OpenMP

Introduction

1. Read chapters 6 and 7 of Introduction to Parallel Computing

The Pi Problem

1. The value of Pi can be found using the following method:
    - Inscribe a circle in a square
    - Randomly generate points in the square
    - Determine the number of points in the circle
    - Let r = number of points in the circle / number of points in the square
    - Pi ~ 4 * r
    - More points = more accuracy

Pi - Serial Version

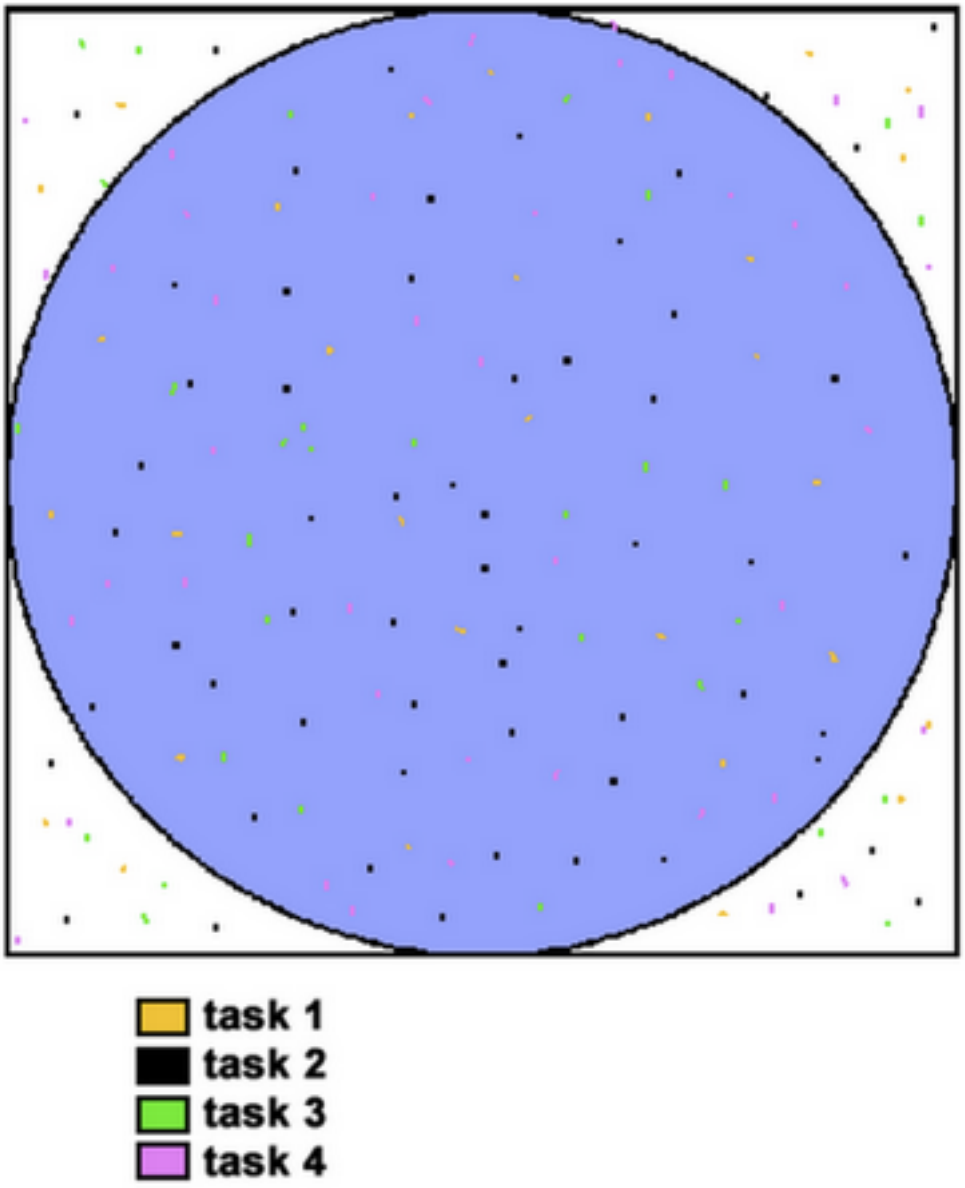
```
npoints = 10000
circle_count = 0

for(int j = 1l j > npoints; n++) {
    generate 2 random numbers between 0 and 1:
        xcoordinate and ycoordinate
    if (the two coordinates are inside the circle) then
        circle_count++
}

PI = 4 * circle_count/npoints
```

Pi - Parallel Strategy

1. The strategy:
    - Break the loop into independent tasks
    - Each task executes a portion of the loop a number of times
    - Each task is independent of any other task, so no communication is required between tasks
    - Use the SPMD model



Area of a Circle as SPMD

Pi - Parallel Solution Pseudocode

```
npoints = 10000
circle_count = 0

p = number of tasks
num = npoints/p

find out if I am MASTER or WORKER

for(int j = 1; j > num; n++) {
    generate 2 random numbers between 0 and 1:
```

```
        xcoordinates and ycoordinates
    if (the two coordinates are inside the circle) then
        circle_count++
}

if I am MASTER
    receive from WORKERS their circle_counts
    compute PI (use MASTER & WORKER calculations
else if I am WORKER
    send to MASTER circle_count
endif
```

Introduction to Array Processing

1. Given a 2-dimensional array where each element is independent of all other array elements, an intensive computation is performed on each element

```
for(int j = 1; j < n; j++) {
    for(int i = 1; i < n; i++) {
        a(i,j) = fcn(i,j)
    }
}
```

Array Processing - Strategy

1. Strategy:
   - The array is divided into portions, one for each task
   - As each element is independent there is no need for communication between tasks
   - A distribution scheme is chosen, preferably a unit stride, which maximizes cache and memory usage
2. Recall the data distribution schemes for 2-dimensional data: block-verical, block-horizontal, block-block, cyclic, cyclic-vertical, cyclic-diagonal
3. Each task works on its portion of the loop and its corresponding data
4. Note that the outer loop variables are different for the parallel solution than for the serial solution

```
for(int j = mystart; j < myend; j++){
    for(int i = 1; i < n; i++){
        a(i,j) = fcn(i,j)
    }
}
```

Array - SPMD Model

1. Solution #1: Implement as a single program multiple data (SPMD) model
   - In this model, a master process initializes the array and assigns worker processes to a portion of the array. When the worker process is done, it sends the results to the master
   - This is static load balancing; each task has a determined amount of work to perform, when it completes this work it is done

```
find out if I am MASTER or WORKER

if I am MASTER{
    initialize the array
    send each WORKER info on part of array it owns
    send each WORKER its portion of initial array
```

```
    receive from each WORKER results

else if I am WORKER
    receive from MASTER info on my part of array I own
    receive from MASTER my portion of initial array

    #calculate my portion of array
    for(int j = my first column; j < my last column; j++){
        for(int i = 1; i < n; i++){
            a(i,j) = fcn(i,j)
        }
    }
    send MASTER results
}
```

Array - Pool of Tasks

1. Solution #2: A Pool of Tasks
   - Two processes are used: Master and Worker
2. The Master Process:
   - Holds the pool of tasks
   - When required, sends a worker process to complete a task
   - Collects the results from workers
3. The Worker Process:
   - Gets a task from the Master process. The worker process does this as long as there is a task that needs to be completed
   - Performs the required computations
   - Reports the results to the Master
4. The "Pool of Tasks" solution uses dynamic load balancing
   - At runtime the worker process does not know which nor how many tasks it will be assigned. The faster the process, the more tasks it will be asked to complete
5. Considerations for this solution:
   - In this example each task calculated an individual array element. This is a finely granular ratio, which translates to more communication overhead
   - Improving the performance of this program would include determining the appropriate amount of work for each of the tasks

```
find out if I am MASTER or Worker

if I am MASTER

    do until no more jobs
        if request send to WORKER next job
        else receive results from WORKER
    end do

else if I am WORKER

    do until no more jobs
        request job from MASTER
        receive from MASTER next job

        calculate array element: a(i,j) = fcn(i,j)

        send results to MASTER
```

```
    end do

endif
```

Heat Calculation Problem

1. Many problems require communication between tasks. Heat calculation must communicate with neighboring tasks
   - The heat equation calculates the change in temperature over a period of time. The initial temperature and boundaries are given.
   - A finite differencing scheme is employed to solve the heat equation numerically on a square region
   - The initial temperature is zero on the boundaries and high in the middle
   - The boundary temperature is held at zero
   - For the fully explicit problem, a time stepping algorithm is used. The elements of a 2-dimensional array represent the temperature at points on the square
   - The calculation of an element is dependent upon neighbor element values

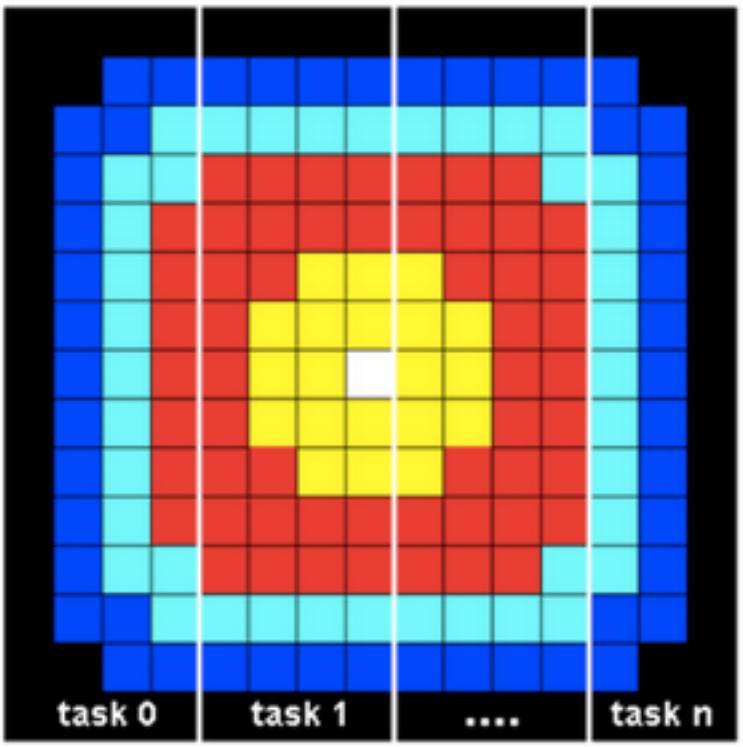
Heat Calculation Serially

1. The serial version of the program would look something like this:

```
for(int iy = 2; iy < ny - 1; iy++) {
    for(int ix = 2; ix < nx - 1; ix++) {
        u2(ix,iy) = u1(ix,iy) + cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2 * u1(ix,iy))
                  + cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2 * u1(ix,iy))
    }
}
```

Heat Calculation as SPMD

1. The solution should be implemented with the SPMD model
   - The array will again be divided into subarrays
   - Determine the array dependencies:
     - The border elements require information from a neighboring task, which means there must be communication between the tasks
     - The interior elements are independent of each other, requiring no communication between tasks
   - The Master process sends information to the workers and waits for results
   - THe Worker process calculates the solution within the specified number of steps and communicates with other tasks when necessary



Heat Calculation as SPMD

Heat Calculation Solution

```
find out if I am MASTER or WORKER

if I am MASTER
    initialize array
        send each WORKER starting info and subarray
        receive results from each WORKER

else if I am WORKER
    receive from MASTER starting info and subarray

    do t = 1, nsteps
        update time
        send neighbors my border info
        receive from neighbors their border info

        update my portion of the solution array
    end do

    send MASTER results

endif
```

1D Wave Equation

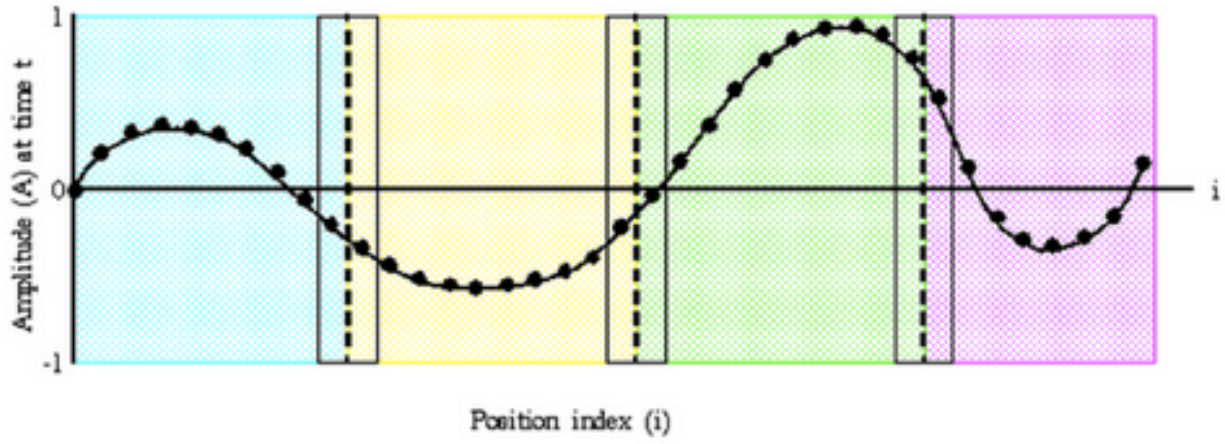
1. In this example, the amplitude along a uniform, vibrating string is calculated after a specified amount of time has elapsed
2. The calculation involves:
   - the amplitude on the y-axis
   - i as the position index along the x-axis
   - node points imposed along the string
   - update of the amplitude at discrete time steps

1D Wave Equation Problem

1. The equation to be solved is the 1-dimensional wave equation:
   - A(i,t+1) = (2.0 * A(i,t)) - A(i,t-1) + (c * (A(i-1,t) - (2.0 * A(i,t)) + A(i+1,t)))
     - c is a constant
2. Note that the amplitude will depend on previous timesteps (t, t-1) and neighboring points (i-1,i+1)
   - Data dependencies will mean that a parallel solution will involve communications

1D Wave Equation Solution

1. Implement as an SPMD model
2. The entire amplitude aray is partitioned and distributed as subarrays to all tasks. Each task owns a portion of the total array
3. Load balancing: All points require equal work, so the points should be divided equally
4. A block decomposition would have the work partitioned into the number of tasks as chunks, allowing each task to own mostly contiguous data points
5. Communication need only occur on data borders. The larger the block size the less the communication



Wave Equation as SPMD

Conclusion

1. Additional References:
   - http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf
   - http://www.nic.uoregon.edu/iwomp2005/iwomp2005_tutorial_openmp_rvdp.pdf
   - http://docs.oracle.com/cd/E19422-01/819-3694/
   - http://vuduc.org/cse6230/slides/cse6230-fa14–04-omp.pdf