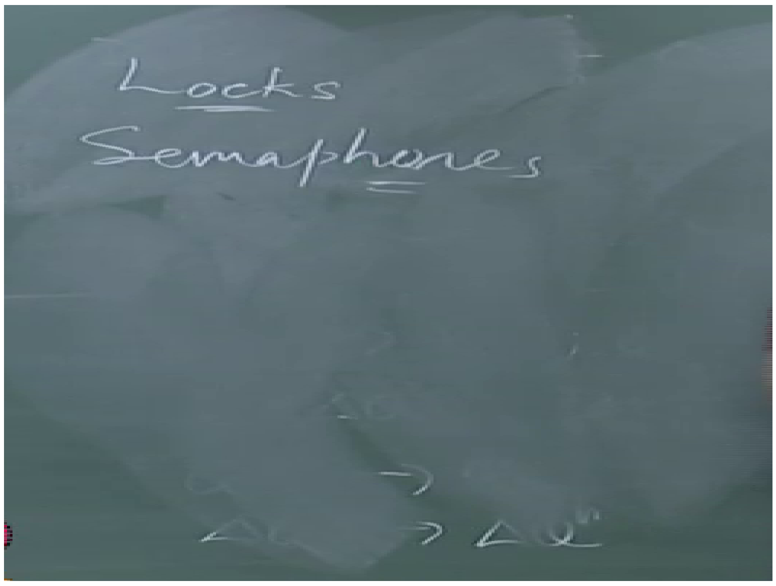
This thread C will write has the right to write, it can write let me put it that way it can write there, whereas D will not write there, you will never have a write conflict. 2 writing 2 threads writing to the same memory location is not going to happen, that is what I am trying to say okay. But you can have D reading from this memory while C is writing that could be a problem right, so you could have when you are saying I am trading.

It is possible that you can always have a situation where one of them get ahead of the other okay this condition is called as the race condition. So it is actually possible for you to get an inconsistent, you can it is possible for you to get garbage, you understand, it is neither the old value nor is it the new value, it is some in between value right. So it is actually possible for you to run into it, possible you to run into a funny situation right.

So though we are guaranteed that only C will do writes, where C's has responsibilities, and D's will only do reads of the relevant parts, there is still that issue okay. So you will here in parallel computing term, you will talk about Mutex- mutual exclusion, so you want the reads and writes, you want exclusion you understand what I am saying, you do not want them to be happening simultaneously. So you want to make sure, so there are ways by which you can do it.

So every time the values on the boundaries are being written, if you look at your programming language it may either allowed you to lock that memory, so that nothing else can read or write from it you can lock it, saying that I am doing something to nobody else can do it, or you can implement those ideas yourself, those ideas are locks yourself okay. So as I said all of these is just to give you a flavor and for me to introduce these terms, these are things that you can go and look up okay.

(Refer Slide Time: 42:30)



So you will hear what terms like locks these are used, if you ever taken a train journey these are all use, semaphores locks, you know what semaphore locks? If you go on a train journey those are those things with lights hanging out that are at various positions, so when it is up that basically means the guy driving locomotive knows that he has to stop right. So far from far off the shape is such that you say I have to slow down, because then you need time to slow down.

So you have a segment of track, and what are you looking for when you have trains in a railroad track? Mutual exclusion right, you know what that is why you have a collision literally, so these things are called collisions, and 2 of them try to do something simultaneously it is called a collision. So to avoid this collision, so what you do is you set up either a flag or semaphore locks, so when in the old days and you can still see it in some railway stations here right.

So there would be a lock, you would be given the privilege of being on the track, you go along then you would be given a lock, and you are on that segment of the track. And when you go to the next railway station, you give up your lock you throw it, there is a little basket like there and you throw the lock, the lock is thrown that is what it was done in the 1800s right, you throw the lock. Because then, so unless you have the lock you cannot really enter that segment of track right.

And as long as you have the lock nobody else has the lock right, so you know that there is no one else on the railroad track, am I making a sense okay. So it is the same idea, basically what you are doing you are saying that I want the memory is where my thread of execution, think of it is a one-dimensional like railroad track, method of execution is going through this memory. And you just want to make sure that 2 of these threads do not collide right.

So you have to have some mechanism by which you ensure that if there are accessing if one of them is accessing some memory, and another thread is now going to come and collide with it okay. So you need to have some mechanism by which lock, you get a lock, so the thread that is going to access that memory can get a lock for that memory. So there are lots of ways maybe people there are lots of implementation, lots of issues that are involved with respect to parallel computing.

It is not that what should I say, I am not saying that you can just go run your program split it up into pieces and it is going to work okay. So there is some element of effort involved, of course there is also since I am constraining myself to CFD, for the most part or the level of parallelism that we have is not the inherent parallelism that we have is not bad okay. So if you divide up the domain things work.

If you divide up the domain and you are careful just make sure that these read write conflicts not occur there is no problem okay. At the other extreme the simplest extreme of parallelism of courses is what is called embarrassingly parallel, so a simple example that would be let us say you wanted to run your program, say this is a this will obviously be a trivial example, just say you wanted to try how your program for various CFL values right.

So in theory you could take 16 CPUs and on each one of them run your program with 16 different CFL values, see the people are smiling, so it is obviously an embarrassingly parallel right. If sigma was a parameter if you discretized in sigma, or you are running if you are running SOR and you are running for different values of omega. So omega can be anything between 0 and 2, you discretize an omega, see I am posting it that way so that I can sort of show it as though it is parallel program.

You discretize it an omega and give each different sets of omega values to different CPUs, does that make sense. But we look at it, as I said people smile because it is obviously an embarrassingly parallel, one of them does not depend at all on the other okay, so in that sense it is an embarrassingly parallel right. So the parallelism is so now you see that there is not only what happens here, but there is also what is the level of coupling that you have in your code.

So if you code the coupling as so tight, that disengaging becomes then that becomes an issue, it is how much effort that you have to put in order to calculate in order to find out a calculate or whatever values that are required, so if you use between let me give you for instance between Gauss-Seidel and Jacobi iteration. We come back here p+1q, pq-1, p-1q, pq-1, see now I have changed it from Jacobi to Gauss-Seidel, you chose Jacobi over for a particular reason right.

But if it is Gauss-Seidel now I have a problem this is n+1, so I cannot if I am starting off at the bottom left hand side I cannot really calculate this still I worked my way up, I mean I can there are ways by which you can do it. But what I am trying to say is if I stick to this I say I want to parallelize this right, I want to parallelize this, then I have a problem because it is very tightly coupled, I cannot do p+1, n+1 unless I have done p n+1 right.

Because when I do p+1, when I want to iterate p+1 and I want to calculate that, this value is what I am calculating right now, and I do not have it I am just calculating it right. And I cannot do pq unless I have done p-1, is that fine. So this is tightly coupled, so though Gauss-Seidel runs faster than Jacobi, Jacobi is easier to parallelize than Gauss-Seidel. Now you have to make your decision which way should I go, so it is not as though here it is you can do this, you know you can do all of them.

So some algorithms may be a little easier to parallelize than other algorithms are right, so it depends on what is the overwhelming need that you have, it is possible they are just running the serial code as faster than running the parallel code, code parallel because of some inherent something like this inherent coupling like this okay. On the other hand, if you say well it does not matter, I do not care, there is a kind of parallelism that you can try to run okay, just say it is just to give you an idea you know.

You can have something like what is like the fugue, you know what a fugue is. So you start here you know I am sure you heard fugues, people usually kids usually do it with row, row, row your boat okay fine right. So what you do is you start here average, average, average, average you work your way down, and once you have done average, average, once you have done, once you have done this, when you have worked your away from maybe I will use that figure instead.

Let me just get back here, once I have gone through here and I have come up here, I can do this now again, I can go to N+2 right I can go to N+2 here, because all the terms required to go to N+2 are there. So while I am going here while my first process is going along like that, the second one can now then start, you understand what I am saying, doing N+2 and trail behind it fine, and by the time the second one comes here the first one is gone one row above that, you can start a third one okay.

So it is just a matter of looking at it and saying, so you can throw your arms up in the air, my god it is tightly coupled I cannot do anything with it, but actually if you sit down and think about it for a while you may be able to figure out a way around fine, is that okay right. So that is as far as accelerating goes, and as I said please remember this is these days it is not that expensive, it is relatively expensive because you need processes right.

So if you want if you want to do multigrid method, if you get a factor of 10 speed up of factor of 100 speed up to get the same factor of 100 speed up with parallel processing, you need 100 CPUs at least, there is a difference right. So that is like a sledgehammer that is an expensive, so that is really like a sledgehammer. So tired out all the others try out SOR, try there are lots of preconditioning right.

There are lots of other algorithmic things that you can do, make sure that you have done all of that stuff, and despite all that yes my program requires parallelization parallelize. Anyway, you should be able to write a program that does parallel 2 CPUs, 4 CPUs possibly 8 CPUs, because if you go out and buy them, they are all dual cores and quad cores anyway right, should be at least be able to use that, is that fine okay thank you.