STATEMENT-LEVEL CONCURRENCY

Concurrency is naturally divided into instruction level, statement level(executing two or more statements simultaneously), program unit level(execute two or more subprogram units simultaneously) and program level(executing two or more programs simultaneously). Here we only discuss unit level and statement level concurrency.

Concurrent execution of program units can occur either physically on separate processors or logically in some time-sliced fashion on a single-processor computer system. Statement level concurrency is largely a matter of specifying how data should be distributed over multiple memories and which statement can be executed concurrently.

Task is a process(thread) running on each processor. A task can communicate with other tasks through shared variables, or through message passing. Because tasks often work together to create simulations or solve problems they must use some form of communication to either synchronize their executions or share data or both.

Synchronization is a mechanism that controls the order in which tasks execute. Two kinds of syschronization are required when tasks share data, cooperation and competition. Cooperation synchronization is required between task A and B when task A must wait for task B to complete some specific activity before task A can continue its execution. Competition syschronization is required between two tasks when both require the use of some resource that can't be simultaneously used.

The methods of providing for mutually exclusive access to a shared resource are semaphores, monitors and message passing.

• Objective: Provide a mechanism that the programmer can use to inform compiler of ways it can map the program onto multiprocessor architecture

• Minimize communication among processors and the memories of the other processors

High-Performance Fortran

• A collection of extensions that allow the programmer to provide information to the compiler to help it optimize code for multiprocessor computers

• Specify the number of processors, the distribution of data over the memories of those processors, and the alignment of data

Primary HPF Specifications.

• Number of processors

!HPF$ PROCESSORS procs (n)

• Distribution of data

!HPF$ DISTRIBUTE (kind) ONTO procs :: identifier_list

– kind can be BLOCK (distribute data to processors in blocks) or CYCLIC (distribute data to processors one element at a time)

• Relate the distribution of one array with that of another

ALIGN array1_element WITH array2_element

Statement-Level Concurrency Example

```
REAL list_1(1000), list_2(1000)
 INTEGER list_3(500), list_4(501)
!HPF$ PROCESSORS proc (10)
!HPF$ DISTRIBUTE (BLOCK) ONTO procs ::list_1, list_2
!HPF$ ALIGN list_1(index) WITH list_2 (index)
!HPF$ ALIGN list_3(index) WITH list_4 (index+1)
 …
 list_1 (index) = list_2(index)
 list_3(index) = list_4(index+1)
```

• FORALL statement is used to specify a list of statements that may be executed concurrently

```
FORALL (index = 1:1000)
```

list_1(index) = list_2(index)

Specifies that all 1,000 RHSs of the assignments can be evaluated before any assignment takes place