

# Supporting Parallelism in Operating Systems & Programming Languages

## Course Introduction

Kornilios Kourtis

<kornilios.kourtis@inf.ethz.ch>

# What is the course about?

## How the system stack supports and manages parallelism

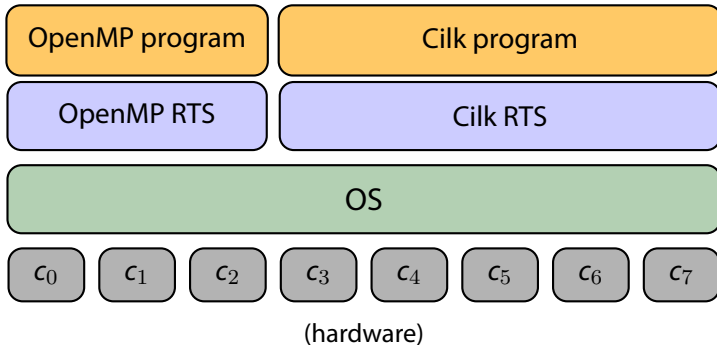
Two layers:

- ▶ Operating System (OS)
- ▶ Parallel Programming Languages (PPLs)

We will examine:

- ▶ each layer individually
- ▶ their synergy (or lack thereof)

## Context: the system stack



# Motivation

- ▶ Parallel machines dominate
  - ▶ parallel programming is difficult
  - ▶ system performance is critical  
(can't just wait for the next-generation faster processor)
- ▶ Cooperation of OS+PLEs
  - ▶ Usually, they are designed separately
  - ▶ OSEs evolved from uni-processors designs
    - ▶ e.g., typical OS designs manage processes/threads, not parallel programs
  - ▶ Parallel programming was not used in general-purpose machines
    - ▶ e.g., parallel programs assume they run alone on the machine

# Benefits from this course

Understand the internals of:

- ▶ parallel programming languages
- ▶ operating systems

But also, become better at parallel programming

# Course Organization

- ▶ **Lectures** (13<sup>00</sup>-15<sup>00</sup>)<sup>1</sup>
  - ▶ arranged by topics (scheduling, synchronization, etc)
  - ▶ design and implementation (this is not a theory course)
- ▶ **Exercises** (15<sup>00</sup>-16<sup>00</sup>)
  - ▶ apply (some of) the lectures' content in practice
- ▶ **Performance assessment**
  - ▶ exercises (≈ 70%)
  - ▶ 15' oral examination (≈ 30%)
- ▶ **Course webpage:**  
<http://www.systems.ethz.ch/courses/fall12013/SPOSPL>

---

<sup>1</sup>time can (and will) be dynamically managed as needed

# Prerequisites

## Familiarity with:

- ▶ basic OS designs
- ▶ systems programming
- ▶ computer architecture

## Relevant Courses by SG

- ▶ Systems Programming and Computer Architecture  
(<http://www.systems.ethz.ch/courses/fall2013/SPCA>)
- ▶ Operating Systems and Networks  
([www.systems.ethz.ch/courses/spring2013/os\\_networks](http://www.systems.ethz.ch/courses/spring2013/os_networks))
- ▶ Let me know if I should move slower (or faster!)

# Contact info

- ▶ mailing list
  - ▶ <https://lists.inf.ethz.ch/mailman/listinfo/spospl-course>
  - ▶ please subscribe
  - ▶ please don't answer questions (even if you know the answer)
  - ▶ don't post spoilers
  - ▶ try google first
    - ▶ it will probably answer faster!
- ▶ [akourtis@inf.ethz.ch](mailto:akourtis@inf.ethz.ch)



# Lectures

- ▶ about:
  - ▶ challenges posed by parallel hardware
  - ▶ design & implementation choices
  - ▶ interfaces (OS ↔ RTS)
- ▶ let's make the course highly interactive
  - ▶ Please interrupt me to ask questions
  - ▶ It will help me determine the best pace

# Lecture Topics Overview

- ▶ Parallelism
- ▶ Scheduling
- ▶ Synchronization
- ▶ Memory Management
  
- ▶ I/O
- ▶ Advanced Topics

# Lecture topics: Parallelism

- ▶ Hardware
  - ▶ current (and future) parallel hardware
- ▶ OS issues
  - ▶ parallelism inside the OS
  - ▶ OS interfaces
- ▶ Parallel programming languages
  - ▶ parallel programming constructs
  - ▶ run-time support

# Lecture topics: Scheduling

i.e., mapping of work units to execution units

- ▶ work units:
  - ▶ processes
  - ▶ kernel-level threads
  - ▶ user-level tasks
  - ▶ loop iterations
- ▶ load-balancing
- ▶ OS scheduling and interfaces
- ▶ Language RTS scheduling

# Lecture topics: Synchronization

- ▶ Primitives
  - ▶ low-level
    - ▶ spinlocks, mutexes, semaphores
    - ▶ barriers
    - ▶ RCU
  - ▶ high-level
    - ▶ Transactional Memory
    - ▶ Message Passing
- ▶ Scaling issues in OS and RTSES
- ▶ scheduling-synchronization issues

# Lecture topics: Memory Management

- ▶ Stack:
  - ▶ non-linear stacks
- ▶ Heap:
  - ▶ Allocation/management
    - ▶ e.g., `malloc()`, `free()`
  - ▶ Garbage Collection
  - ▶ Sharing Memory
  - ▶ NUMA (e.g., page migration OS policies)

# Exercises: Overview

- ▶ teams of 2
- ▶ design and implementation (in C)
- ▶ 2-3 week projects
- ▶ related to lecture material

# Exercise Format

- ▶ Initial exercise presentation
- ▶ First attempt to solve the problem
  - ▶ first implementation
  - ▶ intermediate report
  - ▶ check progress
- ▶ Discussion
  - ▶ first attempt discussion
  - ▶ questions/issues/problems
- ▶ Final implementation / report
  - ▶ performance measurements (?)
- ▶ <<for bonus points...>>
  - ▶ feel free to improvise!



# Programming environment for exercises

## Linux

- ▶ gcc
- ▶ GNU libc
- ▶ GNU/Linux-specific extensions are OK! (e.g., `-D_GNU_SOURCE`)
- ▶ pthreads
- ▶ use `-O2`
- ▶ please use `-Wall -Werror` gcc flags
- ▶ assume `x86_64` if needed  
(but: make sure to keep the arch-dependent parts separate)

# libraries/OS facilities

- ▶ man pages
- ▶ documentation (e.g., glibc)
- ▶ google
- ▶ source code
  - ▶ find, grep, ack
  - ▶ cscope
- ▶ ask (mailing list)

# Coding: Keep it Simple

- ▶ Debugging takes up most of the time
  - ▶ in parallel programming doubly so
- ▶ Break up your program to small, testable, units/versions
- ▶ Make sure that each of these units/versions work before proceeding
  - ▶ It will save you time in the long-run!

# Coding: Optimizations

- ▶ don't try optimizations
- ▶ really, **don't do it!**
- ▶ If you have to do it:
  - ▶ make sure that it works correctly first
  - ▶ where is the bottleneck?
  - ▶ keep original (simple) version
  - ▶ try optimization and compare
  - ▶ is it worth it?
- ▶ but, **don't do it!**

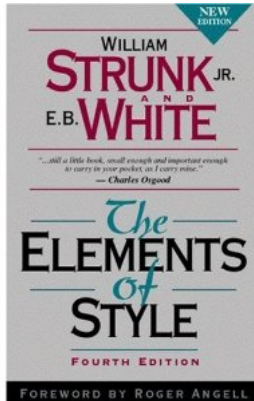
# Exercise Deliverables: Overview

- ▶ Deliverables
  - ▶ code
  - ▶ reports
- ▶ via a git tree
  - ▶ I will setup a git tree for each team
- ▶ Delaying deliverables
  - ▶ delayed intermediate reports are not graded
  - ▶ final reports lose half their grade for each delayed week

## Exercise Deliverables: Reports

- ▶ be concise
- ▶ pdf files (please do not send me .doc\* files)
- ▶ answer to questions
- ▶ brief discussion of design if asked
- ▶ things you found interesting
- ▶ selecting a way to present your data using graphs is part of the exercise
- ▶ be concise

(book suggestion)



## Exercise Deliverables: Code

- ▶ git repository

I might need to take a closer look, so please include:

- ▶ a Makefile for building it
  - ▶ typing make should build everything
  - ▶ ...cleanly
- ▶ readable source code
  - ▶ please read Documentation/CodingStyle of linux kernel
  - ▶ indent utility
- ▶ command line help
- ▶ scripts can also be helpful



# Poll

(1/2)

- ▶ Do you have access to a multicore linux machine for development?
  - ▶ laptop?
  - ▶ how many cores?
- ▶ Have you ever written a parallel program?  
if yes,
  - ▶ in what language/environment
  - ▶ have you ever written code that uses locking?
  - ▶ for more than one objects?
  - ▶ deal with deadlocks?
  - ▶ deal with scalability problems?

# Poll

(2/2)

- ▶ How comfortable are you with the linux programming environment?
  - ▶ do you know what the (2) stands for in read(2)?
  - ▶ experience with typical development tool-chain?
    - ▶ make, gcc, glibc, ...
  - ▶ experience with typical UNIX tools?
    - ▶ find, grep, ...
  - ▶ system calls? (e.g., mmap())
  - ▶ pthreads
  - ▶ can you find your way around the linux kernel/gcc/glibc code?
- ▶ Have you ever written assembly code?
  - ▶ Would you like to?

# Things to do ASAP

- ▶ Subscribe to the mailing list
- ▶ Find a partner for the exercises
  - ▶ Should I pair you randomly?

# Questions?