November 2013

Volume 4 Issue 1

JoCsE

Journal Of Computational Science Education

Promoting the Use of Computational Science Through Education

ISSN 2153-4136 (online)

Volume 4, Issue 1 **November 2013**

JOCSE

Journal Of Computational Science Education

The Journal Of Computational Science Education (JOCSE), ISSN 2153-4136, is published quarterly in online form, with more frequent releases if submission quantity warrants, and is a supported publication of the Shodor Education Foundation Incorporated. Materials accepted by JOCSE will be hosted on the JOCSE website, and will be catalogued by the Computational Science Education Reference Desk (CSERD) for inclusion in the National Science Digital Library (NSDL).

Subscription: JOCSE is a freely available online peer-reviewed publication which can be accessed at `http://jocse.org`.

Contents

Introduction to Volume 4 Issue 1

Steven I. Gordon
Editor
Ohio Supercomputer Center
Columbus, OH
sgordon@osc.edu

Forward

In this issue we provide a articles that present excellent examples of computational science educational materials and sources of materials for teaching computational science. Yasar provides an overview of a general education curriculum that uses computer modeling to both build analytical skills and teach basic concepts in science and mathematics. He documents the success of an introductory course and also summarizes the course sequence that move students from using existing models to programming skills that build new models.

Shiflet and Shiflet present an article on the use of probabilistic models to trace the age of populations over time. The models were applied in several classes focusing on the mathematical concepts and the scientific understanding of the students. Markov chain models were used to model the populations of a variety of species, tracking them by age and the probabilities of survival and birth.

Ringenberg and Magana provide a review of STEM focused computational materials that are web accessible. They provide an interesting analysis of the number and types of materials available along with a table of the resources and their classification of their contents.

Chen *et al* provide a review of a research program for undergraduates in mathematics that focuses on computational mathematics. They describe the developments within their department, the creation of a research workshop, and the impacts of that program on their students and their department.

Finally, two student papers summarize their computational projects and the impacts of those projects on their own learning. Tudor and Space describe experience as interns working on a massively parallel Monte Carlo code using GPUÕs. Crawford and Toth describe the use of the knapsack problem to introduce parallel computing and describe the lessons learned from the project from both the student and instructor view.

Computational Math, Science, and Technology (C-MST) Approach to General Education Courses

Osman Yaşar
The College at Brockport
State University of New York
Brockport, NY 14420
Tel: +1 (585) 395-2595
oyasar@brockport.edu

ABSTRACT
In this paper, we present a computational approach to teaching general education courses that expose students to science and computing principles in engaging contexts, including modeling and simulation, games, and history. The courses use scalable curriculum modules organized in layers of increasing difficulties in order to balance learning challenges and student abilities. We describe the computational pedagogy followed in these modules and courses, with particular attention to the simulation-based course, namely introduction to computational science, to present a case study for those considering similar initiatives.

General Terms
General Education, Pedagogy, Games, History, Natural Sciences

Keywords
Modeling and Simulation, Abstraction, Computational Thinking

1. INTRODUCTION
Recent increases in power, access and affordability of digital technology have impacted scientific research, industrial design, and education. Educators stated at the turn of the century that, when used in the context of applications, technology would support higher-order thinking by engaging students in authentic, complex tasks within collaborative learning contexts [26]. More recently, the National Science Teachers Association (NSTA) described computation as a "third pillar" of scientific inquiry, accompanying experiment and theory [19]. It cited a growing body of evidence that using models and simulations, students learn better since they are actively engaged in "doing," rather than passively engaged in "receiving" knowledge. Within the scientific computing community, the role of computation had long been recognized and brought to the classroom through training by many organizations such as Shodor Foundation, and through formal degrees and courses by other institutions [15, 23], including ours [28-33]. However, now that we have help from educators and pedagogy experts to promote computational science education in more fundamental ways, we get a second chance to address some of the challenges we have faced.

While recruitment challenges can be linked to a general lack of interest and preparation by nation's high school students [2, 5-6, 9, 11, 16-17], computational science education requires additional preparation in multiple domains (math, programming, and sciences) that not every college student is willing to undertake. A fundamental (pedagogical) approach, including a focus on computational thinking skills [27], could bring all stakeholders together in a way not only to reform the computing education but also push scientific thinking into mainstream to address underlying causes of the rising category-5 storm in nation's K-12 education [16-17].

The importance of math and computational skills for STEM workforce has been noted in many reports; including the projections by the Bureau of Labor Statistics [3], National Science Board statistics [21], and surveys by the American Institute of Physics [1]. AIP surveys taken at regular intervals (in 1999 and 2010) of physics majors, 5+ years after finishing an undergraduate degree, indicate that some of the important job skills continue to be scientific problem solving, teamwork, computer programming, design and development, simulation and modeling, math skills, and technical writing (See Fig. 1).

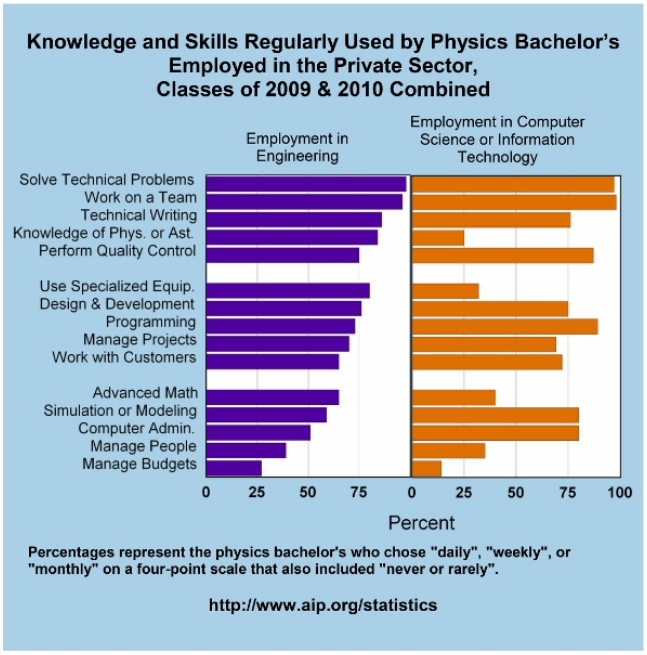

Figure 1: Results of the most recent AIP survey in 2010 [1].

While the demand for computationally competent science, technology, engineering, and mathematics (STEM) workers is an unprecedented opportunity, enrollments have gone down steadily in recent years. The pipeline between institutions of higher education (IHE) and K-12 seems to be broken [2, 5-6, 9, 11, 16-17]. The issue of why science is not as engaging as other subjects is complex, but according to the Relevance of Science Education (ROSE) study, student attitudes towards STEM become increasingly negative as a country advances economically, which suggests this phenomenon to be deeply cultural [22]. Learning science is demanding and it requires application, discipline and delayed gratification; values that contemporary culture does not seem to encourage. So, innovative and engaging ways of teaching science and computing are necessary.

2. COMPUTATIONAL SCIENCE EDUCATION AT BROCKPORT

Established in 1998, the computational science degree (BS and MS) program at Brockport attracted high parameter students and promoted research experience in an undergraduate institution [28-33]. Success stories from alumni hired by the software industry include multiple offers from the same company, offers for significant others as incentives, promotion to senior positions upon hiring, and many more. Those hired as teachers could teach multiple subjects (math, programming, and general science) due to their diverse background. These examples all point out to the benefits of a broad education to improve one's marketability and job orientation at a time when the likelihood of working at a job not related to one's field of study is greater than 50% [21].

While the computational science (CPS) program has attracted to Brockport students who normally would go to a higher tier school, it has graduated only a handful (~5) of students annually. Concerns over the number of freshmen entering the program led to an outreach effort by the program in 2003 to address the IHE-K12 pipeline issue as described earlier. An institute was formed in partnership with local school districts (Rochester City SD and Brighton Central SD) and national organizations (Shodor Foundation, Krell Institute, and Texas Instruments) to train secondary school teachers on the computational approach to math, science, and technology (C-MST). Improved teacher retention and student achievement reported by partnering districts drew national attention to this initiative, including testimony before the U.S. Congress. Over the past decade, institute staff and participants (faculty and teachers) created a large inventory of curriculum modules and lesson plans that are currently being used in the introductory-level general science courses described here.

While the computational approach to STEM education has been recognized a novel strategy to improve the technical workforce, curricular and recruitment challenges have slowed its growth. Brockport has revised its degree programs and courses several times to update and diversify its curriculum in several fronts, including: a) science of computing (simulation tools, programming, parallel computing, numerical and statistical methods, visualization, technical writing, and computing principles), b) science done computationally in application domains, and c) education done computationally (pedagogy, teacher training and K-12 student outreach). While some of the application courses such as *computational-x* (x: biology, physics, etc.) and teacher education courses cover deep (x) content to support STEM majors, others include service courses, such as those described here, under general education category to spread the benefits to all college freshmen across the spectrum. What follows in the next section is the computational pedagogy we used in these courses, particularly in CPS 101, to draw both STEM and non-STEM majors into learning about computing and sciences. We believe that this pedagogical approach is also relevant to a recent initiative by the College Board to implement a new AP course on computational thinking [25]. The impact on the nation's STEM education can be significant.

3. C-MST PEDAGOGY

While 'attention to details' is important to master a skill, we all have a limited memory to store information. The most pervasive strategy to improve memory performance (and information retrieval for problem solving) is organizing disparate pieces of information into meaningful units [14]. *Abstraction* skills can help with that by simplifying, categorizing, and registering key information and knowledge for quicker retrieval and processing. The act of abstraction is an *inductive* process by which we sort out details and connect the dots to arrive at more general patterns and conclusions [24]. While abstraction is essential for cognition, there are other benefits all around us. Since the nature itself employs abstraction by hiding the atomic-level motion and the cellular phenomena, we get the benefit of seeing the bigger picture. Computer scientists use abstraction to write large-scale complex codes (such as operating systems, compilers, and networking) where the complexity is distributed into seemingly independent layers and protocols of the code in such a way to hide the details of how each layer does the requested service. We all use abstraction in our daily lives. For example, when we go to a restaurant, we order our meal and not worry about how they cooked it in the kitchen. Those who worry and check it out once or twice cannot possibly afford doing it all the time. Abstraction skills can be improved beyond what was inherited, through training, education, additional knowledge and experience.

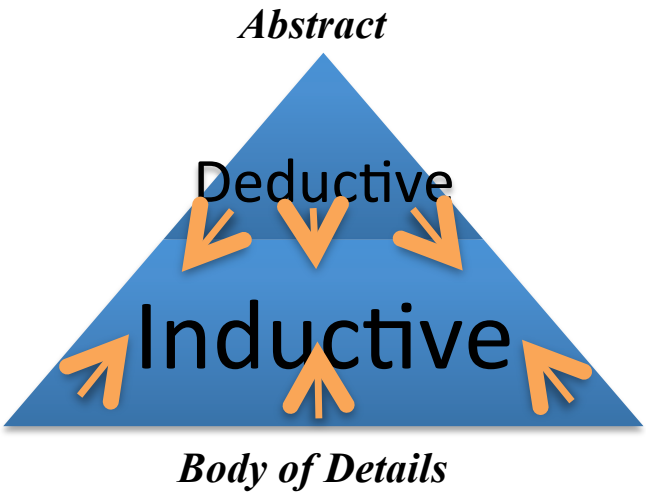

Figure 2: Illustration of the informational organization and the resulting deductive/inductive instructional pedagogies.

Computational modeling uses abstraction by its simplification of the reality. Such simplification helps scientists eliminate certain parameters and focus on what is being studied. Another benefit of computational ***modeling*** is that it supports *deductive* learning. Modeling enables the learner to grasp important facts surrounding a topic before revealing the underlying details. In a sense it helps to do a reverse engineering by gradually leading the learner to the details that support the abstracted knowledge (See Fig 2 for a schematic view of inductive and deductive processes). S***imulation***

adds another level of benefit by providing a dynamic medium for the learner to conduct scientific experiments in a friendly, playful, predictive, eventful, and interactive way to test hypothetical scenarios without having to initially know the underlying science concepts. Together, both computational modeling and simulation lead to a *deductive pedagogy* by first introducing a topic from a simplistic framework and then moving deeper into details after learners gain a level of interest to help them endure the hardships and frustration of deeper learning. Such a stepwise progression in learning is consistent with the pedagogical framework *Flow* [8] and scaffolding strategy to balance skills with challenges as illustrated in Fig. 3.

Figure 3: Illustration of Optimal Flow in learning [8].

According to a national report [18], at early stages computational modeling approach to STEM education should involve *easy experimentation* (learners must be able to quickly set up and run a model using an intuitive user interface, with no knowledge of programming or system commands) and *high interactivity* (models need to evolve quickly and include smooth visualizations for providing interactions and feedback to users). Using existing computational models, instructors can start general science education via games and simulations without exposing them to STEM principles right away. Students can get to modify an existing model, or create one from scratch. Tools such as Interactive Physics (IP) and AgentSheets (AS) can be used to create many fun things that could engage students into science experimentation. They also provide easily discoverable links (i.e., buttons for controlling the run-time and accuracy) to underlying ***principles*** of computational and scientific modeling.

After initial experimentation with modeling in the context of a game or science topic, students can be introduced to a simple principle of mathematical modeling (*new = old + change*) which eventually (and quickly) leads the learner into understanding several aspects of computational thinking [27], including decomposing a problem into smaller chunks, computational cost for more accuracy, and the need to use a programming language in order to handle complexity and increasing number of data points (due to decomposing the problem into much smaller chunks). The mathematical foundation of modeling and simulation can be taught in terms of building functional relationships (such as $y=f(x)$) using the *rate of change* equations and the above algebraic equation ($y_{new} = y_{old} + dy$) where y_{new} and y_{old} are *new* and *old* values of *y,* and *dy* is the *change* from the *old* to the *new*.

As an example, consider finding a direct relationship, $y=f(x)$, based on the rate of change (derivative) $dy/dx= 2x$, and the initial condition $y=0$ when $x=0$. The analytic answer to this mathematical integration is $y = x^2$. However, students can be led to find an answer through *numerical integration* instead, by constructing a table of *(x, y)* data points starting from *(0, 0)* for different choices of increment in *x* values (*dx= 1, 0.5, 0.1,* and so on). When the numerical results are compared to the analytic solution ($y= x^2$) for these cases, students could be led to discover the correlation between the step size (*dx*) and the accuracy of the numerical results: such as *the smaller the dx, the more accurate the answer*. While a human can calculate a few data points by hand when *dx* is 1, or 0.5, the need for automation (and accuracy) becomes obvious for smaller *dx* values such as 0.1 or 0.05. Excel can be used to automate the calculation and graph the $y=f(x)$ curves, but for much smaller step sizes (*dx*), such as 0.001, 0.0001, or 0.0000001, students might discover that even Excel cannot be of help in those computationally intensive cases. The need for finer and faster automation, via computer programming (example shown in later sections), becomes evident as the only way to obtain highly accurate results.

In an after-school project, several 9th graders from Brighton High School (NY) were able to replicate IP results for the harmonic motion (Fig. 4) by first applying Excel (Table 1) and then Python (programming language) to algebraic formulas for the position ($x_{new} = x_{old} + dx$) and velocity ($v_{new}= v_{old} + dv$) of the spring-driven object at times ($t_{new} = t_{old} + dt$) separated by interval *dt*. Here, time (*t*) is an independent variable and *change* in *x* and *v* are: $dx= v \cdot dt$ and $dv= a \cdot dt$, where acceleration *(a)* is *Force/mass*. The force applied by a spring unto an attached box is $F= - k \cdot x$, where *k* is the stiffness coefficient of the spring and *x* is the displacement of the box from the equilibrium position (x=0). The following year, these students modeled orbital motion of planets when the formula for the force, causing the *change*, was given to them ($F=G \cdot M \cdot m/x^2$; where *G* is a Universal Constant, *M* and *m* are masses of the Sun and planet separated by distance *x*).

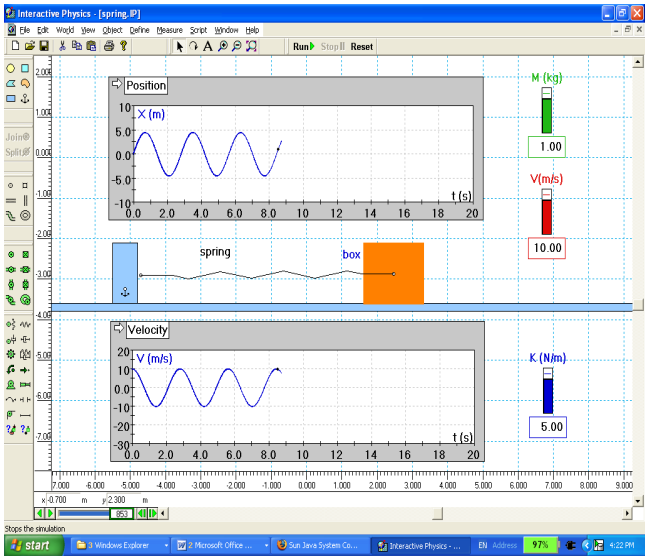

Figure 4: Simple harmonic motion using Interactive Physics.

Table 1: Simple harmonic motion using Excel (dt= 0.125).

t(s)	v(m/s)	x(m)	t(s)	v(m/s)	x(m)
0.000	10.00	0.0	1.250	-8.97	1.49
0.125	10.00	1.25	1.375	-9.90	0.26
0.250	9.22	2.40	1.500	-10.06	-1.00
0.375	7.72	3.37	1.625	-9.44	-2.18
0.500	5.61	4.07	1.750	-8.08	-3.19
0.625	3.07	4.45	1.875	-6.08	-3.95
0.750	0.29	4.49	2.000	-3.61	-4.40
0.875	-2.52	4.17	2.125	-0.86	-4.51
1.000	-5.13	3.53	2.250	1.96	-4.26
1.125	-7.33	2.62	2.375	4.62	-3.69

While these high school students were exceptions, this (deductive + inductive) pedagogical approach [13] does show a ***path*** that can be promoted in both IHE and K-12 classrooms. It starts with a deductive approach by using modeling to introduce the learner to important facts surrounding a topic. Then, by running hypothetical scenarios and investigative projects through simulations, they are encouraged to discover relevant principles of computing and sciences in an inductive fashion. We call the above approach 'C-MST pedagogy' because of the local context; others may call it differently (i.e., Project-First, then Principles just-in-time [11, 20]). Pros and cons of deductive and inductive learning have been a topic for many years in language training. Many contemporary programs use a combination to get double benefit [13].

The computational approach to STEM education encourages inquiry, involves projects, and facilitates team-based instruction. It puts the learner at the center of a constructivist experience. While it uses a deductive approach to engage learners into a STEM topic, it emphasizes the importance of abstraction skills to support inductive learning. By linking computing to science through the computation of ***change***, it provides a motivation for science majors to learn programming and for computing majors to learn more about science. Its motivational and deductive/inductive cycle can be used to broaden participation in computing and sciences among female students [11, 20].

Table 2: Gen-Ed Computing courses and their enrollments

	2010	2011	2012	Total
101 Intro to Computational Sci.	12	20	38	70
105 Games in Sciences	18	52	55	125
302 History of Sci. & Tech.	6	22	25	53

4. GEN-ED COMPUTING COURSES

We have developed and taught three computing-based general science courses, including CPS 101 Introduction to Computational Science, CPS 105 Games in Sciences, and CPS 302 History of Science and Technology. While the primary topic of this article is CPS 101, we want to give a brief overview of all these 3 courses. Launched in 1998, CPS 101 was taught by the author in full capacity (25 students per semester) until 2007 when a new faculty was assigned to teach it. The new instructor's tendency to teach it merely as a programming course with high level mathematics brought the course down to extinction. Through support from an NSF Course Development grant in 2010, the content of CPS 101 was shifted back from 'differential equations and computer programming' to its original content of problem solving at a more fundamental level as described later. This modification brought the course back to life again (see Table 2 for enrollments), vindicating the fundamental computational thinking approach to introductory computing [27] and computational science education [29]. Currently, CPS 101 uses simulation tools such as IP to teach students basic science concepts without having to require a deep level of mathematics and knowledge of the natural laws. The CPS 105 uses AgentSheets (AS; an agent-based modeling tool) to demonstrate science applications in the context of games and environmental issues. The CPS 302 uses an introduction to science and computing in the context of history, again supported by demonstrations using tools such as IP and AS.

Table 3: External/Internal factors affecting enrollments.

How did you hear about this course?								
CPS ↓	Friend		Advisor		Department		Other	
	2011	2012	2011	2012	2011	2012	2011	2012
101	7%	10%	40%	25%	7%	0%	46%	65%
105	34%	34%	18%	4%	2%	0%	46%	60%
302	17%	12%	44%	20%	16%	0%	23%	66%

To meet the needs of students with various backgrounds, a contextual learning is critical for students' success and it improves interest in technology while generating enthusiasm towards sciences [12]. Surveys reveal interesting observations on *student attitude vs. the context* (in which science topics were taught in these courses). They all had a broad appeal; drawing students from 28 departments, including non-science majors. In fall 2010, the College approved these as General Education courses in natural sciences. To meet the Gen-Ed designation in natural sciences, more general science content had to be added to the original syllabus. These modifications seem to have triggered an increase in the enrollment, as shown in Table 2. Interest by non-STEM majors increased. More than 60% of currently enrolled students in these courses are non-STEM majors, while this ratio was about 35% before these modifications when the main focus was on computing and mathematics.

According to the course surveys, the games-based course (CPS 105) was the most popular; 34% of enrollments in this course came from the word of mouth among friends. The evolving self-interest also seems to be high as about 80% of students in this course wanted to take another course on the same topic (See section on Results and Discussion). The simulation-based course (101) and the history-based course (302) were often recommended by student advisors. The Gen-Ed status contributed significantly to enrollments as can be seen from the 'Other' category in Table 3. As we moved from 2011 to 2012, even more students enrolled under 'Other' category as a consequence of either easiness of online scheduling or popularity of Gen-Ed designation; or both. The 2012 data shows a significant increase in students' self interest and a decrease in advisement by faculty and department.

4.1 Course Description: CPS 101

The weekly schedule for CPS 101 is shown in Table 4. Course materials include class notes and user manuals for software tools (such as IP) and the online C-MST curriculum modules open to public at *www.brockport.edu/cmst*. IP is used to model, simulate, and explore a wide range of physical phenomena, including harmonic motion (springs and pendulum), falling objects, trajectory of projectile, energy conservation, orbital motion, Kepler's Laws, Newton's second law of motion, and electrostatic oscillator. Through IP, students are able to build projects, conduct digital experiments, and investigate physical events without deeply knowing or memorizing the laws of physics. Users are allowed to set up their own physical world, choose physical

parameters, monitor the position, velocity, energy, and elapsed time and also create control buttons to facilitate simulation. Visuals images and data from IP can be transferred to geometrical software (such as Geometer's Sketchpad, GSP) to measure angles, distances, and areas needed for proofs or other calculations. IP simulation data can also be transferred in numerical format to Excel for further analysis.

A big advantage of the IP is what you see is what you get. It is interactive and it greatly enhances instruction and helps students build their confidence and success in learning. The use of IP tool is straightforward, and students are able to build their own projects after a couple of weeks training. A screen shot of simulating orbital motion is shown in Fig. 5, where the planets are represented with small circles and corresponding masses. For example, the earth (of mass 5.9 x 10^{24} kg and orbital velocity of 6.65x10^4 mph) is placed at 150x10^6 km from the Sun (mass of 1.89 x 10^{30} kg). The IP images can be transferred to GSP to measure the distances and areas to prove Kepler's laws. For example, Fig. 5 shows the proof of the 3rd law, which states that for each planet the square of its period (T^2) is proportional to its semi-major R^3; or $(T_1/T_2)^2 = (R_1/R_2)^3$ for any two planets.

Table 4. Weekly schedule for CPS 101

Week 1: Conduct surveys to examine math and CS skills. Discuss survey results. Show videos on the role of computational modeling & simulations in science & industry.
Week 2: Discuss the role of modeling in scientific inquiry and industrial design (show examples). HW #1: Short essay on the 'role of modeling' in research, industry, and education. Computer Lab: Introduce Interactive Physics with examples
Week 3: Computer Lab: Continue Interactive Physics (IP) training. HW #2: Design an IP project that demonstrates students' comfort level with IP.
Week 4: IP Labs: a) Simple harmonic motion: Generate position and velocity plots of a moving object attached to a spring; b) Pendulum: Examine the time it takes for a complete swing vs. initial velocity. c) Falling objects: Examine motion under different gravitational forces on the Earth and the Moon. HW #3: Report effects of elasticity, friction, and air resistance on motion in one of these labs.
Week 5: Discuss principle of mathematical and computational modeling: *new=old+change*. Discuss functional (i.e., y= x^2) and behavioral (i.e., dy= 2x · dx) relationships in tabular, formulated, and graphical forms. Discuss the Rate of Change (ROC) and the difference between Average & Instantaneous ROC. Test #1 (functions, ROC, and forms of representation).
Week 6: Continue numerical integration with examples by hand first and later with Excel. Discuss the role of integration step in reducing error and the role of computational power to afford smaller steps. HW #4: numerical integration by hand.
Week 7: Discuss the role of hardware (storage, processing, and communication) and software (data locality, memory usage, system software, and programming style) on performance and accuracy needed for problem solving. Introduce programming concepts using Python language. Conduct a midterm exam.
Week 8: Break
Week 9: Continue programming discussion with examples in Python. Hands-on experience on programming in a computer lab. Test #2 (factors affecting computational performance). HW #5 (on computer programming).
Week 10: Review use of multiple tools (IP, Excel, and Python) for modeling. Lab: Redo harmonic motion using *new = old + change* computations via Excel and Python. Learn about Newton's law of motion ($F=m\cdot dv/dt$) as a cause of *change* in velocity & position. Compare IP, Excel & Python results.
Week 11: Lab: Trajectory of Projectile: Use IP & Excel to study 2-dimensional motion. HW #6: Write a Python program that computes the trajectory of a rock thrown up at an angle.
Week 12: Lab: Conservation of Energy & Momentum. Discuss potential & kinetic energy of earlier examples. Use IP to graph potential and kinetic energies of objects. Examine effects of friction and air resistance.
Week 13: Lab: Orbital motion: Watch videos on orbital motion and space explorations. Learn about gravitational force $F=G\cdot M\cdot m/r^2$ as a cause of *change* in position of planets. Simulate orbital motion in 2-D using Excel and then Python.
Week 14: Discuss Kepler's laws and use IP to simulate multiple planets around the Sun. Lab: HW #7 (team project: proof of Kepler's Laws). Introduce agent-based modeling using AgentSheets (AS).
Week 15: AS Lab: Model collective behavior of agents. HW#8: Design an AS project. Re-visit HW #1 to improve the essay on 'role of computation'. Review for Final Exam: Discuss and review scientific concepts learned.

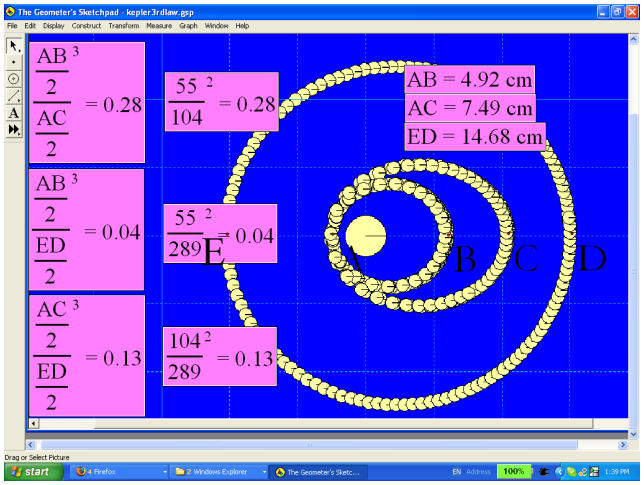

Figure 5: Orbital motion of several planets around the Sun. Orbits and periods are shown to prove Kepler's 3rd Law.

4.1.1 Programming with Excel

While IP is a good tool to expose students to many physical concepts, computational STEM education needs to move beyond just using tools. Our previous experience indicates that students need to eventually understand the underlying mechanism of simulation and modeling and to flexibly master and apply acquired knowledge rather than practice rote memorization of scientific laws. In CPS 101, students are required to model a physics phenomenon by computer simulation using IP, and then solve the same problem via Excel and later by writing a computer code using a language such as Python.

To use Excel for generating position and velocity values of an object that is subject to an external force, students need to designate three columns in an Excel worksheet to these variables as shown in Table 1. The columns were cut into half and put side by side for the purpose of fitting the data into a frame for this

article. The first row in each column holds variable names and the 2nd holds initial values *(t=0 sec, v = 10 meters/s* and *x= 0 meters)* and constants *(m= 1 kg* and *k= 5 Newton/meters)*. The 3rd row holds expressions computed in the following order: $t + dt \rightarrow v + (-k{\cdot}x/m){\cdot}dt \rightarrow x + v{\cdot}dt$ where *t, v,* and *x* are linked to their own values on the previous row; except that the value for "*v"* in the $x + v{\cdot}dt$ expression is linked to the newly computed value of *v* (on the same row) in order to move forward updated information. The expressions in the 3rd row can be copied and pasted to the rest of the rows below until *t* reaches maximum time *(T)* desired. This is where the limitations of Excel come into play. The visible and scrollable screen might not accommodate the whole simulation range when the integration time step *(dt)* is very small. If one chooses *dt* to be 0.000001 sec, then one needs 1,000,000 rows to do an Excel computation for just only 1 second.

In two-dimensions, the above equations need to be expanded to include position, velocity, and acceleration in additional dimensions:

$$x_{new} = x_{old} + v_x{\cdot}dt; \quad v_{xnew} = v_{xold} + a_x{\cdot}dt; \text{ where } a_x = (x/r) \cdot a,$$

$$y_{new} = y_{old} + v_y{\cdot}dt; \quad v_{ynew} = v_{yold} + a_y{\cdot}dt; \text{ where } a_y = (y/r) \cdot a,$$

and

$$r^2 = x^2 + y^2 \quad \text{and } v^2 = (v_x)^2 + (v_y)^2$$

Using these algebraic equations along with interplanetary acceleration between Earth and the Sun $(a = F/m = G{\cdot}M/r^2 = 1.26x10^{14}\ N{\cdot}km^2/kg \cdot 1/r^2;$ where *G* is a Universal Constant and *M* and *m* are masses of Sun and Earth), we get the orbital track seen in Fig. 6. At t=0, we assumed that the Earth's orbital velocity was given by v_x=0 and v_y= 29.79 km/s and its position was given by *y*=0 and *x*= 1.50x10^8 km. What is shown in Fig.6 may not be the most accurate track, but qualitatively it is representative of a planet's orbit. Some planets have more elliptically looking orbits. The above calculations are given for *dt* = 5 days, however smaller time steps (*i.e., dt=1* day) could produce more accurate tracks. Again, that is where the limitations of Excel come into play, just like the million data points mentioned above. With computer programming, these limitations can be overcome. Computations with higher resolution and automation need use of programming.

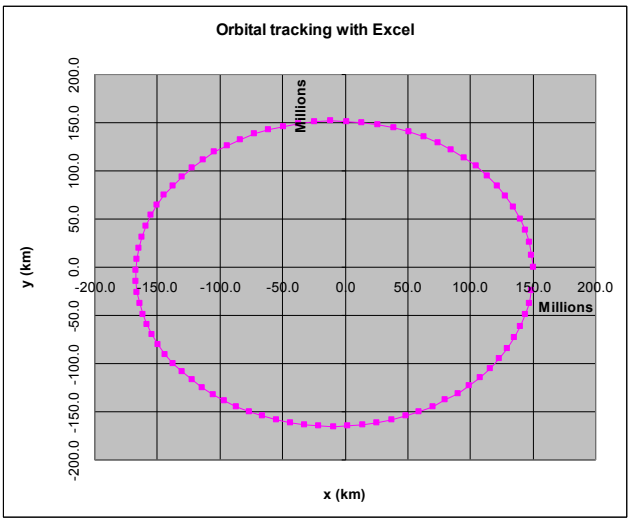

Figure 6: Orbital tracking of the Earth using Excel.

4.1.2 Programming with Python

In the past we used Fortran and C but we have recently switched to Python. The switch to Python was based on three major reasons, including relative easiness and quickness with learning of Python as a computer language, its simple and short constructs, and less error-prone coding. Python is a general-purpose, object-oriented, high-level programming language, which comes with extensive standard libraries and supports the integration with other languages and tools. It is increasingly used in scientific computing, web development, and database operations. Python can be learned in a couple of weeks for basic operations; it is open-source and platform-independent, and it can be installed on almost any computers free of charge. An introduction to basic syntax, input/output functions, repetition structures (loops), and algorithmic thinking is adequate to carry out programming assignments necessary for computing a mathematical or logical expression repetitively, recursively, or iteratively. Students can write simple loops to compute and generate data points for a number of problems listed in the course syllabus including falling objects, trajectory of projectile, harmonic motion, and orbital motion. Below is a sample algorithm for one-dimensional Newtonian motion that can be easily extended to two- and three-dimensions.

Input initial position (x), velocity (v), and time (t)
Input time step (dt), maximum time (T), mass (m)
While t <=T:
    Output position (x), velocity (v), and time (t)
    Compute force F & acceleration a = F/m
    Compute velocity in x direction v = v + a × dt
    Compute position in x direction x = x + v × dt
    Update the time t = t + dt
End of While Loop

5. RESULTS AND DISCUSSION

We have used a mix-methods approach [7] to examine the context, pedagogy, and the tool set used in computational gen-education courses. Table 5 and 6 show some of the statistics from student surveys. The tables show a multi-year data for each course. To examine whether there is any statistical difference between 2011 and 2012 responses (due to different instructors, pedagogies, or software tools), we computed z-scores assuming a normal distribution approximation to these binomial surveys [4]. The column p indicates the confidence level that results in each row may be different due to a nonrandom effect. Normally, any confidence level below 90% is less than significant. Plus, it is difficult to infer meaningful results from our research due to small sample sizes (20-25 students per sample). However, by triangulating these survey results with instructors' classroom observations and student grades, and with experimentation of a similar approach with other audiences (K-12 teachers and students), we would like to make a few preliminary conclusions.

Almost all students in the three gen-education courses liked project-based learning, which involved design of a game or a science experiment. More than 95% of them recommend others to take these courses. A significant portion of students (60%-80%) thought that modeling improved their understanding of science concepts and motivated them to pursue additional courses in computing and sciences. Although there are challenges of learning multiple fields in a single course, student skills can grow along with challenges to provide them an optimum *Flow* experience (Fig. 3). While a sizable number of students thought they did not initially have necessary background and skills, they eventually

overcame these difficulties through professor's help, practice, project-based learning, and scaffolding. The level of frustration has gone down in all courses and this may be due to several factors, including the use of more friendly tools, change of instructor in CPS 101, and improvements in the way we teach them. In 2011, the level of frustration was as high as 32% in CPS 101 (due to mathematics and programming), yet the percentage of students who thought that the new skills and knowledge would help them in future courses was also very high. With the change from heavy programming and mathematics to the use of more friendly tools such as IP and AS, students felt more engaged and confident but they did not see a high prospect of using the new toolset and interdisciplinary approach in other classes (down from 96% to 56%). This may be cultural and time needs to pass before it settles like the other two courses where the level of frustration is low (<10%), desire to take another course on the topic is high (70%-80%), and confidence in the later usage of newly learned skills is also high (74%-80%).

Table 5: Survey results from CPS 101. The last column (p) shows statistically the confidence level that there is any difference between responses in 2011 and 2012 for each row. The z scores are calculated based on two proportions [4].

Survey Questions (Q)	Y E S		Difference?	
Responses are in percentages (%).	2011	2012	z	p (%)
1. Recommend this course?	97	94	0.46	35
2. Like to take another course?	44	63	1.20	78
3.Modeling improve science learning?	68	82	1.02	70
4. Like project-based learning?	90	94	0.47	36
5. Had necessary background?	60	75	1.01	70
6. Your skills a match for challenges?	78	82	0.32	26
7. Ever felt frustrated?	32	25	0.49	38
8.Skills may help you in later classes?	96	56	2.96	99
9. Changed your major after?	10	13	0.30	24

Table 6: Survey results for CPS 105 and 302.

Q	CPS105				CPS302			
	Y E S (%)		Difference?		Y E S (%)		Difference?	
	2011	2012	z	p(%)	2011	2012	z	p (%)
Q1	100	100	0	0	100	96	1.01	68
Q2	85	80	0.47	36	50	70	1.44	85
Q3	78	70	0.64	48	61	83	1.73	92
Q4	98	100	0.71	52	95	90	0.67	50
Q5	69	91	1.94	95	88	80	0.77	56
Q6	80	91	1.10	73	100	92	1.44	85
Q7	26	7	1.81	93	20	5	1.60	90
Q8	85	74	0.96	66	89	80	0.88	62
Q9	6	9	0.40	31	0	4	1.01	69

Classroom observations and attendance records from instructors indicate a significant improvement in student behavior and participation in hands-on lab activities. While the attendance rate in the lecture session in classroom was around 70%, it jumped to 90% in the computer lab. They seem highly engaged in lab activities and involved in practicing different computational tools. A different instructor taught the CPS 101 in 2012, and this may have impacted the course dynamics. Other more systematic changes mentioned before (shift away from programming and differential equations) took place before the change of instructors, however, we should note that the new instructor (Sounthone Vattana; MS in Computational Science-Brockport; and MS in Educational Technology-Robert Wesleyan) is a former K-12 teacher and appears to have pedagogical skills with deep content knowledge in mathematics, programming and general science. He also teaches the other two gen-education courses.

Beyond its college-level use as reported above, the C-MST approach has been introduced, through teacher training, into secondary school classrooms in two partnering school districts (Rochester City SD and Brighton Central SD). The content of teacher training and its overall impact on teacher retention and student achievement are being documented in other publications, but here we will briefly mention the surveys on student engagement. Majority of the 200 trained teachers agreed that using modeling tools (IP, AS, Excel, and GSP) in their classrooms increased student engagement. 100% of science and 97% of math teachers agreed that C-MST initiative made math and science concepts more comprehensible to students. Student reaction to modeling (versus traditional techniques) was found to be 97% favorable in math and 77% in science classes. 100% of technology, 72% of math, and 31% of science teachers reported observed improvement in problem solving skills. This order may be linked to low reliance and utilization of mathematical and computational skills in science courses as a result of limited access to computers and possibly lack of available science-related modeling examples. However, while science classes utilized technology less, in instances where it was utilized, it led to a deeper understanding of science topics than it did for math topics (83% in science and 76% in math).

While computational modeling has been shown as an effective pedagogy to expose students to science concepts in an incremental fashion, by using tools that hide the underlying mathematics and science involved in the simulations, it can also motivate them to learn computer programming. By using multiple tools (IP, Excel, and Python) to solve the same problem, students had a chance to weigh advantages of each tool and conclude first-hand that more accurate and faster computation of *new =old + change* for a large number of data points will require computer programming. Additionally, various programming concepts (variables, loops, memory hierarchy, and data types, etc.) are learned in the process. For example, to simulate the orbital motion of an object with Python, a number of variables are needed to store elapsed time, time increment, acceleration, velocity, and position. To predict the velocity and position at the next time step, mathematical operations are used based on the relationships among acceleration, velocity, and position. To find the relations of velocity or position with respect to time, a loop is used to perform repeated calculations. To ensure correctness, the calculations of acceleration, velocity, and position have to be put in sequential while logically right order. Finally, in the context of applications, it is easy for students to understand why and how to learn computer programming. They show more willingness to learn computer programming in order to tackle real-world applications.

A strong link is established between computing and natural sciences through the computation of *change*. For example, change in position and in velocity requires computation of acceleration, which requires knowledge of the Force. This not only links mathematical, computational, and scientific inquiry, it also reinforces in an inductive way the 'learning the fundamentals of laws of nature' and it simplifies the great complexity of the universe into a handful of natural laws (gravity, electromagnetism, and nuclear interactions) that one can learn in a general science course. At the same time, a link between mathematics (numerical integration) and science applications is established, which can be used both in math courses (in the context of *rate of change*, *building functions,* and *modeling*) and in science courses (in the

context of *computational thinking* (CT) as a method of science inquiry). The new K-12 learning standards support teaching of CT skills as early as the 5th grade (See http://www.corestandards.org/ for math and http://www.nextgenscience.org/ for science standards).

6. CONCLUSION

The practice of teaching introductory computing courses in the context of natural sciences (or vice versa) is a promising means of enhancing both General Education and the STEM education. When computational tools are used, students seem more engaged in the class, and their attitude toward learning is more active. While science majors get to use simulation tools and computer programming to solve science problems, math and computer science majors get to establish a link to natural sciences at an abstract level, through computation of change caused by natural laws, which projects science in a universal and simplistic framework. Those with curiosity could then acquire a deeper knowledge and pursue additional courses or even a career either in mathematics, computing, or natural sciences. About 50% of non-STEM majors in these reported courses have shown an orientation to add STEM to their education; some (40%) through additional courses and some (10%) through a degree. We believe that solving real world problems by writing computer programs urges students to acquire knowledge through scientific inquiry beyond memorizing laws of science, encourages them to spawn ideas of computational thinking, and fosters them to develop habits of scientific thinking.

A major contribution of computational approach to STEM education is that it integrates math, science and computing in a single unit, exposes and trains students with multiple skills, which are useful in their future careers. While we suggest inclusion of computationally oriented general education science courses for all college freshmen [11-12], attention to a more fundamental concept, computational thinking (CT) at secondary school level is also needed as a long-term strategy [25]. A focus on CT could not only help improve science and computing education at college level but it might also push scientific thinking into mainstream to address underlying causes of the rising Category-5 storm in nation's K-12 education [25, 16-17]. Since abstraction is part of a CT skillset, through modeling and simulations, students can learn not only abstraction skills but also a whole set of other CT skills such as algorithmic thinking, decomposing a problem into smaller chunks, understanding the computational cost for more accuracy, and realizing the need to use a programming language in order to handle complexity and increasing number of data points.

The findings presented here cannot be generalized due to small sample size, limited audience, and lack of reliable and valid instrumentation to assess knowledge, attitudes, and skills. The education research in computing is very new, unlike physics, mathematics, and statistics [10]. We need to pay more attention to findings of learning science. As outlined in [14], we need to: a) draw out and work with the preconceptions and misconceptions of learners; b) help them take control of their learning in a constructivist environment; c) teach subject matter in depth with many examples, and d) employ pedagogical approaches such as metacognition, scaffolding, and project-based learning. The roles for assessment need to be expanded beyond traditional testing to use frequent formative assessment that would help make students' thinking visible to themselves, their peers, and instructors.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) funds via Grant #0942569. We would like to thank to faculty and teachers whose efforts contributed to the development, teaching, and assessment of the reported courses and materials.

8. REFERENCES

[1] AIP Survey. Important Knowledge & Skills Used on the Job. American Institute of Physics. http://www.aip.org/statistics/trends/highlite/; 1999: /bachplus5/figure2.htm. 2010: /emp3/figure4b.htm.

[2] Augustine, N. 2007. *Is America Falling Off the Flat Earth?* Washington, D.C.: The National Academic Press.

[3] BLS Report. 2010. The Bureau of Labor Statistics. Occupational Employment Statistics. http://www.bls.gov/oes/2010/may/stem.htm.

[4] Brase, C. H. and Brase, C. P. 2012. *Understandable Statistics*. 10th Edition. ISBN: 0840048386. Also, see Stat Trek web site on Hypothesis Test: Difference Between Proportions. http://stattrek.com/hypothesis-test/difference-in-proportions.aspx.

[5] Congressional Research Service (CRS) Report. 2008. STEM education: Background, Federal Policy, and Legislative Action. http://fas.org/sgp/crs/misc/RL33434.pdf.

[6] Computing Curricula 2005: The Overview Report. A Cooperative Project of the Association for Computing Machinery (ACM), the Association for Information Sciences (AIS), and the IEEE Computer Society (IEEE-CS).

[7] Creswell, J. W. 2012 *Educational Research: Planning, Conducting and Evaluating Quantitative and Qualitative Research*. 4th Edition. Pearson Education, Inc.

[8] Csikszentmihalyi, M. 1990. *Flow: The Psychology of Optimal Experience*. New York: Harper Collins.

[9] Cuny, J. 2011. Transforming Computer Science Education in High School. *IEEE Computer*, June 2011, 107-109.

[10] Fincher, S. and Petre, M. 2005. *Computer Science Education Research*. Taylor&Francis e-Library: London and New York.

[11] Goode, J. and Margolis, J. 2011. Exploring computer science: A case study of school reform. *Transactions on Computing Education*. 11(2).

[12] Guzdial, Mark. 2009. Teaching computing to everyone. *Communications of the ACM* 52: 31.

[13] Haight, C. E., Herron, C., and Cole, S. P. 2007. The Effects of Deductive and Guided Inductive Instructional Approaches on the Learning of Grammar in the Elementary Foreign Language College Classroom. *Foreign Language Annals*, 40 (20), 288-301.

[14] How People Learn: Brain, Mind, and School. 2000. The National Academies Press. Wash., D.C. http://www.nap.edu.

[15] Landau, R. 2006. Computational Physics: A Better Model for Physics Education? *IEEE Comp. in Sci & Eng.*, 8 (5), 22-30.

[16] NAP Report. 2007. Rising Above The Gathering Storm. Washington, D.C.: The National Academy Press. http://www.nap.edu/catalog/11463.html.

[17] NAP Report. 2010. Rising Above The Gathering Storm, Revisited: Washington, D.C.: The National Academy Press. http://www.uic.edu/home/Chancellor/risingabove.pdf.

[18] NSF Report on Cyberlearning. 2008. Fostering Learning in the Networked World. National Science Foundation. http://www.nsf.gov/pubs/2008/nsf08204/nsf08204.pdf.

[19] NSTA (National Science Teachers Association). 2008. Technology in the Secondary Science Classroom. (Eds) Bell, L. R., Gess-Newsome, J., and Luft, J. Washington, DC.

[20] Repenning, A. 2012. Programming Goes Back to School. *Communications of the ACM*, 55 (5), 35-37.

[21] Science and Engineering Indicators. 2010. National Science Board. http://www.nsf.gov/statistics/seind10/c2/c2s2.htm.

[22] Sjøberg, S. and Schreiner, C. 2005. How do learners in different cultures relate to science and technology? Results and perspectives from the project ROSE (http://roseproject.no). *Asia Pacific Forum on Science Learning & Teaching*, 6, 1-16.

[23] Swanson Survey. 2010. A Survey of Computational Science Education. By C. Swanson. The Krell Institute, http://www2.krellinst.org/services/technology/CSE_survey/.

[24] Tenenbaum, J. B., Kemp, C., Griffiths, T. L. & Goodman, N. D. 2011. How to Grow a Mind: Statistics, Structure, and Abstraction. *Science,* 331, 1279-1285.

[25] The College Board. 2011. AP CS Principles Course. http://www.csprinciples.org. Also see June 2012 *ACM Inroads*.

[26] Wenglinsky, H. 2005. *Using Technology Wisely: The Keys to Success in Schools*. New York: Teachers College Press.

[27] Wing, J. M. 2006. Computational Thinking, *Communications of the ACM*, Vol. 49, No. 3, 33-35.

[28] Yaşar, O., Rajasethupathy, K., Tuzun, R., McCoy, A. and Harkin, J. 2000. A New Perspective on Computational Science Education, *IEEE Comp. in Sci & Eng*, **5** (2), 74-79.

[29] Yaşar, O. 2001. Computational Science Education: Standards, Learning Outcomes and Assessment. *Lecture Notes in Computer Science,* 2073, 1159-1169.

[30] Yaşar, O. and Landau, R. 2003. Elements of Computational Science & Eng. Education, *SIAM Review*, 45, 787-805.

[31] Yaşar, O. 2004. C-MST Pedagogical Approach to Math and Science Education. *Lect Notes in Comp Sci*, 3045, 807-816.

[32] Yaşar, O., Little, L., Tuzun, R. Rajasethupathy, K., Maliekal, J. and Tahar, M. 2006. Computational Math, Science, and Technology, *Lecture Notes in Comp Science*, 3992, 169-176.

[33] Yaşar, O., Maliekal, J., Little, L. J. and Jones, D. 2006. Computational Technology Approach to Math and Science Education. *IEEE Comp. in Sci & Eng.,* 8 (3), 76-81.

Introducing Transition Matrices and Their Biological Applications

Angela B. Shiflet
Department of Computer Science
Wofford College
Spartanburg, S. C. 29303 USA
001-864-597-4528
shifletab@wofford.edu

George W. Shiflet
Department of Biology
Wofford College
Spartanburg, S. C. 29303 USA
001-864-597-4625
shifletgw@wofford.edu

ABSTRACT
The Blue Waters Undergraduate Petascale Education Program (NSF) sponsors the development of educational modules that help students understand computational science and the importance of high performance computing. As part of this materials development initiative, we developed two modules, "Time after Time: Age- and Stage-Structured Models" and "Probable Cause: Modeling with Markov Chains," which develop application problems involving transition matrices and provide accompanying programs in a variety of systems (C/MPI, C, MATLAB, Mathematica). Age- and stage-structured models incorporate the probability of an animal passing from one age or stage to the next as well as the animal's average reproduction at each age or stage. Markov chain models are based on the probability of passing from one state to another. These educational materials follow naturally from another Blue Waters module, "Living Links: Applications of Matrix Operations to Population Studies," which provides a foundation for the use of matrix operations. This paper describes the two modules and details experiences using the resources in classes.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education - Computer Science Education, Curriculum

General Terms
Design, Experimentation, Measurement.

Keywords
Computational Science, Matrices, Linear Algebra, Educational Modules, High-Performance Computing, Petascale, Blue Waters, Undergraduate.

1. INTRODUCTION
With NSF funding, the Blue Waters Undergraduate Petascale Education Program [1] is helping to prepare students and teachers to utilize high performance computing (HPC), particularly petascale computing, in computational science and engineering with the following three initiatives:

- Professional Development Workshops for undergraduate faculty
- Research Experiences for undergraduates
- Materials Development by undergraduate faculty for undergraduates

The goal of the Materials Development initiative is "to support undergraduate faculty in preparing a diverse community of students for petascale computing."

For this program, the authors developed and class tested the computational science related modules "Time after Time: Age- and Stage-Structured Models" and "Probable Cause: Modeling with Markov Chains," which are available at [2] and [3], respectively, on the UPEP Curriculum Modules site. This paper describes and discusses the modules and experiences using both in the course Modeling Biological Networks and class testing the first module in Linear Algebra and a course on Modeling and Simulation at Wofford College [4].

Several of the students in the classes at Wofford are pursuing an Emphasis in Computational Science (ECS). By taking Calculus I, Introduction to Programming and Problem Solving (in Python), Data Structures (in Python and C++), Modeling and Simulation, and Data and Visualization and doing a summer internship involving computation in the sciences, Bachelor of Science students may obtain an ECS [5]. Matrices are an important data structure in numerous computational models, and introducing transition matrices and eigenvalues with a variety of applications provides motivation to students in mathematics, computer science, and the other the sciences as well as in the Emphasis in Computational Science.

2. MODULES
2.1 Pedagogy
Prerequisites for the modules "Time after Time: Age- and Stage-Structured Models" and "Probable Cause: Modeling with Markov Chains" are minimal, requiring an understanding of matrix multiplication and the maturity to read the material but no programming or calculus background. Those who do not know how to multiply matrices or how to multiply a matrix times a

vector might wish to cover first another Blue Waters module, "Living Links: Applications of Matrix Operations to Population Studies," by the same authors [6].

Students using the modules at Wofford College ranged from first- to fourth-year with majors from biology, chemistry, computer science, environmental studies, mathematics, physics, and undecided. The modules provide the biological background necessary to understand the applications; assuming an understanding of matrix multiplication, the mathematical background needed to complete the exercises and projects; and references for further study. Multi-part quick review questions throughout (three (3) in "Age- and Stage-Structured" and sixteen (16) in "Markov Chains") with answers at the end of the modules provide immediate feedback. The modules also have exercises (five and three, respectively) for reinforcement and practice and project assignments (eight or nine, respectively) for further exploration using a computational tool.

To aid in exploration of the multi-scale aspects of the science and the computing process, example solutions involving serial and parallel model development accompany the modules. For an age-structured model, serial programs are available in MATLAB, Mathematica, and C, while HPC programs in C with MPI illustrate parallel parameter sweeps and matrix partitioning. Bioinformatics programs using Markov models to help locate genes are available in MATLAB, C, and C/MPI. (Blue Waters Student Intern Jesse A. Hanley implemented a matrix partitioning program, and Intern Whitney E. Sanders developed the parameter sweeps and Markov model in C/MPI.) Several datasets for use in projects also accompany the modules.

2.2 Age- and Stage-Structured Matrices: Module Content and Applications

"Time after Time: Age- and Stage-Structured Models" considers situations that classify individuals in a species by age, such as Years 1, 2, and 3, or stage, such as larvae, juvenile, and adult. Solutions employ matrices to determine the intrinsic growth rates, the proportion of each group in a stable distribution, and how sensitive the long-term population growth rate and predicted time of extinction are to small changes in parameters. We can employ the latter to determine the best category to target for conservation efforts for endangered species and for eradication efforts for pests.

Figure 1 presents a state diagram for a problem with the states denoting ages (Year 1, 2, or 3) of a bird. The left-pointing arrows represent fecundity or reproduction: A Year 2 (ages 1-to-2 years old) mother has a mean of five (5) female offspring, while a Year 3 (ages 2-to-3 years old) mother has four (4) female offspring on the average. The right-pointing arrows indicate survival rates of $P_1 = 15\%$ and $P_2 = 50\%$ from Year 1 to Year 2 and from Year 2 to Year 3, respectively. The information can be consolidated into a matrix, called a Leslie matrix, as follows:

$$\begin{bmatrix} 0 & 5 & 4 \\ 0.15 & 0 & 0 \\ 0 & 0.50 & 0 \end{bmatrix}$$

The module shows that over time the percentage of eggs/chicks stabilizes to 82.06% of the total population, while Year 2 birds comprise 12.05% and Year 3 birds are 5.90% of the population. Moreover, eventually each age group changes by a factor of $\lambda = 1.0216$ (102.16%) from one year to the next, and this λ is the dominant eigenvalue for the matrix.

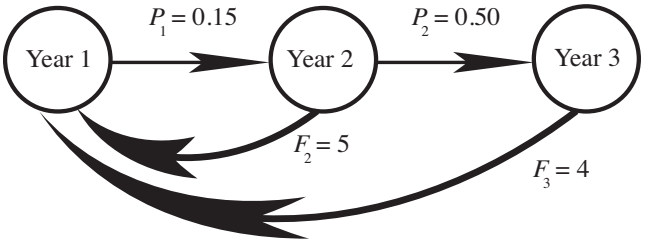

Figure 1. State diagram for age-structured problem

The sensitivity of λ with respect to P_i, $(\lambda_{new} - \lambda) / (P_{i,new} - P_i)$, measures the numeric impact on λ of a change in P_i. For small changes in P_i, the module shows that λ is most sensitive to changes in survivability of Year 1 birds, P_1. Thus, conservationists should probably concentrate their efforts on helping eggs and nestlings survive.

The module also covers a stage-structured model of the Indo-Pacific lionfish, an invasive and destructive species to reef habitats. Figure 2 illustrates that the model also includes probabilities for an animal remaining at the juvenile and adult stages. From this information, we can form a matrix similar to the Leslie matrix, called a Lefkovitch matrix, as follows:

$$\begin{bmatrix} 0 & 0 & 35315 \\ 0.00003 & 0.777 & 0 \\ 0 & 0.071 & 0.949 \end{bmatrix}$$

Module material and a project explore intrinsic grow rate, stable population distribution, and sensitivity analysis to make recommendations for controlling this menace.

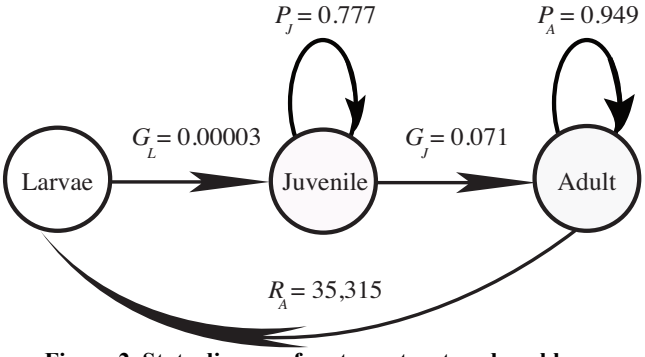

Figure 2. State diagram for stage-structured problem

2.3 Markov Chains: Module Content and Applications

The module "Probable Cause: Modeling with Markov Chains" also considers biological problems whose solutions involve transition matrices. Markov chain models (MCM) are based on the probability of passing from one state to another. Developing the necessary probability theory and biological background, the module solves a variety of problems using MCM from predicting the behavior of animals to locating genes in the DNA.

One problem considers a simplified system where the monkey is only in two states, eating (E) and resting/sleeping (R). Figure 3 enumerates the probabilities of moving between states, and the following transition matrix captures this data:

$$\begin{bmatrix} 0.6 & 0.2 \\ 0.4 & 0.8 \end{bmatrix}$$

As time passes, the module shows that the proportions approach 1/3 and 2/3 for eating and resting, respectively.

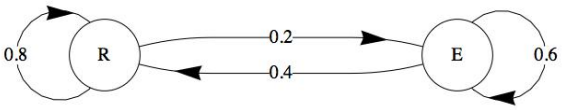

Figure 3. State diagram for Markov chain problem

We can employ such Markov models in a variety of problems in bioinformatics, which deals with the organization of biological data, such as in databases, and the analysis of such data. In a similar fashion to the example of changes of state for the monkey, a Markov chain can model the mutation process in DNA.

Moreover, the GeneMark algorithm employs Markov models to help locate genes in a DNA sequence. In many organisms, the sequence of bases CG appears less that we would expect from random occurrences of C and G independently. However, small regions, called CpG islands, upstream (before) of many genes are rich in the sequence CG; so we can employ CpG islands to locate genes. The module body develops a simplified 1st-order Markov model using the probabilities of bases and pairs of bases, while a project considers the more involved GeneMark 5th-order Markov algorithm employing probabilities involving quintets and sextets of bases.

2.4 High Performance Computing in Modules

Following the aims of UPEP, both modules have example programs and sections that focus on high performance computing (HPC) related to their particular applications. The section "Parameter Sweeping with High Performance Computing" in the age-structured module discusses the utility of parameter sweeping, or executing a model for each element in a set (often a large set) of parameters or of collections of parameters. As stated in the module, "The results can help the modeler obtain a better overall picture of the model's behavior, determine the relationships among the variables, find variables to which the model is most sensitive, find ranges where small variations in parameters cause large output changes, locate particular parameter values that satisfy certain criteria, and ascertain variables that might be eliminated to reduce model complexity" [7]. Besides being very useful, such parameter sweeping is embarrassingly parallel.

An algorithm for finding genes discussed in the Markov chains module is also embarrassingly parallel. Using a particular Markov model to score every subsequence of 200 bases, high scores indicate a greater likelihood that the subsequence is in a CpG island and that a gene is to follow. Multiple processes can evaluate scores for different subsequences, speeding the task significantly. Besides this specific example, a section on "High Performance Computing and Bioinformatics" discusses the utility of high-performance computing in a variety of other applications in bioinformatics.

2.5 Blue Waters UPEP Internship Involvement

During the summer of 2010 and following academic year, student Jesse Hanley held a Blue Waters UPEP Internship to develop parallel versions of programs using C and MPI to support "Age- and Stage-Structured Models" and other modules. The following year, Blue Waters intern Whitney Sanders continued Jesse's efforts with HPC programs for both modules discussed in this paper. Their programs accompany the modules on the NCSI UPEP Curriculum Modules site [2]. Jesse's is planning to work in the HPC field, and both students will be pursuing graduate work.

2.6 Exercises and Answers in Modules

After the body of educational material, each module contains a section with multi-part exercises, while a subsequent section has answers to selected parts. For example, the section in the "Age- and Stage-Structure Models" includes five exercises with one problem from the research literature involving loggerhead sea turtles.

2.7 Projects

After the exercises, each module presents eight or nine large projects for students to complete as individuals or with a team. Instructions indicate to develop sequential or high performance computing versions.

"Age- and Stage-Structure Models" has projects based on the research literature including ones on Uinta ground squirrels, skate, red-cockaded woodpeckers, lionfish, Pacific salmon, Furbish's lousewort, and cane toads in Australia. An additional project involves determining and graphing the speedup factor versus the number of processes for various parameter sweeps.

The projects in the "Markov Chains" module include problems on the shapes of epithelium cells, succession in a forest, the dynamics of cattle fecal shedding of a pathogen, development of the BLAST algorithm for non-gapping local alignment of DNA segments, determination using the GeneMark algorithm of the most likely candidates for subsequences being in CpG islands, the Stepping Stone Model useful in genetics, and DNA sequence evolution.

3. TESTING AND EVALUATION

3.1 Class Testing of Age- and Stage-Structured Module

"Age- and Stage-Structured Models" was class tested in three courses, Modeling and Simulation in fall of 2011, Modeling Biological Networks in January of 2012, and two sections of Linear Algebra in spring of 2012. In the first class, two Emphasis in Computational Science (ECS) students successfully implemented a system dynamics model for the age-based example in the text.

Four biology majors and one triple major in chemistry, mathematics, and computer science considered the module in greater depth in a January interim on Modeling Biological Networks taught by the authors. During the interim, students take only one course not in the usual curriculum. Each day, students attended class for three hours, where they made presentations and worked on projects, and then continued developing projects between classes. Four of the students were freshmen, while one was a sophomore; and three of the students are pursuing the ECS. Before considering "Age- and Stage-Structured Models," the class worked through two MATLAB tutorials and the module "Living Links: Applications of Matrix Operations to Population Studies" to gain a background in matrices [6]. Students read each module and worked exercises before class and took a short quiz on the quick review questions in class. Over a three-day period, each student individually or with a partner developed and presented an age-structured and a stage-structured project. In all, the class successfully completed six different models using MATLAB.

Also reading the material before class, forty-one (41) students in two sections of Linear Algebra taught by Dr. Ted Monroe studied

age-structured models one day the last week of classes in Spring, 2012. Students in this course, typically sophomores, come from a variety of majors and minors including mathematics, computer science, biology, chemistry, and physics. During class, the professor focused more on the mathematics and less on the biology for about 30 minutes. Examining the bird population diagram in Figure 1, he ensured that the students understood the system of equations, notation, and the matrix-vector equation. He reminded the class that the system is equivalent to a matrix-vector equation and that that they had looked at linear difference equations previously. Unlike problems where populations stabilized, he noted that this population problem led to growth over time. Also, the professor pointed out the population growth equations in annual and exponential form. However, the focus of the class was on the connection of the model to eigenvalues and eigenvectors.

3.2 Evaluation of Age- and Stage-Structured Module

Immediately after using the material, students in the interim on Modeling Biological networks completed a questionnaire about the module. The questionnaires had the students rate the following statements from 1 (strongly disagree) to 5 (strongly agree):

- I understood the science applications in the module.
- I understood the mathematics in the module.
- The module was readable.
- The Quick Review Questions helped me understand the material.
- The exercises helped me understand the material.

Means of the four responses were 4.5, 4.0, 4.25, 4.75, and 4.5, respectively.

In Linear Algebra, the professor had the students complete the questionnaire at the beginning and the end of class. Excluding students who had not read the material in the "before" category, Table 1 summarizes the results for questions 1-3. Unfortunately, the page with answers to the Quick Review Questions was not included with their materials, reducing their effectiveness. Moreover, students were not required to work exercises before class. Thus, particularly for the "Before" column, answers to the first three questions are more meaningful than those for the last two questions.

Table 1. Means, 1 (strongly disagree) to 5 (strongly agree)

Question	Before	After
I understood the science applications in the module.	4.22	4.45
I understood the mathematics in the module.	4.24	4.74
The module was readable.	4.16	4.40

Two elaborated on the above scores, "I enjoyed reading about applications," and "The mathematics in this module was easy to understand and the questions helped to reinforce what I had read."

Some of the comments in the questionnaire given at the beginning of class on what the student liked best about the module follow: "The module was easy and enjoyable to read. The information on the turtles was really interesting." "I liked how it was able to apply to real-life situations." "Instantly visualizable, elegantly simple, easy to understand what each structure/value represented." "I liked that the math we are learning now can be applied to a real-world problem to help understand endangered species." "I found it interesting that the example used actually converges to a specific percentage ratio." "Quick review questions and examples." "The introduction was very engaging and the mathematics was displayed very clearly and concisely." "I liked the subject matter and the mathematical applications." "I enjoyed the complete description of everything discussed and its relevance to science." "The module was very readable. The science applications were well explained, and the examples were helpful." "I liked that the module was easy to follow and the applications were clear." "The mathematical procedures were very easy to follow. I also liked the real-life applications of this kind of math." "The module was set up very well. The introduction and topic were interesting." "I thought the explanations of the science was interesting so it made following the math easier." "I enjoyed the idea of linking mathematical models to explain and possibly solve world issues I appreciate personally as an environmental enthusiast." "I liked the science applications. I'm a science person in general, and I also enjoy math, so putting the two together makes me happy." "The projected population growth rate [using the dominant eigenvalue] was very interesting." "I liked the biology aspect of it! (I'm a biology major, and really like how math works with living systems.)"

The few students who indicated any difficulties with the module were challenged by the mathematics, particularly the concept of "eigenvalue." However, on the questionnaire at the end of class, they indicated they now understood the concept and how to compute the eigenvalues. One student found typographical errors in the module, which the authors have corrected.

3.3 Class Testing of Markov Chains Module

After completing the "Age- and Stage-Structured Models," for three days the Modeling Biological Networks interim class turned its attention to "Markov Chains." Reading the module before class and completing two exercises, the group checked the exercises in class and had a quiz on selected quick review questions. Students in pairs or individually developed and presented two models each with the class completing a total of five projects.

3.4 Evaluation of Markov Chains Module

With the same questionnaire as for "Age- and Stage-Structured Models," averages on the questions (4 responses) were 5.0, 4.0, 4.5, 4.5, and 4.25, indicating that the most challenge came from the mathematics in the module. Some responses to a question on what the student found most difficult in the module reinforced this perception: "Figuring out the math with the transition matrices and the length-normalized log-odds score," and "I didn't fully understand some of things in the module until we talked about them in class the next day (i.e., transition matrices and ultimate distributions)."

However, other remarks on what the student liked best included the following: "It was easy to understand, and it gave everything to the reader that the reader needed to know." "Great explanations of calculating probabilities." "The definitions/rules sections always help me understand the material in these modules. Having answers to the QRQ's is also helpful for understanding the

content." "I enjoyed the quantity of helpful exercise and quick review problems."

Some of the additional comments were as follows: "I have always found probabilities to be difficult, but this module helped me understand them much better." "I like that all of the material ties into real-life problems. It makes everything much more interesting and sometimes more understandable as well." "I found this module very helpful. I was fully able to quickly grasp the information provided in each section and use the knowledge to work the review questions." "This is a very well thought out, organized, and helpful module."

One suggestion was, "Include answers to the exercises." In response, the authors added a section of answers to selected exercise parts.

4. CONCLUSION

"Time after Time: Age- and Stage-Structured Models" and "Probable Cause: Modeling with Markov Chains" and their associated programs in MATLAB, Mathematica, and C/MPI are available on the UPEP Curriculum Modules website [2, 3]. Class testing of the modules in Modeling Biological Networks, Modeling and Simulation, and Linear Algebra helped refine the modules and showed their utility in introducing applications of matrices, eigenvalues, parameter sweeping, and HPC concepts. High questionnaire scores and enthusiastic comments from undergraduates in three different types of courses verify the conclusion that "Time after Time: Age- and Stage-Structured Models" and "Probable Cause: Modeling with Markov Chains" can be an effective educational modules in a variety of classes, levels, and settings.

5. REFERENCES

[1] National Computational Science Institute Undergraduate Petascale Education Program (UPEP). http://computationalscience.org/upep Accessed 3/5/11.

[2] Shiflet, A. and Shiflet, G. 2012. "Time after Time: Age- and Stage-Structured Models." National Computational Science Institute Undergraduate Petascale Education Program (UPEP) Curriculum Modules, UPEP Curriculum Modules site. http://www.shodor.org/petascale/materials/UPModules/ageStructuredModels/ Accessed 5/21/12.

[3] Shiflet, A. and Shiflet, G. 2012. "Probable Cause: Modeling with Markov Chains." National Computational Science Institute Undergraduate Petascale Education Program (UPEP) Curriculum Modules, UPEP Curriculum Modules site. http://shodor.org/petascale/materials/UPModules/probableCause/ Accessed 5/21/12.

[4] Wofford College. http://www.wofford.edu/ Accessed 3/5/11.

[5] Computational Science - Wofford College. http://www.wofford.edu/computationalscience/ Accessed 3/5/11.

[6] Shiflet, A. and Shiflet, G. 2011. "Living Links: Applications of Matrix Operations to Population Studies." National Computational Science Institute Undergraduate Petascale Education Program (UPEP) Curriculum Modules, UPEP Curriculum Modules site. http://shodor.org/petascale/materials/UPModules/populationMatrices/ Accessed 5/21/11.

[7] Luke, Sean, Deeparka Sharma, Gabriel Catalin Balan. "Finding Interesting Things: Population-based Adaptive Parameter Sweeping." 2007. In GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. Pages 86-93. ACM.

STEM-Based Computing Educational Resources on the Web

Tatiana Ringenberg
Purdue University
401 Grant St.
KNOY 231
West Lafayette, IN
tringenb@purdue.edu

Alejandra Magana
Purdue University
401 Grant St.
KNOY 231
West Lafayette, IN
admagana@purdue.edu

ABSTRACT

This paper explores the scope of educational resources found on the web along with teaching and learning materials that can assist in integrating computational thinking into the classroom. Specifically, this paper focuses on finding and describing existing learning environments that integrate computational thinking into a STEM discipline. This survey provides initial steps towards the creation of a comprehensive list of STEM-based resources, on the web, which can help teachers to supplement and support their decision making when creating STEM curriculum.

Keywords

Computational thinking, STEM, learning resources, supplemental materials web-based educational materials.

1. INTRODUCTION

Computing is increasingly used to extend the capabilities and therefore findings of scientific research. For example, computing has enabled scientific breakthroughs by facilitating researchers through computerized instrumentation and detailed simulations to generate, visualize, and understand large amounts of scientific data. In tandem with the pervasive role of computing in science and engineering, there is a growing recognition of the importance of computational thinking.

Computational thinking [1] has been recognized as a collection of understandings and skills required for new generations of students who are proficient not only at using tools, but also at creating them and understanding the nature and implication of that creation [2].

For the scope of this study we refer to computational thinking as the combination of disciplinary knowledge (e.g., physics, biology, nanotechnology) [3] with thought processes (e.g., engineering thinking, quantitative reasoning, algorithmic thinking, systems thinking) involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively analyzed by an information-processing agent [4]. This requires using a set of concepts, such as abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts [5, 6].

2. BACKGROUND

Learning science, science education and cyberlearning research and funding have resulted in high-quality internet-based learning resources that can now be re-purposed to emphasize computational thinking. In conjunction, there has been an increase of teachers who use online resources for planning and executing their learning activities [7, 8]. While educational digital libraries exists, teachers frequently turn to the internet to find learning resources [9]. Though it is known that these teachers are searching for resources on the web, it is unclear what criteria they are using to choose resources for use in the classroom.

In a study focused on investigating educators' expectations and requirements for the design of educational digital collections for classroom use [10], the authors found that in addition to the quality of the resource, these users expected additional contextual information in the resource. Similarly, a different study that focused on a case study of how teachers find and use online resources [11], the authors identified that teachers interested in finding these resources were seeking on them characteristics such as age-appropriateness, accuracy and contemporariness [11].

This paper explores the landscape and characteristics of STEM-based computing educational resources found on the web together with teaching and learning materials that can facilitate the integration of computational thinking into the classroom. Specifically, this paper describes strategies and instructional media of existing learning environments that can integrate computational thinking into a STEM discipline together with lesson plans, activities and other curricula organized by specific grade levels. The guiding research question for this study is:

> *What are the characteristics of web based resources that fall under the definition or are identified as resources that can promote STEM-based computational thinking?*

3. METHODS

We followed the approach created by Bagiati and colleagues [12] to identify resources that can be described as promoting computational thinking in STEM disciplines. We started by identifying a list of resources mentioned on national reports and in portals that compile resources related to computational thinking. We then made a distinction among portals. Portals strictly created for research purposes were not included in this study. Portals with resources created by the same contributor were considered a single source and were only listed once within

this study. Portals housing resources from different contributors and organizations were considered on a per-resource basis. Each website found within a portal was documented and noted as having been found in that particular portal. Any resource in the portal not pertaining to STEM was discarded from this study. Websites, within a given portal, which were found to be abandoned or for which the URL was found to no longer existing, were also discarded.

The next step was to identify additional resources on the web. These resources were found using a combination of the following search terms in Google: "computational thinking," "science," "technology," "engineering," "math," "STEM resources," "simulations," "lessons," "supplement," "K-12," "animations," and "teaching." Once all available resources through these searches were identified, we proceeded with their analysis. Our analysis started with identifying the primary instructional methods, instructional planning methods, scope of resources and evidence of research (e.g., publications, evaluations, feedback, testimonials or surveys) for each resource.

To identify the instructional method we identified whether the resources included simulations, games, programming environments, videos, images, animations, lectures, or books. We also noted whether instructional planning resources were available. Instructional planning resources are those supplemental materials consisting of motivation, orientation, information, application or evaluation activities [13]. Instructional planning resources were separated into lesson plans, tutorials, learning modules, activities, sample code, curriculum links, assignments and homework.

The procedure used in identifying the existence of research, instructional planning methods, instructional methods and program scope, consisted of applying the three click rule [14]. The three click rule refers to a heuristic applied to the design of the navigation for usable websites that argues that a user of a website should be able to find any information with no more than three mouse clicks [14]. Under this heuristic we assumed that users such as teachers may become frustrated if they cannot find the information within the three clicks, thus the name of "the three click rule." The complete search and analysis was performed for a second time to ensure accuracy and consistency.

4. ANALYSIS AND RESULTS

Results from this study identified 64 resources that were classified as STEM-based computational educational resources on the web, with 55 resources that can integrate computational thinking into K-12 classrooms to support learning in other STEM disciplines. Of the 55 resources, 31 were found to contain science materials, 22 contained technology materials, 14 contained engineering materials and 23 contained math materials.

As described earlier, the identified resources were then classified according to the primary instructional method. Table 1 depicts the number of resources found for each instructional method. Resources utilizing multiple instructional methods were also noted. These categories are described below.

Simulations refer to working representations of reality describing a model that may require some input parameters and then are executed by the learners. Programming environments refer to computing environments that embed a programming language mostly used for creating interactive stories and games. Games refer to electronic and interactive media played by means of manipulating images.

Videos were defined as the reproduction of visual images. Illustrations refer to static pictures and drawings while animations refer to a sequence of images to create the illusion of movement. Lectures were defined as notes or presentations. Books refer to electronic compendium of written materials. For the purpose of this study, books were also defined to include resources with a significant amount of text.

TABLE I
TYPE OF INSTRUCTIONAL METHOD

Instructional Method	Number of Resources
Simulations	25
Videos	13
Games	12
Books	10
Animations	9
Illustrations	9
Programming environment	8
Lectures	4

As shown in Table 1, eight different types of instructional methods were identified.

We also identified whether each of the resources included any supplemental material that could serve as an instructional planning resource. As shown in Table 2, there were eight types of planning resources identified. Supplemental materials are used by educators as a blueprint or guidance to teach and incorporate the primary instructional tools.

Lesson plans refer to structured goal and objectives provided to teachers for a specific day's topic. Learning modules are organized collections of content that can range from a single lesson (i.e., a week-long activity with the goal of learning a single concept) to an entire curriculum (sets of week-long activities encompassing interrelated concepts, principles, procedures and problem-solving for a specific course or grade level). Learning modules often contain multiple instructional methods. In this case, a tool would be listed as a Learning Module as well as the other categories under which it would fall.

TABLE II
TYPE OF INSTRUCTIONAL PLANNING RESOURCE

Instructional Method	Number of Resources
Assessments	26
Activities	26
Lesson plans	23
Curriculum link	17
Learning modules	14
Tutorials	13
Sample code	8
Homework assignments	1

Tutorials refer to instructions, in text or video form, which provide a teacher with a guide on how to integrate a particular resource into the classroom or how to use a particular resource.

Activities refer to in-class interactions designed to teach students through doing. Activities are generally short term and cover a single concept. Sample code refers to code provided, by the resource or by site contributors, for a particular programing language. The goal of sample code is to teach by example. Curriculum links refer to ties made from the resource to a standard and accepted curriculum. Assessments refer to quizzes

and tests that are used to gauge learners' progress to an extent. Assessments are generally monitored by the resource and results are provided to either the teacher or target audience. Homework assignments refer to work that is given outside of the classroom to reinforce concepts learned in the classroom.

TABLE III
RESOURCES BY GRADE LEVEL

Grade Level	Number of Resources
Pre-K	4
K	40
1	40
2	41
3	43
4	43
5	44
6	45
7	44
8	41
9	48
10	46
11	44
12	44
College	18
All	16
Varies	10
Not available	2

Another measurement taken was the target audience for each resource. The target audience is defined as the student grade level for which a given resource was created. Table 3 shows the number of resources available for each grade level. Each number in the first column of the table represents its corresponding grade level. The category "K" lists number of resources found for kindergarten levels. The category "C" represents resources targeted for college level students. The category "A" represents resources that are created for all ages and not for a specific grade level. "NA" category refers to resources that did not have grade level information available. The "V" category is for resources whose grade levels depend on the sub-resources.

Figure 1 shows the resources found to be used by audiences at multiple grade levels.

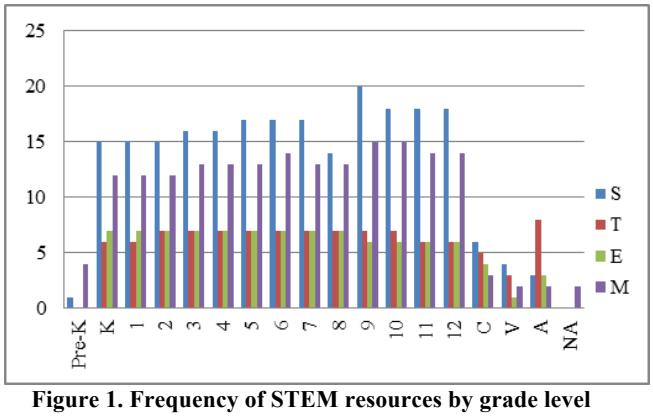

Figure 1. Frequency of STEM resources by grade level

Additionally we conducted searches aiming to identify available evidence of research or evaluation for each of the resources. Evidence of research is defined as some form of proof that the resource is effective or has been tested in some way. Table IV below divides these resources into publications, testimonials, evaluations, surveys and feedback.

TABLE IV
TYPES OF EVIDENCE OF RESEARCH

Instructional Method	Number of Resources
Publications	21
Feedback	5
Testimonials	4
Evaluations	3
Surveys	2

Publications provide formal evidence of the efficacy of a resource. Publications take the form of journals, conference papers, books and articles. Testimonials are responses from users provided by the resource. Evaluations are feedback in the form of pre-established criteria. Evaluations can be done by teachers, users or outside sources. Surveys are informal feedback in the form of multiple-choice questions asked to users about the product. Feedback is defined as reactions or concerns of users to a particular resource. There is either an email address provided for which feedback can be given or there is a specific link for feedback.

Our criteria in the analysis were that all forms of resource evidence must be available on the resource website or linked to the resource website. For instance, if a resource has no publications but provides an HTML link to a website that has used its resource for publications; the resource receives credit for publications. Resources that contained multiple forms of evidence were listed under each type of evidence under which they fell. Approximately 31 of the total of resources had no visible evidence of research or evaluation at all.

The last analysis concerned resource scope. Short resources were defined as those resources which were meant to be done within a single sitting. This means that, if used in class, it would take at most an hour or two of class time. Moderate resources are those resources which take more than a single class period, but no more than a week. Long resources are those resources that take an extended amount of time. They often include an entire curriculum. It was found that 44 resources included short resources, 22 included moderate and 16 included long.

Appendix A lists all resources identified through in this paper as well as all of the associated statistics as previously discussed.

5. DISCUSSION

This descriptive study provides an in depth view of (a) available web-based STEM computational resources, (b) primary tools used to teach STEM and (c) supplemental instructional tools that can be used by teachers to integrate the primary instructional tools. From the graphs and statistics listed in this paper, some trends can be suggested.

As an observational note, independent STEM resources are very difficult to find. Though there are several resource portals, many of them have either the same information or outdated and abandoned resources. Resources not indexed in portals were difficult to find and had to be encountered through the right combination of search terms. This led to an abundance of unusable resources to be discarded. Central, up to date, resource hubs are needed to make finding computational thinking resources easier for teachers.

Results from Figure 1 suggest that web resources are available, fairly consistently from K-12 grade levels. It is worth noting that there was a spike in resources for the 9-10 grade levels. Some resources specifically targeted for these grade levels were found particularly in math.

Educational resources for Pre-K, College, all ages and a varying audience are shown to be lacking. A potential explanation for this trend may be due to the broad nature of the age appropriateness of resources for each of the audiences. K-12 audiences have more focused curriculums with very specific standards that are more easily followed by resources. Audiences such as college professors and students, however, contain too many sub-audiences with varying curricula. Another possible explanation for the lack of resources outside of the K-12 audiences is the breadth of STEM disciplines outside of these.

From highest to lowest, the instances of STEM resources were as follows: science (31), math (23), technology (22) and engineering (14).

We expected the greatest quantity of resources to occur in science. This is due to the broadness of the topic. It is worth noting that most science resources were focused on individual areas of science. For instance, BioQuest is an excellent example focusing largely on biological sciences.

Upon first examination of technology and math, it appears that they are nearly on par. However, when broken down by grade level, it is evident that more math resources were available to each grade than technology resources. Around 30% of resources at each grade level, within K-12, were math resources whereas only about 17% were technology resources. This shows that, although there are several technology resources at our disposal, not many of them cover the full spectrum for K-12 audiences.

The lack of engineering resources is not unexpected. Engineering is a more advanced discipline that largely isn't taught in K-12. Engineering is also a difficult category to define without the use of science, technology and math.

Most of the resources found combine multiple branches of STEM. For instance, out of the 14 engineering resources found in this study, only 4 had solely engineering resources. It is believed that this is largely due to (a) the ambiguity of the understanding of the term engineering and lack of preeminence of traditional stipulated standards at the K-12 level and (b) the multidisciplinary nature and reliance of engineering on science, technology and math.

There are 38 more occurrences of supplementary resources than there are of the primary tools. Further analysis shows that simulations are the only significant source of primary instructional tools. Though there is an abundance of simulations at 25 total occurrences, the other primary instructional tools are somewhat lacking. In the future, we will look further into the other primary instructional tools to understand this gap. It is possible that this gap exists due to the nature of STEM.

Even though simulations were the most prevalent primary resources, they were the least explained. There were hardly any tutorials or guides on how to use the simulations. Many of these simulations were not intuitive. This leaves teachers or even learners often guessing its functionality.

Learning objectives were removed from the scope of this study due to our inability to easily find them. Very few resources had learning objectives listed. Most of the objectives that were found were based on the resource as a whole, not the individual units within each resource. There were few sites that listed the objectives based on each learning unit; however, these were not enough to be examined in this study. Learning objectives was not the only category that was removed from the findings.

6. CONCLUSION AND IMPLICATIONS

Research studies have reported that searching and verifying online learning resources poses a challenge to teachers already pressed for time [15]. As a result of this investigation we concur that independent STEM-based computational resources are very difficult to find. Though there are several resource portals, many of them have either the same information or outdated and abandoned resources. Resources not indexed in portals are difficult to find and had to be encountered through the right combination of search terms. This led to an abundance of unusable resources that had to be discarded.

Similarly, over half (31/55) of the total usable resources did not provide at least one of the categories used in this paper. This suggests that (a) STEM web resources may not be user friendly or (b) STEM resources are not well supported. All of the information that could be found within three clicks was added to this paper. Anything beyond three clicks was not included because it may exceed most users' tolerance for searching. This observation goes hand-in-hand with the lack of instructions for simulations. If resources are to be effective they must be well explained and must contain all necessary information within a reasonable number of clicks.

Research has also identified that many educators limit their information seeking primarily within search engines and do not take full advantage of educational-related digital libraries specifically designed and crated for specific teaching and learning needs [15]. While digital libraries can solve the problem of finding these resources, there is still the issue of additional characteristics users are looking for in each of them. As more and more of these resources move to digital libraries, instructional designers and educational researchers should keep in mind that educators are not only looking for a collection of resources, but also additional well-documented contextual information (e.g., age-appropriateness) and evidence of their high quality as well (e.g., accuracy and scientific evidence). This study provides initial steps towards that goal by providing a list of STEM-based computational resources on the web that includes useful additional information, which can help teachers and parents make decisions and eventually integrate these resources easily for educational purposes.

7. REFERENCES

[1] Wing, J.M., *Computational thinking*. Communications of the ACM, 2006. 49(2): p. 33-35.

[2] Soh, L.K., et al., *Renaissance computing: an initiative for promoting student participation in computing*. 2009.

[3] [NRC], *Report of a workshop on the pedagogical aspects of computational thinking*, in *National Research Council*. 2011, National Research Council of the National Academies: Washington, D.C.

[4] Cuny, J., L. Snyder, and J.M. Wing, *Demystifying Computational Thinking for Non-Computer Scientists*. Work in progress, 2010.

[5] [CSTA]. *Operational definition of computational thinking*. Computer Science Teacher Association 2012 [cited 2012 March 15]; Available from: .

[6] Barr, V. and C. Stephenson, *Bringing computational thinking to K-12: what is Involved and what is the*

role of the computer science education community? ACM Inroads, 2011. 2(1): p. 48-54.

[7] Hedtke, R., et al., *Service industry for teachers? Using the Internet to plan lessons.* European journal of education, 2001. 36(2): p. 189-193.

[8] Recker, M., *Perspectives on Teachers as Digital Library Users: Consumers, Contributors, and Designers.* D-Lib Magazine, 2006. 12(9): p. 2.

[9] Williams, T., *Teachers' Link to Electronic Resources in the Media Center: A Local Study of Awareness, Knowledge and Influence.* 2004.

[10] Sumner, T., et al. *Understanding educator perceptions of "quality" in Digital Libraries*. in *Proceedings of Joint Conference of Digital Libraries* 2003. New York, NY: IEEE.

[11] Recker, M.M., J. Dorward, and L.M. Nelson, *Discovery and use of online learning resources: Case study findings.* Journal of Educational Technology and Society, 2004. 7: p. 93-104.

[12] Bagiati, A., et al., *Engineering Curricula in Early Education: Describing the Landscape of Open Resources.* Early Childhood Research & Practice, 2010. 12(2).

[13] Newby, T., et al., *Instructional technology for teaching and learning: Designing instruction, integrating computers, and using media.* Educational Technology & Society, 2000. 3: p. 2.

[14] Porter, J., *Testing the three-click rule.* User Interface Engineering, 2003.

[15] Perrault, A.M., *An Exploratory Study of Biology Teachers' Online Information Seeking Practices.* School Library Media Research, 2007. 10.

AUTHOR INFORMATION

Tatiana Ringenberg, Undergraduate Student of Computer and Information Technology at Purdue University, tringenb@purdue.edu

Alejandra J. Magana, Ph.D. Assistant Professor of Computer and Information Technology at Purdue University, admagana@purdue.edu

Appendix:

Resource Name	Website (URL)	Grade Level	STEM field	Instructional Planning Resource	Instructional Method	Evidence Of Research	Length Of Resource
CyberChase	http://pbskids.org/cyberchase/	Pre-K-12	M	A	G, V	P,E	Short
eNLVM	http://enlvm.usu.edu/ma/nav/bb_school.jsp?sid=emready&coid=all	Pre-K-12	M	LP, LM, A			Short
Teacher Vision	http://www.teachervision.fen.com/	Pre-K-12	SM	LP, A, AS	B, I, G, V		Short - Moderate
eGFI	http://teachers.egfi-k12.org/	K-12	E	LP, A			Short
iCoachMath.com	http://www.icoachmath.com/static/AboutUs.aspx	K-12	M	CL, AS, T			Short
Let's Go Learn	http://letsgolearn.com/	K-12	M	AS, A, CL, LM	V,G		Short - Moderate
Mathalicious	http://www.mathalicious.com/	K-12	M	LP, CL, LM	V,I, B	SU	Short - Moderate
BrainPop	http://www.brainpop.com	K-12	S	LP, LM, T, A, CL	G, V, A	P	Short
Froguts	http://www.froguts.com/	K-12	S	LM, AS	S	P	Short
HHMI's BioInteractive	http://www.hhmi.org/biointeractive/stemcells/animations.html	K-12	S	LP, LM, A	S, V, L, A		Short
illumin	http://illumin.usc.edu/	K-12	S	LP	I, B		Moderate - Long
United States Department of Agriculture: Agriculture in the Classroom	http://www.agclassroom.org/	K-12	S	LP, LM, A	S, G, V		Short - Moderate
Engineering Is Elementary	http://www.mos.org/eie/20_unit.php	K-12	STEM	LP ,LM, CL, AS		P, E	Short

Resource Name	Website (URL)	Grade Level	STEM field	Instructional Planning Resource	Instructional Method	Evidence Of Research	Length Of Resource
Engineering Pathway	http://www.engineeringpathway.com/ep/k12/k12_curricular_res.jhtml;jsessionid=KOMDU3BKENICZABAVRSSFEQ	K-12	STEM			NOT APPLICABLE	
Lesson Planet	http://www.lessonplanet.com/lesson-plans/science	K-12	STEM	LP, LM, A, CL		F,A	Short - Moderate
NASA's Simulation-Based Aerospace Engineering Teacher Professional Development Program	https://simaero.rti.org/pages/Coursework.aspx	K-12	STEM			NA	
Teach Engineering	http://www.teachengineering.org/	K-12	STEM	LP, A, AS	I	P	Short - Moderate
teachers' domain	http://www.teachersdomain.org/browse/?fq_hierarchy=k12.sci.engin	K-12	STEM	CL, LM, A, LP	V, I		Short
Know It All	http://www.knowitall.org/nasa/ksnn/index.html	K-12	STM	LM, A	S, I	F	Short
Intel Computer Clubhouse Network	http://www.computerclubhouse.org/	K-12	T	In Person Instruction	PE	NA	
Concord Consortium	http://www.concord.org/	K-12, College	S	A, LM	S	P,A	Short
PhET	http://phet.colorado.edu/en/simulations/category/by-level	K-12, College	SM	A, AS, HW	S, A, L	P	Short
ROBOLAB @ CEEO	http://www.ceeo.tufts.edu/robolabatceeo/	K-12, College	TE	LP, T, A, SC, CL, LM	PE	A	Short - Long
Computer Science Teachers Association	http://csta.acm.org/	K-12, college, industry	T			P	Full Curriculum
Women @ SCS	http://women.cs.cmu.edu/	K-12, Women	T	CL	I		
Kid's Field Day	http://www.ksre.ksu.edu/fieldday/kids/	K-5	S	LM	I, S		Short
Mad Dog Math	http://www.maddogmath.com/about.html	K-6	M	A,AS, LM		E	Short - Long
The K-8 Aeronautics Internet Textbook	http://wings.avkids.com/	K-8	SE	LP, A	I		Short
Explore Leanring	www.explorelearning.com	3-12	SM	LP, CL	S, G	P	Short
Scratch	scratch.mit.edu	3-12	T	T	S, PE, V	P	Moderate
Computer Science in a Box	http://www.ncwit.org/unplugged	5-9	T				Full Curriculum
WISE	http://wise.berkeley.edu/	5-12	S	LP, LM, T, AS	S, A	P,A	Short - Moderate
EcoScience Works	www.fbr.org/swksweb/esw.html	6-8	S	LP, AS, A, CL, LM	S	P	Long
Engineer Your Life	http://www.engineeryourlife.org/	6-8, Girls	E		V		
sciencecourseware.org	http://nemo.sciencecoursesware.org/	6-12, college	S	AS,T, CL, LM	S, V	T,A	Short
Stock Trak	http://www.stocktrak.com/	6-12	SM		S		Long
Mathematics for the Digital Age and Programming in Python	http://www.skylit.com/mathandpython.html	9-12, college	M		B		Long

Resource Name	Website (URL)	Grade Level	STEM field	Instructional Planning Resource	Instructional Method	Evidence Of Research	Length Of Resource
DNA Learning Center	http://www.dnalc.org/about/	9-12, college	S	A	S, A	AR	Short - Long
Math Open Reference	http://www.mathopenref.com/index.html	10	M	AS, LM	I, B	T	Short
EDICS	http://ocw.mit.edu/ans7870/resources/edics/index.htm	College	E		S, A, L, B		Long
Statistics and Visualization for Data Analysis and Inference	http://ocw.mit.edu/resources/res-9-0002-statistics-and-visualization-for-data-analysis-and-inference-january-iap-2009/	College	TE		L	F	Long
MathWorks	http://www.mathworks.com/company/events/webinars/wbnr56249.html?id=56249&p1=869881405&p2=869881423	College	EM	T	V,S, PE	P	Short - Moderate
Evolution of Physical Oceanography	http://ocw.mit.edu/resources/res-12-000-evolution-of-physical-oceanography-spring-2007/	College	S		B		Long
Agripedia	http://oir.fod.msu.edu/OIR/BioAg/bioag_gen.asp	College	SE	CL			
DDA Medical Simulations	http://www.zeroonezero.com/medical/simulations/training-simulations/medical-technology-simulation.html	College	ST	A	S, A	P	NA
Open Sees	http://opensees.berkeley.edu/OpenSees/home/about.php	College	STEM	SC	S, PE		Short
Quizzes with a THEME	http://csta.acm.org/Resources/sub/ResourceFiles/BruceMaxwellThemeQuizzes.pdf	College	T	SC, AS	PE		Short
TryEngineering	http://www.tryengineering.org/	A	E	LP, CL	G	SU	Short - Moderate
Topology and Geometry Software	http://www.geometrygames.org/	A	M	T	G		Short
Discovery: Dinosaur Central	http://dsc.discovery.com/dinosaurs/dinosaur-games/dinosaur-viewer/dinosaur-viewer.html	A	S	LM	S, A, G		Short
Nobel Prize: All Educational Productions	http://www.nobelprize.org/educational/all_productions.html	A	S	LP, LM	G		Short
Google: Exploring Computational Thinking: Lessons and Examples	http://www.google.com/edu/computational-thinking/lessons.html	A	STEM	LP, SC, A, CL			Short - Moderate
Alice	http://www.alice.org/	A	T	T, LP, A, SC, CL, LM	S, PE, B	P,T	Short - Long
Beginner Developer Learning Center	http://msdn.microsoft.com/en-us/beginner/default.aspx	A	T	SC, T	V, B		Short
Computational Fairy Tales	http://computationaltales.blogspot.com/p/posts-by-topic.html	A	T		B	F	Short
Computer Science Unplugged	http://csunplugged.com/	A	T	A,CL	V	P	Long
Furby Autopsy	http://www.phobe.com/furby/	A	T	T, A			Moderate
Phrogram	http://phrogram.com/content/about.aspx	A	T	SC, LP, T	S, PE	T	Short - Long
GameMaker (by YoYo Games)	http://www.yoyogames.com/	A	TE	T, A, SC	S, PE, G		Short-Long

Resource Name	Website (URL)	Grade Level	STEM field	Instructional Planning Resource	Instructional Method	Evidence Of Research	Length Of Resource
NCWIT	http://www.ncwit.org/work.practices.html	A, Women	T				
BioQuest	http://bioquest.org/	V	S	A, LM, CL	S	P	Short - Moderate
Molecular Workbench	http://mw.concord.org/modeler/	V	ST	LM	S, A		Short - Long
Discovery Education	http://stem.discoveryeducation.com/	V	STEM	LP, CL, LM	V, G		Short - Long
Inter*activate*	http://www.shodor.org/interactivate/activities/	V	STM	LM, AS	S	F	Short

Grade Level: A=All; V=Varies NA=Not Available
Instructional Planning Resource: LP=Lesson Plans; T=Tutorials; A=Activities; SC=Sample Code; CL=Curriculum Link; AS=Assessments; HW=Homework Assignments; LM=Learning Module
Instructional Method: S=Simulation; PE=Programming Environment; G=Games; V=Videos; I=Illustrations/Pictures; A=Animations; L=Lectures; B=Books
Evidence of Research: P=Publications (Journals and Papers); A=Articles (magazines and online); T=Testimonials; SU=Surveys ; F=Feedback; NA=Not Available; AR=Annual Reports; E=Evaluations

Transformation of a Mathematics Department's Teaching and Research Through a Focus on Computational Science

Yanlai Chen
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8438
yanlai.chen@umassd.edu

Gary Davis
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8739
gdavis@umassd.edu

Sigal Gottlieb
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8205
sgottlieb@umassd.edu

Adam Hausknecht
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8322
ahausknecht@umassd.edu

Alfa Heryudono
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8516
aheryudono@umassd.edu

Saeja Kim
University of Massachusetts Dartmouth
285 Old Westport Rd
Dartmouth, MA 02747, USA
+1 508-999-8325
skim@umassd.edu

ABSTRACT

Undergraduate teaching that focuses on student-driven research, mentored by research active faculty, can have a powerful effect in bringing relevance and cohesiveness to a department's programs. We describe and discuss such a program in computational mathematics, and the effects this program has had on the students, the faculty, the department and the university.

Keywords

Computational science; scientific computing; undergraduate research; interdisciplinary research; transformation.

1. INTRODUCTION

We describe aspects of a transformation of a mathematics department that was initiated by a focus on computational science training for undergraduates and interdisciplinary research in computational science. National Science Foundation (NSF) support for this training initiative was instrumental in focusing undergraduate teaching around computational science and scientific computing, and also in assisting the development of a vision of the department as a major research and teaching over a period of four years. A significant feature of these changes is that teaching and research in scientific computing have become department in scientific computing. Execution of this vision led to some substantive changes in department structure and functioning progressively integrated one with the other, to the extent that motivated students have manifold opportunities to carry out significant research efforts from their earliest years, and to be mentored by research active faculty and graduate students.

2. DEPARTMENTAL DEVELOPMENT OVER 4 YEARS

In Spring 2008 the Department of Mathematics at the University of Massachusetts Dartmouth had two active researchers in numerical analysis, and one scholar in applied linear algebra, as its entire computational mathematics group. Additionally the Department had no graduate degree (M.S. or Ph.D.) in mathematics. By Summer 2008 the Department had lost one of the active researchers in numerical analysis. Fast-forward four years to Fall 2012 and the Department had the following features:

- A five-year NSF funded training program ($789,000) to train mathematics undergraduates in computational science, beginning September 2008.
- Seven active researchers in scientific computing, with a focus on numerical methods for solving partial differential equations. Their areas span finite-elements methods, radial basis functions, spectral methods, multi-scale methods, time discretizations, model order reduction, and uncertainty quantification. This group comprises two Full Professors one Associate Professor, and four Assistant Professors.
- Two Full Professors engaged in computational mathematics/scientific computing education research and scholarship.
- New computationally oriented mathematics courses, including a sophomore level course on scientific computing.

- Establishment of a University Office of Undergraduate Research. One of the authors was Foundation Director of this new Office, and CSUMS-RESCUE is an exemplar program for the Office.
- A Ph.D. program in Computational Science and Engineering.
- Over $2.5 million in competitive research, training or equipment grants over a four year period, in which Department of Mathematics faculty were PI or CoPI.
- A lead role in a University Center for Scientific Computing and Visualization Research, under the leadership of one of the authors (Gottlieb), involving partners from engineering, oceanography, and physics.
- A high performance computing facility, built through a partnership with mathematics and physics, and grants from the NSF and the Air Force Office of Scientific Research.
- A proposal for a BS/MS interdisciplinary program in Data Science initiated by one of the authors, sponsored by the Department of Mathematics, and supported by the Dean of Engineering and Applied Science and the University administration.
- A strong fit with the University strategic plan: the Provost and Chancellor now support Scientific Computing as a major research and training area for the University.

3. THE NSF COMPUTATIONAL SCIENCE TRAINING GRANT

Four faculty obtained an NSF training grant, CSUMS-RESCUE, for computational science for mathematics undergraduates. Before discussing the impact of this grant on the Department of Mathematics, we take a look at the structure and function of the undergraduate research workshops that became the visible face of the computational science program. Following that, in the next section, we examine in more detail the effect of this training grant on the structure and functioning of the Department and the University that led, in four years, to the developments described in the previous section.

3.1 Structure of Undergraduate Research Workshops

From the outset we conceived the CSUMS-RESCUE program as consisting of undergraduate research workshops, constructed as practical research sessions, in which students are inducted and mentored in research in computational science. We accept students into the program on the recommendation of instructors in other courses, and students at all academic levels, Freshman through Senior, are represented. Mathematics majors who work on a research project in a group with at least one other student are currently paid (approximately $18 per hour) for a maximum of two regular semesters and one summer. Summer sessions typically last 10 weeks with 20 hours a week. Mathematics majors, students from other disciplines, and those who wish to work singly, are free to participate at any time without receiving payment. We require all participating students to have a faculty research mentor. Student mentors might be from another department – physics or engineering typically, outside of mathematics.

Students work in groups on a single project, or related projects. They are required to present short progress reports every few weeks. The final assessment is based on individual write-ups of projects, progress talks and a final presentation.

A faculty member is instructor of record for the research workshop during semester (Fall and Spring) and typically two or more other faculty, often research advisers, will be present during the workshop meeting times. One of the authors was the first instructor for the research workshops in Spring 2009 and has played a major role in their format and structure since then. Faculty who are not instructors of record do not receive not compensation during Fall and Spring. During the Summer session, faculty are allocated specific weeks in which they run the workshop and they are compensated at their regular salary levels during those weeks.

3.1.1 A marketplace of ideas

The format of the seminar is that of a seminar/workshop in which students regularly present accounts of ongoing research in presentations normally lasting about 10-15 minutes. This length of presentation was chosen as being representative of times allotted to talks at conferences, so giving students valuable practice in concise and informative presentations. The in-class presentations are commonly followed by questions and suggestions from other student and attendant faculty. It is during these presentations that students learn to appreciate the interdisciplinary nature of computational mathematics and scientific computation, commonly offering suggestions for new lines of investigation, new computational techniques, or new ideas for consideration. The cohesion of the cohort is built in part by the regular interchange of results, approaches, and ideas. Additionally, students commonly work on their research projects during the computational seminar time, and will regularly walk around and join with other students to see what they are doing, and offer help and suggestions. The atmosphere of the seminar is best described as a market place of ideas and techniques where faculty present work to build an atmosphere of interest and respect across widely differing computational mathematics projects. Seeking common ground, and finding computational techniques and analyses that extend across these different applications has been a major goal of the seminar, and students quickly buy into this interdisciplinary approach.

3.2 Recruitment

Recruitment of students into the research workshop was initially through recommendations from instructors in other courses, especially early calculus courses. A group of three faculty - the PI and a Co-PI for the NSF grant, as well as the Director of Consulting & Data Management for the project – formed the admissions committee. When an instructor identified a promising student in their class they would talk with one or more members of the admissions committee who would most commonly interview the recommend student and ascertain their interests as well as existing or likely research advisers. When the program had been running for some semesters it was not uncommon for students to hear about it by word of mouth and come looking for a member of the admissions committee to see if the experience was right for them. Considerable effort was put into recruiting under-represented minorities in computational science, particularly women and African-American students.

3.3 Building a Cohort

The CSUMS program was designed to maximize the opportunities for peer support and collaboration. To this end, our students present their work to each other and become involved in each other's projects. The reliance on MATLAB, Mathematica, Python, and Latex requires our students to work together and collaborate with each other, and the relaxed atmosphere in the computer classroom encourages the students to talk, ask, answer, and generally communicate well. Some of our students have done their best work in pairs. For example in the 2009 Summer workshop, two students worked together on coordinate metrology, two on collocation methods, aliasing, and the Runge phenomenon, with another student joining them for parts that overlapped with her research interests in wavelets and Fourier analysis. Two students formed a strong team, centered on their interest in cryptography, and presented a joint paper at the SIAM annual meeting. In addition, another student and a graduate student spent a lot of time on the implementation of MPI and parallel Python on the computers in the room. We consistently try to match up students to work on the same or similar topics, but this does not always match their interests and research style. Students talk to each other, learn from each other, challenge and advise, write summaries and code together and debug each other's work. It is difficult to describe how we measure success in forming a cohesive cohort. One indication is the fact that our students decide to take the same sections of courses, they prefer to stick together and they discuss their projects outside of the CSUMS environment. Another indicator of success is the atmosphere in the computer room. It is a dynamic, exciting, collaborative atmosphere, which is fun to be in. The faculty like the atmosphere so much that they often hang out there and have impromptu meetings and talks.

3.4 Enrolment Numbers

The National Science Foundation supported research training workshops in computational science began Spring 2009 and will continue under this funding model until Summer 2013. To date, 70 students have participated in these workshops, several of them for 3 semesters, and a few returning, unpaid, for another semester. Table 1, below, lists the enrolment numbers from Spring 2009 through Fall 2012.

Table 1. Research training workshop enrolments

Semester	# students	Semester	# students
Spring 2009	15	Summer 2010	11
Summer 2009	12	Fall 2010	11
Fall 2009	18	Spring 2011	16
Spring 2010	8	Summer 2011	11
Fall 2011	12	Fall 2012	8
Spring 2012	9	Spring 2013	16
Summer 2012	8	Summer 2013	9

3.5 Conference Participation

Funding from the NSF has meant that funded students are able to travel, at no cost to themselves, to regional and national conferences and to participate in those conferences as active participants, presenting posters on their research, or delivering talks or refereed papers, or as interested observers.

Each April students are strongly encouraged to present posters or talks at both the Massachusetts Statewide Undergraduate Research Conference at the University of Massachusetts Amherst, and at the University of Massachusetts Dartmouth Sigma Xi Exhibition. Additionally, students participating in the Summer research workshops are encouraged to attend and, where possible, participate through poster presentations or talks in the annual conference of the Society for Industrial and Applied Mathematics (SIAM). Students were also strongly encouraged to participate in local, national and international meetings and conferences. Some details are shown below in Table 2.

Table 2. Number of students attending Summer conferences

Year	Summer Conference	#students
2009	5th MIT Conference on Computational Fluid and Solid Mechanics	12
	SIAM Annual Meeting, Denver, CO	4 + 1 grad
2010	Northeast Section of the Mathematical Association of America, Salve Regina University, Newport, RI,	8 + 1 grad
2010	SIAM Annual Meeting, Pittsburgh, PA,	9 + 1 grad
2011	NSF-CBMS Radial Basis Function Conference, UMass Dartmouth, Fall River, MA	11
	7th International Congress on Industrial and Applied Mathematics, Vancouver, BC, Canada	9
2012	SIAM Annual Meeting, Minneapolis, MN	10
2013	SIAM Annual Meeting, San Diego, California, USA.	9

3.6 The Summer Experience

The Summer research workshops were more intensive than those during Fall and Spring semesters. Students meet 12:00 PM – 5:00 PM Monday-Thursday for 10 weeks during the Summer. This intensive experience has proved most helpful in introducing students to the research experience and mentoring them in their first steps in research. The semester classes were held Tuesdays and Thursdays for 75 minutes per class, for a total of 15 weeks. Not only were the students in Summer getting 5 times the in-class research exposure, they were also getting it in more concentrated form. As a result, we found that students who began research experiences in computational mathematics in the Summer generally stuck with their research projects better than students who began in the Fall or Spring semesters.

3.7 Effect on Students

The question of how an intensive and sustained research experience for undergraduates affects individual students has been tackled in several studies. Kardash [1] reports that an undergraduate research experience can enhance skills differentially; Seymour *et al.* [2] report, in a study of undergraduate science research, that an overwhelming majority of participants experienced positive gains, including areas such as: "thinking and working like a scientist; clarification/confirmation

of career plans (including graduate school); and enhanced career/graduate school preparation (9%)". Hunter *et al.* [3] found that in relation to undergraduate scientific research, faculty focused on gains related to apprenticeship in scientific research while students focused more on skills development.

3.7.1 Surveys

A Likert attitude survey was administered to participating students toward the end of the spring 2009 semester. Significant are the questions with a high (>1) or low (<-1) z-score. We infer from answers to questions with a high or low z-score, that students, on average:

- liked having external speakers;
- found it helpful to be with a group of students from different backgrounds;
- learned a lot about their chose research topics by being part of the CSUMS cohort;
- found the faculty who attended the seminar to be helpful and appreciated the style of interaction;
- learned a lot about computation and programming;
- felt comfortable approaching faculty to advise them;
- found seminar presentations by students to be important;
- agreed with the emphasis on MATLAB;
- did not want student talks restricted to 10 minutes;
- felt that students with less than high GPAs should also be recruited to the program;
- were comfortable with giving external presentations;
- did not find presenting stressful;
- appreciated the diversity of interests of student participants.

This survey provides evidence that the cohort structure of the program is a beneficial feature.

A 2010 survey of students indicated that students rate giving in-class presentations positively, scoring 4.35 on a 5 point scale, ranging from Very Negative to Very Positive, when asked: "Please rate your overall experience of talking in class. My overall experience of giving in-class talks is that it has been, for me personally: [Very Negative through Very Positive] "

Comments included:

- This greatly improved my confidence in public speaking.
- I have gained valuable experience for speaking in class. I have learned from the CSUMS group on how to better my presentations.
- Great feedback. Excellent direction from Professors and Students alike.
- Personally, I get very nervous giving talks, but with each presentation it does seem easier, and when you know your research well, talks can be a very good way of getting feedback. Talks also make you rethink and understand your work better, in order to allow the audience to understand.
- Although nerve-racking at first, class presentations helped make in the long run. It improved my public speaking skills, but also when explaining the material it really let me know what I needed to re visit to get a better understanding. Plus it was a good way to see what everyone else was working on, and learn some new things.
- For me, all the presentations and talk helped me to improve my presentation skill for my later profession. Feedback is always very helpful and a lot of the questions asked really help me understand my projects even better.

3.8 Productive Outcomes

A significant effect on participating students has been a realization that research, unlike course-work, is often messy, with numerous backtracks and changes of direction. Students have come to realize that their professors and mentors do not always know everything and that learning through research is a partnership. Faculty members, for their part, have been willing to trust students, even those with relatively modest achievements in mathematics, to find their way through a research problem with advice and mentoring. A very telling example of this is a young man who worked on fitting normal inverse Gaussian distributions to distributions of percentage obesity by U.S. county for the years 2004-2009. His mathematics achievements prior to this were modest. Yet, motivated by the research topic he chose, and mentored and assisted by two faculty members, he learned how to practically implement maximum likelihood methods for complicated parametric distributions, and to use those to say something statistically important about the data. With the two faculty members, this student is in the process of writing up his research for publication. Other, more mathematically talented students have written professional research papers with their faculty mentors, and others have been stimulated to seek summer research experiences elsewhere, following their CSUMS-RESCUE experiences. Some, who went on to quantitative research environments, have written to us to say how important were the technical skills they learned during their time in the computational science research workshops. These skills included programming skills – MATLAB and Python particularly – data analysis skills and statistics more generally, linear algebra, writing in LaTeX and Beamer, and presentation skills.

3.9 Less Productive Outcomes

Not all student experiences in the CSUMS-RESCUE program were as positive as the faculty would have liked them to be. Some of the issues that arose over the period the program has been funded by NSF include:

- Students who, despite being paid, did not engage with their research topics. Such students were more likely to have been admitted to the program as part of a group, and agreed to the project as part of that group, without being fully motivated to carry out research.
- Students who consistently did not listen carefully to other student presentations, either working on a computer or idly staring out a window.
- Some highly mathematically talented students, who see their mathematical career in teaching, are not amenable to thinking computationally about mathematical problems and generally do not like programming.
- Students who, despite heroic efforts on the part of participating faculty, just were not able to show the necessary technical ability, skills, or attitude to carry out computational science research projects.

This last point has several aspects to it, and requires further study and elucidation, in our view. While all faculty are delighted to find exceptionally talented students who soar when exposed to a research environment, we have an educational mission to productively educate all students, and a belief that undergraduate research can be empowering for a majority of students, and can and should be integrated into the undergraduate curriculum, in line with a 1989 NSF report [4].

Additionally, one of us (Davis) tried unsuccessfully for several summers to run computational science training workshops for local high school teachers. The work of Jacobs *et al* [5] has recently come to our attention, and the programs described there may be helpful in this regard.

4. EFFECT OF THE COMPUTATIONAL SCIENCE TRAINING GRANT ON THE DEPARTMENT AND THE UNIVERSITY

4.1 Effect on the Department

4.1.1 New Faculty

The NSF computational science training grant played a significant role in focusing the entire Department on computational mathematics and computational science. In the Fall of 2008 two new appointments were made in numerical analysis at the Assistant Professor level. This brought to three the number of active researchers in numerical analysis. Since then new tenure-track faculty appointments have all been in the area of computational mathematics, with 3 additional faculty appointed as of 2012. Additionally, another faculty member changed her research from commutative algebra to computational applied mathematics, and obtained promotion to Full Professor on the basis of her new research.

4.1.2 External Review

An AQAD external review of the Department of Mathematics was conducted in 2009. The reviewers commented on the CSUMS-RESCUE program as follows:

"A most notable, very recent development in this area has been the Research in Scientific Computing in Undergraduate Education (RESCUE) project … It would be hard to overstate the positive impact of this activity. The major concrete outcomes have been a program of undergraduate student research projects and a seminar course in which students present the results of their projects. These provide excellent vehicles for obtaining "hands-on" experience applying modern mathematical and computational methods to real-world problems and making presentations on the results. During our visit, we attended a seminar session during which several students gave presentations about their research projects. We were very impressed by the timely mathematical content and societal relevance of the projects presented, as well as the overall polish of the presentations. In a later meeting with a group of students, most of whom participated in RESCUE, we heard strong words of praise and appreciation for the program. Students liked working on projects that interested them, enjoyed the close working relationships with faculty advisors, and appreciated the funding (up to 10 hours of work per week) provided by the grant to participate in the project. (One student said that the funding allowed her to quit her second part-time job.) We also observed that RESCUE appears to involve many faculty members, including a number who were not originally involved in the RESCUE grant proposal. Indeed, seven faculty members meet every other week to work together on this project. Our summary opinion is that the RESCUE project has become a centerpiece of the department's programs that appeals to both students and faculty and that is helping greatly to add cohesiveness and relevance to the department's programs."

4.1.3 Research Advisers

With only two exceptions, faculty in the Department of Mathematics, have been research advisers to students in the computational science program. As faculty increased in numbers, at least six faculty were actively involved in semester and Summer workshops – advising students, overseeing their work and mentoring them through the research process.

Many students were interested in research topics for which the principal, often sole, adviser was a faculty member in Engineering or Physics. This had an effect of bringing faculty in Engineering, Mathematics and Physics with a computational science interest closer together. Faculty from Engineering Departments and Physics gave computationally oriented talks to undergraduate students. Faculty in Mathematics gave talks on software development to the undergraduate students, to which faculty from other Departments occasionally attended. Gradually, a spirit of cooperation over research advising, more general research projects, and attendance at seminars in other Departments became common. Undergraduate students in the computational science program are expected and encouraged to attend research presentations from speakers from outside the University.

4.1.4 New and Modified Courses

In 2007 one of the authors (Hausknecht) introduced a course on Scientific Programming as an upper level course. For the early versions of the course he used Java, Octave and TEMATH [6]. He learned about Visual Python from an ICTCM talk he attended, and switched to Python for the following reasons:

- The potential of Visual Python to graphically illustrate ideas of computational mathematics;
- Many mathematicians and scientists were using Python as an open source replacement for MATLAB;
- Python can be used for general computing including gluing together programs written in FORTRAN and C;
- Many people find Python easier to learn than other languages.

One of the authors was also motivated by the Department's shift to numerical/applied mathematics, which became more focused with the advent of the CSUMS-RESCUE program.

After trialing for several semesters at the upper level the course on Scientific Computing has now been institutionalized as a sophomore level offering for all mathematics students. The course description includes:

"Calculus-based programming covering conditionals, loops, arrays, file I/O, libraries, data types, and operating system commands. This course provides a project driven introduction to programming using a selection of mathematics programming tools, scripting languages, and traditional languages …"

Calculus 2 is a pre-requisite for the Scientific Programming course, and either Calculus 3, Differential Equations, or Linear Algebra is a co-requisite. The purpose of these pre- and co-requisites is to restrict enrolment in this sophomore level course to students with a strong enough mathematical background, and continuing study in mathematics, to be able to cope with the scientific examples used in the course.

To a large degree this course in Scientific Programming fulfills the mathematics major requirements for a computing course, and is one that is more clearly directed to mathematics majors. Recently the Department of Physics has decided to use this course as a preferred training for students in programming that is more closely aligned to physics majors.

Additionally, several courses including differential equations, mathematical modeling, numerical analysis, and mathematical statistics have been revised to include a computational emphasis.

In Spring 2012 one of us (Heryudono) designed and taught a graduate level course in High Performance computing, using MATLAB as a high level language to access a high performance computer cluster. The experiences of Shiflet & Shiflet [7] indicate that such a course could also be accessible to undergraduates.

4.2 Effect on the University

The CSUMS-RESCUE solidified ties for faculty particularly in Engineering Departments, Mathematics, and Physics, with some input from Biology and Chemistry. Faculty from different Departments talked more about research, including research plans. An atmosphere developed in which collaboration seemed not only possible but genuinely desirable. An issue that had been on the mind of one of the authors (Gottlieb) for some time now seemed eminently possible, and so the Scientific Computing Group [8] was born which led very rapidly to University approval for a Center for Scientific Computing and Visualization Research, with very strong support and input from the Dean of Engineering and Applied Science. A result of this growth is the alignment of the Department of Mathematics with the University strategic plan in which computational mathematics and scientific computing more generally, are a central part of that plan.

5. CONCLUSIONS

An embryonic computational mathematics group was in danger of slipping away without a plan and vision for the future. Faculty worked on the CSUMS-RESCUE training grant which, as the Department's external reviewers stated: "… has become a centerpiece of the department's programs that appeals to both students and faculty and that is helping greatly to add cohesiveness and relevance to the department's programs." That cohesiveness and relevance picked up steam through a focused faculty recruitment plan, the establishment of a Center for Scientific Computing and Visualization Research, a Ph.D. program, establishment of a University Office of Undergraduate Research, developments in Data Science through a proposed BS/MS, and marked improvement in competitive research and training grants.

Research and teaching in the Department are now more integrated than they were four years ago. Students who participate in the research training program are in regular contact with research active faculty and graduate students. The number of students participating in the computational science research program in a given semester is a relatively small percentage of the total number of mathematics majors. However, students can participate in the research program any time from the second semester of their Freshman year (after completing Calculus 1) up to their Senior year, students have a total of 10 semesters (including summers) in which to participate. This means the program has impacted a significant percentage of mathematics majors to date. Teaching mathematics, for almost all tenured and tenure/track faculty has become more focused on computation and on research opportunities for students.

The computational science research training program is institutionalized in the Department. How strong student participation will be once NSF funding is no longer available remains to be seen. Establishment of an Office of Undergraduate Research in the University has lead to a fund-raising effort for undergraduate research in general, and a focus on the importance of undergraduate research in the curriculum for which the CSUMS-RESCUE program is an exemplar. The establishment of a Ph.D. program in Computational Science and Engineering, a Center for Scientific Computing and Visualization Research, and a planned Data Science BS/MS program, have all extended and solidified the opportunities for undergraduate students to become deeply involved in serious scientific research in computational science as part of their normal undergraduate experience.

.

6. ACKNOWLEDGMENTS

This work was supported by National Science Foundation grant DMS- 0802974 "RUI: CSUMS: Research in Scientific Computing in Undergraduate Education (RESCUE)". The views and opinions expressed here are those of the authors and do not reflect or represent the views of the National Science Foundation.

We acknowledge the strong support of Robert Peck, Dean of Engineering and Applied Science at the University of Massachusetts Dartmouth.

7. REFERENCES

[1] Kardash, C.M. 2000. Evaluation of an undergraduate research experience: Perceptions of undergraduate interns and their faculty mentors. *Journal of Educational Psychology*, 92(1), 191-201.

[2] Seymour, E., Hunter, A., Laursen, S.L. and DeAntoni, T. 2004. Establishing the benefits of research experiences for undergraduates in the sciences: First findings from a three-year study. *Science Education*, 88(4), 493–534.

[3] Hunter, A., Laursen, S.L. and Seymour, E. 2007. Becoming a scientist: The role of undergraduate research in students' cognitive, personal, and professional development. *Science Education*, 2007, 91(1), 36–74.

[4] National Science Foundation. 1989. *Report on the National Science Foundation disciplinary workshops on undergraduate education*. Washington, DC.

[5] Jacobs, P. and Houchins, J. 2012. Building a project methodology to provide authentic and appropriate experiences in computational science for middle and high school students. *Journal of Computational Science Education*, 3(1), 11–18.

[6] Hausknecht, A. O. and Kowalczyk, R. E. *TEMATH* (Tools for Exploring Mathematics) http://www.faculty.umassd.edu/adam.hausknecht/temath

[7] Shiflet, A.B. and Shiflet, G.W. 2010. Testing the waters with undergraduates (If you lead students to HPC, they will drink). *Journal Of Computational Science Education*, 1(1), 33-37.

[8] UMass Dartmouth Scientific Computing Group. http://umassdcomputing.org

STUDENT PAPER: Solving the Many-Body Polarization Problem on GPUs: Application to MOFs

Brant Tudor
University of South Florida-Tampa
3720 Spectrum Blvd, IDRB 210
Tampa, FL 33620 USA
btudor@mail.usf.edu

Brian Space
University of South Florida-Tampa
3720 Spectrum Blvd, IDRB 210A
Tampa, FL 33620 USA
bspace@mail.usf.edu

ABSTRACT
Massively Parallel Monte Carlo, an in-house computer code available at http://code.google.com/p/mpmc/, has been successfully utilized to simulate interactions between gas phase sorbates and various metal-organic materials. In this regard, calculations involving polarizability were found to be critical, and computationally expensive. Although GPGPU routines have increased the speed of these calculations immensely, in its original state, the program was only able to leverage a GPU's power on small systems. In order to study larger and evermore complex systems, the program model was modified such that limitations related to system size were relaxed while performance was either increased or maintained. In this project, parallel programming techniques learned from the Blue Waters Undergraduate Petascale Education Program were employed to increase the efficiency and expand the utility of this code.

General Terms
Algorithms, Design

Keywords
Blue Waters Undergraduate Petascale Education Program, CUDA, GPGPU, MOF, Parallel Programming, Polarization

1. INTRODUCTION
Metal-Organic Frameworks (MOFs) are highly porous, crystalline materials characterized by inorganic clusters, or nodes, connected via organic linkers. The linking molecules are roughly linear and force a relatively high level of space between the inorganic nodes. Consequently, these materials are remarkable in their high surface areas, which suggest great opportunities for applications such as gas storage via physisorptive processes. The ability to selectively control pore size, polarity and placement of functional groups on the linkers provides further opportunity for the engineering of materials suited for specific separations or catalytic activity. In order to rationally design such materials, it is desirable to understand how they work on a molecular level. For example, it would be useful to know how exactly how and why each node, linker or functional group's place within the MOF improves or retards the process of interest. Additionally, the identification of non-existent MOFs with useful properties, or the identification of useful, overlooked properties on existent MOFs, is another widely held aim.

To that end, accurate, efficient simulation of MOF materials is an area of active research. A program developed in-house, Massively Parallel Monte Carlo (MPMC), has demonstrated its effectiveness in MOF-centric and related simulations [1-3]. This program has been successfully employed to generate sorption isotherms for MOFs with high fidelity to experiment [1]. Crucial to the accuracy of such isotherms is a careful accounting of the polarization energy of the MOF, and, unfortunately, this task has proven to be a computational bottleneck [4]. Early versions of MPMC had a limited ability to utilize GPGPUs to perform these calculations. Although a significant performance boost was realized, the system size was constrained by the amount of shared memory on the card, effectively limiting the simulations to approximately 2000 atoms on the available hardware (a number only suitable for simulation of smaller MOF systems).

2. BACKGROUND
Polarization calculations in MPMC are conducted using the Thole-Applequist model [5, 6]. This model assigns each atomic site a point dipole whose interactions with all the other dipoles of the system are dictated by many-body polarization equations. Using a set of training molecules, a 3x3 polarizability tensor is calculated for each site. Then, in a static electric field, each dipole, $\vec{\mu}$, is thus represented by the product of the calculated polarizability tensor, α, and the field vector at that point, $\vec{E}^{stat}$:

$$\vec{\mu} = \alpha \vec{E}^{stat} \quad \textbf{(1)}$$

In this model, the dipole for a molecule is then treated as a collection of *N* atomic-point dipoles, which are summed to give the net dipole for the set [4]:

$$\vec{\mu}_{mol} = \sum_{i}^{N} \vec{\mu}_i = \sum_{i}^{N} \alpha_i \vec{E}_i^{stat} \quad \textbf{(2)}$$

Here, $\vec{\mu}_i$ is the dipole for an individual site, α_i is the polarizability tensor for the site, and $\vec{E}_i^{stat}$ is the electrostatic field vector at that point, for each site, *i*, in the molecule. The Thole-Applequist system is then treated as a collection of *N* dipoles and a dipole field tensor, $T_{ij}^{\alpha\beta}$. The elements of *T* are the complete set of tensors describing every induced dipole-dipole interaction in the system [4]. The product of the dipole field tensor, *T*, and a system dipole results in the many-body induced-dipole contribution to the electric field, $\vec{E}^{ind}$, at the dipole site. The dipole field tensor was designed to contain the entire induction contribution, allowing the assignment of a scalar point polarizability, α° for each site, instead of the polarizability tensor [4]:

$$\alpha_i \vec{E}_i^{stat} = \alpha_i^{\circ}\left(\vec{E}_i^{stat} + \vec{E}_i^{ind}\right) \quad \textbf{(3)}$$

$$= \alpha_i^{\circ}\left(\vec{E}_i^{stat} - T_{ij}^{\alpha\beta}\vec{\mu}_j\right) \quad \textbf{(4)}$$

If $\vec{\boldsymbol{\mu}}$ is treated as a vector, each entry of which is one of the system dipoles (each of those a vector), equation **(5)** is the result. A similar "super vector" is formed by treating vectors of the static electric field (at a point in space corresponding to each of the dipoles) in an identical fashion, the result of which is equation **(6)**.

$$\vec{\boldsymbol{\mu}} = \begin{pmatrix} \vec{\mu}_1 \\ \vec{\mu}_2 \\ \vdots \\ \vec{\mu}_N \end{pmatrix} \quad \textbf{(5)}$$

$$\vec{\boldsymbol{E}}^{stat} = \begin{pmatrix} \vec{E}_1^{stat} \\ \vec{E}_2^{stat} \\ \vdots \\ \vec{E}_N^{stat} \end{pmatrix} \quad \textbf{(6)}$$

Additionally, if matrices A and B are defined as

$$A = \left[(\alpha^{\circ})^{-1} + T_{ij}^{\alpha\beta}\right] \quad \textbf{(7)}$$

$$B = A^{-1} \quad \textbf{(8)}$$

the problem is reduced to two compact matrix equations, **(9)** and **(10)**. Matrix *A* is thus constructed such that each element is the 3x3 matrix T_{ij}. Each element of matrix *B* is also a 3x3 matrix—the site polarizability tensor characterizing each site's response to an electric field [4].

$$A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{E}}^{stat} \quad \textbf{(9)}$$

$$\vec{\boldsymbol{\mu}} = B\vec{\boldsymbol{E}}^{stat} \quad \textbf{(10)}$$

The system dipoles can therefore be found by inverting matrix *A* (giving *B*) and solving equation **(10)** directly. However, the size of matrices required to model typical MOF systems renders the computation required for matrix inversion impractical. MPMC solves these equations by guessing at the value of each point dipole and solving equation **(9)** iteratively.

3. MPMC

3.1 Limitations of the Initial Solution

MPMC typically solves for the system dipoles iteratively [7]. The initial guess for each dipole is simply the product of the scalar point polarizability and the electrostatic field vector at that point. Each dipole is considered sequentially, and is marginally corrected according to the induced contribution calculated using all the other dipoles in the system. This process is repeated for each dipole (thus concluding a single iteration), and the whole process is then repeated for the entire system until convergence to within a specified tolerance is realized. MPMC also has the ability to solve this problem through matrix inversion, but, as previously mentioned, this method is only viable for small systems.

Additionally, the original version of MPMC included support for finding the system dipoles using a General Purpose Graphics Processing Unit (GPGPU) device. This algorithm performed the iterative process previously described with only a few key differences. First, each step of the calculation updated every dipole in the system, whereas the serial algorithm incorporated the Gauss-Seidel numerical iterative technique. In this method, newly calculated dipole data replaces old dipole data as soon as it becomes available. The new values are then used in calculating all the remaining dipoles in the system. This technique can significantly decrease convergence times, but since, in the parallel algorithm, all the newly calculated dipoles become available simultaneously, the Gauss-Seidel technique was not implemented.

A test for convergence of the GPGPU polarization calculation was not implemented in the original version of MPMC. Hence, the computation would run for a preset number of iterations and results were delivered without any way of estimating their accuracy.

Finally, simulations utilizing the GPGPU device were limited to 2048 atoms due to the manner in which MPMC employed the GPU's shared memory system. This constraint renders the GPGPU algorithm useful only in simulations of relatively small system size. A MOF simulator should ideally be able to handle system sizes of 10,000+ atoms in order to be useful for several MOFs of current and future interest to investigators.

3.2 Updated Program Model

Several changes to improve and expand the functionality of MPMC were realized.

3.2.1 Maximum System Size Expansion

The 2048 atom cap imposed on simulations was the first limitation addressed during the course of this project. In the updated program model, each GPU thread was assigned a single system dipole. Each thread calculates its dipole's interaction with every other dipole in the system, and sums these interactions to arrive at the dipole vector to be used in the next iteration. Since

every thread needs access to the vector data of every other dipole, it only makes sense to load the dipole information into shared memory so that each thread in the block can access it. This precludes the need for each of these threads to access the data individually from global memory (a relatively time consuming process to be avoided when possible) [8]. However, since shared memory is fairly limited, it is impossible to fit all the dipole data in this memory system simultaneously (for moderate to large-sized systems). This situation is amenable to a tiled model of data handling such that the complete set of dipole information resides in global memory and is moved in and out of shared memory as needed, one block at a time **(FIGURE 1)**. Organizing the data in this manner shifts the limitation of system size from shared memory to one of global memory and/or maximum grid size. Obviously, the global memory of the GPGPU device must be large enough to hold dipole data for the entire system. However, since each thread is responsible for a dipole and each thread block executes a limited number of threads, the maximum grid size (which dictates the total number of blocks) is ultimately responsible for determining the maximum number of threads [8], and therefore the maximum number of dipoles (i.e. atoms). Fortunately, on current hardware, the system sizes imposed by these limitations number in the millions of atoms, thus transforming MPMC's prohibiting considerations from those of system size to one of computational duration.

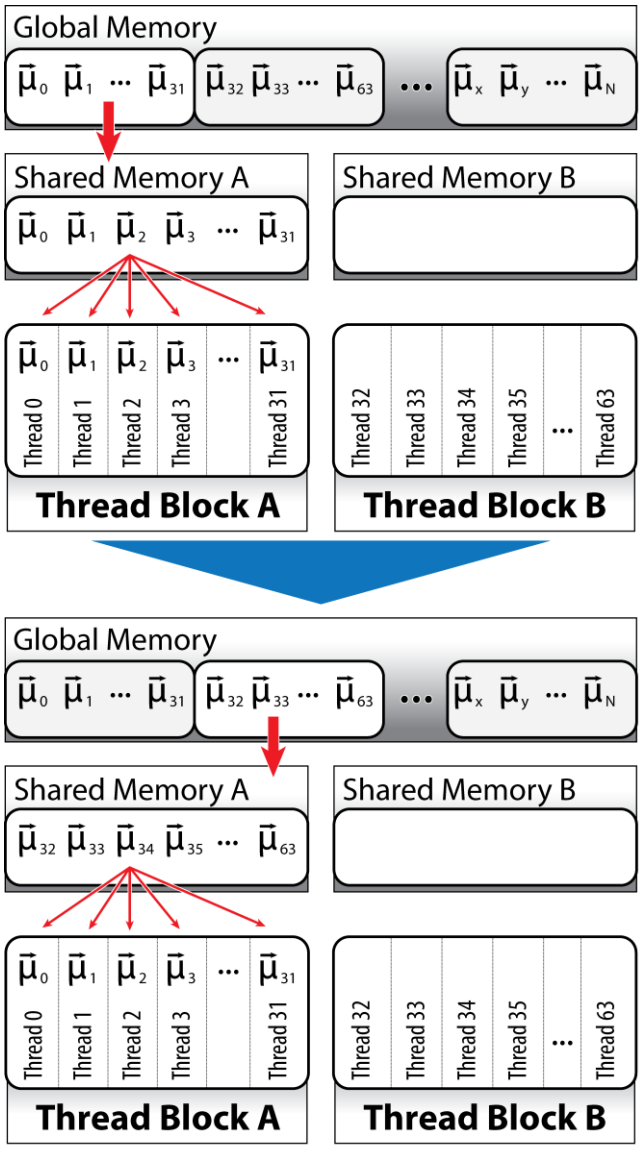

FIGURE 1: In this example, Thread Block A (TBA) is executing, while B sits idle. In the top panel, TBA pulls global vector data for the first block of dipoles. Once all threads have processed this information, it pulls data for the second block, as illustrated in the bottom panel. This continues until TBA has processed all dipole-dipole interactions for which it is responsible. Other thread blocks will do likewise (either concurrently or sequentially) until all dipole-dipole interactions in the system have been calculated.

3.2.2 Gauss-Seidel in Parallel

The original GPGPU algorithm did not attempt to implement the Gauss-Seidel iterative method of using newly calculated dipole information in the calculations for later dipoles. From outside the GPU kernel, all the dipoles appear to be updated simultaneously, so a treatment of this nature simply is not possible. However, from inside the kernel, once a thread block has completed, it is possible for each thread to overwrite its value in global memory with its newly calculated value **(FIGURE 2)**. This treatment will allow any subsequent calculations to use the latest available information for their own computations. This technique updates a block of dipoles at a time, and as such effects a coarse-grained version of the Gauss-Seidel method. Typically, several thread blocks will be executing concurrently and these blocks will not be able to take advantage each other's updates, thus it is expected that this modification will only become significant on larger system sizes where only a small portion of the total number of the required thread blocks can run concurrently.

3.2.3 Convergence Verification

Prior to this work, MPMC set a fixed number of iterations for the GPGPU algorithm and the level of convergence obtained after this number of iterations was what any dependent calculations were forced to use. After extensive testing, it became apparent that, in many cases, the set number of iterations was sufficient for a high level of convergence. However, in some cases it was not. Worse, the program was unable to tell if a set of dipoles converged, so the user received no warning that their calculation may be suspect.

From inside the kernel, before each thread updates its data in global memory (for Gauss-Seidel), modifications were made such that each thread now copies its original dipole data into a local register. The difference between the old dipole data and the newly calculated dipole is squared and stored in an output array which can then be examined by the function that launched the kernel. Outside the kernel, in the calling function, the transfer of the squared-difference data from the GPGPU device to the host machine can take a significant amount of time compared to a single iteration. In some cases, the transfer duration can take *longer* than a single iteration, more than doubling the length of the total calculation. To mitigate this effect, the squared dipole differences are only downloaded and examined after every tenth iteration.

3.2.4 Energy Calculations in Parallel

The Monte Carlo portion of MPMC aims to identify low-energy system configurations. As such, the purpose of calculating the

system dipoles is to quantify an energy contribution from polarization effects. The time required to calculate this energy tends to vary widely. Kinetic and coulombic energies are also considered and the combined time required for these calculations, depending on the duration of the polarization energy, can be mildly to highly significant by comparison. Finally, MPMC can calculate an energy contribution due to van der Waals effects. This computation relies on matrix diagonalization and, when utilized, invariably takes the longest of any of the calculations. Using the Open Multi-Processing API (OpenMP), MPMC is now able to split into three concurrent threads of execution, one of which is responsible for both the kinetic and coulombic energy calculation, another of which is responsible for the polarization energy calculation, and the last of which is responsible for calculating the van der Waals energy contribution.

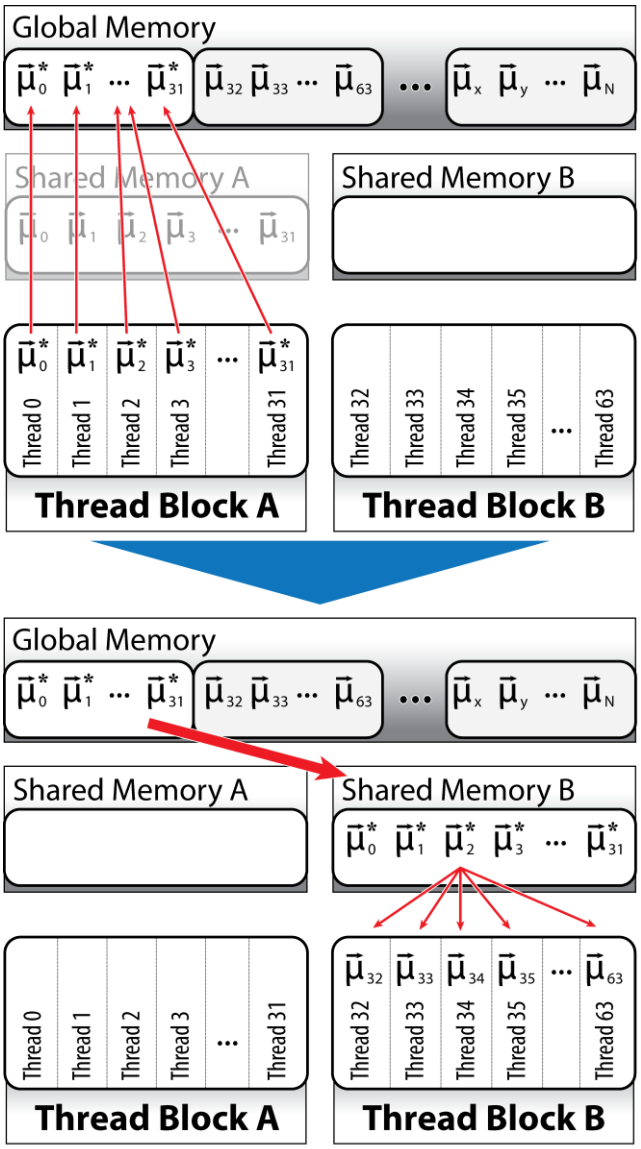

FIGURE 2: A block-wise, parallel Gauss-Seidel iterative method. In the top panel, Thread Block A (TBA) has finished executing and immediately updates the global data with the newly calculated values for its dipoles (updated values are denoted with an asterisk). In the lower panel, when Thread Block B (or any subsequent thread block) executes, it is able to leverage the latest calculated values for TBA's dipole data in its own calculations, speeding convergence.

3.2.5 Van der Waals Calculations Using MAGMA

The final modification made to MPMC was to utilize the Matrix Algebra on GPU and Multi-core Architectures library (MAGMA) in order to compute the van der Waals energy contribution. The original routine calls for a matrix diagonalization via the LAPACK routine `dsyev_()`. It was a simple matter to construct an alternate routine, to be used in the event that a GPGPU device was detected. The two routines were practically identical in all respects except that the new one makes a call to MAGMA's `magma_dsyevd()` instead of to the equivalent LAPACK function.

4. RESULTS

The updated version of GPGPU portion of MPMC is able to reproduce the results of the original with perfect fidelity for system sizes less than or equal to 2048 atoms, in approximately the same amount of time. For larger systems, no direct comparison can be made since the older version is unable to produce a result, although the computation is performed six to eight times faster on the GPU than the CPU. Comparing GPU results against data obtained through matrix inversion, presumed exact, reveals that calculations on typical systems are within five percent error.

Performance increases due to the multi-threaded, OpenMP handling of the energy calculations, though present, is difficult to quantify. The combined calculation time for the kinetic and coulombic contributions represents roughly 10 to 50 percent of the total calculation time, and this figure varies widely from iteration to iteration. Effectively, the total calculation time is now reduced to the duration of whichever calculation takes the longest (coulombic/kinetic, polarization, or van der Waals), plus a small penalty for the overhead required to establish the threads. On test systems, the net speedup of the multithreaded treatment was typically around 20 percent.

The use of the MAGMA routine in the calculation of the van der Waals energies is able to exactly reproduce the LAPACK result. However, the calculations are completed in approximately half the time.

5. FUTURE WORK

The accuracy of the GPGPU polarization calculation is lower than ideal, on the order of three percent error. Different techniques are being tried in order to increase the accuracy of these results, as well as to decrease convergence times. Additionally, the version of MPMC under discussion was designed to simulate a crystalline material and a single species of sorbate. Currently, efforts are underway to modify the program such that it can simulate multiple sorbate species simultaneously introduced into the material.

6. REFLECTIONS

The summer portion of the Blue Waters Undergraduate Petascale Education Program (BW-UPEP) provided training and instruction at the Urbana-Champagne campus of the University of Illinois. During this program, various technologies and techniques for scientific coding on parallel and supercomputer architectures were

discussed and elucidated. Of particular interest to this project was the training on GPGPU programming through NVIDIA Compute Unified Device Architecture (CUDA) as well as the Open Multi-processing API (OpenMP) maintained by the OpenMP Architecture Review Board. The workshop introduced students to various algorithmic models, concepts and issues that were particularly useful to the current project, such as deconstruction of large repetitious problems into loosely coupled blocks appropriate for efficient handling by GPGPU devices, concurrent processing of dissimilar tasks through multi-threading, and, perhaps most importantly, how to leverage both techniques within a single program. Resources for learning any one of the technologies abound, but an area where the program excelled was instruction on how to effectively harness all these technologies to work together within a single project.

Through the work started during the BW-UPEP program, I was able to foster a deep understanding of the architecture sitting underneath the hood of various high performance computing systems. Whereas before, I had only superficial experience with supercomputers, I currently develop scientific software and perform research computation on my own university's local research computing cluster, as well as on many of the computing systems made available through the NSF's Extreme Science and Engineering Discovery Environment (XSEDE) project. Speaking from personal experience, I believe undergraduates who have an interest in scientific computing stand to gain a considerable amount of confidence, experience and expertise by attending such a program as the BW-UPEP. The abundant knowledge and support available during the development of various pedagogical codes, as well as the guidance received regarding submission of these jobs to actual work environments (research computing clusters of universities with ties to the program), made it much easier to "leave the nest" and create and submit my own computational jobs to world-class research computing facilities throughout the academic world.

I am currently in the early stages of my Doctoral program in theoretical and computational chemistry at the University of South Florida, and the skills and knowledge acquired through the BW-UPEP program have definitely helped to jumpstart my career therein. The time saved by not having to start from scratch in learning the basics of HPC coding (*or* the ins-and-outs of interaction with research computing environments) may have shaved a semester or more off my time in graduate school. In classes oriented around high performance computation and scientific coding, I find that while my peers spend much of their time trying to frame the posed problems in a manner suitable for parallel computation, the practical experience gained through the Blue Waters program often allows me to skip this step and immediately begin to identify opportunities to make the code more efficient in terms of the low-level hardware, e.g. efficient use of cache, shared memory systems, coalesced memory accesses, etc. My association with the BW-UPEP has proven to be an invaluable advantage in this regard and my ardent gratitude toward the program remains steadfast.

7. AKNOWLEDGMENTS

Funding for this work was provided by the National Science Foundation's Office of CyberInfrastructure through the Blue Waters Undergraduate Petascale Education Program.

The authors would like to acknowledge the use of the services provided by Research Computing at the University of South Florida. Simulations and development were performed on the University of South Florida's Research Computing Center, CIRCE, with additional support received from the Space Foundation (Basic and Applied Research).

8. REFERENCES

[1] J. L. Belof, *et al.*, "A Predictive Model of Hydrogen Sorption for Metal−Organic Materials," *The Journal of Physical Chemistry C*, vol. 113, pp. 9316-9320, 2009.

[2] *An Accurate and Transferable Intermolecular Diatomic Hydrogen Potential for Condensed Phase Simulation*, 4, 2008.

[3] J. L. Belof, *et al.*, "On the Mechanism of Hydrogen Storage in a Metal−Organic Framework Material," *Journal of the American Chemical Society*, vol. 129, pp. 15202-15210, 2007.

[4] J. L. Belof, *Theory and simulation of metal-organic materials and biomolecules*. Tampa: Theses and Dissertations. Paper 1851. http://scholarcommons.usf.edu/etd/1851, 2009.

[5] J. Applequist, *et al.*, "An Atom Dipole Interaction Model for Molecular Polarizability. Application to Polyatomic Molecules and Determination of Atom Polarizabilities," *Journal of the American Chemical Society*, vol. 94, pp. 2952-2960, 1972.

[6] B. T. Thole, "Molecular polarizabilities calculated with a modified dipole interaction," *Chemical Physics*, vol. 59, pp. 341-350, 1981.

[7] J. L. Belof, "Massively Parallel Monte Carlo (MPMC)," http://code.google.com/p/mpmc/, 2007.

[8] (2012). *NVIDIA CUDA C Programming Guide Version 4.2*.

Parallelization of the Knapsack Problem as an Introductory Experience in Parallel Computing

Michael Crawford †
University of Mary Washington
1301 College Avenue
Fredericksburg, VA 22401
+1 518-338-5739
mcrawfor@umw.edu

David Toth
University of Mary Washington
1301 College Avenue
Fredericksburg, VA 22401
+1 540-654-1693
dtoth@umw.edu

ABSTRACT
As part of a parallel computing course where undergraduate students learned parallel computing techniques and got to run their programs on a supercomputer, one student designed and implemented a sequential algorithm and two versions of a parallel algorithm to solve the knapsack problem. Performance tests of the programs were conducted on the Ranger supercomputer. The performance of the sequential and parallel implementations was compared to determine speedup and efficiency. We observed 82%-86% efficiency for the MPI version and 89% efficiency for the OpenMP version for sufficiently large inputs to the problem. Additionally, we discuss both the student and faculty member's reflections about the experience.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer science education.

General Terms
Experimentation.

Keywords
Parallel computing, education, performance, knapsack problem.

1. INTRODUCTION
The 0-1 knapsack problem is an optimization problem with the goal of selecting items with weights and values in order to maximize the value of the items selected while keeping the total weight of the items below a set value [1]. In contrast to the bounded and unbounded variants of the knapsack problem that allow multiple copies of an item to be placed in the knapsack, in the 0-1 version of the knapsack problem, an item is either put in the knapsack or not [1]. The 0-1 knapsack problem is an NP-complete problem [1].

In an undergraduate parallel computing course, students learned to develop parallel programs with OpenMP and MPI. Because the course's instructor has found that NP-complete problems can help illustrate a number of concepts that can be useful when studying parallel computing, students were required to choose an NP-complete problem to have their programs solve. Students developed their programs using a multi-core server on campus and on their own systems and lab systems using the Bootable Cluster CD software [2]. Once the students had debugged their programs, they were able to run them on the Ranger supercomputer at the Texas Advanced Computing Center (TACC) at the University of Texas at Austin. Ranger, which was just recently decommissioned after this project was completed, contained 3,936 distinct compute nodes with 16 general-purpose CPU cores each, for a total of 62,976 cores [3]. Running on Ranger allowed the students to compare the performance of the sequential version of their programs with the performance of parallel programs with OpenMP using 16 cores and using MPI with 16, 32, and 48 cores.

It is important to keep in mind that we were not attempting to devise a better algorithm than existing ones, but were focused on using the problem as a way of learning to use OpenMP and MPI and conduct some performance testing. Therefore, our results are not the important contribution of this paper. What's important is the learning that this project facilitated and our reflections about it. The first author, Michael Crawford, was a student in the second author's undergraduate course. This paper is written primarily from the student's perspective and describes the student's experience completing the course's final project.

2. RELATED WORK
A significant amount of research has been done on the 0-1 knapsack problem, which has numerous applications in business. Balas and Zemel developed an algorithm, as did Fayard and Plateau, and Martello and Toth [4, 5, 6]. Pisinger also developed algorithms, as have others [7, 8]. More recently, parallel algorithms have been discussed by a number of people. Loots and Smith developed a variation of a branch-and-bound algorithm to solve the problem for large numbers of objects [9]. Chen and Jang also developed parallel algorithms, as have others [10]. Even more recently, Pospichal et. al. have developed a parallel genetic algorithm to solve the 0-1 knapsack problem using GPUs [11].

† Undergraduate student

3. METHODOLOGY

3.1 Sequential Algorithm

Since the goal of the project was to become comfortable with OpenMP and MPI and to conduct some performance testing, rather than to implement an optimal algorithm, we chose to implement a brute force algorithm to solve the problem. The brute force algorithm could be parallelized with OpenMP and MPI with only minor changes and was easier to parallelize than a more complicated algorithm. The algorithm generates each possible permutation using the C++ standard library's next_permutation function. As each permutation is generated, the items in the permutation are placed in the knapsack from left to right, as long as there is space. Once an item in a given permutation is encountered that does not fit in the knapsack, no further work is done with that permutation and it receives the score of the items that did fit in the knapsack. If the score of the permutation is the largest one so far, the permutation and its score are saved as the current maximum. Then the next permutation is generated and scored until all permutations have been generated and scored and the permutation with the maximum score has been determined. To ensure reliable and reproducible results, the item sets were pre-generated and stored in a text file which was used by all three versions of the program. This allowed for easy testing of the sequential algorithm to prove it finds the optimal set of items, and subsequently easier verification of both parallel versions.

3.2 Parallel Algorithm

We created a parallel version of the program with OpenMP and then another one with MPI. Like the sequential algorithm, both the OpenMP and MPI algorithms were brute force. The parallel algorithms divided up the different permutations to test amongst the available CPU cores, with each core running a thread in the OpenMP version and each core running an MPI process in the MPI version. Each thread or MPI process tested an equal share of the permutations. Our sequential brute force, lexicographic permutation-dependent algorithm is nontrivial to parallelize, as each permutation is determined by the permutation before it. In order to divide the set of distinct orderings into a subset for each thread or MPI process, the factorial number system, or factoradic, was used [12, 13].

Factoradic allows one to calculate a specific lexicographic permutation of a set of numbers without having to generate each permutation between the first permutation and the one you are trying to generate. Thus, using the OpenMP thread ID number or the MPI rank and how many permutations per thread or MPI process will be computed, it is possible to determine the lexicographic permutation that any given thread or MPI process begins on.

The algorithm for OpenMP used factoradic to determine the starting permutation of each thread. Using the thread ID number and how many permutations per thread needed to be computed, each thread determined its own starting permutation and worked until it finished its assigned chunk of permutations to compute. Each thread kept its own best combination. After calculating the best combination from all of its permutations, in a critical section, each thread would compare its maximum value to the current global maximum value and update the global maximum if its value was greater than the current global maximum. The algorithm for MPI was identical to the OpenMP one, with the exception that at the end of their calculations, each MPI process sent its best permutation and corresponding score to the master process, which, starting with its own best as the default compared all of them to find the most optimal knapsack items. At the end of the calculation, the master node printed the result.

4. TECHNICAL RESULTS

When there were fewer than 11 items that could be selected to put in the knapsack, the sequential version of the program took less than 1 second. While one might think there is nothing to be learned from running a parallel version of the program for that few items, we observed that the MPI version of the program using 48 cores took 1 second which illustrates the overhead associated with it and shows that for small input sizes, the parallelism can actually cause slowdown. With 11 or more items, the sequential version of the program began to take larger quantities of time and took almost 62 minutes to complete for 14 items. In contrast to that, the MPI version of the program took less than two minutes when running on 48 cores on the same computer. The full set of running times for the sequential, OpenMP, and MPI versions is shown in Table 1.

Because the times the programs took to run were measured in seconds, it wasn't possible to accurately calculate the speedup and efficiency when there were fewer than 12 items that could be put in the knapsack. For 12 items, we could not calculate the speedup or efficiency of the MPI program that used 48 cores. We were able to put a very coarse lower bound on the speedup and efficiency. The speedups and efficiencies for the OpenMP version and the MPI version of the program are shown in Table 2 and Table 3 respectively. For 13 and 14 items, the speedup observed with OpenMP is not quite as close to proportional to the number of cores used as one might expect for an embarrassingly parallel implementation (0.94 and 0.89 for 13 and 14 items, respectively). This could have been a result of the algorithm used, which could result in a number of computations being short-circuited by some threads while other threads might have fewer computations that are short circuited. It is also possible that computations that are short circuited by some threads run further into their process than the ones short circuited by other threads. This could result in some threads needing to do more calculations than other threads, thus resulting in a load that is not balanced perfectly, which might account for some of why the speedup was not proportional to the number of cores.

In terms of speedup, the MPI version of the program when run with 48 cores saw a speedup of almost 40 for both 13 and 14 items. With MPI, using 16, 32, and 48 cores split on 1, 2, and 3 nodes respectively, we observed 86%, 85%, and 83% efficiencies with 14 items. The lower efficiency as the number of cores increased may have been due to the additional network traffic needed by using multiple compute nodes or also due to a computation imbalance between the nodes or between the cores on the nodes. We did expect MPI to be slightly less efficient than OpenMP, however, due to the required network communication. More interesting was that for the instance where there was only 1 possible item to put in the knapsack, the MPI run using 48 cores that took longer than the sequential and OpenMP runs and the MPI runs using fewer nodes (and cores).

With a brute force algorithm in parallel, we expected to observe linear speedup. To our surprise, our program did not achieve as close to linear speedup as we expected. However, we believe that the number of objects that could be put in the knapsack was too small to produce a good test, which might have caused this. Increasing the number of items available and the knapsack's capacity might have resulted in closer to linear speedup. This was

limited due to limits on the amount of hours on the supercomputer allocated to each student. We noticed that the speedup and efficiency were better with OpenMP than with MPI, which we believe was due to the extra network communication required for the MPI version of the program.

Table 1. Runtime of the Programs (sec)

Items Available to Put in Knapsack	Sequential Version	OpenMP Version (16 Cores)	MPI Version 1 Node (16 Cores)	MPI Version 2 Nodes (32 Cores)	MPI Version 3 Nodes (48 Cores)
1	0	0	0	0	1
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0
11	2	0	0	0	0
12	23	3	2	1	0
13	316	21	23	12	8
14	3716	260	271	137	93

Table 2. Speedup of the Parallel Versions

Items Available to Put in Knapsack	OpenMP Version Using 16 Cores	MPI Version Using 1 Node (16 Cores)	MPI Version Using 2 Node (32 Cores)	MPI Version Using 3 Nodes (48 Cores)
12	7.67	11.50	23.00	> 23
13	15.05	13.74	26.33	39.50
14	14.29	13.71	27.12	39.96

Table 3. Efficiency of the Parallel Versions

Items Available to Put in Knapsack	OpenMP Version Using 16 Cores	MPI Version Using 1 Node (16 Cores)	MPI Version Using 2 Node (32 Cores)	MPI Version Using 3 Nodes (48 Cores)
12	0.48	0.72	0.72	>0.48
13	0.94	0.86	0.82	0.82
14	0.89	0.86	0.85	0.83

5. FUTURE WORK

There are several ways this work can be extended in the future. The first task is to create an OpenMP/MPI hybrid version of the program and conduct performance testing to see how that compares to the other versions of the program. We note that since Ranger has been decommissioned, we will need to redo the performance testing on a new supercomputer, such as Stampede, which replaced Ranger. The second task that should be done is to port the algorithm to run on CUDA-enabled GPUs and compare the performance of that version of the program to the results from the Stampede performance tests. We note that all the measurements in the future should be done in msec instead of

seconds, so more accurate comparisons can be made with small numbers of items. We would also like to implement different sequential and parallel algorithms such as branch-and-bound and dynamic programming algorithms to solve the same problem so we can see parallel performance comparisons of those algorithms in comparison to the brute force algorithm.

6. REFLECTIONS

6.1 Student Reflections

From a learning perspective, I think that this exercise was incredibly fruitful. Over the last couple of years, I have been exposed to a number of computationally intensive problems within the domains of theoretical computation and modeling and simulation. Knowledge on parallel processing is invaluable for the next generation of both computer scientists and natural scientists.

This exercise introduced me to the challenges and opportunities of high performance computing. It led me to consider the challenges of implementing parallel programs to solve problems in a way more theoretical classes cannot. While I was acquainted with NP-complete problems from my course on the theoretical foundations of computer science, this project gave me the opportunity to truly understand what it means for a problem to be NP-complete. It was enlightening to write an algorithm to "solve" an NP-complete problem; actually watching the runtime of the algorithm grow at a factorial rate with more items is humbling and seeing the limitations of a supercomputer is profound.

In retrospect, there were a number of changes I would make to both how I did the project and the course itself.

First, I wish I had paid far more attention to overhead as I coded my project. My algorithm had the potential to achieve nearly linear speedup, but I only saw roughly 85 percent efficiency. I wonder if my algorithm could have been written in such a way as to decrease the overhead and increase efficiency. In the future iterations of this class, it would be helpful to see a lesson on limiting overhead in HPC projects.

Second, we did not write programs that combined MPI with OpenMP, and I would have liked to have seen the performance of an MPI & OpenMP hybrid version of my algorithm.

Third, we did not talk about the impacts of using different compilers and optimizations during the course, so when I encountered a major performance difference of the same code running on the same hardware when it was compiled using different compilers, I was surprised. Previously, my mental model of compilers had naïvely assumed their equality for practical purposes. A brief exploration of compiler optimizations would be an interesting module for the course.

Fourth, each student was only given 3000 hours of CPU time on the supercomputer to conduct the performance testing. In the future, increasing this amount of time would provide the opportunity to test an OpenMP version of the program with 2, 4, and 8 cores in addition to with 16 cores, which might provide additional interesting results in terms of speedup.

Lastly, in order to learn the basics of parallel computing libraries like MPI and OpenMP, we were encouraged to solve our NP-complete problems using brute force algorithms. However, it would be interesting to see the difference in speedup and efficiency that other types of algorithms to solve the same problem would achieve. For example, what kind of speedup and efficiency would we see using a branch and bound versus a dynamic programming algorithm? While it would be impossible to expect students taking a three credit hour course to test three versions of each algorithm, each using a different parallel platform, it would have been a valuable juxtaposition to see different types of parallel algorithms pitted against each other.

As a student at a small liberal arts university, it was empowering to run code on one of the fastest computers in the world. The experience taught me humility in the face of computational intensity and provided me the tools to think in parallel. It will be a priceless course for students in the future.

6.2 Instructor Reflections

This course project spanned 8 weeks of a 15-week semester, making it one of the larger projects students do in our courses. The idea behind assigning such a large project was to give the students an opportunity to gain experience with not only the technical aspects of the material, but also with other important skills like time management, creating a poster, and giving a talk. The project was supposed to give students the opportunity to demonstrate mastery of the basics of OpenMP and MPI, as well as to perform some performance comparisons and give them experience running a program on a supercomputer. A side goal was to introduce the students to a variety of NP-complete problems.

Although I have taught a parallel computing course twice before, this was the first time I integrated this project and the use of a supercomputer into the course. Because of that, I learned a number of lessons, including that too many students will procrastinate if they are not given enough intermediate deadlines for a big project, as was the case with this project. A number of students were unable to complete the project, only developing an OpenMP implementation and not completing an MPI implementation or the performance analysis portion of the project. In general, the students that did not complete the project did not lack technical ability, but rather, they simply did not start the project until the last couple of weeks of the semester. Next time I teach the course, I will have parts of the projects due every 1-2 weeks.

Each student had to select an NP-complete problem for the project, but I limited the number of students who could choose the same problem to 2. This was done to force the students to produce a variety of posters and talks, so they would not all be the same and to keep things more interesting, as we as to prevent cheating. This caused some students to choose harder problems than they should have chosen. In the future, I will sacrifice the variety of the posters and talks to make the students more successful.

Students were supposed to develop a sequential program to solve their problem for small instances and then run the program to see how large an instance their program could solve in 24 hours. This instance of the problem was then supposed to be used as the baseline for their performance comparisons. Because the instance was supposed to take close to 24 hours to solve, I did not tell the students to do their timing in milliseconds because that granularity should not have been required for the performance comparison. However, some students benchmarked their code on systems other than the supercomputer and that led to them running their code for instances of the problems that were too small for the timing granularity of seconds to be as useful as needed. Because many students didn't run their program on the supercomputer until right before the project was due, they did not have time to test the programs with larger instances if the problem instances they used were too small. In the future, I will make sure that the students do their timing in msec instead of seconds.

The students were told that they each had 3000 hours to use on the supercomputer for the projects. At the end of the semester, a couple students said they wished they could have had more time on the supercomputer. After the projects were completed, those student were told they could use the remaining time from the educational grant that had not been used by others, but that was during finals week and the students didn't get around to using the extra time. Because students were supposed to have a program that ran for almost 24 hours sequentially and Ranger had 16 cores per node, doing that would have used 384 of the students 3000 hours. Since the students were supposed to run the program with OpenMP on one node and MPI on 1, 2, and 3 nodes, if they had no speedup, they would have consumed 3072 hours. I expected they would get significant speedup, but that there would also be problems and students would need to run their programs more than once to complete the project. Therefore, 3000 hours per student seemed reasonable. In the future, I want to rethink that.

6.3 Suggestions For Instructors

Our recommendations to a faculty member who adopts this project for their course are:

1. Have the entire class do the same NP-complete problem or half of the class do one and the other half of the class do another. We recommend the 0-1 knapsack problem and the traveling salesman problem.
2. Have the students who did one problems compare their speedups and efficiencies for their implementations and discuss the reasons for any differences.
3. Have prebuilt implementations of the problem(s) with data set(s) that students can run their algorithm on and compare their algorithm's solutions to the known correct solution.
4. Do not bother with presentations, which took time that might be better spent on other tasks. In particular, teaching the basics of CUDA would be helpful. However, having the students create a poster might still be useful.
5. Ensure that the compiler used on the supercomputer is available to the students on the local computer where they develop their solutions to ensure consistency. Discuss the different optimization levels for the compiler and ensure all the students use the same level.
6. Ensure students conduct timing in msec rather than seconds.
7. Have intermediate deadlines so that students do not get behind and are all ready to run their code on the supercomputer before the end of the course.
8. Request twice as much time on the supercomputer as you think the students will need to conduct the performance comparisons to ensure that even with mistakes, students have adequate time to test their implementations.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation through XSEDE resources with grant ASC120039: "Introducing Computer Science Students to Supercomputing in a Parallel Computing Course" and by the Texas Advanced Computing Center (TACC), where the supercomputer we used was located. We wish to thank XSEDE and TACC for their support.

8. REFERENCES

[1] Knapsack Problems: Algorithms and Computer Implementations, Silvano Martello, Paolo Toth, 1990. J. Wiley & Sons.

[2] BCCD | Bootable Cluster CD. http://bccd.net/.

[3] Texas Advanced Computing Center – Ranger-User-Guide. http://www.tacc.utexas.edu/user-services/user-guides/ranger-user-guide. Updated 10/30/12.

[4] Balas, E. and Zemel, E. An algorithm for large zero-one knapsack problems. *Operations Research*. 28 (1980).

[5] Fayard, D. and Plateau, G. An algorithm for the solution of the 0-1 knapsack problem. *Computing*. 28 (1982), 269-287.

[6] Martello, S. and Toth, P. A new algorithm for the 0-1 knapsack problem. *Management Science*. 34 (1988).

[7] Pisinger, D. A minimal algorithm for the 0-1 Knapsack Problem. *Operations Research*. 45 (1994), 758-767.

[8] Pisinger, D. An expanding-core algorithm for the exact 0-1 Knapsack Problem. *European Journal of Operations Research*. 87 (1993), 175-187.

[9] Loots, W. and Smith, T. H. C. A parallel algorithm for the 0-1 knapsack problem. *International Journal of Parallel Programming*. 21:5 (Oct. 1992), 349-362.

[10] Chen, G. and Jang, J. An improved parallel algorithm for 0/1 knapsack problem. *Parallel Computing*. 18:7 (July, 1992), 811-821. DOI=http://dx.doi.org/10.1016/0167-8191(92)90047-B.

[11] Pospichal, P., Schwarz J., and Jaros, J. Parallel Gneetic Algorithm Solving 0/1 Knapsack Problem Running on the GPU. *Proceedings of the 16th International Conference on Soft Computing MENDEL* (Brno, Czech Republic, June 23-25, 2010), 64–70.

[12] Irene's coding blog: Factorial base numbers and permutations. http://irenes-coding-blog.blogspot.com/2012/07/factorial-base-numbers-and-permutations.html. Updated July 22, 2012.

[13] Factorial number system - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Factorial_number_system.

November 2013
Volume 4 Issue 1

TABLE OF CONTENTS

ISSN 2153-4136 (online)