A Curricular Experience With Parallel Computational Thinking: A Four Years Journey

Edusmildo Orozco, Rafael Arce–Nazario, José Ortiz–Ubarri, and Humberto Ortiz–Zuazaga
Department of Computer Science
University of Puerto Rico–Río Piedras
Emails: edusmildo.orozco1@upr.edu, rafael.arce@upr.edu, jose.ortiz@hpcf.upr.edu, humberto.ortiz@upr.edu

Abstract—**In this article we document the process by which we inserted parallel computational thinking throughout the undergraduate computer science program at the University of Puerto Rico–Río Piedras. Our target audience is faculty and other stakeholders from CS departments similar to ours that might be interested in pursuing a similar endeavor. We discuss our initial motivations, the curricular redesign strategy, results, and experiences, as well as the educational resources created along the way.**

I. Introduction

It is a commonly accepted fact that, in the foreseeable future, sustained computational performance and reliability will be increasingly dependent on exploiting the parallelism available throughout the whole spectrum of computing platforms, e.g., from embedded systems to high–end multi–core servers [1]. The implications of this reality to CS education are evident: the ability to design effective solutions by means of parallel and distributed computing should be a required competency for all CS undergraduates.

However, up to now, traditional undergraduate CS programs have relegated their teaching of parallel computational thinking (PCT) to advanced level or even elective courses. We share the opinion articulated by some CS experts that the importance and complexity of PCT merits its introduction early and throughout the undergraduate CS curriculum [2], [3].

Our team is finishing its fourth year of a project titled "Asserting Parallel Computational Thinking into Undergraduate 4–year Computer Science Curriculum" to enhance our undergraduate CS curriculum with PCT. We define PCT as a set of competencies that are essential for CS students to thrive as professionals in a field that is increasingly reliant on parallelism to sustain performance/reliability in applications (see Section II).

To the best of our knowledge, projects for enhancing CS curricula through parallelism date as far back as 1992 [4], [5]. A few undergraduate programs in universities similar to ours seem to have embraced parallelism as an essential component throughout their undergraduate programs [6]. Nevertheless, their curriculum redesign and evaluation processes have not been properly disseminated to allow adoption by others. The reality and composition of every CS department is unique, which implies that even the adoption of curricular ideas from others requires at least a structured process of self–assessment and adaptation. In the process of improving our curriculum we are applying sound and tested educational procedures (i.e., the backward curricular design method). Additionally, we are documenting our progress and making all materials and results available to the public [7], so that others can collaborate with us, adopt or adapt the components of this work that they feel applicable to their curricula, and even learn from our experiences. In particular, the University of Puerto Rico (UPR) system has six programs that can directly benefit from PCT immersion: four CS programs, one in computer engineering, and one information systems program. All but one (which began on 2013) are ABET–accredited. Additionally, there are two ABET–accredited computer engineering programs in private universities in Puerto Rico. One of our goals is to establish collaborations with these and other stakeholders, to share our experiences, methods, results, and materials to instill PCT throughout their CS curricula.

Our efforts at UPR–Río Piedras

After going through a profound process of curricular revision to comply with ABET accreditation standards, university regulations, and a self–commitment to excellence, the Computer Science Department at UPR–Rio Piedras (UPR–RP) decided to take advantage of the momentum and requested funding from the NSF CPATH program to permeate our entire undergraduate curriculum with PCT. Our project, "Asserting Parallel Computational Thinking into Undergraduate 4–year Computer Science Curriculum", was approved in the third quarter of 2009, and since 2010 our curriculum has undergone meticulous scrutiny looking for opportunities where PCT can be instilled. Back in 2010, we knew that several institutions [8], conferences [9], and interest groups [10] were actively discussing ways to approach the problem of the lack of PCT in the undergraduate curriculum but no agreement about PCT standard content, methodologies or outcomes were in place. We started our endeavor with the goal of ending up with a stronger undergraduate curriculum emphasizing PCT and with the aim that our case could be replicated or at least considered by other CS departments similar in size and in composition to ours.

Almost two years ago, we presented our work in progress and plans for completion in [11]. In the present article we discuss our experiences and provide a description of the complete process of curriculum redesign, the courses impacted and the academic resources created/adapted.

The remaining of this paper is organized as follows. Section II describes our operational definition of PCT and the main competencies focused by our efforts. In Section III we briefly describe our CS undergraduate program and our initial motivations towards PCT. Section IV summarizes the educational methodology used for the curriculum redesign process. Section V presents brief discussions of the major educational activities and findings from the remaining six PCT redesigned–and–deployed courses. In Section VI we discuss our lessons learned, challenges, and opportunities, and make our own recommendations. Section VII features three modules developed to be used with the LittleFe educational cluster. Lastly, Section VIII presents our conclusions and future endeavors.

II. Defining and focusing PCT

We envision PCT as a specialization of the concept of Computational Thinking (CT) presented in [12]. PCT is inherently conceptual rather than a programming tool or technique. To guide our efforts, we define PCT by establishing the main competencies we would like to instill in our students:

(1) The ability to abstract a solution as concurrent activities. This skill goes beyond mere hierarchical and modular abstraction since it involves consideration to issues such as communication, synchronization and use of common resources.

(2) The skills to identify opportunities for parallelization in task processing and data dependencies.

(3) The capacity to understand advantages of parallelism at different abstraction levels. For example, in computer systems the abstraction levels can be hardware, system architecture, operating systems, user applications, and user data. The ability to take advantage of properties intrinsic to these levels in order to materialize the benefits of the parallel solution is a very important skill.

(4) The ability to effectively express ideas about parallelism in written and in verbal form. The ability to express PCT concepts in a theoretically sound manner as well as in more intuitive ways is critical to the dissemination of PCT concepts.

III. Our curriculum

The Computer Science Department at UPR–RP is composed of nine full–time faculty members, all actively involved in various fields of research, and support personnel. We offer a 4–year bachelor of science since 2002. As of this writing, fall semester of academic year 2013–2014, there are 108 undergraduate students enrolled in our program. Our programs' computer science course sequence is illustrated in Figure 1. The students take a total of 17 CS courses, 14 of which are mandatory, and three CS advanced elective courses. Within the College of Natural Sciences, our department prides itself with a high percentage (25%) of students involved in undergraduate research.

During years 2007–08 our program underwent accreditation by the American Board for Engineering and Technology (ABET). The process of preparation to comply with the accreditation guidelines helped us focus our assessment strategies and gathered valuable data regarding program performance and identified areas for improvement. There was a unanimous consensus among members of our faculty that we should take advantage of the momentum created by the ABET accreditation process to further improve the quality and pertinence of the entire program.

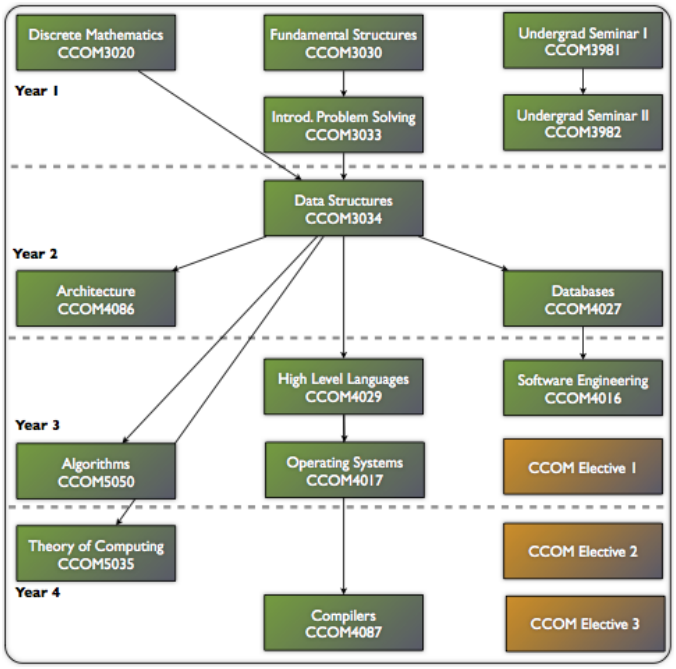
Discrete Mathematics CCOM3020
Fundamental Structures CCOM3030
Undergrad Seminar I CCOM3981
Year 1
Introd. Problem Solving CCOM3033
Undergrad Seminar II CCOM3982
Data Structures CCOM3034
Year 2
Architecture CCOM4086
Databases CCOM4027
High Level Languages CCOM4029
Software Engineering CCOM4016
Year 3
Algorithms CCOM5050
Operating Systems CCOM4017
CCOM Elective 1
Theory of Computing CCOM5035
CCOM Elective 2
Year 4
Compilers CCOM4087
CCOM Elective 3

Fig. 1. Sequence of courses in the undergraduate curriculum of the CS Department at UPR–RP.

In our reflection, PCT stood out as a set of competencies which, although they were of paramount importance, were not sufficiently emphasized in our curriculum. As a result of the traditional course descriptions and the way that the curriculum is structured, students could graduate from the program without ever being explicitly exposed to PCT concepts.

An elective course in parallel processing had been offered twice in the past four years. Most of the students who enrolled were in their fourth year, and enrollment averaged six students. The instructor, Dr. Orozco, observed that, when faced with a parallel processing problem, students automatically defaulted to a sequential solution before even considering the existence of a parallel counterpart. Sequential thinking was deeply entrenched in our fourth year students minds. At this stage in their course sequence, it was too late to remedy such reasoning with a single parallel programming elective course.

IV. Methodology and Tools

The initial stages of our project convened most of the CS program professors in several sessions to reflect upon the meaning of parallelism/concurrency and its pertinence to their courses. The Backward Design (BD) curricular methodology was recommended by our educational consultants to help us in the course/curriculum redesign [13]. This model proposes three stages for curricular restructuring: (1) identify the desired educational objectives for the students, (2) determine acceptable evidence, and (3) plan learning experiences and instruction. Each professor that had committed to introducing PCT concepts related to his/her course was asked to fill a questionnaire with guide questions that allowed him/her to identify course conceptual contents, skills, and attitudes related to PCT. Some of the questions contained in the questionnaire were: (a) what are the big ideas of the course?, (b) what is the knowledge that students will learn?, (c) what knowledge and skills are required to develop the desired understanding?, (d) what evidence should be collected to determine whether or not the understanding has been developed, the knowledge and skills attained, and the standards met?, and (e) what sequence of learning and teaching activities will enable students to perform well?

BD emphasizes prioritization of knowledge, skills and attitudes early in the design process. The instructors were asked to classify these into one of three levels of curricular emphasis (i.e., mastery levels as in Bloom's taxonomy [14]): (1) essential to learn ("big ideas"), (2) important to know and do, and (3) worth being familiar with. In our opinion, this exercise is one of the most important reflections that professors should perform when planning the addition to or modification of any subject of a course. It helped us put into perspective the importance of the PCT concepts in comparison to the rest of the course content, prioritize the specific PCT concepts to introduce, and it simplified the decision as to which 'traditional' course concepts were to be given less attention once the PCT changes were introduced, e.g., the victims of the zero–sum game.

Besides helping decide course–specific actions, the faculty reflections and BD–driven sessions helped us draw a strategic mapping of the various concepts onto the core courses. Figure 2 illustrates the comprehensive PCT–infusion plan as a course vs. concept matrix. The main purpose of this plan is to ascertain that all of the PCT key concepts are covered in at least one of the core courses and if possible, concepts are presented from different perspectives and levels of difficulty, e.g., concept follow–through. For instance, the basic concept of a thread is discussed from the perspective of multi–threading and multi–core processors as early as in the "Fundamental Structures" course. The concept is followed–through in the "Problem Solving with Programming" course when teaching students how to parallelize for–loops, and in the "Data Structures" course when students learn how to create and synchronize threads to implement multi–threaded versions of divide–and–conquer algorithms. Finally, notions related to operating systems such as thread creation, scheduling, and management are reinforced in the "Operating Systems" course.

Another important consideration to take into account when assigning a PCT concept to a course was the ease of adaptation of the concept into the existing content of a course. Many of the key themes in PCT are easily relatable to "traditional" content. For instance, discussion of repetition structures in an introductory programming course presents a smooth transition into data dependency and opportunities for parallel processing. In some cases, the parallelism is even already in the traditional content, albeit in an implicit manner. Thus, the educators job becomes making the parallelism explicit in their discussions and teaching activities. For instance, the discussion of adders (ripple carry vs. carry lookahead) in the Computer Architecture course can be used to emphasize the tradeoffs in speedup vs. resources.

The specific themes and activities covered in each course are part of the materials available online at our project's website [7].

CS courses	1. Data-parallel model	2. Moore's Law	3. Task-graph model	4. Shared-memory	5. Amdahl's law	6. Threads	7. Task decomposition	8. Data decomposition	9. Synch & asynch tasks	10. Distributed-memory	11. Message-passing	12. Master-slave	13. Pipelining
Fundamental structures	x	x	x			x		x	x				
Prob. solving with program.			x	x			x	x				x	
Data structures	x			x		x		x	x				
High level languages			x				x		x				
Computer architecture	x	x		x	x	x	x			x	x		x
Operating systems				x		x		x	x	x	x		
Introduction to databases	x		x				x	x	x	x		x	
Algorithms	x	x	x	x	x	x	x	x					
Parallel processing	x	x	x	x	x	x	x	x	x	x	x	x	

Fig. 2. Pairing of PCT key concepts to core courses. The shaded courses are discussed in [11].

V. Redesign and course deployment

Ideally, the order in which PCT conversion affects core courses should follow the course sequence. Otherwise, in advanced courses, the student levels of PCT exposure will be too disparate and the professor will spend precious class time explaining elementary concepts. In our project, the initial sequence for the PCT conversion of core courses was dictated by the availability of instructors. Thus, instead of converting the elementary courses first and then proceeding to intermediate and advanced courses, we had to start with the courses that had been assigned to the principal investigators of this project and then expand to courses taught by other faculty members. For the redesign of each of the targeted courses, the professor identified to lead the redesign of a course met several times with the educational consultants to discuss the BD and assessment tools. From this, the main outcomes were: a plan to interweave PCT concepts from Figure 2 into the course, the identification of the previous content that will be affected, syllabus changes, learning activities and assessment methods. Starting Spring 2013, all the redesigned courses had been taught at least once.

We proceed to briefly discuss the major educational activities and findings from the courses that have been redesigned and deployed since [11].

A. Fundamental Structures of Computer Science (CCOM 3030)

This course presents a panoramic view of different subjects of computer science and their interconnections. By its own nature, this course lend itself to introduce many PCT basic notions through activities as described in CS Unpluged [15]. A specific activity performed by the instructor was to simulate a parallel processor to convert a binary to a decimal number with the students in the classroom. This exercise served to illustrate the notions of divide–and–conquer, running time, load balancing, directed acyclic graphs (DAGs), and concurrency. Additionally, one lecture was dedicated to a discussion of DAGs and concurrency, and another to MapReduce. Pre/post tests about the discussed concepts were offered at the beginning and at the end of the semester. Among the salient results that will require our attention on future offerings of this course are: (1) one lecture (without student practice) was not enough to grasp a strategy like mapReduce, and (2) although most students were able to express instruction dependency using a DAG, the concurrency notions that can be extracted from the DAG were not as evident.

B. Introduction to Problem Solving with Programming (CCOM 3033)

This course provides the basic tools for students to learn problem solving with programming through the use of computer algorithms. Initially, this course was designed to expose students to the use of control and loop structures, functions, recursion, basic data structures and objects, with a sequential focus even neglecting natural parallelism that comes from, say divide–and–conquer algorithms.

In order to assert PCT in CCOM 3033, in each lecture we included the construction of a DAG to highlight data dependencies and concurrency among instructions. Our aim was to start developing the necessary skills to identify opportunities for parallelization from an early stage of the course. In the loops topic, students were also exposed to divide–and–conquer and map–and–reduce techniques, which, at the same time, generated discussions on race conditions and synchronization. The lectures included examples, figures, and in–class exercises, besides bringing the laboratory experience on how to create and manage threads using the Qt C++ framework [16].

A pre/post test was developed to assess the student performance. With an average score of 1.5 out of 8, the pre–test showed what was already expected: most of our students in the introductory class do not have a clear idea on how to solve problems computationally, even less an idea on how to solve problems via PCT. However, the post–test results (average score 6.7 out of 8) are quite encouraging in that they reaffirm that PCT knowledge can be included and assimilated at a basic level. We anticipate that introducing PCT from an early stage will help change the current behavior of senior students in which they automatically default to sequential solutions when faced to problems that can be solved in parallel.

C. Operating Systems (CCOM 4017)

Unlike most courses in the undergraduate CS curriculum, Operating Systems (OS) has always included topics implicitly related to PCT, e.g., interprocess communication, race conditions, concurrency, deadlocks, synchronization, shared memory and threads. Furthermore, our current textbook includes a chapter on multiple processor systems, which discusses more advanced topics related to PCT [17], e.g., multiprocesors, multicomputers, multi–cores, distributed systems, distributed memory, computer clusters, and scheduling multiprocessor systems. For this reason, there was no need to add much new material to the OS course, but rather reorganize the lectures to make PCT explicit. Instead of devoting most of the course discussion to operating system concepts from a single processor system point of view, the content on multiprocessor systems, which is usually left for the end (as an extension of single–processor systems) was merged with the regular content of the course. Thus, multiprocessor systems become an integral concept throughout the course rather than an isolated topic. The new topics integrated to the course as an extension of classical OS topics were: current distributed file systems (DFS), included in the discussion on file systems, and security in distributed systems which was added in the discussion about OS security. The topics were presented through lectures with examples and case studies; two lab experiences in which students had to create and manage threads, and use divide–and–conquer to solve a specific problem; one class project to simulate multiprocessor scheduling; and one oral presentation discussing a paper related to DFS.

A pre/post test was developed to assess student performance. The average score in the pre–test was 3.4 out of 8, and the average score in the post–test was 5.2 out of 8. Although the post–test shows some improvement over the pre–test, it was not as significant as we were expecting. Further analysis of the scores showed that most students failed to provide correct answers to questions about topics discussed earlier in the course. The reason for this could be that the post–test was administered late in the semester and without previous notice. Similar questions to those in the post–test were used in regular exams and the students seemed to have performed better. Thus, we conjecture that most of the students do gain an aptitude for the topics once they have had time to reflect and practice outside the classroom activities.

Two new projects were designed to improve the results obtained on the first assessment. The first project was implemented in Fall 2012. The idea was to give students a final project where they were encouraged to refresh the PCT material learned across the course, and to provide them with a nifty assignment of one of the most important topics of the OS course whose PCT material was currently the most time limited: File Systems. In the previous year the students learned about DFS through a presentation related to research or a particular implementation. In Fall 2012, instead, students learned the material through assigned article readings and also with the implementation of a simple DFS using Python. This DFS consisted of a metadata server, a collection of data nodes, and a client application to copy files from and to the DFS. In order to copy a file to the DFS, the client asks the metadata server for a list of available data nodes to distribute pieces of the files, then it establishes connections to the data nodes and transfer their pieces of the file. If the copy is successful, the client sends an update message to the metadata server so that it can keep track of how the file was distributed. To copy the file from the DFS, the client asks the metadata server for the file storage information, and then the client asks the data nodes for the file pieces and merges them into the original file. This project provided opportunities for the discussion of data replication, system availability, and fault tolerance.

The project was assigned to collaborative teams of 4 students. With the aim of integrating balanced teams, groups were selected based on their course performance and Python programming skills. At the beginning, a few members of some teams were hesitant about the complexity of the work. However, at the end every group completed the project and provided positive feedback: “this project was a bit challenging but awesome” and “it should definitely be assigned again.”

The second project was designed as a preparation for their final DFS project. The most frequently expressed concern by the collaborative teams on the DFS project was their lack of previous exposure to socket programming. Although all the teams overcame their expressed limitations, the instructor of the course decided to modify one scheduling project from the simulation of a scheduler of a multiprocessor system, to be a scheduler of jobs submitted by resource constrained mobile devices with the purpose of exposing the students to hands–on socket programming early in the course. The intention was to include socket message–passing with the topics already covered by the project: synchronization and race conditions.

D. Introduction to Data Management (CCOM 4027)

This course provides third or fourth year students the concepts of data management mainly through databases. Here they learn about relational databases as well as the SQL language. To assert PCT, three lectures (4.5 hours) were included to present distributed, parallel and shared databases including architectures that will allow for a better scaling of a parallel or distributed database system. We also presented concepts in parallelizing queries using query plans taking into account the various data partition schemes.

E. Computer Architecture (CCOM 4086)

The Computer Architecture course has traditionally touched upon many aspects that are part of the PCT vocabulary: pipelines, threads, and speedup. To permeate it with PCT, the strategy being followed is to make explicit how parallelism can be exploited at the many levels of a computer system: from the circuit level, through the support by the instruction set and performance enhancing techniques such as multithreading and pipelining. Some activities in support of PCT learning are:

- Students implemented ripple carry and carry lookahead adders using LogiSim [18] and were asked to compared the pros and cons of each in terms of performance and resource utilization.
- Students implemented a single–cycle simplified MIPS processor (also using LogiSim), and compared the single–cycle design with the design of a pipelined version.
- Expanding on a thread and mutex exercise done in CCOM 3033 students learn the hardware support in MIPS for atomic operations (llw, load linked word and scw, store conditional word) and implement a simple mutex. Later, they are asked to design hardware support to implement one of these instructions.
- The instructor gave a presentation on GPUs, the CUDA programming model, and demonstrated programs adapted to take advantage of data parallelism on the GPU platform.

The first two activities are traditionally taught as part of the course and thus it was only required that the instructor highlight the parallelism and its importance even at the lowest levels of abstraction. Only four of seven students were able to satisfactorily complete parallel carry lookahead exercise in Spring semester 2011. We will provide more guidance to students this semester and try again. The third and fourth activity were new. The third activity is specifically designed to build on PCT content in a prior course, in consultation with faculty in both courses. In Fall 2012, the first time we gave the assignment, only 21 out of 38 students were able to successfully design a circuit for scw. This assignment will be redesigned to give the students more guidance. The fourth activity is a new addition to the textbook used in class, and the time allotted during its first implementation (1.5 hours) was found to be too short to develop any learning depth beyond the first two levels of Blooms' taxonomy, e.g., remembering and understanding. A CUDA programming lab experience is planned for this semester, to supplement the lecture.

F. Design and Analysis of Algorithms (CCOM 5050)

This course introduces undergraduate students to the notions and techniques that are used as building blocks in the design and analysis of algorithms. In order to permeate our Algorithms course with PCT we decided to include dynamic multithreaded programming (DMP) as is treated in [19]. The main reasons to choose this theme were: (1) it can be presented as a simple extension of sequential programming by adding a few key words and, (2) parallel performance analysis can be achieved using the same mathematical tools except by adding the notions of critical path, work, and span. A natural place to include DMP material was right after divide–and–conquer fundamentals. A total of two lectures (3.0 hours) were devoted to include the following topics and corresponding activities: DMP as an abstract model of parallel programming, concepts of work, critical path, span, and parallelism; extension of a suitable sequential algorithm to a parallel one by inserting key words spawn, sync, and return into the appropriate lines. Details of assessment, objectives, examples, activities, and exercises can be found in [7]

VI. Lessons learned, challenges and opportunities

As a growing department, one of the challenges we encountered was incorporating new faculty hires into the project. The newcomers missed the essence of the start of the project. They did not participate in all meetings, discussions, and exchange of ideas that enriched the conception and development of the whole idea. More fundamentally, they lacked the commitment acquired by the initial team. In order to deal with these and other issues we make the following recommendations:

(1) Train the trainer: carefully mentor and engage all instructors that will be in charge of courses related to the project. In particular, as soon as a new faculty is hired, train her/him in the specific goals, methodologies, new topics, and activities that could speed–up the learning process. Actively inform them about available conferences or workshops relevant to the goals of the project (for instance, The SC Conference Series [9], the ACM Special Interest Group on Computer Science Education [10], and the National Computational Science Institute [20], to name a few). An incentive for the new hire is that attendance to these forums contributes to his/her professional development.

(2) Prepare hands–on material: avoid themes that are overly specialised, i.e., stay clear of topics/tools from your own research. Prepare material that requires as a pre–requisite only common knowledge and is easy to digest and adopt by others.

In Section VII we describe a set of educational modules that we developed for using the LittleFe computational cluster together with the Bootable Cluster CD (BCCD) [21]. These are publicly available through our project's website.

(3) Be ready: if you are part of the leading team, be prepared to take over in case you have to do the work yourself; sometimes instructors are not ready/willing to or not available for teaching with new approaches or new material, unless there is a mandate to do so (for instance, an accreditation).

(4) Curricular changes are painful: in our experience, making official changes to our curriculum has opened many opportunities as well as challenges. We have used the opportunities to improve and update the pertinence of our CS curriculum and, hence, the quality of our graduates. We also have faced challenges such as the zero–sum game and administrative burdens. For instance, even though all the planned core courses have been redesigned and taught at least once, only three syllabi are in the pipeline for official, campus–level approval.

(5) Engagement: Provide workshops and trainings in current trends. Invite guest speakers from the academia and the private sector to make your community see the theory come to life. For instance, we sponsored a three–day (12 hour) Cloud Computing workshop, where the participants studied the concepts, learned to use the common tools, and formed teams which proposed ideas and created prototypes for cloud–based applications.

(6) Acquire the right equipment: we had access to remote computational resources for education but it was too burdensome on authentication and validation of accounts to be used in the classroom. If possible, try to get your own equipment. Something in the line of the LittleFe [22] or building your own cluster with a fair budget will suffice [23].

(7) Textbooks: certainly there is a number of good books on PDC for stand–alone courses [24], [25], [26], they are targeted mostly for upper–division undergraduate or even for graduate level students. However, we, as educators facing the necessity of spreading PCT on the whole spectrum of the CS curriculum, need approaches such as the Book Project of CDER [27] together with contributions from a large portion of the academic and industry communities.

VII. Available Resources

In our opinion, one of the most practical contributions that our project provides to instructors of courses instilled with PCT is a list of hands–on modules developed to be used with the LittleFe cluster and this section is devoted to introduce them [28]. The rest of the resources developed as part of our project, i.e., presentations, pre/post tests, questionnaires, assignments, as well as curricular redesign documents are available through the website of our project [7].

Hands–on programming exercises and code demonstrations/experiments are an essential educational resource for teaching introductory parallel computing concepts. The design of effective and applicable exercises can be challenging to the instructor, especially since many libraries or frameworks for parallel programming require intricate system setups and training students in low–level details to get even the most basic programs to work. In our experiences, low–level details in parallel computation exercises lead, among other things, to student (and instructor) frustration and deter the students from focusing on the essential concepts.

Besides preparing modules for learning MPI4py and for setting up the LittleFe cluster with Disco [29], we also have designed a set of easy–to–deploy hands–on educational modules for the LittleFe educational Cluster, which can be used to introduce students to concepts such as: the map–and–reduce and master–slave models, load distribution, and scaling. The programming exercises use the Python MPI4py and Disco (map–reduce) libraries to significantly deemphasize attention to low–level details and allow time to complete solutions to more elaborate problems. Each module is constructed around a real–world or easy to grasp research problem to showcase the importance of parallel computing and to motivate the students. Our growing list of modules includes: two modules for teaching and practicing MPI distributed memory programming concepts and a module to practice MapReduce basics.

A. Module 1: Learning MPI distributed memory programming through Costas arrays

The first module uses the enumeration of Costas arrays to introduce the Master/Worker paradigm using MPI4py. The classic (non–parallel) solution to this problem uses a backtracking algorithm to list all permutations that meet the Costas property [30]. As expected, the time complexity of such a solution grows factorially with the size of the sequence. A parallel solution is easy to explain and implement: let the master compute all permutations up to a given position, then distribute these sub–permutations to be completed independently by the workers. Figure 3 shows the search of Costas sequences of size 6. The initial permutations represent a small portion of the total computation, thus parallelization achieves almost linear speedup.

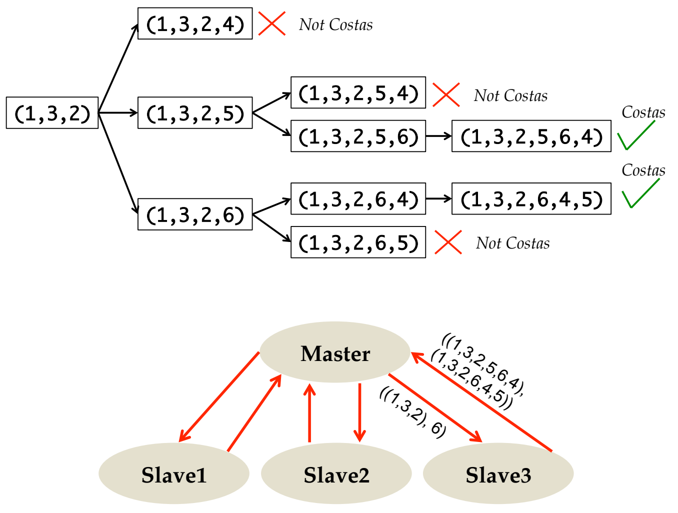

Fig. 3. (a) Computation of Costas arrays using backtracking, e.g., computational branch of the Costas arrays of size 6 that start with sub–permutation (1,3,2). (b) Master/worker diagram of Costas problem. Master sends sub problems to slaves. Slaves return the solution to the sub problems.

B. Module 2: Learning MPI distributed memory programming through password cracking

The second module uses *password cracking* as a the motivating application. Password cracking is a method by which an intruder that gains access to an encrypted passwords file can deduce the original (clear text) passwords using a dictionary and brute force, e.g., word in a dictionary is encrypted and compared to the encrypted passwords in the password file. We use the password cracking example to teach how to solve problems by dividing the problem in sub problems of approximately the same size among the available compute nodes. Figure 4 illustrates our approach: the complete password file is sent to all the compute nodes (MPI broadcast), and the dictionary file is divided in (approximately) equal parts and distributed among the compute nodes.

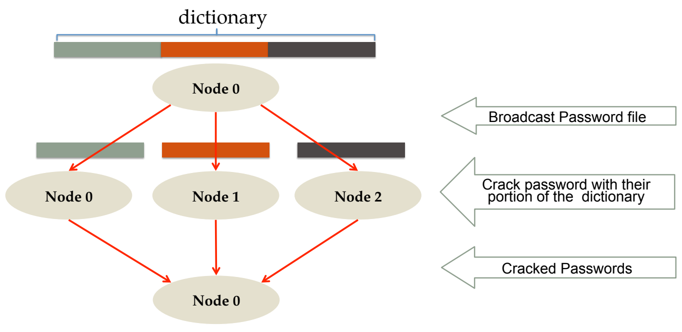

Fig. 4. Map/Reduce diagram of the password cracking problem. Node 0 broadcasts the password file to all nodes. The dictionary is divided in approximately the same size for each node. Cracked passwords are sent back to the master.

C. Module 3: Learning MapReduce basics through NetFlow data analysis

The motivating application for this module is network data processing. In computer networks, aggregation of traffic data per host helps to identify anomalies such as (1) an IP in use that is not delegated, (2) an IP that is generating more traffic than normal, and (3) an IP that is not generating traffic. NetFlow is a network protocol that aggregates connection information from one host to another over a certain period of time (flow). NetFlow files consist of large quantities of connection information such as IP addresses of the source and destination hosts, and the traffic in each connection (e.g., 5 minutes of UPR network traffic can be as high as 6.5MB, or 363,149 lines of flows). Thus, processing them takes considerable processing effort.

In this module, we use NetFlow data processing as an example of an application that can be accelerated using Disco MapReduce. Simplified code for this application is presented in Figure 5, wherein (1) the source and destination IPs are mapped by their addresses and valued by their traffic octets (2) the reducer gathers the sorted key–values and computes their aggregated sums.

VIII. Conclusions and Future Work

The idea of instilling PCT in the undergraduate curriculum continuum has been around for a while. However, its implementation is not trivial and deserves a systematic curricular treatment to succeed in the long run.

```
def map(line, params):
  # Split the data into an array:
  # data[0] is the source, data[1] is destination
  # data[5] is the octet

  data = line.split()
  yield data[0], int(data[5])
  yield data[1], int(data[5])


def reduce(line, params):
  for ip, traffic in kvgroup(sorted(iter)):
    yield ip, sum(traffic)
```

Fig. 5. Map and reduce functions for basic NetFlow data processing

The PCT–assertion strategy established by any department will be influenced by factors such as faculty member strengths and weaknesses, their availability for training other members in PCT educational strategies, and the existing curricular sequences. The CS education community can help by sharing their experiences and educational resources with programs that are in the early stages of adoption. Most importantly, we need the concert of a broader community effort such as the CS2013 task force and the NSF/IEEE–TCPP Curriculum Initiative to provide guidelines that every program in CS should instill in their students. We believe that our PCT project aligns us with the coming parallel and distributed computing standards.

What becomes of the elective Parallel Processing course once the PCT has been infused throughout the curriculum? Currently it is offered once a year and will remain as an elective until "official" standards about PCT are released. Our department has considered the strategy of transforming it into a capstone–like course that includes advanced topics not taught on other courses and that summarizes knowledge and skills as PCT mini-projects. Besides deciding the fate of this course, our future work includes establishing collaborations with other CS/IT programs in Puerto Rico, to share our experiences, methods, results, and materials to instill PCT throughout their own curricula. Furthermore, we will keep improving and adding to our educational resources.

Acknowledgments

The project "Asserting Parallel Computational Thinking into Undergraduate 4–year Computer Science Curriculum" is funded by the NSF–CPATH award Number 0938995. The educational modules discussed in Section VII were developed as part of the requirements for the recipients of the LittleFe educational computer cluster. The last author is partially supported by the NSF–EPSCOR award Number 1010094. We would like to thank all members of our faculty, students, and administrative personnel for their enthusiasm, support, and commitment to the PCT project. We would also like to thank the former PI of this project, Dr. Peter Musial, for his initial commitment to the project.

References

[1] H. Sutter, "The free lunch is over: A fundamental turn towards concurrency in software," *Dr. Dobbs Journal*, vol. 30, no. 3, pp. 202–210, 2005.

[2] L. Ivanov, H. Hadimioglu, and M. Hoffman, "A new look at parallel computing in the computer science curriculum," *Journal of Computing Sciences in Colleges*, vol. 23, no. 5, pp. 176–179, 2008.

[3] S. K. Prasad, A. Gupta, K. Kant, A. Lumsdaine, D. Padua, Y. Robert, A. Rosenberg, A. Sussman, C. Weems *et al.*, "Literacy for all in parallel and distributed computing: guidelines for an undergraduate core curriculum," 2012.

[4] M. J. Meredith, "Introducing parallel computing into the undergraduate computer science curriculum: a progress report," in *ACM SIGCSE Bulletin*, vol. 24, no. 1. ACM, 1992, pp. 187–191.

[5] C. H. Nevison, "Parallel computing in the undergraduate curriculum," *Computer*, vol. 28, no. 12, pp. 51–56, 1995.

[6] School of Computing, Informatics, and Decision Systems Engineering. (2011) Intel at ASU . [Online]. Available: http://cidse.engineering.asu.edu/outreach/intel-at-asu/

[7] UPR–RP CPATH Project. (2013) Asserting parallel computational thinking into undergraduate, 4–year computer science curriculum. [Online]. Available: http://knuth.uprrp.edu

[8] Carnegie Mellon, Computer Science Department. [Online]. Available: http://www.csd.cs.cmu.edu/education/bscs/

[9] Supercomputing Conference 2012–HPC Educators Program. [Online]. Available: http://sc12.supercomputing.org/content/hpc-educators

[10] ACM Special Interest Group on Computer Science Education. [Online]. Available: http://www.sigcse.org/

[11] E. Orozco, R. Arce-Nazario, P. Musial, C. Lucena, and Z. Santiago, "Asserting parallel computational thinking into an undergraduate computer science curriculum," in *Computer Science Education: Innovation and Technology CSEIT*, 2011.

[12] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[13] G. P. Wiggins and J. A. MCTIGHE, *Understanding by design*. ASCD, 2005.

[14] B. Bloom, M. F. Englehart, W. Hill, and D. Krathwohl, *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. New York, Toronto: Longmans, Green, 1956.

[15] T. Bell, I. H. Witten, and M. Fellows. (2010) Computer Science Unplugged. [Online]. Available: http://csunplugged.org

[16] Qt, a cross–platform application and UI framework. [Online]. Available: http://qt-project.org/

[17] A. S. Tanenbaum, *Modern operating systems*. Prentice Hall Englewood Cliffs, 2007, vol. 3.

[18] A graphical tool for designing and simulating logic circuits. [Online]. Available: http://ozark.hendrix.edu/~burch/logisim/

[19] T. H. Cormen, R. L. Leiserson, Charles E Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[20] National computational science institute. [Online]. Available: http://computationalscience.org/

[21] S. M. Diesburg, P. A. Gray, and D. Joiner, "High performance computing environments without the fuss: the bootable cluster cd," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 8–pp.

[22] C. Peck, "LittleFe: parallel and distributed computing education on the move," *Journal of Computing Sciences in Colleges*, vol. 26, no. 1, pp. 16–22, 2010.

[23] O. Abuzaghleh, K. Goldschmidt, Y. Elleithy, and J. Lee, "Implementing an affordable high–performance computing for teaching–oriented computer science curriculum," *Trans. Comput. Educ.*, vol. 13, no. 1, pp. 3:1–3:14, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2414446.2414449

[24] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing*. Benjamin/Cummings Redwood City, 1994, vol. 110.

[25] P. Pacheco, *An introduction to parallel programming*. Access Online via Elsevier, 2011.

[26] M. J. Quinn, *Parallel Programming*. TMH CSE, 2003, vol. 526.

[27] NSF/IEEE–TCPP Curriculum Initiative. (2013) Call for proposals: Book project on parallel and distributed computing topics for undergraduate core courses. [Online]. Available: http://www.cs.gsu.edu/~tcpp/curriculum/?q=CFP_Book_Project

[28] J. Ortiz-Ubarri and R. Arce-Nazario, "(Extended abstract) Modules to teach parallel computing using Python and the LittleFe Cluster," in *Supercomputing (To appear)*, 2013.

[29] S. Papadimitriou and J. Sun, "Disco: Distributed co–clustering with map–reduce. ICDM," 2008.

[30] R. A. Arce-Nazario and J. R. Ortiz-Ubarri, "Enumeration of costas arrays using GPUs and FPGAs," in *Reconfigurable Computing and FPGAs (ReConFig), International Conference*. IEEE, 2011, pp. 462–467.