Parallelism In the Computer Science Curriculum

At a Four-Year Liberal Arts College

Steven Bogaerts, Brian Howard, Scott Thede, and Gloria Townsend
Department of Computer Science
DePauw University
Greencastle, IN, USA

DEPAUW UNIVERSITY

Introduction

Our theme in introducing parallelism and distributed computing (PDC) concepts into our curriculum is to focus on *integration*, as opposed to *insertion*. Historically, any PDC content in an undergraduate curriculum appeared in the context of operating systems, and perhaps algorithms or an upper-level elective. It is undeniable that PDC is of such significance today that it cannot be relegated solely to such courses.

Here, we outline our methodology and experiences of integrating PDC into undergraduate courses at various levels, in connection with the proposed core curriculum on PDC. We show that each course carries ample opportunities to consider PDC in a manner that is both level-appropriate and in harmony with the traditional topics of the course.

CS1
1st year, core, 54 students

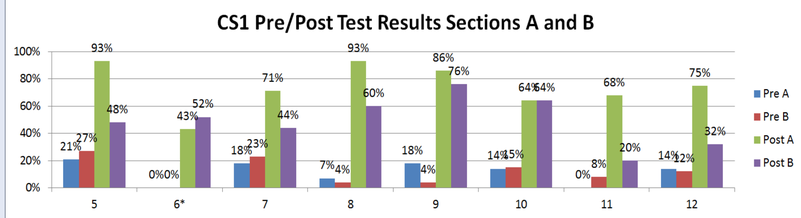
Syllabus

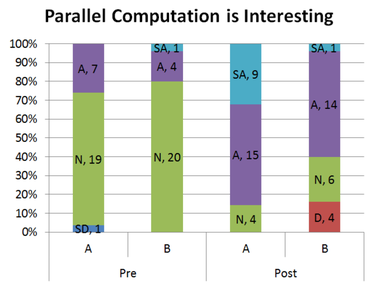
- Introductory programming in Java: classes, selection, iteration, arrays
- Parallelism - Discussion and non-technical exercises, some Java code exploration

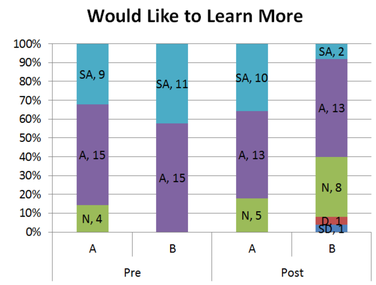
Curriculum Initiative Topic Coverage	Section A	Section B
Architecture – Multi-core	K	K
Programming – Shared Memory	C	K
Programming – Distributed Memory	C	K
Programming - Task / Thread Spawning	A	K
Programming – Tasks and Threads	C	K
Programming – Synchronization	C	K
Programming – Concurrency Defects	C	K
Programming – Performance Metrics	K	K
Cross-Cutting – Why and What is PDC?	C	C
Cross-Cutting – Non-Determinism	K	K
Cross Cutting – Power Consumption	K	K

Integration

- Section A: About 4 hours of contact time
  Section B: about 1.5 hours of contact time
- **Parallelism in real life**
  - Shoveling driveway - embarrassingly parallel
  - Digging a small hole - harder to divide the task
  - Running a company - lots of communication required
- **Inter-process communication**: counting the people in a building
  - *Passing data at process creation*: "I've counted 27 people in the basement. Please go count the other floors and determine the total."
  - *Message-passing*: "Whenever you finish a floor, send me a text message with that result. I'll merge your results with mine."
  - *Shared memory*: "Whenever you finish a floor, open up our shared Google Doc. Add what you just counted to the total already in the doc, and erase the old total. I'll do the same as I go."
- **Race conditions**
  - Interleaving of two processes executing critical section `x++;`
- **Locks**
  - Lord of the Flies conch analogy
  - Protecting critical sections
  - Deadlock
- **Join**
  - Analogy: On a walk with children, they stop to pick some flowers, eventually I tell them to `join` me as I continue on the walk. I wait for them to do so.
- **CS1Thread Java class**
  - Handles details of private Thread field for start(), join(), sleep(int).
  - Class must be extended, run() defined.







Evaluation Results

- Students in both sections learned more about parallelism
- Section A, having spent more time, shows stronger learning outcomes.
- Section A shows *stronger increase in interest*, desire to learn more.

Conclusion

If sufficient time is not available to cover deeply a given set of parallel concepts, these results suggest it may be preferable to reduce the size of the set, rather than to cover the entire set at a more shallow level.

Parallel Programming and Advanced Algorithms
3rd / 4th year, elective, 15 students

Syllabus

- Divide & Conquer, recursion, reduction, graph algorithms
- Functional language parallelism (Erlang)
- Parallelism: shared memory, message passing, synchronization, deadlock, dependencies, performance

Curriculum Initiative Topic Coverage	Level
Architecture – Message Passing	A
Programming – Distributed Memory	A
Programming – Functional / Logic Languages	A
Algorithm – Asymptotics	C
Algorithm – Time	C
Algorithm – Speedup	C
Algorithm – Space Compression	C

Integration

- Not quite half the time in the course (approximately 16 hours) was devoted to parallel topics. Of this time, roughly 6 hours were spent learning to program in Erlang (without parallelism). Thus, there were approximately 10 hours spent focusing on parallel topics.

Evaluation

- Surveys were given to the students. However, these surveys were created before the content of the course had been finalized. In particular, many of the questions on the surveys focused on memory management and collision issues, which do not occur in Erlang. Therefore, results of the surveys were not indicative, in our opinion, of the learning that occurred in the course.

Paradigm Shifts in Science: Parallelism Module
1st year, elective, 40 students

Syllabus

- An interdisciplinary introduction to paradigm shifts and the discovery process in various science fields. One module was on parallelism, covering basic terminology and history, inter-process communication via message passing and shared memory, modeling real-world phenomena, data races, deadlock, and locks.

Curriculum Initiative Topic Coverage	Level
Algorithms – Broadcast	A
Architecture – Multicore	C
Architecture – Shared vs. Distributed Memory	A
Programming – Shared Memory	A
Programming – Message Passing	A
Programming – Tasks and Threads	C
Programming – Critical Regions	C
Programming - Deadlocks	C
Programming – Data Races	C
Cross-Cutting – Why and What is PDC?	C

Integration

- The module was done in the context of an interdisciplinary science course, with no expectation of prior student knowledge in programming, much less parallelism.
- Some topics were introduced through hands-on physical exercises and analogies:
  - A system for making peanut butter and jelly sandwiches
  - Shoveling a driveway
  - Sorting cards in ways mimicking a single-core processor working alone, 2 processors communicating via message passing, and a dual-core processor communicating via shared memory
  - Counting and adding in binary using students as simple on/off switches
- Programming exercises were done in Scratch
  - Basic programming concepts like variables, if statements, and loops
  - Communication between actors working simultaneously, via broadcast
  - Detecting and correcting data races

Data Structures
1st / 2nd year, core, 50 students

Syllabus

- Abstract data types: lists, stacks, queues, hash tables, trees
- Object-oriented implementation, recursion, basic algorithm analysis
- Parallelism: sorting, searching, recursive divide-and-conquer algorithms

Curriculum Initiative Topic Coverage	Level
Programming – Shared Memory	C
Programming – Distributed Memory	C
Programming – Data Parallel	A
Programming – Synchronization	A
Algorithm – Speedup	A
Algorithm – Divide and Conquer	A
Algorithm - Recursion	A

Integration

- Divide and conquer parallelism as further practice of recursion, with use of stacks and trees for modeling of the computation.
- An analogy of cutting potatoes in parallel in a kitchen