How to Build a Fast HPC n-Body Engine From Scratch

Eric Peterson[1]
Marmion Academy
1000 Butterfield Rd
Aurora, IL 60502
epeterson@marmion.org

Max Kelly[2]
Rose Hulman Institute of Technology
5500 Wabash Ave
Terre Haute, IN 47803
maxwellrkelly@gmail.com

Dr. Victor Pinks II[3]
Marmion Academy
1000 Butterfield Rd
Aurora, IL 60502
vpinks@marmion.org

ABSTRACT
Communicating and transferring computational science knowledge and literacy is a tremendously important concept for students at all levels of education to understand. Computational knowledge is especially important due to the tremendous impact that computer programming has had on all scientific and engineering disciplines. As technology evolves, so must our educational system in order for society to evolve as a whole. We undertook direct instruction of a computational science course, and have developed a curriculum that can be expanded upon to provide students entering technical disciplines with the background that they need to be successful. The course would provide insight to the C programming language as well as how computers function at a more basic level. Students would undertake projects that explores how to program simple tasks and operations and ultimately ends in a final project aimed at assessing the knowledge accumulated from the course.

CCS Concepts
• *Social and professional topics* • *Social and professional topics~Computing education* • Social and professional topics~K-12 education

Keywords
Computing education, K-12 Education

1. INTRODUCTION
Computational science as a discipline uses modern tools of computer science combined with a mathematical approach to problem solving to tackle issues that are relevant to science. Computational science requires knowledge from three disparate fields: computer and information sciences, which is used to develop the software and data structures to solve computationally interesting problems, numerical and non-numerical approaches to modeling, which can be used to represent data for scientific problems, and computing infrastructure that the software can be run on.

A problem of scientific interest is usually first understood in terms of a model that predicts or explains observation, which serves as a template for software engineers to develop a computer program that allows the model to be studied, and then is finally executed on a computing platform. Virtual modeling helps to see the expected outcome of experiments and ideas without real world execution. This helps save time and money. Today companies utilize computational science to do just that. Other uses of computational science and virtual modeling are to view objects and interactions that are difficult to view under normal circumstances. Particle interaction is the best example of this. The class that this paper will help outline was all about modeling a particle, something that cannot be seen by the naked eye, as it moved through space.

Computational Science is not widely taught. There are schools that will cover one of the fields of computational science in great detail, but will provide little or no detail on the other two. The goal of this course was to provide a strong foundation in all three areas.

DOI: https://doi.org/10.22369/issn.2153-4136/8/2/6

2. RELATED WORK
The approach that was developed at Marmion Academy was heavily based upon the instruction that the SHODOR educational foundation developed for use in the Blue Waters Student Internship and the Petascale Institute. The approach of the institute combined lectures on the theory of parallel programming and high-performance computing with practical exercises that reinforced the concepts. We sought to adapt this approach for use at Marmion Academy, and used the XSEDE training roadmap available from HPC university as a starting point to developing computational literacy in our students.

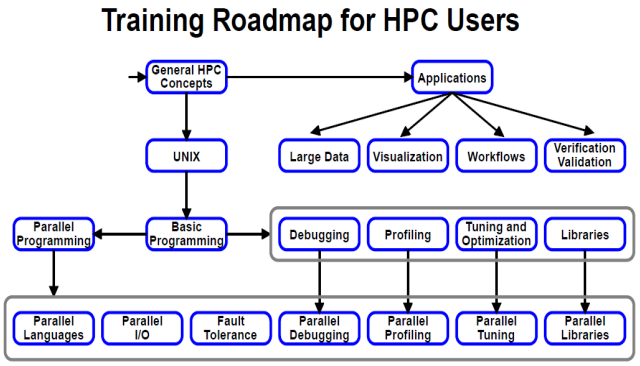

Figure 1. XSEDE Training Roadmap

3. CHALLENGES

There are some significant challenges that are faced when introducing a computational science curriculum into a high school environment. The primary challenge faced is the lack of computing backgrounds amongst the majority of the course enrollees. Most students taking the course do not have any familiarity with programming languages, none do they have any exposure to a Linux operating system, and few have taken the mathematic and scientific coursework necessary to understand the scientific models that will be covered. Also, the high school format with 45 minute class periods limits the amount of time that instructors have to cover new material. With crowded student course schedules, students have very little time outside of class to self-study or prepare.

The students' lack of knowledge and available time greatly hindered the results of the course. Several were able to learn the material, but almost all struggled in applying the material even at the end of the year. Students lost interest in the class as the year progressed. Additionally, majority of the students enrolled in the course were seniors and became afflicted with senioritis making it difficult for them to learn and pay attention. As stated previously, students had problems applying concepts that had been introduced throughout the entire year. The same questions were constantly being asked by students showing their lack of ability to grasp what the class was teaching.

Several students had difficulty with the language as well. The C programming language has a lot of rules and syntax to it. Students were very confused with how to structure their code and irked the students once they figured out how "dumb" a computer really is. Several students that did express interest in the class say that the difficulty of the language being used dissuaded them from pursuing a career in computational science in the future. The environment they were programming in, Cygwin, also confused the students because it used keystrokes that the students were unfamiliar with. It caused a lot of frustration for the students since they might accidentally delete a line of code due to the unfamiliar key strokes.

4. COURSE OUTLINE

Using the roadmaps available from HPC University, we developed a prototype curriculum for the course. The curriculum was developed with the express goal of introducing high school students with no computing background. We began with a conceptual introduction to the ideas of high performance computing, computer architectures, parallel programming, and data visualization. We would then proceed with a tutorial of the Linux command-line interface, along with the basic skillset needed to run and submit jobs on the Blue Waters system. The program used for this was Cygwin due to its ability to emulate a Linux environment and because of the instructor's familiarity with it. Next, we would introduce a computer programming language that the students would use when implementing the algorithms that model solutions to scientific problems. The language that we chose for this course was the C programming language, due to the large codebase of examples from the Petascale Institute and the language's high degree of hardware optimization. Throughout the course, topics relevant to software development would be introduced, such as best practices, debugging, libraries, and profiling. This was done through lectures and class examples as well as projects that the students would work on throughout the week. The projects would implement the topics discussed in class in order to help students understand their importance. The difficulty of the projects was extremely low due to the constant difficulty in understanding the C programming language and issues with the Cygwin environment. After the students were confident in their knowledge of basic programming, we would then proceed to parallel programming concepts and techniques using resources such as MPI. Due to time constraints and overall difficulty in the instruction the students were not able to receive instruction in parallel programming. The course would conclude with a project that would encompass all they had learned throughout the course. We decided that the final project should be an N-Body simulation of a particle's position on a Cartesian Plane in order for the students to demonstrate a proper amount of knowledge from the course. This project would involve computing the continual position of the particle using loops and dynamically updating the variables involved with the particle's position as well as printing to the screen. The final project would be graded based on how well constructed the student's code was able to output the results, the accuracy of the results, the student's understanding after a small Q&A, and finally if the code was well documented. All percentages for appropriate grading scales as well as a more documented step by step walkthrough of how the students would be taught are shown in the attached course syllabus after the acknowledgements and references.

5. COURSE INSTRUCTION

The course ran for 40 weeks, with one 45-min session every day. We utilized a project-based teaching method, through which the students were graded based on their ability to work through and complete in-class practical projects and materials. Each week consisted of a combination of the following: lectures on new concepts and new materials, practical lectures that the students could follow along with, or in-class projects and exercises. We also needed a development environment that would simulate the Blue Waters development environment, and for this role we used Cygwin. Cygwin allows us to simulate a Linux environment on a Windows system, and allows for the student to practice the basics of the Linux command-line, which is the only interface that they would be exposed to on Blue Waters. Resources from the Petascale Institute were used to familiarize the students with job processing on Blue Waters, include the "Time to Science" demonstration intended to demonstrate the performance benefits of increasing node size on a job. A basic workflow guide was also created by the course instructors for use as a simple in-class tutorial on basic operations on the Blue Waters system.

The first semester was extremely different from the second. The instructors decided that the students would be graded on completion rather than an assessment. This changed at the end of the first semester because the students were prioritizing other classes since they were finding the class extremely easy. It was found that students were not learning the material and constantly asking questions that they should have been able to answer. Second semester saw a lot of grading on projects that was not seen previously. While this was met with a lot of frustration from the

1 - Undergraduate Student

2 - Undergraduate Student

3 - Principal Investigator

students, they did begin to take the class more seriously and focused on learning and becoming more proficient with the C language.

6. COURSE RESULTS

The final project for the course was for the students to create an n-body simulator using the skills they had learned in the class throughout the year. The students were given three weeks to work on the simulator. The goal was to accept input for a time function and then track the position of a particle on a standard Cartesian plane. The students were required to print out a graphic inside of a terminal showing the coordinates of the particle in relation to the origin. The equations that the students used to model the particle were explained. Debugging and algorithm help were also provided as well.

As stated in Section 3, the students had difficulty throughout the course. Due to the ease of use with modern technology, students had difficulty grasping the basic implementations and syntax of the C programming language. Their lack of knowledge in programming greatly hindered the progress of the course. Questions were continuously asked throughout the year about topics that were extremely basic and demonstrated in every project. Examples include the scope of a variable, variable assignment, and syntax for both for and while loops.

The project proved very challenging for several of the students. The most issues became clear when the students had to figure out how to graph the particle's position. Students were familiar with printing out to the screen by the end of the course, but were unsure how to do it with continuous updates to an object as it changed position. Others had problems figuring out how to use the C programming language to accomplish what the project required and were much slower to finish.

Overall the project was a moderate success. The students had the most problems with learning the C language because of its foreign nature. Students constantly struggled with basic concepts such as creating variables or managing the scope of a program. The students constantly asked the same questions such as "How do I make this?" for extremely basic concepts such as object and variable creation or Boolean logic. After interviewing some of the students, it was determined that most of the issues encountered were because of the approach taken in the first semester where grading was not weighted as heavily. Instruction could not be slowed down without sacrificing time to teach other concepts that would be needed for use in the final project. Students were also extremely unfamiliar with programming syntax. Several students were able to overcome this obstacle, but others struggled up until the very end of the class.

These faults do not hinder the results of the class. Of the original twelve students that were enrolled in the course, only a single student dropped the course. The student who dropped did not drop for academic purposes, but for issues with scheduling. All the other students who completed the course finished with very good grades due to their ability to finish the projects assigned and demonstrate all knowledge required of them. All of the students were still able to complete the final project and all had extremely good results. Several of the students also became extremely adept at utilizing the C programming language for their own use. Additionally, several of the students expressed interest in going into an engineering or programming related field, which require computational science knowledge. Students expressed extreme satisfaction in being able to learn programming since it will almost definitely help them as they are transitioning to college.

7. CONCLUSIONS

Throughout the course there were several issues that had to be overcome and others that could not be accounted for. Were this course to be taught in the future, we have listed several issues that we encountered and solutions that we feel would be most effective.

- We feel that the course instruction has been informative on how to best approach future computational science instruction. To improve and extend our instruction into the future, we feel that more emphasis be placed upon the applications of computational models, and less emphasis be placed upon computer programming instruction.
- Computer programming instruction requires a significant amount of time to adequately prepare the student, and thus is not a proper use of limited class time when the focus is to prepare students for the applications of computational science. It is therefore more appropriate to require the student to be comfortable with a programming language prior to entering the course, or to learn how to program outside of class time.
- Another option is to use a language that is simpler to use and requires less class time to attain familiarity with. Our primary candidate for a simple language included Scientific Python.
- Based on frustrations programming in a command-line environment, we recommend the use of an integrated development environment (IDE) when developing applications on a local system, and the use of command-line tools only on remote systems.
- A more rigorous grading methodology should be used in order to properly motivate the students and encourage active student participation.

8. REFLECTIONS

I feel that this experience has been very beneficial to me in my academic and career development, and I will take away a great deal from this work. The Petascale Institute especially was a tremendous opportunity to expand and polish my knowledge using the resources that SHODOR had available. The chance to work with colleagues from other institutions and learn from experts in the field of computational science education was an enlightening experience. I look forward to the chance to continue using the skills I gained from Blue Waters in my future academic work.

9. ACKNOWLEDGMENTS

I would like to thank Marmion Academy for all their help and support, especially Dr. Victor Pinks II for his guidance and expertise. I would also like to thank the teaching staff from SHODOR, whose codebase and teaching materials heavily inspired this course. Acknowledgements would not be complete without Max Kelly, whose help authoring this publication was invaluable. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National

Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

10. REFERENCES

[1] *Blue Waters Petascale Institute,* Available at: *http://shodor.org/petascale/workshops/bw2015/*

[2] *Cygwin Project,* Available at: https://www.cygwin.com

[3] *Unix Tutorial*, Available at: *http://www.tutorialspoint.com/unix/*

[4] *XSEDE Roadmap Training*, Available at: http://hpcuniversity.org/RoadmapSite/index/

[5] Steve Qualline (1997) *Practical C Programming*, 3 edn., : O'Reilly Media.

[6] Dr. Victor Pinks (2015), *Fall 2015 Marmion Academy CT-STEM Syllabus*

[7] Eric Peterson (2015) *A Very Basic Blue Waters Workflow Guide*

[8] Aaron Weeden (2015) *Time to Science Instructions,* Available at:

http://shodor.org/~aweeden/TimeToScience.pdf

[9] *BCCD Tutorials,* Available at: http://bccd.net/wiki/index.php/Tutorials

Appendix 1. Fall 2015 CT-STEM Syllabus

1st Semester - Fall 2015

Computational Science & Engineering (CT-STEM) Syllabus & Guidelines

Section 1

Textbook: Curricular outline will coincide with materials from the Computational Science Institute (http://www.computationalscience.org/) , the CT-STEM program designs at https://osep.northwestern.edu/projects/ct-stem , and from resources on HPC University (http://hpcuniversity.org/)

This is an introductory level course on the application of computational thinking to the solution of general science problems guided by the inherent processes of the Scientific Method. This course draws on the three pillars of science – theory, computation and experiment. Computational elements will be performed using the programming language of C that will be taught at an introductory level. Physical experiments will be performed to enhance computer simulation experiments as needed. This course is an introduction of computational thinking principles in a problem-based learning environment. A major piece of the learning process will be focused on using high performance computing (HPC) as the platform for computational science & engineering modeling via the Blue Waters system (see: http://www.shodor.org/petascale/). Some prototyping of HPC code will be performed on the Marmion Academy Raspberry Pi HPC cluster. Grading is formative.

Note: *Pre-requisites: Pre-Calculus with Trigonometry (completed or currently enrolled)*

- **Topic by Order**
- What is Computational Science and why?
- Federal, State and Local efforts to promote computational science
- Basic pre-calculus and basic linear algebra math review
- The Scientific Method as built on the three pillars of science (theory, computer simulation, and experiment)
- Outlining the n-body simulation semester project (why and how)

- General HPC concepts
- Basic Linux/Unix overview and command-line interface tutorial
- Overview of computer programming and C language basics (more advanced elements are taught in context of the project)
- How to log into and use HPC resources like Blue Waters and the Raspberry Pi HPC cluster
- Test run sample C code

- Introduction to parallel programming and parallel programming languages like MPI, OpenMP, CUDA
- Debugging, profiling, and optimization of serial and parallel code

- Creating a single particle (1-body) 3D simulation (step-by-step)

  - Introducing the creation of computer representations of a single classical particle in a box
  - Moving the particle through numerical integration
  - Moving the particle in 1D then a single particle in 3D
  - Applying Periodic Boundaries to the box
  - Running the simulation with C and visualizing the output

- Creating a many particle (n-body) 3D simulation (step-by-step)
  - Moving from 1D to 3D simulations – considerations of force
  - "It's all about forces" lecture
  - Realism of computer simulations – "If the forces are realistic, the simulation will be realistic" lecture
  - How we use physical experiments to improve computer simulation realism
  - Running the 3D n-body simulation for :
    - Planetary bodies and astronomical size simulations
    - Monatomics (homogeneous and heterogeneous systems)
    - Hard spheres to simulate macroscopic objects in our 'big' world
    - Chemical simulations and experiments (how they compare)
    - Biological simulations and experiments (how they compare)

- Biological simulation of wolf and sheep populations
- Chemical statistics of motion as applied to the Boltzmann distribution of velocities

All programming projects and physical experiment labs are defended with an oral presentation and examination immediately after completion of the write-up. All lab reports will follow the MLA research report standards. Bad grammar, plagiarism and spelling will also be checked /scored

Grading

Grading is calculated against a formative project-based model. Emphasis is placed on quality and timely completion of assignments. Homework grades are reduced 10% for each day late past the assigned due date. Procrastination and/or poor time management can have the most devastating effect on your grade. Stay on task. It's about following the path and doing the work. You will be given multiple attempts (except on exams and quizzes) to make them acceptable. If you do the work to the level of quality that I ask, you will get a good grade.

All students will be held accountable for their behavior according to the Policies outlined in the Marmion Academy Student/Parent Handbook 2015 – 2016. Verbal bullying such as negative characterization of other students or negative characterization of other student's behavior will be considered on par with physical bullying and not tolerated.

Quizzes (online; in class; note quizzes; C quizzes)	15%
Quarter Exam (comprehensive)	15%
Homework & Projects (online homework; in class projects)	20%
Labs (computer simulation analyses and/or physical laboratory experiment)	20%
Semester Exam (comprehensive test + successful 3D n-body simulation project)	30%

If you need to contact us: E-mail vpinks@marmion.org or epeterson@marmion.org

Appendix 2. A Very Basic Blue Waters Workflow Guide

A Very Basic Blue Waters Workflow Guide

Author: Eric Peterson, Marmion Academy

Created: June 2015

Last Modified: May 2016

Notes: Commands are in bold and are preceded by a dollar sign **$**. [ ] tokens should be replaced with the appropriate information, minus the [ ]. Text that is to be inserted into files is in italics. Text in the following font will be actual output from the system: `example system output`

1. First, SSH into Blue Waters (steps are different depending on type of account, either training or regular account):

   $ssh [your username]@bwbay.ncsa.illinois.edu or
   $ssh [your username]@bw.ncsa.illinois.edu

   `Password: [enter your password here]`

2. Use basic commands to change directories, display directories/files, move and copy files and directories, edit files (use basic CLI tutorial by Mubeen)

3. Requesting resources on Blue Waters, either XE or XK nodes:

   a. There are two different ways to request resources:
      i. Interactively (allows you to submit jobs and see output immediately):

         $ **qsub –I [resource list]**

      ii. Batch:

         $qsub [batch script file].pbs

   b. Resource lists:
      i. This is where you request the number of nodes, type of nodes, number of processors, amount of time required, and more. One example for an interactive mode:

         **$qsub -I –l nodes=1:ppn=[32 / 16]:[xk / xe] **
         -l walltime=1:00:00

      ii. For batch files:

         #PBS -l [resource list]

   c. Batch files: (Allows you to request resources that will run without direct intervention and waste less of your computing time)
      i. Basic structure of a PBS file:

         #!/bin/bash
         cd $PBS_O_WORKDIR
         #PBS -l [resource list]

#PBS -o [pathname of output file]
#PBS -e [pathname of error file]

aprun ./[name of program]

4. Running your job:
   a. On an interactive session:

      $aprun –n [number of processes] ./[name of program]
   b. In a batch file:

      aprun –n [number of processes] ./[name of program]

5. After the job has been submitted, you will see the job ID of your job appear at the command line. Example of a job submission (through batch):

```
Job submitted to account: jt4
1817863.nid11293
```

6. Checking the status of your jobs will be critical (It also allows you to check what job ID you received):

   $qstat -u [your username] or
   $qstat -u `whoami` (if you don't know your username)

7. When your job has finished (Q in stat indicates a queued job, R indicates a running job, and C indicates a completed job):

   a. In batch:
      You will receive files (hopefully in the same directory as the program) with the name : [program name].pbs.[e / o][job ID], unless you chose different names for the output and error files.

   b. In Interactive mode:
      You will see your results through standard output (the screen) unless re-directed.

8. Compiling your programs:
   a. Compiling your programs may depend on if you are using CUDA, MPI, openMP, openACC, or none of those, but the basics are as follows:
      i. Once you have finished editing your .c file, use the CRAY C compiler to compile your program with the following flags:

         $cc -o [program name] [program name].c –l[libraries]

      ii. This will create a file named [program name].
   b. If you are using openACC, you may need to add:

      $Module load craype-accel-nvidia35
      –h pragma=acc (add this to compiler line before -o flag)

   c. When using openMP, remember to set the number of threads:
      export OMP_NUM_THREADS=[number of threads]

d. If you are using CUDA, you may need to compile your CUDA section separately, then link the CUDA to your main program

$nvcc -o [program name].o -c [program name].cu
$CC -o [program name] [program name].o [program name].c

9. Makefiles:
   a. Makefiles are a way of automating the compilation process, and ensuring that all dependencies and flags are set properly every time you compile.
   b. Basic structure of a makefile: (example):

all:
    make
[program name]: [program name].c
    CC –o [program name] [program name].c
clean:
    rm –rf [program name]

10. Performance Testing:
    a. This guide will talk briefly about the GNU Profiler (gprof). First swap the environment to the GNU programming environment:

$module swap PrgEnv-cray PrgEnv-gnu

    b. Compile a program with the –pg and –g options enabled: (this enables profiling)

$gcc –pg –g –o [program name] [program name].c

    c. Now we actually run our program to generate the profiling info:

$ ./[program name]

    d. This generates a file called "`gmon.out`". Now we use gprof to visualize the profiling data, additionally providing it another run of the program to give it more data:

$gprof ./[program name] gmon.out > profiling_data

    e. Then, we can see the final data by using less:

$less profiling_data

11. Editing a file using a text editor (vi):
    a. vi is a simple text editor that is available on blue waters and all Unix-based systems. It can invoked simple by typing:

$vi or
$vi [file name]

    b. vi has two basic modes:
        i. Command mode, keyboard presses will be executed as commands. The default mode that vi starts in. To enter command mode, press **ESC**.

ii. Insert mode, keyboard presses will be insert or edit text in the file. To enter insert mode, press '**i**' on the keyboard.

c. In command mode, you can use the direction keys to move through the file, or use:
   i. **Ctrl-f** to move up one screen through the file.
   ii. **Ctrl-b** to move down one screen through the file.
   iii. '**gg**' to move to the top of the file.
   iv.
d. In command mode, you can use keyboard presses to edit:
   i. '**dd**' will remove an entire line
   ii. '**yy**' will yank a line, and p will place the yanked line where your cursor is located.
   iii. '**u**' will undo the last change you made

e. In command mode, to close and save files:
   i. '**:w**' will write the file to disk but not quit
   ii. '**:wq**' will write the file to disk and quit
   iii. '**:q!**' will quit without saving (useful if you made unwanted edits)

f. In command mode, you can navigate to a line by pressing

   '**:[line to navigate to]**'

g. In command mode, searching for a pattern is relatively painless:

   '**/[pattern to search for]**'