Parallel Computing Topics
in the Computer Science Curricula 2013 (CS2013) Strawman Draft[1]

Joel Adams, Dick Brown, Libby Shoop, CSinParallel Workshop, SC 2012

To accommodate different institutional curricular models, CS2013 spreads its "core" topics between two tiers:

- Tier 1: All programs should cover all topics/hours in this tier.
- Tier 2: Programs should cover 80-90% of the topics/hours in this tier. (Departments can tailor that 80-90% to their local situation.)

From p. 33 of the Strawman document:

Knowledge Area	CS2013 Tier1	CS2013 Tier2	CS2008 Core	CC2001 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 Visual Computing	2	1	3	3
HC-Human-Computer Interaction	4	4	8	8
IAS-Security and Information Assurance	2	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	42	0	47	38
SE-Software Engineering	6	21	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social and Professional Issues	11	5	16	16
Total Core Hours	**163**	**142**	**290**	**280**

All Tier1 + All Tier2 Total	**305**
All Tier1 + 90% of Tier2 Total	**290.8**
All Tier1 + 80% of Tier2 Total	**276.6**

[1] http://ai.stanford.edu/users/sahami/CS2013/strawman-draft/cs2013-strawman.pdf

What is in the PD area? From p. 121 of the Strawman document:

PD. Parallel and Distributed Computing (5 Core-Tier1 hours, 9 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/Formal Models and Semantics			Y

What is in the SF area? From p. 157 of the Strawman document:

SF. Systems Fundamentals [18 core Tier 1, 9 core Tier 2 hours, 27 total]

	Core-Tier 1 hours	Core-Tier 2 hours	Includes Electives
SF/Computational Paradigms	3		
SF/Cross-Layer Communications	3		
SF/State-State Transition-State Machines	6		
SF/System Support for Parallelism	3		
SF/Performance	3		
SF/Resource Allocation and Scheduling		2	
SF/Proximity		3	
SF/Virtualization and Isolation		2	
SF/Reliability through Redundancy		2	

In its Appendix (*The Body of Knowledge*), the Strawman document explains each area of CS2103 in depth, including topics to be covered and learning outcomes.

SF is seen as more fundamental, so let’s examine it before we examine PD…

SF/Computational Paradigms

[3 Core-Tier 1 hours]

[Cross-reference PD/parallelism fundamentals: 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; PD/parallelism fundamentals focuses on the application structuring concepts for parallelism.]

Topics:

- A computing system as a layered collection of representations
- 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 gates, flip-flops, counters, registers, PL); Logic expressions, minimization, sum of product forms
- Application-level sequential processing: single thread [xref PF/]
- 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

Learning Outcomes:

1. List commonly encountered patterns of how computations are organized [Knowledge].
2. Describe the basic building blocks of computers and their role in the historical development of computer architecture [Knowledge].
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, couple shopping for food, wash-dry-fold, etc.) [Knowledge].
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 [Knowledge].
5. Design and simulate a simple logic circuit using the fundamental building blocks of logic design [Application].
6. Write a simple sequential problem and a simple parallel version of the same program [Application].
7. Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved [Evaluation].

SF/System Support for Parallelism

[3 Core-Tier1 hours]

[Cross-reference: PD/Parallelism Fundamentals]

Topics:

- Execution and runtime models that distinguish Sequential vs. Parallel processing
- System organizations that support Request and Task parallelism and other parallel processing paradigms, such as Client-Server/Web Services, Thread parallelism(Fork-Join), and Pipelining
- Multicore architectures and hardware support for parallelism

Learning Outcomes:

1. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers [knowledge].
2. Recognize that hardware, VM, OS, application are just additional layers of interpretation/processing [knowledge].
3. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers [knowledge]
4. Construct a simple program using methods of layering, error detection and recovery, and reflection of error status across layers [application].
5. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging [evaluation].

SF/Performance

[3 Core-Tier 1 hours]

[Cross-reference PD/Parallel Performance]

***Topics**:*

- Figures of performance merit (e.g., speed of execution, energy consumption, bandwidth vs. latency, resource cost)
- Benchmarks (e.g., SPEC) and measurement methods
- CPI equation (Execution time = # of instructions * cycles/instruction * time/cycle) 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 [Knowledge].
2. Describe Amdahl's law and its implications for parallel system speed-up when limited by sequential portions, e.g., in processing pipelines [Knowledge].
3. Benchmark a parallel program with different data sets in order to iteratively improve its performance [Application].
4. Use software tools to profile and measure program performance [Evaluation].

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:

[Core-Tier1]

- Multiple simultaneous computations
- Goals of parallelism (e.g., throughput) versus concurrency (e.g., controlling access to shared resources)
- Programming constructs for creating parallelism, communicating, and coordinating
- 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)
  - Lack of liveness/progress (deadlock, starvation)

Learning outcomes:

1. Distinguish using computational resources for a faster answer from managing efficient access to a shared resource [Knowledge]
2. Distinguish multiple sufficient programming constructs for synchronization that may be inter-implementable but have complementary advantages [Knowledge]
3. Distinguish data races from higher level races [Knowledge]

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)

Learning outcomes:

1. Explain why synchronization is necessary in a specific parallel program [Application]
2. Write a correct and scalable parallel algorithm [Application]
3. Parallelize an algorithm by applying task-based decomposition [Application]
4. Parallelize an algorithm by applying data-parallel decomposition [Application]

PD/Communication and Coordination

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

Topics:

[Core-Tier1]

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

[Core-Tier2]

- Consistency in shared memory models
- 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

[Elective]

- Consensus
  - (Cyclic) barriers, counters, or related constructs
- Conditional actions
  - Conditional waiting (e.g., using condition variables)

Learning outcomes:

1. Use mutual exclusion to avoid a given race condition [Application]
2. Give an example of an ordering of accesses among concurrent activities that is not sequentially consistent [Knowledge]
3. Give an example of a scenario in which blocking message sends can deadlock [Application]
4. Explain when and why multicast or event-based messaging can be preferable to alternatives [Knowledge]
5. Write a program that correctly terminates when all of a set of concurrent tasks have completed [Application]
6. Use a properly synchronized queue to buffer data passed among activities [Application]

7. Explain why checks for preconditions, and actions based on these checks, must share the same unit of atomicity to be effective [Knowledge]
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 [Application]
9. Describe at least one design technique for avoiding liveness failures in programs using multiple locks or semaphores [Knowledge]
10. Describe the relative merits of optimistic versus conservative concurrency control under different rates of contention among updates [Knowledge]
11. Give an example of a scenario in which an attempted optimistic update may never complete [Knowledge]
12. Use semaphores or condition variables to block threads until a necessary precondition holds [Application]

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 (embarassingly) parallel algorithms
- Parallel algorithmic patterns (divide-and-conquer, map and reduce, others)
    - Specific algorithms (e.g., parallel MergeSort)

[Elective]

- Parallel graph algorithms (e.g., parallel shortest path, parallel spanning tree) (cross-reference AL/Algorithmic Strategies/Divide-and-conquer)
- Producer-consumer and pipelined algorithms

Learning outcomes:

1. Define "critical path", "work", and "span" [Knowledge]
2. Compute the work and span, and determine the critical path with respect to a parallel execution diagram [application]
3. Define "speed-up" and explain the notion of an algorithm's scalability in this regard [Knowledge]
4. Identify independent tasks in a program that may be parallelized [Application]
5. Characterize features of a workload that allow or prevent it from being naturally parallelized [Knowledge]
6. Implement a parallel divide-and-conquer and/or graph algorithm and empirically measure its performance relative to its sequential analog [application]
7. Decompose a problem (e.g., counting the number of occurrences of some word in a document) via map and reduce operations [Application]
8. Provide an example of a problem that fits the producer-consumer paradigm [Knowledge]
9. Give examples of problems where pipelining would be an effective means of parallelization [Knowledge]
10. Identify issues that arise in producer-consumer algorithms and mechanisms that may be used for addressing them [Knowledge]

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 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 area presents the topic from the hardware perspective.

[Core-Tier1]

- Multicore processors
- Shared vs. distributed memory

[Core-Tier2]

- Symmetric multiprocessing (SMP)
- SIMD, vector processing

[Elective]

- GPU, co-processing
- Flynn's taxonomy
- Instruction level support for parallel programming
  - Atomic instructions such as Compare and Set
- Memory issues
  - Multiprocessor caches and cache coherence
  - Non-uniform memory access (NUMA)
- Topologies
  - Interconnects
  - Clusters
  - Resource sharing (e.g., buses and interconnects)

Learning outcomes:

1. Describe the SMP architecture and note its key features [Knowledge]
2. Characterize the kinds of tasks that are a natural match for SIMD machines [Knowledge]
3. Explain the features of each classification in Flynn's taxonomy [Knowledge]
4. Explain the differences between shared and distributed memory [Knowledge]
5. Describe the challenges in maintaining cache coherence [Knowledge]
6. Describe the key features of different distributed system topologies [Knowledge]