Programming Assignment 6
Due Tuesday, March 21st

Objective
To learn about and practice using OpenMP. At the end of this assignment you should have a good understanding of the limits of Shared Address Space parallelization going in two direction: (i) what kind of performance you can expect using OpenMP, the most direct method of parallelizing your code, and (ii) the kinds of codes that are likely to be limited by basic overhead inherent in OpenMP.

Prerequisites (to be covered in class or through examples in on-line documentation)
Programming – Basics of OpenMP and PThreads.

Eng-Grid note: For this lab you are best off using the 8 core BME rather than the ME processors. You are more likely to get the entire CPU for your entire run. Also, the shared memory model is more straightforward (than, e.g., the 32 core ME nodes).

Assignment

Part 1: OpenMP basics

1. "Hello World!" (or "ell HWoo!dlr")
Reference code: test_omp.c
The reference code prints "Hello World!" by having each character printed by a different thread. Parallel Section is used.
Task: Write a similar function, but this time use **Parallel For**. Have each iteration print a different character.
Deliverable: Your code and your output.

2. Parallel For
Reference code: test_omp_for.c
The reference code has three different functions with two different versions each: serial baseline and OpenMP. The three functions are, respectively, compute bound, memory bound, and neither (overhead bound).
Task: (Analogous to Part 3 of Assignment 5) Find the intrinsic OpenMP overhead. That is, how long does it take to do operations necessary for all likely threaded codes – **create, pass parameters, join**. In OpenMP these operations are implicit rather than explicit, but they must be executed just the same! To find the break-even point between the serial and multithreaded versions you may want to adjust the array size and work-per-thread.
Deliverable: Results and brief explanation. Bonus for a formula based on Amdahl's Law. Bonus for comparison with Pthreads.

3. MMM, 3 loop version
Reference code: test_mmm_inter_omp.c
The reference code has two 3 loop functions for MMM: ijk and kij. Each has two versions, the serial baseline and the OpenMP.
Task: Experiment with two things, "shared" lists and pragma placement
a) Start by generating results for a variety of matrix sizes. As always, a few runs should be completely in cache and a few not.

b) Loop indices are supposed to be private variables (by default). Test this out by moving the loop indices to/from the "shared" list and rerunning the code.
c) It may seem obvious that the best place to put the pragma "parallel for" is around the outermost loop. Test this assumption by moving the pragma inside first to the middle loop and then the inner loop. For each, rerun the code.
Deliverable: Results and brief explanation.

Part 2: OpenMP on real programs
For both of these, the goal is to take a naïve approach to using OpenMP and see what happens (e.g., compared with the approach in Part 1.3 and the threaded approach). If you have time see if you can optimize the OpenMP directives.
1. SOR
Task: Parallelize the SOR codes from Assignment 6 using OpenMP.
2. MMM
Task: Parallelize your best previous MMM code using OpenMP.
Deliverables: your code, results, and a brief explanation.