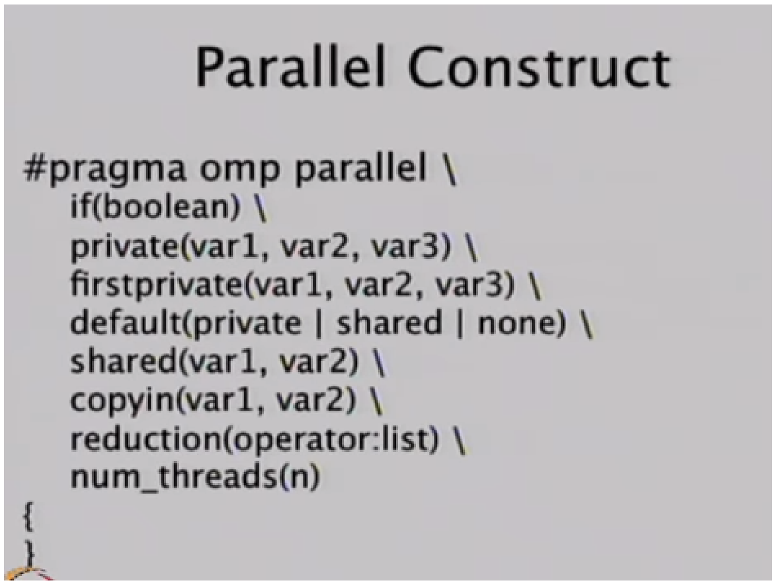
Parallel Computing

Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 02
Lecture No # 07
Open MP (Contd.)

Okay let us being we were talking about few open MP construct and there was something I had said that I will figure out and then come back with the details which was the copy in operator from the parallel construct.

(Refer Slide Time: 00:53)



Parallel Construct

```
#pragma omp parallel \
    if(boolean) \
    private(var1, var2, var3) \
    firstprivate(var1, var2, var3) \
    default(private | shared | none) \
    shared(var1, var2) \
    copyin(var1, var2) \
    reduction(operator:list) \
    num_threads(n)
{
}
```

What I was telling about copy in was actually a for some clause called copy private which is not a part of construct not a part of parallel construct okay. What copy in actually what I had said was copy in is a way for it is a similar to first private except any threads values can be broadcast to others okay. That is the somatic for threads private copy private where all this combination from some time have to figure out which is which.

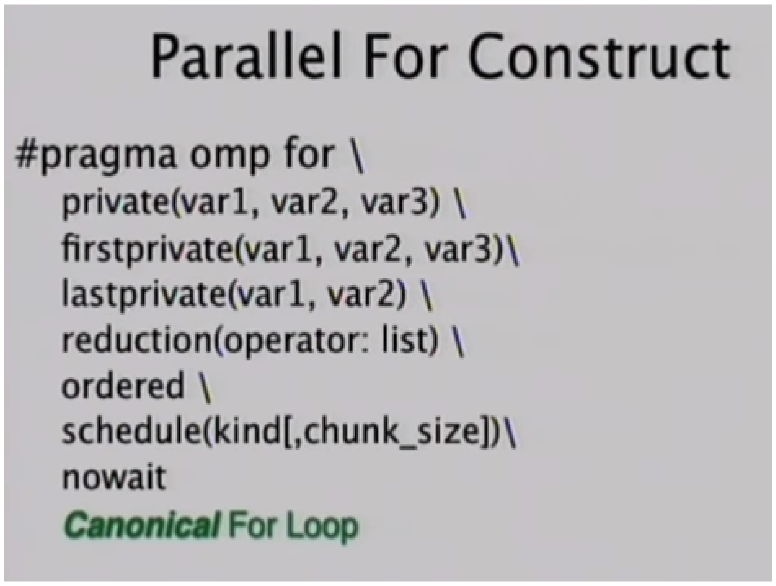
Copy private which we will look at later just to make sure that understand that all allows for one of the threads values to be copied at the end. First private is whatever the master had is going to get copied to everybody last private is last iteration that is going to get copy and copy in is in fact the special type of special private but it is specifically for variables that are declared to be thread private okay.

So now you got all variance of copy thread then private and probably in also there are variables that are shared in and private that we had said will be on the clause but their variables that earlier mentioned in my first or second slide about open MP there are thread private okay these variables are not created when a parallel construct is created now there scope is global they exist outside the parallel construct which means that two parallel constructs they do not make sense between two constants.

Because it is private for a threat but it is kind of a global thread global private which is called thread private. Meaning that I got two parallel construct and I declare something to be thread private it automatically is available here as well as here okay. Thread private is essentially but global in nature and not just has the scope of that construct and for the thread private you have to use copy in not first variable not first private.

Okay not let us get back we are we were looking into various clauses of now copy is just like any other last private first private right. You are saying the copy in these variables those variables must be declared to be thread private earlier on before this parallel construct begin okay.

(Refer Slide Time: 04:08)



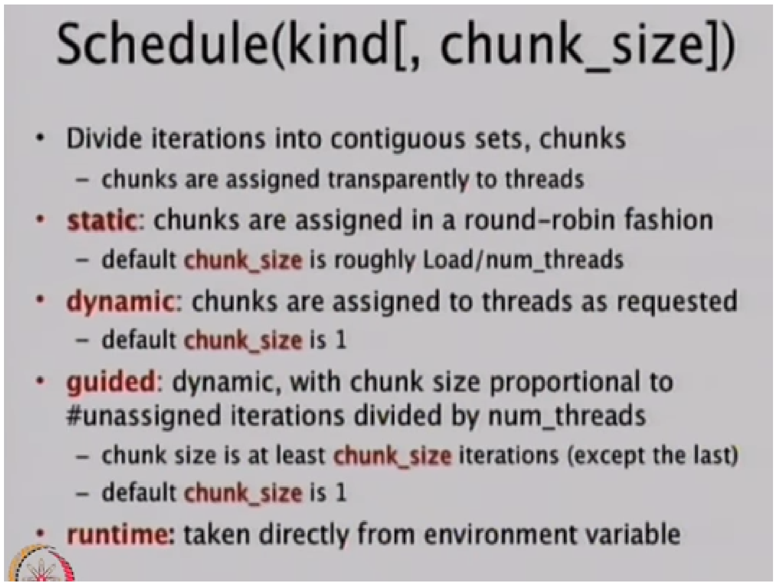
Parallel For Construct

```
#pragma omp for \
    private(var1, var2, var3) \
    firstprivate(var1, var2, var3)\
    lastprivate(var1, var2) \
    reduction(operator: list) \
    ordered \
    schedule(kind[,chunk_size])\
    nowait
    Canonical For Loop
```

Okay that too are the parallel for construct which a either in the context of OMP parallel for or OMP parallel followed by OMP for. We have looked at all these one thing that we were going to talk about schedule one thing I was just discussing earlier is that when you say that these for construct has something that is ordered so you declaring that there is some ordered clause in there will have a (()) (04:41) later on for all the (()) (04:42).

And later on somewhere you say ordered which means there is something some section of code that is ordered, You are saying that section of code must be executed in the order of the loop itself sequentialized is with respect to the ordering of the loop that occurred if you had not been running in the group it is the sequential ordering of the loop. No way we already talked about there is a implicit barrier at the end if you do not want that barrier and we will see some examples where you may not want it you can simply send not wait then whoever is done with the for loop or there part of the for loop can go on and do other things.

(Refer Slide Time: 05:29)



Schedule(kind[, chunk_size])

- Divide iterations into contiguous sets, chunks
  - chunks are assigned transparently to threads
- static: chunks are assigned in a round-robin fashion
  - default chunk_size is roughly Load/num_threads
- dynamic: chunks are assigned to threads as requested
  - default chunk_size is 1
- guided: dynamic, with chunk size proportional to #unassigned iterations divided by num_threads
  - chunk size is at least chunk_size iterations (except the last)
  - default chunk_size is 1
- runtime: taken directly from environment variable

Alright here is the schedule clause which takes at least a kind what type of schedule you want and there is a additional parameter that it can take which has been listed has chunk size. There are four different values that the schedule can take one static which is the default which says that I have got for loop to do it has some number of iterations and I have got some number of thread let us say M iterations to run among N threads okay.

It is going to break those iterations into some groups of iterations okay chunks so for example it says 0 to 4 is one chunk 5 to 8 is the another chunk and so the number of chunks will be the total number of iterations divided by the chink size okay. Roughly it may not be exactly divisible so the static says that I am going to assign the first chunk to the first thread second chunk to the second thread and the number of processor is N.

Nth chunk to the Nth thread and again the N + first chunk to the first thread okay so the chunks are going to go round away. You can declare certain chunk size so if you say chunk size is 1 for example then that means the first iterations is done by the first loop iteration is done by the first thread second by second third by third and then after N thread and N + first iteration will be done by the first again okay.

If you do not specify a chunk size because chunk size s is an optional parameter it says that roughly equal the chunks will be given will be roughly equal in size. I think all implementation has continuous but the specs do not say continuous this is not actually specified in the spec. So the implementation you will use is doing this okay. It is basically the specs is roughly equal an it is taken by most implementation to say that if it is exactly divisible then the chunk size is going to be number of iteration divided by number of processors.

Otherwise some of them will have on more okay the dynamic schedule is you do a job then go ask for the next iteration. So it in the order threads become available to perform an iteration so in the beginning suppose you have N threads and everybody was ready because there is no other code this is where parallel for occurred then one junk will be given to the first one chunk will be the second one chunk to the third one chunk to the last and then whoever gets done with their iterations first gets the next chunk okay.

The chunk size is again determined by you in this case the default chunk is one you do not say chunk size and the first iteration to the first, second to the second, Nth to the Nth and then whoever gets done first gets the N + first alright. There is a variant of dynamic called guided is similar to dynamic chunk are late default is still 1 but now instead of saying that I have given some chunks to some processors and whoever gets done gets the next chunk.

The assignment of the chunk size is dynamically changed not only assignment of the chunk okay so the assumption is that I have got so much work remaining to do my original chunk size may have been 10 but now I have only 10 things to do. If i give all 10 to the first processor that came in and just behind it 2 or 3 more coming in they will not just got anything. So I am going to keep adjusting how much I get based on how much work I have remaining okay.

And the last one run time basically defers says you it an environment variable which will be one of these things either static or dynamic or guided and so it will be determined at run time not part of the code itself okay. Only one is allowed right a give parallel for has to have a uniquely defined schedule either if there is one at the clause then there is only one on the clause. It is only for this for another one there is some other schedule may be not right.

It is basically saying I do not know what to do the user know best and the user know runs do this for everybody. What is the importance of chunk size? That is the dynamic yeah basically it says that I have got so much work that remains to be run in the first one dynamic it is says that I am going to dynamically assign whoever gets free gets the next chunk okay. That chunk is predetermined you said that I want to do 10 things at a time so next processors get 10 things even if I have nothing more to do.

In the guided you say I have got only 10 more things remaining and I have got 13 processors all of them or busy because one guy is asking for it because it is go done. But I am expecting others will also get done soon enough. And so instead of saying you take all 10 am going to say you take just 10 by 13 of them in this case one because minimum one as to be assigned. And I am hoping that very soon that others will also free up and by the time you get done with one others will also be following you.

The original chunk size is part of the if you look at the junk size option it says kind comma chunk size. So you provide a chunk size and that is what going to be used in the beginning right and it is default by one it is one by default but then it gets dynamically adjusted there is only one algorithm that says that I have got so many things to do. So many threads to do them and so many things to do keep changing dynamically so things remaining to do divided by the number of processors I have.

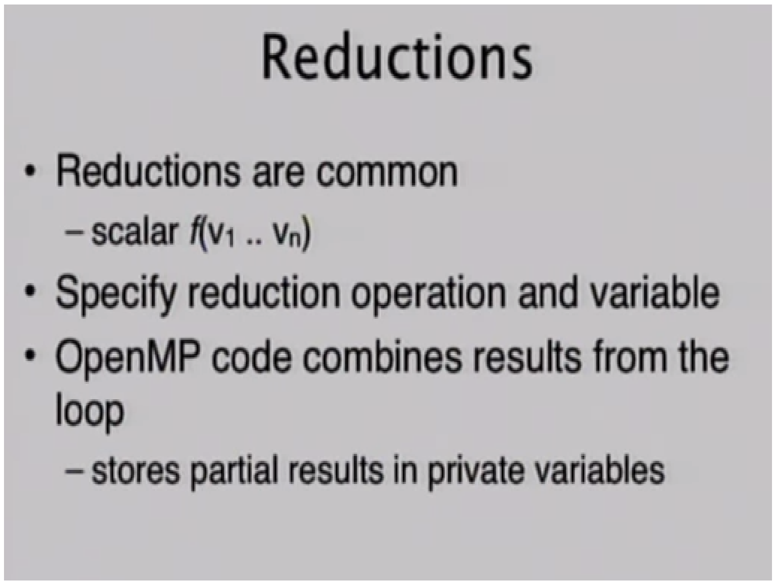
You would normally not by the construct itself you would normally have them be independent right. Sometimes there are areas you may need to be done in a certain sequence but other rest can be independent for example everybody look at the some computation which we had I have got S = 0 then everybody I adding something to it right. If suppose that something is a big computation I am adding the results of a big MONTE CARLO simulation or if so some big simulation okay.

Addition is going to be done in two or three instructions but what to had is going get done in 1000 of constructs. So I let everybody to do those they are independent 1000 things first and then in there is a small ordered section you can simply says + = to whatever you result was and then is going to get added sequence. Not necessarily not always it is supposed to be the most dynamic the most adoptable.

But again it depends there is an overhead associated with it and one is the actual implementation overhead there is more work to do to figure out how to manage it. But the other is that you may end up suppose you had this chunk of 10 example that I was taking you had 10 things to do and these 10 things can be done fast enough that you might has well give to one rather than hope that somebody else will come in.

Because this guy either is the fast processor much faster processor than everybody else right. So this guy can do this those 10 and nobody else will come ask for more in which case if you waited then the same guy will come back again saying give me one more and you will give only one more okay. So there is a load dependence on it and it is also an algorithmic overhead okay.

(Refer Slide Time: 15:16)



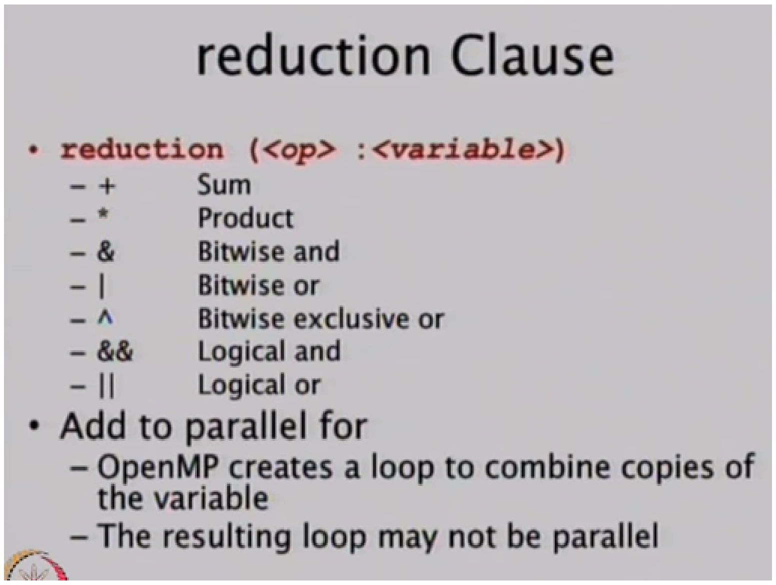
Reductions

- Reductions are common
  - scalar $f(v_1 .. v_n)$
- Specify reduction operation and variable
- OpenMP code combines results from the loop
  - stores partial results in private variables

Alright the other thing was part of for construct was the reduction which we looked at the context of but again may be its reputation revision and it is not just you see that reductions need to be done so often that they are will look at efficient algorithm to do reductions in various context also.

But in this case efficiency is not desire and not mandated by the standard it is simply says that everybody is generating some value and there is going to be some written code that compiler will generate which is going to sum up or perform some operation right at some function F which is get performed on values we want to VN with for N threads.

(Refer Slide Time: 16:17)



reduction Clause

- **reduction (<op> :<variable>)**

– +	Sum
– *	Product
– &	Bitwise and
– |	Bitwise or
– ^	Bitwise exclusive or
– &&	Logical and
– ||	Logical or

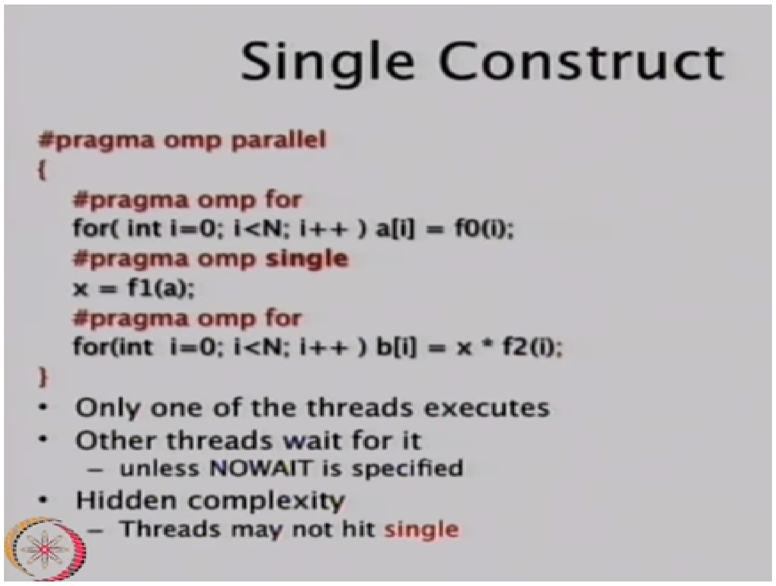
- Add to parallel for
  - OpenMP creates a loop to combine copies of the variable
  - The resulting loop may not be parallel

And in more detail these are the kinds of thing you were to do right you can say add or multiply or bitwise operate on them there are also some symbolic versions of it are not listed here. If you want to simply do Boolean and do lots of things I have tried Boolean and is already listed here that there are non-symbolic, non-operator symbolic versions of many of these also other examples of that would be min okay.

I want to find to find the min of these many things that is not an operator so symbolic things that is the operation is min and the variable is some private variable that you have then all the values will get lost except the masters. So that is where it says that in fact it says that resulting loop may not be parallel in fact is says so in the spec that there is no assumption on the efficiency of how the reduction is performed okay.

We will do algorithm to do it efficiently in parallel but it is in fact I do not even know what GCC does whether it takes all of those variable sums them up on one of the course meaning inside one of the threads or it does it.

(Refer Slide Time: 18:00)



Single Construct

```
#pragma omp parallel
{
    #pragma omp for
    for( int i=0; i<N; i++ ) a[i] = f0(i);
    #pragma omp single
    x = f1(a);
    #pragma omp for
    for(int  i=0; i<N; i++ ) b[i] = x * f2(i);
}
```

- Only one of the threads executes
- Other threads wait for it
  - unless NOWAIT is specified
- Hidden complexity
  - Threads may not hit single

I do not think it does it fact we are no done with forecast this another construct called single which is where that copy private that I was refer into is used. I have not listed all the in fact I could not listed here because I have given it as example rather than either a definition of the constant. But one of the clause is for the single construct the copy private okay and the single construct basically has the notion similar to the for ordered construct but a little different for enforce order to instruct you wanted to make the iteration happen in that part of the code happen in the sequence that the sequential code would have done okay.

Here you say that there is this piece of code that I need to be run by one of the thread I do not care which it is I do not want to replicate it for N threads okay. Here is the example which says I have a function F1 value okay and it is cannot be parallelized function F1 is sequential function each of the threads is going to take the results of F1 and then do something with it okay. So there is something that happens in parallel computes F0 for all the iteration values and then one of the threads computes F1 okay.

It is going to be available to everybody F1 is going to shared or in this case F is going to be shared. And then everybody computes F2 but multiplies with this X the scale factor is whatever their F2 values okay. You could have put entire thing in one OMP for section but then everybody would have computed after and if you do not want it F2 is for some reason high resource function.

Then you cannot parallel if you if it is high resource function and it can be paralyzed and that would be the idea you just do parts of F2 different places. But if it cannot be then you simply do it at one place the benefit is that whoever got to that singe first with got done with its said its part of the far iterations early. We will get to do the single will start on single early okay and others can simply.

That is why in this case in this case you cannot and that is where no wait there is an implicit barrier at the end of arc and if you want then no need it then you can say no wait. A would so here A is it is need to be flushed before the single but here A the entire is being used okay. And I do not know if I did intentionally that if one element of A was getting used you still have to flush it okay.

But there is some flushes going to happen by default at the end of arc so when barrier happens all of the shade variable will get flushed. So here do not need explicit flush but for this to work yes you need that flush. No it is supposed to end that far is over right now there is a single and at the end of the single at the end of single is also barrier unless you want to know it. But others are not doing it they are still there is a far region there is a parallel region okay.

If you do not want in this case you want everybody to do it because everybody is going to use the X value that this will generates if they proceeds they will get some chunk okay. So in this case you want that value if you do not want if you do not want that barrier then again you use no wait on (()) (23:10). It could be possible it just complicates the model a little bit right. So instead of saying that I want barrier where everybody has to come I want a barrier where n first can go and the remaining can get so there is more complicated semantics can be build in.

But it is rare for that have to be there yes in this case there is no wait one thing is going to be true for all barriers and here that is implicit for example in single. You can put it in an IF you can say IF OMP get thread num is less than 3 just looking up some example then pragma single do this and go up what is going to happen anybody who is beyond the 3 is going to encounter single ok.

Of course the they wont do the thing that's single is doing because the entire thing is part of the if conditional. So they threads come into single part and only one of those will actually do it the remaining 7 just skip that part it is possible to write that code. They would continue with the second one they would but the three threads that went through single will never make any progress because the other 7 will never reach the barrier right.

So there was a if conditional that says if my ID is less than 3 OMP single do this thing that only one guy as to do and then N brace and if right what happen the first three encounters the signal because they go into the if you see the OMP signal OMP single. The other set across the if okay so they proceed further there is not single for that but these three whoever of the three gets there first will do the single at the end of the single that guy will wait for the other two to come up but also other said because the group is make of ten people.

So these three people will keep waiting and the other seven will go and whatever they are they have to do or whatever they did not have to do and if they reach other barrier then they will gets stuck over there. Barrier are with respect to that group the current parallel group if you had written no wait yeah no wait there is no barriers the implicit barrier is just not there single is about the closest parallel as if there is there are more threads around it but you are only barreling for your group.

Okay that is going to bring us to the sections construct and I am going start that as the next section now we are at section 7B which brings us to the section construct. Sections construct is very similar to the parallel construct except that region which we said in the parallel construct multiple threads are created and every thread is going to run this region. And if you want different things done then you say if my idea is 0 and this is my ID is 1 do that and so on.

Section formalizes that it say this is the parallel section so you say pragma OMP sections then you create section pragma OMP section and every threads is going to run one of the sections okay. Yes which threads runs with section is unknown it is basically a switch case version of it is state of loop statement. In this case it is basically not ID dependent right for example not because that is parallel everybody is reaching right everybody is checking everybody who can right it is possible that multiple task get assign to the same thread but everybody is checking.

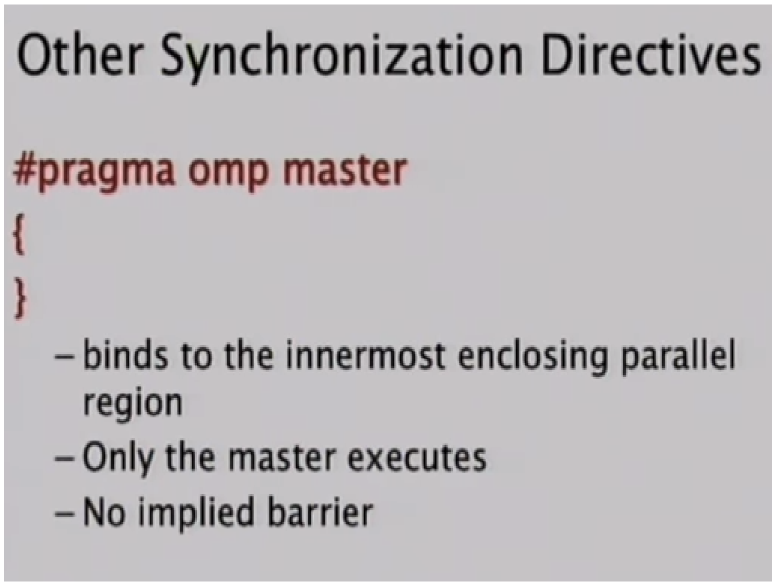
Then if your ID is not there you go to something else you ID is not there he goes or something else. It is more a syntactical difference the other things that is has is a slightly different set of clauses it has this copy private clause which parallel for does not and copy private no wait that was the single construct that was copy private the one that executes it I miss that single construct as copy private.

Because one of the N or whatever the member size is going to execute the single part and it is that threads value that is going to get written in to the masters copy at the end of it so that is how single does copy private. So that depends on whether it is for private or not if it not then somehow you initialize it the other thing that so section is somewhere between the parallel and the for.

Because in parallel there is no notion of reduction but in section you can actually have a reduction at the end of the section I think there is rise. So that everybody gets through their part of the section and then go on to the next thing no so at end of the construct does not mean end of code right each guys section will be done and it is going to do another section is still part of the same section but before anybody go out of the sections construct that needs a value.

Alright those are the main constructs for determining which thread runs what there are other constructs section is in parallel.

(Refer Slide Time: 32:05)



Other Synchronization Directives

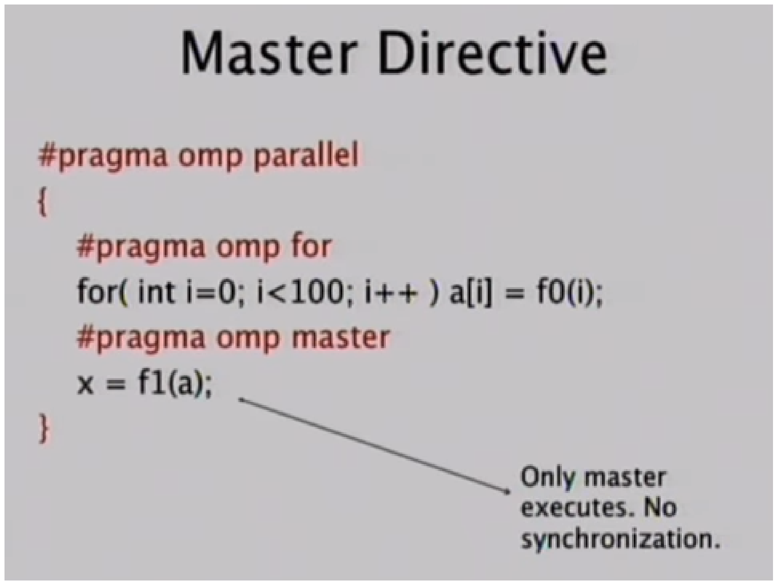
```
#pragma omp master
{
}
```

- binds to the innermost enclosing parallel region
- Only the master executes
- No implied barrier

Example of sections when you have completely so for example somebody is had firefox and internet explorer right. So you can have that in sections browsers and then have if section or the first session can be internet explorer the section may be firefox the third section will be chrome or whatever this is not white synchronization directive yet. This is probably titled little too aggressively.

It is very similar to the single construct it is the master construct it says that only master will execute this part you are master thread not any of the members which means that master is in done with other things others will have to others there is no barrier here. So others do not have to wait if master is not done then this is not done yet. This part has not been done the variables can be shared between this and the non yeah they just keep it like it does not exist for them.

(Refer Slide Time: 33:35)



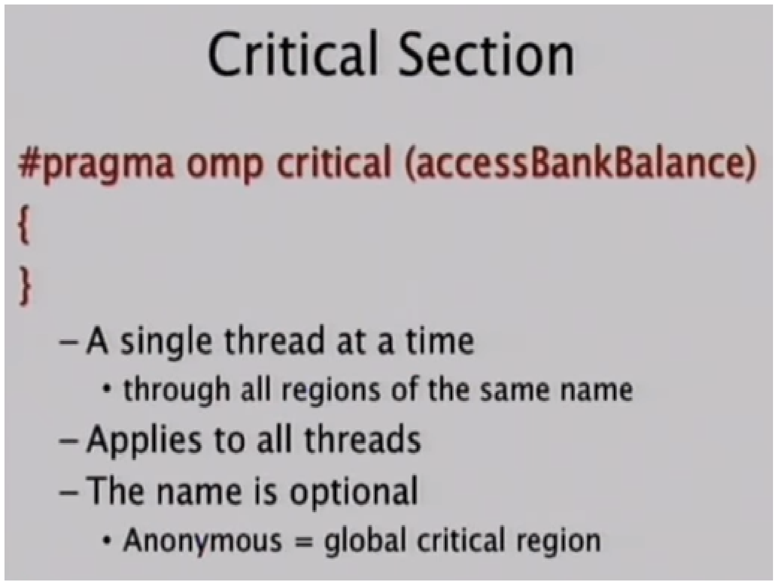
Master Directive

```
#pragma omp parallel
{
   #pragma omp for
   for( int i=0; i<100; i++ ) a[i] = f0(i);
   #pragma omp master
   x = f1(a);
}
```

Only master executes. No synchronization.

So here is an example just looked up example is basically the same code we had earlier but in this case the master does it alright.

(Refer Slide Time: 33:46)



Critical Section

```
#pragma omp critical (accessBankBalance)
{
}
```

- A single thread at a time
  - through all regions of the same name
- Applies to all threads
- The name is optional
  - Anonymous = global critical region

Now we are going to talk about this synchronization the primitives on the other threads would have computed A before. You have no idea whether they did it before master after. No the end of master so in this example in that example yeah master appears immediately after for then everybody is done there for before they do anything after the master. This is the critical section which any concurrent piece of code needs to have.

And so instead of directly saying you lock this you unlock that although there is such a facility you simply say pragma OMP critical section give it a name and then write some code it says that everybody in that group is going to do it only one at a time. It is similar to ordered excepts you do not know who is going to get there first.

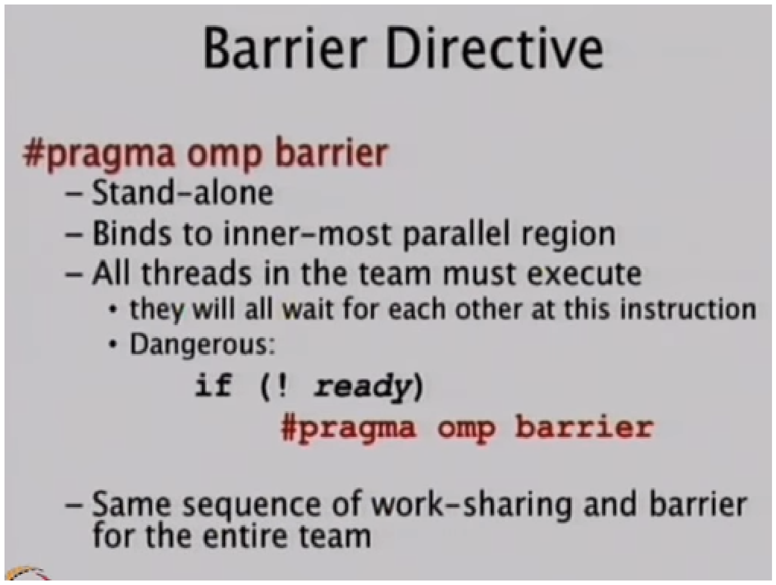
If you and in fact it is more powerful than ordered also because this is a global name okay when you say pragma OMP critical access bank balance in any in inside your program which has the name shared space. Any parallel for parallel constructs anywhere any thread anywhere will not get into to a critical section named access bank balance as long as you are here it is a variable so internally it is locked.

So that means you are putting a lock variable called access variable bank balance which means anybody anywhere else cannot get that one any other thread in you program. So for loop is undefined over here right either the for loop is out and no so it is access bank balance it not a variable it is a section so you say anywhere it says pragma OMP critical access bank balance and then there is piece of code a block of code that is being protected by access bank balance.

Nobody can get it any block of code which is protected by access bank balance okay and if you do not provide a name then it is a global critical section nobody anywhere can get into any such unnamed critical region. If you call it no name nobody anywhere can go right only that part of the only that protected yeah whatever is protected. Barrier is another synchronization primitive we have already talked about it many times.

In this case there is not much to act to barrier and in fact I do not think I do have a notion of name barriers. So you simply say pragma OMP barrier and this is not global this is only in the context of your group which means any thread in your group cannot go pass the barrier unless everybody reaches there okay.

(Refer Slide Time: 38:32)



Barrier Directive

#pragma omp barrier

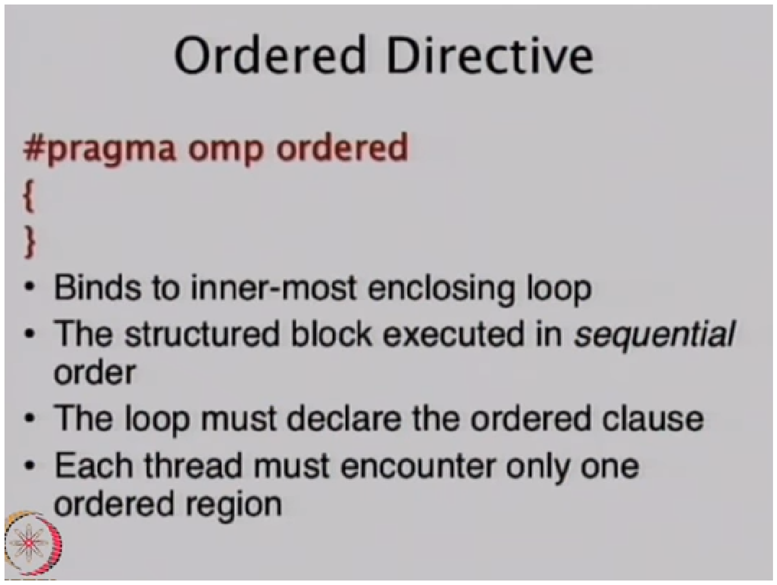
- Stand-alone
- Binds to inner-most parallel region
- All threads in the team must execute
  - they will all wait for each other at this instruction
  - Dangerous:

```
if (! ready)
      #pragma omp barrier
```

- Same sequence of work-sharing and barrier for the entire team

And again this same story applies if you put it inside if statement then you are asking for top.

(Refer Slide Time: 38:44)



Ordered Directive

```
#pragma omp ordered
{
}
```

- Binds to inner-most enclosing loop
- The structured block executed in *sequential* order
- The loop must declare the ordered clause
- Each thread must encounter only one ordered region

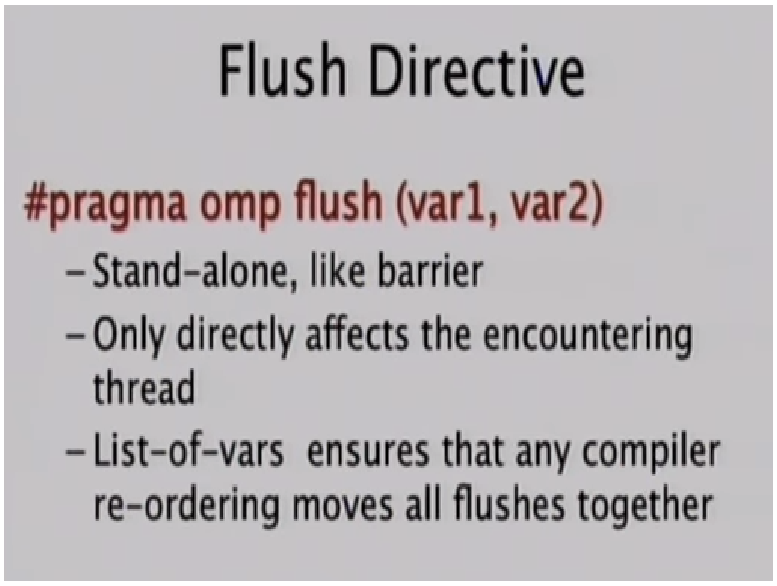
Ordered directive we have already talked about this is a another synchronization place it is inside I can be inside for region it can also be inside a sections region okay. When it is inside a section region then ordered by the sequence in the section okay you can have an ordered section which is inside a sections section inside the section there is an order.

So sections is just kind of saying that sections are beginning then each section is a region of code inside the sections right the sections is just made of section constructs and inside the section constructs you may encounter an order. Inside the section construct that can only be section constructs right so the order is one of the section constructs right how do you make the same order appear in all the section constructs by making the function call which has the ordered constructs.

So if you a make a call everybody makes a function call to an ordered to a function which has an ordered construct inside then everybody is encountering the same ordered. That is going to get run in the order lexical order of the sections okay whoever the first section whichever thread happens to execute the first section will do that part first and whichever happens to execute the second section will do that part next and so on okay.

So order as notion of a predefined order if it is coming from the loop then it is the iteration order where it is coming from the section then it is the order which the listing is no completion yes. So this is the says binds to inner most enclosing that is not complete because that is not complete for loops sequential is true because the sections also as a sequential order and that is that.

(Refer Slide Time: 41:43)



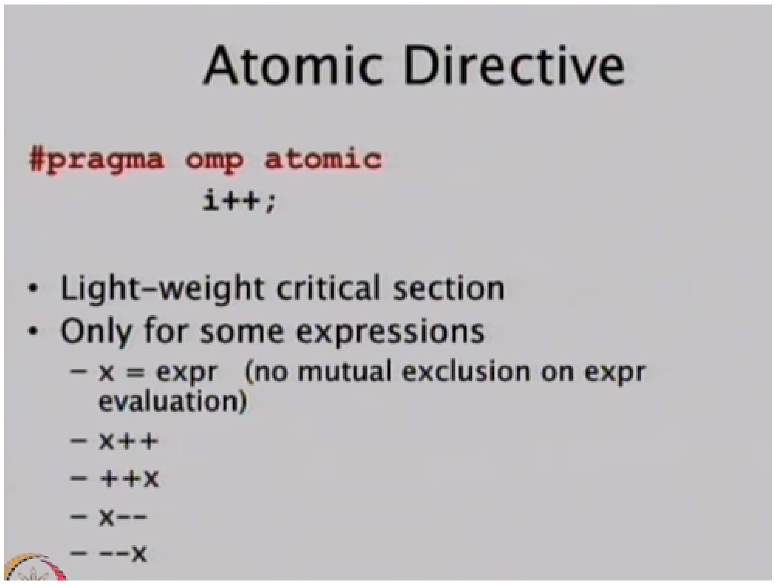
Flush Directive

#pragma omp flush (var1, var2)

- Stand-alone, like barrier
- Only directly affects the encountering thread
- List-of-vars ensures that any compiler re-ordering moves all flushes together

There is a flush directive we have already seen before this also we get a flush set and all the flush sets that have anything intersecting that anywhere common among them will have to be seen in the same every single thread okay. Atomic directive is there you can add something to something without worrying about it been broken into several instructions and so there are few atomic examples are here.

(Refer Slide Time: 42:24)



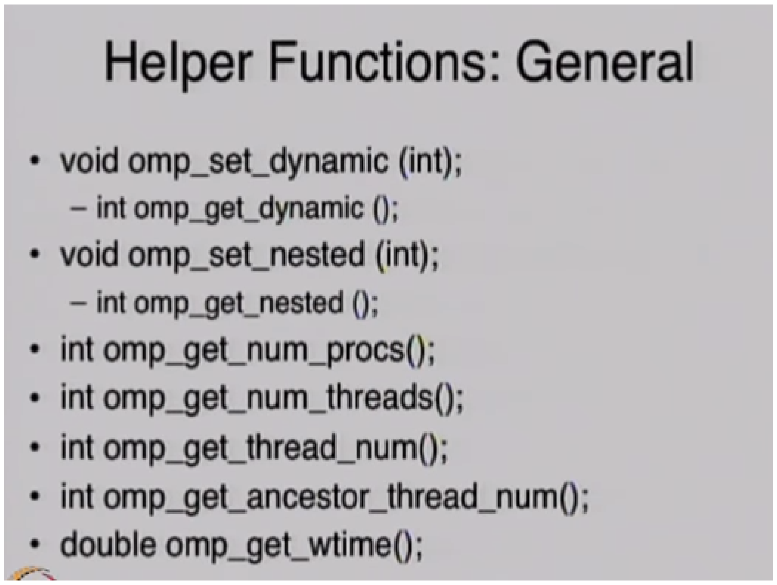
Atomic Directive

```
#pragma omp atomic
        i++;
```

- Light-weight critical section
- Only for some expressions
  - x = expr (no mutual exclusion on expr evaluation)
  - x++
  - ++x
  - x--
  - --x

So this is like having the notion of atomic without locking something what happen to the lock I think it is coming here.

(Refer Slide Time: 42:35)



Helper Functions: General

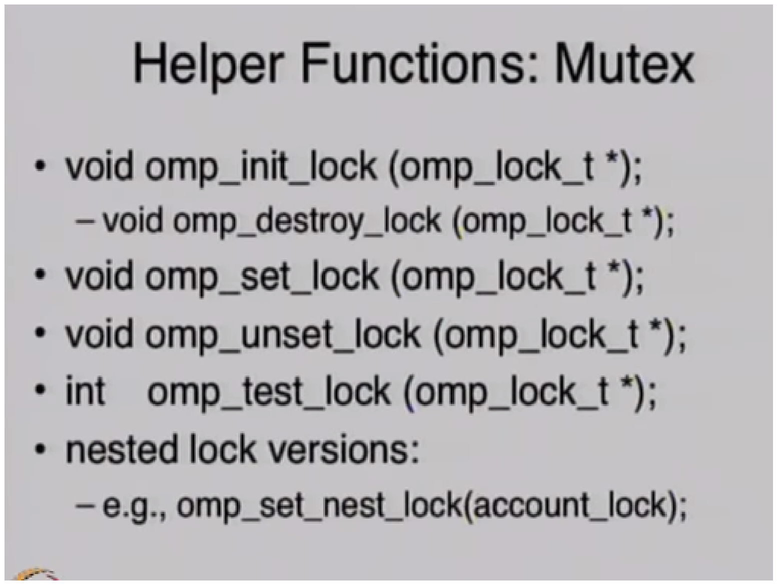
- void omp_set_dynamic (int);
  - int omp_get_dynamic ();
- void omp_set_nested (int);
  - int omp_get_nested ();
- int omp_get_num_procs();
- int omp_get_num_threads();
- int omp_get_thread_num();
- int omp_get_ancestor_thread_num();
- double omp_get_wtime();

Because they are not a part of the section they are not part of a construct but a function call you seen i believe most of the function call earlier there are many more I have not listed all of them of all of course. These are probably the ones you are most likely to use including the last one which tells you the current wall time in seconds you can set nesting enable nesting by saying set nested 1.

Get number of processor get number of threads get thread number and many more okay it is so that wall time is not fully specified in this specs it is possible yes it is possible okay. So what is only guaranteed that if the same threads makes two calls to our time then that threads are that much time between those two calls. So implementation like multi core Intel will use the same clock.

In your case when you run it on any of the threads you will get consistence results but that is not specified and the specs. So another implementation where may be two different machines where these processors are some of the simulating shared memory that need not been this. In the case of Intel architecture the same clock was to go processors do not have multiple clock generators okay.

(Refer Slide Time: 44:51)



Helper Functions: Mutex

- void omp_init_lock (omp_lock_t *);
  - void omp_destroy_lock (omp_lock_t *);
- void omp_set_lock (omp_lock_t *);
- void omp_unset_lock (omp_lock_t *);
- int   omp_test_lock (omp_lock_t *);
- nested lock versions:
  - e.g., omp_set_nest_lock(account_lock);

Here are this are the functions that I was thinking of you need to declare a lot so that there is a special lock type OMP lock underscore T and you need to declare a variable of this type and you need to initialize it by calling OMP in it clock and (()) (45:13) and then you get to set lock and unset lock okay. And so if the two people trying to set the same lock then only one is succeed.

There is a slightly weaker version of set next nested lock where if you hold this lock then this will succeed the column will succeed. If you call OMP set lock on the same lock twice same thread cause twice then second one will block because somebody holds the lock it does not hold that it does not care that hold it. But if you do it with set next nest lock then if you hold it then you will get it.

And internally there is a counter it says you got it three times you do have to unset it twice if you do unset nest lock you do two unset nest locks but if you simply do unset lock without nesting on the same lock so you call you are so I think it is non-confirming to mix the two although I am not 100% sure about that which means that the intent is that you set lock and you would not set lock or you set next lock as many times as you like but you set unset also as many times that many times.

This is more general you do not want to create section somebody in that section may want to have the same lock and in fact you may be multiple locks depending on the condition for some reason you need three locks to operate on three separate things in some part. So critical section is bit rigid although much more convenient a simpler version simpler interface. But locks will give you more power you can set a set of locks and then unset some of them.

So you hold five locks and not hold the other three critical section is basically a structure version of lock if you will. You cannot nest the section critical may not be nested ever inside a critical region. If it is a different way it is different they are not related to each other or if it is no name then does not have a name then again it cannot be nested and just because you are not nesting does not mean you cannot get into their locks but at least it helps to not get into data’s or rather if you nest it it helps to get in read locks all right.

There are also this quotient a notion of closely nested okay and the next set of four things are not allowed to be closely nested first is never nested and closely nested means one after the other. For example when you say OMP for that means it is nested with in respect to each other or sections that means it is that the second sections is nested within the first sections. What is an example of an not (()) (49:08) you say sections and then in one of the sections you have an another for region another parallel region.

So that thread is working on five ball and then in there each of them for a more sections okay. So you basically cannot closely nest work sharing or barrier constructs inside either another work sharing construct or critical region or an ordered construct or a master construct. Work sharing is for sections meaning that there is code to be run and the numbers will run parts of it you cannot have a master region inside a work sharing region can only inside the parallel region and you cannot have an ordered region inside a critical region which make sense let us stop this section.