COMP90025 Parallel and multicore computing
Project 2: Gravity simulation using MPI

September 18, 2023

Project 2 requires developing and submitting a C++ MPI program and a written report. Project 2 is worth 20% of the total assessment and must be done individually. The due date for the submission is nominally **11.59pm on the *Wednesday* of Week 12**. Submissions received after this date will be deemed late. A penalty of 10% of marks per working day late (rounded up) will be applied unless approved prior to submission. The weekend does not count as working days. A submission any time on Tuesday will attract 10% loss of marks and so on. Please let the subject coordinator know if there is anything unclear or of concern about these aspects.

If you are not familiar with C++, note that most C programs are also valid C++. You are welcome to write a program as if it were just C, without using C++ featues. To help this, the skeleton has been written in C-style C++; you are welcome to rewrite it in more idiomatic C++ if you wish. The choice of C++ is mainly to give you access to C++'s random number libraries, however you may also find its container and vector classes useful.

1 Background

This project will involve simulating the "gravitational" attraction of objects in D-dimensional space. It combines a basic building block of machine learning (clustering) with a physics simulation.

A widely used clustering method is k-means. This starts by choosing k initial vectors called centroids, and then alternating between two steps:

1. Each point to be clustered is assigned to the nearest centroid
2. Each centroid is replaced by the mean of all the points assigned to it.

The performance of k-means depends heavily on how the initial centres are chosen. A simple way to do this is to choose k points from the original data set, but there is a high chance this will not lead to a good solution. If they are not spread out well through the data, the clusters will be uneven. A good way to choose initial centres is the k-means++ algorithm. That is as follows:

1. One of the input points is chosen uniformly at random as the first centre.
2. For $n = 2$ to k
   (a) Find the distance $d(i)$ from each point i to the nearest centre that has been chosen so far.
   (b) Choose centre n at random, from a *weighted* distribution, with the probability of choosing i proportional to $d(i)^2$. (Note that the probability of choosing an existing centre is 0. You can treat that as a special case if you like.)

Note that step 2(a) doesn't require recalculating distances to centres before centre $n - 1$.
Both k-means and k-means++ are described well in Wikipedia.

We will be clustering data drawn from a Gaussian mixture model (GMM). A c-component Gaussian mixture model is described by c means, c standard deviations (or, more generally, covariances – but we won't need those), and c probabilities, which sum to 1.

To generate a random variable from a Gaussian mixture model, first choose which component it belongs to. This requires you to generate a variable from a weighted distribution, weighted by the c probabilities. You can use the standard C++ class `std::discrete_distribution`, (`#include <random>`).

The next step is to generate a Gaussian ("normal") random variable for that cluster. For this project, we will use a simple case where each component of the Gaussian vector is independent of the others, and has the same standard deviation. Many ways to generate a "standard normal" (mean 0, standard deviation 1) are mentioned at `https://stackoverflow.com/questions/2325472/generate-random-numbers-following-a-normal-distribution-in-c-c` You can use the standard C++ class `std::normal_distribution`, as the skeleton does, but please do not use any other libraries like boost. You can use algorithms or code you find on the web, but if you don't write your own generator, remember to cite the source of the code.

From the standard normal, you obtain the Gaussian with the given mean and standard deviation by first multiplying all coordinates by the standard deviation and then adding the mean of each component.

Each node will start by generating a large number of random points in D-dimensional space, with a clustered distribution.

Nodes will then cooperate to perform k-means clustering of those points.

The points will then be shuffled among nodes so that each node contains the points of a single cluster.

After this, the points will "move" in space – that is, a point at location $(x_1, x_2, x_3, ..., x_n)$ will in one iteration will be replaced by a point at location $(y_1, y_2, y_3, ..., y_n)$ in the next iteration – subject to "gravity", assuming all points are of equal mass and are initially at rest.

Gravity in D dimensions is the same as gravity in 3 dimensions:

1. Each point has a position and velocity.
2. At each time step, the velocity is increased by an amount proportional to the sum of the forces acting on it.
3. The force of object i on object j is proportional to $1/d(i,j)^2$, where $d(i,j)$ is the Euclidean distance between the points, and is directed toward the other point.

2 Assessment Tasks

1. Write C/C++ code to do the following:
   (a) Read as input, from a text file specified on the command line,
      i. an integer N specifying the number of points,
      ii. an integer D specifying the dimension,
      iii. a Gaussian mixture model, specified by
         A. an integer c specifying the number of components
         B. c lots of
            - D doubles: the mean of this component
            - 1 double: the standard deviation of each dimension of this component
            - 1 double: the probability of this component.
   (b) Generate N points from the GMM, with N/k per node, on k nodes.
   (c) Cluster the N points using k-means, optionally with k-means++
   (d) Transfer the clusters so that each cluster is on a single node.

(e) Assume each point is a physical body, and all bodies have equal mass. Simulate the motion of the bodies under gravity, assuming they are initially at rest (not moving).

Choose suitable approximations to trade accuracy against speed.

(f) Stop when the points are about twice as close to each other as they were at the start. You can measure this by, for example, the variance of the point coordinates being reduced by a factor of 4, where the variance of m variables x_i is given by $\frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})^2$, and $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$.

2. Evaluate the performance of your code, in terms of the speed of execution and the quality of the approximations. Determine how much of the time is spent in each phase of the code (generating random points, clustering, gravitational dynamics).

   Suitable values may be $N = 10^6$, $k = 16$, $D = 4$, $c = 4$.

   Some trade-offs between speed and accuracy you may consider are:

   (a) k-means doesn't find a perfect clustering. You run it multiple times and take the best output.

   (b) The effect of gravity due to an individual point a long way away is small. You can replace the force due to many points by the force due to one "average" particle with the sum of the masses. This is explained in the lecture notes. How should you decide when points are far enough away to do this?

   (c) Points that started off close to each other may end up being far apart. If you use the approximation in part b above, is it worth updating the groupings occasionally?

   (d) The algorithm given in the lecture notes for the N-body problem is called the Euler method. There are much faster methods, which are more work to implement (and often more work to understand). See `https://en.wikipedia.org/wiki/Midpoint_method` and `https://en.wikipedia.org/wiki/Numerical_methods_for_ordinary_differential_equations`.

   Even if you don't have time to implement these, you can discuss their advantages and disadvantages, if you wish. In particular, consider how they interact with parallelization.

3. Write a minor report (3000 words (+/- 25%) not including figures, tables, diagrams, pseudocode or references) with the following sections and details:

   (a) Introduction (400 words): define the problem as above in your own words and discuss the parallel technique that you have implemented. Present the technique using parallel pseudo-code. Cite any relevant literature that you have made use of, either as a basis for your code or for comparison. This can include algorithms you chose not to use along with why you didn't use them.

   (b) Methodology (500 words): discuss the experiments that you will use to measure the performance of your program, with mathematical definitions of the performance measures and/or explanations using diagrams, etc.

   (c) Experiments (500 words): show the results of your experiments, using appropriate charts, tables and diagrams that are captioned with numbers and referred to from the text. The text should be only enough to explain the presented results so it is clear what is being presented, not to analyse result.

   (d) Discussion and Conclusion (1600 words): analyze your experimental results, and discuss how they provide evidence either that your parallel techniques were successful or otherwise how they were not successful or, as may be the case, how the results are inconclusive. Provide and justify, using theoretical reasoning and/or experimental evidence, a prediction on the performance you would expect using your parallel technique if the number of nodes were to increase to a much larger number; taking architectural

aspects and technology design trends into account as best as you can - this may require some speculation.

For each test case, there will be a (generous) time limit, and code that fails to complete in that time will fail the test. The time limit will be much larger than the time taken by the sequential skeleton, so it will only catch buggy implementations.

(e) References: cite literature that you have cited in preparing your report.

Use, for example, the ACM Style guide at `https://authors.acm.org/proceedings/production-information/preparing-your-article-with-latex` for all aspects of formatting your report, i.e., for font size, layout, margins, title, authorship, etc.

3 Assessment Criteria

Assessment is divided between your written report and the degree of understanding you can demonstrate through your selection and practical implementation of a parallel solution to the problem. Your ability to implement your proposed parallel solution, and the depth of understanding that you show in terms of practical aspects, is called "software component". In assessing the software component of your project the assessor may look at your source code that you submit and may compile and run it to verify results in your report. Programs that fail to compile, fail to provide correct solutions to any problem instances, and/or fail to provide results as reported may attract significant loss of marks. The remaining aspects of the project are called "written report". In assessing the written report, the assessor will focus solely on the content of the written report, assessing a range of aspects from presentation to critical thinking and demonstration of a well designed and executed methodology for obtaining results and drawing conclusions.

The assessment of software component and written report is weighted 40/60, i.e. 40% of the project marks are focussed on the software component and 60% of the project marks are focussed on the written report.

Assessing a written report requires significant qualitative assessment. The guidelines applied for qualitative assessment of the written report are provided below.

4 Quality Assessment Guidelines

A general rubric that we are using in this subject is provided below. It is not criteria with each criterion worth some defined points. Rather it is a statement of quality expectations for each grade. Your feedback for your written assessment should make it clear, with respect to the quality expectations below, why your submission has received a certain grade, with exact marks being the judgement of the assessor. Bear in mind that for some assignments, independent discussions (paragraphs) in the assignment may be attracting independent assessment which later sums to total the final mark for the assignment. As well, please bear in mind that assessors are acting more as a consistent reference point than as an absolute authority. Therefore while you may disagree with the view point of the assessor in the feedback that is given, for reasons of consistency of assessment it is usually the case that such disagreement does not lead to changes in marks.

Quality expectations:

=80% - H1 A very good, excellent or outstanding discussion, with at most only minor improvements to conceptual expression or wording that can be identified. A grade in this range is generally considered to reflect the possibility of continuing with research higher degree study in the future and usually about 10% to 20% of students would be awarded this grade.

70%-79% - H2 A good discussion with no significant shortcomings, however there are one or more aspects of the discussion that can be clearly improved. Some concepts may be awkwardly expressed or in doubt.

65%-69% - H3 A reasonable discussion that addresses the question but with one aspect of the discussion that is significantly poor in writing style, understanding or missing all together. Usually 75% of students would receive a grade of H3 or above.

50%-64% - P The discussion does not entirely address the question - it is considered to be off topic in some ways, and there is more than one aspect that is significantly poor in writing style, understanding or missing all together.

0%-49% - F The discussion shows a clear lack of understanding/effort, or clearly misunderstood or underestimated what was expected and/or has significant writing style issues. Usually less than 5% of students would receive this grade.

When considering writing style, The "Five C's of Writing" is adapted here as a guideline for writing/assessing a discussion:

Clarity - is the discussion clear in what it is trying to communicate? When sentences are vague or their meaning is left open to interpretation then they are not clear and the discussion is therefore not clear.

Consistency - does the discussion use consistent terminology and language? If different terms/language are/is used to talk about the same thing throughout the discussion then it is not consistent.

Correctness - is the discussion (arguably) correct? If the claims in the discussion are not logically sound/reasonable or are not backed up by supporting evidence (citations), or otherwise are not commonly accepted, then they are not assumed to be correct.

Conciseness - is the discussion expressed in the most concise way possible? If the content/meaning/understanding of the discussion can be effectively communicated with less words, then the discussion is not as concise as it could be. Each sentence in the discussion should be concisely expressed.

Completeness - is the discussion completely covering what is required? If something is missing from the discussion that would have significant impact on the content/meaning/understanding of what it conveys, then the discussion is incomplete.

5 Submission

Your submission must be via Canvas (the assignment submission option will be made available closer to the deadline) and be either a single ZIP or TAR archive that contains the following files:

- `solution.cc`: Ensure that your solution is a single file, and include comments at the top of your program that show how to compile and run your program, including examples of how to test the correctness and performance of your program, using additional test case input files if you require.
- `Makefile`: A simple makefile to compile your code.
- `Report.pdf`: The only acceptable format is PDF
- test case inputs: Any additional files, providing test case inputs, that are needed to follow the instructions you provide in your solution's comments section. (Note that we will also test your code on our own test cases to check for correctness.)