

**Homework #2****Due Date May 4, 2005, 5pm**

We are going to use OpenMP for the same problem we did in HW1. Consider a code in which a Laplacian smoothing is iteratively performed on the  $a(i, j)$  array with a smoothing coefficient  $\epsilon = 0.1$ . The value of  $b(i, j)$  is computed from all neighbors including the four diagonals using the stencil described in the following code.

```

program main
.....
dimension x(n1),y(n2)
dimension a(n1,n2),b(n1,n2)
c initialize array x and y
  do i=1,n1
    x(i)= 1./float(n1)*(.5+(i-1))
  end do
  do j=1,n2
    y(j)= 1./float(n2)*(.5+(j-1))
  end do
c initialize array a
  do j=1,n2
    do i=1,n1
      if (x(i).lt.0.5) then
        a(i,j)= cos( x(i)+y(j))
      else
        a(i,j)= sin( x(i)+y(j))
      end if
      b(i,j)=a(i,j)
    end do
  end do
c perform Laplacian smoothing in interior points
  do n=1,iter
    do j=2,n2-1
      do i=2,n1-1
        b(i,j)= a(i,j) + epsilon * (
&          a(i-1,j+1)+  a(i,j+1)+a(i+1,j+1)
&          +a(i-1,j )-8.*a(i,j )+a(i+1,j )
&          +a(i-1,j-1)+  a(i,j-1)+a(i+1,j-1)
&          )
      end do
    end do
    a=b
  end do
end

```

**Problem #1:** Write an OpenMP program that uses the “parallel do” (“parallel for” in C) construct. Check that the parallel version gives the same results as the serial version. Run the code using a varying number of threads from 1 to 16, with  $n_1 = 1024$ ,  $n_2 = 1024$ ,  $iter = 100$  and compare the execution times. Make a table with the number of threads, the execution time, and the value of  $a(512,512)$  at the final iteration. Note the load on `junior.stanford.edu` or `gorilla.stanford.edu` at the time of running your benchmarks and try to do them (if you can) at a time when the machine is not being used. Show your results in a plot showing the parallel speedup and provide the source code commented as discussed in class (5% of the total grade.)

**Problem #2:** Write an OpenMP program that manually distributes the domain  $[0, 1] \times [0, 1]$  using  $p_1$  processors in the  $x$  direction and  $p_2$  processors in the  $y$  direction (follow the SPMD example in lectures 10 and 11 and use only the parallel region construct)

Run the code with  $n_1 = 1024$ ,  $n_2 = 1024$ ,  $iter = 100$  and compare the execution times for the following sets of  $p_1, p_2$ : (1,1),(1,2),(1,4)(1,8)(1,16), (2,2),(2,4),(2,8), (4,1),(4,2),(4,3),(4,4). Show your results in a plot showing the parallel speedup and provide the source code commented as discussed in class (5% of the total grade.)

**Problem #3:** Compare the times from the OpenMP implementations with the results from the MPI implementations (from HW1) and discuss your observations.

*Note: make sure to provide a brief discussion of the philosophy of your OpenMP parallel implementations for both Problems 1 and 2.*

---

### Compiling OpenMP programs on `junior.stanford.edu`:

To use OpenMP, these are the flags that you need:

```
/opt/SUNWspro/bin/f95 -xopenmp -xloopinfo hw2.f
```

```
/opt/SUNWspro/bin/cc -xopenmp hw2.c -lm
```

See the man pages in `/opt/SUNWspro/man` for `f77`, `f90`, `f95` and `cc` for more details.

To change the number of threads, you can use the environment variable `OMP_NUM_THREADS`.

```
setenv OMP_NUM_THREADS 4 (will set the value to 4 for csh,tcsh)
export OMP_NUM_THREADS=4 (will set the value to 4 for ksh,bash)
```

To get accurate timings, you can use the `timex` command:

```
timex hw2
```

Check the load of the machine with `uptime`, try to run the code when the machine is empty for good scalability results. The code should run in less than 15s with 1 thread and in less than 1s with 16 threads.