We Need Community Effort to Achieve PDC Adoption!

Erik Saule, Kalpathi Subramanian
Computer Science
UNC Charlotte
Charlotte, NC, USA
Email: {esaule,krs}@uncc.edu

Jamie Payton
Computer and Information Sciences
Temple University
Philadelphia, PA, USA
Email: payton@temple.edu

***Abstract*—The Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) released a PDC guideline in 2012 and soon after IEEE/ACM included PDC topics in their CS curriculum guidelines in 2013. CDER is currently working to update the PDC guidelines. The primary strategy to achieve adoption of PDC in early CS course is the courseware approach where pedagogical materials are developed and made available for instructors to integrate PDC content into their courses. content.**

Despite the creation of many new materials, adoption of PDC content has been fairly slow. In this position paper, we present a framework to classify the marks one need to hit to develop courseware that is adoptable, portable, engaging, and easy to find. We review some of the efforts to improve PDC adoption in the US and we hypothesize that these efforts have not consistently exhibited the attributes necessary for wide adoption.

Our position is that while the courseware that have been developed by the community are essential stepping stones, we will not see PDC adoption until a larger community can be brought to bear on the problem. We need to enable composing and remixing the existing materials and develop new materials to cover the needs of a diverse range of institutions.

***Index Terms*—PDC curriculum; courseware; adoption of PDC in early CS courses; student engagement;**

I. Introduction

Parallel and Distributed Computing (PDC) has become a more important topic since internet became ubiquitous and multicore systems became the defacto standard. Single core system are essentially only found nowadays in highly specialized environments. And most software are developped out of asynchronous distributed components such as databses, front-end back-end systems, webservices, and cloud components.

Yet PDC has remained an advanced topic in most curricula. The Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) released NSF/IEEE-TCPP curriculum guidelines in 2012 [1], and a new iteration of these guidelines is currently under beta [2]. Adding a PDC course in a computing curriculum will likely not bring PDC education to all of computer science: a more promising strategy is to integrate PDC topics all across the undergraduate curriculum, from early CS courses such as programming and data structures, to more advanced courses such as operating systems, or data mining. Further, the importance of PDC in Computer Science curriculum was recognized by ACM and IEEE who integrated PDC topics in their 2013 Computer Science curriculum guidelines [3] as a dedicated area, but also spread them in multiple places in the guidelines.

The development and adoption of these guidelines at a national level indicates that there is an understanding that PDC topics are important. But at the local levels in US universities (and we suppose over the world) of all kinds, the integration of these topics in courses has been slow. This slow adoption does not come from a lack of effort. The PDC education community has produced books [4], [5], assignments [6], unplugged activities [7], and course modules [8]. The community has also produced self-paced educational materials [9]. All these efforts are good efforts, however, we argue that they are too small in scope.

This approach to PDC adoption is the courseware approach. The core idea is that instructors need materials in their classes and will only adopt PDC in their classes if there are ready-to-teach materials for it. In this position paper, we argue that *the courseware approach is the only approach that can work; but we will need a much larger community effort to make it viable.*

II. What makes good courseware?

We first need to understand what attributes contribute to good and open courseware. Courseware can be anything used in a class, from a single activity to a complete module with lecture materials, demonstrations, textbook reference, activity, assignment, and assessment questions. Good courseware, however, do not exist in the absolute. Courseware get adopted by a particular instructor, for a particular class in a particular program, to impact a particular set of students. A courseware that may not be effective in a particular class could be perfect for another class. We cannot ignore the dynamics of the adoption of courseware, the diversity of instructors, programs, courses, and students where that adoption happens.

Let's review some of the properties we want out of good courseware:

a) Cover some technical topics accurately at a particular level of depth: This is what we usually think of when we think of good courseware: how well does it explain a particular topic to reach a particular level of understanding. But we can not decouple that property from who the students are. Some students may be more mathematically inclined; some students may have transferred from a different program; some may not

have scored particularly well on a previous class, not having developed some key intuition. A courseware will not matter to an instructor if they do not believe their students can benefit from it using their current knowledge of the topics.

b) Engage students.: There is only so long students are willing to do the hard work believing that "it will be useful eventually", "trust me, this is good for you". Courseware should take every opportunity to engage the students with its content. There are frames of thoughts on engagement such as the MUSIC model [10]. Some students may want to engage with content because it is challenging and they like a challenge. Maybe they engage with content because it is hands-on and they like being practical, which is the strategy of unplugged activities [7]. Maybe the content connects with a form of entertainment that they enjoy, which is a strategy for game based courses [11], [12]. Maybe the content makes them work on a social problem that they care about [13], [14]. Not every courseware can hit all these marks for all students, but instructors should attempt to hit some of them[10], and try to connect to students in their class by carefully mixing engagement strategies.

c) Justify why the techniques are needed: Expressing convincingly why a particular technique is needed is often harder than we expect. In a sense parallel computing topics have issues similar to topics in algorithms. The purpose of techniques is often to reduce the time it takes to perform some computation. But for students to find the content relevant, they need to perceive that the time it takes to execute the computation to be a real problem. This can be generally hard to do in practice, because we would need to ideally present a real computational problem that students can understand with their level of understanding of computing, that they can execute with their level of access to computational resources, and they can meaningfully work on.

d) Address the level of expertise of the instructor: Instructors may not have been properly trained themselves in PDC. The instructors need to be confident that they can meaningfully teach that content; that they can answer questions of students when they come. A courseware that is self-explanatory or that come with well-designed instructor packages will be more likely to be adopted.

e) Seemlesly integrate in a class: If you are trying to bring in content it needs to integrate in the class with minimal effort. There is a technical aspect to that the materials should be language agnostic or be available in the language of the class. The material should be compatible with the operating system normally used in the class. This is particularly a problem for PDC courseware since packages may assume the availability of a particular software environment, or a specific batch scheduler, or a given amount of memory or computational power. A similar effect is centered on instructor and instruction time. If using a topic forces a significant amount of side notes then it will take more time to cover, in preparation and instruction time.

f) Enable to build a curriculum: Individual courseware that covers a particular topic can be useful. But it is really the case when one enables an instructor to make a comprehensive class that is internally coherent and consistent with previous and follow-on courses that adoption will happen.

g) Be discoverable: Finally courseware is not useful if it can not reach the instructors and the students that need them. We need courseware to express clearly what they do and do not do.

Simply by looking at the list of properties, we can see that building good and useful courseware is a tall order. It is difficult to make courseware that will have all the properties that are desirable. Or at least it will be hard if they are products of a single person or a small group. We need a community large enough for courseware to be refined by different teams, and composed and remixed. Then we should be able to produce collectively the courseware for each instructor to find the materials that will make a difference in their class.

III. Existing efforts

A. Peachy Parallel Assignments

Peachy Parallel Assignments [6] are an effort that is taking place at PDC education workshops such as EduPar, EduHPC. Patterned after the Nifty Assignments [15] that are presented at SIGCSE, Peachy Parallel Assignments are meant to be good parallel computing assignments that are tested, adoptable, cool, and inspirational. There are currently 22 assignments in the Peachy collection.

The assignments are designed to be engaging and to tackle a particular point of the curriculum. They can be a very good starting point to justify why parallel computing is important. On the other hand, they all tend to tackle the same point of the curriculum: basic parallelisation of loops, usually with OpenMP. The assignments assume that the instructor can pick up the materials without additional context; and in some cases the assignments are about solving a PDE or executing a complex algorithm that will need to be explained to students.

And Peachy Parallel assignments tend to be technology bound (like most parallel computing assignments). They are essentially C programming assignments for CS1; but most CS1 courses are taught in Java or Python. So the assignments will need to be adapted by instructors to be useful.

Finally the assignments are hardly discoverable. One need to read the description of an assignment to understand what the assignment does and whether or where it could be used.

Peachy parallel assignments can be very useful resources. But they will need to be remixed and indexed to reach a broader audience.

B. PDC Unplugged

PDC Unplugged [7] is inspired from *CS Unplugged* which curates a set of activities that bring (without computers) a physicality to teaching computer science concepts [16]. There are currently 38 activities in the PDC Unplugged repository covering a wide range of topics, including scheduling, race conditions, latency, and many others.

PDC Unplugged activities hit many of the marks of good materials. They are really good at engaging students and give

them an experience that they will remember in the long run. Because the activities are physical, they can easily be done by instructors even with little training and are independent of computing environment which helps integrating them. The PDC Unplugged repository classifies the activities based on the CS2013 knowledge units and the NSF-IEEE TCPP PDC curriculum guidelines which helps figuring out which activities can be relevant to you.

By nature unplugged activities tend to be not technically very deep because they are designed to give a physical understanding of an underlying technical concept. They also often do not connect very well to particular applications because the activity is not a technical activity. So while these activities can be very useful and impactful in a curriculum they can only extend a module rather be a module in itself.

C. CSinParallel

CSinParallel [8] is a collection of modules, currently 26, to teach parallel computing at various levels.

CSinParallel takes the courseware approach seriously. The system enables to search for modules using a particular language or technology and for particular courses. Each module comes with an intro page that provides summaries, learning goals, and context for use. The modules come with a description in a consistent format for ease of crawling through the documents.

Some modules come with instructor notes, but most do not. Also the modules are usually technically competent but provide very little motivation for students in term of interesting application, social contextualisation, or broader engagement strategies. Some of the modules refer to local files or local systems.

This collection hits some good marks and gets the strategy right. Though the effort lacks in offering a panel of engaging materials for different populations. And searching for most combination of language and course lead to a mostly empty module offerings in CSinParallel.

D. iPDC

iPDC [17] is an effort from Tennessee Tech to improve PDC education. The project published 7 unplugged and 9 plugged modules, mostly targetted at CS1 and CS2. The materials explain well concepts like data races, parallel loops, and synchronization. The plugged acitvities contain code snippets for C++ and Java relying on OpenMP (and for Java, Pyjama [18]). The modules do not have an instructor package, but the iPDC project runs training workshops for instructors who want to adopt the modules. The modules are annotated with covered PDC Concept and Bloom levels to help instructors identify which module may apply.

The modules are meant to be short, introductory, technical. As such, they typically do not attempt to be particularly engage or justify why the parallel techniques are necessary. Certainly, an instructor using these modules would have justify the need for techniques externally. Though their brevity is also an advantage as they can be inserted in a class without taking a significant amount of time.

E. EduWrench

EduWRENCH [9] is a set of self-paced modules to teach some of the concepts of parallel computing. The main strategy is to have textual description with activities which are based on simulation thanks to the SimGrid simulation framework [19]. The simulation can run large systems from the student laptop. And the activities are deployed on the student's machine thanks to Docker containers, which enable running activities in a portable way across operating systems. EduWRENCH provides mappings between the pedagogical modules and the NSF/IEEE-TCPP curriculum guideline for PDC.

By design the modules require no programming which helps with adoption as they rely on running, observing, and interacting pre-programed demos. This also means that by design the modules focus on teaching PDC concepts but lacks in teaching how to use them in practice.

The modules focus on delivering quality content without trying to contextualize the techniques and information to engage the students. As such eduWRENCH relies on external motivation for the student to complete the modules. The modules also do not contain additional information for instructors assuming that the instructors are already confortable teaching that material, contextualizing, and making practical programming assignments for it.

F. Full Course

The community also publishes courses in a format akin to the dump of the course generated by a Learning Management System (such as Canvas or Blackboard). One of the authors of this paper published a parallel computing course for undergraduate students [20], ITCS 3145, which we will take as an example.

It is good to have a complete course. The materials have been used and as such they have been tested and cover content at some depth. Hopefully the content has some engagement strategies and are paired with activities in ways that justify the need for the techniques. ITCS 3145 presented in [20] does some of that but not consistently and to the benefits of a varied group of students.

A course dump usually does very little to be adoptable by other instructors. ITCS 3145 does not contain instructor notes on how an instructor should approach the course structure and strategies. The course also assumes a particular programming language (C++) and a particular hardware environment (a MOAB-based computing cluster with particular implementations of MPI available). As such the dump of the course can be helpful for inspiration; it could be picked apart for parts, or adapted for local environment. But it is in no sense directly reusable. A course dump is also not particularly discoverable.

IV. BRIDGES

BRIDGES [21] is toolkit that has been developed by the authors of this paper to improve the engagement and motivation of CS majors in freshmen and sophomore level courses. BRIDGES does not focus on PDC topics or their adoption, but it focuses on data structures and algorithms topics, which

has similar difficulties: both are only used to solve large scale problems. The strategies deployed by BRIDGES apply to PDC education directly. We present them next.

BRIDGES provides engagement mechanisms through programming assignments that connect students to real-world datasets through an easy-to-use API and also supports visualizations of student generated data structures or algorithmic outputs. Visualization of graphs and quad-trees can be seen in Fig. 1. BRIDGES gives access to a variety of real-world datasets including book collections, maps, lyrics of songs, earthquakes, elevation maps, actor-movie relations, etc.

To be more widely accessible, the BRIDGES API is supported in 3 languages: C++, Java and Python, with full documentation, tutorials and support by its authors. To date, BRIDGES has impacted over 2000 students across roughly 20 colleges and universities in the United States through its adoption strategies. Feedback from extensive surveys and reflections of students and instructors have pointed towards a largely positive experience.

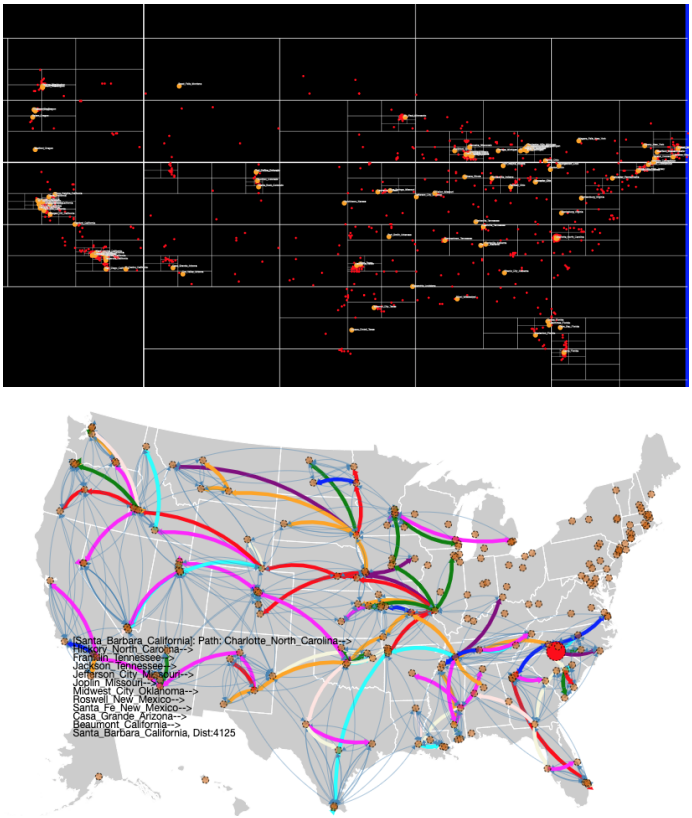

Fig. 1. BRIDGES Examples. (Top) Illustrates a data set of 1000 US cities with latitude/longitude data represented in a quadtree representation, (Bottom) Dijkstra's single source shortest paths algorithm applied to a subset of the same dataset with limited paths between states. The visualization shows the shortest path from Charlotte, North Carolina (big red circle) to all other cities with the labeled path to Santa Barbara, CA.

In order to better support CS instructors in early CS courses (CS1, CS2, Data Structures and Algorithm Analysis), the BRIDGES site also has an assignment repository [22] with a collection of assignments that can be used by students in any of these courses. Assignments have a description, goals (or learning outcomes), scaffolds or starter code in all three languages and expected outputs. Solutions are also available to adopters on request. Assignments are organized by topic as well as by course level. Additionally, assignments are carefully designed (some are adapted from Nifty assignments) to emphasize core CS concepts in data structures and algorithms, as well as concepts in CS1/CS2 using the Game Grid API. Finally, BRIDGES clearly provides context or application with each assignment, by using real-world data that helps students see the value of learning core CS concepts and algorithms. One can design courses using BRIDGES for most programming assignments in CS1 [23] or Algorithms [24].

While the BRIDGES assignment repository helps adopters to using them, it still leaves adopters facing challenges towards adoption. BRIDGES does provide good and technically strong programming assignments, but not complete modules (containing lectures, quiz questions, demonstrations, etc.) or full courses at this time.

BRIDGES assignments have not been designed yet to be customizable to a particular student population by ethnicity or gender. Though the BRIDGES framework is naturally constructed to be able to swap datasets in and out, enabling to potentially offer multiple equivalent assignments for each topic/learning outcome that would be of interest to different populations.

Finally, BRIDGES requires a bit of a learning curve (as can be expected from any new tool used in a class) to use it as part of an assignment; though the overhead of adopting BRIDGES is amortized for classes adopting multiple BRIDGES assignments as the setup can directly be reused.

V. CS Materials

Many of the PDC efforts listed in Section III suffer from a lack of discoverability. Though it does not seem that each module developer should reinvent strategies to be discovered. The problem is fundamentally that we need to make sure courseware get indexed properly so as to be easily identifiable. We present one of our efforts toward building such an index and explore the benefits of having such an index.

CS Materials [25], [26] is a system under development by the authors that allows educators to map their course materials to nationally accepted curriculum guidelines. The overall goal of CS Materials is to facilitate adoption of PDC materials in early CS courses (CS1, CS2, Data Structures, Algorithm Analysis), and in particular, find equivalent materials/algorithms that can be substituted with PDC content. In order to accomplish this goal, the system provides the means for CS instructors to assess their learning materials (lectures, quizzes, exams, assignments, videos, etc.) against topics and learning outcomes defined by accepted national guidelines. CS Materials currently uses the ACM 2013 Curriculum Guidelines [3] and the NSF/IEEE-TCPP 2012 PDC curriculum guidelines [1], however it is designed to be extended or updated as the guidelines change.

CS Materials allows sophisticated search and analysis by tags defined by the ACM 2013 guidelines, that are either knowledge units, topics or learning outcomes; this permits courses to be compared against each other, assess differences between two sections of the same course or ensure proper sequencing of topics within a program. An interactive interface for entering course materials and visualizations to see alignment and differences between materials (from individual material, modules or full courses) facilitate the analyses. Learning materials can also hosted by the system, ensuring that materials consistently stay within the system, avoiding the issues of broken links to materials, that are common in similar repositories.

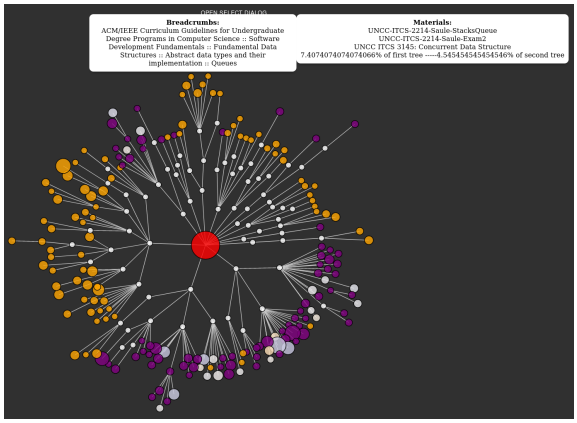

Fig. 2. Similarity between a data structure course and a parallel computing course. orange and purple nodes denote topics which are completely different while light blue nodes denote common topics.

Figure 2 present the common topics between two set of materials: the materials of a data structure course and the materials of a parallel computing course. The highlighted node is the ACM/IEEE topic for queues. We can see the data structure course covers queues in a lecture about Stacks and Queues and in an exam. The parallel computing class also covers queue in a lecture about concurrent data structure. This type of view can help an instructor identify from a set of vetted materials (for instance Peachy Parallel Assignments) which materials could be deployed in their class.

Once we have multiple definition of what a particular course is, one can build models of that course using CS Materials [25]. This can enable us, the PDC experts to think abstractly about data structures courses rather than think about one particular instance of a Data Structure course. Figure 3 shows how the PDC materials in CS Materials currently map to an abstract model of Data Structure extracted from DS courses currently in the system.

The strengths of the CS materials system lie in terms of search and analyses of existing course materials, identifying possible locations for adding PDC materials, and assessing gaps in coverage, which are all quite useful; however, CS Materials depends on the community's robust contribution of materials of all elements that make up a course or module. Similar to other systems, it is not a magic bullet for building

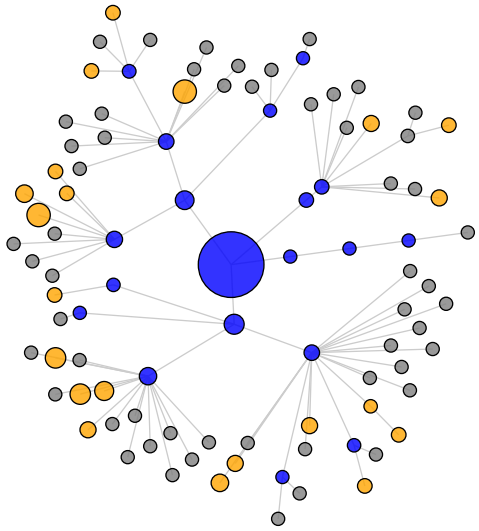

Fig. 3. Coverage of modeled data structure classification item by all PDC materials in CS Materials. Gray nodes are topics/learning outcomes that currently have no PDC materials mapped to them. The system has hover features to discover what these topics are.

full modules or courses; instructors will still require some effort to integrate the material into their course, and in some cases, more effort due to differences in programming language, conformance to their program's learning outcomes, etc.

Yet we believe that a system like CS Materials can help index the existing courseware that we produce as a community. More than simply indexing and mapping between materials and topics and learning outcomes, the system should probably also index across the dimensions presented in Section II. It will allow to account for engagement strategies (such as application domains) and adoption critical information (such as language and operating system).

VI. Discussion: What should the PDC community do?

A. *Better courseware covering more bases*

The courseware approach to the adoption of PDC in Computer Science curriculum relies on materials being available for instructor to import them into their courses. For materials to be integrated in courses across a wide range of universities, the materials need to hit many marks, delivering accurate information, engage students, be adoptable by instructors to form a coherent class, and are easily discoverable. Current efforts in the community do not hit all the marks needed for widespread adoption. But they are stepping stones and models for validated approaches to develop materials that can be adopted. We showed through the BRIDGES project that one can reuse lots of the core effort to engage a wide variety of students and institutions.

It is unlikely that we can develop materials that are technically excellent, applicable across dozens of contexts, require no instructor training or adoption effort, and engage students through different pedagogical styles, while also choosing problems that demonstrate social relevance to their lives; and at

the same time motivating the need for parallel computing. But what we can do is to enable a *compose and remix* approach to the problem. It would be much easier to compose a module from smaller pieces that include 1) a contextualization document that present convingcingly why a particular technique is technically and socially important (which could be a well crafted YouTube video), 2) an activity that helps students develop understanding at a conceptual level (for instance, from PDC unplugged or from eduWRENCH), 3) technical description of how that concept applies in a context that is localized (maybe to the language or execution environment), 4) and a reinforcing practical activity, such as using a Nifty, Peachy, or BRIDGES approach.

This would reduce the load on instructors significantly; materials for contextualization and conceptual understanding could be reused directly from different sources. Localized technical descriptions may need some light adaptation from existing material. Reinforcing practical activities are usually the ones that require the most careful attention and development effort; but that can be provided by a medium size community strike force. Decomposing the problem into multiple types and goals are needed to leverage expertise in the community from multiple sources, broadening the scope of potential contributors.

B. Connecting instructors with courseware

Multiple efforts have gone into developing materials for PDC education. The challenge is to connect the instructors with these materials. Collections of materials like PDCUnplugged, CSinParallel, or Peachy assignments are useful, but they are narrowly scoped.

To be able to quickly identify materials, we need to curate PDC educational content and index them, so as to be able to easily find materials that can be integrated within a particular class, accounting for goals (engagement, explanations, practice), topics, technologies, instructor knowledge, and student population. Systems like CS Materials that offer the combination of search and analyses capabilities to facilitate curation will be key. There may need to be community discussion to identify what dimensions need to go in such an index to be most helpful. For this to be succesful, this would require a significant effort from the community to contribute and classify their learning materials. But notice that such an effort requires a different type of skill than content creation. It enables a different type of expert to make meaningful contributions toward PDC adoption.

C. Addressing instructor gaps

Instructors can be wary of teaching content on which they are not experts. And not many CS instructors have receieved formal training in parallel computing, the majority having taken at most a single related class in graduate school. Acknowledging and addressing that issue upfront will be critical.

One approach that is centered around courseware lies in developing *instructor packages* that explain how to teach the content and give more context and more complex information. The idea is to bring the instructor to a higher level of understanding of the content. This is done in many US K-12 schools where teachers use well crafted packages on topics they lack deep or foundational understanding. Some of the chapters in the CDER books [4], [5] are designed for the benefit of instructors. A second approach is to directly train instructors through workshops. This is an approach that has been used in efforts like iPDC, CSinParallel, and CDER who organize instructor training workshops. While this approach has trained hundreds of instructors, scaling to 30,000 CS professors (as per US Bureau of Labor Statistics) will require the trained instructors train other instructors themselves.

Both instructor packages and workshop approaches will require coordinated efforts and commitments from the community, to make sure the efforts synergize and are communicated widely for maximum impact.

D. Convincing local programs

All these strategies assume that instructors are convinced of the importance of integrating PDC in their courses, have the will and time to perform this integration, and are allowed to perform the change. But in many cases, radical changes like this only happen once the degree programs change.

To enable such a change, we will need to engage with a different set of actors: program and department administrators, program educational committees, certification agencies (SACS, ABET). The community will need to gather the evidence to make the case from technical, social, and economical perspectives, so as to convince these actors.

VII. Conclusion

The courseware approach to PDC adoption has delivered slower than expected outcomes. We presented in this position paper the factors that we believe contribute to the slow adoption. While this paper certainly has a US bias, we believe that the core issues and solutions apply globally.

We will not succeed as disjoint small groups of experts. We need to pull efforts together and form a concerted front to address the PDC adoption challenge as content developers, content remixers, content curators, PDC course designers, instructor trainers, and public advocators.

Acknowledgments

This work is supported by grants from the National Science Foundation (CCF-1652442, DUE-1726809, and OAC-1924057).

References

[1] NSF/IEEE-TCPP Curriculum Working Group, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing : Core topics for undergraduates," CDER, Tech. Rep., 2012, available at http://www.cs.gsu.edu/~tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-version1.pdf.

[2] S. K. Prasad, T. Estrada, S. Ghafoor, A. Gupta, S. C. Kant, K., A. Sussman, R. Vaidyanathan, C. Weems, K. Agrawal, M. Barnas, D. W. Brown, R. Bryant, D. Bunde, C. Busch, D. Deb, E. Freudenthal, J. Jaja, M. Parashar, C. Phillips, B. Robey, A. Rosenberg, E. Saule, and C. Shen, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing -core topics for undergraduates, Version II-beta," CDER, Tech. Rep., 2020, available at https://tcpp.cs.gsu.edu/curriculum/?q=system/files/TCPP%20PDC%20Curriculum%20V2.0beta-Nov12.2020.pdf.

[3] Joint Taskforce on ACM Curricula, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM/IEEE Computer Society, 2013. [Online]. Available: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

[4] S. Prasad, A. Gupta, A. Rosenberg, A. Sussman, and C. Weems, Eds., *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann,, 2015.

[5] ——, *Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*. Springer International Publishing, 2018.

[6] "Peachy parallel assignments," https://grid.cs.gsu.edu/∼tcpp/curriculum/?q=peachy.

[7] S. J. Matthews, "PDCunplugged: A free repository of unplugged parallel distributed computing activities," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 284–291.

[8] R. Brown, L. Shoop, and J. Adams, "CS in parallel," https://csinparallel.org/.

[9] H. Casanova, R. Tanaka, W. Koch, and R. Ferreira da Silva, "Teaching parallel and distributed computing concepts in simulation with wrench," *Journal of Parallel and Distributed Computing*, vol. 156, pp. 53–63, 2021.

[10] B. Jones, "Motivating students to engage in learning: The music model of academic motivation," *International Journal of Teaching and Learning in Higher Education*, vol. 21, no. 2, pp. 272–285, 2009.

[11] P. Drake and K. Sung, "Teaching introductory programming with popular board games," in *Proc. of ACM SIGCSE*, ser. SIGCSE '11, 2011, pp. 619–624.

[12] K. Sung, R. Rosenberg, M. Panitz, and R. Anderson, "Assessing game-themed programming assignments for CS1/2 courses," in *Proc. of GDCSE*, ser. GDCSE '08, 2008, pp. 51–55.

[13] M. Buckley, H. Kershner, K. Schindler, C. Alphonce, and J. Braswell, "Benefits of using socially-relevant projects in computer science and engineering education," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004, pp. 482–486.

[14] M. Goldweber, J. Barr, T. Clear, R. Davoli, S. Mann, E. Patitsas, and S. Portnoff, "A framework for enhancing the social good in computing education: a val ues approach," *ACM Inroads*.

[15] N. Parlante, "Nifty assignments," 2018. [Online]. Available: http://nifty.stanford.edu/

[16] Computer Science Education Research Group , https://csunplugged.org/en/.

[17] S. Ghafoor, M. Rogers, D. Brown, and A. Haynes, "Integrating parallel and distributed computing in introductory programming classes," https://www.csc.tntech.edu/pdcincs/.

[18] Vikas, N. Giacaman, and O. Sinnen, "Pyjama: Openmp-like implementation for java, with gui extensions," in *Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 43–52. [Online]. Available: https://doi.org/10.1145/2442992.2442997

[19] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Simgrid: a sustained effort for the versatile simulation of large scale distributed systems," arXiv, Tech. Rep. 1309.1630, 2013.

[20] E. Saule, "Experiences on teaching parallel and distributed computing for undergraduates," in *Proc of IPDPSW 2018*, May 2018.

[21] K. Subramanian, E. Saule, and J. Payton, "BRIDGES (Bridging Real-world Infrastructure Designed to Goal-align, Engage, and Stimulate),," 2021. [Online]. Available: http://bridgesuncc.github.io/

[22] ——, "BRIDGES Assignment Repository," 2021. [Online]. Available: http://bridgesuncc.github.io/newassignments.html

[23] A. Beckman, M. Mcquaigue, A. Goncharow, D. Burlinson, K. Subramanian, E. Saule, and J. Payton, "Engaging early programming students with modern assignments using bridges," in *Proc. CCSC CP*, 2020.

[24] J. Strahler, M. Mcquaigue, A. Goncharow, D. Burlinson, K. Subramanian, E. Saule, and J. Payton, "Real-world assignments at scale to reinforce the importance of algorithms and complexity," in *Proc. CCSC NE*, 2020, conference.

[25] A. Goncharow, M. Mcquaigue, E. Saule, K. Subramanian, P. Goolkasian, and J. Payton, "CS-Materials: A system for classifying and analyzing pedagogical materials to improve adoption of parallel and distributed computing topics in early cs courses," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 316–330, 2021.

[26] E. Saule, K. Subramanian, and J. Payton, "CS Materials," 2021. [Online]. Available: https://cs-materials.herokuapp.com/