UCT CS Honours 2018

High Performance Computing Module Assignment

(**version 1**)

Lecturer: Michelle Kuttel

Due date: Monday **22 October** 2018, **10 pm**

1 Aim

This project will give you experience in parallel programming using both OpenMP and MPI, as well as validation and profiling of parallel codes. In this project, you will: write serial and parallel software to analyze simulation data; evaluate the parallel programs experimentally and write a clear, detailed report to present and discuss your methods and results.

2 Problem Description

For this project, you will analyze the data produced by simulation of an atomic system: a set of **n** atoms simulated for **t** time steps. For the purposes of the assignment, you can consider atoms as just moving points in space. The problem is as follows. Consider a set **S** of **n** points in 3D space simulated for **t** timesteps. Given a subset **A** of the points in **S,** a disjoint subset **B** of **S**, and a cut-off, **k,** for every time step you must output the **k** closest **A-B** pairs and the distance between them, in order of increasing distance.

Obviously, as **S**, **A**, **B** and/or **k** increases, the number of calculations required increases.

You will write a serial algorithm (in C or C++) to solve this problem, as well as parallel versions in both OpenMP and MPI.

3 Code requirements

3.1 Input

The code should allow specification of an input file (using flag "`-i input_file`") where the text input file lists:

```
Name of the trajectory file in dcd format

k

Atom indices comprising particle set A (a range or itemized list of
comma separated values or mix of two)

Atom indices comprising particle set B (a range or itemized list of
```

```
comma separated values or mix of 2)
```

For example:

```
The_simulation.dcd

2

1,25-28,67,101

101-605
```

3.1.1 Dcd file format

You will be provided with binary simulation files of various sizes for testing. Each binary file will contain a (potentially very long) record (or **trajectory**) of the positions of the **n** simulated atoms in the system at each time step **t**. Atoms are identified by an index, in the range **0** to **n-1**.

The files are in the "CHARM dcd" format (a single precision binary FORTRAN format) so they are transportable between computer architectures (though if the endianness of the machine on which the DCD file was written differs from the analysis machine, it will need to be flipped).

The exact format of these files is very ugly, but widely used. You will only need to access to atom positions (stored in Å) at each timestep (which is stored in internal units). The DCD format is structured as follows (FORTRAN UNFORMATTED, with Fortran data type descriptions):

```
HDR      NSET    ISTRT   NSAVC   5-ZEROS NATOM-NFREAT    DELTA   9-ZEROS
`CORD'   #files  step 1  step    zeroes  (zero)          timestep  (zeroes)
                         interval
C*4      INT     INT     INT     5INT    INT             DOUBLE  9INT
================================================================================
NTITLE           TITLE
INT (=2)         C*MAXTITL
                 (=32)
================================================================================
NATOM
#atoms
INT
================================================================================
X(I), I=1,NATOM          (DOUBLE)
Y(I), I=1,NATOM
Z(I), I=1,NATOM
================================================================================
```

You may adapt existing code to read your dcd files (e.g. https://www.ks.uiuc.edu/Research/vmd/plugins/doxygen/dcdplugin_8c-source.html) or use an external library.

3.2 Output

Your code must output a simple text file (using flag "`-o output_file`") of comma-separated values. Each line in the file lists a time step (`integer`), the atom pair defined as an atom index from subset **A** (`integer`) and the atom index from

subset **B** (`integer`) and a distance in Å (floating point). For example:

```
101,25,3,4.567

101,1,2,5.67

102,1,2,4.35

102,25,3,5.55
```

3.3 Parallelization

You need to implement and compare three versions of your code: an **optimal** serial version, an OpenMP version and an MPI. These all need to be validated and the parallel versions run and benchmarked against the serial version for different inputs, numbers of cores and nodes.

3.4 Submission

You must submit a code archive including running/installation instructions in a README file.

4 Report

You must also hand in a report containing the following sections.

- Introduction
- Serial algorithm
  - Description
  - Implementation
  - validation
- Parallel algorithm: OpenMP
  - Description
  - Implementation
  - Validation
- Parallel algorithm: MPI
  - Description
  - Implementation
  - Validation
- Benchmarking
  - Methods
  - System Architecture
  - Results and Discussion
- Conclusions

- The *Introduction* must explain the problem and highlighting its suitability for

parallelization.

- For the three algorithms, the *Description* sections should include a theoretical estimate of the running times. The *Implementation* sections must explain your approach to solving the problem and any important implementation details. The *Validation* sections should explain how you ensured that your implementation was correct.
- Under *Benchmarking*, the *Methods* section must explain how you timed your algorithms (with different input, cores etc.) and how you measured speedup. Make sure that you do this properly, for different input sizes and consider the different components of the application. *System Architecture* should detail the machine architectures you tested the code on. The *Results and Discussion* section must document the **speedup** (not timing) of your parallel implementations. This section must have graphs and, if necessary, tables (do not include voluminous tables in the text – add them as appendices and graph the results). Graphs should be clear and labelled (title and axes). You should compare and contrast OpenMP and MPI and discuss any problems or difficulties encountered.
- The *Conclusions* section must summarize the main results and findings.

Please do NOT ask the lecturer for the recommended numbers of pages for this report. Say what you need to say: no more, no less.

5 Assignment submission requirements and assessment rules

- You will need to create, **regularly update**, and submit a GIT archive of your code.
- Your submission archive must consist a technical report and your solution code archive (including a **Makefile** for compilation/running).
- Submit your report as a separate (from the source code) file **IN PDF FORMAT** named **HPC**_*STDNUM001*.**pdf,** replacing STDNUM001 with your student number. **Incorrect formats will incur a 5% penalty.**
- Upload the files and **then check that they are uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.
- The usual late penalties of 10% a day (or part thereof) apply to this assignment.
- The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.
- Note well: submitted code that does not run or does not pass standard test cases will result in a mark of zero. **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.

.