Teaching Parallel Programming and Software Engineering Concepts to High School Students

Adam Rifkin

CRPC-TR93321-S
May 1993

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Teaching Parallel Programming and Software Engineering Concepts to High School Students

1 Introduction

This paper takes the stand that it is never too early to teach so-called "hard" concepts in computer science. Specifically, basic principles in parallel algorithm development and software engineering can be introduced to students first learning about computers. The key is to present ideas in a manner that is simple, fun and suited to the audience. During this paper we discuss an interactive exercise we developed to test this premise, based on sorting algorithms, that we conducted with one hundred minority students aged 14 to 17, on March 19, 1993. We present evidence that our students, relative neophytes to high level computer science notions, had fun while learning alleged difficult concepts.

1.1 Motivation

Many students are first exposed to computers in high school computer classes, which is unfortunate because these classes usually emphasize writing sequential program code. As a result, students develop initial prejudices when thinking about using a computer to solve a problem: instead of using software principles to think of designing a good parallel or sequential algorithm, they often immediately turn to programming. This can lead to incorrect and inefficient solutions. Students should be introduced to software engineering principles — specifically, *good parallel and sequential algorithm development* — from the beginning. This can be done at a high level which is fun and exciting to students, as discussed in this paper; later on, students can be introduced specifically to such notions as efficiency analysis and correctness verification. James Cook [Coo93] draws a fine analogy between teaching students to program and to write prose:

> Teaching programming is like teaching writing: you have to *inspire* the student. Coding is like typing, a necessary skill for communicating concepts, but

one that will come with practice. The most important part of algorithm design is the idea—translating an algorithm into code is a secondary step, tantamount to scribing a poem.

This observation is a global one which must be addressed by the leaders of computer science. In late 1992, the National Research Council issued a full report [Cou92] addressing what the field of computer science and engineering currently comprises, how the field is presently doing, what the field should be doing, and what the field needs to do in order to prosper. In reference to education, the report points out the need for emphasizing fundamental principles.

A CS&E education cannot provide comprehensive exposure to all computing problems that might be encountered later on, but rather should provide a good foundation on which to build. With a good understanding of the basic intellectual paradigms of CS&E, graduates can more fully exploit computing.

Fundamental principles should be presented at the beginning of a person's computer science education. If we can initially present software engineering concepts as a natural way of thinking about solving problems (figure 1), students will have a knowledge base that enables them to develop correct, efficient parallel and sequential software as they mature as programmers.

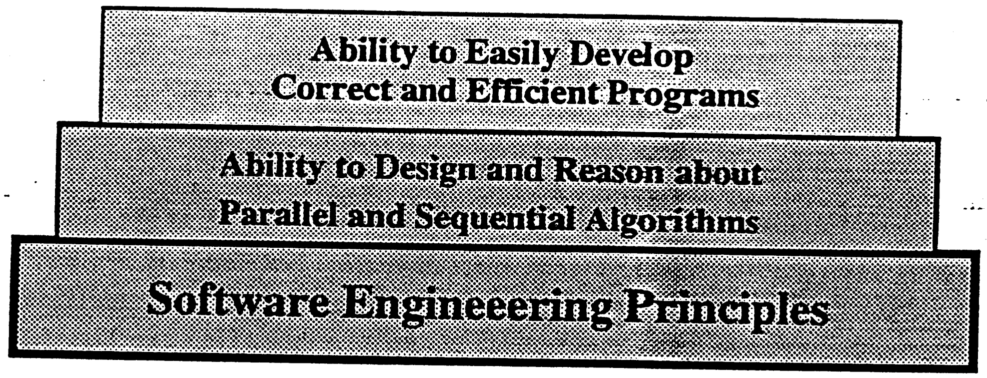

Ability to Easily Develop
Correct and Efficient Programs
Ability to Design and Reason about
Parallel and Sequential Algorithms
Software Engineeering Principles

Figure 1: The foundations for good programming habits should start from the beginning.

We designed a workshop for a group of minority high school students who were novices in the field of computer science. The workshop was devised to demonstrate that a few basic software engineering and parallel algorithm design principles are, in fact, intuitive ways to think about solving some problems. In the following segments we discuss our reasons for choosing this target group and these workshop goals.

Why software engineering? Habits in computer programming are established early. Therefore the introduction of software engineering principles during a student's formative period of programming will help develop good habits from the start.

Why parallel algorithm design? Most introductory parallel algorithm textbooks, such as [J92], are designed for advanced undergraduate or beginning graduate college students, because a parallel solutions to problems are commonly thought to be difficult. This is unfortunate, because parallel computing is playing an increasingly important role in computer science and offers great promise for future progress of computer technology. It is also unfortunate because for some classes of problems, parallel programming is actually a more natural way of designing a solution. One bold introductory parallel programming textbook [Les93] claims that "parallelism is as important and fundamental as is sequentiality," yet today many programmers believe parallel programming is difficult. If we can teach new students that parallel solutions to some problems are as *intuitive* as sequential solutions to some problems, we can remove the bottleneck ingrained prejudice which prevents many seasoned programmers from attempting to learn parallel programming.

Why high school students? The May 1993 issue of the *Communications of the ACM* is dedicated to technology for K-12 education. As noted in the viewpoint column of that issue [Bea93], "computer literacy has become as important as the three R's." A study on teaching computers in secondary schools [Bec93] notes that "those of us who lap up each new computer application as if it required no effort or at least no discipline to master its functions may forget that novices require a certain period of exploration and basic algorithmic instruction before they can expect to be productive." Increasingly, these "novices" are high school students receiving their first formal computer instruction. By targeting our efforts at these students, we encourage good practices from the start.

Why minority students? The trends cited by the National Research Council [Cou92] are discouraging: "Women and non-Asian minorities continue to be underrepresented... at all levels in the CS&E educational pipeline." If minority high school students can be inspired to investigate computer science in an environment which is fun and exciting, they may be more confident in choosing to pursue further education and eventually a career in computer science.

1.2 CRPC Workshop

More than one hundred Los Angeles County high school students visited Caltech March 18 and 19, 1993, for a computer awareness program. The two-day event, dubbed "Computers: The Machines, Science, People, and Jobs!", was a pilot program designed to encourage minority teenagers to consider computer careers, by exposing them both to the science and technology associated with computer science, and to successful professionals who work in the field.

Sessions focussed on the history and evolution of computers, their modern applications, and the future possibilities in the field. Participating students traveled to different sites on the Caltech campus for seminars, workshops, and video presentations, and got to see and report to each other about more than two dozen on-site applications of computer technology and computational sciences.

By exposing the young scholars to examples of real computers in use, professors sought to support the following program objectives:

- To inspire curiosity about computers, by showing them to be fun and exciting.
- To demystify and humanize computer science through face-to-face interaction with professionals in the field.
- To preview the challenges and rewards of a computer-related career.
- To develop participant confidence that they can succeed in such a career.

The Caltech event was organized by James Muldavin for the National Science Foundation-sponsored Center for Research on Parallel Computation, jointly run by Caltech and Rice

University, and with cooperation from MESA – Mathematics, Engineering, and Science Achievement.

As described in [Mul93], the overall goal of the event was "to encourage minority youth to consider preparation for a career in the computational sciences." This paper discusses the workshop we conducted as an introduction to parallel and sequential computer programming concepts, with an emphasis on software engineering principles through good algorithm design.

2 Workshop Description

The workshop was an experiment. We wanted to teach, but we also wanted to learn about teaching. Many tradeoffs were considered, as illustrated by figure 2; we wanted to balance keeping the students' interest with teaching them a few specific concepts. Therefore, the workshop was designed to be fun, interactive, and entertaining, but it should also be educational and simple to prepare and run.

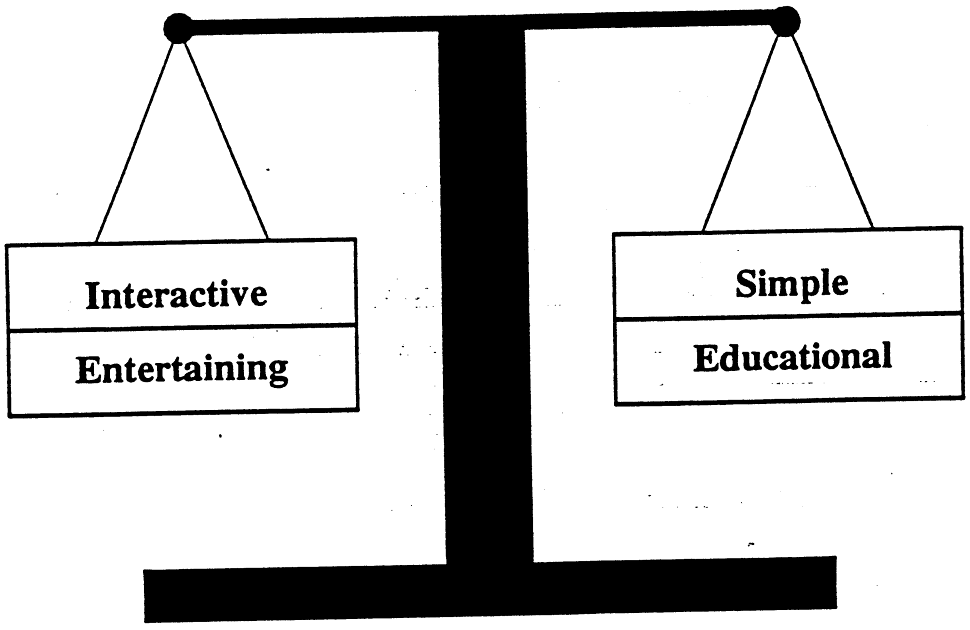


Figure 2: Balancing student interest with instructional virtue.

Constraints. The activity was designed to be useful for 6 to 144 students. It was designed to last only 50 minutes. It was to be held in a small field outside.

Goals. It was our intention to encourage the students to learn more about computer science in the future, and to convince them that computer science is fun. We also wanted them to think critically about issues involved in program design. We therefore set a goal to convey the following points:

- Computer science concepts are not difficult.
- Computers operate by following a specific set of rules.
- More than one set of rules ("algorithms") can be designed to solve a problem.
- Good algorithm design is important for obtaining correct and efficient solutions.
- Parallel operations are sometimes a natural way of thinking, and can have more efficient performance than sequential operations.

Activity Overview. What makes our approach unique is the emphasis on fun. We decided to use the task of sorting a list of numbers to convey our points; allowing each student to "be a number" in the list would provide interactivity. Sorting is a well-defined problem; "without doubt sorting has received more attention in the computer science literature than any other algorithmic task" [MS91, §8]. After posing the sorting problem to the students, we would guide them through three different algorithms. These algorithms were chosen based on how simply they could work using the students as numbers, and based on how well they illustrated the workshop's goals:

1. *Bubblesort* [MS91] to illustrate a sequential solution.
2. *Even-Odd Transposition Sort* [CM89] to illustrate a natural parallel version of Bubblesort.
3. *Parallel radix sort* [MS91] as an alternative parallel solution for comparison.

2.1 Outline of Workshop

The 50-minute workshop took the following format.

- Students were given a short handout the previous day to familiarize themselves with the flavor of the workshop.
- The workshop began with a 20-minute talk by Professor Mani Chandy, briefly discussing computer science, good algorithm design, and parallel programming. The talk also went over the problem of sorting a list of numbers, and the three algorithms we were going to explore. Audience participation was encouraged, and the underlying emphasis of the talk was that computer science is challenging, but not difficult and often fun. §2.2 summarizes the talk.
- We then brought the students outside and divided them into nine groups of approximately the same size. Each group was led by a volunteer from Caltech.
- Three 10-minute exercises were performed by the group, one for each of the three algorithms discussed, led by instructions given by workshop leader Mani Chandy. Coordination was handled by the group leaders. The algorithms as we performed them are discussed in §2.3 (Bubblesort), §2.4 (Even-Odd Transposition Sort), and §2.5 (Parallel Radix sort).
- The workshop closed with an iteration of the goals. Students were then asked to complete a short evaluation form. A synopsis of these evaluations is provided in §3.

2.2 Key Points of Talk

The 20-minute talk was divided into two sections; the first gave background which included the following points:

- Computers are machines which have changed the way we live. During this activity, the students would learn about how computers work today, and how they may work in the future.
- A computer is just a machine with many parts. One part is called a processor, and it can only act according to a specific set of rules.

- A human computer programmer solves a problem using a computer by designing a specific set of rules to solve the problem. These rules form a computer program, consisting of one or more algorithms designed to solve parts of the original problem.
- An algorithm is a specific set of rules constructed to solve a given problem. Sometimes it is not so easy to come up with a good set of rules which efficiently solves a problem.
- Many times there exists more than one algorithm for a given problem, and it is up to the computer programmer to determine which algorithm to use. The students would explore three algorithms for one problem during this activity.
- Today, most computers contain only one processor. Some parallel computers (machines with more than one processor) have been built to solve problems faster, but they are not as widely used.
- Some people think that parallel computer programming is difficult. This activity would help the students to understand that many problems can be solved easily (and more quickly) using parallel algorithms.

The second part of the talk described the problem of sorting a group of numbers in ascending order. For example, consider six numbers out of order:

$$\boxed{42} \quad \boxed{55} \quad \boxed{16} \quad \boxed{62} \quad \boxed{17} \quad \boxed{15}$$

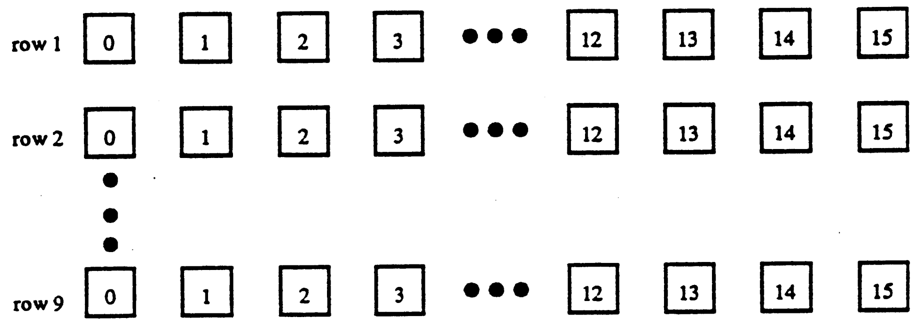
The goal would be to sort the numbers:

$$\boxed{15} \quad \boxed{16} \quad \boxed{17} \quad \boxed{42} \quad \boxed{55} \quad \boxed{62}$$

How can one tell a computer to do this? The students were told to remember that we must give a specific set of rules for the computer to follow. Student solution suggestions were then entertained, after which the three different solutions we would be employing were discussed.

The format of the group activity was then discussed. Students would be divided into nine groups of about the same size. A supervisor would coordinate each group. Each group would perform the three sorting exercises at the same time, each corresponding to a different sorting algorithm.

The field where the activity was to occur was arrayed with nine rows of lines, one for each group:



The nine groups could each perform the exercise at the same time, and groups were small enough to be easily manageable. At the beginning of each exercise, each person in each group stood on a space in his or her group's line. Line spaces were colored for use in the second algorithm (see §2.4). Instructions were given assuming 16 people per group; groups accommodating less than 16 people were able to adapt.

After each person in the group was standing on a space, the Caltech student would then give each student a card with a number inside. Different groups had differently colored cards. Cards were folded cardboard $8\frac{1}{2}$" by 11" sheets, with a unique integer printed inside.

Students were told the eventual goal of each exercise: to get the smallest number in the group standing at space 0, the next smallest at space 1, and so on, with the biggest number in the group standing at the highest numbered space in the line.

For each exercise, then, the person essentially became a piece of data representing his or her number. Each group then followed an algorithm, as designated by each exercise, with sorting being performed by having students physically move from space to space according to a set of rules. It was stressed that students follow the rules of each exercise specifically, since computers operate by a set of rules and cannot deviate. To prevent chaos from reigning supreme, we asked that the students not talk during the exercise; in this manner we could show them how communication could be achieved without talking. The only communication which needed to be done during each exercise could be performed by students discreetly showing each other the numbers on their cards. Students could only show the contents of their cards when specifically instructed to do so.

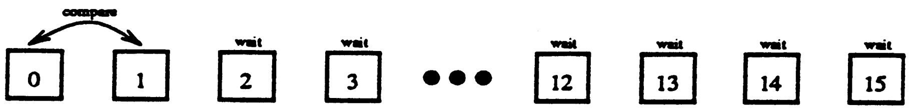
After each exercise, the cards were returned to the group coordinator, and students were asked to consider the strengths and weaknesses of each algorithm. After the three exercises had completed, students filled out short evaluation forms.

2.3 Bubblesort

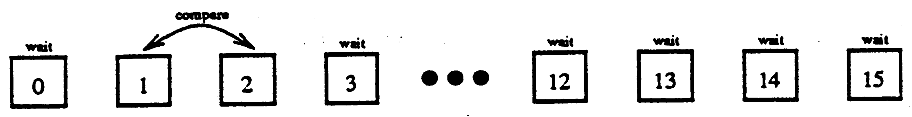
Students were informed that the first algorithm was a sequential technique called Bubblesort. Numbered cards were handed out randomly by group leaders as students occupied their spaces in the line.

This exercise consisted of up to 16 rounds. The start of each round was signaled by the workshop leader. During each round, the group coordinator walked down the line. Each round consisted of up to 16 phases of two people comparing numbers. The group coordinator pointed to the people who should to compare numbers as he or she walked down the line. During the comparison, the two people were told to reveal their numbers to only each other. If, after the comparison, the person standing on the lower numbered space had a number *less* than the person standing on the higher numbered space, the two people would *remain* standing on their spaces. If the person standing on the lower numbered space had a number *greater* than the person standing on the higher numbered space, the two people would *switch* spaces.

For example, during the first phase of a round, the people standing at spaces 0 and 1 were pointed to by the group coordinator, signifying that they should compare their numbers:



If the number at space 0 was less than the number at space 1, the people holding those numbers stayed put. Otherwise, they switched places. Each round continued down the group line with comparisons performed in a similar manner. For example, during the second phase of a round, the people standing at spaces 1 and 2 compared their numbers:



If the number at space 1 was less than the number at space 2, the people holding those numbers stayed put. Otherwise, they switched places. The rest of the phases in each round were similarly coordinated. When a group leader reached the end of his group line, he raised his hand. When the workshop leader saw that every group leader's hand was raised, he announced that the next round could begin. In this manner, groups were synchronized, so that they were all expected to finish at the same time.

After round 0, the person in each group standing at the highest numbered space was holding the largest number. Students were asked to observe why, in terms of invariants of the algorithm we were using. After later rounds, the person at the second highest numbered space was holding the second largest number, the person at the third highest space was holding the third largest number, and so on. Students were asked to observe why, in terms of invariants. After the last round, the smallest number was at space 0. At this point the list of numbers was sorted. We explained why this should be, in terms of invariants. Students were asked to consider that the list might actually have been sorted in less than 16 rounds, and that, even though *they* knew the list was sorted, a computer would not have such a gestalt recognition.

At the end of the exercise, students returned their cards to their group coordinators, who then shuffled them for handing out for the next exercise. During the shuffling, students were asked by the workshop coordinator to consider the strengths and weaknesses of Bubblesort and how it could be improved. This led into the next sorting activity.

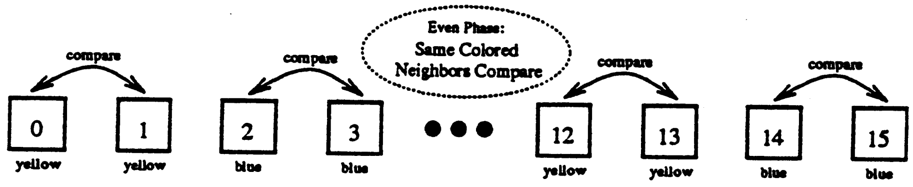
2.4 Even-Odd Transposition Sort

One of the problems with Bubblesort resided in efficiency of communication: even if the group had 16 members, only two were allowed to communicate by showing their cards, at any given time. Students intuitively realized that this was tedious and slow. The Even-Odd

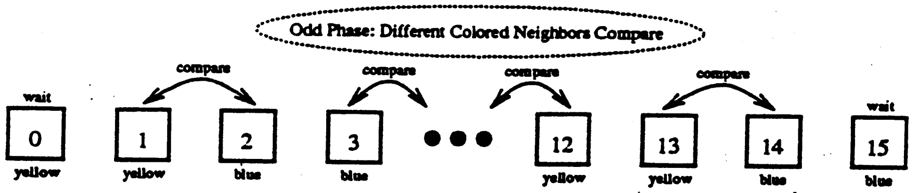
Transposition Sort was presented as a natural parallel extension of Bubblesort, with parallel communication allowing for improved performance. Whereas Bubblesort was inefficient because only two people compared numbers at a time, for Even-Odd Transposition Sort, all pairs of neighbors could compare numbers at each phase.

Numbered cards were handed out randomly by group leaders, with students standing on the spaces on which they ended the previous exercise. The exercise broken into rounds consisting solely of an "even" and an "odd" phase. The start of each phase of each round would be signaled by the workshop leader. Students were told that comparisons of numbers would work the same way as in Bubblesort.

During the "even" phase of a round, neighbors standing on the same colored spaces compared numbers:



After the comparison and optional movement, the person with the smaller number was standing on the lower numbered space. During the "odd" phase of a round, neighbors standing on differently colored spaces compared numbers:



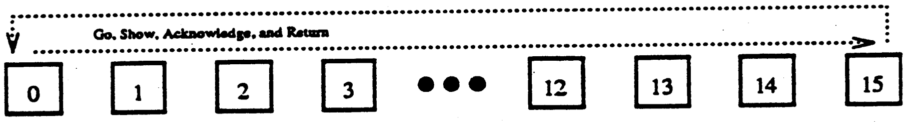
After the comparison and optional movement, the person with the smaller number was standing on the lower numbered space. Students were asked to note that rounds did not need to be coordinated by the group leaders. Students observed, in terms of invariants, why the list was becoming "more sorted" with each phase, and why, after the last round, the list of numbers was be sorted. A short mention of efficiency analysis was given, in terms of performance.

At the end of the exercise, students returned their cards to their group coordinators, who then shuffled them for handing out for the next exercise. During the shuffling, students were asked by the workshop coordinator to consider the strengths and weaknesses of Even-Odd Transposition Sort and how it could be improved. They were also asked to consider what makes Even-Odd Transposition Sort a parallel algorithm, why this algorithm was more intuitive to them, and how this algorithm compares with Bubblesort.

2.5 Parallel Radix Sort

To further advance the concepts of good design and parallel algorithms, we introduced the final sorting exercise, in which a global broadcast could be used instead of communication between neighbors. Students found this to be the most intuitive way of thinking of the problem of sorting; if we had just given them numbers initially and told them to sort the numbers using whatever method they wanted, chances are good that they would have used a parallel style of thinking involving a global broadcast.

Numbered cards were handed out randomly by group leaders, with students standing on the spaces on which they ended the previous exercise. The exercise, instead of being broken into rounds of phases, was played as a race. The workshop coordinator indicated go. For this exercise, the group leader walked down the line with the student standing at space 0, revealing his or her number to every other student standing in place:



The roving student knew another student had seen his or her number by means of an acknowledgement; in this case, this was achieved through hand raising. An acknowledgement signified that the roving student could continue down the line, and the group leader coordinated this roaming. Meanwhile, every student standing in the line kept a private counter. Whenever a student saw a number *less* than his or her number, his or her private counter would be increased by 1. Private counters all started with a value of 0. After the roving student had shown and received acknowledgements by every student in the line, he or she would return to his space, and the group leader would coordinate the next student's movement down the line, first showing his or her number to students standing on lower numbered spaces, and then showing it to students standing on higher numbered spaces.

After every student had a turn to walk down the line, the group coordinator indicated that everyone should simultaneously leave the space he or she was standing on, and walk to the space labeled with the same number as his or her private counter. After doing this, the list of numbers was now sorted. The reason why the spaces in the line were numbered starting with zero is that the smallest number in the line had a private counter value of zero – at no time did that student encounter a number revealed to be smaller, so his or her private counter was never increased by 1. Students were asked to intuitively think, in terms of invariants, about why the list should be sorted at the end of Parallel Radix Sort.

Students returned their cards for the final time, and were asked to consider how this algorithm differed from the other two sorting algorithms in terms of strengths and weaknesses. They were asked to consider the differences between methods of communication and between parallel and sequential solutions for the same problem. They were also informed that these are not the only techniques for sorting, and were asked to consider other solutions which might exist.

3 Synopsis of Results

Students were allowed to keep the colored numbers and places as souvenirs. After the workshop, an evaluation form was given to every student containing six quantitative questions and two qualitative questions, addressing and assessing various aspects of the workshop. In all, 81 completed surveys were received, and the responses as a whole were quite positive. An overwhelming majority of the students learned something, had fun, and got an overall better impression of computer science. The questions, response frequencies and averages to the six quantitative questions are provided.

How much did you learn doing this activity? (none = 1, a little = 2, some = 3, a lot = 4)

Response Frequencies				*Average*
1's	2's	3's	4's	*Response*
2	8	36	35	3.28

How much fun did you have doing this activity? (none = 1, a little = 2, some = 3, a lot = 4)

Response Frequencies				*Average*
1's	2's	3's	4's	*Response*
1	7	45	28	3.23

How valuable were the group discussions? (poor = 1, fair = 2, average = 3, good = 4, excellent = 5)

Response Frequencies					*Average*
1's	2's	3's	4's	5's	*Response*
0	6	15	50	10	3.79

What is your overall impression of this workshop? (poor = 1, fair = 2, average = 3, good = 4, excellent = 5)

Response Frequencies					*Average*
1's	2's	3's	4's	5's	*Response*
0	3	16	33	29	4.09

Would you like to study computer science further? (definitely not = 1, probably not = 2, perhaps = 3, definitely = 4)

Response Frequencies				*Average*
1's	2's	3's	4's	*Response*
1	4	56	20	3.17

How favorably do you feel about computer science because of this workshop? (very bad = 1, bad = 2, okay = 3, good = 4, very good = 5)

Response Frequencies					*Average*
1's	2's	3's	4's	5's	*Response*
0	1	19	36	25	4.05

3.1 Student Comments

In addition to the quantitative questions, students were invited to write extended answers to two other questions. Responses varied in size and content.

What is the most important thing you feel learned today? About one-third of the students mentioned the value of communication and/or cooperation, about one-quarter mentioned parallel programming, and about one-fourth mentioned how computers work. Among the noteworthy answers were:

- "A lot about my future field of study."
- "Communication and cooperation are always important."
- "I feel that I learned that computer science is a field which is more interesting and not as hard as I thought it was."
- "I learned how to sort numbers without [verbal] communication."
- "I learned the basic concepts of parallel computing which I have not been exposed to before."
- "I've learned somewhat how computers work and how fast you can work with communication."
- "Machines cooperate just like people."
- "Parallel is faster than sequential."
- "Sequential programming is stupid."

- "That communication and cooperation make programming faster. And that programming is not monotonous and dull but it is exciting and FUN!"
- "That parallel programming is fun and faster and more fascinating than sequential programming."
- "The possibilities of using many processors to accomplish a goal faster than with a single processor."
- "Through logical reasoning, communication, and cooperation, work can be done rapidly."

Feel free to add any additional comments you might have. Many of the students mentioned how much fun they had participating in the workshop. Among the noteworthy comments were:

- "A very fun activity."
- "Computer science is an excellent field."
- "Extension of time would be better on this project."
- "I enjoyed the lecturing."
- "I felt that I was being talked down to. The ideas were explained simplistically. I would have liked to develop the ideas further or learned more about them."
- "I'm glad you went slowly enough with the explanations to make sure EVERYONE understood the concepts."
- "In the future, I'd like to see a bit more depth added to the activity by perhaps giving more complex examples of practical applications of parallel computers."
- "I like how we did the sequencing and it was somewhat fun. I liked it even though it's dumb."
- "I think the more the students participate, the more they learn."
- "I think this is a great program and it should continue!!!"

- "I've had NO previous experience with programming, even though both my parents are computer programmers, and this workshop gave me a good overview of how you can command a computer."
- "I would like to have more time so that Professor Chandy could talk more about parallel programming. Thanks a lot Mr. Chandy! I enjoyed your lecture and game!"
- "The group activity is a good way to demonstrate the concept of parallel computation. Maybe show them more applications of parallel computation besides sorting."
- "This is a very educational program for students."
- "This process was slow at first but then we got the hang of the game and it was good."
- "We should have more time meeting the presenters."
- "Words cannot express my feelings."

3.2 Critique

It is difficult to gauge how much the students actually learned, since there are no specific metrics that could be used except for the student evaluations. Nonetheless, the positive feedback manifested is phenomenally encouraging for the design of other similar activities to engage at the beginning of a student's computer science education. Responses to questions 4 through 6 reflect the success of this exercise. In just 45 minutes, we were able to demonstrate by example the value of good algorithm design, parallel methods of thinking, and communication considerations. Though we may not know how much the individuals learned, we are confident that each student took something away from the workshop.

In designing, managing, and evaluating this exercise, we learned about education principles as well. Future versions of this exercise, as well as other activities involving problems unrelated to the sorting problem, could be designed in a similar manner. The exercises could be less complicated, and we could provide discussion sessions during which students could bounce around their ideas about what has been done. Future workshops should also provide a means for personal discovery, by allowing students to think of their own creative solutions to problems. Activities should be designed with the size and educational background of the target audience in mind. In fact, it is conceivable that similar activities could be designed for adults and grade school children as well.

4 Conclusions

Beginning students can learn at a high level some fundamental principles of computer science through well-designed workshops. When developing such a teaching activity, certain aspects should be addressed and balanced: exercises should be fun, simple, interactive, and have specific concepts to teach. Such exercises need not explicitly employ computers; often concepts can be taught better without them. Through the design and evaluation of the workshop presented in this paper, we see that early encouragement of students may build strong foundations of computer science principles within the individuals.

Through use of the sorting workshop described in this paper, we were able to get one hundred minority high school students excited about the prospects of future work in computer science. Through the use of fun exercises, we were able to teach a few basic principles of software engineering: the importance of good algorithm design, the tradeoffs involved with an algorithm selection decision, and the inherent methods of thinking about parallel and sequential computer programming. We were also able to convey some basic principles of a relatively new technology — parallel programming — which may prevent prejudices towards sequential programming for problem solving in the students' future instruction. Other activities can be similarly designed with these goals in mind.

Acknowledgements

Special thanks go to K. Mani Chandy and John Thornley, who were instrumental in designing this workshop, and to Peter Carlin, who helped develop the parallel radix sort segment. We are grateful to the following people who have commented on our work and earlier drafts of this paper: Ulla Binau, James Cook, Robert Harley, Rohit Khare, Rustan Leino, Berna Massingill, and Paolo Sivilotti. We also thank the following people, without whose help our workshop would have failed: JoAnn Boyd, Diane Goodfellow, H. Peter Hofstee, James Muldavin, Gail Stowers, and Ted Turócy. This work was supported in part by the NSF Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008. The government has certain rights in this material.

References

[Bea93] Gary Beach. Viewpoint: Building a foundation. *Communications of the ACM*, 36(5):13–14, May 1993.

[Bec93] Henry Jay Becker. Teaching with and about computers in secondary schools. *Communications of the ACM*, 36(5):69–72, May 1993.

[CM89] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation.* Addison Wesley, May 1989.

[Coo93] James Cook. Personal communication, 1993.

[Cou92] National Research Council. *Computing the Future: A Broader Agenda for Computer Science and Engineering.* National Academy Press, 2101 Constitution Avenue, NW, Washington, DC 20418, October 1992.

[J9́2] Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley, 1992.

[Les93] Bruce P. Lester. *The Art of Parallel Programming.* Prentice Hall, 1993.

[MS91] Bernard M.E. Moret and Henry D. Shapiro. *Algorithms from P to NP - Volume I: Design and Efficiency.* The Benjamin/ Cummings Publishing Company, 1991.

[Mul93] James C. Muldavin. A pilot computer and computational sciences awareness program for minority youth at the california institute of technology. Center for Research on Parallel Computation Minority Youth Program, March 1993.