OpenMP Programming on Intel CPUs
Mike Giles

Practical 1: a simple finite difference code

This practical is based on the simple 2D finite difference code discussed in Lecture 2.

The main objectives in this practical are to learn about:

- the way in which the execution of the code is affected by various environment variables and compiler flags
- the way in which performance is affected by changes in the loop order
- the use of the `reduction` clause

The practicals can 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/prac1 .
   ```

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`.

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. Also try using more threads than the number of cores in your system to see what happens.

6. Try turning off OpenMP all together by editing the `Makefile` to delete the `icc` compiler flag `-qopenmp` (or the `gcc` flag `-fopenmp`), and see what happens.

7. Turning full parallelisation and vectorisation back on by going back to the original `Makefile`, see what happens when you swap the i and j loops.

8. Also, see what happens if you put the `#pragma omp parallel for` before the inner loop instead of the outer loop.

9. Modify the code to compute the root-mean-square change in the solution at each timestep, using a `reduction` clause on the parallel loop, as discussed in Lecture 2.

10. Finally, look at the reported performance in terms of flops/s and bytes/s. Make sure you understand how they have been calculated (ask questions if things are not clear) and think about how it compares to the peak which the hardware is capable of.

    Reminder: use `lscpu` to find out the details of the CPU you are using, and divide the L1 and L2 cache sizes by the number of cores in each chip to get the amount per core.

    How much data do you think is being moved from L3 to L2 cache?

11. The second code `fd3.c` is a 3D version of the algorithm. Experiment with the tile parameters and the use of the `collapse` clause to try to maximise the performance.

    Think about how much data is being moved each iteration, noting that the arrays are too big for the L2 and L3 caches. How does the DDR5→L3 bandwidth compare to the theoretical peak?