Parallel and Distributed Computing Topics in the Curriculum 2013 Core

Joel Adams

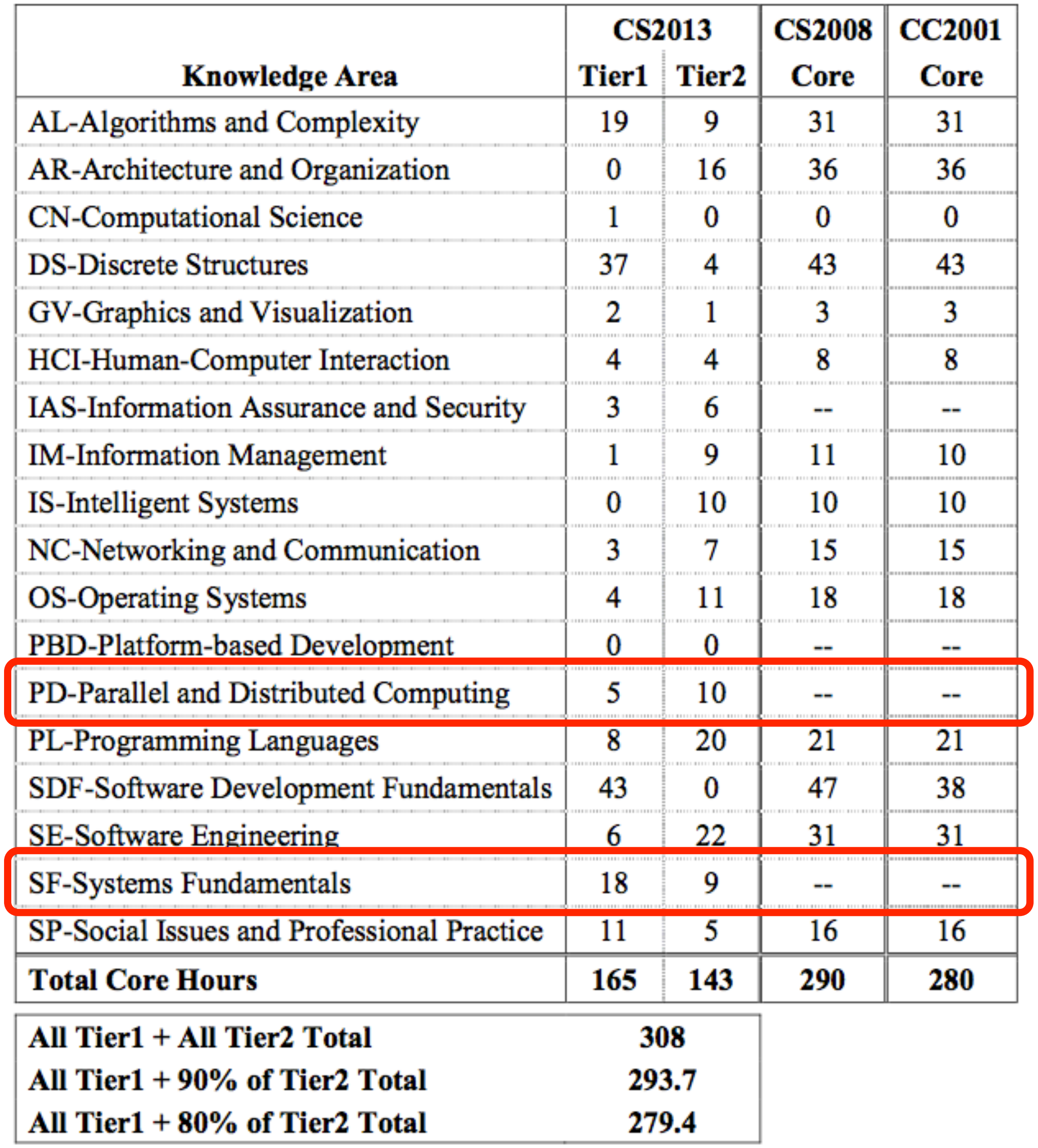
Department of Computer Science

Calvin College

CS Curriculum 2013 (CS2013)

- The CS2013 final report is available:
  - http://ai.stanford.edu/users/sahami/CS2013/
- To accommodate different kinds of programs, CS2013 spreads the CS "core" knowledge units (KUs) across two *tiers*:
  - All programs should cover 100% of the Tier 1 KUs
  - Programs should cover 80-90% of the Tier 2 KUs
- Different programs may decide to cover different Tier 2 KUs.

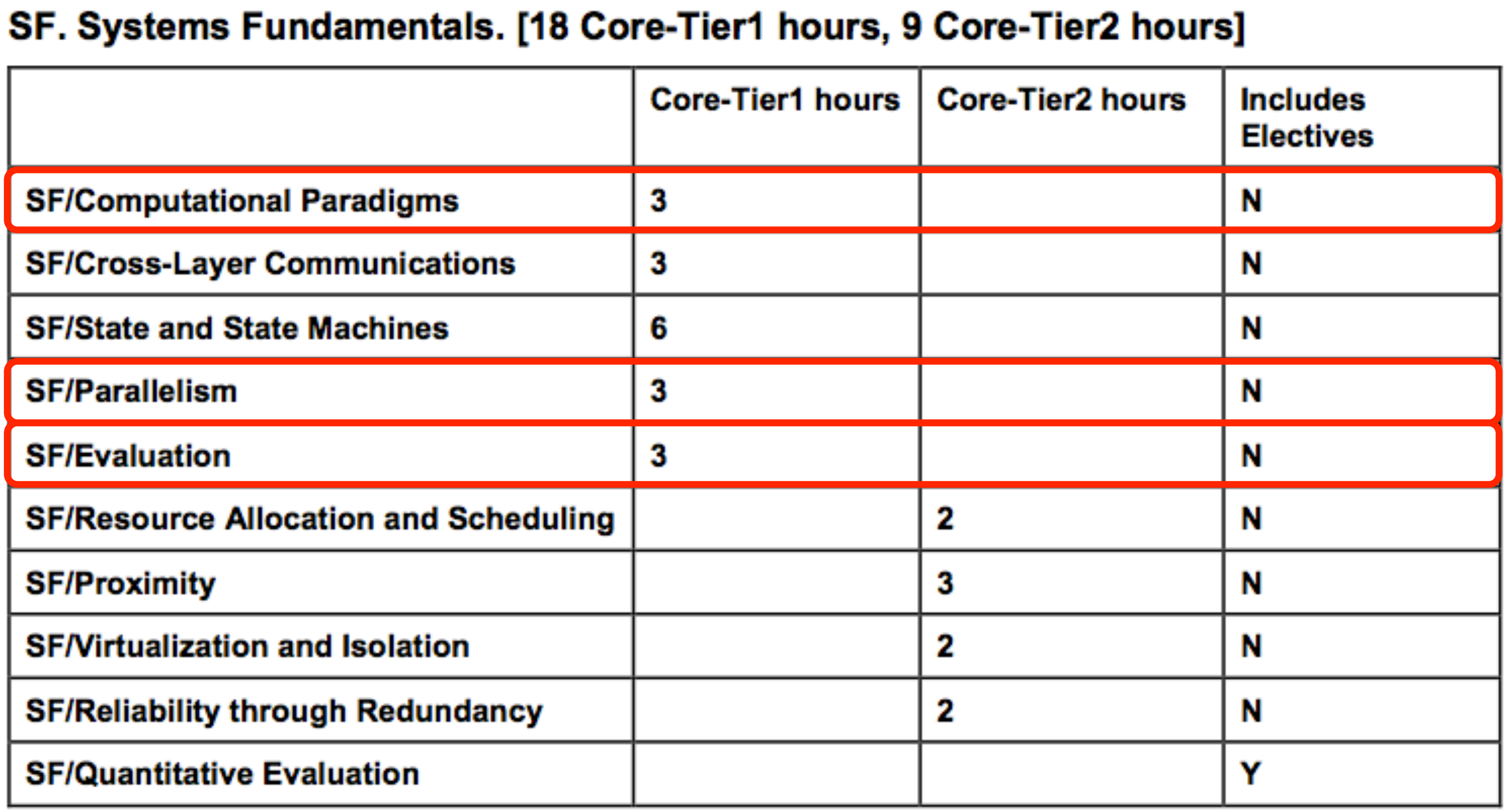
CS2013 Knowledge Areas (p. 40)



Knowledge Area	CS2013		CS2008	CC2001
	Tier1	**Tier2**	**Core**	**Core**
AL-Algorithms and Complexity	19	9	31	31
AR-Architecture and Organization	0	16	36	36
CN-Computational Science	1	0	0	0
DS-Discrete Structures	37	4	43	43
GV-Graphics and Visualization	2	1	3	3
HCI-Human-Computer Interaction	4	4	8	8
IAS-Information Assurance and Security	3	6	--	--
IM-Information Management	1	9	11	10
IS-Intelligent Systems	0	10	10	10
NC-Networking and Communication	3	7	15	15
OS-Operating Systems	4	11	18	18
PBD-Platform-based Development	0	0	--	--
PD-Parallel and Distributed Computing	5	10	--	--
PL-Programming Languages	8	20	21	21
SDF-Software Development Fundamentals	43	0	47	38
SE-Software Engineering	6	22	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social Issues and Professional Practice	11	5	16	16
Total Core Hours	**165**	**143**	**290**	**280**

All Tier1 + All Tier2 Total	**308**
All Tier1 + 90% of Tier2 Total	**293.7**
All Tier1 + 80% of Tier2 Total	**279.4**

What is in SF (p. 186)?



SF. Systems Fundamentals. [18 Core-Tier1 hours, 9 Core-Tier2 hours]

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
SF/Computational Paradigms	3		N
SF/Cross-Layer Communications	3		N
SF/State and State Machines	6		N
SF/Parallelism	3		N
SF/Evaluation	3		N
SF/Resource Allocation and Scheduling		2	N
SF/Proximity		3	N
SF/Virtualization and Isolation		2	N
SF/Reliability through Redundancy		2	N
SF/Quantitative Evaluation			Y

SF/Computational Paradigms

[3 Core-Tier1 hours]

The view presented here is the multiple representations of a system across layers, from hardware building blocks to application components, and the parallelism available in each representation. Cross-reference PD/Parallelism Fundamentals.

Topics:

- Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; Datapath + Control + Memory)
- Hardware as a computational paradigm: Fundamental logic building blocks; Logic expressions, minimization, sum of product forms
- Application-level sequential processing: single thread
- Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers
- Basic concept of pipelining, overlapped processing stages
- Basic concept of scaling: going faster vs. handling larger problems

SF/Computational Paradigms

[3 Core-Tier1 hours]

The view presented here is the multiple representations of a system across layers, from hardware building blocks to application components, and the parallelism available in each representation. Cross-reference PD/Parallelism Fundamentals.

Learning Outcomes:

1. List commonly encountered patterns of how computations are organized. [Familiarity]
2. Describe the basic building blocks of computers and their role in the historical development of computer architecture. [Familiarity]
3. Articulate the differences between single thread vs. multiple thread, single server vs. multiple server models, motivated by real world examples (e.g., cooking recipes, lines for multiple teller machines and couples shopping for food). [Familiarity]
4. Articulate the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem. This can be motivated by the simple, real-world examples. [Familiarity]
5. Design a simple logic circuit using the fundamental building blocks of logic design. [Usage]
6. Use tools for capture, synthesis, and simulation to evaluate a logic design. [Usage]
7. Write a simple sequential problem and a simple parallel version of the same program. [Usage]
8. Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved. [Assessment]

SF/Parallelism

[3 Core-Tier1 hours]

Cross-reference PD/Parallelism Fundamentals.

***Topics**:*

- Sequential vs. parallel processing
- Parallel programming vs. concurrent programming
- Request parallelism vs. Task parallelism
- Client-Server/Web Services, Thread (Fork-Join), Pipelining
- Multicore architectures and hardware support for synchronization

SF/Parallelism

[3 Core-Tier1 hours]

Cross-reference PD/Parallelism Fundamentals.

***Learning Outcomes*:**

1. For a given program, distinguish between its sequential and parallel execution, and the performance implications thereof. [Familiarity]
2. Demonstrate on an execution time line that parallelism events and operations can take place simultaneously (i.e., at the same time). Explain how work can be performed in less elapsed time if this can be exploited. [Familiarity]
3. Explain other uses of parallelism, such as for reliability/redundancy of execution. [Familiarity]
4. Define the differences between the concepts of Instruction Parallelism, Data Parallelism, Thread Parallelism/Multitasking, Task/Request Parallelism. [Familiarity]
5. Write more than one parallel program (e.g., one simple parallel program in more than one parallel programming paradigm; a simple parallel program that manages shared resources through synchronization primitives; a simple parallel program that performs simultaneous operation on partitioned data through task parallel (e.g., parallel search terms; a simple parallel program that performs step-by-step pipeline processing through message passing). [Usage]
6. Use performance tools to measure speed-up achieved by parallel programs in terms of both problem size and number of resources. [Assessment]

SF/Evaluation

[3 Core-Tier1 hours]

Cross-reference PD/Parallel Performance.

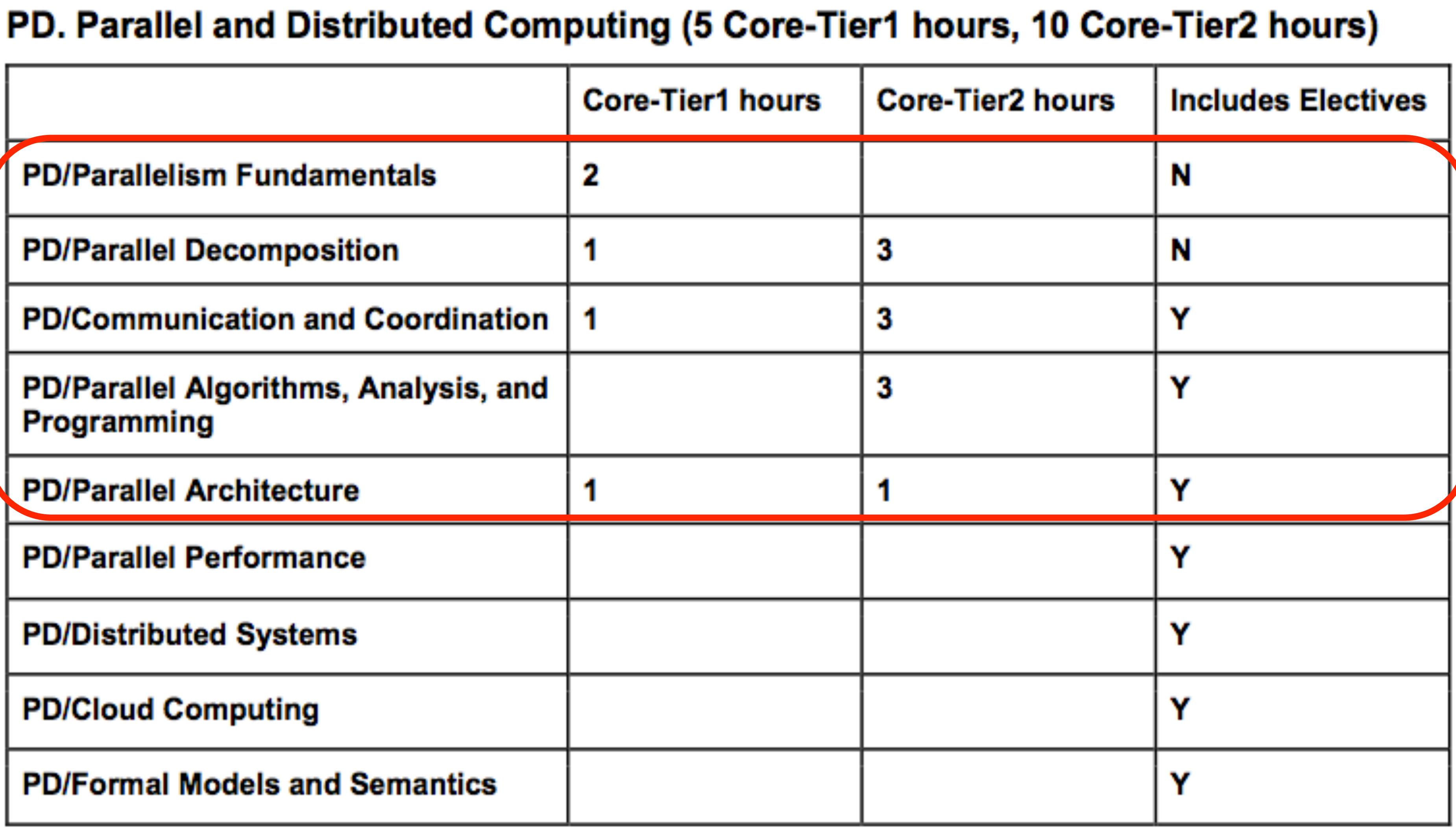
***Topics**:*

- Performance figures of merit
- Workloads and representative benchmarks, and methods of collecting and analyzing performance figures of merit
- CPI (Cycles per Instruction) equation as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
- Amdahl's Law: the part of the computation that cannot be sped up limits the effect of the parts that can

***Learning Outcomes**:*

1. Explain how the components of system architecture contribute to improving its performance. [Familiarity]
2. Describe Amdahl's law and discuss its limitations. [Familiarity]
3. Design and conduct a performance-oriented experiment. [Usage]
4. Use software tools to profile and measure program performance. [Assessment]

What is in PD (p. 147)?



PD. Parallel and Distributed Computing (5 Core-Tier1 hours, 10 Core-Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
PD/Parallelism Fundamentals	2		N
PD/Parallel Decomposition	1	3	N
PD/Communication and Coordination	1	3	Y
PD/Parallel Algorithms, Analysis, and Programming		3	Y
PD/Parallel Architecture	1	1	Y
PD/Parallel Performance			Y
PD/Distributed Systems			Y
PD/Cloud Computing			Y
PD/Formal Models and Semantics			Y

PD/Parallelism Fundamentals

[2 Core-Tier1 hours]

Build upon students' familiarity with the notion of basic parallel execution—a concept addressed in Systems Fundamentals—to delve into the complicating issues that stem from this notion, such as race conditions and liveness.

Cross-reference SF/Computational Paradigms and SF/System Support for Parallelism.

Topics:

- Multiple simultaneous computations
- Goals of parallelism (e.g., throughput) versus concurrency (e.g., controlling access to shared resources)
- Parallelism, communication, and coordination
  - Programming constructs for coordinating multiple simultaneous computations
  - Need for synchronization
- Programming errors not found in sequential programming
  - Data races (simultaneous read/write or write/write of shared state)
  - Higher-level races (interleavings violating program intention, undesired non-determinism)
  - Lack of liveness/progress (deadlock, starvation)

PD/Parallelism Fundamentals

[2 Core-Tier1 hours]

Build upon students' familiarity with the notion of basic parallel execution—a concept addressed in Systems Fundamentals—to delve into the complicating issues that stem from this notion, such as race conditions and liveness.

Cross-reference SF/Computational Paradigms and SF/System Support for Parallelism.

Learning outcomes:

1. Distinguish using computational resources for a faster answer from managing efficient access to a shared resource. (Cross-reference GV/Fundamental Concepts, outcome 5.) [Familiarity]
2. Distinguish multiple sufficient programming constructs for synchronization that may be inter-implementable but have complementary advantages. [Familiarity]
3. Distinguish data races from higher level races. [Familiarity]

PD/Parallel Decomposition

[1 Core-Tier1 hour, 3 Core-Tier2 hours]

(Cross-reference SF/System Support for Parallelism)

Topics:

[Core-Tier1]

- Need for communication and coordination/synchronization
- Independence and partitioning

[Core-Tier2]

- Basic knowledge of parallel decomposition concepts (cross-reference SF/System Support for Parallelism)
- Task-based decomposition
  - Implementation strategies such as threads
- Data-parallel decomposition
  - Strategies such as SIMD and MapReduce
- Actors and reactive processes (e.g., request handlers)

PD/Parallel Decomposition

[1 Core-Tier1 hour, 3 Core-Tier2 hours]

(Cross-reference SF/System Support for Parallelism)

Learning outcomes:

[Core-Tier1]

1. Explain why synchronization is necessary in a specific parallel program. [Usage]
2. Identify opportunities to partition a serial program into independent parallel modules. [Familiarity]

[Core-Tier2]

3. Write a correct and scalable parallel algorithm. [Usage]
4. Parallelize an algorithm by applying task-based decomposition. [Usage]
5. Parallelize an algorithm by applying data-parallel decomposition. [Usage]
6. Write a program using actors and/or reactive processes. [Usage]

PD/Communication and Coordination

[1 Core-Tier1 hour, 3 Core-Tier2 hours]

Cross-reference OS/Concurrency for mechanism implementation issues.

Topics:

[Core-Tier1]

- Shared Memory
- Consistency, and its role in programming language guarantees for data-race-free programs

[Core-Tier2]

- Message passing
    - Point-to-point versus multicast (or event-based) messages
    - Blocking versus non-blocking styles for sending and receiving messages
    - Message buffering (cross-reference PF/Fundamental Data Structures/Queues)
- Atomicity
    - Specifying and testing atomicity and safety requirements
    - Granularity of atomic accesses and updates, and the use of constructs such as critical sections or transactions to describe them
    - Mutual Exclusion using locks, semaphores, monitors, or related constructs
        - Potential for liveness failures and deadlock (causes, conditions, prevention)
    - Composition
        - Composing larger granularity atomic actions using synchronization
        - Transactions, including optimistic and conservative approaches

PD/Communication and Coordination

[1 Core-Tier1 hour, 3 Core-Tier2 hours]

Cross-reference OS/Concurrency for mechanism implementation issues.

Learning outcomes:

[Core-Tier1]

1. Use mutual exclusion to avoid a given race condition. [Usage]
2. Give an example of an ordering of accesses among concurrent activities (e.g., program with a data race) that is not sequentially consistent. [Familiarity]

[Core-Tier2]

3. Give an example of a scenario in which blocking message sends can deadlock. [Usage]
4. Explain when and why multicast or event-based messaging can be preferable to alternatives. [Familiarity]
5. Write a program that correctly terminates when all of a set of concurrent tasks have completed. [Usage]
6. Use a properly synchronized queue to buffer data passed among activities. [Usage]
7. Explain why checks for preconditions, and actions based on these checks, must share the same unit of atomicity to be effective. [Familiarity]
8. Write a test program that can reveal a concurrent programming error; for example, missing an update when two activities both try to increment a variable. [Usage]
9. Describe at least one design technique for avoiding liveness failures in programs using multiple locks or semaphores. [Familiarity]
10. Describe the relative merits of optimistic versus conservative concurrency control under different rates of contention among updates. [Familiarity]
11. Give an example of a scenario in which an attempted optimistic update may never complete. [Familiarity]

PD/Parallel Algorithms, Analysis, and Programming

[3 Core-Tier2 hours]

Topics:

[Core-Tier2]

- Critical paths, work and span, and the relation to Amdahl's law (cross-reference SF/Performance)
- Speed-up and scalability
- Naturally (embarrassingly) parallel algorithms
- Parallel algorithmic patterns (divide-and-conquer, map and reduce, master-workers, others)
    - Specific algorithms (e.g., parallel MergeSort)

PD/Parallel Algorithms, Analysis, and Programming

[3 Core-Tier2 hours]

Learning outcomes:

[Core-Tier2]

1. Define “critical path”, “work”, and “span”. [Familiarity]
2. Compute the work and span, and determine the critical path with respect to a parallel execution diagram. [Usage]
3. Define “speed-up” and explain the notion of an algorithm’s scalability in this regard. [Familiarity]
4. Identify independent tasks in a program that may be parallelized. [Usage]
5. Characterize features of a workload that allow or prevent it from being naturally parallelized. [Familiarity]
6. Implement a parallel divide-and-conquer (and/or graph algorithm) and empirically measure its performance relative to its sequential analog. [Usage]
7. Decompose a problem (e.g., counting the number of occurrences of some word in a document) via map and reduce operations. [Usage]

PD/Parallel Architecture

[1 Core-Tier1 hour, 1 Core-Tier2 hour]

The topics listed here are related to knowledge units in the Architecture and Organization (AR) knowledge area (AR/Assembly Level Machine Organization and AR/Multiprocessing and Alternative Architectures). Here, we focus on parallel architecture from the standpoint of applications, whereas the Architecture and Organization knowledge area presents the topic from the hardware perspective.

***Topics*:**

[Core-Tier1]

- Multicore processors
- Shared vs. distributed memory

[Core-Tier2]

- Symmetric multiprocessing (SMP)
- SIMD, vector processing

PD/Parallel Architecture

[1 Core-Tier1 hour, 1 Core-Tier2 hour]

The topics listed here are related to knowledge units in the Architecture and Organization (AR) knowledge area (AR/Assembly Level Machine Organization and AR/Multiprocessing and Alternative Architectures). Here, we focus on parallel architecture from the standpoint of applications, whereas the Architecture and Organization knowledge area presents the topic from the hardware perspective.

Learning outcomes:

[Core-Tier1]

1. Explain the differences between shared and distributed memory. [Familiarity]

[Core-Tier2]

2. Describe the SMP architecture and note its key features. [Familiarity]
3. Characterize the kinds of tasks that are a natural match for SIMD machines. [Familiarity]