ABET Accreditation: A Way Forward for PDC Education

Sherif G. Aly
Department of Computer Science and Engineering
The American University in Cairo
Cairo, Egypt
sgamal@aucegypt.edu

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
haidar@acm.org

Rajendra K. Raj
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
rkr@cs.rit.edu

Sanaa Sharafeddine
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
sanaa.sharafeddine@lau.edu.lb

***Abstract*—With parallel and distributed computing (PDC) now wide-spread, modern computing programs must incorporate PDC within the curriculum. ACM and IEEE Computer Society's Computer Science curricular guidelines have recommended exposure to PDC concepts since 2013. More recently, a variety of initiatives have made PDC curricular content, lectures, and labs freely available for undergraduate computer science programs. Despite these efforts, progress in ensuring computer science students graduate with sufficient PDC exposure has been uneven.**

This paper discusses the impact of ABET's revised criteria that have required exposure to PDC to achieve accreditation for computer science programs since 2018. The authors reviewed 20 top ABET-accredited computer science programs and analyzed how they covered the required PDC components in their curricula. Using their own institutions as case studies, the authors examine in detail how three different ABET-accredited computer science programs covered PDC using different approaches, yet meeting the PDC requirements of these ABET criteria. The paper also shows how ACM/IEEE Computer Society curricular guidelines for computer engineering and software engineering programs, along with ABET accreditation criteria, can cover PDC.

***Index Terms*—Parallel and distributed computing, computing programs, curricular guidelines, ABET accreditation.**

I. Introduction

Parallel computing is often traced to the *D825* multiprocessor system in the 1960s [1], and distributed systems to early computer networks and client-server computing in the 1970s [2]. Concurrent programming also became commonplace around the same time-frame, with synchronizing communicating processes formalized by Disjktra [3] and Hoare [4]. Nowadays, much of the dramatic advances in a wide range of disciplines are enabled by parallel and distributed computing (PDC). For example, Internet of Things (IoT) provides the basis for computing devices of heterogeneous nature, technology, and capability to cooperate and deliver added value services across diverse industries. Mobile healthcare, industrial automation, holographic teleportation, autonomous vehicles, smart environments, digital currency mining, and GPU computing in general are examples of systems enabled by revolutionary advances in communications and parallel computing [5]. Such systems are founded by services that inherently need parallel processing, inter-service communication, and distributed monitoring and control.

Given the state of today's computing, it is crucial that computing professionals understand PDC intimately, and for computing education programs to incorporate PDC concepts, not just at the graduate level but also at the undergraduate level. Although awareness of the need for PDC education has existed for more than a decade, computing education has been slow at formalizing this need and making it actionable. In the case of the dominant Computer Science (CS) discipline, the ACM and IEEE Computer Society took some initial steps in the CS2013 [6] curricular guidelines to highlight the importance of PDC as one of the core tier-1 and tier-2 areas. Moreover, the guidelines expected additional coverage of PDC concepts intrinsic to operating systems, computer architecture, database management systems, and computer networks. More recently, the visionary CC2020 described draft PDC competencies for CS [7, pp. 112–123].

Despite these developments, many computer science programs have not paid sufficient attention to including PDC in their curricula, and ultimately preparing graduates to meet today's needs despite the availability of textbooks [8], [9], as well as extensive curricular resources developed by the Center for Parallel and Distributed Curriculum Development and Educational Resources (CDER) [10], PDC Unplugged [11], and researchers [12]–[16].

Recognizing these professional needs, as well as the lack of sufficient incentives, ABET's [17] Computing Accreditation Commission (CAC) revised its criteria for undergraduate Computer Science programs by requiring coverage of PDC concepts [18]. These criteria were first released in 2018, and since 2019, all ABET-accredited CS programs are required to include PDC in their curricula. As ABET typically follows a

six-year review cycle, it is likely that all [currently] 398 ABET-accredited CS programs will have coverage of PDC content by 2024.

This paper explores the impact of the revised ABET's CS Program Criteria on the incorporation of PDC content into modern ABET-accredited computer science programs. Section II discusses the history and formulation of the PDC requirements in the ABET CAC Criteria, along with approaches to covering PDC not only to meet the ABET CAC Criteria requirements, but also to enhance them. Section III presents the results of a survey of top ABET-accredited CS programs to understand how they support PDC concepts across their curricula. Using three examples of ABET-accredited programs, Section IV expands on actual implementation of the different approaches to satisfy the PDC requirements of the CAC Criteria. Section V explores whether the ABET engineering criteria [19], along with the ACM/IEEE Computer Society curricular guidelines for computer engineering [20] and software engineering [21], essentially require coverage of PDC concepts to ensure that graduates are prepared to enter engineering practice. The paper concludes with remarks about how PDC coverage can be improved in computing and engineering programs.

II. Using Accreditation to Move PDC Forward

The world of accreditation differentiates between *institutional accreditation* where regional or national accreditation organizations provide assurance that a university as a whole reliably provides quality education and *program accreditation* where accrediting bodies use disciplinary guidelines to establish and verify the quality of the degree program and increasingly, the characteristics of the program graduates [22]. Program-level accreditation focuses on areas such as student advising, graduation requirements, faculty qualifications, curriculum standards, and available facilities. In the US (and over 40 other countries), ABET [17] has been the accrediting body for programs in computing and engineering, through its Computing Accreditation Commission (CAC) and Engineering Accreditation Commission (EAC). The rest of this section examines PDC requirements in the CAC Criteria, specifically Computer Science; Section V explores PDC in Software Engineering and Computer Engineering.

A. ABET's Computer Science Accreditation Criteria

ABET's program criteria across all its commissions are created by the professional society representing the corresponding discipline. For instance, for computing disciplines including computer science, the professional society is CSAB, the joint body created by ACM and IEEE Computer Society [23]. As mentioned earlier, the ACM and IEEE Computer Society have jointly developed different curricular guidelines for a variety of computing disciplines. In the case of CS, the 2013 ACM/IEEE Computer Society's Computer Science Curricular Guidelines (CS2013) [6] have been the latest. These guidelines highlighted the importance of parallel and distributed computing and noted that "parallel and distributed computing has moved from a largely elective topic to become more of a core component of undergraduate computing curricula."

> ...
> The curriculum requirements specify topics, but do not prescribe specific courses. For CS, the requirements are: at least 40 semester credit hours (or equivalent) that must include (among several other topics):
>
> - ...
> - Exposure to computer architecture and organization, information management, networking and communication, operating systems, and **parallel and distributed computing**.
> - ...

Fig. 1. Computer Science Program Criteria—Curriculum

Therefore, the accreditation criteria for CS developed by ABET draw substantially from CS2013 [6], which defines parallel and distributed computing to encompass the following [6]:

1) An understanding of fundamental systems concepts such as concurrency and parallel execution, consistency in state/memory manipulation, and latency.
2) Understanding of parallel algorithms, strategies for problem decomposition, system architecture, detailed implementation strategies, and performance analysis and tuning.
3) Message-passing and shared-memory models of computing.

CC2020 [7] reiterates the above knowledge areas and recommends specific topics including a coverage of a parallel divide-and-conquer algorithm, critical path, race conditions, processes, deadlocks, and properly synchronized queues.

What follows examines how PDC is required by the CAC Criteria for accrediting Computer Science programs and what approaches can be used by accredited programs to meet this requirement.

Program-level accreditation of Computer Science (CS) programs has been ongoing since the 1985-86 academic year. At present, the CAC accredits 398 CS programs around the globe using the *ABET Criteria for Accrediting Computing Programs (Criteria)* [18]. The CS Criteria comprise the *General Criteria* that apply to all computing programs and the *Computer Science Program Criteria* that are specific to CS programs.

This paper focuses on the CS programs because they are now required to cover PDC by the corresponding criteria, within the curriculum requirements, as depicted in Fig. 1. In other words, in line with the CS2013 curricular requirements [6], the ABET CS Program Criteria require exposure to computer architecture and organization, information management, networking and communications, operating systems, and parallel and distributed computing. The criteria explicitly point out that these requirements are flexible, and do not necessarily ask for courses to be introduced into programs, but rather topics or knowledge areas that ought to be covered somewhere in the program requirements. For example, some topics may

be covered in a relatively small number of lectures embedded within an existing course.

The phrase "exposure to" used in these criteria is in line with the CS2013 curricular requirements. With the anticipated continued growth of PDC in computing, the authors assume the next decennial revision of the ACM CS guidelines will emphasize PDC even more. In the meantime, as discussed in the next two sections, many excellent programs in CS are already emphasizing PDC, even beyond what the ABET CS Criteria require.

B. Meeting the PDC Exposure Requirement in the Computer Science Criteria

The Computer Science Criteria are flexible and provide several avenues for the PDC content to be included. Opportunities thus exist across various courses within a Computer Science curriculum to include PDC exposure. For the accreditation criteria to be satisfied, a program must include PDC coverage in required coursework to ensure all graduating students receive the required exposure. Based on curricula formulated by CDER [10], three core PDC concepts have been clearly identified namely, *concurrency*, *parallelism*, and *distribution* [24]. These topics can be covered in courses such as computer programming, algorithms design, operating systems, databases, networking, and architecture.

Newhall [25] explored how PDC could be included when planning or revising the curriculum:

1) Achieve an early exposure and maturity of PDC topics to allow learners an opportunity to achieve further in-depth knowledge where needed before graduation.
2) Design a curriculum that intentionally overlaps some topics across the curriculum.
3) Provide breadth of knowledge as well as appropriate in-depth coverage.
4) Expose learners to the topics in multiple sub-disciplines as opposed to coursework that only covers PDC topics.

Most programs follow one or both of two major approaches to incorporating any content in curricula. In the first approach, programs dedicate at least one required course to cover the required knowledge units for any required area. For PDC, such a required course would need to contain essential parallel programming concepts, as well as more advanced parallelism and distribution concepts. This content can be developed following the guidance provided by CS2013 [6] or the draft PDC competencies mentioned in CC2020 [7].

The dedicated-course approach is the obvious approach, as it provides faculty and students with the opportunity to explore a wide array of PDC topics while emphasizing others that may have been offered in other courses in the program. A course like this could tap into the vast amount of resources that are made available by Intel, NVIDIA, and IBM, among others, which include course materials, textbooks, free compilers, free training, and in some instances free hardware. Also, CDER [10] and PDC Unplugged [11] provide a multitude of curricular content and labs that can be easily included.

On the other hand, due to limits imposed by faculty availability and expertise, available number of course credits, and related constraints, a program may find that it is simply not possible to add a dedicated course to the program curriculum. However, as PDC concepts are inherent in various computing areas, it is not hard to integrate different parts of the knowledge area into existing courses in the curriculum, rather than create a dedicated course. Thus, PDC coverage is supported at the periphery of the curriculum by scattering the knowledge units into various required courses. Some programs would probably find this approach more practical to satisfy accreditation requirements while providing the students with some exposure to PDC. Table I shows how a typical accredited computer science program may choose to include coverage of PDC across different courses such as Systems Programming, Computer Organization or Architecture, Operating Systems, and Database Systems.

III. PDC in 20 Top Accredited CS Programs

To understand how ABET-accredited programs have incorporated PDC in their curriculum, the authors collected data from 20 ABET-accredited computer science programs that are ranked in the top 100 by US News [26].

The collected data was studied with a focus on required courses that included PDC components. The course descriptions and the learning outcomes were analyzed using information that were published online. A weighted sum of all courses that tackle specific components of the PDC knowledge area was computed. Fig. 2 shows common topics covered by the surveyed programs that help to support PDC for ABET Accreditation. Fig. 3 shows the percentage of courses that encompass these topics. It is worth noting that out of the 20 surveyed programs, only one program had a dedicated parallel programming course while the remaining programs used multiple courses to cover PDC topics.

Most modern CS programs offer the following courses, several of which are required:

1) A typical *operating systems* or *systems programming* course can include coverage of concurrency, atomicity, shared memory programming, and inter-process communication. Programming with threads can also be covered in a more elaborate way using client-server programming in a computer networks course or in systems programming course;
2) A *database management* course can incorporate distributed computing concepts including transactions processing, scheduling concurrent transactions, transactions locks, and deadlocks;
3) A *computer organization* or *architecture* course can incorporate Amdahl's law and its implication on the performance of a particular parallel algorithm, speed-up and scalability, Flynn's taxonomy, the concept of multicore processors, multiprocessor caches and cache coherence, pipelining, instruction level parallelism, and vector and SIMD computers. Advanced computer archi-

TABLE I
MAPPING DIFFERENT PDC CONCEPTS TO TYPICAL COURSES

	Systems Programming	Computer Organization/ Architecture	Operating Systems	Database Systems	Computer Networks
Programming with threads	×		×		×
Transactions processing				×	
Parallelism and concurrency	×	×	×	×	×
Shared-Memory programming	×		×		
Inter-Process Communication (IPC)	×		×		×
Atomicity	×		×		
Performance measurement, speed-up, and scalability		×			
Multicore processors		×			
Shared vs. distributed memory	×	×	×		
SIMD and vector processors		×			
Instruction Level Parallelism		×			
Flynn's taxonomy		×			
Client-server programming	×				×
Memory and caching	×	×	×		

Fig. 2. PDC Topics Used by Surveyed Programs for ABET Accreditation

tecture courses can also cover concepts of shared and distributed memory;

4) *Programming language* courses can incorporate support for parallel and distributed computing concepts, including but not limited to, programming with threads, programming language constructs and APIs for thread creation and management, networking to support distributed computing, inter-process communication and synchronization, along with virtual machine and translator support for parallelization;
5) Selected parallel algorithms and related theoretical analysis and applications can be covered in a *design and analysis of algorithms* course;
6) A course in *software engineering* can provide a coverage of modeling and notations used for parallel behavior as well as design considerations of distributed components.

These results show–at a high level–that the ABET Computer Science criteria have already begun to have their anticipated impact on incorporating PDC concepts in undergraduate CS programs. Three different ABET-accredited CS programs on three different continents are now examined for achieving PDC coverage.

IV. CASE STUDIES: PDC IN CS PROGRAMS

This section examines three ABET-accredited computer science programs from the authors' own institutions to study the actual coverage of PDC topics.

A. Case Study: Lebanese American University

The Computer Science Program at the Lebanese American University incorporated parallel programming early in 1996 in the form of a fully-fledged course based on the Network of Workstations (NOW) model [27], [28] using the Message-Passing Interface (MPI). The course content was later updated to include GPGPU manycore programming and a brief introduction to deep learning as a case-study to showcase the power of parallelism [29]. It should be noted that in addition to this required course, students explore PDC concepts in various required courses including *operating systems*, *computer organization*, and *database management systems*.

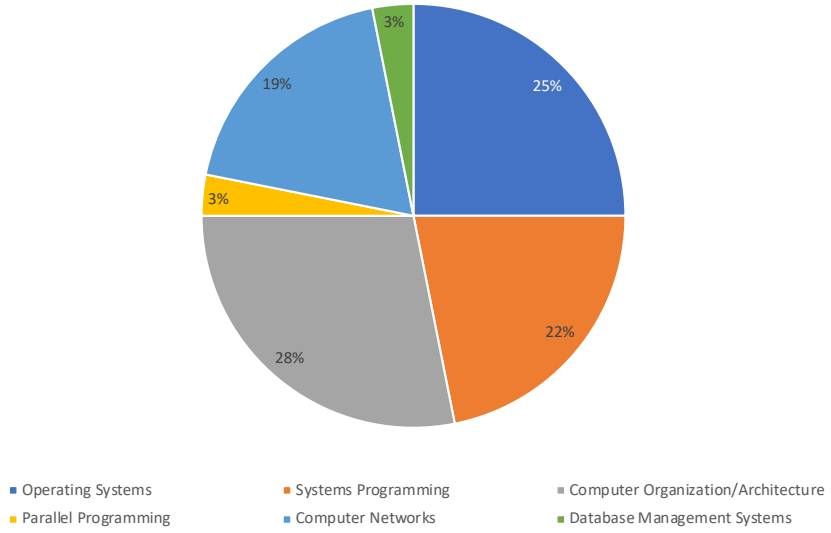

Fig. 3. Courses for PDC Content by Surveyed Programs for ABET Accreditation

1) Meeting the PDC Program Criteria: The dedicated course introduces relevant parallel architectural trends and aspects of multicores in addition to the design and analysis of parallel programs. The course description is:

> This course provides an introduction to parallel programming with a focus on multicore architectures and cluster programming techniques. Topics include relevant architectural trends and aspects of multicores, writing multicore programs and extracting data parallelism using vectors and SIMD, thread-level parallelism, task-based parallelism, efficient synchronization, program profiling, and performance tuning. Message-passing cluster-based parallel computing is also introduced. The course includes several programming assignments to provide students first-hand experience with programming, and experimentally analyzing and tuning parallel software.

The course learning outcomes are that students will:

1) Understand the challenges of as well as the motivations for using parallel programming.
2) Demonstrate an ability to analyze the efficiency of a given parallel algorithm.
3) Demonstrate an ability to design, analyze, and implement programming applications using multicore and manycore systems.

In addition, students need to complete labs, a milestone project in each part, one written report per milestone, and to make one oral presentation. As the course is being used to assess communication skills, all material is graded for grammar, spelling, and style.

The course delivery is easily broken down into three parts. The first part introduces the history of PDC and the driving forces behind its recent resurrection, tackling issues such as the continuous decrease of feature size, the increase in the number of transistors per chip, clocking, performance scalability, excessive power consumption, and the rise of multicore processors.

The second part of the course focuses mostly on multicore programming using `Pthreads` and `OpenMP`. Students are introduced to shared-memory issues including thread-level parallelism, task-based parallelism, data races, false sharing, memory contention, synchronization, and extracting data parallelism using vectors and SIMD. This part of the course is supported by *Intel*, which provides hands-on labs as well as free licenses for the *Intel Parallel Studio* compiler.

In the third part, which is roughly 60% of the course, students are introduced to the concept of manycores focusing on single instruction, multiple threads (SIMT) execution model that is inherent in GPU devices. In this part, students develop manycore applications using `CUDA C` and `OpenACC`, and learn advanced memory management techniques as well as using concurrent streams. This part of the course is supported by NVIDIA's Deep Learning Institute (DLI) which provides all students access to the NVIDIA's online training platforms. Typically, students earn two NVIDIA certificates in this part, an OpenACC and a CUDA C. Labs are completed on the cloud using NVIDIA's accelerated computing environment.

The parallel programming course is also a cornerstone for meeting the ABET criteria. In fact, the course is used to meet multiple performance criteria in ABET's Student Outcome 2 (Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline) and Student Outcome 3 (Communicate effectively in a variety of professional contexts), as well as the curricular requirement for PDC exposure.

B. Case Study: The American University in Cairo

The Computer Science and Engineering Department at the American University in Cairo offers two ABET accredited programs: one in Computer Science, and another in Computer

Engineering. Both programs follow an early maturity pedagogical approach to ensure students learn key knowledge units as early as possible in the curriculum. The intent is to provide students with an ability to innovate faster, and create more meaningful contributions before graduation.

The CS program does not require a dedicated course that covers PDC topics, yet the knowledge units to support this requirement are satisfied across various other courses.

1) In the fundamental course sequences, students are exposed to a basic ability to create and control threads, as well as a simple client server connectivity.
2) In the courses of computer organization, and architecture, multiprocessing, thread level parallelism, pipelining, instruction level parallelism, superscalar architectures, VLIW architectures and architectures based on dynamic scheduling such a the non-speculative and the speculative versions of Tomasulo's architectures are covered.
3) The operating systems course provides coverage at a substantial depth for multi-threading, speedup, multiprocessing, mutual exclusion, synchronization, deadline and starvation, and scheduling on single and multiprocessor systems.
4) In software engineering, students are exposed to the development of semi-complex systems that involve distribution of components and some parallel functionality and distribution of components.
5) In concepts of programming languages, design considerations for various programming languages are discussed, including support for networking, and threading. Students work on a project using one of the trending languages, and must present many of the language features including language support for parallelism and networking, and the effect of garbage collection on currently executing processes and threads.
6) Even though there is a course dedicated to distributed systems within the department, it is only required at the moment for the Computer Engineering program. This fundamentals of distributed computing course covers topics ranging from modeling and specification to consistency and inter-process communication, load balancing, process migration, and distributed challenges. A demanding project helps students master some of the fundamental and advanced topics of the domain.

C. Case Study: Rochester Institute of Technology

In 2010, i.e., almost a decade prior to the CAC Criteria requirement for PDC to be covered, the faculty in RIT's computer science program recognized that few modern computers consisted of a single box with a single CPU, and that modern computers typically include multi-core CPUs forming clusters of machines connected by a local network, or are units of distributed systems interconnected over the Internet. The faculty decided then that all computer science students needed to know not only how these these parallel and distributed systems function but also how to write programs for these computing systems. To address this goal, it was decided to create a coherent single semester-long course that would allow students learn the foundations of PDC including principles of parallel computing, basic network protocols, principles of network security, basic architectures of cluster and grid computing, and build additional depth in multithreaded programming and techniques for developing network applications.

Another important consideration was that these topics needed to be covered in a single course, so that students could understand the interrelationships between these topics, such as the synergies between multithreaded programming and network programming. The focus was not to cover these topics in depth, but instead to provide the breadth of knowledge and skills in PDC for each computer science student. The intention was that students would be ready to take advanced electives in these topics, which could include operating systems, networking, systems programming, distributed systems, parallel computing, grid computing, as well as cryptography and cybersecurity. Before this change, the program had required courses in operating systems and in networking. Obviously after the change, modified courses in operating systems and networking were created as electives to explore advanced topics in greater depth.

The course, *Concepts of Parallel and Distributed Systems*, has been taught since Fall 2013 when RIT transitioned to a semester system. The course description is:

> This course is an introduction to the organization and programming of systems comprising multiple computers. Topics include the organization of multicore computers, parallel computer clusters, computing grids, client-server systems, and peer-to-peer systems; computer networks and network protocols; network security; multithreaded programming; and network programming. Programming projects will be required.

The course learning outcomes are that students will:

1) Explain the concepts of processes, threads, and scheduling.
2) Develop multithreaded programs.
3) Explain the concepts of computer networking, the layered network architecture, network security, and network communication with connections and datagrams.
4) Develop network application programs.
5) Explain the concepts of distributed system architectures and middleware.
6) Explain the concepts of parallel computer architectures.

The main topics covered in the course include the following: multithreaded computing (including processes, threads, scheduling, synchronization, deadlock, starvation. and multicore computers); networked computers (including client-server, connections, application protocol design, and socket and datagram programming); network protocols and security; distributed systems (including different architectures, middleware, distributed objects, and web services), and parallel computing (including architectures and middleware).

In addition to this required course, students explore PDC concepts in earlier courses, starting in their second required programming course in the freshman year, where Java threads and synchronization are explored in depth. The required course, Mechanics of Programming, covers pthreads in depth. Another required course, Concepts of Computer Systems, also covers pipelining and other related PDC concepts.

V. PDC in Computer and Software Engineering Programs

ABET's engineering accreditation criteria [19] for computer engineering (CE) and software engineering (SE) programs do not explicitly mention PDC. They, however, do require the program curriculum to "provide adequate content for each area, consistent with the student outcomes and program educational objectives, to ensure that students are prepared to enter the practice of engineering" [19]. Modern curricula for these engineering programs are typically guided by the curricular guidelines developed by ACM and IEEE Computer Society's Computer Engineering [20] and Software Engineering [21]. In essence, any modern computer engineering or software engineering program that lives up to these curricular guidelines will need to cover PDC concepts. In other words, it is likely that ABET's engineering criteria, being based on the ACM/IEEE Computer Society curricular guidelines, require the PDC coverage through courses that are inherently required for both programs.

The computer engineering curriculum guidelines (CE2016) delineate twelve broad knowledge areas for the relevant body of knowledge [20]. Knowledge areas are decomposed into associated knowledge units with a list of learning outcomes each. Knowledge units are classified as core and supplementary units. Core units of each knowledge area constitute its minimal required knowledge and skills that have to be covered within a program curriculum. Computing algorithms, computer architecture and organization, systems resource management, and software design represent knowledge areas of computer engineering that explicitly address parallel and distributed computing concepts through core knowledge units.

Table II lists the knowledge areas that cover PDC-related core knowledge units. As per CE2016, most of the pertinent skills associated with the latter knowledge units are expected to be minimally met with relatively basic ability, while a few of them are set to a higher level of accomplishment.

TABLE II
PDC in Computer Engineering Knowledge Areas [20]

Knowledge Area	PDC-related Core Knowledge Units
Computing Algorithms	Parallel algorithms/threading
Architecture and Organization	Multi/Many-core architectures Distributed system architectures
Systems Resource Management	Concurrent processing support
Software Design	Event-driven and concurrent programming

Similar to CE2016, the software engineering curriculum report SE2014 defines a body of core knowledge that represents the knowledge and skills for which there is consensus that any software engineering graduate should acquire [21]. This core knowledge is coined as software engineering education knowledge areas and referred to as SEEK. SEEK comprises 10 knowledge areas with a set of associated knowledge units, some of which are designated as essential in relevance to the core knowledge. CE2016 defines a collection of topics that include the cognitive skill level at which each topic of a given knowledge unit is expected to be attained. Three cognitive skill levels are defined with application being the highest level.

As shown in Table III, computing essentials area is a main knowledge area of SEEK that entails the foundation for the design and construction of software and includes one unit on construction technologies. The former unit emphasizes two PDC-related topics, namely concurrency primitives and construction methods for distributed software. Both topics are classified as essential to the core and expected to be met at the application level.

TABLE III
PDC in Software Engineering Knowledge Areas [21]

Knowledge Area	PDC-related Core Topics
Computing Essentials	Concurrency primitives (e.g., semaphores and monitors) Construction methods for distributed software (e.g., cloud and mobile computing)

With reference to the above, any computer engineering or software engineering program addresses concepts on parallel and distributed software with varying levels of attainment. Students understand the intrinsic performance gains of such software, albeit they are aware that the potential gain is highly dependent on how its different parts are set to interact among each other in terms of dependencies, communications, and competitions. Per ACM/IEEE Computer Society curricular guidelines, SE and CE programs do equip their students with tools to combat the challenges of concurrency. Parallel and distributed software can indeed apply to the complex software-related ABET program criteria for CE and SE. ABET program criteria for computer engineering require that students learn to analyze and design complex software among others. Although analysis and design of application-specific algorithms represent one individual knowledge unit of computing algorithms area in computer engineering, incorporating PDC concepts described in both knowledge areas on computing and software design yields complex software, thereby meeting the corresponding ABET program criterion.

With reference to the ABET program criteria for software engineering, the curriculum must include software engineering tools appropriate for the development of complex software systems. The attainment of the specified PDC-related topics on concurrency and construction methods for distributed software as per SE2014 with the recommended cognitive skill level allows strengthening the compliance with this criterion.

In short, even though the ABET program criteria for computer and software engineering do not explicitly require PDC coverage, it is clear by looking at the curricular guidelines for

both programs, it is simply not possible for program graduates in these disciplines to graduate without learning PDC. The computer engineering and software engineering programs at the authors' institutions anecdotally verify this claim.

VI. Final Remarks

Given the ubiquity of PDC in all aspects of modern society, it is incumbent upon contemporary computing and engineering programs to include appropriate coverage of these topics. Accordingly, parallel and distributed computing knowledge areas are emphasized in the latest ACM curricula, notably CC2020 [7]'s draft competencies for PDC within CS.

Program accreditation also has a role to play in moving curricula forward. The latest ABET's CAC criteria started requiring PDC exposure for CS programs in 2018 [18]. These criteria already have led to systematically increasing PDC coverage, as illustrated by our survey of 20 of the top accredited CS programs and the detailed examination of how three accredited CS programs cover PDC. Two approaches to cover PDC were observed: a predominant one that uses multiple topics across various computer science courses, and a second approach that uses a dedicated PDC course. Both approaches are viable and meet the current ABET criteria.

It is expected that PDC will continue to gain importance in computing curricula with recent computing development and applications. Engineering, on the other hand, has a more general approach in line with the centuries of engineering education and practice. Thus, this paper also explored whether today's engineering graduates are also exposed to PDC concepts to be ready for careers in their disciplines. The conclusion is that ABET program accreditation has played a useful role in infusing PDC topics into CS programs around the world at a faster pace, thus leading to stronger and wider PDC coverage.

Acknowledgment

Raj acknowledges support provided by the US National Science Foundation under Awards 1922169 and 2021287.

References

[1] J. P. Anderson, S. A. Hoffman, J. Shifman, and R. J. Williams, "D825 - a multiple-computer system for command amd control," in *Proceedings of the December 4-6, 1962, Fall Joint Computer Conference*, (New York, NY, USA), p. 86–96, Association for Computing Machinery, 1962.

[2] M. Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, p. 967–1009, Oct. 2016.

[3] E. W. Dijkstra, "The structure of the THE-multiprogramming system," *Commun. ACM*, vol. 11, p. 341–346, May 1968.

[4] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, pp. 666–677, Aug. 1978.

[5] I. F. Akyildiz, A. Kak, and S. Nie, "6g and beyond: The future of wireless communications systems," *IEEE Access*, vol. 8, pp. 133995–134030, 2020.

[6] ACM and IEEE Computer Society, "Computer science curricula 2013," tech. rep., ACM Press and IEEE Computer Society Press, December 2013. Accessed: November 14, 2017.

[7] A. Clear, A. Parrish, J. Impagliazzo, P. Wang, P. Ciancarini, E. Cuadros-Vargas, S. Frezza, J. Gal-Ezer, A. Pears, S. Takada, H. Topi, G. van der Veer, A. Vichare, L. Waguespack, and M. Zhang, "Computing curricula 2020 (CC2020) paradigms for global computing education," tech. rep., ACM/IEEE, 2020. December 2020, https://www.acm.org/binaries/content/assets/press-releases/2021/march/computing-curricula-2020.pdf.

[8] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3rd ed., 2016.

[9] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

[10] CDER Center, "NSFIEEE-TCPP Curriculum Initiative," 2020. https://tcpp.cs.gsu.edu/curriculum/?q=node/21183.

[11] S. J. Matthews, "PDC Unplugged," 2021. https://www.pdcunplugged.org/.

[12] S. K. Prasad, A. Gupta, A. Rosenberg, A. Sussman, and C. Weems, *Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*. Springer International Publishing, 2018.

[13] A. Fernández, C. Fernández, J. A. Miguel-Dávila, M. A. Conde, and V. Matellán, "Supercomputers in the educational process," in *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'19, (New York, NY, USA), p. 548–553, Association for Computing Machinery, 2019.

[14] V. M. Olivera and J. L. G. Sánchez, "Thinking in parallel: Supercomputing education," in *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'19, (New York, NY, USA), p. 531–533, ACM, 2019.

[15] H. Farian, K. M. Anne, and M. Haas, "Teaching high-performance computing in the undergraduate college cs curriculum," *J. Comput. Sci. Coll.*, vol. 23, p. 135–142, Jan. 2008.

[16] A. Qasem, D. Bunde, and P. Schielke, "ToUCH: Teaching Undergrads Collaborative and Heterogeneous Computing," Aug. 2020. https://github.com/TeachingUndergradsCHC/modules/blob/master/touch_overview.pdf.

[17] ABET, Inc., "ABET webpage," 2021. https://abet.org.

[18] ABET, Inc., "Criteria for accrediting computing programs. effective for reviews during the 2020-2021 accreditation cycle," 2021. https://www.abet.org/wp-content/uploads/2019/12/C001-20-21-CAC-Criteria-MARK-UP-11-30-19-Updated-2.pdf.

[19] ABET, Inc., "Criteria for accrediting engineering programs. effective for reviews during the 2020-2021 accreditation cycle," 2021. https://www.abet.org/wp-content/uploads/2020/09/EAC-Criteria-2020-2021.pdf.

[20] ACM and IEEE Computer Society, "Computer engineering curricula 2016: Curriculum guidelines for undergraduate degree programs in computer engineering," Dec. 2016. https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf.

[21] ACM and IEEE Computer Society, "Software engineering 2014: Curriculum guidelines for undergraduate degree programs in software engineering," Feb. 2015. https://www.acm.org/binaries/content/assets/education/se2014.pdf.

[22] M. Oudshoorn, R. K. Raj, S. Thomas, and A. Parrish, "The value of abet accreditation to computing programs," in *2018 ASEE Annual Conference & Exposition*, (Salt Lake City, Utah), ASEE Conferences, June 2018. https://peer.asee.org/31135.

[23] CSAB, Inc., "CSAB webpage," 2021. https://csab.org.

[24] R. K. Raj, C. Romanowski, J. Impagliazzo, S. G. Aly, B. Becker, J. Chen, S. Ghafoor, N. Giacaman, S. Gordon, C. Izu, S. Rahimi, M. P. Robson, and N. Thota, "High performance computing education: Current challenges and future directions," in *Working Group Reports on Innovation and Technology in Computer Science Education*, 2020.

[25] T. Newhall, A. Danner, and K. C. Webb, "Pervasive parallel and distributed computing in a liberal arts college curriculum," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 53 – 62, 2017. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing.

[26] US News, "Best Computer Science Schools," 2018. https://www.usnews.com/best-graduate-schools/top-science-schools/computer-science-rankings.

[27] T. E. Anderson, D. E. Culler, D. A. Patterson, , and the NOW team, "A case for now (networks of workstations)," *IEEE Micro*, vol. 15, p. 54–64, Feb. 1995.

[28] J. C. Adams, J. Caswell, S. J. Matthews, C. Peck, E. Shoop, and D. Toth, "Budget beowulfs: A showcase of inexpensive clusters for teaching pdc," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, (New York, NY, USA), p. 344–345, Association for Computing Machinery, 2015.

[29] Lebanese American University, "Computer Science Curriculum," 2020. https://catalog.lau.edu.lb/2020-2021/undergraduate/programs/bs-computer-science.php.