


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

**Module No # 02**  
**Lecture No # 06**  
**Open MP (Contd.)**

(Refer Slide Time: 00:27)

Environment Variable	Ways to modify value	Way to retrieve value	Initial value
OMP_NUM_THREADS *	omp_set_num_threads	omp_get_max_threads	Implementation defined
OMP_DYNAMIC	omp_set_dynamic	omp_get_dynamic	Implementation defined
OMP_NESTED	omp_set_nested	omp_get_nested	false
OMP_SCHEDULE *			Implementation defined

 Also see construct clause: **num\_threads, schedule**

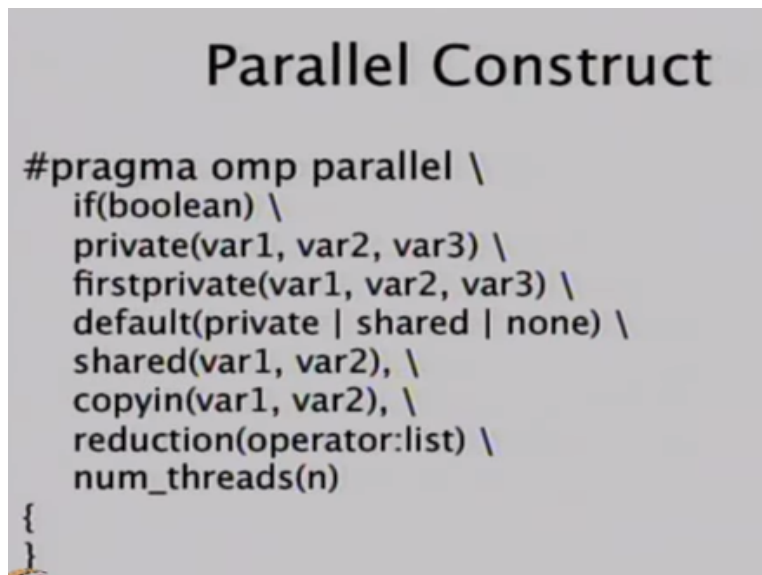
Control for the struts that we were welcome back to the strut again to look at the details of it but the basic strut if you recalls at pragma OMP parallel do this okay. And there are certain things need to be controlled one of them was how many threads to start okay. So that kind of control is being shown in this table there is control there are three difference places where you can specify varies controls one is in the environmental variable okay.

You change the environmental variable change on the program something else happens this time the other is actually calling some function which sets one of these. For example it may sets N number of threads to run in parallel region and then later on you can set some other parallel depending on what you are trying to do these are light weight job may be you can draw more of them heavy duty jobs you want to run only few of them. And the third is that the construct itself okay.

Typically the environment variable will be read first when you make function called that is going to overwrite anything that the environmental variable set and if you specify at the construct then it is going to overwrite any function called that you might be okay. So here are this some of the most although you would not yet understand all of it none threads you will understand okay. dynamic will I will talk about when you get to it.

We will has something to do it how the work among these different threads gets allocated similarly I scheduled is also related to the same allocation and nested says whether you can nest parallel regions inside other parallel regions okay again you can set up by setting a variable environment variable or just call enable set nesting improve.

**(Refer Slide Time: 03:00)**



```
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)
{
}
```

Now let us look at this parallel construct and these are again the construct itself comes with the following 10 times you say ash pragma OMP whatever name of the pragma is followed by some parameter as well which we call clauses number of clauses. So this is the parallel construct or you make parallel pragma and the clauses here are if private first private default shared copy in reducing in none threads.

Each clause in turns has some parameters and you can list any number of them some clauses can be repeated some clauses can appear only once. For example you cannot say num threads more than once you cannot say 5 threads and then later on in the same construct you run 10 threads it

would not make any sense okay. But you can say these variable say some later on other variable shared what you cannot do is that this variable is shared and later shared the same variable is private.

So there are to be some consistency let the compiler is going to enforce let us very briefly look at behavior that these process control if is basically conditioner for whether or not to run these region as a parallel region okay. You can say if I have too much work to do or if the CPU's are not overloaded by somebody else running a big then run in parallel. Otherwise just use it single core machine okay.

So at any parallel construct you can conditionally make that construct parallel or leave it sequential. Private says these are the variable that are going to be private and shared has these are the variable is going to be shared I said something about how the private variables need to be initialized and need to be collated okay. Initialize business can only one way which it is supported right now it is first private it says that this private variable has been declared.

So everybody has to have a copy but the masters copy is somehow the special copy and everybody is going to initialized with masters copy. So it is like a broadcast of one value to every processor whenever whoever encounter the parallel construct is the master he is also going to have the copy of that variable everybody else is going to get the same value. Either using copy in or it is variable that is declared outside it is just become private here right you it is a variable that may be declared outside at this point making N-1 extra copy of it.

In which case master already has a value no you say what is what anything defined inside the scope is by default private but anything outside you get to choose. Automatically shared no you can make copies of it by using first or just have copy replica that having addresses but not initialized using this is calling it private okay for example when you use a variable as a for loop when it is going to be private and everybody is going to initialize their own.

They do not need anybody else to initialize it for them so need not always be initialize but if you need somebody to set some value before you being then you do first private or you locally get it

from some shared into this private various ways in which you might have it. Default is basically saying that everything that is cleared inside is not private but shared okay. So you can change the default so either you can say this variable is shared right or you can say the default is that the things are shared so everything is shared.

Everything that is being used so let me verify when you declare something inside the scope this is not talking about those variables this is talking about variables that exist at this time when you making the parallel or construct right and you get to choose whether they are going to be shared or private. Ideally you would have everything single variable that is accessed inside this construct be said either shared or private okay.

Instead of doing that you can say the default is shared but these three are private okay but you do not have to this is just sugar coating more than anything else not giving you any extra power. That is private then you can cannot make it sure copy in is similar to first private except it need not come from the master it can come from one of the anyone of the threads. First private is it is a private variable meaning that everybody is going to get a copy but all the copies will be initialized with the masters value of that variable.

You do not mention right basically this is a variable that was declared and you said private. So whoever encountered this construct is the master he is going to keep that address and -1 new variables addresses so basically that variable .1, .2, .3 will be created and everybody will get will actually be referred into those different addresses when they referred to this variable but inside the compiler is going to introduce a small loop that says copy from that address to that address . 1, 2, 3 in a loop okay.

Yes it is there is no group share it is part of copy in construct itself the copy in should have something more. No destination is everybody else right the source need to be specified somehow I do not I have never used copy. I believe that copy in the essentially that these are the variables that will get copied somewhere and there is some other construct that let you put in these very or set in these variables from a set of other thread okay.

I am going to take a look at that specs to make sure of the usage how that copying happens but it is essentially some generalization of its private none threads is how many threads to run. Reduction is the clause that how to collate the results before this area started there as one variable N-1 copies was created after the area ends again will be one variable. This all the .1, .2 will go away the allocated so should those variable get lost values get lost or should be somehow folded in into this main variable the masters copy.

And the reduction will talk a little bit more about it then it is a generic primitive that any honest to goodness parallel program will use somewhere it is essentially saying take all things and apply some operation in it typically you expect this operations to commutative if it is not commutative then there is going to be some understanding of what order that is going to be private as well as associative.

And the kinds of operations that are going to be supported for example for open MP it says take all of them take there some that some goes into the master okay. I will take all of them take their maximum value and maximum goes into the master okay. So max had multiply these are the types of operations that you would see in a reduction of a reduction operator with the set of variables that need to be reduced. In fact that need to look at how to do reduction efficiently.

Open MP in specs does not say anything about how the reduction will get which means that reduction might be getting done one of the course if I have got the 100 course then 1 core doing 99 operations is going to be again depends on the level of operation the complexity of operation but 1 core doing 99 operation will typically be slower than they somehow together doing the operation.

And it is that together when we typically assume that they are committed to as well as associative okay. We will come back to it there is some common sense question there is some common sense restrictions you cannot jump into the parallel construct you cannot say go to some place into a parallel construct. You cannot say from a parallel dual construct go to some sequential area with would not make sense.

The clauses have to be well defined right when you say this is number of threads this is something is known at that point I should evaluate something that is positive value. Typically in an implementation there will be max how many threads you can create when you have a positive value that is greater than that max it will just get truncated. The other thing that you would again expect that there is only one if clause only one number threads to create clause.

If there is two clauses and one evaluates to and the other evaluates to false what I suppose to do. So these are some common sense restrictions. So it depends on the underlying threads implementation every thread implementation will because it is maintaining a table for every thread remember every thread it is going to have some limits. Want to say you have a 1000 threads not just memory there may be for example it may say I have got a variable that is an hint for number of threads.

This is over exaggeration but if I did have that then it would mean that I cannot have more than 2 to 32 threads okay. Things where restrictions of that type which are system driven system will tell you how many threads are there at a give time there is a different variable that is how may threads will actually going to run that will by default be the number of processors you have but you can change either by environment variable or by calling a function or by setting it over the other.

And even you can query how many processers there are and then do something different based on the number. You would not typically set it to a specific number of threads unless you know that there are only two things you can do which can be done in parallel. So in if you are really have a task that is broken into many subtasks that is to be done in parallel and then you will query and say I have got it is generally not a good idea to run only as many as threads as there are number of processers.

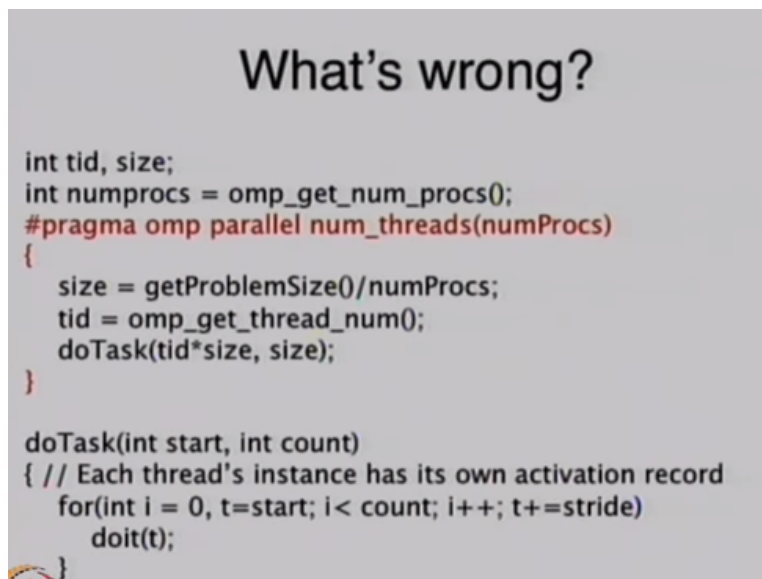
How many more is a little system dependent so for example we talk about CUDA it is got 16 processers. But if you run 16 threads at a time it is going to be incredibly slow similarly on a CPU of you got 4 core you would want to run at least 4, 5 times than may be even 10 times but

that may be a over any kill. Anything probably 10 minutes going to slow down so again that is again architecture and system dependent.

It will generate a normal C code locks you have to take care yes right so flush is it is going to make sure that they are getting ordered but if you want this flush to happen before that flush that is you job. That will be a compiler no if you get the twice it can know that that is when it is getting encountered so it will probably insert code that value is between this number and this number run it otherwise do some exception crash it talking about the thing but something.

That is run time now does not say that all of this is happening at compile time it is basically in fact open MP does not say anything it says if you do any of these things then it is a non-confirming programming with which means I do not know what is going on it is system dependent okay.

**(Refer Slide Time: 20:06)**



```
What's wrong?

int tid, size;
int numprocs = omp_get_num_procs();
#pragma omp parallel num_threads(numProcs)
{
    size = getProblemSize()/numProcs;
    tid = omp_get_thread_num();
    doTask(tid*size, size);
}

doTask(int start, int count)
{ // Each thread's instance has its own activation record
  for(int i = 0, t=start; i < count; i++; t+=stride)
    doit(t);
}
```

So we have understood some of these things let us take a quick look at an example and is something wrong with that example because it says tasks what is wrong. And this is exactly the kind of thing we are talking about I have got bunch of things to do I have figured out how many processors I have am going to do that many places okay.

So it declares the thread ID and a size of the task it is asking how many processors do I have by calling this function OMP get number of blocks and then it runs a parallel pragma with none threads equal to num props and this is not needed because that is default. But I want to do it and inside the pragma it is looking first at how many things I have get problems sized how many tasks I have it sees that I have got number processors is going to divide the number of tasks by the number of processors.

And that is the size of number of tasks that each of those of the processors it is going to get and this assuming that they are all the same size and integer. No the slight new task is happening here it is a construct meaning that it is a structured area where it is going there and coming back which is that part of this. Go to means go there and never come back right so that is the different things.

Otherwise it will be very useless pretty you cannot make any functional cause everything else to be physically there that would make sense okay. So it is look like how many to assign for time being it is just assume that it is just a multiple of the number of processors I have and then it is going to call this function and do tasks starting at ID 10 size. Size elements from there and then will do task will be running on different processors each instance of do task will be running on different processors.

And do task will simply a for an I into equal to 0  $T = \text{start}$  I as a count I have a simplified it I have a slide earlier. So there is no need for a jumping through to a slide it is just going to go  $I = 0$  to  $I++$  do it.  $T += \text{stride}$   $T+$  also do not worry about how T is being calculated sits there was some information about we have to jump through data to get to where you need to be. So I will actually fix this code this that T does not make sense.

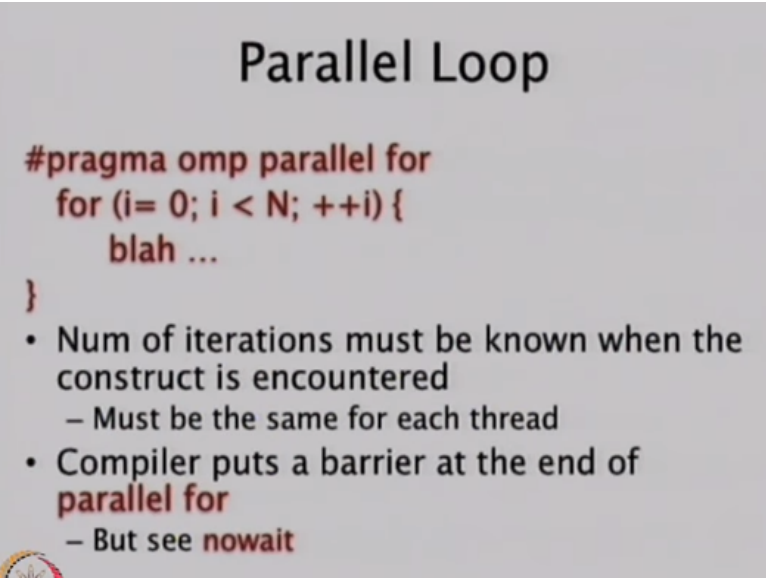
Just imagine that it is  $I = \text{start}$  to  $\text{start} + \text{count}$  do I so TID is shared right that is the default as a result of which everybody is going to try to write into that variable their value and one of them will succeed I do not know which. What else there is nothing that is necessary wrong because in basic thing here is that TID needs to be private so that is where the problem is then and this what



Just be cleared inside and it automatically become private that is the default automatically you could simply say private TID still declare outside another way of doing. But what you would really like is because this has everybody started asking if I have and things to be how many processors to run and how to allocate them that is expected thing you could want to do in a parallel programming I have a got a bunch of things to do.

I have got a bunch of processors to do them at I just want to say do those many things at these many processors okay. And so instead of writing all of this loop I should simply be able to say I have got these many processors I have got tasks go to them and that is what exactly this parallel loop is for. I am going to stop in the middle of the slide I wanted introduce it before stopping.

**(Refer Slide Time: 25:25)**



**Parallel Loop**

```
#pragma omp parallel for
  for (i= 0; i < N; ++i) {
    blah ...
  }
```

- Num of iterations must be known when the construct is encountered
  - Must be the same for each thread
- Compiler puts a barrier at the end of **parallel for**
  - But see **nowait**

It is in this case there are two different pragma's that are getting combined so it is saying pragma OMP parallel for. For is not one of the clock process of the parallel pragma so think of these as new pragma new construct it is called parallel for okay. And now you say this is the loop for I=0 less than I less than N do it. And now the compiler is going to do right that loop that we are writing along with all the extra baggage that comes with does not devisable.

The number of task is not devisable by the number of processers yes so the compiler is going to expand to some version of the code we had earlier okay so compiler is going to make sure that I = 0 to size is being run at one place size + 1 to 2 size is being run at second place and so on. If

iterations are not independent then would not do this. But there are construct that you can still use for the parts of iterations that may being independent okay.

It would simply adding number to the previous generated sum for example you are summing some values then that this would not make sense it would not be done in this fashion you would have to come away that different algorithm or all together to add N numbers. So let us stop for this session and will being in just a minute but I will answer your question. So let us continue we were taking about this parallel loop I have not officially introduce the pragma yet but this lead into the pragma.

That means it is going to make to the same so that depends on the global context and the memory has somebody else is written to the memory okay. And I have also written then it is undefined which are those two actions is going to happen whether the memory value is going to come to me or am I going to wait for it. Because two peoples have written essentially ignore the flush for being what has happened.

This fellow has written it was in local memory this fellow also written it was in local memory but then got flushed. As the result of which these two get top of it so the two rides that are happened which are not being related to respect to each other. So that was one of the consistency rules that two writes happen which are not synchronized then you do not know there result is undefined.

If this one does not write yeah then that flush will update the local copy the reader's copy so flush is not a function the flush does not anything in fact you cannot do flush in any context because it is only place where a macro can exist. It is not a proper function alright it is in effect of macro so it is going to create code for you it is going to do all that. Because there are two flushes and either this flush as to before that flush that flush has to happen before this flush because they have their same set.

And whichever you order the things that before flush automatically gets ordered before the other flush. So there is something got ordered before the flush and the axis that was the checking what

ordered after the session. So that axis before that flush before this flush before that check so that made the relationship between these two. Flush will write to that memory it is basically make those memory copies the same somehow cache they get from the memory.

Here N does not turn in the number of threads N is the number of task you have number of threads again that will be in the clause. The clause will be very similar to the parallel clause you can also get it from the memory processor you may get it from the function call you may call earlier all those things we talked about for the parallel construct all apply okay. Will talk there is much that needs to be discussed it is not clear how the iterations are going to get allocated through K processors okay.

But again the basic rule is things to be known similar to parallel constructs when this for is called I need to know how far is far is going to run. How many iterations at the time the function is (( )) (31:30). At the end of this for construct there will be barrier placed by the compiler and then fork and then everybody joins and then move on. In this parallel fork construct also a barrier is places which means once every of the threads is done then the master will proceed.

But here there is notion of nowait meaning that if you get to the end of the far you can proceed. You need to proceed to another alternation of the cost. Otherwise what is going to happen is that does any iteration then everybody done the another iteration then everybody one the other iteration okay.

**(Refer Slide Time: 32:31)**

## 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
```

We have to write nodes just look at the constant so nowait is one of the clauses for the far constant it has the same private first private it has the last private instead of the reducing not in addition to the reduction. What might the last private be? The last iteration value right in the parallel for all equal but here there is notion of ordering among the task there is reason  $I=0$   $I=N$  or  $N-1$  is the  $N-1$  fellow was the last.

Whatever value he wants to write ends okay if there are you are doing the iterations that is updating something. So what would have happened in a sequential case you would have kept updating something and the last things updating would have being the final value. So it is saying that even if that guy did not finish the past that is going to determine how this is going to updated all okay.

It is not a specialized collation right so one of the variable wins out versus the data being combined in certain fashion. There is another clause that is here was not there in the parallel clause which is schedule or order yeah I missed the order. Ordered is also something that is part of parallel for the ordered so we will talk a little more about it later on. There is also a construct call ordered alright this ordered is only saying that there is an ordered construct somewhere in the body of this loop okay.

And that ordered construct does what was asked that ordered construct as to be executed in order okay. You cannot be two different threads cannot be in that ordered part in that same time it is kind of some critical section it is not atomic. So inter leaving is not allowed it is implicit locking yes so that ordered it is basically telling the compiler to do something later on it is some initialization that need to be yet.

But there is a separate ordered construct that this is the section everything can be done independently but this is section that updates some values or do whatever doing something that is shared okay. So even if the entire loop is not doing independent things that do not depend on each other. The section is the only thing that's shared and everything can else happen in parallel but when we get to the section wait for the turn.

Originally that is what was intended the way it is implemented the way in fact I think it is also specified it is that it is a critical session meaning that at this point that parallel thinks are gained to get ordered it need not be in the 0 to N sense okay. Which means that if you need them in 0 to 1 ordered even if only in that section then you use in some kind of locking and to cannot get locking until once lock has been it can acquire one lock which only one can release.

Then it becomes at the end of the ordered does not mean that it is barrier okay so it could not be because it is only one person in the ordered session. One person go to the order it proceed and does whatever else to do and then as soon as that precedes that next person can come in. That the set of instructions is not atomic right because yes in fact remember that this parallel for that may be nested in another parallel which does not have parallel for loop at all.

So there are other threads which are kind of not your brothers but cousins who have nothing ordered they just keep going through that. You can turns out you cannot have to orders inside one loop allowed only one order this order is with respect to this loop yeah so this is everything as the binding context and the order is binding context is the simplex loop and the other thing that does not exist in the parallel constant is the schedule.

And it is basically a way to determine how to allocate or way for the user to control to specify how to allocate the task among the processors. I have got some end things to do some P processors to do them at which N which fraction of N is going to go to which processor okay. This loop as to be a economical for loop for some where you do to initial values that variable is less than some final value and that variable ++ it can be + = something.

So minor variant are allowed that entire grammar is given in the specification what the grammar for that far is allowed to be. And you cannot breakout of that loop okay it is similar to the jump thing. Restriction wise again you would want remember although in the example I had I said pragma OMP parallel for. But you if you look at this it is just pragma OMP for which means you are expecting used to be inside parallel constants.

So you do a parallel and somewhere you do parallel for inside for somewhere you do parallel OMP for which one so that was so what I had was pragma OMP parallel for which is the shortcut for pragma OMP parallel pragma OMP for and for loop okay. So when the outside parallel construct was encountered and threads where created okay now inside that a new set of task description are provided.

Suppose there was no parallel I said parallel for and then have a parallel loop say parallel and for loop what would happen everybody in that group would be doing it from 0 to 1. When you set of that is this is not a independent piece of work but a shared work okay and so now you have the same members of the group allocate the space. No repetition of schedule no wait ordered is this one thing nowait there is no breakup because everybody is doing everything unless they encounter this parallel for.

So when you say pragma OMP parallel for it is a new parallel constant okay it is all only in the construct of parallel. I will explain that yes everything is that for is allowed there so this is what happen I have a parallel construct at that point 10 threads were created okay there each doing something each is going to encounter up an OMP form not a regular for loop OMP form followed by actual loop. Earlier it would have said that is for me to do and going to do this loop

now everybody is going to set am not going to do that entire loop let me figure out what I should do okay.

And this is how all this schedule and all parameter are coming and then execute only section of that problem okay. So that is the way to get it happen. That schedule is not required over there because until the encounter this for there is not subdivision of work everybody is doing everything every when one or any of those threads encounter this they have to written how much should I do okay and then for clauses will help it figure that out.

Yes no new threads unless a parallel or construct nobody is going to distribute right there is now central control over here. For us thread to reach there oh I am thread number 3 and this is parallel for and here are the parameters for scheduling and everything that means thread number 3 from  $I=10$  to 30. So it is going start at higher 10 okay and last thing that it says oh wait I will this would not necessarily make sense because it is schedule size and chunk size and it says that chunk size must be known just like all those others constant things something that does not depend on the loop variables than has no side effects of all those that.

We will come to that shortly I will kind is what type of distribution to do and chunk size is how many chunks. Instead of saying you take  $I=0$  you take  $I=1$  you take  $I=2$  it might make sense you go from 0 to 5 you go 0 to 10, 6 to 11 or whatever that is an option it says it can either say only kind or kind form of chunks bracket means optional that is the Unix terminology or regular expression terminology.

**(Refer Slide Time: 46:18)**

## Firstprivate and Lastprivate

- Initial value of **private** variable is unspecified
  - **firstprivate** initializes copies with the original
  - Once per thread (not once per iteration)
  - Original exists before the construct
- The original copy lives after the construct
- **lastprivate** forces sequential-like behavior
  - thread executing the sequentially last iteration (or last listed section) writes to the original copy

Okay here is some more explanation of the first private and the last private I have already given you the basic idea but this is going to be there in the slides for you to look at later again flush private is going to initialize it and the last private is going to leave one of the values that is at the end.

(Refer Slide Time: 46:42)

## Firstprivate and Lastprivate

```
#pragma omp parallel for firstprivate( simple )
for (int i=0; i<N; i++) {
    simple += a[f1(i, omp_get_thread_num())]
    f2(simple);
}

#pragma omp parallel for lastprivate( doneEarly )
for( i=0; (i<N || doneEarly; i++) ) {
    doneEarly = f0(i);
}
```

Here is one example the first example you have parallel for which means that for loop is being done in a shared fashion. And the first private variable is simple which means before this being simple was declared simple was set to 0 and what is happened is everybody get simple = 0 and everybody is starts adding to it is simple it is section okay it is summation it is part of summation but still there is so any copies of input.

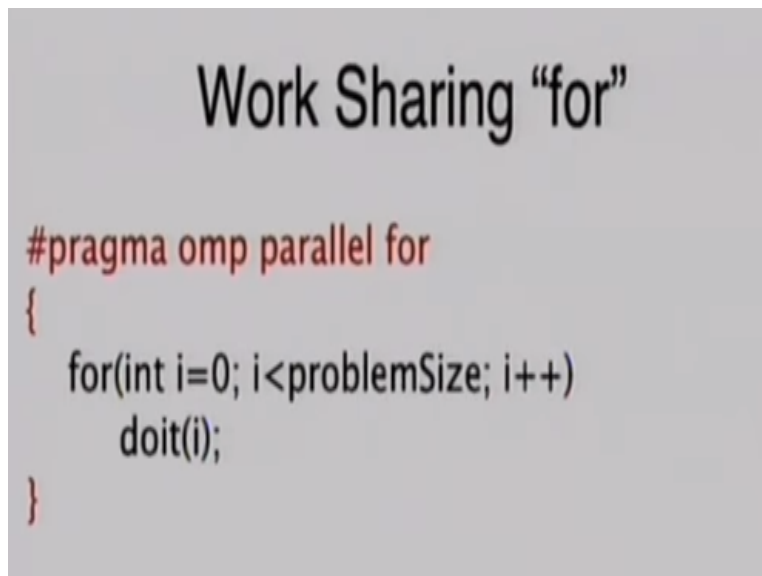


So at the end of it somebody have to collate it that is not in the first one okay it is hidden name I have to simple but it could have been done using last private. The second example now actually sorry this is the intent here will the simples get added and that would have been reduction not a last private. The last private example is the second one where there is some variable called early.

And things as got something to do and but the various and condition which says that I know now the answer I do have to go through may be am searching for something and as soon I found it I have to continue it through the rest of it okay. So it keeps running the loop running every time last done early as happened and then early is being is set by again some function that says I am value number this said this to done early okay.

This is the not the great example last private so this can be done much simply I think it has not allowed it is clearly said this last private that means everybody's is done if anybody got done early then something is wrong here and we will go through another example of last private when we come to the end of the description of the examples there also we will look at the last private there but the main thing I want to point out was what happens to that program.

**(Refer Slide Time: 50:27)**



```
Work Sharing "for"  
  
#pragma omp parallel for  
{  
    for(int i=0; i<problemSize; i++)  
        doit(i);  
}
```

The program which I had shown earlier with just a parallel construct this parallel and manually figuring out who is going to do what and that can be using using the parallel is for by simply

writing this and that is basically the idea that something instead of having lots of people want to do it again repeat it every time provide syntactic sugar on top of the parallel code okay.

We are going to stop here what I want to begin with or what the thing that talk about next is order schedule means how do you decide who runs where and how much at a time okay any questions? Flush between what between the cache variable than that the code being generate flush does not determine anything right code that is being generate in place of flush that code determines.

Because the flush is has some variables that go with the flushes somebody says that this have been arrived flush on this variable before after you did flush meaning that you are out of date. Yes exactly but in software okay anything else it is similar to (( )) (52:15) yes it is not strictly cache coil it is not providing any cache flow that is right but in order to determine what to do that dirty bit can help you to figure this out.

It is not problem in the model because it is very easy to provide cache coherence by not having cache you misunderstand me. I am not saying not having a hardware cache there is a notion of a cache in the programming model it may say that there is no notion of cache meaning that sorry there is no notion of having a different temporary view of some main memory which means that you let the hardware to take care of keeping cache coherence.

If you have something in cache you can make sure that you are running on a piece of hardware that is going to keep that piece of cache coherent from some other processors piece of cache this is going one level above it and saying what if you do not allow that anything else.