Parallel Programming:
Design of an Overview Class

Christoph von Praun
University of Applied Sciences
Nuremberg, Germany
praun@acm.org

This work was supported by an IBM Innovation Award grant.
Summary

• Design of a 3rd year introductory parallel programming class in the Bachelor curriculum: ‘Orientation’ class

• Key characteristics of the class
  – Organization of topics follows the Tiers of parallelism
  – Uses programming language X10
  – Strong focus on lab sessions

• Teaching materials are available online
Computer Science

**Master + Bachelor Curriculum**

### Elective classes

- **The Art of Multiprocessor Programming**
- **Graphics Programming with CUDA**
- **Scientific Computing**
- **Parallel Programming**
- 'Orientation' class

### Semester

- **1**
  - thesis
- **2**
  - thesis
- **3**
  - thesis
- **4**
  - thesis
- **5**
  - thesis
- **6**
  - thesis
- **7**
  - thesis

*X10 Workshop, San Jose - June 4, 2011*
Influences on parallel programming classes

- Scientific computing
- High-performance computing
- Computer architecture
- Software architecture
- OS
- Programming languages

Parallel programming
Influences on parallel programming classes

- Scientific computing
- Computer architecture
- High-performance computing
- Operating system (OS)
- Software architecture
- Programming languages

Parallel programming
Outline

• Tiers of parallelism
• Course structure and contents
• Role of X10
• Student feedback and experience
Tiers of parallelism

• Original idea due to Michael L. Scott:
  – “Don’t start with Dekker’s algorithm …” [1]
  – “Making the simple case simple” [2]

• Development of parallel software (parallel programming) can be based on techniques at different abstraction layers
  – progressively less complexity at higher abstraction layers
<table>
<thead>
<tr>
<th>parallelization</th>
<th>techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) automatic or implicit</td>
<td>parallelizing compiler</td>
</tr>
<tr>
<td>(2) deterministic</td>
<td>fully independent computations or serialization</td>
</tr>
<tr>
<td>(3) explicitly synchronized</td>
<td>critical sections, transactions</td>
</tr>
<tr>
<td>(4) low-level (with race conditions)</td>
<td>implementation of threads, synchronization mechanisms, non-blocking data structures</td>
</tr>
</tbody>
</table>

- **High-level** (simpler)
- **Low-level** (more complex)
### Parallelization Techniques

<table>
<thead>
<tr>
<th>Parallelization Techniques</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) automatic or implicit</td>
<td>parallelizing compiler</td>
</tr>
<tr>
<td>(2) deterministic</td>
<td>fully independent computations or serialization</td>
</tr>
<tr>
<td>(3) explicitly synchronized (data race free)</td>
<td>critical sections, transactions</td>
</tr>
<tr>
<td>(4) low-level (with race conditions)</td>
<td>implementation of threads, synchronization mechanisms, non-blocking data structures</td>
</tr>
</tbody>
</table>
Goal of the class:

Students should be conscious about ‘their’ tier when developing a parallel program.

Encourage students to move programming activity to higher tiers in the abstraction hierarchy.
Tier-1: automatic or implicit parallelism

• Auto-parallelization through compilers

• Parallel kernels: parallelism encapsulated in libraries
  – LAPACK, etc.

• Parallel frameworks: framework organizes parallelism, synchronization and communication, programmer supplies sequential kernels
  – Map-reduce
  – Web application frameworks, e.g. WebSphere
  – etc.
Tier-1: automatic or implicit parallelism

- **Auto-parallelization** through compilers
- **Parallel kernels**: parallelism encapsulated in libraries
  - LAPACK, etc.
- **Parallel frameworks**: framework organizes parallelism, synchronization and communication, programmer supplies sequential kernels
  - Map-reduce
  - Web application frameworks, e.g. WebSphere

Sequential semantics.
Tier-2: deterministic parallelism

• Independent computations:
  – parallel array languages (FORALL loops)
  – parallel containers (e.g., STAPL, Intel Concurrent Collections, Hierarchically Tiled Arrays)

• Concurrent computations with dependencies that follow deterministic idioms:
  – reduction, scan
Tier-2: deterministic parallelism

• **Independent computations:**
  – parallel array languages (FORALL loops)
  – parallel containers (e.g., STAPL, Intel Concurrent Collections, Hierarchically Tiled Arrays)

• **Concurrent computations with dependencies that follow deterministic idioms:**
  – reduction, scan

  **Semantics through serialization + sequential reasoning.**
Tier-3: explicitly synchronized, data-race-free

Three principal programming models

• **Event-based**

• **Thread-parallel with shared memory**
  – critical sections
  – condition variables

• **Message-based**
  – send/receive
  – collective communication
Tier-3: explicitly synchronized, data-race-free

Three principal programming models

• Event-based

• Thread-parallel with shared memory
  – critical sections
  – condition variables

• Message-based
  – send/receive
  – collective communication

Semantics through interleaving of program blocks
Tier-4: low-level, with race conditions

- Programming with shared memory
  - atomic load and store
  - atomic compare and swap
- Platform-specific (Java, X86, ...)
- Sequential consistency is often a simplifying assumption
  - e.g. teaching Dekker’s algorithm
Tier-4: low-level, with race conditions

- Programming with shared memory
  - atomic load and store
  - atomic compare and swap
- Platform-specific (Java, X86, ...)
- Sequential consistency is often a simplifying assumption
  - e.g. teaching Dekker’s algorithm

Semantics through interleaving of statements, possibly not sequentially consistent
Outline

• Tiers of parallelism
• Course structure and contents
• Role of X10
• Student feedback and experience
Roadmap of the class (15 weeks)

1. Motivation (1 week)

2. Principles (1 week)

3. Tier-1 (1 week)

4. Tier-1 (1 week)

5. Tier-2 (7 weeks)

6. Tier-3 (2 weeks)

7. Topics not addressed in this course

Tier-4 (1 + 2 week)
Motivation (1 week)

• Hardware trend:
  – Moore’s Law continues
  – frequency scaling limited by power density: multicores

• Performance: Software need to be parallel
  – challenges (Amdahl’s Law)
  – opportunities (Gustafson’s Law)

• Energy: Throughput-oriented computing can save energy

Lab session
• Pencil and paper
Principles (1 week)

• Simple model for concurrent computations
  – partial orders of operations
  – synchronization vs ordinary operations
  – happens-before relation
• Explain semantics of X10 language constructs
  – async
  – finish, for-async
  – atomic

Lab session
• Parallel prime number testing
Tier-4 (1 week)
[low-level with race conditions]

• Race conditions
• Non-determinacy
  – associative non-determinism (floating point)
  – atomicity violation: lost-update problem
• “Interleaving” semantics

Lab sessions
• Numeric integration
Tier-1 (1 week)
[automatic or implicit parallelism]

• Challenges of loop parallelization
  – intro to data dependencies
  – difficulties and limitations of dependence analysis on some loop scenarios

• Parallel frameworks
  – Map-Reduce, Web-applications

Lab sessions

• Development of Map-Reduce applications
  (framework provided)
Tier-2 (7 weeks)  
[deteministic parallelism]

Patterns for algorithmic problem decomposition:  
according to T. Mattson, B. Sanders, B. Massingill,  

- Data parallelism  
  - geometric decomposition, recursive data  
  - data locality issues  
- Task parallelism  
  - task parallel, divide and conquer  
  - task scheduling / load balancing issues
Lab sessions

- Data parallel
  - heat-transfer
  - matrix multiply
  - algorithms for reduction and prefix-sum

- Task parallel
  - map-reduce framework implementation
  - merge-sort
  - traveling salesman

Tier-2 (7 weeks)
[Deterministic parallelism]
Tier-3 (2 weeks)  
[explicitly synchronized]

- Pattern: Pipeline parallelism
- Producer-consumer communication through concurrent queues
  - critical sections
  - conditional synchronization

Lab sessions
- Array-based concurrent queue with explicit synchronization (atomic blocks)
Tier-4 (2 weeks)
[low-level with race conditions]

• Programming with race conditions
• Memory models (SC, TSO)

Lab sessions
• Lamport’s concurrent non-blocking queue
  (1 consumer / 1 produce non-blocking queue)
• Observe non-SC behavior of Java
Topics not addressed in the course

• Patterns for ...
  – ... locality / reducing data access latency
  – ... load balancing / distribution of work
  – ... enhancing parallelism
  – ... distribution data

• Performance debugging
Outline

• Tiers of parallelism
• Course structure and contents
• **Role of X10**
• Student feedback and experience
X10

• Pragmatic choice:
  – Syntax familiar to students: “extension of sequential Java”
  – Simple things can be expressed with succinct syntax
  – X10 can express programs at tiers (1)-(3)
    memory model not specified → use Java at tier (4)

• Class was not X10 ‘only’
  – students could choose their own language for projects
  – X10 language tutorial provided separately
Student feedback on X10

“The language should not be used in future classes, since parallel programming is simplified significantly, and for that reason, one does not run into issues and problems that occur when conventional programming languages are used for parallel programming.”

“Takes a while to be familiar with the type system / type inference.”

“Usability of X10 IDE needs to be improved” [March–June 2010]
Outline

• Tiers of parallelism
• Course structure and contents
• Role of X10
• Student feedback and experience
Has the course been well-structured and did the structure support your learning?

Feedback collected from 16/21 participants.
The number of topics and the volume of material presented in class was ...

![Diagram showing student feedback with categories: too few, perfect, too much. The data points are distributed around the categories with 2 for too few, 9 for perfect, and 5 for too much. The average (s) for the data points is 0.63.]

Student feedback (2/3)
Did the lab sessions help you to learn and understand the materials presented in class?

![Survey Results Diagram]

- **Never**
  - 1
  - 1

- **Sometimes**
  - 2
  - 2

- **Always**
  - 6
  - 7

**Practical Relevance of the Lecture Content**
- **Too little (1)**
- **Too much (5)**

**Exercises (Set)**
- **Inadequate (1)**
- **Deeply enriching (5)**

**Exercises (Content)**
- **1.56 (1)**
- **4.19 (5)**

**Please assign a course grade ranging from 1 (excellent) to 6 (sufficient).**
Criticism

• Focus of discussion on correctness, not performance

• Focus on ‘higher layers’ in the abstraction hierarchy
  – Less complex than lower tiers
  – Assumption: People educated in our school are more likely to do parallel programming at higher rather than lower tiers

• Language X10 not widely used in practice
Conclusions

- “Tiers of parallelism” is a fruitful concept
  - course structure
  - orientation for students
- Focus on lab sessions important
  - provided skeletons and solutions for every exercise
  - few students could choose their own language
    (typically much more complex than X10)
- X10 turned out to be very good choice
  - succinct expression of programs at different tiers
  - steep learning curve
Sources


Thank you for your attention.

Teaching materials are available at
http://www.in.ohm-hochschule.de/professors/praun/pp