Minimum Time, Maximum Effect: Introducing Parallel Computing in CS0 and STEM Outreach Activities Using Scratch

Russell Feldhausen
Kansas State University
Manhattan, KS
russfeld@ksu.edu

Scott Bell
Kansas State University
Manhattan, KS
rsbell@ksu.edu

Daniel Andresen
Kansas State University
Manhattan, KS
dan@ksu.edu

ABSTRACT

This paper presents our experiences and outcomes using Scratch to teach parallel computing concepts to students just learning about computer science. We presented versions of this material to middle school and high school girls during a STEM workshop and then to undergraduate university students enrolled in an introductory computer science course. Using the Scratch development environment, students are able to build, modify and observe the changes in the performance of applications which utilize multi-threaded, concurrent, operations. This includes scenarios which involve more advanced topics such as race conditions and mutex locks.

Developing these materials has allowed us to introduce these concepts in a programming environment much earlier than we have previously, giving instructors in down-stream courses the ability to build upon this early exposure. Survey results show that this approach resulted in a significant increase in both of these areas. For example, the number of students in our CS0 course who felt they could apply parallel programming to other problems using Scratch more than doubled, rising from 25 to 55 (out of 61 students that responded to both surveys). Likewise, the number of students who felt they understood what parallel programming means rose from 27 to 56. These results were achieved after just *one* class period. Similarly, 27 of the 37 girls responding to the workshop survey felt that they were capable of learning to write computer programs and 22 of 41 indicated they had an interest in a job using HPC to solve problems.

General Terms

Scratch, HPC, Parallel Programming, Outreach, K-12, CS0

1. INTRODUCTION

The increasing availability of low-cost, high-performance computing (HPC) hardware has led to a growth in research and development areas that utilize this computing power [2, 8]. This has also led to a growing demand for professionals who are interested in, and capable of, implementing these concepts. At the same time, while computer science departments are showing signs of recovering from the recent declines in enrollment, we are finding it challenging to attract enough students into computer science departments, and specifically into studying HPC, to meet this continually growing demand. This is especially true among women. The percentage of CS bachelor's degrees earned by women decreased from 27% in 2001 to 18% in 2010 [14].

It has been shown that a large percentage of students do not select CS as a major because they have either no idea about, or have misconceptions about, what computer scientists do [6]. This paper describes one step we have taken to help build student interest in pursuing a degree in computing, and specifically to build student interest in HPC. We developed hands-on activities that provide students with a glimpse of how useful HPC concepts can be and how integral this field is becoming in computer application development.

This new approach allows students to experiment with more complex, real-world problems, earlier in their academic career. They are able to focus more on the parallel problem solving approach, without the syntactic overhead encountered when using other common programming environments [15]. Historically, students in our program do not gain experience implementing HPC concepts until their junior or senior year. With these activities, we are now able to introduce this material effectively to students with *no previous programming experience.*

By introducing HPC using a simplified programming environment (Scratch), we remove several of the difficulties students encounter when they first try to implement parallel operations. The Scratch environment allows applications to launch multiple threads based on a single broadcast event. Our investigations show that the Scratch environment will utilize multiple processors if they are available, thus allowing students to easily observe the benefits (and dangers) of parallel computing. We tested a simple wind forecasting simulation on a quad-core system and as Figure 1 shows, by varying the model resolution and the number of threads utilized, there is a definite correlation between computing time and the number of threads used to generate the simulation output.

Using Scratch, we have presented this material so that middle and high school students can visualize and understand the benefits of parallel computing and the variety of ways in which it can be utilized. Student surveys from both K-12 students and students in an introductory CS course show that our work has been successful in building student

XSEDE '14, July 13 - 18 2014, Atlanta, GA, USA.
Copyright 2014 ACM 978-1-4503-2893-7/14/07$15.00.
http://dx.doi.org/10.1145/2616498.2616568 .

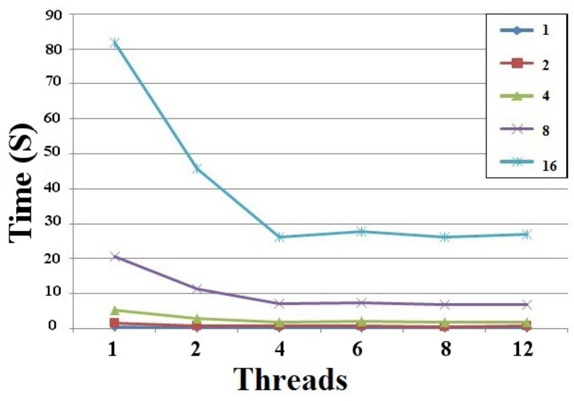


Figure 1: Simulation time vs. number of threads used for a range of model resolutions

interest and self-efficacy in relation to HPC concepts. For example, after our lesson, over 85% of CS0 students agreed with the following statements after completion of this material:

- "I could apply parallel programming to other problems in Scratch."
- "I understand what parallel programming means."
- "I would enjoy a job that involves using multiple computers working together to solve science problems."
- "I would be interested in performing research projects that involved using parallel programming."

All of this has been accomplished with students who are just learning programming and in a short period of time (1 or 2 class periods).

2. BACKGROUND

2.1 GROW and EXCITE Programs

The Kansas State University Office for the Advancement of Women in Science and Engineering (KAWSE) sponsors both the Girls Researching Our World (GROW) and the Exploring Science, Technology and Engineering (EXCITE) programs [1]. These are outreach programs targeted at building interest in STEM topics among middle and high school girls, respectively. Attendees are organized into cohorts of 10-15 students and participate in activities from several STEM disciplines over the course of each event (which vary from 1 to 3 days in length). Events are held on Saturdays throughout the school year, and longer camps are held during the summer. Over the past 3 years, the two programs have been able to host an average of 60 girls per year on the Kansas State campus.

The focus of these events is to show ways that people working within various STEM fields help make the world a better place. Sessions typically last less than an hour, and tend to be very hands-on, exploratory type activities. Within the Computing and Information Sciences Department, we give attendees an opportunity to explore some facet of computer science that allows them to see how our field might fit into their visions of future STEM related careers. During the summer of 2013, we decided to use a wind

Figure 2: Students collecting data from the wind modeling application.

modeling program developed in Scratch to show students how modern meteorologists utilize today's high-performance computing equipment to process and model weather patterns [9].

2.2 Introductory Computer Science Course

The need for parallelism to be taught throughout the CS curriculum is not a new discussion. We can find articles such as [13], written in 1995, that discuss the need to incorporate these ideas into various courses. However, this has remained an elusive goal due to the lack of both affordable hardware that can support such operations and a programming environment that students are able to access easily.

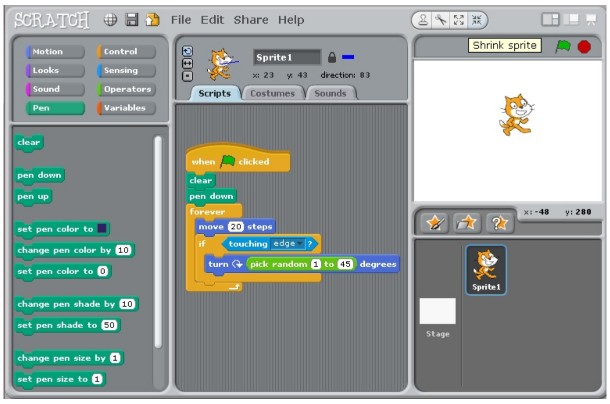
These limitations are beginning to erode, however. With today's affordable, multi-core processors, and a variety of user-friendly development platforms available, there is momentum building to provide students with HPC experiences earlier in the curriculum [3, 4, 5, 7]. Our goal is to provide students with this experience during their first computer science course.

Our department's introductory course (CS0) covers a broad array of topics from computer science. Besides just learning about the technical side of the field, the course also provides students with a view of how computer science relates to other disciplines. A large percentage of course time is spent discussing and investigating problems in areas such as bio-informatics, robotics, security, scientific computing, simulations and web technologies. Students enrolled in this course are typically majoring in either computer science or information systems and must take this course prior to taking their first programming course.

Given the growing need for graduates that are comfortable with high performance computing concepts, we have begun to introduce these concepts into this course. The first step took place during the Spring, 2013 semester.

2.3 Scratch

Figure 3 shows a screen shot of the Scratch programming environment which was developed by researchers from the Lifelong Kindergarten Group at MIT [12]. Scratch is a drag and drop environment, with keywords assigned to various shaped command "blocks" which can only connect together in syntactically correct ways. This greatly reduces the syntax issues that many beginning programming students strug-



```
when [flag] clicked
clear
pen down
forever
    move 20 steps
    if touching edge ?
        turn pick random 1 to 45 degrees
```

Figure 3: Screen shot of the Scratch Environment

gle with. We have found that this enables students of all ages to learn basic programming concepts and explore advanced computer programming concepts more easily. We have used Scratch in a wide variety of environments to introduce programming to students in classrooms and outreach activities from early elementary school (3rd-4th grade) level through the university level.

Over the last few years, we have been increasing the use of Scratch in our CS0 course. As was mentioned in section 2.2, the objective of this course is not a high volume of programming, which students will experience in the introductory programming course (CS1). This course is focused on showing students various ways that computer science fits into today's world and to help them begin thinking about problem solving in a computational manner. In developing Scratch-based simulation assignments for this course, we discovered that we can demonstrate some parallel computing concepts on multi-core processors using this programming environment.

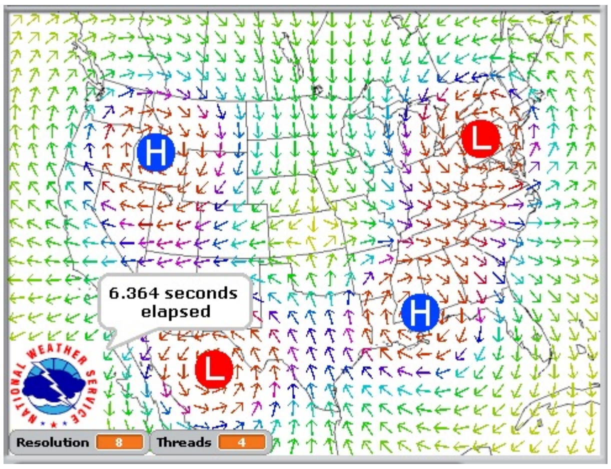
This led us to the idea that we could begin introducing these concepts to students earlier in our curriculum, giving them insight into the benefits of HPC well before they would typically explore such material (typically in an architecture or operating systems course which most students take as a junior or senior).

3. METHODOLOGY

Our objective for this project is to introduce more students, earlier in their academic careers, to the possibility of using high performance computing to solve large scale problems. There have been three stages to the project thus far. The examples, lesson plans and problem assignments can be found at: `http://bit.ly/russfeld_hpc`.

3.1 Measuring Wind Model Performance

During the summer of 2013, we hosted groups of students from both the GROW and EXCITE programs. With these sessions, we are typically very short on time (about 40 minutes per session), and in addition to our work with this application, we try to include a tour of the Beocat computing cluster on campus to show the students what a real high-performance computing system looks like [10]. Therefore, we utilized a pre-built simulation to allow students to test and discover the benefits of high performance computing in a tangible way.



Resolution 8
Threads 4

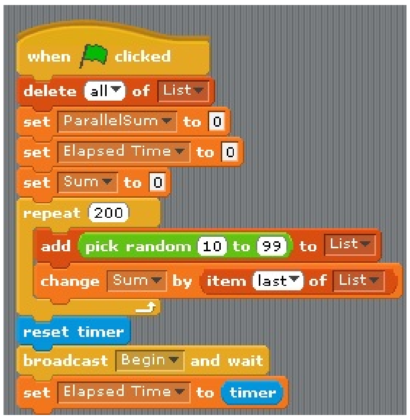
Figure 4: Wind model output

Since students in our target age group would be familiar with weather maps, we focused specifically on a wind map simulation. Given our time constraints, we could not ask the students to do any major development on the application. Instead, we walked through the program, explaining how it functioned and then let the students experiment with the application we had built to measure performance benefits and limitations. We used Scratch as the development environment because it provides very easy to use graphical operations and an easy to follow programming interface. This enables us to explain the program's functionality to students with no programming background.

We began each session with a 5 to 10 minute discussion of what high performance computing means. More specifically, we discussed what a multi-threaded application is, and finally introduced the idea of having multiple processors handle individual tasks simultaneously. As the students grasped that idea, we were able to introduce our simulation.

As you can see in Figure 4, the model background is a map of the United States. There are 4 pressure centers which can be relocated by dragging them around the map. The user is able to adjust the resolution of the model by setting the density of vector values that should be calculated and displayed (a value from 1 to 16). Additionally, the user can set the number of threads that are utilized by the application when performing these calculations (a value from 1 to 12). When the program runs, the number of calculation points is determined by the resolution and the wind vector at each point on the map is computed based on the distance that point is from the 4 pressure centers. At the completion of a model run, the application displays the time it took to compute the final wind map.

While this is a simplistic model for performing such a calculation, it gives students a good representation of how something we take for granted on a daily basis (weather forecasting) is actually performed. We assigned each pair of students a group of tests to run (some combination of resolutions and thread counts). Computation times were recorded in a table on the board and then plotted on a graph such as that shown in Figure 1. Once this was complete, we discussed the results with the students, having them try to explain why the improvement in performance leveled off once we reached 4 threads. After some discussion, we would arrive



```
when  clicked
delete all of List
set ParallelSum to 0
set Elapsed Time to 0
set Sum to 0
repeat 200
    add pick random 10 to 99 to List
    change Sum by item last of List
reset timer
broadcast Begin and wait
set Elapsed Time to timer
```

Figure 5: This code generates a list of 200 random numbers and then launches the other threads which do the adding.

at the idea that this was the result of hardware limitations (we were using computers containing quad core processors) and that we could potentially achieve even better performance by adding additional processing power.

3.2 Building a Summing Algorithm

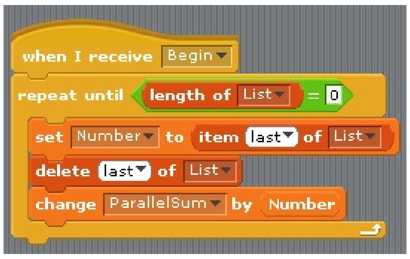
In the Fall of 2013, this same application was incorporated into our CS0 course. In previous semesters, the course had included material and information about HPC, but there was not an accompanying activity. We took the opportunity to include the simulation activity to further develop students' understanding of HPC concepts.

In the first lesson, students learned about computer hardware and how modern computers are designed. This included a brief introduction of fixed program and stored program computers, and the Von Neumann architecture present in most modern computers. We also discussed several hardware limitations and bottlenecks of previous hardware designs and how those have been overcome in later designs.

As part of that lesson, we presented the weather simulation as a way for students to see how the hardware present in a computer affects the performance of programs. As with the sessions described in Section 3.1, students were divided into groups and given a set of tests to run using the simulation while recording the elapsed time of each test. The data was then collected and graphed on the board, and students were asked to interpret what the data could tell us about the hardware present in the systems.

For the second lesson, we built upon that foundation by discussing the future of computing and how we must overcome the limitations of today's systems. Specifically, many computing tasks today are much larger than a single system can handle, as well as being so complex that they require an extraordinary amount of CPU processing time to complete. We then introduced HPC and showed how HPC is already being used today to solve some of these big problems. Following that, we allowed students to tour Beocat, the HPC cluster at K-State, to see firsthand what a real HPC cluster looks like.

After the tour, we introduced a new activity built in Scratch to help further demonstrate how HPC concepts are applied in a real computer program. The Scratch activity generated a random list of numbers and calculated their sum, and students were given the task of recalculating the sum of the numbers in the fastest way possible. We encouraged them to use multiple threads in their calculation, represented in Scratch as multiple sprites, each performing their own part of the calculation. As the students worked on that task, we slowly introduced one way of solving the problem, and attempted to show how it would operate in the real world.



```
when I receive Begin
repeat until length of List = 0
    set Number to item last of List
    delete last of List
    change ParallelSum by Number
```

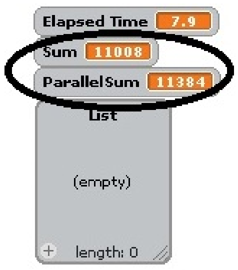
Figure 6: This code repeatedly pulls numbers from the list and adds them to the global sum until the list is empty.

The new activity worked well, allowing students some time to work hands-on with simple HPC concepts. This first attempt was rough around the edges. It was lacking many details that would help show how such a system would perform in the real world. For example, due to the way that Scratch is built, the students never became aware of pitfalls such as race conditions. As an initial attempt, we saw the potential in this activity and planned to build upon this the following semester.

3.3 Fixing the Concurrency Problem

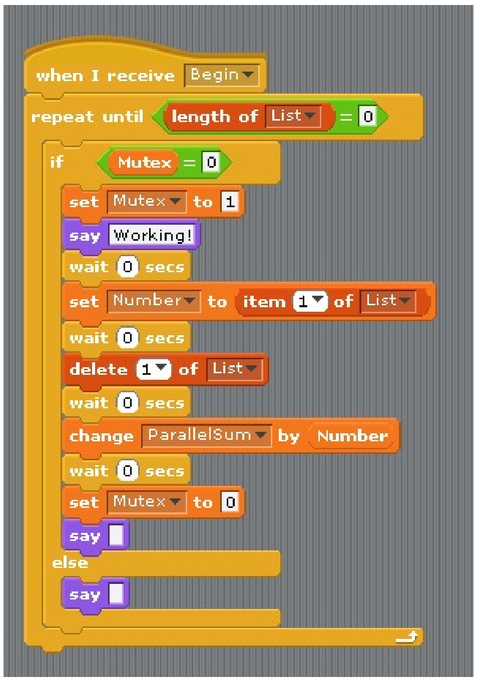
In the Spring of 2014, we decided that we wanted to introduce a more challenging and deeper investigation of parallel processing to the introductory students. We wanted to include the issue of race conditions and show students how such a problem might be mitigated through the use of a locking mechanism. To do this, we focused solely on the adding program similar to the one introduced during the Fall 2013 semester. Figures 5 and 6 show the code we developed for this project.

Figure 5 shows code which builds a list of 200 random numbers, restarts an internal Scratch timer and then launches a number of threads using a broadcast message. Figure 6 shows the code which those threads will follow. Each thread will pull values from the list and add them to a global 'total' variable. When we first implemented this application, we were surprised that we did not see any errors in the



```
Elapsed Time 7.9
Sum 11008
ParallelSum 11384
List
(empty)
length: 0
```

Figure 7: This output shows program variables and the differing sums when the race condition occurs.



```
when I receive Begin
repeat until length of List = 0
    if Mutex = 0
        set Mutex to 1
        say Working!
        wait 0 secs
        set Number to item 1 of List
        wait 0 secs
        delete 1 of List
        wait 0 secs
        change ParallelSum by Number
        wait 0 secs
        set Mutex to 0
        say
    else
        say
```

Figure 8: Modified code including mutex locking operations.

sum (a race condition never materialized). It turns out that Scratch is optimized to prevent such problems from occurring. The system assumes that the block of code within a looping structure is atomic and cannot be interrupted except at the very end of the loop block [11]. However, wait operations can be added between these blocks to allow for context shifts, and the related race conditions. This is true even if the wait length is set to 0. So, we could now demonstrate errors caused by a race condition, as shown in Figure 7. Note that the sum of the values computed by the parallel computation is not the same as the true sum which was computed as the values were added to the list.

Once the students understood the causes and ramifications of the concurrency problem, we demonstrated the use of a mutex lock to solve this problem, as shown in Figure 8. This, of course, introduced yet another issue. Based on how we set up the mutex lock, we lose the benefits of parallel operations and at this point, we challenged the students to find a way to minimize the problem.

4. RESULTS

Our initial use of this material during the summer of 2013 showed how promising our approach could be. Over 40 high-school and middle-school girls attended our sessions during the EXCITE and GROW workshops. Students were given a survey at the very end of the entire GROW/EXCITE Workshop, and 22 of 41 indicated they had an interest in a job using HPC to solve problems while 27 of 37 felt they were capable of learning to write computer programs.

Our experience during the Fall 2013 semester within the CS0 course was similarly positive. Students displayed interest in learning about how modern computers function and how larger problems can be tackled using HPC. The majority of the students were able to quickly understand

Question	p value	effect
Q1: I can learn how to write computer programs.	0.4531	0.141
Q2: I enjoy writing computer programs.	< 0.01	0.461
Q3: I would enjoy a job that involves using multiple computers working together to solve science problems.	0.024	0.461
Q4: I believe my experience learning Scratch will help me when I learn other programming languages in the future.	< 0.01	0.331
Q5: I understand what parallel programming means.	< 0.01	0.745
Q6: I would like to learn more about parallel programming.	0.383	0.118
Q7: I would be interested in performing research projects that involved using parallel programming.	< 0.01	0.416
Q8: Parallel programming is too difficult for me to understand.	0.038	0.267
Q9: I could apply parallel programming to other problems in Scratch.	< 0.01	0.696

Table 1: Statistical summary of survey items calculated using Wilcoxon signed-rank test (n=61).

the concepts presented in the weather simulation activity, and several of them experimented with pushing the performance boundaries of the application by running it with various other settings other than the ones they were directed to test.

Additionally, students were interested in the new multi - threaded sum activity. Students discussed ways that they would go about solving the problem even if they were not sure exactly how to build them, and by the end of the session several groups were expanding upon their working solutions to see how many threads they could use without crashing the program. Overall, this second iteration of the material gave us a very good look at how well the program and lessons can be adapted for a college-level course, and we were very optimistic about how well future lessons would be received.

Given the positive results experienced during the first two trials with this material, we approached the Spring 2014 semester with a plan to more closely study the effects the material was having with students. Students were given the option to participate in a survey-based study (they were given the opportunity to turn in a blank survey if they chose not to participate). Surveys were given before and after the lessons covering the HPC material. As part of the survey, students were asked to respond to various statements (shown in Table 1) using the 5 pt. scale shown here:

- Agree
- Somewhat agree
- I don't know
- Somewhat disagree
- Disagree

The statements included in the survey, shown in Table 1, covered student interest in programming as well as interest

	Agree		Neutral		Disagree	
	pre	post	pre	post	pre	post
Q1:	61	60	0	0	0	1
Q2:	51	58	9	2	1	1
Q3:	55	55	2	3	4	2
Q4:	48	53	1	0	12	8
Q5:	27	56	12	2	22	3
Q6:	57	55	3	2	1	4
Q7:	44	53	11	2	6	6
Q8:	7	11	19	3	35	47
Q9:	25	55	32	3	4	3

Table 2: Summary of survey results with agree/disagree responses aggregated.

and self-efficacy as it pertains to parallel computing concepts.

The results of these surveys are shown in Figure 9, and the statistical analysis is given in Table 1. We have also included the numerical results showing the number of students that agreed, were neutral or disagreed with each statement in Table 2. In this table, we aggregate 'Somewhat agree' with 'Agree,' and 'Somewhat disagree' with 'disagree.' Of particular note is the increase in agreement for statements 5 and 9. These responses show the growth in student confidence that occurred over the course of the lessons. Other statements show similar, although not as dramatic, improvements.

Questions 1 and 6 displayed a decrease in agreeing students by 1 and 2 students respectively. However, these results were found to be statistically insignificant when checked using the Wilcoxon signed-rank test (p values less than 0.05 are considered to be significant). The pre-treatment results for both of these statements started out with a high percentage (greater than 90%) of students agreeing, so there was little room for a significant positive change to either of these statements.

Overall, the results were outstanding, with greater than 80% of the students agreeing to all of the statements except statement 8 in the post surveys. Note that statement 8 is "Parallel programming is too difficult for me to understand," so we are wanting a negative response for this statement. Thus, the inverted responses to this statement denote a positive result. The results from these surveys given during the Spring 2014 semester match our observations of the interest and understanding that students had while participating in these activities in the classroom during the Fall 2013 semester.

5. SUMMARY

Using Scratch, we have discovered a method to easily present HPC concepts to students with little or no programming background in a meaningful way. Our experiences show that students over a wide range of ages (middle school to college freshmen) are able to comprehend these ideas and survey results show that their interest and self-efficacy have grown as a result. The number of students who felt they understand what parallel programming is and who think they can apply parallel programming to other problems doubled in our introductory course.

In the future, we plan to expand the scope of these simulations to include other subject areas within computer science. One such area of interest is bio-informatics. A proposed simulation would display several organisms of the same species with different traits present. Students would then analyze the genetic makeup of the organism and attempt to deduce which traits were caused by different parts of the makeup.

In addition, we plan to enhance the activities used in the introduction to computing science class to include more HPC concepts such as a work queue or producer/consumer relationships. We would also like to evaluate how well these Scratch-based activities could be used to introduce more advanced HPC concepts to juniors and seniors in an architecture or operating systems course.

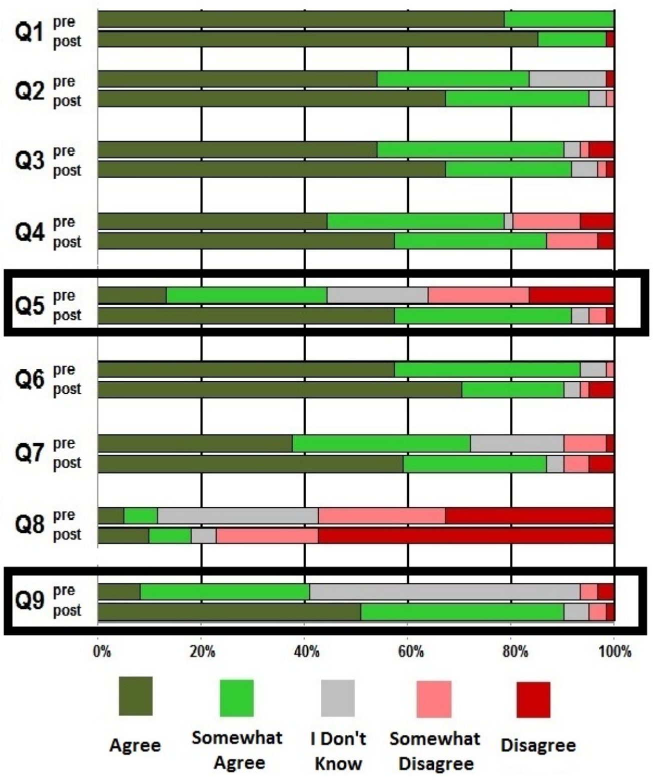


Figure 9: Results from pre and post surveys. We were especially interested in the results from statements 5 and 9 (statements are listed in Table 1), which both had over 100% growth in the number of students who agree or somewhat agree.

References

[1] K-State Office for the Advancement of Women in Science and Engineering. `http://www.k-state.edu/kawse/`.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[3] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A View of the Parallel Computing Landscape. *Commun. ACM*, 52(10):56–67, Oct. 2009.

[4] R. Brown and E. Shoop. Modules in Community: Injecting More Parallelism into Computer Science Curricula. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 447–452. ACM, 2011.

[5] K. B. Bruce, A. Danyluk, and T. Murtagh. Introducing Concurrency in CS 1. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 224–228, New York, NY, USA, 2010. ACM.

[6] L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. *SIGCSE Bull.*, 38(1):27–31, Mar. 2006.

[7] D. J. Ernst and D. E. Stevenson. Concurrent CS: Preparing Students for a Multicore World. *SIGCSE Bull.*, 40(3):230–234, June 2008.

[8] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU Cluster for High Performance Computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 47. IEEE Computer Society, 2004.

[9] R. Feldhausen, S. Bell, and D. Andresen. Engaging Students in STEM Fields through High Performance Computing., 2013. Poster presented at ASEE-Midwest Section Annual Conference, Sept 18, Salina, KS.

[10] Kansas State University Computing and Information Sciences. Beocat. `http://beocat.cis.ksu.edu/`.

[11] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 2010.

[12] MIT. Scratch, MIT Lifelong Kindergarten. `http://scratch.mit.edu`, 2013.

[13] C. H. Nevison. Parallel Computing in the Undergraduate Curriculum. *Computer*, 28(12):51–56, 1995.

[14] NSF. NSF Women, Minorities and Persons with Disabilities in Science and Engineering. `http://www.nsf.gov/statistics/wmpd/2013/`.

[15] J. M. Wing. Computational Thinking. In *VL/HCC*, page 3, 2011.