CS599: High Performance Computing

Final Project

1 Preliminaries

You are expected to do your own work on your project (unless you do a group project). You may (and are encouraged to) engage in discussions with your classmates regarding the assignments, but specific details of a solution, including the solution itself, must always be your own work. See the academic dishonesty policy in the course syllabus.

2 Submission Instructions

You should turn in an electronic archive (.zip, .tar., .tgz, etc.). The archive must contain a single top-level directory called CS599_project_NAME, where "NAME" is your NAU username. Inside that directory you should have all your code (no binaries and other compiled code). In the event that I cannot compile your code, you may (or may not) receive an e-mail from me shortly after the project deadline. This depends on the nature of the compilation errors. The project needs to be turned in via BBLearn.

3 Project "Presentation"

This is just a short ∼5 minute presentation describing your project and what you learned.

4 Group Projects

Since the projects are open-ended, it may be useful to work in a group to brainstorm ideas. You may work in groups of 2 or 3, depending on the project. Note that if you work in a group, I will have higher expectations than if you work alone. But this should not prevent you from working in a group. E-mail me to ask about working in a group by at least one week before the project proposal deadline.

5 Important Dates

- Project disseminated: Monday March 1, 2021
- Deadline to work in a group: Monday March 8, 2021.
- Project proposal due (at the latest): Wednesday March 17, 2021. However, please try to prepare your project proposal much earlier than this date. Upload it to BBLearn and e-mail me to let me know you have submitted it for review.
- Project preliminary report due: Wednesday April 7, 2021
- Project due (report, source code): Friday April 30, 2021
- Total time to work on the project: 61 days.

6 Project Options:

1. **Design your own project** – You are highly encouraged to design your own project which could be for your MS thesis, PhD dissertation, job, etc. Why? Because if you are motivated to work on a project, then you are more likely to do a better job and get a good grade. It would be beneficial if the

project helped you towards degree completion. I am happy to meet with you and your advisor (if you have one) to outline a project that helps you make progress towards your degree.

2. **Select a project from a list of projects** – These are rough outlines of project ideas. Many of them I have not given much thought to, but they seem like they would make interesting final projects.

3. **Select a project that is related to my research** – These are projects that are related to my research. The project outlines are a bit more detailed, but that doesn't mean they require more work than the other projects.

7 Overall Project Constraints

You can use the following languages, APIs and libraries: C, C++, OpenMP, PThreads, CUDA, OpenCL. Note that we do not learn about all of these in class (e.g., CUDA), but if you already have a background in one of these languages, you may use it if desired. If there is a library not included above, feel free to e-mail me to inquire if you can use it, and what you intend to use it for.

8 Designing Your Own Project

There are many different types of projects that can be designed. Some, if not most projects are of the form: "I have some code that I want to make more efficient" (i.e., parallelize), others might be more theoretical, such as: "I would like the understand the effect of network topologies on application performance". Regardless of the type of project, it must not be a trivial undertaking. The project is worth 35% of your grade, so you should expect to spend a substantial amount of time working on it.

Examples of trivial implementations typically include embarrassingly parallel programs. If the solution is trivial and there isn't anything computationally interesting, then you won't be able to write a 5 page report on your project. Here are some examples:

- I want to find the biggest number in a list of numbers using MPI. Problem: embarrassingly parallel.
- I want to calculate π using the Monte Carlo method on a bunch of computers. I will use a hybrid approach using Pthreads and MPI. Problem: embarrassingly parallel; tried to make it more convincing by using a hybrid approach.
- Shared-memory matrix multiply: might be interesting depending on what you want to do. But, typically you want to compute a matrix larger than the memory capacity of your machine, so you need to use distributed-memory.

9 Project Deliverables

All project deliverables must be written using the IEEE Transactions template. I will provide you with this template. I would very much prefer it if you use LaTeX for this, but there is a Word template as well. If you do not want to install LaTeX on your machine, you can use the online editor, Overleaf, instead.

9.1 Project Proposal [5 Points]

Try to prepare this proposal as soon as possible and e-mail me when you have uploaded it to BBlearn for review. See the important dates section for the final proposal deadline.

You must write a 1-3 page proposal on your project design. This is very important if you are designing your own project, and a bit less important if you are selecting a project from the predefined list. Since the projects will vary substantially, some of the project proposal elements will be more (or less) important. I will read the project proposals and within a few days approve or require revisions to the proposals. If your proposal requires revisions, this is likely because your project is not challenging enough, or you did not

describe the project in sufficient detail. You may need to revise your proposal or write a new one. Thus, it's important that you give this substantial thought. If you work in a group, you only need to submit 1 project proposal. At minimum, the project proposal (for programming projects) should outline the following below. The proposal will be slightly different if you do not select a programming project.

- Your project idea; i.e., what the project entails. Maybe this includes some pseudocode, a flow chart, or something else that illustrates what you intend to do.
- Anticipated challenges in the project, as related to the concepts we have learned (or not) in class. I cannot possibly cover all shared- and distributed-memory concepts, and other high performance computing issues in a single course. Feel free to incorporate other concepts into the project.
- Preliminary ideas for a proposed solution. How will you solve your problem?
- A preliminary benchmark or result. For instance, you may have a piece of code you want to parallelize or scale, and you show how long it takes to run the program under typical conditions with 1 process. Show what you may expect to achieve when optimizing the program. For instance, I'd like to process 100 GB of data across 4 nodes on Monsoon. This should take roughly 1 hour.
- A checklist of milestones that you expect to achieve, including the expected outcome of the project and when you expect to achieve these milestones. Your project will be partially graded based on this checklist, so it is important to outline achievable goals. Including a Gantt chart is probably overkill.

9.2 Project Preliminary Report [10 Points]

Part way through the project you will submit a preliminary progress report. The progress report should describe the progress you have made thus far, and will reference the milestones outlined in the project proposal. Any changes to the project timeline or scope should be discussed. The preliminary report should be about 3 pages. A checklist of items to include is below:

- An overview of the progress made on your project.
- Specific challenges you have addressed in the project thus far.
- Remaining items that you need to address in your project.
- Preliminary results. If you are improving the performance of an application, a result may include: a plot of time vs. threads or processes, a speedup plot, weak scaling results, number of cache misses before and after optimization, the number of nodes accessed in a data structure, etc. If you are not improving the performance of an application, then your preliminary results will differ depending on the project.
- Copy and paste the checklist that you developed in the project proposal and check off those tasks that have been completed. Add tasks as necessary with accompanying discussion.

The preliminary report should not be used to propose to study something. This is to be completed in the project proposal.

9.3 Final Project Report [20 Points]

Turn in a report on your project that is $\gtrsim$5 pages in length. The items in the preliminary report can be included, but updated to reflect new progress that you made on the project. The final project report should have the same structure as a peer-reviewed paper. Example sections are outlined below with a brief description of each.

- Introduction: What are you studying and why is it important? Give an overview of the problem and the contributions you have made to solving the problem.
- Background: Any background information that you believe is relevant. Do a brief literature review (depending on your project) and report what other research has done to solve the problem. Briefly contrast your approach to the literature.
- Description of your approach to the problem studied: This should include all technical detail required to understand the problem and how you solved it. This will be the main section of your report. Example material may include figures, pseudocode, and algorithm descriptions.
- Results: The results of your project. If you are improving performance this will show how various optimizations impact performance. Include experimental methodology here, such as the number of time trials, the hardware platform, compiler settings (optimization) and versions, libraries, APIs, datasets, and other relevant information needed to reproduce your work.
- Discussion and Conclusion: Discuss the outcome of the project and conclude the work. Highlight interesting results and lessons learned from the project.

10 List of Projects

10.1 Literature Review

Do a literature review on a research area in high performance computing. This project is probably much more difficult than it seems, as writing a good literature review is known to be very difficult. The literature review will likely be 10–20 pages, and contain roughly 20-40 references (this is a guideline, not a requirement, the page length and number of references may vary). If you decide to write a literature review, please let me know, as your proposal, preliminary report, and final report will be different than that outlined above.

A literature review should do the following:

1. Outline the state-of-the-art in a particular research area. Give the major points or "takeaways" from the literature review.
2. Categorize the literature. For example, shared-memory vs. distributed-memory algorithms, exact vs. approximate algorithms, etc.
3. Describe some of the open questions in the area. What are the major challenges?
4. What should be studied in this area in the future? These should be things that lead to interesting research directions.

10.2 Batch Scheduler

The idea for this project is to simulate batch scheduling algorithms and see how they behave for various scenarios, and assumptions. You would have to build a batch scheduler (that keeps track of the schedule and of compute node usage). This scheduler should implement FCFS, conservative backfilling, and Extensible Argonne Scheduling System (EASY). The input to the simulator should be actual supercomputer workloads as available from The Parallel Workloads Archive (http://www.cs.huji.ac.il/labs/parallel/workload/). Interesting things to study would be the impact of the accuracy of the user estimates for job durations (in simulation you can replay the estimates given by users, but also "fake" perfectly accurate estimates or fudge them in whatever way).

10.3 Distributed-memory Matrix Multiply

Implement a distributed-memory matrix multiplication that implements multiple algorithms (Cannon, Fox, Snyder). There could be a shared-memory component as well.

10.4 Distributed-memory Stencil

Implement a distributed memory stencil application with all of the bells and whistles in terms of data distribution. There could be a shared-memory component as well.

10.5 Shared-memory Sorting

Do a literature review on fast multi-threaded sorting. Download available code. Perform performance evaluations. Implement some of the techniques in your own sorting code. What works? What doesn't? Why? One big issue here is locality.

11 Projects Related to My Research

11.1 Parallel Distance Similarity Self-Join

A simple operation that is used in many applications is calculating distances between points using the Euclidean distance. For example, if $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, then the distance is as follows: $d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. You will calculate the distances between points and report the total number of distances between points that are within some distance threshold, ϵ. You will take as input a dataset D. Each point is denoted as $p_i \in D$ where $i = 1, ..., |D|$. Thus, you will count the total number of occurrences where each $p_i \in D$ are within ϵ of each other. As ϵ increases, the total number of points within ϵ should increase. If you are familiar with databases, this is considered a join operation between a table and itself, denoted as follows: $D \bowtie_\epsilon D$.

The Euclidean distance is reflexive, i.e., $d(p_1, p_2) = d(p_2, p_1)$. You must "double count" these pairs. That is, if p_1 and p_2 are within ϵ then p_2 and p_1 are also within ϵ, and both must be counted towards the total number of points within ϵ of each other. You must also add to the total count the distance between a point and itself. E.g., $d(p_1, p_1) = 0$ will always be within ϵ. These requirements make the total count easier to compute, as these are considered corner cases.

The brute-force algorithm for computing the distances between all pairs of points is called the Nested Loop Join and is $O(n^2)$. You may use this for validating your work to determine if you got the correct result.

```
//Nested Loop Self-join pseudocode

int N=numPoints;
for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
        {
                distance=calcDistance(data[i],data[j])
                if (distance <= epsilon)
                {
                        count++
                }
        }

        printf("Total within the distance: %d ",count);
```

You must not simply parallelize the nested loop join. It is embarrassingly parallel and inefficient. The point of the assignment is to use other methods to reduce the total amount of work. There are two ways to accomplish this: (i) do not compare all points to each other; and (ii) perform less work when determining if two points are within ϵ of each other. You must use (i) above, which is the only way to reduce the quadratic time complexity of the algorithm.

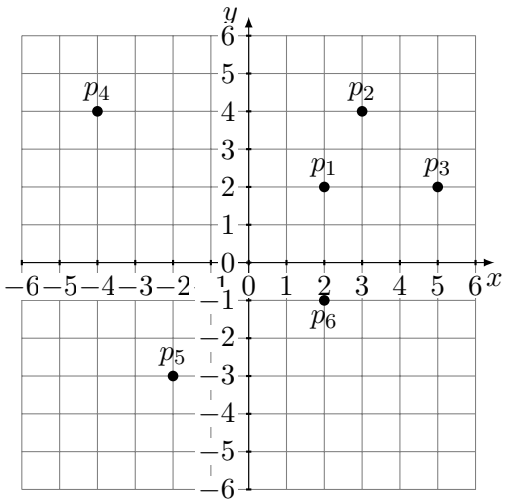

Figure 1: Example calculating distances.

As an example, consider Figure 1. If the the distance is $\epsilon = 3$, then p_1 and p_2 are within ϵ of each other. However, p_1 and p_5 are too far away from each other. Thus, it does not make sense to calculate the distance between p_1 and p_5. You should examine ways to prune similar distance calculations. Hint: look up spatial indexing schemes online.

You must parallelize this program using shared and/or distributed memory. Your program only needs to work in 2-D (not n dimensions).

11.2 Parallel Density-Based Clustering

Clustering is used in data analysis workflows to find objects that are similar to each other. In spatial applications, this means objects that are nearby each other in space are grouped together. In this project, you will parallelize the DBSCAN clustering algorithm. The pseudocode for DBSCAN is below.

```
//pseudocode from wikipedia.

DBSCAN(DB, dist, eps, minPts) {
   C = 0                                                /* Cluster counter */
   for each point P in database DB {
      if label(P) ≠ undefined then continue             /* Previously processed in inner loop */
      Neighbors N = RangeQuery(DB, dist, P, eps)        /* Find neighbors */
      if |N| < minPts then {                            /* Density check */
         label(P) = Noise                               /* Label as Noise */
         continue
      }
      C = C + 1                                         /* next cluster label */
      label(P) = C                                      /* Label initial point */
      Seed set S = N \ {P}                              /* Neighbors to expand */
      for each point Q in S {                           /* Process every seed point */
         if label(Q) = Noise then label(Q) = C          /* Change Noise to border point */
         if label(Q) ≠ undefined then continue          /* Previously processed */
         label(Q) = C                                   /* Label neighbor */
         Neighbors N = RangeQuery(DB, dist, Q, eps)     /* Find neighbors */
```

```
            if |N| ≥ minPts then {                    /* Density check */
               S = S ∪ N                              /* Add new neighbors to seed set */
            }
         }
      }
   }
```

This algorithm is characterized by an *irregular instruction flow*. Clusters grow by chaining points on to each other. This results in a bunch of if statements, as the neighborhood searches add more points to the cluster (see wikipedia for an example). Due to the irregular instruction flow, this algorithm provides only modest speedups in shared-memory, which makes it challenging to parallelize.

Parallelize the algorithm in shared and/or distributed memory. Similarly to the Parallel Self-join above, you must not perform the $O(n^2)$ version of this algorithm (comparing all points to each other in the range queries). Transform the algorithm to reduce the number of point comparisons. You must test your clustering algorithm on two real-world datasets, but they can be limited to 2 dimensions. This project has overlap with the self-join.

11.3 Parallel K-Means

You will implement a K-means clustering algorithm in shared- or distributed-memory that is optimized and does not compute all of the distance calculations. Some example algorithms include: Elkan's Algorithm, the Annulus Algorithm, Drake's Algorithm, and the Yinyang Algorithm. You may not implement Lloyd's algorithm (the naive K-means algorithm).

11.4 Parallel K-Nearest Neighbors

Lookup the K Nearest Neighbors problem online. Solve the 2-D KNN problem using distributed memory without using the $O(n^2)$ algorithm. This project has overlap with the self-join and clustering.

12 Some Things To Study

Here are some examples of things you might want to study, depending on the project. Remember to use the techniques/methods used in the assignments (e.g., use a single node generation, use the exclusive flag, etc.).

1. You parallelize an application. Performance comparison between: 1) shared-memory only; 2) distributed-memory only; 3) hybrid with threads and processes.
2. If you have a GPU or want to use one on Monsoon, you execute your program using CUDA or OpenCL. Compare CPU vs. GPU implementations.
3. How much load imbalance do you have in your distributed- or shared-memory implementation? How did you address load imbalance?
4. In distributed-memory applications with lots of communication: were you about to hide some of the communication overhead by overlapping it with computation?
5. How does your program scale on Monsoon? Show strong and/or weak scaling results. All projects improving performance require reporting the scalability of the algorithm, ideally on multiple nodes. It may be neat to run your program on > 100 cores, depending on the project.

13 Miscellaneous Q&A

I will add to this section as I receive/think of more misc. questions and answers. Feel free to ask questions on the BBLearn forum as well.

- Scenario: Let's say you're running your program on up to 32 MPI ranks on Monsoon. You're performing a performance evaluation to see how your program scales on 1, 2, 4, 8, 16, 32 ranks. You collect your results, describe what you found and write up your results. Then, you realize you found a bug in your code the day before the project is due! So you rerun your scripts to perform the performance evaluation. For some reason when you execute your script for 16 MPI ranks this time, the job fails. You have the data for all of the data points for a plot except for 16 ranks. In this case, don't worry and just note that your experiment failed for 16 ranks.
- If your project requires parallelizing a program in shared and/or distributed memory, then you must use Monsoon for assessing scalability. You may not use a laptop or desktop that only contains a few cores.
- If your project requires parallelizing a program in shared and/or distributed memory, you must use reasonable input sizes. For example, your algorithm should not run well below 1 second when parallelized. Small input sizes typically have large overheads relative to the overall execution time.