FINAL REPORT – Reading Course on Scientific & High-Performance Computing

Questionnaire

1. From the list of topics covered:

```
[ sn | I ]  Scientific Computing Motivation
[ K | NA ]  Introduction/Review of C++
[ K | NA ]  Modular Programming
[ K | NA ]  Libraries
[ K | NA ]  Make
[ K | NA ]  Version Control
[ K | NA ]  Debugging
[ K | NA ]  Unit Testing
[ K | NA ]  File I/O: self-describing formats, netCDF, HDF5, ...
[ K | NA ]  Numerics
[ sn | si ]  Numerical Linear Algebra
[ N | si ]  ODEs
[ N | I ]  Molecular Dynamics, N-body simulations
[ N | si ]  PDEs
[ N | I ]  FFT
[ K | I ]  Random numbers
[ N | I ]  Monte Carlo
[ sn | si ]  Profiling
[ sn | si ]  Optimization
[ N | I ]  Supercomputing/HPC
[ N | I ]  Shared-memory programming, OMP
[ N | I ]  Distributed-memory programming, MPI
[ sn | I ]  Other parallel approaches, technologies
```

Please add any topic(s) that we may have covered and is not included in the list above.

1.a) Please describe which topics, from the ones covered in the material were:

- new **(N)**,
- not new but some elements were novel **(sn)**,
- already known **(K)**, i.e. not new.

1.b) Also indicate whether these were:

- interesting **(I)**,
- somehow interesting **(si)**,
- not quite appealing **(NA)**.

2. Please indicate which topics you consider should have been covered with *more* or *less* details?

My favourite part ws the Molecular dyncmics part, and my peer's presenatation on it. I would love to have an instructor lead lecture devoted to it. This is really biased though, as this specifically pertains to my interests. I would also love to indulge in that topic, after the termination of this course. What should be covered less? I think in the first couple weeks, we spent a lot of time covering material, that was already covered in A48, B07, B09, etc. I think it would be a good idea to shorten that to a week, so we have more time to dive deeper into the other more interesting newer topics.

10. Please provide any comments about the provided *lectures notes* on "Practical Scientific Computing" by R.Van Zon and M.Ponce.

I loved the notes, they were perfectly split up into sections. They provided you with all the background infromation you need about a topic in a short and concise way. The consiceness is perfect as it peaks interest, and you can go about learning the details on your own. I would not have the textbook any differently.

11. If this would have been a *curricular* course offered to CMS students, please provide comments, recommendations or suggestions in how to improve, change, and/or adjust the material covered.

Reduce time spent on more trivial sections, about modularity, git, makefiles, etc. More presentations, I feel like these were the best oppurtunity to learn, I think it may be difficult to have a lot of these presentations in a big class though. I also think more time should be spent on the second half of the course.

11.i Which topics/courses would you say would have been useful to know before in order to take the best advantage of this course? CSCA48, CSCB09, CSCC69, CSCB07, MATA22

Report

i. Write a report describing the exercises that you solved, including which parts you found the most challenging ones for each of them and how did you tackle them.

Please indicate the location (URL) of the repository where you worked on these problems.

Repo with all exercises: https://github.com/SalikChodhary/CSCD92

Exercises part 1 were simple, as the point of them was to just get our hands wet with coding in C++. I did all 3 exercises. Ex1 was simple, just a sliding window, that calculates the average of the window. Ex2 sounded hard, but turned out to be really easy as well, it essentially outputted the values of f and g, two functions that plot the lissajous shapes. Ex3 was also very simple, it just required printing out the values of a simple function, and also printing out the count.

Exercises part 2 were more crunchy an dinteresting, I loved working on them. The first one and my favourite one was the 1D Nagel-Schrechenberg Traffic Model. I love cars, and I always have wanted to study traffic patterns, and this was my first time working on something related to that. I wrote the data generation part in C++, and the visualization part in python. Since, it's just easier to use matplotlib in python.

Then, I also worked on the eigenvectors exercise, because I did an assignment on Linear Algebra, so it made most sense, to try out the things I discussed in that presentation. The most chanllenging part for me was the setup for LAPACK, it almost took me a day. This was because I could not find the LAPACK module. Then, when I found it, I was not compliling as per MKL instructions, and was getting unrecognized errors, but I was able to solve them by reading more documentations, and looking at other examples.

After that, I worked on the parallel workflow assignment and the integration and shared memory parallelization assingments. The latter was easy, and just required following steps. The hardest part was remembering the commands to use, as I did this a couple weeks after our lecture on this topic. So, I just had to go through the lecture notes again, to figure out what exact commands to use. Then, in the second integration assignment, it was hard to figure out the pragma syntax, but after numerous attempts, I was able to figure them out. The hardest part was the scaling analysis, I just wrote 2 scripts to handle "weak" and "strong" scaling. It was fun, and can see this being used in the industry a lot, as there is a lot of parallelizing in the real world.

ii. Write a short report about what you consider is the most interesting aspects of Scientific Computing and High-Performance Computing. Also include comments on how you consider (if there is any of) these topics that could result to be useful for your career path.

Most interesting - N-body simulations

N-body simulations are computational techniques used to model the behavior of systems comprising multiple interacting particles. These simulations enable scientists and researchers to study the dynamics of a wide range of phenomena, such as the motion of celestial bodies, molecular interactions, and fluid dynamics. The core principle involves numerically solving the equations of motion for each particle while accounting for the forces and interactions between them. By employing these simulations, we can gain insights into the emergent behavior of complex systems that are otherwise difficult to study through traditional analytical methods.

Application in the Self-Driving Car Industry: One of the most intriguing applications of N-body simulations lies in the realm of the self-driving car industry. The development and validation of autonomous vehicles require extensive testing under diverse conditions, which is often impractical and unsafe to conduct solely in real-world environments. N-body simulations offer an invaluable solution by allowing engineers to create virtual environments where they can model the interactions between vehicles, pedestrians, and infrastructure elements. These simulations can accurately replicate complex traffic scenarios, weather conditions, and potential collisions, providing a controlled testing ground to evaluate the safety and efficiency of self-driving algorithms.

Benefits and Contributions: The benefits of utilizing N-body simulations in the self-driving car industry are multifaceted. Firstly, the ability to test various scenarios without real-world risks significantly accelerates the development process, reducing both time and costs. Secondly, the simulations enable the exploration of extreme and rare events, aiding in the identification of potential vulnerabilities in the self-driving system. Moreover, the gathered data can be used to optimize algorithms and improve decision-making mechanisms, thereby enhancing the overall reliability of autonomous vehicles. Ultimately, the integration of N-body simulations in the self-driving car industry holds the promise of safer and more advanced autonomous systems.

Relevance to My Career Path: As an aspiring professional in the self-driving car industry, the profound relevance of N-body simulations to my career path is evident. Engaging in the development and optimization of autonomous systems requires a deep understanding of their behavior in complex and unpredictable environments. By mastering the art of N-body simulations, I can contribute to the creation of safer and more efficient self-driving algorithms. This expertise will enable me to tackle real-world challenges, innovate solutions, and advance the field of autonomous transportation.

iii. Consider that you are given a code which is required to be optimized. The code can take dataset and the size of the data sets affects the different metrics of the performance (i.e. running time, memory utilization, IO, communication, etc.)

a) What steps would you take in order to study and draft a performance analysis report of this code?

Optimizing code for better performance involves a systematic approach to identifying bottlenecks, analyzing different metrics, and implementing improvements. Here's a step-by-step guide on how to study and draft a performance analysis report for the given code:

Understand the Code:

Begin by thoroughly understanding the codebase. Identify the main algorithms, data structures, and dependencies. This understanding will help you identify potential areas for optimization.

Profile the Code:

Use profiling tools to identify performance bottlenecks. Profilers help you pinpoint which parts of the code are consuming the most resources and taking the most time. This will guide your optimization efforts towards the most impactful areas.

Identify Bottlenecks:

Analyze profiling results to identify bottlenecks. These could be CPU-intensive operations, excessive memory allocation, inefficient data access patterns, or suboptimal algorithm implementations.

Optimization Strategies:

Depending on the bottlenecks, apply appropriate optimization strategies. These could include:

Algorithmic improvements: Replacing inefficient algorithms with more efficient ones. Data structure optimizations: Using data structures that minimize memory usage and improve data access patterns. Parallelization: Leveraging multi-threading or parallel processing for CPU-bound tasks. Caching: Storing frequently accessed data in memory to reduce IO operations. I/O optimizations: Reducing the frequency of file/database/network operations. Memory management: Avoiding memory leaks and reducing memory fragmentation. Code refactoring: Organizing and restructuring the code to improve readability and maintainability.

iv. You are in the process of submitting an allocation request for compute usage to one of the national systems, for which you must submit an scaling analysis of the jobs you will run. For details on this process, see https://alliancecan.ca/en/services/advanced-research-computing/research-portal/resource-allocation-competitions/rac-frequently-asked-questions

a) What type of information should you submit?

Project Description: Provide a clear and concise overview of your research project. Describe the scientific objectives, methodologies, and significance of your work.

Computational Workload: Outline the computational tasks you plan to execute. Explain the algorithms, simulations, data analyses, or other computations involved.

Scaling Analysis: Detail the scaling analysis of your jobs. This analysis assesses how the computational workload's performance scales with the number of processors or cores used. Include strong and weak scaling analysis if relevant (explained further in the next section).

Resource Requirements: Specify the amount of compute resources (CPUs, GPUs, memory, storage, etc.) needed for each job and the total resource requirement for the entire project.

Software and Tools: List the software, libraries, and tools required to run your jobs. Highlight any specialized software that may need installation.

Execution Strategy: Explain how you plan to distribute and parallelize your computations across the available resources.

Timeframe: Indicate the duration of your project and the estimated time required for each computational job.

b) Should you also submit a strong and weak scaling? Justify. Both strong and weak scaling analyses are valuable to assess the efficiency of your computational jobs:

Strong Scaling: Strong scaling measures how well a parallel program performs as the number of processors (or cores) increases while keeping the problem size constant. You should submit a strong scaling analysis to show how the execution time changes as the resources are scaled up. This is important to determine the efficiency of your code and identify potential bottlenecks in your algorithms or parallelization strategy.

Weak Scaling: Weak scaling analyzes the performance of a parallel program as both the problem size and the number of processors are increased proportionally. Submitting a weak scaling analysis demonstrates how well your code handles larger problems with increased computational resources. It helps in assessing whether the computational efficiency remains consistent as the workload grows.

Submitting both strong and weak scaling analyses provides a comprehensive view of your code's performance characteristics. It shows how well your code scales with additional resources and helps allocate the appropriate amount of computational power for your research. It also allows the allocation committee to understand the efficiency of your code across various scenarios and make informed decisions about resource allocation.