EE257/GP258 **Lab/Homework #4**
Applied Optimization Laboratory Handout #12
Due: Thurs., Feb. 7

In this exercise we will be writing parallel code and executing it on the CEES computing cluster. To access the CEES cluster, log on using ss as follows:

ssh –X cees-cluster –l <sunetid>

This should allow you to log on to the head node of the cluster. You create your program using the editor on the head node, and then execute it on one of the compute nodes using a script such as the run.sh script you can download from the web page.

If you download or create a script like run.sh, you simply type

source run.sh <your command string>

This executes the command you enter on the compute node, and writes the results to data files you create in the program. Any text output from the program gets written into a file called OUT. This file is written in append mode, so it will contain a record of everything you run until you delete the file and start over.

For example, suppose you have a program called "test" that requires command line arguments. You submit this as

source run.sh test arg1 arg2 arg3

and so on.

Problem 1 – Parallel loops.

Write a program with a single loop that displays a message along with the loop variable, such as

```
do i=1,10
        print *,'At loop: ',i
end do
```

Parallelize the loop and compile it with and without the openmp switch. Record the order that the loop executes and compare the results with and without the switch.

Problem 2 – Shared and private variables.

Write a program to create an array 3000 by 3000 in size, in which each element contains the square root of the absolute value of the product of the sine of the row number and the

cosine of the column number, both in radians. In your loops, set a floating point variable called row equal to the row number and a floating point variable call column equal to the column number.

Print out the first 10 diagonal elements of the array.

Run the program with and without the openmp switch, and compare the time required for the program to run in both cases. Ensure that you get the same results each way by examining the 10 diagonal elements described above.

You can get the time required for sections of your program to execute using the secnds() call. If you call secnds(0.), the function returns the time since midnight in seconds. Calling it with a different argument gives the time since that value. So calling it as

t0 = secnds(0.0)

<code you want to time>

t1 = secnds(t0)

returns elapsed time in seconds in the variable t1.

Problem 3 – Parallel least-squares evaluation.

Start with your code from assignment 2, problem 4, where you were asked to compute a solution for the spectrum using the smoothness constraint. Download the data file from the web page as your input, and assume that the kernel function from the previous lab apply. You can use the same matrix inversion subroutine as before.

Select an "optimal" value for the multiplier gamma by plotting the rms error of the solution as a function of gamma for 1000 values of gamma ranging from 0.001 to 1.0. (If you have scaled your problem differently than I have, your range may be different. Use an appropriate range, but evaluate over 1000 different values).

Parallelize the loop containing the various values of gamma. Do not try to parallelize the matrix inversion code, it will suffice to parallelize the gamma loop in the main program. Run the code on the CEES nodes for both parallel and single threaded versions. Estimate the increase in run speed by timing the code. Plot both the single threaded and parallel versions of rms error vs. gamma to check for any differences in result.

Problem 4 – Parallel principal components analysis.

Compute a principal components decomposition of the seven data files found on the web page. Each is an image consisting of 7875 pixels in 7518 lines. The seven files

correspond to seven different imaging wavelengths ranging from the visible through the infrared.

Create a rgb image using the three first principal components from these seven. As before, time the program with and without parallelization to see what factor of speed increase you can achieve.