A Project-Oriented Organization for an Introductory Computational Science Course

Josef M. Breutzmann, Ph.D.
Department of Mathematics, Computer Science and Physics
Wartburg College
100 Wartburg Blvd.
Waverly IA, 50677
josef.breutzmann@wartburg.edu

Abstract

Dealing with students of diverse majors, mathematical background and programming experience presents a challenge to interdisciplinary courses such as "Introduction to Computational Science." The author's solution to this problem was to base the course on a sequence of interesting and illustrative problems/projects. This approach allows the physical models, mathematical models, numerical techniques, parallelization techniques and visualization techniques appropriate to the problem to be presented, discussed, implemented, and digested in context. The project orientation provides a welcome focus for the background mathematics, numerical techniques and programming details.

The course was offered in the Fall 2005 semester. The project-oriented strategy was successful for the expected reasons and the course was able to excite students with a wide range of backgrounds. They were able to complete the projects and understand the modules without being overwhelmed by whatever portion of the mathematics or programming in which they were less prepared than their peers.

Background

Among the many factors that have lead the department to investigate the area of computational science as a possible emphasis for our program there are three we wish to mention in this paper. They are firstly, the nature of our department, secondly, the possibility of cooperation with our colleagues in the other sciences, and thirdly, the way in which computational science combines multiple aspects of our discipline and related disciplines.

Our department is a combined mathematics, computer science and physics department. Many of our faculty teach in more than one of these areas; many of our students major or minor in more than one of these areas. Being in the same department, our faculty often have combined interests in these areas. Faculty collaboration within the department and with others outside the department is becoming more common. A computational science minor or emphasis will enhance these strengths of the department and provide opportunities for students to add value to their science majors with computational methods or to their computing majors with scientific applications.

At our institution, the faculty is *grouped* using a three-division model: the social sciences, the humanities/fine arts and the natural sciences. To take best advantage of this political structure, we in the natural sciences can use interdisciplinary programs such biochemistry, bioinformatics, and computational science to strengthen bonds and communication lines across departmental boundaries and provide collaboration opportunities for faculty and student research.

Finally, and perhaps most importantly, computational science seems a very natural way to integrate multiple areas within the department. Computational science combines physical models, mathematical models, programming, high performance computing, and visualization. Each of these topics is of independent interest within the department. An emphasis in computational science would bring these topics into close cooperation and allow opportunities for collaboration and cross-fertilization.

For example, one of our on-going projects involves 3-D stereoscopic visualization (Zelle and Figura). This group of faculty and students is constantly on the lookout for applications to visualize. Computational science would provide them with data and applications in need of visualization. Reciprocally, their need for applications to visualize would spur the computational science group to consider attacking new problems.

Similar cross-fertilization has already occurred between an astronomy professor and the visualization group and between a neuroscience professor and the visualization group. We hope that computational science can enhance these collaborations and provide the opportunity for more.

The first course to be developed when creating a computational science program, whether a major, minor or an emphasis, is an introductory course that can be used as a platform to introduce the concepts needed in the subsequent courses, and provide techniques that can be useful in solving problems in advanced science courses. It also needs to attract students to the area who are already involved in the major programs. We believe that it must be accessible to science majors, mathematics majors, and computing majors alike. The diversity of preparation in these students presents the major challenge for such a course. If students are going to take the course early enough to be useful, the instructor can expect at most a semester or two of programming and a semester or two of mathematics. The thesis of this paper is that using a project-orientation for the course can alleviate many of the difficulties that this diversity presents.

The Problems Faced

The diversity of preparation among the students presents the most significant problem for the would-be instructor of an introductory computational science course. We want it to be taken early and we want it to be open to a wide range of science and computing majors. This being the case, and having to work in the context of a four-year liberal arts college, we cannot expect students to have already taken more than a few basic courses in programming and mathematics. Many computational science courses start with the assumption that students have already taken such courses as linear algebra and differential equations on the mathematics side, multiple courses including a numerical methods course on the programming side, and are well into a science major of some type. A small school with limited faculty resources cannot run courses with deep prerequisite structures because the student numbers will not support the frequency of offerings that would be necessary. Accordingly, one of our goals for this class was to minimize, within reason, the number and level of the prerequisites.

Local computer resources can also be a problem. In our case, we were limited to a lab consisting of nineteen Pentium-4 machines running Debian Linux. A later section of this paper discusses the software used to implement the parallel environment, and our visualizations. It suffices at this point to say we used MPI through pypar, a Python implementation for parallelization. We ran this on top of the lam network-parallelism system. For most of the visualizations we used either VPython or plain vanilla ppm files. On the numerical computation side we used the Numeric package for Python. All of these software products are open-source and are available free of charge from their respective web sites and other sources (see reference section).

Why a Project Orientation

When designing the course, we felt our main object was to make it accessible to a wide variety of students. In searching for a text we discovered that each one we looked at was focused for only narrow portion of our perspective audience. There were texts so heavy on the theory of differential equations that they would have disheartened any non-math

major. There were texts so heavy on the parallel programming details that they would have done likewise to any non-computing major. We found none that would make the science majors comfortable at the start. Furthermore, we intended to use Python as main language for the course. This is the language we use in our first year programming sequence and would be the most comfortable for the students. We did not want to force students to learn a whole new language. With this in mind we also did not want a text that provided its examples in a language new to the majority of the students. So, while we put many resources on reserve for student use, we did not require them to purchase a computational science text book.

Since finding a text that would lay out the course for us was doomed, we decided to use the sequence of projects that we wanted the students to implement as our organizing principle. In the end, this decision turned out to be the most critical one for the course. It allowed us to achieve our most important goal and had some additional benefits.

The project-oriented approach modularizes the course so that each project could be approached more or less independently of the others. This allowed students to focus on the projects more closely aligned with their interests and their strengths. Success on these near and dear projects lead to success on the others they found less interesting or more challenging. The goal of making the class accessible to a wide audience was, for the most part, attained by this means.

One of the additional benefits from the independent nature of the projects was that students could continue to work on previous projects after the class had moved on to a new one. This allowed slower students to continue working on the basic project and the better students to enhance earlier projects.

Another of the additional benefits came from the fact that any single project could be proposed at a variety of levels. For each project there was a basic version that followed more or less readily from the material given in class. With the basic version complete the students could then ‘enhance’ the project in any or all of the areas of better numerical techniques, larger problem instances, better user interfaces, better visualizations and more sophisticated parallelization techniques. On the one hand, the basic level gave all students the experience and satisfaction of producing a working solution. On another, the enhancements gave the students the opportunity to explore the topic further and add depth to their understanding. The grading scheme encouraged students to enhance at least a couple of the projects beyond the basic level.

The Projects Selected

We attempted to select projects that would exhibit variety in subject, numerical method, parallelization method and visualization method. The goal was to have students use a variety of tools and to offer problems from a number of different mathematical and scientific disciplines.

For the first project we examined Lotka-Volterra predator-prey model (Sharov; Mathews). This project from biology offered a relatively simple set of differential equations to solve and had solutions that were 2-dimensional. We could visualize the solutions with simple graphs showing either or both of the populations as a function of time or the plot the two against each other to demonstrate the cyclic nature of the solutions. Since this project did not lend itself to parallelization, it was a good one to start with while learning the basics of parallel programming as a separate thread in the class. After solving the given instance the students could enhance the project to investigate other initial conditions, other rate parameters or even three species models. The later would be easier to visual after we had done the 3-D modeling in VPython in the third project.

In parallel (pun intended) but independently of the Lotka-Volterra project, we introduced the basics of parallel programming using lam/mpi and pypar. This cluster-computing model would allow us to build parallel applications to speed-up later projects. The students did simple "hello world" type routing in each of four virtual network topologies: hypercube, grid, ring and tree. This allowed coverage of some basic parallel processing content and gave them an introductory experience using pypar.

The next project was to reproduce the Lorenz attractor in 3-D (Bourke; Lorenz). The main purpose of this project was to introduce 3-D graphics using VPython. The classic Lorenz attractor is the solution of a trivariate set of differential equations and as such was a natural for 3-D visualization. We used VPython to produce 3-D images that could be viewed in stereo using either red-blue glasses or polarizer glasses. The red-blue images could be viewed on any color screen. The polarized images could be viewed using one of our in-house dual projection systems (Zelle & Figura). Since this project did not benefit from parallelization and introduced a new visualization technique, it fit well here early in the course at the same time students were working out the previous network topology project.

The next project used actual data gathered by the forty-foot educational-use telescope at the National Radio Astronomy Observatory – Green Bank (Heatherly). We used radio signal strength data acquired as the telescope swept up and down along the azimuth while the sky rotated by. The Cygnus A radio source was the target. The students were given the signal strength at a collection of points in a saw-tooth pattern covering a small section of sky. The project was intentionally left very open ended. Instructions were along the lines of "Create a visualization that illustrates the features present in this data." Some possibilities were discussed and a few simple examples were shown to illustrate a few starting points. Student interest lead to a discussion of Delaunay triangulations and other useful tools for this type of project. After that discussion we discovered Delaunay triangulation was a built-in feature of VPython, something we were unaware of previously. The students came up with some very clever and unexpected techniques. With this project we introduced PPM (portable pixel map) files as an easy-to-generate, cross-platform image format.

By mid-term we were ready for an application that would benefit from parallel programming methods. We had done a few simple things with parallel programming and

moving on to a visualization of Newton's Basins of Attraction allowed us to introduce a simple master-slave model of parallel computation. In constructing an image of the basins, we created slave processes that could create one horizontal line of the image and a master process to apportion the tasks dynamically and reassemble the final image. The advanced version of the project allowed the user to draw a rectangle across a portion of the image and dynamically zoom-in on the selected region. With approximately sixteen slave processors, the response time was quite usable at just about 7 seconds to generate an 800x800 pixel image. The technique could be extended quite easily to a wide variety of Mandelbrot, Julia and fractal type images.

Our second major project involved solutions to boundary-value problems for the two-dimensional Laplace equation. While many applications are possible, we chose to think of our problem as one of calculating steady state temperatures. We envisioned a room with a fireplace, a few windows and insulated walls. We set the boundary values and solved the implied system of equations using a standard iterative method. The basic version of the project used a mesh (4 nearest neighbors) model of parallelization with each of $N = n^2$ processes computing 1/N th of the image and sharing boundary values after each computation pass. The final result was visualized in a variety of ways by the different student teams. The simplest technique was using ppm files. One of the teams extended the project to a three dimensional room, using the 3-D Laplace equation over a 3-D grid and visualized the result by showing slices of the room under mouse control.

Our third and final major project, a personal favorite of the instructor's, was the simulation of colliding (more accurately, the close approach of) galaxies. Using simple three-dimensional Newtonian physics, the students created galaxies of stars orbiting a galactic center. To reduce computational complexity, we placed a high mass black hole at the center of each galaxy and considered only interactions between stars and the black hole and not betwixt the stars themselves. When the students were able to create single galaxies that appeared to behave normally, they then created two or more and put them into relative motion and watched the interactions between these galaxies. Unfortunately, the best-known parallelization techniques for this problem are for shared-memory multiprocessing and are not useful for the message-passing model we were using. With little time left in the semester it was not possible to explore parallelization of this model. Nor was there time to consider pair-wise stellar interactions. We believe that we would have had the computational power necessary to consider the all the pair-wise interactions for simple instances of galaxies but time ran out in the semester. VPython provided the 3-D visualizations for this project. Students could again produce stereo images in either red-blue or polarized modes. In fact, with the stars in motion, the 3-D effect was quite striking. With even a few thousand stars the motion was fairly smooth and quick.

The Six-Stage Approach

Each project was developed using a six-stage approach. The stages of development included,

(1) understanding the physical model,
(2) developing the mathematical model,
(3) adapting appropriate numerical techniques to solve the mathematical model,
(4) considering possible visualization tools available,
(5) implementation and testing, and
(6) exploring possible parallelization techniques to allow larger problem instances to be solved (serial versions of small instances were usually developed first).

Since the projects were quite diverse, not every stage was equally developed for every project. The projects were based on well-known computationally intensive problems from a variety of mathematical and scientific areas. As such, the first three stages were not usually developed anew but adapted to the students' various backgrounds.

As seen above, the physical models were taken from astronomy, biology, meteorology, physics, chaos, and mathematics. The plan to include a chemistry application was dropped since the class make-up did not include any chemistry students, and time was at a premium.

Most of the mathematical models involved partial differential equations. None of the students had taken a differential equations course and in fact only a minority had taken multivariate calculus. The project-orientation allowed us to discuss the particular equations in context and very concretely. Because of this the students seemed able to understand the significance of the various component parts of the equations even though everything was new to them. They were subsequently able to discuss the meanings of the systems of equations in class and on tests. We believe that one of the principal strengths of the project-oriented approach is that students that would have been unable to cope with a course that started with a more abstract approach to differential equations were however quite comfortable dealing with them in context and could appreciate their ubiquity in scientific computation. For our projects involving systems of differential equations, the Lotka-Volterra simulation, the Lorenz attractor, and galaxies simulation, the Runge-Kutta quadrature method was sufficient for our purposes. For the Laplace boundary problem we used a Gauss-Seidel Iteration Method. For the enhanced versions of that project we discussed the possibility of using an over-relaxation technique.

Computational Tools Used

The introductory programming sequence at Wartburg is taught in Python ("Python"; Zelle). To minimize the learning curve for the students, and because we believe that Python is a powerful and easy to use platform, we looked for tools that would be available as Python libraries. Having had success using MPI, Message Passing Interface, in prior parallel processing classes, we compared a couple of Python interfaces for MPI,

pypar and pyMPI (Neilson; Miller). We settled on pypar. While not a full MPI implementation, it was easy to use and contained the portions of MPI that we expected to use in the course. It worked out well for the course but for more serious applications would have been inadequate. This is mainly because it lacks the ability to create processes dynamically.

This MPI implementation runs on top of lam, Local Area Multicomputer, a network framework originally developed at the Ohio Supercomputer Center ("LAM/MPI"). The most intricate part of the lab setup for the course was installing and configuring the lam component of the system. Once configured for the system and after each student was configured, it ran seamlessly throughout the semester requiring no additional maintenance.

The computer lab that was used as the resource for the course contained nineteen relatively standard Pentium-4 machines running Debian Linux. With lam and pypar students were able to start up multiple processes that were allocated evenly to all the available machines. Since the lab machines are connected through a high-speed switch, performance within the lab was good. The communication delays were all local and timing experiments showed that they behaved as expected.

The Numeric package for Python provided array data type and matrix operations for projects. Numeric was also built into the VPython package (see next section), so it was convenient to use it in parallelization component of the projects as well ("Numerical"; "VPython").

For the most part, our small cluster of standard machines was quite adequate to the task. Only the galaxy collision project would have used more horsepower if the enhanced versions were to be attempted. Some students attempted to create animations of a few of the simulations. They also found that more horsepower would have allowed them to generate more frames using a smaller time step and thus would have yielded better frame rates.

Visualization Tools Used

For the basic level projects, we limited ourselves to the resources available in VPython, ppm files, and an elementary graphics package that comes with the text for our introduction to programming course. We made use of many features found in VPython. The first was the module to create graphs of functions. We were able to plot data on an x-y coordinate system with all of the labeling, scaling and other features needed for our projects.

For most of the projects we used VPython's 3-D modeling features. VPython plots various mathematical objects in three-space with options for coloring, labeling etc. Students were able to display these objects statically or set them in motion using simple programming techniques. Its stereo modes also allowed us to view stereo images with

either red-blue or polarized glasses. The students found that the stereo images provided a greater understanding of the nature of the output.

We used .ppm files for saving flat images (Poskanzer). The transportability and ease of generation of these files was ideal for the course. It allowed results to be transferred from the lab to other platforms, to the students' own machines and to the instructor's machine. They could easily be written by one part of the program and read back in and displayed by another part. They were also compatible with the elementary graphics package from CS1 that was mentioned earlier.

A few student teams experimented with creating movies of the simulations. In particular, one team attempted to make a movie file using output from the galaxy collision simulation. Another created a time lapse of the Laplace iteration to show (as they thought) heat "flowing" from the fireplace and warming up the room.

Results

In the first offering in Fall 2005, nine students of very diverse backgrounds enrolled in the course. It included students with only one programming course and senior CS majors with many such courses. It included math majors and a student with no college calculus courses. They were majoring in computer science, computer information systems, biology, biochemistry, physics and engineering.

Despite the diversity, all were able to get involved with the material. All were able to complete the basic projects. Most were able complete enhanced version of at least two projects. One exceptional student completed enhanced versions of all the projects. We believe that if we had used a traditional textbook-based approach, most of the students would not have connected with the material so nicely and would not have achieved as much as they did.

In addition to the projects, students also wrote two short papers; one on a current scientific computing project they found on the internet or the news, and a second on the computing resources at one of the supercomputing centers working on scientific computation. These short papers along with a short presentation of each added a non-programming component to the course and exposed the students to the *real world* of scientific computing.

We were very pleased by student reaction to the course. One student on their anonymous course evaluation wrote, "I loved the approach. In all honesty this has been the most interesting/fun class I have taken at Warburg. I learned more than in any prior course!" Another stated, "I like the project oriented concept. (Even though it was) more work than I had planned … I really enjoyed it. My favorite class this semester." A third said, "I liked the approach. The idea of learning about something, how it works, then applying it is something I enjoy." And finally one recorded, "I liked the concept of a project oriented approach with background on each topic." There were no negative comments on the

evaluations.

Conclusion

The project-oriented approach ameliorated some of the problems of a diverse student audience. The mathematical and programming techniques and models are given context and largely demystified for the students. A deep understanding of the derivations and theoretical aspects of the mathematical techniques used for particular projects were not necessary for the student to understand how to model a problem, and to implement and interpret a solution. The students appreciated the immediacy of learning a technique and immediately applying it to an interesting problem.

References

Bouke, Paul. *The Lorenz Attractor in 3-D.* Swinburne University of Technology. 22 Mar. 2006. <http://astronomy.swin.edu.au/~pbourke/fractals/lorenz>

Heath, Michael. *Scientific Computing: An Introductory Survey.* 2nd ed. Boston: McGraw-Hill, 2002.

Heatherly, Sue Ann. *The Forty Foot Telescope.* National Radio Astronomy Observatory – Green Bank. 22 Mar. 2006. <http://www.gb.nrao.edu/epo/forty.shtml>

LAM/MPI Parallel Computing. Indiana University. 22 Mar. 2006. <http://www.lam-mpi.org>.

Lorenz, Edward. "Deterministic Nonperiodic Flow." *Journal of Atmospheric Sciences* 20 (1963): 130-141.

Lucquin, Brigitte, and Oliver Pironneau. *Introduction to Scientific Computing*. Chichester UK: John Wiley & Sons, 1998.

Mathews, John. *Internet Resources for the Lotka-Volterra Model.* CSU-Fullerton. 22 Mar. 2006. <http://math.fullerton.edu/mathews/n2003/lotkavoltera/Lotka-VolterraBib/Links/Lotka-VolterraBib_lnk_1.html>

Miller, Patrick. "pyMPI – An Introduction to Parallel Python Using MPI." Livermore National Laboratories. 11 Sep. 2002. 22 Mar. 2006 <http://www.llnl.gov/computing/develop/python/pyMPI.pdf>

Nielson, Ole. *Pypar*. 29 Dec. 2004. 22 Mar. 2006. <http://datamining.anu.edu.au/~ole/pypar>.

Numerical Python Homepage. Source Forge. 22 Apr, 2006. <http://numeric.scipy.org>.

Pachero, Peter S. *Parallel Programming with MPI.* San Francisco: Morgan Kaufmann, 1997.

Poskanzer, Jef. *PPM File Format.* Source Forge. 3 Oct. 2003. 22 Mar. 2006. <http://netpbm.sourceforge.net/doc/ppm.html>.

pyMPI: putting the py in MPI. Source Forge. 22 Mar. 2006. <http://pympi.sourceforge.net>.

Python Programming Language – Official Website. 22 Mar. 2006. <http://python.org>.

Sharov, Alexei. Lotka-Volterra Model. 12 Jan. 1996. 22 Mar. 2006. <http://www.gypsymoth.ento.vt.edu/~sharov/PopEcol/lec10/lotka.html>

VPython. 22 Mar. 2006. <http://VPython.org>

Wilkinson, Barry and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* 2nd ed. Upper Saddle River, NJ: Pearson-Prentice Hall, 2005.

Zelle, John. *Python Programming: An Introduction to Computer Science.* Wilsonville OR: Franklin Beedle A& Assoc., 2004.

Zelle, John and Charles Figura. "Simple, Low-Cost Stereographics: VR for Everyone." *SIGCSE '04 Proceedings. SIGCSE Bulletin* 36.1 (Mar. 2004): 348-352.