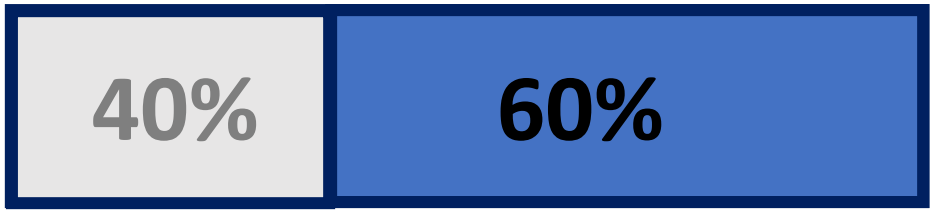
HPC Assignment 2:

Distributed memory parallelism with MPI

In this assignment, use MPI to parallelise your optimised lattice Boltzmann code, and run on all the cores of up to 4 nodes of Blue Crystal



Assignment description

Ballpark "Compute times" for a flat MPI code on 4x28 cores are:
128x128: 1.7s
256x256: 6.0s
1024x1024: 14.0s

- Start with your optimised OpenMP LBM code
- Use MPI "Single Program Multiple Data (SPMD)" distributed memory parallelism to run your code from one core up to all cores of **4 nodes**
- Serial optimisations will help
- Produce a short report discussing your findings (4 pages)
  - Describe the optimisations that you tried and your approach to parallelism
  - Explain why your optimisations improved performance
  - Include an analysis of how well your code scales from one up to all cores, in increments of one core

Coursework submission

- Your submission will be made via BlackBoard and should include:
  1. A **4 page** report in PDF form, which must include:
     - Your name and user id
     - A description of your MPI parallelisation;
     - Comparisons of your MPI performance vs optimised serial;
     - An analysis of the scalability of your code from one core up to all cores in a node;
  2. The working code you used to generate the results in your report.
- Your code must pass the Python check script “check.py” included in the repo.

Marking scheme – 50%+

- To achieve a mark of 50%+, you will need:
  - A reasonable 4 page report that demonstrates you have a some understanding of the main issues
  - MPI code that works, and that runs at reasonable speed
- Your time on all 4x28 cores of Blue Crystal phase 4 should be consistent with the table on page 2 of this assignment

- You should be able to get a simple MPI version working in 2-3 days, with a bit more time to write it up

Marking scheme – 60%+

- To achieve a mark of 60%+, you will need:
  - A well-written, 4 page report that clearly demonstrates you have a good understanding of the main issues
  - Code that successfully uses MPI parallelism, achieving performance when using all the cores of a node similar to the ballpark times we give in the table on page 2 of this assignment

- Allow 4-5 days to get to this level, including writing the report

Marking scheme – 70%+

- To aim for a first (70%+), you'll need:
  - An excellent 4 page report
    - Includes more detailed analysis, such as correctly applied roofline or similar
  - Code that:
    - Applies further MPI techniques that improve performance above those we've described. These may include code transformations beyond those discussed in class
    - Delivers performance on all cores that achieves a good fraction of STREAM memory bandwidth

- You should be able to complete this assignment in 7-10 days, avoid the temptation to keep working on it much beyond that (diminishing returns)
  - It should only take half that time to do a simple MPI version which, along with some interesting experiments and a decent report, should be good enough to earn 60%+

Submission specification

- Your **report** which must be in a file called "**report.pdf**",
  - Lower case r: "**report.pdf**" NOT "**Report.pdf**"
- Your **source code**, i.e. "**d2q9-bgk.c**"
- Your **makefile**, called "**Makefile**"
- An **env.sh** file containing any module commands you need to use to load specific compilers or anything required else to run your code properly. Our automaker script will just source your env.sh file then run 'make' before running your executable, so make sure this is all that's required
- **Don't** modify the timing code in the starting code, as we'll use this to automatically extract timing information from each submission
- We must be able to reproduce any runtimes you quote in your report by compiling and running the code that you submit
- Please **zip these files up and submit one zip file**, ***don't*** submit them as separate files in Blackboard

Which parts of the code do you need to time?

- For the MPI assignment, we ask you to time 3 different stages of your code:
  - **Initialization** (Initialize the data structures and load values from file, each MPI rank has data needed for compute after this stage).
  - **Compute** (The programs main loop, this is what was timed for the previous assignment).
  - **Collate** (This section of the code collects data back from the MPI ranks).
- A **total time** will also be output, representing the accumulative time of all 3 stages.
- The output format can be found in the source code of the `advanced-hpc-lbm` repo.

MPI LBM Program Output

```
/* write final values and free memory */
printf("==done==\n");
printf("Reynolds number:\t\t%.12E\n", calc_reynolds(params, cells, obstacles));
printf("Elapsed Init time:\t\t\t%.6lf (s)\n",    init_toc - init_tic);
printf("Elapsed Compute time:\t\t\t%.6lf (s)\n", comp_toc - comp_tic);
printf("Elapsed Collate time:\t\t\t%.6lf (s)\n", col_toc  - col_tic);
printf("Elapsed Total time:\t\t\t%.6lf (s)\n",   tot_toc  - tot_tic);
```

- Full code at https://github.com/UoB-HPC/advanced-hpc-lbm/blob/bcp4-slurm/d2q9-bgk.c

Things to avoid!

- Invalid commands in env.sh: -1
- Missing env.sh: -3
- Invalid Makefile syntax (line endings or spaces): -1
- Missing build files or otherwise broken build: -3
- Minor validation issue (~1 failure): -1
- Major validation issue (3+ failures): -3
- Catastrophic (almost complete) validation failure: -5
- No name on report: -1
- Altered program output: -1

How the automarker works

1. Starts with a clean environment
   - See: https://github.com/UoB-HPC/hpc-course-getting-started/blob/master/2_Modules.md#unloading-modules
2. Runs "**source env.sh**" to load your chosen modules
3. Runs "**make**" to build your application and check that d2q9-bgk has been produced
4. Runs the application: "**./d2q9-bgk input.params obstacles.dat**"
5. Runs the checking script

If any of the steps above results in an error, a failure is reported at that point. Otherwise, the time is recorded. We recommend you go through these steps manually with your submission, e.g. by redownloading your submitted zip file, to avoid any surprises with the automarker.

Plagiarism checking

- The HPC assignments are all for individuals, they are **not** group work
- We will check **all** submitted code for plagiarism using the MOSS online tool
  - MOSS ignores the example code we give you
  - MOSS will spot if any of you have worked together or shared code, so **please don't!**
- We'll also check all submitted reports using the TurnItIn tool, which will find any shared text
- So please don't copy code or text from each other! You **will** get caught, and then ***both*** the copier and original provider will get a **0** for the whole assignment.

Summary

Remember, you'll get marks for:

- A well written, comprehensive, report
- An MPI code that successfully explores most of the optimisations we suggest

Have fun exploring your first multi-node, distributed memory parallel programs!