SICSA Int. Summer School on Advances in Programming Languages
OpenMP Practical Exercise

These exercises will give you experience of writing and controlling OpenMP programs on a multicore. This document is available on the web at `www.macs.hw.ac.uk/ trinder/ParDistr/AiPLOpenMPLab/` the advantage being that you can paste commands from the pages.

The exercises use examples from the Lawrence Livermore National Laboratory (LLNL) OpenMP Tutorial, as referenced below.

1 Setup and Hello World example

1. The following commands assume you're using the Bash shell. Either translate all the commands to your favourite shell or switch to Bash:
   `bash`
2. The support for OpenMP has improved greatly in recent Gnu C compilers, so link to a recent version. To save you re-aliasing you may wish to include this in your `.profile`:
   `alias ogcc='/usr/local/gcc4.3/bin/gcc'`
3. Copy `omp_hello.c` from section 3 of `https://computing.llnl.gov/tutorials/openMP/exercise.html`.
4. Review the source code and note how OpenMP directives and library routines are being used.
5. Log on to a multicore machine, e.g.
   `lxpara2`
6. Compile `omp_hello.c` for parallel execution:
   `ogcc -fopenmp omp_hello.c -o hello`
7. Run with the default number of cores:
   `hello`
8. Set the number of cores to 4 and rerun:

   ```
   export OMP\_NUM\_THREADS=4
   hello
   ```

2 Data Parallelism Example

1. Copy `omp_workshare1.c` from section 3 of `https://computing.llnl.gov/tutorials/openMP/exercise.html`.
2. Review the source code to understand the data parallelism introduced by the `for` directive.
3. Compile it and run it with 4 threads
4. Now run it with a sort to make the output more comprehensible as follows: `omp_workshare1 | sort`

3 Control Parallelism Example

1. Copy `omp_workshare2.c` from section 3 of `https://computing.llnl.gov/tutorials/openMP/exercise.html`.
2. Review the source code to understand the control parallelism introduced by teh `section` directive.
3. Compile it and run it several times with 4 threads and observe any differences in output.
4. Reflect: Because there are only two sections, you should notice that some threads do not do any work. You may/may not notice that the threads doing work can vary. For example, the first time thread 0 and thread 1 may do the work, and the next time it may be thread 0 and thread 3. It is even possible for one thread to do all of the work. Note that the *result computed is always the same, but which threads do the work is non-deterministic.*

4 Reduction

1. Copy `reduction.c` from `www.macs.hw.ac.uk/ trinder/ParDistr/`.
2. Review the source code and compile it
3. Run it.

5 OpenMP Programming

5.1 Munge

1. Copy the sequential `munge.c` from `www.macs.hw.ac.uk/ trinder/ParDistr/`.

2. Review the source code, compile, and run it sequentially.
3. Adapt the program to be parallel using OpenMP
4. Evaluate the parallel performance as the number of cores ranges over 1, 2, 4, 6 and 8.

5.2 TotientRange

1. Copy the sequential `TotientRange.c` program available from `www.macs.hw.ac.uk/ greg/courses/F21DP2/TotientRange.c`
2. Review the source code, compile, and run it sequentially.
3. Adapt the program to be parallel using OpenMP
4. Evaluate the parallel performance as the number of cores ranges over 1, 2, 4, 6 and 8.