Parallel and Distributed Computing

Introduction to OpenMP

Throughout this class you are encouraged to consult the documentation in `http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf` whenever you have questions on the OpenMP directives or APIs.

1. This problem tries to familiarize the students with the work environment and the compilation of OpenMP code.

   Consider the following program (`omp-1.c`):

   ```
    1. int main(int argc, char *argv[]) {
    2.
    3.   int i, tid;
    4.
    5. #pragma omp parallel private(i,tid)
    6.   {
    7.       tid = omp_get_thread_num();
    8.
    9. #pragma omp for
   10.       for(i = 0; i < NUMITER; i++)
   11.           printf("Thread: %d\titer=%d\n", tid, i); fflush(stdout);
   12.
   13.       printf("Thread %d, almost...\n", tid); fflush(stdout);
   14.       printf("Thread %d, done!\n", tid); fflush(stdout);
   15.   }
   16.
   17.   printf("All threads have finished!\n");
   18. }
   ```

   a) Start by compiling the code above, don't forget to add the "`-fopenmp`". Execute this program with a different number of threads (by setting `OMP_NUM_THREADS`).

   b) Add the "`nowait`" clause to the for directive in line 9. Did the output change? Why?

   c) Add a barrier between lines 13 and 14:
   `#pragma omp barrier`
   Compare the output with a) and b).

2. In the second problem you are asked to parallelize an inner loop with data dependencies. The basic code that should run in parallel is as follows:

   ```
   for(iter = 0; iter < numiter; iter++)
       for(i = 0; i < size - 1; i++)
            V[i] = f(V[i], V[i+1]);
   ```

   where we define a simple function to manipulate the data.

   ```
   #define f(x,y) ((x+y)/2.0)
   ```

   a) Download, compile and run the sequential version of the code (`omp-2.c`).
   b) Create a simple (and incorrect) parallel version by adding a parallel directive to the inner loop. Try to find an execution with a different output than the sequential version. Why can this happen?
   c) Fix the parallel version by copying vector `V` from iteration i-1 before executing iteration i, and using the copied values as arguments to function `f` in iteration i.
   d) Write a modification to the parallel version that avoids copying vector `V` in all iterations.