OpenMP Programming on Intel CPUs
Mike Giles

Practical 2: a Monte Carlo solver

This practical is based on the Monte Carlo solver discussed in Lecture 3.

The main objectives in this practical are to learn about:

- the use of `#pragma omp threadprivate` to define private storage for a separate random number generator for each thread
- the use of `#pragma omp parallel` to create a parallel section executed by multiple threads
- the use of `omp_get_thread_num()` to obtain a thread ID so that different threads perform different calculations
- the use of the MKL library to generate a block of Normally-distributed random numbers
- how to maximise performance by generating large blocks of random numbers which are still small enough to fit within the L1 or L2 cache

The practicals can again be carried out on the system called `mimic`, or on any other Maths Institute server or desktop.

What you are to do is as follows:

1. Create and go into a course subdirectory within your account, and then copy all of the course files into it by using the command

   ```
   cp -r ~gilesm/html_work/OpenMP/prac2 .
   ```

2. You may need to use the command

   ```
   source /opt/intel/oneapi/setvars.sh
   ```

   to set lots of environment variables associated with using the Intel compiler.

3. Compile the code using the command

   ```
   make
   ```

   which follows the instructions in the `Makefile`. Note the additional flag to link to the MKL library.

4. Set thread pinning by using the command

   `export OMP_PROC_BIND=true`

   set the number of OpenMP threads by using the command

   `export OMP_NUM_THREADS=4`

   and run the code and see the performance you obtain.

5. Try varying the number of threads and see whether the execution speed is proportional to the number of threads

6. Try varying the size of the `RNV` which controls how many random numbers are generated in each block. Try sizes as small as 1k, and as large as 10M.

7. Following the example of Practical 1, modify the code to compute and print the amount of memory traffic being generated, remembering that the random numbers have to be written by the random number generator, and then read by the path generator.

   Hence, compute and print out the corresponding Bytes/s and compare it to the peak which the hardware is capable of.

8. Modify the code to treat `sum1, sum2` as two elements in an array, using the array reduction syntax described in lecture 2.

9. Modify the code again to use `#pragma omp parallel` instead of `#pragma omp parallel for`, manually determining how many samples need to be calculated for each thread, based on thread index and total number of threads.