The Design of a Practical Flipped Classroom Model for Teaching Parallel Programming to Undergraduates

Dirk Colbry
Michigan State University
East Lansing, MI
colbrydi@msu.edu

ABSTRACT

This paper presents a newly developed course for teaching parallel programming to undergraduates. This course uses a flipped classroom model and a “hands-on” approach to learning with multiple real-world examples from a wide range of science and engineering problems. The intention of this course is to prepare students from a variety of STEM backgrounds to be able to take on supportive roles in research labs while they are still undergraduates. To this end, students are taught common programming paradigms such as benchmarking, shared memory parallelization (OpenMP), accelerators (CUDA), and shared network parallelization (MPI). Students are also trained in practical skills including the Linux command line, workflow/file management, installing software, discovering and using shared module systems (LDMOD), and effectively submitting and monitoring jobs using a scheduler (SLURM).

Keywords

Computational science, Flipped classroom, Parallel programming.

1. INTRODUCTION

Established in 2015, the Department of Computational Mathematics Science and Engineering (CMSE) at Michigan State University (MSU) represents a new discipline at the intersection between methods (math and computer science), domain applications (science and engineering) and computation (programming and large-scale computing). CMSE’s mission is to advance the use of computational methods in all areas of scientific research and engineering within the university [1]. This includes the training of undergraduate and graduate students from a wide variety of STEM (science, technology, engineering, math) and non-STEM majors in how to best utilize computation as they become experts in their own fields. Our first two introductory courses (CMSE 201 and 202) teach students programming, computational modeling techniques [2], and tools for computational modeling (similar to and motivated by software carpentry [3]). Our latest course, which is the focus of this paper, is “Methods in Parallel Programming” (CMSE 401). This course is intended for advanced students who would like to speed-up their research and utilize advanced computational hardware.

© 2021 Journal of Computational Science Education
DOI: https://doi.org/10.22369/issn.2153-4136/12/2/10

By the end of CMSE 401, students will be able to:

- Give examples of major science and engineering domains that use parallel programming and of the common types of algorithms that need large scale computing (e.g. the seven dwarfs of HPC).
- Demonstrate the ability to access, navigate, and use a variety of advanced computing systems with remote Linux connections (ssh, module systems, BASH, text editing, file systems, software install and building, environment variables, schedulers, etc.).
- Analyze software by conducting profile and benchmark studies with different parameters and options. Explain the bottlenecks and scaling of the code and present results to peers with predictions of times and scaling.
- Summarize the fundamentals of parallel programming concepts, including strong and weak scaling, Amdahl’s Law, communication overhead, locks, and racing conditions.
- Explain differences between major parallel hardware and software paradigms. Compare and contrast the different approaches and be able to choose appropriate tools for a given problem.
- Develop and evaluate parallel codes using a variety of paradigms, including pleasantly parallel, shared memory parallelization (e.g. OpenMP), accelerator (e.g. GPUs and FPGAs), shared network parallelization (e.g. MPI, Hadoop, and Charm++), and parallel libraries (e.g. cupy, numba, mkl, fftw and blas).

The remainder of this paper discusses the major components of the design of CMSE 401, gives selected examples, and provides some limited analysis of the material though student feedback.

2. COURSE DESIGN

This course uses a “flipped classroom” model, where students spend class time doing hands-on practice activities with instructors and classmates, while traditional lectures are replaced with time outside of class reading and watching videos. When done correctly, this model of teaching is believed to provide a richer learning environment for students [4].

2.1 Jupyter Notebooks

All of the course materials are provided to the students using a Git repository and Jupyter notebooks [5]. The use of Jupyter notebooks may be confusing, since Jupyter notebooks are traditionally linked to Python, which is not a traditional language when considering computational performance and parallelization. However, Jupyter notebooks are rich and efficient communication tools that combine the benefits of a multimedia webpage, LaTeX, and executable example code. We develop Jupyter notebooks as a kind of interactive textbook that, when used properly, is an effective way to organize course content and communicate with students. Although we do spend some time doing Python examples (all the students are familiar with Python from prerequisite courses), most of CMSE 401 is taught using the C family of languages (C, C++, and CUDA), which are also familiar to students from another C++ prerequisite.

One particularly useful Jupyter feature is the `%%writeout` "magic" command that allows the contents of a Jupyter cell to be written out to a file. This feature, in combination with the ability to execute bash commands using the "`!`" prefix, allows a Jupyter notebook to provide example code in any language, compile the code and run it all from within the notebook. In this way, students can have fully "literate" programming [6] with explanations right next to example code.

2.2 Course Hardware Resources

Students in CMSE 401 have access to a Jupyterhub server [7], the university's High Performance Computing Center (HPCC) [8], and multiple XSEDE resources though a teaching allocation [9]. This variety of hardware was chosen to expose students to different interfaces and help them generalize their understanding of computing hardware, in the hopes that they will develop a strong foundation of understanding and be able to figure out how to utilize new resources as they are developed in the future.

In addition to traditional hardware, during the first two weeks of the course, students were introduced to two different "portable" clusters. The first was a 7-node Raspberry Pi-based system, based on the Tiny Titan (https://tinytitan.github.io/), and the second was six (6) MacBook Pros connected using a small, off-the-shelf routing hub. Each of the laptops was installed with BCCD [10] inside of a virtual desktop. Students explored both systems during class as a hands-on learning activity focused on how to connect computers in a commodity cluster format. After building this commodity cluster in class and running examples, students also toured the campus HPC facilities. These hands-on lessons and in-person tours were motivating and helped students get excited about the topics they would be studying in CMSE 401.

We also experimented with a Jupyterhub server equipped with GPGPUs and CUDA support. Since CUDA would not work on many of the students' computers, this CUDA-enabled Jupyterhub server turned out to be a useful asset when introducing students to the language.

2.3 Assignments and Assessments

In the spring of 2019, CMSE 401 met three times a week for 70 minutes. Before each class, students completed a pre-class assignment, consisting of reading, videos, and practice problems. During class, the instructor reviewed questions that came up during the pre-class activities, and then students worked individually, in pairs, and in groups on example problems. Students also worked individually on more open-ended and in-depth problems in the form of homework assignments, which were due approximately every two weeks. Three times during the semester, students were given timed exams (two midterms and one final) to help assess their learning. Finally, at the end of the semester, students presented work on individual projects relating to topics taught in class. The remainder of this section describes these activities and assignments in more detail.

2.3.1 Pre-Class Assignments

These assignments are given to students in the week prior to class and include reading, multiple short videos (5–15 minutes each), example code, and practice questions. Students are expected to go through the materials before class, so that they are ready to participate in the in-class activities. These pre-class assignments are not graded; instead, students fill out a survey at the end of each pre-class assignment with questions similar to the following:

- Approximately how long (in minutes) did this assignment take for you to complete?
- What questions do you have, if any, about any of the topics discussed in this assignment after working through the Jupyter notebook?
- Do you have any further questions or comments about this material, or anything else that's going on in class?
- Based on what you've learned in the pre-class activities, what are you hoping to learn more about in class?

These questions are designed to get an idea of where students are struggling, so the instructor can address issues during class.

2.3.2 In-Class Assignments

Before class, instructors review all questions from the pre-class assignment survey, group them by topic, and develop a mini-lecture to help structure the class time most effectively. These mini-lectures vary in length depending on the issues students highlighted from the pre-class assignment. While the instructor has in-class activities planned, it is more important to address student questions and make sure they understand the pre-class assignments than to "get through" the day's materials.

After the mini-lecture, students work through the in-class notebooks. Students are expected to help each other out and work ahead on different questions if they get stuck on one particular problem. The goal here is to train students so that they are able to find solutions themselves, with instructors available to give suggestions and encouragement in order to avoid frustration. Instructors focus on helping students understand concepts and jargon; instead of solving problems for the students, instructors walk them through a variety of problem-solving techniques and suggest terms and phrases that they could use to search for helpful solutions on the Internet.

2.3.3 Homework Assignments

Homework assignments are designed to let students explore. Although many of them start out very similarly to in-class assignments, the idea for homework is to push students and get them solving multiple problems end-to-end. Students need to figure out how to download data, write code (including submission scripts), submit jobs to schedulers, interpret results, and visualize/share their results with their peers. A key component of the CMSE 401 homework assignments is a "creative component" that allows students to do something different and creative. Examples include a contest to see who can get the fastest code, trying out a new dataset, or exploring a software package. Again, the learning goals focus on exploration and problem solving in the context of large-scale computing in order to help students develop both familiarity with specific tools and creative problem-solving skills. We hope this approach also makes CMSE 401 more fun for students.

2.3.4 *Exams*

There are two midterms and a final exam in this course. Given the highly interactive and collaborative nature of the course, these exams provide an opportunity to individually assess student knowledge and skills. In all other assignments, students are expected to work together and support each other's learning, but that approach can make it difficult for instructors to identify areas where individual students are struggling. Timed exams, where students work alone, provide an assessment of individual knowledge and progress.

Of course, excellent students who have a deep understanding of the material may not perform well on timed exams — just as some students are excellent at taking tests but may not be able to perform as well in less structured scenarios. Exams in CMSE 401 are primarily seen as learning tools and try to reflect real-world scenarios. Thus, all exams are open-network: no one programs in a vacuum, and we are assessing students' ability to find answers and develop solutions using all of the resources that would be available to them in a real-life setting. Exams include four (4) problems, each with five (5) component questions. Although the questions relate to each other, we try to write them in such a way that they can be answered correctly even if previous answers are wrong. Students' informal feedback suggests that the exams are famously challenging — yet also rewarding. Even students struggling in the course have proven able to demonstrate their knowledge through these exams, and, although these exams are primarily used as a summative assessment tool, instructors are able to formatively assess progress and adjust course content and individual student learning goals. The exam grades are just one factor in students' overall learning, and thus are a relatively small percentage of students' final grades.

2.3.5 *Student Projects*

At the end of the semester, students present unique projects that demonstrate some aspect of what they learned over the semester. At a minimum, projects are expected to contain some sort of benchmark timing comparison. However, instructors are very flexible and encourage projects that relate directly to "real-world" problems that students are encountering in their work or other classes. For example, working with an existing faculty to download, install, and run a code on the HPC is considered an excellent project for CMSE 401. Another good project is to download a parallel library or language, get it working on the HPC, and do a benchmark comparison between some of its features (e.g., Tensorflow was quite popular). Students may not necessarily do much parallel programming in their projects; instead, we focus on the more common issue of workflow management and performance measurements, as these are the tools that researchers need to utilize advanced computing systems. Some example titles of student projects include:

- Ising Model Optimization
- Numerical Relativity with Numba
- MPI Poission Equation with MPI4Py
- OSCAR (Operational Research in Scala)
- Utilizing TensorFlow for Machine Learning in Biomedical Imaging
- Parallel Optimization of Sabermetric Quantifier
- Optimizing Garfield++ For Use in Simulating a Nuclear Detector
- Parallel Optimization in FLASH
- A Charm++ Parallel Stock Market Simulator
- Breast MRI Classification using TensorFlow
- Classifying Dog and Cat Images Using TensorFlow
- Penalization of TDCI

Student projects have multiple milestones through the semester, and students present progress to their peers. Although each student works on their project individually, time is given both in-class and out of class for students to share their work, and collaborative feedback and peer review are highly encouraged.

3. COURSE SCHEDULE AND TOPICS COVERED

The semester is divided into approximately 15 weeks, and the overall course covers the following major topic areas:

Major Topic 1 — Benchmarking and compilers

Major Topic 2 — Tools of the trade (remote systems, software installs and schedulers)

Major Topic 3 — Shared memory parallelization

Major Topic 4 — Accelerators

Major Topic 5 — Shared network parallelization

In practice, rather than being a linear progression of content, these topics are woven together throughout the semester. For example, in the first few weeks of class, students are exposed to a mini cluster (Raspberry Pi and laptop BCCD cluster) and are running a variety of parallel examples (shared memory, shared network, and GPUs). When they see these topics again later in the semester, the previous exposure has prepared them to jump in and program them on their own. A more detailed list of individual modules follows:

1. How a cluster is born — basic introduction to clusters, big-iron, little-iron and accelerators
2. Languages and Compilers — Benchmarking of both interpreted (Python) and compiled languages (C/C++), code optimization (compiler flags), introduce/review Big-O notation, and practice benchmarking.
3. Command line scripting (BASH), and accessing remote systems (SSH and SCP)
4. Schedulers — unique components of a shared system (schedulers and module system) and writing single core and pleasantly parallel examples to the scheduler (SLURM)
5. Shared Memory Parallelization — students are introduced to shared memory parallelization (OpenMP) and shown/encouraged to work on personal laptops
6. Shared Memory Parallelization — more about loops and programming options; goal is to become familiar with the variety of OpenMP capabilities and not necessarily become masters
7. Accelerators — introduction to accelerator coding (CUDA) and comparisons with shared memory programming, submitting jobs to a scheduler
8. More Accelerators — learning the basics of CUDA and writing their first program
9. More Accelerators — discuss the good and bad about CUDA, understanding thread blocks and tiling — where does it work and where does it fall apart?
10. Shared Network Parallelization — understanding network throughput and latency, benchmarking MPI code on different numbers of cores and nodes

11. Shared Network Parallelization — writing their first MPI program, debugging MPI code and improving performance
12. Hybrid Systems

While this is a rough outline of the topics, plenty of room was included in the 15-week schedule to allow instructors to adapt the pacing for more or less difficult topics, respond to student feedback, and give plenty of time for students to work on homework assignments and projects.

4. EXAMPLE

Whenever possible, instructors try to ground classroom examples using real-world scientific and engineering problems as motivation. Throughout the semester, students are shown how what they are doing connects directly to on-going research. This means we try to avoid spending too much time on "toy" examples such as sorting, calculating pi, or making games (although these examples can be useful). For the interested reader, samples of classroom materials have already been drafted and can be downloaded from the following git repository:

https://github.com/colbrydi/CMSE401_Examples.git

These examples include Jupyter notebooks that contain the following:

- A pre-class assignment that includes videos on using the command line and ssh keys
- An in-class assignment on CUDA programming on a GPU enabled node running Jupyter
- Shared Memory Parallelization example homework
- An example project template
- An example exam

These examples demonstrate the style and pedagogical approach of CMSE 401. The course is being offered a second time during the spring of 2021, and all of the course materials will be available as an Open Education Resource (OER) by the summer of 2021 at the course website (http://cmse.msu.edu/cmse401). Instructors interested in the instructor materials are encouraged to reach out to the author, as we are happy to provide additional instructor notes and answers.

5. STUDENT FEEDBACK

Although no formal evaluation of the materials was conducted for this paper, all university courses are evaluated using a 21-question survey, which 12 of the students completed. The students are able to choose a rating (from the following) for each question.

1 = (S) — Superior: exceptionally good

2 = (AA) — Above Average: better than the typical

3 = (AV) — Average: typical of courses or instructor

4 = (BA) — Below Average: not as good as the typical

5 = (I) — Inferior: exceptionally poor course or instructor

Please note that this course evaluation tool is known to be fairly biased and is being reworked by the university. The author also acknowledges that the results presented do not include a controlled reference point. However, the data do provide some context. A selected summary of the results can be reviewed in **Error! Reference source not found.**.

Table 1. Summary of student feedback grouped by type.

Composite Factors	Mean	Std
Instructor Involvement (Questions 1–4)	1.10	0.23
Student Interest (Questions 5–8)	1.73	0.38
Student-instructor Interaction (Questions 9–12)	1.31	0.51
Course Demands (Questions 13–16)	1.79	0.71
Course Organization (Questions 17–20)	1.55	0.56

Table 2 shows a sample of feedback questions given to the students. Based on this feedback and some informal polling, students reported that the course was challenging which is reflected in their end of semester survey evaluations. Specifically students found the course to be highly enjoyable (Question 21) while also being intellectually challenging (Question 6). Probably the biggest informal complaint was the difficulty and length of the homework (Question 14).

Table 2. Selected questions that reflect student feedback to the content and format of the course.

#	Question	Mean	Std
3	The Instructor's concern with whether the students learned the material	1.17	0.39
4	Your Interest in learning the course material	1.17	0.39
5	Your general attentiveness in class	1.83	0.39
6	The course as an intellectual challenge	2.25	0.75
7	Improvements in your competence in this area due to this course	1.42	0.67
10	The Student's Opportunity to ask questions	1.42	0.67
12	The appropriateness of the amount of material the instructor attempted to cover	1.33	0.65
13	The appropriateness of the pace at which the instructor attempted to cover the material	1.75	0.97
14	The contribution of homework assignments to your understanding of the course material relative to the amount of time required	2.08	1.00
15	The appropriateness of the difficulty of assigned reading topics	1.67	0.78
17	The course Organization	1.42	0.67
20	The adequacy of the outlined direction of the course	1.33	0.49
21	Your general enjoyment of the course	1.17	0.39

Overall, the instructors are also very satisfied with the course and plan to make significant improvements when it is taught again in the Spring of 2021.

6. ACKNOWLEDGEMENTS

Many of the details for this course were conceived during Shodor's 2018 Community Building for Parallel Computing Curriculum Development workshop. Shodor has taken an amazing lead in developing real world motivated examples in education (http://www.shodor.org/). I also want to specifically thank David Joiner for providing the example 1D wave equation code for the first benchmarking homework; this was a great first project and really helped start the discussion for compilers options and vectorization.

7. REFERENCES

[1] D. Colbry, M. Murillo, A. Alessio, and A. Christlieb, "Computational Mathematics, Science and Engineering (CMSE): Establishing an Academic Department Dedicated to Scientific Computation as a Discipline," *JOCSE*, vol. 11, no. 1, pp. 68–72, Jan. 2020, doi: 10.22369/issn.2153-4136/11/1/11.

[2] D. Silvia, B. O'Shea, and B. Danielak, "A Learner-Centered Approach to Teaching Computational Modeling, Data Analysis, and Programming," in *Computational Science – ICCS 2019*, Cham, 2019, pp. 374–388.

[3] G. Wilson, "Software carpentry: getting scientists to write better code by making them more productive," *Computing in Science & Engineering*, vol. 8, no. 6, pp. 66–69, 2006.

[4] L. Abeysekera and P. Dawson, "Motivation and cognitive load in the flipped classroom: definition, rationale and a call for research," *Higher Education Research & Development*, vol. 34, no. 1, pp. 1–14, Jan. 2015, doi: 10.1080/07294360.2014.934336.

[5] T. Kluyver *et al.*, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016, pp. 87–90.

[6] D. E. Knuth, "Literate Programming," *Comput J*, vol. 27, no. 2, pp. 97–111, Jan. 1984, doi: 10.1093/comjnl/27.2.97.

[7] "JupyterHub," *GitHub*. https://github.com/jupyterhub (accessed Jun. 09, 2020).

[8] D. Colbry, W. Punch, and W. Bauer, "The Institute for Cyber-Enabled Research: Regional Organization to Promote Computation in Science," San Diego, California, USA, Jul. 2013.

[9] J. Towns *et al.*, "XSEDE: Accelerating Scientific Discovery," *Computing in Science Engineering*, vol. 16, no. 5, pp. 62–74, Sep. 2014, doi: 10.1109/MCSE.2014.80.

[10] B. Lu, "Use bootable Linux CD (BCCD) to teach cluster and parallel computing concepts: conference workshop," *J. Comput. Sci. Coll.*, vol. 24, no. 5, p. 142, May 2009.