|epcc|

Multicore Performance Issues
Exercise Notes

Getting started

Copy the tar file containing the serial traffic modelling code from the course web pages to your account and unpack it with the command:

```
tar -xvf OMP-traffic.tar
```

1 Serial code

First we will look at how the cache affects the performance of serial codes. Typically, we expect calculations on smaller problem sizes to be faster as the entire dataset can fit into the faster cache memory.

1. Add timing to the code so that you measure the elapsed time of the main iteration loop (i.e. excluding the initialisation phase). You can time code as follows:

   ```
   tstart = omp_get_wtime();
   ... code to be timed in here
   tstop  = omp_get_wtime();

   tcalc  = tstop - tstart;
   ```

   where `tstart`, `tstop` and `tcalc` are double-precision variables.

2. Convert the time into a performance metric by computing the number of cells updated per second.

3. Run the code using a single thread with `NCELL` set to 10, 1000, 1000, . . ., 100000000 (one hundred million).

4. Make a graph of the performance against the problem size.

5. Is the way the performance varies consistent with what you would expect given that the cache sizes are

   - Level 1: 32 KB per core
   - Level 2: 256 KB per core
   - Level 3: 30 MB shared by all cores

   and noting that an integer variable occupies 4 bytes?

2 Parallel code

Now we will look at how cache coherency overheads and NUMA architectures can affect the performance of parallel codes.

1. Parallelise both the update loop in the `updateroad` function and the copy-back step in the main program. These can each be done with the addition of a single `parallel for` or `PARALLEL DO` directive, but note that you will also have to use a reduction variable in `updateroad`.
2. Using the largest value of NCELL (i.e. one hundred million), measure the performance as a function of the number of threads from 1 up to a maximum of 24. You should do these studies on the ARCHER compute nodes, i.e. submit the jobs to the batch queue with `qsub`.
3. How does the performance change if you do not parallelise the copy-back step?
4. To simulate cache thrashing, use a shared variable in `updateroad` rather than a reduction variable. How does this affect the performance on 24 threads? Note that you will no longer get the correct answer!
5. Before the serial call to `initroad`, insert a parallel loop that initialises `oldroad` and `newroad` to zero; this will ensure NUMA-aware memory allocation.

   Now re-measure how the performance varies from 1 to 24 threads and compare to your previous results obtained with purely serial initialisation; can you see the NUMA effects? Are they consistent with what you would expect from the architecture of the ARCHER nodes which comprise two sockets each containing a 12-core CPU?
6. What happens to the performance if you do the parallel initialisation in the reverse direction, i.e. looping from `i = NCELL` to `i = 1` in steps of -1?