EduPar Virtual Poster Session

Jesús Cámara
Dept of Engineering and Technology of Computers
University of Murcia
Murcia, Spain
jcamara@um.es

José-Carlos Cano
Dept of Engineering and Technology of Computers
University of Murcia
Murcia, Spain
josecarlos.canol@um.es

Javier Cuenca
Dept of Engineering and Technology of Computers
University of Murcia
Murcia, Spain
jcuenca@um.es

Toshiyuki Maeda
Faculty of Management Information
Hannan University
Matsubara, Japan
maechan@hannan-u.ac.jp

Mariano Saura-Sánchez
Dept of Mechanical Engineering
Technical University of Cartagena
Cartagena, Spain
msaura.sanchez@upct.es

Lewis Tseng
Computer Science
Boston College
Boston USA
lewis.tseng@bc.edu

Akiyoshi Wakatani
Faculty of Intelligence and Informatics
Konan University
Kobe, Japan
wakatani@konan-u.ac.jp

Martina Barnas
Computer Science
Indiana University Bloomington
Bloomington, IN, USA
mbarnas@iu.edu

***Abstract*—This paper provides an overview of posters accepted for the EduPar 21 poster session. Poster sessions have been an important part of the EduPar workshops, providing an opportunity to facilitate interactions and fostering the community. After a hiatus caused by the COVID-19 pandemic we decided to resume the poster session tradition and to hold the first virtual poster session in EduPar's eleven years history.**

***Index Terms*—Parallel Computing Education, Engineering Education, Data Science Education, Curriculum Design, Assignments, Pedagogical issues, Pedagogical tools**

I. Introduction

Parallel and Distributed Computing (PDC) is critical for most computing activities. Computing devices, including home and office PCs, laptops, and mobile devices, contain multicore CPUs and GPUs; the ever increasing use of web-based services, cloud computing and the Internet of Things, is not possible without distributed computing. Yet, parallel and distributed computing is still not mainstream in computer science education. This underscores the urgency to adapt education to prepare students for addressing the challenges of current and emerging computing technologies.

The EduPar workshop, now in its 11th year, is a regular workshop of the IPDPS conference since its second year. Against the backdrop of IPDPS, a major conference focusing on parallel and distributed computing, EduPar provides a global forum to share findings and experience in addressing one of the most pressing needs in education related to computer technology. Additionally, this effort is in coordination with the TCPP curriculum initiative (http://www.cs.gsu.edu/ tcpp/curriculum). However, since its inception the workshop has served an additional important purpose – fostering the community. For this reason, in addition to regular papers the workshop encourages submissions of posters. The poster session has been an important part of the workshop program. Dedicated to work in progress or experience reports by early adopters, the poster session allows for exchange of ideas and informal conversations.

Unfortunately, the COVID-19 pandemic interrupted the cherished tradition of EduPar poster sessions. In 2020, we were forced to hold the 10th anniversary EduPar online. This led us to cancel the poster session, in hopes that in 2021 it would be resumed. This year, sadly, our community is still not able to come together in person in a physical location. However, having had more than a year of experience of online engagement, be it teaching or meetings, we decided to take a risk and hold the first virtual poster session in EduPar's history.

We have received five submissions of which three are to be presented at our virtual event. We applaud the authors for their willingness to embark with us on the exciting but unprecedented attempt to reimagine a poster session online.

This paper is organized as follows:

In Section II, Jesús Cámara, José-Carlos Cano, Javier Cuenca and Mariano Saura-Sánchez describe *PARCSIM: a Learning Support Tool for Parallel Programming Students* - a tool that will be used in a Computer Science senior level course Programming Multi-core Architectures at the University of Murcia, Spain.

In Section III, *Teaching Distributed Systems by Teaching Cryptocurrency: Challenge, Motivation, and Curriculum Design*, Lewis Tseng shares experience in teaching distributed systems to students with minimal background in computer systems and computer networking at Boston College.

Section IV, *Evaluation of web-based tools for beginners in AVX and MPI parallel programming education* by Akiyoshi

Wakatani and Toshiyuki Maeda describes a web-based educational system for beginners of parallel programming such as CUDA, AVX, OpenMP, POSIX threads, MPI and OpenACC, that is being develop, including evaluation with two small groups of students (16 and 6).

II. PARCSIM: a Learning Support Tool for Parallel Programming Students - Camara, Cano, Cuenca and Saura-Sánchez

This work was supported by the Spanish MICINN and AEI, as well as European Commission FEDER funds, under grant RTI2018-098156-B-C53.

Parallel computing has become an essential learning subject in recent years, mainly due to the appearance of heterogeneous systems composed of several multicore CPUs and general purpose graphic accelerators, GPGPU. The heterogeneity of today's hardware platforms has made it possible to address many complex scientific problems. Although in many cases mathematical models already exist for these problems, their numerical resolution was not feasible in an acceptable time due to the limited computational capacity of previous generation computers. However, modern platforms support the application of parallelism techniques to solve this kind of problems by sharing the computation among several computational units, thus reducing execution times. In this context, the student who is being trained in parallel and distributed programming must face the challenge of designing a space-time map that reflects the distribution of the different tasks that make up the problem to be solved among the different computational units available. In addition, it must be considered that these tasks can have, at the same time, internal parallelism, which involves a complete management of multilevel parallelism by tuning a set of adjustable parameters such as the number of CPU threads to generate, the library or routine to be used for each computation and the number and type of GPUs to be incorporated in the problem resolution.

In this work Camara, Cano, Cuenca and Saura-Sánchez present PARCSIM (PAR-allel C-omputations SIM-ulator), a software simulation tool that shows, in an interactive way, the advantages of applying parallelism techniques as well as the selection of the appropriate library according to the type of computation to be performed on a given hardware platform. PARCSIM allows the user to capture algorithms that solve scientific problems using matrix algebra techniques. This task is performed thanks to the graphical user interface that collects the algorithm computations and their dependencies, making up a graphical parallel algorithm model. After that, this tool experiments with computations using different parallelism parameters and numerical libraries, analysing the execution times obtained in each case. On the other hand, this tool offers a self-optimizing execution mode that can help non-expert users in the selection of the best values for the parameters, proposing the temporal order of the computations to allow better exploitation of parallelism in a particular compute node, knowing the number of CPU cores and the number of available GPUs.

A. *A course on Programming Multi-core Architectures*

The tool is going to be used in the course Programming Multi-core Architectures, PMA, which is part of the last semester of the Computer Science Degree at the University of Murcia, Spain. This Degree is organized into four years, with five specializations in the last year: Computer Engineering, Software Engineering, Computation, Information Technology and Information Systems. The students in the fourth year have already studied Algorithms and Data Structures, Computer Architecture and Concurrent Programming. PMA is mandatory for the students in the Computer Engineering intensification. A small number of students (about fifteen per year) choose this intensification, therefore teaching is personalized and focused on the work of each student. The course is designed to introduce students to programming techniques and tools for hybrid platforms composed of a multicore CPU together with one or more accelerators.

PMA is structured in 3 blocks: the first block is focused on multicore CPU programming, using OpenMP [**?**] for the practical work. After this part, the general organization of GPU architecture is studied, learning GPU programming using CUDA [**?**]. Finally, the last block of the course is dedicated to face the challenge of creating efficient programs for hybrid platforms, considering different heterogeneous task-mapping techniques and properly combining OpenMP and CUDA programming.

Along the semester, in practical sessions, students have to perform a series of basic training exercises in each one of the working environments of the 3 blocks: CPU, GPU and CPU+GPU. After this training phase, each student must tackle the development of a whole hybrid parallel program, starting by programming an OpenMP version of this program for multicore CPU, followed by a CUDA version for a GPU and finally, making a hybrid implementation for a compute node composed of a multicore CPU together with several GPUs of different characteristics. The proposal introduced this year consists of using PARCSIM as a compass to guide students in planning and developing this hybrid program. The use of this tool will allow each student to have a global vision from the beginning of how the effective workload distribution among the computational units will occur depending on both the structure of the program and the proposed task distribution technique. In this way, the student will be able to readjust the design decisions at an early stage of development, making it easier to study possible alternatives. For the chosen alternative at each step, the actual performance obtained with the corresponding execution will be compared with that shown by PARCSIM, in order to verify the accuracy of the simulations of this tool. Finally, the student will have to describe in the documentation accompanying the work the different design alternatives that were considered, the performance study shown by PARCSIM for each one of them, as well as the partial conclusions obtained in each case.

At the end of this first pedagogical experience with PARCSIM, in May, the authors will draw a set of conclusions based

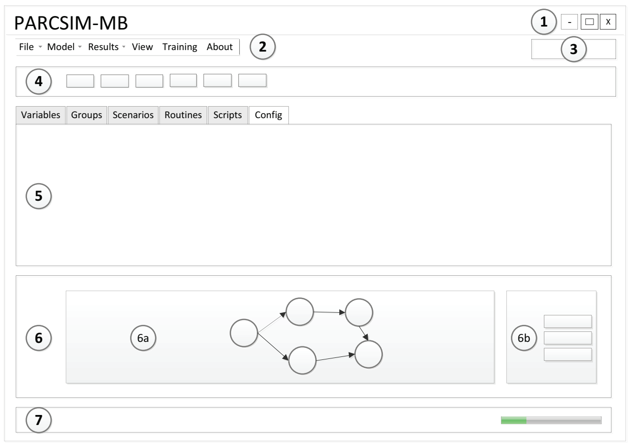

Fig. 1. Schematic view of the graphical interface provided by PARCSIM.

on the learning results obtained in comparison with those of previous years, as well as the opinions of students and teachers.

B. PARCSIM Interface

PARCSIM includes a user-friendly interface, which can ease the initiation of the students to the efficient exploitation of multicore+multiGPU nodes. Figure 1 shows a general outline of the graphical user interface, which consists of the following main elements:

1) Window buttons.
2) A menu bar with access to the software's functionalities with submenus.
3) A text box displaying the name of the parallel algorithm model.
4) A toolbar containing a selection of shortcuts to the main utilities.
5) The main work area, where the information of the algorithm model is captured, with its execution scenarios (size and type of its data, basically) and its set of adjustable parameters (for each function: library to be used, number of threads, number of GPUs...), and where the general configuration of the simulator is established.
6) A window to display the graph corresponding to the edited algorithm model. In this part of the interface, two areas can be distinguished:
    a) The graph itself, which is automatically updated when the user creates new algorithm elements or updates dependencies in the algorithm model.
    b) A toolbar containing shortcuts to utilities that allow to modify the display of the graph, such as rotating or resizing it.
7) A status bar showing information about the hardware and operating system on which PARCSIM is running.

III. Teaching Distributed Systems by Teaching Cryptocurrency: Challenge, Motivation, and Curriculum Design - Tseng

Boston College (BC) is *not* an engineering school; hence, most of the students in our department do not plan to become an engineer that builds or maintains a large-scale distributed systems. Most students take electives in their senior year; therefore, it is highly likely that students in my class do not have any experience in operating systems, nor computer networking. Moreover, many of them do not even have an idea what a distributed system is.

In this poster, I share the lessons I learned in teaching our students at Boston College.

A. Challenges

This section enumerates key challenges that I faced.

- Students are very diverse and majority of students do not view software engineers as a top career choice.
- Most students do not follow technical news and advanced development of exciting technology. For example, many students I taught have not heard about Blockchain and cannot identify high-level differences between different cloud computing paradigms.
- Many students only write code in class, and they are afraid to learn a new language or a new platform. Students are usually only comfortable with Python or Java.
- BC is focusing more on liberal art and science; hence, students take less core computer science (CS) courses.

In short, students are under-prepared for an intensive distributed systems course. These challenges make it extremely difficult to teach distributed systems in depth. Ideally, one should cover important theoretical topics in lectures and have students implement non-trivial distributed systems as machine problem assignments. However, in my experience, I would need to spend at least a few weeks to go over system programming, socket programming and proper communication abstractions to prepare students for such machine problems.

Consequently, I did not have enough time to cover more intriguing real-world systems and applications in my course.

Another difficulty is that students are not motivated while hearing examples and applications in large-scale systems. For example, they do not appreciate the important of consensus protocols that coordinate machines in the cloud. Two reasons I learned after multiple exit interviews with alumni are: (i) without a first-hand experience in coding sophisticated programs that interact with networks, students could not appreciate the necessity of dealing with unstable network and potential failures; and (ii) students are more used to software and applications that they can interact with (e.g., front-end or mobile apps), and have difficulties in imagining and grasping mechanisms under the hood.

B. A Solution: Cryptocurrency

One important lesson I learned is motivation.

Lesson 1: Motivate students with something they can see the importance.

While there are many exciting and important distributed systems, one system that I find a perfect fit is *cryptocurrency*. Most BC students are business-minded, and many of them indeed follow news in investment and stock market. With recent surging prices in cryptocurrency, students immediately see the importance and interest in this topic.

In addition, I enumerate key benefits of designing my course surrounding cryptocurrency:

- Cryptocurrency touches most important concepts in distributed systems: consensus, gossip, leader election, peer-to-peer systems, consistency, storage, replication, multi-party computing, security, PKI, digital signature, etc.
- It is easier for students to relate to the technical challenges. This is mainly because it resonates with their personal experience in using electronic transfer.
- Students are easier to see the adversarial challenges. To name a few, Byzantine miner, double-spending, message loss, and eclipse attack. Essentially, it resonates well with their mental models that someone may tamper the system to "steal money." Once they are comfortable in thinking of failures and attacks, they can gradually grasp the importance of ensuring reliability and correctness.
- Cryptocurrency contains many topics that are beyond CS, which are appealing to our students, as many BC CS students have a minor or even another major. Some relevant topics include privacy, game theory, economic analysis, and energy consumption, etc.

C. Lesson Learned

This section shares important lessons in teaching key concepts and skills in distributed systems by teaching cryptocurrency.

Lesson 2: Start with a simple abstraction, even if it deviates away from practical systems.

Most cryptocurrency is based on some kind of consensus and Blockchain protocols; however, those designs are aimed for performance or security, not understandability. Therefore, I present my own abstraction of a cryptocurrency, as illustrated in Figure 2. The outer circle denotes a node (or an entity) in the system, which has two layers: user layer and system layer, and interacts with other nodes via a message-passing network. The abstraction focuses on the behavior of each account, especially on one variable $\textsc{Coin}(A)$, the amount of coins that currently owned by node A. A derived variable $\textsc{Balance}(A)$ returns the number of coins in $\textsc{Coin}(A)$, i.e., the size of $\textsc{Coin}(A)$. These two variables are included in red circle in the figure. The goal of a cryptocurrency system is to support operations that transfer coins between $\textsc{Coin}(A)$ and $\textsc{Coin}(B)$ for nodes A and B in the system. Hence, we define and describe our abstraction centered around $\textsc{Coin}(A)$.

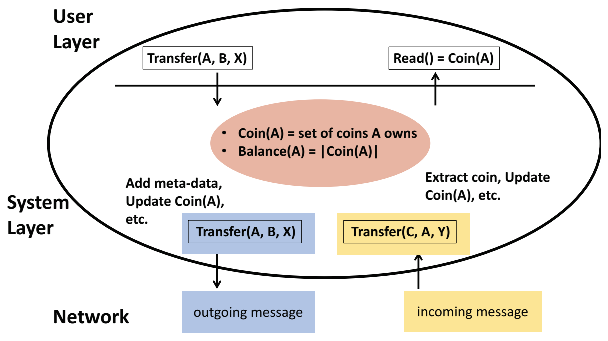

Fig. 2. **Abstraction of a Cryptocurrency System.**

As shown in the user layer in Figure 2, a user interacts with the system via two APIs:

- **Transfer**: operation $\textsc{Transfer}(A, B, X)$ is invoked by sender A, which transfers a set of coins X to receiver B.
- **Read**: operation $\textsc{Read}()$ invoked by node A returns the *current* set of coins that A owns.

The current set of coins takes account of (i) the initial balance at node A, (ii) all the prior outgoing transfer by node A, and (iii) all the incoming transfers that have been "applied" by the system layer of node A.

The system (or middleware) layer implements the logic of exchanging coins and maintaining $\textsc{Coin}(A)$. Typical Blockchain realizes this layer using a consensus protocol.

Lesson 3: Start small and build one component at a time.

With a simple abstraction, students can build each component independently. Ignoring the performance issue, a cryptocurrency roughly has the following three major components: hash pointer (for chaining the Blockchain), computation puzzle (for mining), and gossip (for exchanging transactions).

Lesson 4: Do not use Python.

As our students do not have extensive programming experience, Python provides too much freedom and too less structure. As a result, students implementations are usually all over the place, and it is extremely difficult to integrate different components together. Moreover, socket programming in Python is very challenging. Finally, the cryptographic library in Python is not well supported or documented.

D. Curriculum Design

I plan to use the following curriculum in the next offering, which is designed for students with basic programming experience and minimal experience in OS and socket programming.

- Introduction & Bitcoin Overview
- Introduction to Systems & Networking
- Multi-Thread programming & Fault-tolerance
- Cryptography
- Data structure & Mining
- Consensus and Consistency
- Put everything together

Each topic will range from one to one-and-half weeks depending on the student's background and experience.

I plan to use Golang as it has a very good support for networking and cryptographic tools. Its syntax is easy to learn, and it comes with a good official tutorial and playground. Plus, it has some strict rules so that students are not as easy to introduce obvious errors. Finally, a rather radical approach is getting rid of the TCP communication entirely, and replacing it with a "simulated channel." In particular, in the first few machine problems assignments, I will use multi-threading to "simulate" different nodes. Golang has a very intuitive Goroutine (a lightweight thread) and Channel (communication between Goroutines) that students can easily exchange messages between Goroutines without dealing with the complication of socket programming, serialization/deserialization, and error handling. Once students become more comfortable with the concept of message-passing, I can gradually introduce socket programming and replace the Channels in Golang by TCP or RPC protocols.

IV. Evaluation of web-based tools for beginners in AVX and MPI parallel programming education - Akiyoshi Wakatani and Toshiyuki Maeda

Part of this research was supported by JSPS KAKENHI Grant Number 18K02920 and 19K03018. This research was also supported in part by MEXT, Japan.

We are currently developing a web-based educational system for beginners of parallel programming such as CUDA, AVX, OpenMP, POSIX threads, MPI and OpenACC. The educational systems for CUDA, OpenMP and POSIX thread systems are completed, and the other systems are in progress. The current version of this system is located at http://pplinux2.is.konan- u.ac.jp/rccE/ppindex.php. This poster shows systems for MPI and AVX, and we also confirm the effectiveness of our system with automatic scoring function by experiments for the education of both AVX and MPI.

Our system consists of two parts. The first part is web contents (pseudo video lecture) that is generated from a PowerPoint file by STORM Maker software, and the second part is programming experiences for syntax issues and grammatical issues. The second part consists of three types of practices: debugging practices (grammatical issues), output estimation practices (semantic issues) and fill-in-the blank questions (both issues). The second part is created by PHP language and utilizes the parallel programming environments executed in the web server.

An experiment for the education of AVX programming consists of the two groups (a normal class and a flipped class). According to the results of 16 university students, there is no apparent difference between the two classes, so if the pre-learning of the flipped class is done at home in advance, students of the flipped class can carry out more programming projects than the normal class. An experiment for the education of MPI programming consists of the following four parts: 1) Learning as a preparation (about 30 minutes), 2) Pre-test (5 minutes), 3) Practices (fill-in the-blank question (20 minutes)), and 4) Post-test (5 minutes). Since the results of the post-test of 4 people among 6 people are superior to or equal to the pre-test, the effectiveness of our web tools (fill-in-the blank question) is confirmed about enhancing the ability of MPI programs.

We completed the educational system for 5 parallel programming environment and are currently developing the system for OpenACC. In the near future, we will complete the total educational system for 6 parallel programming environments, and will evaluate the effectiveness of the system by practically applying the system to university courses.

A. Parallel Programming For Beginners

Along with the progress of semiconductor technology, the integration of transistors on a chip has been dramatically accelerated, and then recent processors contain many processing cores. Especially, GPUs (Graphic Processing Unit) contain more than 5000 cores on a chip, and such parallel systems are utilized not only for graphic applications but also for deep learning applications and large-scale numerical simulations. Parallel systems are classified into two types: one is a shared- memory system, where OpenMP is used for multiple processing cores and AVX (Advanced Vector Extensions) is used for SIMD (Single Instruction Multiple Data) units, and another is a distributed-memory system, where MPI (Message Passing Interface) is frequently used. GPUs have two features: a shared-memory system and a distributed-memory system, since the memory of the GPU is shared by the GPU cores while the memory of the CPU and GPU are distributed. Users need to pay close attention to these two characteristics when developing GPU programs using CUDA and OpenACC. Therefore, parallel programming is an important skill for today's programmers, but it is not easy for beginners to acquire such a skill.

In this poster, we develop web applications for the education of parallel programming using AVX and MPI for beginners, and our web applications include an automatic exercise generation and an automatic scoring feature. Such a system is developed by using PHP language based on our previous system, and we classify programming exercises into the following three types: a) debugging practices (syntax practices) for understanding the grammatical issues, b) output estimation practices (semantics practices) for understanding the flow of a program and c) fill-in-the-blank question (integrated practices), and then we construct a web-based application that automatically generates lots of programming exercises by using program templates and a scripting language.

B. Web Tools For Education of Parallel Programming

Our previous literature studied programming language education. Similarly, the core of the education of parallel programming of course is the grammatical issues such as many APIs and their usage. Firstly, we provide a debugging practice as syntax practices that learners remove bugs from programs having bugs about grammatical issues such as APIs. Secondly, for writing correct programs for parallel computers, it is important to learn SIMD programming and programs with data transfer between computing nodes by using MPI functions. To enhance such an ability, we provide output estimation practices as semantics exercises that users guess the results of data transfer in an MPI program and the results of SIMD computation without executing it. Lastly, we also provide an integrated practice that contains both syntax issues and semantics issues in order to enhance the total ability of parallel programming. The integrated practice is a fill-in-the- blank question, so it is easy for users to repeat the practice. This practice is very simple and easy, so it should be carried out before syntax and semantics practices.

C. Experiments

1) AVX programming: Our experiment for the education of AVX programming consists of the following two groups: 1) Normal class (9 people): 90-minute lecture and 90-minute programming project, 2) Flipped class (7 people): 30-minute pre-learning by using web contents (video lecture) generated by STORM Maker, 60-minute programming practices by web tools and 90-minute programming project. We compare the results of the programming project.

Web contents using PowerPoint are generated as the pre-learning, which include the overview of basic parallel programming, the usage of MPI functions and the data transfer example. The web contents are generated by STORM Maker. Namely, the speech text about each slide of the web contents is described in the note part of the PowerPoint file, which is automatically transformed into the audio data by the speech synthesis feature.

According to the results of 16 university students, there is no apparent difference between the two classes, so if the pre-learning of the flipped class is done at home in advance, students of the flipped class can carry out more programming projects than the normal class.

2) MPI Programming: Our experiment for the education of MPI programming consists of the following four parts: 1) Learning as a preparation (about 30 minutes), 2) Pre-test (5 minutes), 3) Practices (fill-in the-blank question (20 minutes)), and 4) Post-test (5 minutes).

The pre-test and the post-test ask about the usage of APIs as syntax issues and the results of the calculation of process ID (rank) of data transfer functions as semantics issues. In particular, the syntax issues mainly contain misspelling and misunderstanding of MPI's APIs, such as the name of include files and the name of functions. The difficulty level of the pre-test and the post-test is the same, and half of the test is about syntax issues and the rest is about semantic issues that users answer the process ID of the source and the destination.

Since the results of the post-test of 4 students out of 6 students are superior to or equal to the pre-test, the effectiveness of our web tools (fill-in-the blank question) is confirmed about enhancing the ability of MPI programs. It should be noted that the wrong answers to the test are mostly semantic issues. It is also confirmed that understanding of process ID is hard for beginners.The 6 university students (age: 21 or 22), have already studied fundamental C language once.

This experiment utilizes only the fill-in-the-blank question due to the limitation of the time of the experiment, and the number of evaluators is also limited. If the debugging practice and the output estimation are included in the contents of the web tools, better effectiveness may be acquired. Our future work includes the experiment using all the practices with more students.

D. Conclusion

We classify programming exercises for learning parallel programming such as AVX and MPI into the following three types: a) debugging practives (syntax practices) for understanding usage of APIs, b) output estimation practices (semantics practices) for calculation of thread ID and program understanding, and c) fill-in-the-blank question (integrated practices). To realize an experimental system, we construct web-based applications that easily and automatically generate several types of programming exercises by using template programs and a PHP scripting language. By our experiments, we also confirm the effectiveness of the systems for both AVX and MPI, which consists of both an electronic materials and our web-based applications, for learning practical parallel programming.

In the future, we will evaluate our proposed system and e-learning contents to provide practical learning systems for other parallel programming. We will also extend our approach to different programming environments such as data science environment in order to contribute to education of programming and software.

V. Final Remarks

The CDER website (grid.cs.gsu.edu/ ˜ tcpp/curriculum/ provides access to the full submitted posters, as well as to all materials presented at this and previous workshop.