Functional (Task) Decomposition
The Functional (or Task) Decomposition Design Pattern

Overall Problem

Thread 0

Thread 1

Thread 2

Thread 3
The Functional (or Task) Decomposition Design Pattern
How is this different from Data Decomposition (such as the OpenMP for-loops)

• This is being done less for performance and more for programming convenience.

• This is often done in simulations, where each “chunk” of the simulation needs to make decisions about what it does next based on what it and the other chunks are doing right now.

• Each chunk takes all of “Now” state data and computes a “Next” state.

• The biggest trick is to synchronize the different chunks of the simulation so that each of them is seeing only what the others’ states are right now. Nobody can be allowed to switch their state to “next” until they are all ready to switch together.

• The synchronization is accomplished with barriers.
Setup the **Now** global variables

Calculate the current Environmental Parameters

Spawn Threads

Watcher

Using the **Now** state, compute into the **Next** variables

DoneComputing barrier

Using the **Now** state, compute into the **Next** variables

DoneAssigning barrier

Copy the **Next** state into the **Now** variables

Print results and increment time

Calculate new Environmental Parameters

DonePrinting barrier

Copy the **Next** state into the **Now** variables
omp_lock_t Lock;

int main( int argc, char *argv[] )
{
    ...

    omp_init_lock( &Lock );

    omp_set_num_threads( 3 );
    InitBarrier( 3 );

    #pragma omp parallel sections
    {
        #pragma omp section
        {
            Watcher( );
        }
        #pragma omp section
        {
            Animals( );
        }
        #pragma omp section
        {
            Plants( );
        }
    } // implied barrier -- all functions must return to get past here
```c
void Animals( )
{
    while( << how to know when finished? >> )
    {
        int nextXXX= << function of all states >>
        . . .
        fprintf( stderr, “Animals waiting at #1.\n” );
        WaitBarrier( );
        fprintf( stderr, “Animals resuming at #1.\n” );

        NowXXX = nextXXX;

        fprintf( stderr, “Animals waiting at #2.\n” );
        WaitBarrier( );
        fprintf( stderr, “Animals resuming at #2.\n” );

        fprintf( stderr, “Animals waiting at #3.\n” );
        WaitBarrier( );
        fprintf( stderr, “Animals resuming at #3.\n” );
    }
}
```
```c
void Watcher( )
{
    while( << how to know when finished? >> )
    {
        fprintf( stderr, "Watcher waiting at #1.\n" );
        WaitBarrier( );
        fprintf( stderr, "Watcher resuming at #1.\n" );

        fprintf( stderr, "Watcher waiting at #2.\n" );
        WaitBarrier( );
        fprintf( stderr, "Watcher resuming at #2.\n" );

        << write out “Now” state of data >>

        << advance time and re-compute all environmental variables >>

        fprintf( stderr, "Watcher waiting at #3.\n" );
        WaitBarrier( );
        fprintf( stderr, "Watcher resuming at #3.\n" );
    }
}
```
omp_lock_t Lock;
int NumInThreadTeam;
int NumAtBarrier;
int NumGone;

void InitBarrier( int n )
{
    NumInThreadTeam = n;
    NumAtBarrier = 0;
    omp_init_lock( &Lock );
}

void WaitBarrier( )
{ 
    omp_set_lock( &Lock );
    {
        NumAtBarrier++;
        if( NumAtBarrier == NumInThreadTeam ) // release the waiting threads
            {
                NumGone = 0;
                NumAtBarrier = 0;
                // let all other threads return before this one unlocks:
                while( NumGone != NumInThreadTeam - 1 );
                omp_unset_lock( &Lock );
                return;
            }
    }
    omp_unset_lock( &Lock );

    while( NumAtBarrier != 0 ); // all threads wait here until the last one arrives
    #pragma omp atomic // and sets NumAtBarrier to 0
    NumGone++;
}
## The WaitAtBarrier() Logic

<table>
<thead>
<tr>
<th>Thread #0</th>
<th>Thread #1</th>
<th>Thread #2</th>
<th>NumInThreadTeam</th>
<th>NumAtBarrier</th>
<th>NumGone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calls WaitFor()</td>
<td>Sets the lock</td>
<td>Sets the lock</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Sets the lock</td>
<td>Increments NumAtBarrier</td>
<td>Increments NumAtBarrier</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Increments NumAtBarrier</td>
<td>NumAtBarrier != NumInThreadTeam</td>
<td>NumAtBarrier != NumInThreadTeam</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Unsets the lock</td>
<td>Stuck at while-loop #2</td>
<td>Stuck at while-loop #2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Stuck at while-loop #2</td>
<td>Calls WaitFor()</td>
<td>Calls WaitFor()</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Sets the lock</td>
<td>Sets the lock</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Increments NumAtBarrier</td>
<td>Increments NumAtBarrier</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>NumAtBarrier == NumInThreadTeam</td>
<td>NumAtBarrier == NumInThreadTeam</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sets NumGone</td>
<td>Sets NumGone</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sets NumAtBarrier</td>
<td>Sets NumAtBarrier</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Stuck at while-loop #1</td>
<td>Stuck at while-loop #1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Falls through while-loop #2</td>
<td>Falls through while-loop #2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Increments NumGone</td>
<td>Increments NumGone</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Returns</td>
<td>Returns</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Falls through while-loop #2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Increments NumGone</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Returns</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Falls through while-loop #1</td>
<td>Falls through while-loop #1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsets the lock</td>
<td>Unsets the lock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Returns</td>
<td>Returns</td>
</tr>
</tbody>
</table>
Sample Simulation Output