

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

**Lecture – 35**  
**Advanced Task handling**

(Refer Slide Time: 00:01)

```
/* Task dependencies */  
/* #pragma omp task depend( in: <list> ) depend ( out: <list> ) depend ( inout: <list> ) */  
Example:  
  
#pragma omp parallel  
{  
  #pragma omp single  
  {  
    int x, y, z;  
  
    #pragma omp task depend( out: x )  
    x = init();  
  
    #pragma omp task depend( in: x ) depend( out: y ) ←  
    y = f(x);  
  
    #pragma omp task depend( in: x ) depend( out: z ) ←  
    z = g(x);  
  
    #pragma omp task depend( in: y, z )  
    finalize(y, z);  
  }  
}
```

The slide includes two dependency diagrams. The first diagram shows a simple task A pointing to task B, with handwritten notes:  $x = \text{init}()$  and  $\text{finalize}(f(x), g(x))$ . The second diagram is a flow graph with nodes: 'init' (top), 'y=f(x)' (middle left), 'z=g(x)' (middle right), and 'finalize' (bottom). Arrows indicate dependencies: 'init' to 'y=f(x)', 'init' to 'z=g(x)', 'y=f(x)' to 'finalize', and 'z=g(x)' to 'finalize'. Underlines in the code and arrows in the diagrams highlight the dependency clauses.

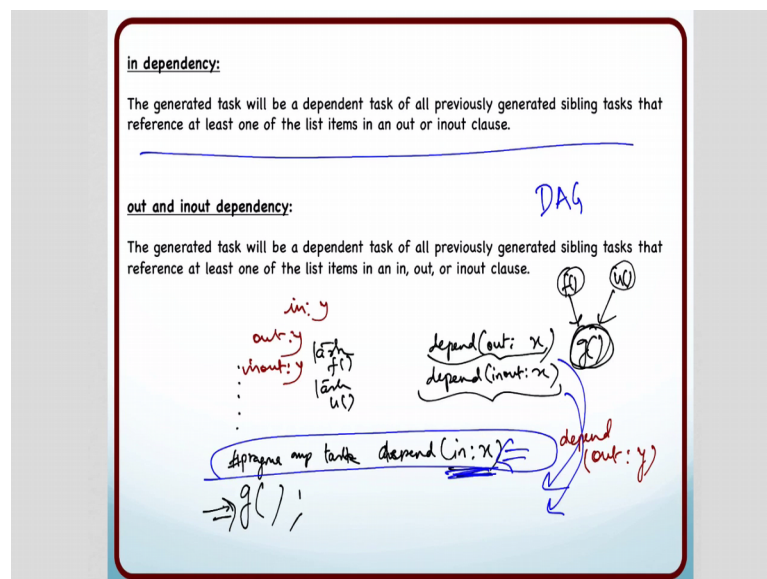
So I want to discuss a little bit more about tasks. Tasks also allow us to capture dependencies; what is the dependency? Dependency basically says that if I have a task A, I want task B to only execute after task A has completed dependencies are supported using hash pragma omp task depend, right using the depend clause what is the syntax. So, you can say depend in and specify a list of variables you can say depend out specify a list of variables and depend in out and specify a list of variables.

And what is the meaning of in out and in-out think of it this way, right in means that I am going to use this variable as input and out means that I am going to write to this variable; I am going to produce this variable as output; I am going to update the value of this variable; the function need not necessarily update that value, but as long as you specified that in the hash pragma omp task depend, openmp will take care that it follows certain rules and inout says that I am going to take this variable as input as well as right to it update it.

How do you use these, right; let us see a simple example. This is a parallel region and inside that a single thread does some work and its going to spawn some task which are then going to be handled by multiple threads, right that is the way tasks work; what am I wanted to compute over here; what I want to finally, compute is I want to compute finalize f of x g of x; that is what I want to compute. So, how do I specify that? So, this is the way to specify it. So, I have these 3 variables x, y, z. Let us say that x is initialized to something first and then all of this happens, right, then I call finalize f x g x x equal to in it is put inside a task which initializes x. So, it is going to specify x is output, right because it is going to set the value of x.

Now, that value of x has to be used by f x and g x. So, there are 2 mode task created y equal to f x and z is equal to g x and these tasks are going to depend on the value of x. So, you say in x and this also says n x and what is this going to produce? It is going to produce output y and this is going to produce output z and finally, this task takes as input y and z.

(Refer Slide Time: 02:52)



So, what does openMP do in this case? So, let us understand the rules of openMP and then come back to this example. So, how are in dependencies handled. So, the generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an out or in out clause. So, what that is saying is that if I

have a task hash pragma omp task and I say depend in colon x what it is going to ensure is that all previously generated sibling tasks what are sibling tasks?

Student: (Refer Time: 03:33)

Tasks created by the same task; whatever task it has earlier created and if it is at the top level than the thread, right.

Student: So, there is a concept of a parent task.

Yeah. So, whenever I create a task and the parent task.

Student: So, I mean a task would know somebody's parent.

Yeah, what does has pragma omp task were do? It waits for all task to complete all the tasks that it has launched.

Right; so, it has the notion of that these are all the tasks that I have launched. So, all those tasks are called sibling task; sibling means what; brother-sister, right. So, all those tasks are sibling tasks.

What it is saying is that this task will be a dependent task this task whatever I am going to do over here, I am going to call a function g. So, this g is going to be a dependent task for all the previously generated sibling tasks whatever tasks have been created before this by the parent task that reference at least one of the list items in an out or inout clause depend in their out list they have x, right and similarly if some task has x in inout, then this task is going to be a dependent task of that task.

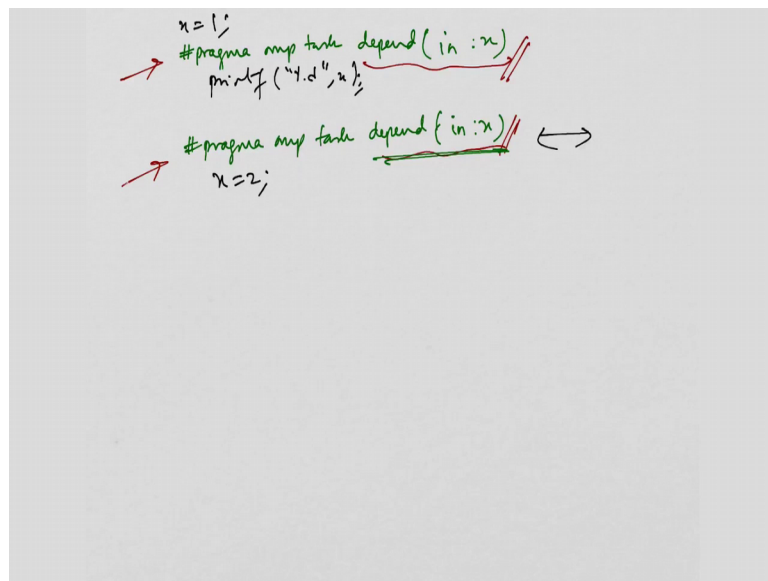
So, if this is let us say f and this is u what; that means, is that g will not be executed until f and u are done; that means, g is a dependent task of f and u that is all its saying why because this is trying to write to x and I have launched this task before launching this task and this task is saying that x is inout and I have launched this task before. So, both this task maybe updating x, how can I execute this task which was launched later and it requires x is input; I cannot execute this task until those tasks are done, all right. So, it is just creating a dependency graph that is what it s doing.

Student: (Refer Time: 05:52) does not actually (Refer Time: 05:55) because they are in sibling tasks.

No, no, no, no. So, this is not sibling task will be dependent task of all previously generated sibling tasks; I have not even seen the task which are going to come yet, I am not going to define a dependence with respect to them. So, what is important here? There is another very important aspect over here; why dependency is should result in a directed a cyclic graph I do not want a cyclic graph if you start looking at forward dependency is you are going to end up with the cyclic graph, it does not like a logical sense also, I mean a from a programmers point of view it is very counter intuitive, right to do something like that, but this is quiet intuitive that if they programmer root launch your task first and launch your task later and if you had an out something he was trying to write out in the first task in something he was trying to consume in the second task then he wanted the dependent, right. So, you do not want to execute these in parallel.

Yeah what does the second dependency is say it says that the generated task will be dependent task of all previously generated sibling tasks that reference at least one of the list items in an in out or in out clause that covers just about everything, right. So, what; that means is that if I have? So, depend out colon y then any previously generated sibling task if it had why in its list of independencies or out dependencies or in out dependencies any of the 3, this will be a dependent task of those tasks, it pretty much cover the everything, right.

(Refer Slide Time: 07:49)



```
x=1;
#pragma omp task depend (in :x)
printf ("%d", x);

#pragma omp task depend (in :x)
x=2;
```

The image shows two lines of handwritten code. The first line is `x=1;` followed by `#pragma omp task depend (in :x)` and `printf ("%d", x);`. A red arrow points to the `#pragma` line, and a red underline is under the `depend (in :x)` part. The second line is `#pragma omp task depend (in :x)` followed by `x=2;`. A red arrow points to the `#pragma` line, and a red underline is under the `depend (in :x)` part. A double-headed arrow is to the right of the second `depend (in :x)` line.

If I see he y in either in out or inout, I am not going to execute this task until that task is done.

Let us say I have these 2 tasks this is producing some output and I call this first and this is taking excise in as you can see this is write into x and this is printing the value of x, right. So, it is kind of like reading the value of x; in this is kind of like writing the value of x. So, note that if you specify these dependencies right that does not mean that you have to actually write to that variable and you actually have to consume that variable, right, you write the code that you want to do, but that is typically what you end of doing.

So, what will happen in this case?

Student: (Refer Time: 08:36).

What will be printed; let us say I have x equal to one outside. So, what will be printed 2.

Student: always.

Always right, you will never see one right because this is a dependent task.

Now, let me change this slightly. So, let me say that this is in and this is out, I say x equal 2 here and I print the value of x; what we will get printed?

What is the dependency for out?

Student: Previously.

All previously generated sibling tasks which refer to x will be executed before this task is executed, right. So, I will definitely be printing before I executive x equal 2 which means that I will always be printing the value 1. Now let me change these to out. Now what will happen?

Student: Still one.

Still one, right because this task; the dependencies says that all previously accruing siblings even when it is in in, out or inout.

Student: (Refer Time: 09:55).

Right; any of the 3; they will be executed before I execute and finally.

What will get printed?

Student: We can observe this.

Yeah. So, these 2 tasks can execute in parallel; any one of them may finish earlier. So, in this case, depending on which order they execute in you might see the output 1 or 2. Let us go back to the example. Now it will make a lot of sense. So, what is this saying; this is creating a graph, what is this graph?

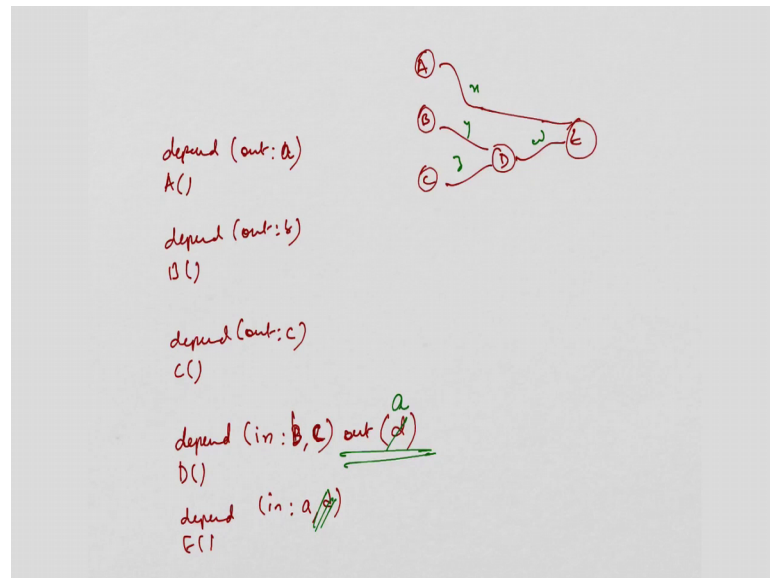
Student: (Refer Time: 10:33).

So, first there is this  $x$  equal to in it, right. So, the sum in it that is going to happen and it has an out of  $x$ , right and then there is this  $y$  equal to  $f$  of  $x$  which has  $x$  as an independency, right. So, this is  $y$  equal to  $f$  of  $x$  and similarly, this  $z$  is equal to  $g$  of  $x$  also has an independency of  $x$ , right and this has an out dependency of  $y$  which means that any appearance of  $y$  in, in out or in-out list before this task has to be completed for, but there is no occurrence, right.

So, there is no other dependency. Similarly over here out  $z$ , but  $z$  as an appear before this. So, no, issues here. So, finally, this is going to have a dependency of  $y$ , this is going to have a dependency of  $z$  and then this final task is depend in  $y$   $z$ . So, it is going to depend on both of these and finally, finalize is going to get executed here.

So,  $y$  and  $z$  can execute in parallel nothing else executes in parallel in a similar manner you can pretty much take any dag.

(Refer Slide Time: 11:58)



And you can design the dependencies for that. So, this is let us say that this is a dag; right. So, how do you do this? So, let us say there some function a here, b here, c here, d here, e here. So, what is the inout dependency; I should specified. So, that a, b, c can execute in parallel d can execute in parallel to a and then e must execute when all of these are done.

Student: (Refer Time: 12:31).

For a, I can say depend.

Student: Out.

Out; a let me not call at the same thing, yeah, small a, right, yeah for b.

Student: (Refer Time: 12:55).

C depend out c, right, depend out a for a, depend out b for b, depend out c for c and then for d.

Student: (Refer Time: 13:12).

Depend.

Student: In.

In.

Student: B comma c.

B comma c and out.

Student: (Refer Time: 13:23).

B, I suddenly started going capitals; b comma c and out d, right and finally, e.

Student: (Refer Time: 13:38) in.

In.

Student: A.

A.

Student: B.

B, I do not need an out; for instance, let me say that I was trying to save some variables, let me just take an example, right, let us say that I said out a here. So, that I only have to say in a over a here, let us say this is the change I made; what would happen; what do we; the repacation of this.

Student: The forth statement will also wait for (Refer Time: 14:15).

D would wait for a; that is the problem it will cause because if a occur is in the list of in out or inout of any task before this that task must complete. So, a must complete before d which is not what we wanted, right, but pretty much you can you know generate any dag that you want just associate a variable with each one of these and you can do it, right used that for specimen in our dependency.

Student: As openMP consider these as names or variables (Refer Time: 14:45).

Variables; variables; you use this variables is specify the dependencies looking at these variables openMP identifies the dependencies. Ideally, if you are saying out x your code is supposed to be writing 2 x; x is an output of your code, right, but does not necessary I mean If you;



Student: So, we can (Refer Time: 15:05).

Do not update  $x$ ; you do not update  $x$ .

Student: (Refer Time: 15:07).

Yeah, you can use any; you can just declare any variables and use.