Educative and Visual Tool for Learning Parallel Programming

by Vong Vithyea Srey

February 26, 2013

Abstract

Initially, the education in parallel computing was offered to only senior level students. This was due to the extremely high cost of computing facilities and the complication of parallel algorithm and parallel programming paradigm. However, the computing revolution has been speedily pushing this situation forward. The computing facilities become more available while the price has decreased. Therefore, the need for knowledge in parallel computing becomes higher and higher. On one hand, most institutions were contributing an enormous effort to integrate the parallel computing into the early stage of undergraduate computer science/engineering curriculum, this includes IEEE Technical Committee on Parallel Processing (TCPP). On the other hand, some institutions were trying to integrate the parallel computing into the high school courses. For an example, in 1999, Weizmann Institute of Science - Israel - did an experiment in teaching parallel computing to high school students. Consequently, this research was conducted to fulfill the gap in teaching parallel programming to junior students who only had some basic knowledge in sequential programming and computer architecture. The tool expected to cover the K (Know the term) bloom level topics that defined by the TCPP curriculum. However, this project was done in a short period of summer time, which in turn, some of the topics were not completely implemented; and they were hoping to be implemented in the future stage of the project. More importantly, this visual tool is purposely built with the enjoyable analogies that related the parallel computing to the real world scenarios, where it is rated as the helpful tool by students (who have tried out this application and volunteered in the unofficial evaluation, which conducted at the last stage of this project.

1 Introduction

Over the last few decades computer architecture has been rapidly revolutionised. The processor speed has been doubled every 18 months [2]. However, the dramatically increase in clock frequency has reached its limit and the excessive heat and power become the main problems [8]. The thread-level parallel

and instruction-level parallel processor hardware, which implemented on von-Neumann architecture using the techniques of dynamic instruction dispatch, branch prediction or speculative execution has no more space to grow [8]. This phenomenon led to the new era of Parallel and Distributed Computing. Multi-core processors are the only key to keep up with the doubling increment on microprocessor performance [1]. Suddenly, the parallel computing appeared to be available everywhere, from desktops and laptops to handhelds and game stations. All computing devices are equipped with dual-core, quad-core or even hexa-core with hyper-threading and programmable graphic cards [3]. Nonetheless, most programmers are hesitate to adopt a parallel programming model and abandon the beneficiaries of multi-core computers [8]. This is because the programmers were trained to think and solve problems sequentially and the concept of instruction is executed one after another is deeply embedded into their mind [6]. Additionally, the parallel programming is more complex and error prone than sequential programming, at least with the parallel programming systems that are in use today [5]. With these modern parallel computing power, the programmers have to effectively use of the increasing levels of coarse-grain parallelism in order to achieve the highest potential performance [1].

IEEE Technical Committee on Parallel Processing (TCPP) knows that it is infeasible to speedily change the programmers to adapt to the parallel programming. As a result, TCPP is proposing to include the parallel programming into the early stage of Computer Science and Engineering studies [10]. However, changing the entire curriculum is a big challenge for both students and lecturers. Therefore, many institutions around the world started to find the way to integrate and adopt the parallel programming into their early stage of curriculums, but in a way that make the students feel excited and interested in the new paradigm of parallelism. In addition, many education providers have noticeably reported their experiments and the results regarding the way the parallel programming was integrated into their curriculum. Some of those have shown that teaching parallel programming explicitly is not easy and well understood.

Based on previous experiments which were done by many institutions, this research aims to put together all of those problems that have been encountered and bring up a new methodology to introduce the parallel programming to the early stage computing students.

2 Related Works

The complications in teaching, learning and working on parallel programming paradigm have led to many discussions in the computing society. Many education providers have been experimenting in integrating this new paradigm into their existing curriculum. Especially, they also wish to integrate the parallel programming into undergraduate level instead of postgraduate level. Some of the remarkable lessons learnt are:

In 1995, The California Institute of Technology, USA, did a research on teaching parallel computing for analysing Electromagnetic Computational (EMC) problems (the detail of the report is in [9]): The programme's goal was to introduce the final part of a graduate level EMC course in solving some complex problems. The teaching programme was based on the 15-hour lectures, which started with the history of parallel computing and moved on to the theory and methodologies used in parallel computing with mathematical examples.

In the history of parallel computing section, the paper shows the trends of computing power developments against the rapidly growth in term of size of the mathematical problems that need to be solved. For example, the figures of the growth of the computing power over the last 20 years was shown in comparison to the growth of the matrix size that need to be LU factored over the same period of time and scale.

It also reported that the programme materials covered many aspects of parallel computing theories. It includes the introductory parts of the pipelining versus the parallelism, both models of parallel computations SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data), processors organization, processor arrays and elementary parallel algorithms.

From the introductory theories above, the programme then focused on the mathematical techniques, which are particularly relevant to the EMC problems. Those mathematical techniques discussed are: matrix multiplication, solving linear equations via Gaussian elimination and LU factorization and Fast Fourier transform.

At the very end of the programme, the result was quite significantly impressive. However, the main issue in parallel computing which found in the programme is the need for the development in a more efficient algorithm that could take full potential of the power of parallel computing hardware architectures.

An Israeli institute, Weizmann Institute of Science, had an attempt to teach parallel programming on high school students in 1999, the fully report given in [4]. The experiment course had an objective to integrate the parallel programming and algorithm to the twelfth grade of the scientifically oriented high school students who were between 17 and 18 years old.

The course materials were divided into 9 chapters, which covered: 1) the basic concept of concurrent and distributed computing; 2) the structure of concurrent algorithm; 3) mutual exclusion problem in a share memory model; 4) binary semaphores; 5) the concept of correctness and introduction of invariants for proving correctness; 6) solving the dining philosophers problem; 7) Linda tuple-space model (as an alternative to synchronization model); 8) transformation from concurrent to distributed systems and introduction to the model of message passing in asynchronous systems; and 9) the discussion on reliable systems in the distributed model.

The course was taught to the students in an analogical way. For instance, all

the problems raised during the class and the algorithm analysis were presented with the real world problems, such as (below is an example presented in [4]):

```
taste the juice
if it is not sweet then
    go to the kitchen
    take a sugar cube
    come back
    add the sugar cube to the juice.
```

The experiment was concluded that the students' performances were almost excellent. There were some improvements on the students' skills. At the end of the course, most students were able to solve problems of process coordination in various models. Most students can analogise many technical problems in the real world and vice versa.

However, since they were less mature and less able to deal with formal materials if compared to the university students, many of them were complaining that the course requires a lot of thinking and it was quite challenging for them. Some felt embarrassed for taking part in the so-called "stupid" dramatisations and others said that they did not like the course.

Furthermore, some difficulties were found during the course. Firstly, students did not understand the concept of the model, where the model was specified by the types and operations. Some students tended to create new operations, which were not defined in the model. Secondly, students always misinterpreted the model operations and made wrong assumptions about the behavior based on informal concepts rather than using the formal rules. Most importantly, some students did not even attempt to use concurrency and gave sequential algorithms to the problems.

In 2012, the early initiation to the parallelism in the computing engineering degree at University of Murcia (UM), Spain (the detail of the report is in [3]): The Computing Engineering School (CES) of UM found out that the integration of parallel programming into the early stage of computer engineer curriculum is a must. By supporting of this idea through the TCPP curriculum proposal, the University launched this project in fall 2011. The project aimed to add the parallel programming into the early stage of the computer engineering degree's curriculum. It should be noted that in the old curriculum, the computer engineering students were introduced the parallelism concept as the optional courses in the last year of their degrees.

This project finished in 2012, where two out of ten courses in the second year (2011-2012) of CES had the project introduced. Advance Computing Architecture (ACA) course was allocated in the first semester while the Concurrent and Distributed Programming (CDP) was allocated in the second semester. Both courses covered parallel computing architecture, parallel programming paradigm, parallel algorithm and crosscutting.

- ACA included Performance Analysis, Pipelining, Control Dependencies, Static and Dynamic Scheduling of Instructions and Memory System Organisation and Performance.
- CDP included Loosely and Strongly Coupled Systems Programming and Classic Programming Paradigms in Distributed Systems.

The courses were split into 8 activities throughout both semesters (the detail of each activity is described in[3]). Each activity started with the introduction of the theory and moved on to the hands-on practice in the SCC lab. This approach was implemented successfully. At the end of each activity an evaluation was done on both students and lecturers. The participants were encouraged to provide feedback on the usefulness of the subjects and the materials covered in the activity. Finally, the students were assessed on their understanding of the course materials.

The conclusion in the report defined that the courses were successfully implemented. The students learnt many different aspects of the parallelism. This fact reinforced that the parallel programming courses were successfully added into the compulsory courses at the very early stage of the computer engineering curriculum. However, despite this remarkable result, there were some difficulties due to the shortage of professors who have high experience in teaching parallel programming.

Dr. Nasser Giacaman from University of Auckland (UoA), New Zealand, introduced a new way of teaching parallel programming using analogies and live coding demonstration to undergraduate students of Computer Science (full report is made in 2012 in[7]): Dr. Giacaman reported about his experiences in teaching the second year students of Computer Science at UoA, where students were passed two courses of programming: the first course was the introduction to the Java programming, and the second course was the extension of the first course with the introduction to the data structure. Meaning the students were oriented to the knowledge of sequential programming in Java. His report mentions that the course was the first one in parallel programming that the students had taken.

The course covered 12 lessons over 4 weeks (3 lessons each week). In the first week, the students are introduced the concept of threads, Java's thread and Runnable API, thread life-cycle, critical sections, mutual exclusion, thread-safety mechanism in Java and the cost-benefit balance of fine grained versus coarse grained thread-safety. In the second week, some real concurrency GUI applications were introduced to the students, where all the theories and the reasoning behind the problems - that can be occurred - related to the first week lessons were exposed. For instance, some buttons were intentionally made the GUI frozen in order to explain the reasons behind the freezing. The last 6 lessons dedicated to the extensions of concurrency to parallel computing.

Every lesson the lecturer always consistently used "real world" analogies to explain the theories of the parallel computing. This teaching technique motivated by the facts that Java is an Object-Oriented language and it makes more

sense while the Object-Oriented language is to model after the "real world". As an example, the lecturer made an analogy of employees in a company. The computer system is viewed as the "company" where "employees" - which can be seen as threads - were hired to sit and work on a "desk" - which is analogized to the processor core.

Added on to the analogies explanation of the theories, some live coding demonstrations were constantly shown to the students during the lectures. Additionally, the hands-on aspect was also brought into teaching. The students were given plenty of opportunities to practice themselves with the live code - which was always made available to students through the course's web page - and also in the lab. Lastly, part of the assignment also consisted of the real practical aspect.

Finally, the students were asked to complete the anonymous online evaluation. Based on the online evaluation results and the student exams' result, it can be confidently concluded that the practical approach is the most effective way in teaching. Moreover, the students found out that the analogies that the lecturer used to relate parallel computing concept to the " real life" scenarios and the live code demonstration was very helpful in learning and understanding the parallel computing theories and concepts.

Beside the four reports, which were briefly presented above, there are plenty of such reports about the experiences in teaching parallel programming. From these reports, it is clear that the parallel programming was introduced only to the senior level of computing students at the very early stage of computing revolution era. A few years later when the parallel computing became more available to the students and the consumers; many education institutions started to search for a different way to teach the parallel programming to their students, especially to undergraduate students at the early stage of their programme. Some reports also showed evidence about the success and effective approach in teaching the parallel programming to the second year undergraduates. Impressively, some institutions, like the Israeli Weizmann Institute of Science can be a good example, which had not only been trying to teach the parallel programming to the undergraduate computing students, but they also tried to bring the parallel programming knowledge to high school students. However, because of the complicated nature of the parallel programming paradigm, it became clear there was a big challenge in teaching this new concept to the students. The fact was not just reflected in high school students, but also in the case of undergraduate students. Consequently, many reports have concluded and recommended that the further work in teaching parallel programming should be done in a simpler and more encouraged way to the students.

3 Target Users

Based on the experiences that have been gathered from the related work above, this educational tool was created with the purpose to teach parallel computing concept in an analogical way to the early stage of computer science/engineering undergraduates. Most reports described earlier clearly showed that stage 2 (year 2) Computer science/engineering students are the most suitable group for introducing the parallel programming, since they have, at least, learned the sequential programming in their first year courses. Which in turn, they have some understanding and knowledge in programming concepts and the basic of computer architecture. All of these sequential programming knowledge is also the foundation that the creator of this tool was expecting the students to have in order to understand the scenarios in the tool.

However, since this visual tool was built with the idea of analogising the parallel computing concepts into the visually "real world" scenarios. It was expected that the students (users of the tool) should be able to understand the scenario easily, even though they have very limited knowledge in computer architecture and the sequential programming. Therefore, it is believed that this tool could possibly be used/learned by even high school students, but those students must have, at a minimum, some knowledge of the basic concept of computer architecture and sequential programming, which is a prerequisite.

4 Covered Topics

The lesson learnt from the section 2 above shows that most of the parallel computing courses cover four sections of the topics; they are Parallel Computing Architecture, Parallel Algorithm, Parallel Programming Paradigm and Cross-cutting/Advance. This finding is also in line with the TCPP curriculum which reported in [10]. However, the TCPP curriculum proposes a lot more topics than the one, which covered in those courses.

Standing on these findings and the TCPP curriculum proposal, this tool is planning to cover those 4 sections of topics, which concluded earlier. Nonetheless, as this tool is targeting the young audiences (it can possibly be used by high school students as long as they have the basic knowledge in sequential programming and computer architecture), therefore the parallel computing knowledge that covered by this tool are included only the basic concepts.

IIt is believed that the basic concepts that covered by this educational tool are the mandatory topics, which can shape the junior level students to have really good knowledge and understanding in extending their study in parallel computing. Especially, the tool also helped the users to develop solutions or algorithms for the parallel programming paradigm. This idea can also be backed up by the TCPP curriculum proposal in [10], because most of the topics covered here were generated from this proposal where the bloom level K is defined (K - stands for Know the term (basic literacy) [10]).

The introduction of basic topics covered in this visual tool are:

- The basic of computing architecture, such as CPU, Data Storage.
- The basic of concurrency and parallel programming paradigm, such as thread, task, thread life cycle.

Because of time constraint (10 weeks of full time or 400 hours), the analysis into this educational tool only covered two sections described above. Nevertheless, the software architecture of this tool was designed and implemented in a way that allows other topics to be carried out and built on the existing results in future researches. In other word, this is an ongoing development. Additional topics that should be added into this research in the future work include, but not limited to, parallel algorithm and crosscutting topics.

5 Game Analogies

The tool was designed to be a fun and enjoyable game, where all characters and the game scenarios are analogised the parallel computing concepts related to the real world. The aim is to motivate the users to pay more attention to the game scenarios and characters by initially introducing them to the real world scenarios that they are familiar with; and at the end of each scenario, a description of the analogy of that character or scenario to the parallel computing concept is presented.

Figure 1: The Screenshot of the Game Theme Where A Crane Operator is Building a Brick Tower

All scenarios in the game are designed in a logical flow, where the users are supposed to start from the first level (Sequential Scenario), to second (Concurrency Scenario), third (Parallel Scenario), and so on.

The analogies are continuously associated to a consistent theme throughout the game. All scenarios are shown with the brick construction area theme, where:

- The system of a/the Crane/s which is/are controlled to build bricks-tower/s can be seen as the computing system. The Figure 1 shows a theme where a crane operator is building a brick tower, which relates to the computer system where the CPU is controlled by a thread to complete a task.
- The Crane is related to the processor.
- The Crane Operator is related to the GUI thread.
- The Customer Service is related to the event thread.

- The Brick is related to the data.
- The Brick Conveyor is related to the data bus.
- The Brick Tower (to be built) is the program's task.

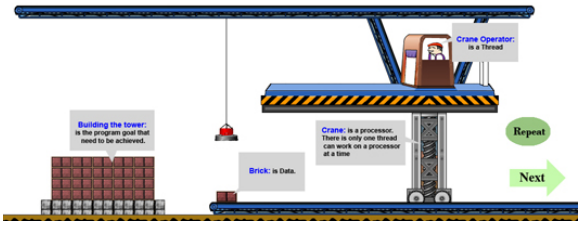
5.1 Sequential Scenario

The idea behind the scene is to explain to users that the processor executes the instructions sequentially - by the main thread, which is sitting on top of the processor - to complete a task. The analogy of this idea is that users are shown a game scene where the main task of the game (main task is analogised to the task that the processor needs to complete) is to build the brick tower. The users are told that there is a Crane Operator who is controlling the Crane (the main thread is sitting on top of the processor). Then the users are asked to command the Crane Operator step by step to build the brick tower. At the end of the game, the users are told that every command (every instructions given to the processor) that users have given to the Crane Operator is completed by the Crane Operator sequentially. The Figure 2shows the screenshot of the last screen of the Sequential Scenario game where users were explained about the analogy of each character in the game.

5.2 Concurrency Scenario

The Figure 3 shows the screenshot of the analogy description screen of the game in the Concurrency Scenario level; where in this level, the users are exposed to a more complex problem. At the beginning the users are shown that the game's task in this level is to build the brick tower. While the Crane Operator (is the analogy of the GUI thread) is building the brick tower, the Customer Representative (who is the analogy of the event thread) is receiving the order to load the bricks onto a truck. The users will be asked to decide whether the new task will be put into the tasks queue and let the Crane Operator complete them sequentially or the new task should be completed by another Crane Assistant (who is seen as another thread). In case the users decided to have another Crane Assistant to complete the new task then another thread will be created to complete the new task concurrently with the GUI thread. This means the users are expected to see the processor can be used to execute instructions by only one thread at a time. At any time, if there is more than one thread sitting on a processor to be executed more than one task, then the threads need to share the resources between each other in order to complete their respective

Figure 2: The Screenshot of The Analogy Screen of The Sequential Scenario



task concurrently. This scenario can be of advantage that tasks are completed concurrently, but there might be some disadvantage as well in terms of the time spent for executions since the resources are shared among those threads.

5.3 Parallel Scenario

In this scenario, the users are given similar problems to the Concurrency Scenario. The users are shown that the Crane Operator is building his brick tower. At the same time, the Customer Service is ordered to complete another task, which is loading the bricks onto the truck. Similarly, the users are asked to decide whether to put the task into the queue for the Crane Operator or call for a Crane Assistant to complete the task. However, in this scenario, the users are given two different Cranes (which analogy relates to two core processors); and each crane is attached to its own brick conveyor (its own bus). Any decision that the users gave, the game will try to inform the users that if there are two core processors with their own resources it is better to have all resources fully used. Meaning that if any one of the core processors is free then it is a waste. Nonetheless, in order to use all of the resources, the users have to create a new thread to occupy the free processor. In other word, for this case (there are two core processors) the users should create a new thread to occupy another processor to complete the new task, while they should leave the GUI thread undisturbed to complete its own task. This analogy is shown in the Figure 4.

Figure 3: The Screenshot of The Analogy Screen of The Concurrency Scenario

6 Evaluation

The tool is purposely built to train the junior students about parallel computing and plan to be used among computer science/engineering students at the University of Auckland. Hoping that an evaluation will be done after the tool has been used in a pilot project. However, since the project was conducted in a short period of time during the summer holiday (when the students were not around at the university) and this report was written before the actual official evaluations took place. Therefore, an

Figure 4: The Screenshot of The Analogy Screen of The Parallel Scenario

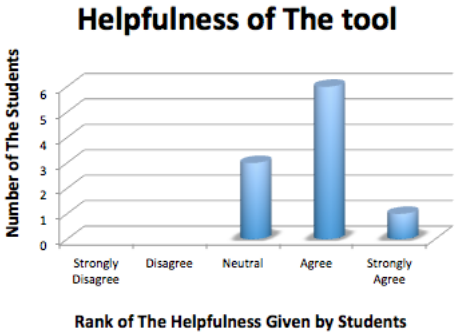
unofficial evaluation was done with 10 students who can be found and available at the university around this time.

The unofficial evaluation was done on the students who were doing their summer school at the Auckland University and some are the researchers' friends. All of them have at least the basic knowledge of sequential programming and computer architecture, but they all have not learnt any parallel programming and never had any thread related knowledge.

Firstly, those 10 students were explained about the tool and its purpose. They were also briefly explained about some analogies, which they are expected to see in the tool. After that, they were asked to use the tool for as long as they wish. Finally, they were asked to complete a set of unofficial questionnaire, which listed in Appendix 1.

The results presented in Figure 5 shows that around 70% of the students ranked this tool as helpful (60% agree and 10% strongly agree) in explaining them the idea and background of parallel computing, while the other 30% neutral. Additionally, some students suggested that the tool should be presented with a fewer text since they had to read a lot of instruction. Others expressed the desire of having a button that can speed up the game while they were playing it. However, there were some students who stood remained muted in giving any comment to improve the tool.

Figure 5: The Helpfulness of The Visual Tool for Learning Parallel Computing



Even though this is not an official evaluation, however all these results and suggestions were taken into the consideration in order to improve the tool in future projects and it is also used as the compass to estimate how successful the project is before it can be officially deployed. It is important to note that these results represented that the tool is considered to be a very helpful tool for the early stage of computer science/engineer students.

7 Conclusions

Since the parallel computing architecture become widely available, there have been many researches and pilot courses on teaching parallel computing to the early stage of computer science/engineer students, all of them found out that the integration of the parallel computing into the early stage of undergraduates in computer science/engineer's curriculum is necessary. This idea is also well supported by the TCPP curriculum. The curriculum proposed by both teaching projects and by TCPP cover four sections, which are: parallel computing

architecture, parallel programming paradigm, parallel algorithm and crosscutting. Similarly, this project aims to produce an analogical education tool that covers the four topics plus targeting the junior level students who have some knowledge in sequential programming and basic knowledge in computer architecture. Furthermore, the purpose of this tool is to help the students in learning parallel computing in a fun and enjoyable way. As a result, the tool was ranked as a helpful tool in learning parallel programming by many junior computer science/engineering students. This is because of the way it uses the analogies to relate parallel computing concepts to the real world scenarios making students enjoy the tool in an entertaining way and at the same time they could learn the complicated parallel computing concept easily. However given the project was done in a limited time, therefore some of the features were not completely implemented. It is hoping that all the missing features and an official evaluation will be done in the near future.

Appendix 1

Unofficial Evaluation Questionnaire

Age :_____ Occupation: _____

Programming Knowledge: ☐Undergraduate in CS/SE ☐Postgraduate in CS/SE ☐Not CS/SE

Parallelism Knowledge: ☐ Never heard ☐ Beginner ☐ Medium ☐ Expert

After you have tried this game,

Do you understand better regarding the sequential programming?

☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

Do you understand better regarding the concurrency programming?

☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

Do you understand better regarding the parallelism programming?

☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

Do you know what is the different between Sequential, Concurrency and Parallelism?

☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

Is this tool helpful in learning the parallel computing concept?

☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

Do you have any features you would like to see?

_____

_____

_____

References

[1] Comp 522: Multicore computing, February 2013.

[2] Moore's law, January 2013.

[3] M.E. Acacio, J. Cuenca, L. Fern'ndez, R. Fern'ndez-Pascual, J. Cervera, D. Gimenez, M.C. Garrido, J.A.S. Laguna, J. Guillen, J.A.P. Benito, and M. Requena. An experience of early initiation to parallelism in the computing engineering degree at the university of murcia, spain. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1289 –1294, may 2012.

[4] Mordechai Ben-Ari and Yifat Ben-David Kolikant. Thinking parallel: the process of learning concurrency. *SIGCSE Bull.*, 31(3):13–16, June 1999.

[5] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The paderborn university bsp (pub) library. *Parallel Comput.*, 29(2):187–207, February 2003.

[6] W.W. Carlson, J.M. Draper, D.E. Culler, K. Yelick, E. Brooks, and K. Warren. Introduction to upc and language specification. *Technical Report CCS-TR-99-157*, May 1999.

[7] N. Giacaman. Teaching by example: Using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1295 –1298, may 2012.

[8] C.W. Kessler. Teaching parallel programming early. 2006.

[9] R. Perez. Parallel computing in the teaching of electromagnetic computational methods for analyzing emc problems. In *Electromagnetic Compatibility, 1995. Symposium Record., 1995 IEEE International Symposium on*, pages 41 –46, aug 1995.

[10] S.K. Prasad, A. Chtchelkanova, F. Dehne, Gouda M., A. Gupta, J. Jaja, K. Kant, A.L. Salle, R. LeBlanc, A. Lumsdaine, D. Padua, Parashar M., V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu. Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing – core topics for undergraduates. *Technical Report*, December 2012.