EARLY ADOPTER PDC Modules for Every Level:

A Comprehensive Model for Incorporating PDC Topics into the Existing Undergraduate Curriculum

EduPar Poster · May 2011 · Anchorage, AK, USA

Konstantin Läufer (presenter)
Chandra Sekharan (dept. chair)
George K. Thiruvathukal
Loyola University Chicago

Institutional Profile

- Loyola U. Chicago: urban, private, Jesuit, liberal arts, ~16k
  - College of Arts and Sciences, ~8k
    - Department of Computer Science, ~200

- 9 full-time faculty
  - 8 CS (7 TTT, 1 clinical)
  - 1 bioinformaticist (1/2 FTE)
  - 1 algebraist (1/2 FTE)

- 100+ undergrad majors in *CS*, *SE*, IT, Networks/Security
- 80+ master's students in *CS*, *SE*, IT

- External funding: NSF S-STEM, NSF BPC lead institution, NSF research grants, industry grants and donations

Carnegie Classification

Level 4-year or above
Control Private not-for-profit

Undergraduate Instructional Program: Bal/HGC
Graduate Instructional Program: CompDoc*/MedVet
Enrollment Profile: MU
Undergraduate Profile: FT4/MS/HTI
Size and Setting: L4/R

Basic: RU/H: Research Universities (high research activity)

Community Engagement: Curricular, Outreach, Partnerships

*CS/SE/IT: up to masters' level

Where Our Graduates Go...

- Industry 80%*
  - midwest, coasts, international
  - consulting, finance, software, telecom, ...
- Academia and Government 15%*
  - Argonne, county admin, local universities
- Graduate School 3%*
  - local, national
- Professional Schools 2%*
  - business, law, medical

*guesstimates

We Are Very Early PDC Adopters

- We have been teaching our students explicit PDC topics since spring 1997.
- Active research program in relevant areas
- NSF research grants
- Industry grants and donations
- Thiruvathukal's book *High Perf. Java Platform Computing* (lives on at hpjpc.googlecode.com)
- Paper in OOPSLA 1998 Educators' Symposium
- We are eager supporters of the NSF/IEEE-TCPP Curriculum Initiative!

The Near Future: Our Proposed Set of Required Core Modules

Goal: regularly and consistently expose all undergraduate majors to PDC core knowledge

Approach:

- push down into *required existing* 2nd-year foundation courses

identify suitable topics from *TCPP 45h sample course* (mostly “K” and “C” level, some “A”)

- package as three-week core PDC modules (20% of our 15-week semester or 30% of a 10-week quarter = 9 hours) → 36h total

Common Undergraduate Foundation

- Calculus I
- CS0 + CS1 + CS2 *(Core)*
- Discrete Structures *(Core, DM)*
- CS 264: Intro to Computer Systems *(Core, Systems)*
- CS 313: Intermediate Object-Oriented Dev (CS & SE only)
- Intro to Scientific and Technical Communication
- Social, Legal, and Ethical Issues in Computing
- Practicum (6 credits, in-house or external)

Other Relevant Existing Courses

- CS 363: Design & Analysis of Comp Alg *(Core, DS/A)*
- CS 372: Programming Languages *(Advanced, Lang)*
- CS 330: Software Engineering *(Advanced, SwEngg)*

PDC Core Module: *Introduction to PDC*

- every semester
- in CS2
- intro topics (arch, prog, algo, “K” and “C” level)
  - target machine models (1.5h)
  - parallel control statements (1.5h)
  - shared memory language extensions & libraries (1.5h)
  - tasks, threads, and synchronization (3h)
  - searching and sorting (1.5h)
- C# as the teaching language (at least for this module)
  - well-designed mechanisms that support these topics
  - foundationally sound teaching materials [Ball et al.]
  - cross-platform via Mono Project

PDC Core Module: *Architecture*

- every fall
- in CS 264 (Systems)
- architecture topics
  - high-level themes (1.5h)
  - classes (4.5h)
    - taxonomy
    - data versus control parallelism
    - shared versus distributed memory
  - memory hierarchy, caches (1h)
  - floating-point representation (0.5h)
  - performance metrics (1h)
  - power Issues (0.5h)

PDC Core Module: *Programming*

- every semester (CS & SE majors)
- in CS 313 (intermediate object-oriented development)
- programming topics *(some up to "A" level)*
  - selected parallel programming notations (1.5h)
  - semantics and correctness issues (4.5h)
    - tasks and threads
    - synchronization
    - defects
  - performance issues (1.5h)
  - tools (1.5h)
- C# as the teaching language for the entire course
  - threads, actors, tasks
  - events
  - software transactional memory

PDC Core Module: *Algorithms*

- every spring (CS majors)
- in CS 363 (Algo)
- algorithm topics
  - parallel/distributed models and complexity (4h)
    - cost of computation, scalability: asymptotics, time, cost, work, speedup, efficiency, space, power
  - algorithmic paradigms (3h)
    - divide and conquer, recursion
    - series-parallel composition
  - algorithmic problems (2h)
    - synchronization
    - specialized computations

The Future: *Our Proposed Set of Advanced/Elective Modules*

- slated for development after core modules
- each module typically offered every three semesters
- in suitable electives (from list on slide *"The Present"*)
- Advanced Programming: parallel prog. and concurrency topics from PL principles and paradigms perspective, using F# or Scala for programming projects
- Distributed Foundations: foundational topics including architecture classes, models and complexity, and concurrency topics
- Distributed Programming and Applications: languages, frameworks, and software architectures for distributed computing, semantics and correctness issues, performance issues, and advanced topics

Tying Everything Together: Roadmap

- summer 2011: develop the core PDC modules
- fall 2011: start offering core PDC modules
- starting summer 2011: develop PDC modules for advanced/elective courses

- key aspect of this proposal
- qualitative and quantitative measurement
- longitudinal measurement over three to five years
- refine our evaluation plan further by working with
  - TCPP and fellow early adopters
  - Loyola's Center for Science & Math Education
- hold workshops for subsequent adopters in the Midwest

Position Statement

[for further discussion]

To teach PDC topics effectively, they should not be taught in isolation. Instead, they should be taught in conjunction with relevant software engineering best practices.

Examples ← “How do you know?” ← Talk to the Practitioners!

- methodology/process
- software architecture
- software design patterns
- automated testing
- continuous integration
- collaboration/social coding/FOSS
- languages: object-oriented, functional, scripting, parallel
- tools: IDE, (D)VCS, build manager, doc generator