Literacy for All in Parallel and Distributed Computing: Guidelines for an Undergraduate Core Curriculum

Prasad, Sushil K. (Coordinator)[1], Gupta, Anshul[2], Kant, Krishna[3], Lumsdaine, Andrew[4], Padua, David[5], Robert, Yves[6], Rosenberg, Arnold[7], Sussman, Alan[8], Weems, Charles[9]

[1] Georgia State University
[2] IBM T.J. Watson Research Center
[3] Intel and George Mason University
[4] Indiana University
[5] University of Illinois at Urbana-Champaign
[6] ENS Lyon & INRIA, France
[7] Northeastern University
[8] University of Maryland
[9] University of Massachusetts

A working group composed of researchers from academia, government, and industry has formulated the first proposed core curricular guidelines on parallel and distributed computing (PDC). The goal of this effort is to ensure that all students graduating with a bachelor's degree in computer science/computer engineering receive an education that prepares them in the area of parallel and distributed computing, preparation which is increasingly important in the light of emerging technology. Instructors who adopt the proposed guidelines will receive periodically updated versions of the guidelines that identify aspects of PDC that are important to cover and that suggest specific core courses in which their coverage might find an appropriate context. Roughly four dozen early-adopter institutions worldwide are currently trying out this curriculum. The early adopters have been awarded stipends through three rounds of competitions (Spring and Fall 2011, and Spring 2012) with support from NSF, Intel, and NVIDIA. Additional competitions are planned for Fall 2012 and Fall 2013.[1] A Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) is being established to carry the work forward; much of the work in the Center is possible due to a new NSF grant.

Introduction

The penetration of parallel and distributed computer (PDC) technology into the daily lives of users via their wireless networks, smartphones, social networking sites and more, has made it imperative to impart a broad-based skill set in PDC technology at various levels in the educational fabric. However, the rapid advances in computing technology and services challenges educators' abilities to know what to teach in any given semester. Other stakeholders in the push to cope with fast-changing PDC technology, including employers, face similar challenges in identifying basic expertise.

The curricular guidelines developed by the working group[2] seek to address this challenge in a manner that is flexible and broad, with allowance for variations in emphasis in response different institutions and different curricular cultures. The field of PDC is changing too rapidly for any inflexible proposal to remain valuable to the community for any length of time. We strive, instead, to identify basic concepts and learning goals that are likely to retain their relevance for the foreseeable future.

The Preliminary Curriculum

Our initial work on a Parallel and Distributed Computing (PDC) curriculum occurred during a planning workshop sponsored by NSF and IEEE/TCPP in Feb., 2010. Building on the outcomes of that workshop, a working group has taken up the challenge of developing (and justifying) PDC curricular guidelines for Computer Science (CS) and Computer Engineering (CE) undergraduates, with particular emphasis on developing a core curriculum that identifies what every graduate should know about PDC. Throughout 2010, the working group deliberated upon various topics and subtopics, specifying both the expected minimum level of coverage and the desired learning outcomes; we employed the well-known Bloom classification as the medium for specifying the desired level of expertise on a topic. The group also developed suggestions

[1] All details (including deadlines) are posted at the *Curriculum Initiative Website at* http://www.cs.gsu.edu/~tcpp/curriculum/index.php. Contact Email: sprasad@gsu.edu.
A version of this article has also appeared in Computing Education, June 2012, China. This work is partially supported by US National Science Foundation under grants IIS 1143533, CCF 1135124, CCF 1048711 and CNS 0950432.

[2] See for curriculum working group members in Appendix A.

on how to teach each topic and guidance on where each topic could potentially be incorporated into a core CS/CE course. Recognizing the differing needs of institutions of various types (liberal arts colleges vs. technical colleges, regional universities vs. large research universities, etc.), sizes, laboratory facilities, local needs and constraints, cultural and geographical settings, etc., the group committed itself not to be prescriptive in its recommendations, but rather to provide alternatives with rationales for each. *A preliminary version of the core curriculum was released in Dec, 2010.*[3]

The Early Adopter Competitions

A major mechanism for evaluating snapshots of our guidelines at different times are the *Early Adopter competitions* that we have thus far run three times: in Spring 2011 using funds from the original grants from NSF and Intel; and in Fall 2011 and Spring 2012, funded by subsequent grants from the same sources and supplemented by GPU card donations from NVIDIA. Aspiring Early Adopters submit a proposal that is evaluated by a committee from the working group. Selected proposals are awarded a small stipend to use as seed money in implementing a PDC curriculum. We selected 16, 18, and 21 institutions, respectively, in the Spring 2011, Fall 2011, and Spring 2012 competitions[4] awarding an average of $1.5K/ institution. The faculty associated with the selected proposals are employing our initial curriculum guidelines in one or more courses at their respective institutions. Each institution is implementing a PDC curriculum in a way that is personalized to its culture and environment.

The EduPar Workshop and Current Activities

In order to allow the Early Adopters, the public, and the working group to benefit from everyone's experiences and evaluations, we organized the first EduPar Workshop, collocated with IPDPS in Anchorage, Alaska in May 2011, to bring together the Early Adopters and others interested in PDC education - primarily to receive the feedback from the Adopters, but also to stimulate discussion of curricular and other educational issues. The inaugural EduPar workshop - the first workshop devoted exclusively to educational matters at IPDPS - was a great success, with attendance in the range of 40-80 throughout the single day of the event.

EduPar'12 was held at Shanghai in May as a regular IPDPS'12 satellite workshop, with 5 regular and 8 short papers, a poster session, and a keynote session, with similar attendance. The accepted papers will appear in the proceedings of the IPDPS workshops and will be uploaded into IEEE XPlore. EduPar'13 will be held in Boston. [5]

We are currently revising the preliminary version of the curriculum. We have worked through Fall 2011 and Spring 2012 on a bi-weekly basis. Feedback and evaluations from Early Adopters are being collected and will be employed for finalizing the revisions in the curriculum and for releasing an initial formal version for the curriculum. We anticipate frequent updates and revisions as we learn more from the experiences of the Early Adopters.

The twin activities of Early Adopter competitions and EduPar workshops will enable the working group to periodically update the curriculum. This will form a series of annual activities for the next few years to solidify the effort and root it within the broader CS/CE community. We are beginning to put together a website for educational resources from academia and industry, and envision a book/tutorial series based on the curriculum.

Upcoming CEDR Center - A Roadmap

CEDR or Center for Parallel and Distributed Computing Curriculum Development and Educational Resources is being established with the help of a NSF grant at Georgia State University. The center aims to carry out four synergistic areas of activities.

- Develop PDC core curricula flexible enough for a broad range of programs and institutions; collaborate with all stakeholders (educators, students, researchers, authors, industry, governments, funding agencies, professional societies and task forces) to maintain currency and facilitate adoption.
- Develop, collect, and synthesize pedagogical and instructional materials for teaching PDC curriculum topics–including slides, modules, tutorials, lectures, books, testing and evaluation tools.
- Facilitate access to state-of-the-art hardware and software resources for PDC instruction and training by instructors and students worldwide in following areas: Hardware architectures - multicores, manycores, shared and distributed memories, high-end machines (in collaboration with NSF-funded national infrastructures, industry, and labs); Program development environments, compilers, debuggers, and performance monitoring and enhancement tools; Sample programs and "industrial-strength" PDC software.
- Organize and administer competitions for early adopters of PDC curricula (winners receiving stipends, equipment, etc.), organize workshops, special sessions, tutorials, and training sessions to foster awareness and adoption of PDC curricula.

Interface to the Broader Community

The CS2013 ACM/IEEE Computer Science Curriculum Joint Task Force has recognized PDC (along with security) as a main thrust area. We are closely interacting with the Task Force, providing expert feedback on the PDC portion of their

[3] See the curriculum in Appendix C.
[4] See the Early Adopter institutions in Appendix B.
[5] The proceedings of the EduPar'11 workshop is at http://www.cs.gsu.edu/~tcpp/curriculum/?q=node/16950; video coverage of all presentations is available, courtesy of Intel, at http://techtalks.tv/events/53/. Proceedings of EduPar-12 is posted at http://cs.gsu.edu/~tcpp/curriculum/?q=advanced-technical-program.

initial draft on PDC in Oct, 2011. We will continue to engage with this and other education-oriented task forces in the hope of having significant impact on the CS/CE academic community.

APPENDIX A: NSF/IEEE-TCPP Curriculum Working Group

1. Chtchelkanova, Almadena (NSF),
2. Das, Sajal (University of Texas at Arlington),
3. Das, Chita (Penn State),
4. Dehne, Frank (Carleton University, Canada),
5. Gouda, Mohamed (University of Texas, Austin, NSF),
6. Gupta, Anshul (IBM T.J. Watson Research Center),
7. Jaja, Joseph (University of Maryland),
8. Kant, Krishna (NSF, Intel),
9. La Salle, Anita (NSF),
10. LeBlanc, Richard (Seattle University),
11. Lumsdaine, Andrew (Indiana University),
12. Padua, David (University of Illinois at Urbana-Champaign),
13. Parashar, Manish (Rutgers),
14. Prasad, Sushil (Georgia State University),
15. Prasanna, Viktor (University of Southern California),
16. Robert, Yves (INRIA, France),
17. Rosenberg, Arnold (Northeastern University),
18. Sahni, Sartaj (University of Florida),
19. Shirazi, Behrooz (Washington State University),
20. Sussman, Alan (University of Maryland),
21. Weems, Chip (University of Massachusetts), and
22. Wu, Jie (Temple University)

APPENDIX B : List of Early Adopters

Spring 2012 Competition

1. North Carolina State University, USA
2. Moravian College, USA
3. Western Oregon University, USA
4. Ohio University, School of EECS, USA
5. University of Cincinnati, USA
6. Purdue University, USA
7. University of Illinois at Urbana Champagne, USA
8. University of Massachusetts Amherst, USA
9. Prairie View A&M University, USA
10. Radford University, USA
11. University of Utah, USA
12. University of Colorado Boulder, USA
13. University of Houston-Downtown, USA
14. Institut Teknologi Bandung, Indonesia
15. Universidade Federal de Campina Grande, Brazil
16. Middlexsex College University of Western Ontario, Canada
17. Universidad Nacional de Córdoba - FaMAF, Argentina
18. University of Victoria, Canada
19. Jadavpur University, Kolkata, India
20. Departamento de Electrónica, Universidad Tecnológica Nacional, Facutad Regional, Argentina
21. Central South University, Changsha, China

Fall 2012 Competition

1. St. Olaf College, USA
2. Kent State University, USA
3. Georgia State University, USA
4. NC A & T State University, USA
5. Ursinus College, USA
6. Southwest Baptist Unversity, USA
7. University of Central Arkansas, USA
8. Florida State University, USA
9. Texas A&M University - Corpus Christi, USA
10. Texas Tech University, USA
11. SPSU, USA
12. University of Puerto Rico, USA
13. University of Murcia, Spain
14. Universidad Nacional de La Plata, Argentina
15. Universidad Tecnológica Nacional, Facultad Regional Bahía Blanca, Argentina
16. Universidad Nacional de San Luis, Argentina
17. Universidad de Buenos Aires, Argentina
18. Universidad Tecnológica Nacional – Facultad Regional Mendoza, Argentina

Spring 2011 Competition

1. Columbia University, USA
2. Hampton University, USA
3. Georgia Institute of Technology, USA
4. Washington and Lee University, USA
5. University of Central Florida, USA
6. Loyola University Chicago, USA
7. Wittenberg University and Clemson University, USA
8. University of Georgia, USA
9. Calvin College, USA
10. Arizona State University, USA
11. Universidad Nacional de Río Cuarto, Argentina
12. University of Pannonia, Hungary
13. Kassel University, Germany
14. Knox College, USA
15. International Institute of Information Technology, Hyderabad, India
16. Universidad Nacional del Sur, Argentina

APPENDIX C: NSF/IEEE-TCPP Proposed Curriculum

Notation

Absolutely every individual CS/CE undergraduate must be at this level as a result of his or her required coursework

K = Know the term

C = Comprehend so as to paraphrase/illustrate

A = Apply it in some way

N = Not in Core, but can be in an elective course

Core Courses:

CS1 - Introduction to Computer Programming (First Course)
CS2 - Second Programming Course in the Introductory Sequence
Systems - Intro Systems/Architecture Core Course
DS/A - Data Structures and Algorithms
DM - Discrete Structures/Math

ADVANCED/ELECTIVE COURSES:

Arch 2 - Advanced Elective Course on Architecture
Algo 2 - Elective/Advanced Algorithm Design and Analysis
Lang - Programming Language/Principles (after introductory sequence)
SwEngg - Software Engineering
ParAlgo - Parallel Algorithms
ParProg - Parallel Programming
Compilers - Compiler Design
Networking - Communication Networks
Dist Systems - Distributed Systems

Note: The numbers of hours suggested in the following tables must be interpreted carefully. Within all tables *except for Algorithms*, the number suggested for a given topic represents a cumulative total across a number of higher-level topics. For example, the number of hours required for achieving the desired "A" level competence in shared memory programming is the total of all hours allocated for "shared memory" across all higher-level topics — in addition to the hours allocated to related topics such as "SPMD," "tasks and threads," and "synchronization." In contrast, the hours allocated to Algorithms topics represent our estimates of the effort required to achieve the desired level of competence solely within the context of Algorithms instruction. This decision reflects our recognition that many Algorithms topics develop concepts and tools that will pervade the coverage of many disparate non-Algorithms topics — the specific list of topics varying from institution to institution. The cumulative number of hours to master a topic is, therefore, impossible to estimate in isolation.

Architecture Topics

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
Classes				
Taxonomy	C	0.5	Systems	Flynn's taxonomy, data vs. control parallelism, shared/distributed memory
Data vs. control parallelism				
Superscalar (ILP)	K	0.25 to 1, based on level	Systems	Describe opportunities for multiple instruction issue and execution (different instructions on different data)
SIMD/Vector (e.g., SSE, Cray)	K	0.1 to 0.5	Systems	Describe uses of SIMD/Vector (same operation on multiple data items), e.g., accelerating graphics for games.
Pipelines				
▪ Single vs. multicycle	K	1 to 2	Systems	Describe basic pipelining process (multiple instructions can execute at the same time), describe stages of instruction execution
▪ Data and control hazards	N		Compilers (A), Arch 2 (C)	Understand how one pipe stage can depend on a result from another, or delayed branch resolution can start the wrong instructions in a pipe, requiring forwarding, stalling, or restarting
▪ OoO execution	N		Arch 2 (K)	Understand how independent instructions can be rescheduled for better pipeline utilization, and that various tables are needed to ensure RAW, WAR, and WAW hazards are avoided.
Streams (e.g., GPU)	K	0.1 to 0.5	Systems	Know that stream-based architecture exists in GPUs for graphics
Dataflow	N		Arch 2 (K)	Be aware of this alternative execution paradigm

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
MIMD	K	0.1 to 0.5	Systems	Identify MIMD instances in practice (multicore, cluster, e.g.), and know the difference between execution of tasks and threads
Simultaneous Multi-Threading	K	0.2 to 0.5	Systems	Distinguish SMT from multicore (based on which resources are shared)
Highly Multithreaded (e.g., MTA)	N		Arch 2 (K)	Have an awareness of the potential and limitations of thread level parallelism in different kinds of applications
Multicore	C	0.5 to 1	Systems	Describe how cores share resources (cache, memory) and resolve conflicts
Heterogeneous (e.g., Cell, on-chip GPU)	K	0.1 to 0.5	Systems	Recognize that multicore may not all be the same kind of core.
Shared vs. distributed memory				
SMP	N		Arch 2 (C)	Understand concept of uniform access shared memory architecture
▪ Buses	C	0.5 to 1	Systems	Single resource, limited bandwidth and latency, snooping, scalability issues
NUMA(Shared Memory)	N			
▪ CC-NUMA	N		Arch 2 (K)	Be aware that caches in the context of shared memory depend on coherence protocols
▪ Directory-based CC-NUMA	N		Arch 2 (K)	Be aware that bus-based sharing doesn't scale, and directories offer an alternative
Message passing (no shared memory)	N		Arch 2 (K)	Shared memory architecture breaks down when scaled due to physical limitations (latency, bandwidth) and results in message passing architectures
▪ Topologies	N		Algo 2 (C)	Various graph topologies - linear, ring, mesh/torus, tree, hypercube, clique, crossbar
▪ Diameter	N		Algo 2 (C)	Appreciate differences in diameters of various graph topologies
▪ Latency	K	0.2 to 0.5	Systems	Know the concept, implications for scaling, impact on work/communication ratio to achieve speedup
▪ Bandwidth	K	0.1 to 0.5	Systems	Know the concept, how it limits sharing, and considerations of data movement cost
▪ Circuit switching	N		Arch 2 (C) or Networking (A)	Know that interprocessor communication can be managed using switches in networks of wires to establish different point-to-point connections, that the topology of the network affects efficiency, and that some connections may block others
▪ Packet switching	N		Arch 2 (C) or Networking (A)	Know that interprocessor communications can be broken into packets that are redirected at switch nodes in a network, based on header info
▪ Routing	N		Arch 2 (C) or Networking (A)	Know that messages in a network must follow an algorithm that ensures progress toward their destinations, and be familiar with common techniques such as store-and-forward, or wormhole routing
Memory Hierarchy				
▪ Cache organization	C	0.2 to 1	Systems	Know the cache hierarchies, shared caches (as opposed to private caches) result in coherency and performance issues for software

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
▪ Atomicity	N		Arch 2 (K)	Need for indivisible operations can be covered in programming, OS, or database context
▪ Consistency	N		Arch 2 (K)	Models for consistent views of data in sharing can be covered in programming, OS, or database context
▪ Coherence	N		Arch 2 (C)	Describe how cores share cache and resolve conflicts - may be covered in programming. OS, or database context
▪ False sharing	N		Arch2 (K)/ ParProg (K)	Awareness, examples of how it originates
▪ Impact on software	N		Arch2 (C)/ ParProg (A)	Issues of cache line length, memory blocks, patterns of array access, compiler optimization levels
Floating point representation				These topics are supposed to be in the ACM/IEEE core curriculum already – they are included here to emphasize their importance, especially in the context of PDC.
Range	K		CS1/CS2/ Systems	Understand that range is limited, implications of infinities
Precision	K	0.1 to 0.5	CS1/CS2/ Systems	How single and double precision floating point numbers impact software performance
Rounding issues	N		Arch 2 (K)/ Algo 2 (A)	Understand rounding modes, accumulation of error and loss of precision
Error propagation	K	0.1 to 0.5	CS2	Understand NaN, Infinity values and how they affect computations and exception handling
IEEE 754 standard	K	0.5 to 1	CS1/CS2/ Systems	Representation, range, precision, rounding, NaN, infinities, sub-normals, comparison, effects of casting to other types
Performance metrics				
Cycles per instruction (CPI)	C	0.25 to 1	Systems	Number of clock cycles for instructions, understand the performance of processor implementation, various pipelined implementations
Benchmarks	K	0.25 to 0.5	Systems	Awareness of various benchmarks and how they test different aspects of performance
▪ Spec mark	K	0.25 to 0.5	Systems	Awareness of pitfalls in relying on averages (different averages can alter perception of which architecture is faster)
▪ Bandwidth benchmarks	N		Arch 2 (K)	Be aware that there are benchmarks focusing on data movement instead of computation
Peak performance	C	0.1 to 0.5	Systems	Understanding peak performance, how it is rarely valid for estimating real performance, illustrate fallacies
▪ MIPS/FLOPS	K	0.1	Systems	Understand meaning of terms
Sustained performance	C	0.1 to 0.5	Systems	Know difference between peak and sustained performance, how to define, measure, different benchmarks

Programming Topics

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
Parallel Programming paradigms				
By the target machine model		5		
SIMD	K	0.5	CS2; Systems	Understand common vector operations including element-by-element operations and reductions.
▪ Processor vector extensions	K		Systems	Know examples - SSE/Altivec macros
▪ Array language extensions	N		ParProg (A)	Know how to write parallel array code in some language (e.g., Fortran95, Intel's C/C++ Array Extension[CEAN])
Shared memory	A	2.0	CS2; DS/A; Lang	Be able to write correct thread- based programs (protecting shared data) and understand how to obtain speed up.
▪ Language extensions	K			Know about language extensions for parallel programming. Illustration from Cilk (spawn/join) and Java (Java threads)
▪ Compiler directives/ pragmas	C			Understand what simple directives, such as those of OpenMP, mean (parallel for, concurrent section), show examples
▪ Libraries	C			Know one in detail, and know of the existence of some other example libraries such as Pthreads, Pfunc, Intel's TBB (Thread building blocks), Microsoft's TPL (Task Parallel Library), etc.
Distributed memory	C	1.0	DS/A; Systems	Know basic notions of messaging among processes, different ways of message passing, collective operations
▪ Message passing	N		ParProg(C)	Know about the overall organization of an message passing program as well as point-to-point and collective communication primitives (e.g., MPI)
▪ PGAS languages	N		ParProg (C)	Know about partitioned address spaces, other parallel constructs (UPC, CoArray Fortran, X10, Chapel)
Client Server	C	1.0	DS/A; Systems	Know notions of invoking and providing services (e.g., RPC, RMI, web services) - understand these as concurrent processes
Hybrid	K	0.5	Systems	Know the notion of programming over multiple classes of machines simultaneously (CPU, GPU, etc.)
By the control statement				
Task/thread spawning	A	1	CS2; DS/A	Be able to write correct programs with threads, synchronize (fork-join, producer/consumer, etc.), use dynamic threads (in number and possibly recursively) thread creation - (e.g. Pthreads, CILK, Java threads, etc.) - builds on shared memory topic above
SPMD	C	1.0	CS2; DS/A	Understand how SPMD program is written and how it executes
▪ SPMD notations	C			Know the existence of highly threaded data parallel notations (e.g., CUDA, OpenCL), message passing (e.g, MPI), and some others (e.g., Global Arrays, BSP library)
Data parallel	A	1	CS2; DS/A; Lang	Be able to write a correct data-parallel program for shared-memory machines and get speedup, should do an exercise. Understand relation between different notations for data parallel: Array notations, SPMD, and parallel loops. Builds on shared memory topic above.

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
▪ Parallel loops for shared memory	A		CS2; DS/A; Lang	Know, through an example, one way to implement parallel loops, understand collision/dependencies across iterations (e.g., OpenMP, Intel's TBB)
▪ Data parallel for distributed memory	N		ParProg (K)	Know data parallel notations for distributed memory (e.g., High Performance Fortran)
Functional/logic languages	N		ParProg (K)	Understanding advantages and disadvantages of very different programming styles (e.g., Parallel Haskell, Parlog, Erlang)
Semantics and correctness issues				
Tasks and threads	K	0.5	CS2; DS/A; Systems, Lang	Understand what it means to create and assign work to threads/ processes in a parallel program, and know of at least one way do that (e.g., OpenMP, Intel TBB, etc.)
Synchronization	A	1.5	CS2; DS/A; Systems	Be able to write shared memory programs with critical regions, producer- consumer communication, and get speedup; know the notions of mechanisms for concurrency (monitors, semaphores, etc. - [from ACM 2008])
▪ Critical regions	A			Be able to write shared memory programs that use critical regions for synchronization
▪ Producer-consumer	A			Be able to write shared memory programs that use the producer-consumer pattern to share data and synchronize threads
▪ Monitors	K			Understand how to use monitors for synchronization
Concurrency defects	C	1.0	DS/A; Systems	Understand the notions of deadlock (detection, prevention), race conditions (definition), determinacy/non-determinacy in parallel programs (e.g., if there is a data race, the output may depend on the order of execution)
▪ Deadlocks	C			Understand what a deadlock is, and methods for detecting and preventing them
▪ Data Races	K			Know what a data race is, and how to use synchronization to prevent it
Memory models	N		ParProg (C)	Know what a memory model is, and the implications of the difference between strict and relaxed models (performance vs. ease of use)
▪ Sequential consistency	N			Understand semantics of sequential consistency for shared memory programs
▪ Relaxed consistency	N			Understand semantics of one relaxed consistency model (e.g., release consistency) for shared memory programs
Tools to detect concurrency defects	K	0.5	DS/A; Systems	Know the existence of tools to detect race conditions (e.g., Eraser)
Performance issues				
Computation	C	1.5	CS2; DS/A	Understand the basic notions of static and dynamic scheduling, mapping and impact of load balancing on performance
Computation decomposition strategies	C			Understand different ways to assign computations to threads or processes
▪ Owner computes rule	C			Understand how to assign loop iterations to threads based on which thread/process owns the data element(s) written in an iteration

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
▪ Decomposition into atomic tasks	C			Understand how to decompose computations into tasks with communication only at the beginning and end of each task, and assign them to threads/processes
▪ Work stealing	N		ParProg (C)	Understand one way to do dynamic assignment of computations
Program transformations	N		Compilers (A)	Be able to perform simple loop transformations by hand, and understand how that impacts performance of the resulting code (e.g., loop fusion, fission, skewing)
Load balancing	C	1.0	DS/A; Systems	Understand the effects of load imbalances on performance, and ways to balance load across threads or processes
Scheduling and mapping	C	1.0	DS/A; Systems	Understand how a programmer or compiler maps and schedules computations to threads/processes, both statically and dynamically
▪ Static				Understand how to map and schedule computations before runtime
▪ Dynamic				Understand how to map and schedule computations at runtime
Data	K	1.0	DS/A; Lang	Understand impact of data distribution, layout and locality on performance; know false sharing and its impact on performance (e.g., in a cyclic mapping in a parallel loop); notion that transfer of data has fixed cost plus bit rate (irrespective of transfer from memory or inter-processor)
Data distribution	K			Know what block, cyclic, and block-cyclic data distributions are, and what it means to distribute data across multiple threads/ processes
Data layout	K			Know how to lay out data in memory to get improve performance (memory hierarchy)
Data locality	K			Know what spatial and temporal locality are, and how to organize data to take advantage of them
False sharing	K			Know that for cache coherent shared memory systems, data is kept coherent in blocks, not individual words, and how to avoid false sharing across threads of data for a block
Performance monitoring tools	K	0.5	DS/A; Systems	Know of tools for runtime monitoring (e.g., gprof, Vtune)
Performance metrics	C	1.0	CS2; DS/A	Know the basic definitions of performance metrics (speedup, efficiency, work, cost), Amdahl's law; know the notions of scalability
Speedup	C			Understand how to compute speedup, and what it means
Efficiency	C			Understand how to compute efficiency, and why it matters
Amdahl's law	K			Know that speedup is limited by the sequential portion of a parallel program, if problem size is kept fixed
Gustafson's Law	K			Understand the idea of weak scaling, where problem size increases as the number of processes/threads increases
Isoefficiency	N		ParProg; Algo2 (C)	Understand the idea of how quickly to increase problem size with number of processes/threads to keep efficiency the same

Algorithm Topics

Note: Recall that the numbers of hours in this table reflect just the coverage within the Algorithms portion of the curriculum. (See the explanatory note earlier)

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
Parallel and Distributed Models and Complexity		7.41		Be exposed to the models and to the intrinsic degree of parallelism of some elementary key algorithms (e.g., maximum-finding, summation)
Costs of computation:		*1.66*		Follow arguments for parallel time and space complexity given by instructor
Asymptotics	C	1	DS/A	Understand upper (big-O) and lower bounds (big-Omega,); follow elementary big-O analyses, e.g., the O(log n) tree-depth argument for mergesort with unbounded parallelism.
Time	C	0.33	DS/A	Recognize time as a fundamental computational resource that can be influenced by parallelism
Space/Memory	C	0.33	DS/A	Recognize space/memory in the same manner as time
Cost reduction:		*1*		Be exposed to a variety of computational costs other than time that can benefit from parallelism (a more advanced extension of "speedup")
Speedup	C	1	DS/A	Recognize the use of parallelism either to solve a given problem instance faster or to solve larger instance in the same time (strong and weak scaling)
Space compression	N	0.33		Be exposed to ways in which the computational resource "space" behaves the same as "time" and to ways in which the two cost measures differ
Cost tradeoffs:		*0.75*		Recognize the inter-influence of various cost measures
Time vs. space	N	0.5	DS/A	Observe several examples of this prime cost tradeoff; lazy vs. eager evaluation supplies many examples
Power vs. time	N	0.25	DS/A	Observe at least one example of this prime cost tradeoff (the literature on "VLSI computation" — e.g., the footnoted books[6] [7] — yield many examples)
Scalability in algorithms and architectures	*C/ K*	*0.5*	DS/A	Comprehend via several examples that having access more processors does not guarantee faster execution --- the notion of inherent sequentiality (e.g., the seminal paper by Brent)
Model-based notions:		*4*		Recognize that architectural features can influence amenability to parallel cost reduction and the amount of reduction achievable
Notions from complexity-theory:		2		Understand (via examples) that some computational notions transcend the details of any specific model
▪ PRAM	K	1	DS/A	Recognize the PRAM as embodying the simplest forms of parallel computation: Embarrassingly parallel problems can be sped up easily just by employing many processors.

[6] F. T. Leighton (1992): Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann, San Mateo, Cal.
[7] J. D. Ullman (1984): Computational Aspects of VLSI. Computer Science Press, Rockville, Md.

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
▪ BSP/CILK	K	1	DS/A	Be exposed to higher-level algorithmic abstractions that encapsulate more aspects of real architectures. Either BSP or CILK would be a good option to introduce a higher level programming model and higher-level notions. Remark that both of these abstractions have led to programming models.
▪ Simulation/emulation	N	1	Algo 2	See simple examples of this abstract, formal analogue of the *virtual machines* that are discussed under programming topics. It is important to stress that (different aspects of the same) central notions of PDC can be observed in all four of our main topic areas.
▪ P-completeness and #P-completeness	N	1	Algo 2	Recognize these two notions as the parallel analogues of NP-completeness. They are the quintessential model-independent complexity-theoretic notions.
▪ Cellular automata	N	1	Algo 2	Be exposed to this important model that introduces new aspects of parallelism/distributed computing --- possibly via games (such as Life)
Notions from scheduling:		2		Understand how to decompose a problem into tasks
▪ Dependencies	A	0.5	CS1/CS2, DS/A	Observe how dependencies constrain the execution order of sub-computations --- thereby lifting one from the limited domain of "embarrassing parallelism" to more complex computational structures.
▪ Task graphs	C	0.5	DS/A; SwEngg	See multiple examples of this concrete algorithmic abstraction as a mechanism for exposing inter-task dependencies. These graphs, which are used also in compiler analyses, form the level at which parallelism is exposed and exploited.
▪ Work	K	0.5	DS/A	Observe the impact of computational work (e.g., the total number of tasks executed) on complexity measures such as power consumption.
▪ (Make)span	K	0.5	DS/A	Observe analyses in which makespan is identified with parallel time (basically, time to completion)
Algorithmic Paradigms		4.5		
Divide & conquer (parallel aspects)	C	*1*	CS2, DS/A, Algo 2	Observe, via tree-structured examples such as mergesort or numerical integration (trapezoid rule, Simpson's rule) or (at a more advanced level) Strassen's matrix-multiply, how the same structure that enables divide and conquer (sequential) algorithms exposes opportunities for parallel computation.
Recursion (parallel aspects)	C	*0.5*	CS2, DS/A	Recognize algorithms that, via unfolding, yield tree structures whose subtrees can be computed independently, in parallel
Scan (parallel-prefix)	N	0.5	ParAlgo, Architecture	Observe, via several examples[8,9] this "high-level" algorithmic tool

[8] G. E. Blelloch (1989): Scans as primitive parallel operations. IEEE Transactions on Computers 38, pp. 1526–1538

[9] F.T. Leighton (1992): Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann, San Mateo, Cal.

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
Reduction (map-reduce)	K/C	*1*	DS/A	Recognize, and use, the tree structure implicit in scalar product or mergesort or histogram (equivalent apps)
Stencil-based iteration	N	0.5	ParAlgo	Observe illustrations of mapping and load balancing via stenciling
Dependencies:	K	*0.5*	Systems	Understand the impacts of dependencies
"Oblivious" algorithms	N	0.5	ParAlgo	Observe examples of these "model-independent" algorithms that ignore the details of the platform on which they are executed. Recognize obliviousness as an important avenue toward *portability*.
Blocking	N	0.5	ParAlgo	See examples of this algorithmic manifestation of memory hierarchies
Striping	N	0.5	ParAlgo	See examples of this algorithmic manifestation of memory hierarchies
"Out-of-core" algorithms	N	0.5	ParAlgo	Observe ways of accommodating a memory/storage hierarchy by dealing with issues such as *locality* and acknowledging the changes in cost measures at the various levels of the hierarchy.
Series-parallel composition	C	*1*	CS2(K), Systems(C)	Understand how "barrier synchronizations" can be used to enable a simple thread-based abstraction for parallel programming. Understand the possible penalties (in parallelism) that this transformation incurs
Graph embedding as an algorithmic tool	N	1	ParAlgo	Recognize this key algorithmic tool for crafting simulations/ emulations.
Algorithmic problems		8.5		The important thing here is to emphasize the parallel/ distributed aspects of the topic
Communication	*C/ A*	*2*		Understand — via hands-on experience — that inter-processor communication is one of the most challenging aspects of PDC.
Broadcast	C/ A	1	DS/A	Use this important mode of global communication; observe enabling algorithms for various platforms (e.g., recursive doubling)
Multicast	K/ C	0.5	DS/A	Recognize other modalities of global communication on a variety of platforms: e.g., rings, 2D-meshes, hypercubes, trees
Scatter/gather	C/ A	0.5	DS/A	Recognize these informational analogues of Map and reduce
Gossip	N	0.5	Dist Systems, Networking	Recognize how all-to-all communication simplifies certain computations
Asynchrony	K	*0.5*	CS2	Understand asynchrony as exhibited on a distributed platform, its strengths (no need for synchs) and pitfalls (the danger of race conditions)
Synchronization	K	*1*	CS2, DS/A	Be aware of methods for controlling race conditions
Sorting	C	*1.5*	CS2, DS/A	Observe several sorting algorithms for varied platforms — together with analyses. Parallel merge sort is the simplest example, but equally simple alternatives for rings and meshes might be covered also; more sophisticated algorithms might be covered in more advanced courses

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
Selection	K	*0.5*	CS2, DS/A	Observe algorithms for finding order statistics, notably min and max. Understand that selection can always be accomplished by sorting but that direct algorithms may be simpler.
Graph algorithms:		1		
Search	C	1	DS/A	Know how to carry out BFS- and DFS-like parallel search in a graph or solution space
Path selection	N	1		
Specialized computations	A	2	CS2, DS/A	Master one or two from among computations such as: matrix product, transposition, convolution, and linear systems; *recognize how algorithm design reflects the structure of the computational problems.*
Convolutions	Optional	1		Be exposed to block or cyclic mappings; understand trade-offs with communication costs
Matrix computations	Optional	1		Understand the mapping and load balancing problems on various platforms for significant concrete instances of computational challenges that are discussed at a higher level elsewhere
▪ Matrix product	Optional	1		Observe a sample "real" parallel algorithm, such as Cannon's algorithm[10]
▪ Linear systems	Optional	1		Observe load-balancing problems in a concrete setting
▪ Matrix arithmetic	Optional	1		Observe the challenges in implementing even "simple" arithmetic
▪ Matrix transpose	Optional	1		Observe a challenging concrete data permutation problem
Termination detection	N/K	1	ParAlgo	See examples that suggest the difficulty of proving that algorithms from various classes actually terminate. For more advanced courses, observe proofs of termination, to understand the conceptual tools needed.
Leader election/symmetry breaking	N/K	2	ParAlgo	Observe simple symmetry-breaking algorithms, say for a PRAM

Cross Cutting and Advanced Topics

Topics	BLOOM #	HOURS	Where Covered	Learning Outcome
High level themes:				
Why and what is parallel/ distributed computing?	K	0.5	CS1, CS2	Know the common issues and differences between parallel and distributed computing; history and applications. Microscopic level to macroscopic level parallelism in current architectures.
Cross-Cutting topics				know these underlying themes

[10] H. Gupta, P. Sadayappan (1996): Communication Efficient Matrix-Multiplication on Hypercubes. Parallel Computing 22 , pp. 75-99.

Topics	BLOOM#	HOURS	Where Covered	Learning Outcome
Locality	C	1	DS/A, Systems	Understand this as a dominant factor impacting performance - minimizing cache/memory access latency or inter-processor communication.
Concurrency	K	0.5	CS2, DS/A	The degree of inherent parallelism in an algorithm, independent of how it is executed on a machine
Non-determinism	K	0.5	DS/A, Systems	Different execution sequences can lead to different results hence algorithm design either be tolerant to such phenomena or be able to take advantage of this.
Power Consumption	K	0.5	Systems, DS/A	Know that power consumption is a metric of growing importance, its impact on architectural evolution, and design of algorithms and software.
Fault tolerance	K	0.5	Systems	Large-scale parallel/distributed hardware/software systems are prone to components failing but system as a whole needs to work.
Performance modeling	N	0.5	Arch 2, Networking, Adv OS	Be able to describe basic performance measures and relationships between them for both individual resources and systems of resources.
Current/Advanced Topics				
Cluster Computing	K	0.25	CS2, DS/A, System	Be able to describe a cluster as a popular local-memory architecture with commodity compute nodes and a high-performance interconnection network.
Cloud/grid Computing	K	0.25	CS2, DS/A, System	Recognize cloud and grid as shared distributed resources - cloud is distinguished by on-demand, virtualized, service-oriented software and hardware resources.
Peer to Peer Computing	K	0.25	CS1, CS2	Be able to describe a peer to peer system and the roles of server and client nodes with distributed data. Recognize existing peer to peer systems.
Consistency in Distributed Transactions	K	0.25	CS1,CS2, Systems	Recognize classic consistency problems. Know that consistency maintenance is a primary issue in transactions issued concurrently by multiple agents.
Web search	K	0.25	CS1, CS2	Recognize popular search engines as large distributed processing systems for information gathering that employ distributed hardware to support efficient response to user searches.
Security in Distributed Systems	K	0.5	Systems	Know that distributed systems are more vulnerable to privacy and security threats; distributed attacks modes; inherent tension between privacy and security.
Social Networking/Context	N	0.5	AI, Distributed Systems, Networking,	Know that the rise of social networking provides new opportunities for enriching distributed computing with human & social context.
Collaborative Computing	N	0.25	HCI, Dist Systems, OS	Know that collaboration between multiple users or devices is a form of distributed computing with application specific requirements.
Performance modeling	N	0.5	Arch 2, Networking	Be able to describe basic performance measures and relationships between them for both individual resources and systems of resources.

Web services	N	0.5	Web Programming, Dist Systems, Adv OS,	Know that web service technology forms the basis of all online user interactions via browser.
Pervasive and Mobile computing	N	0.5	Mobile Computing, Networking, Dist System	Know that the emerging pervasive and mobile computing is another form of distributed computing where context plays a central role.

About the Authors

Prasad, Sushil K.
Professor of Computer Science at Georgia State University
Director of DiMoS Lab
Former Chair, IEEE Computer Society Technical Committee on Parallel Processing (TCPP)
Homepage: http://www.cs.gsu.edu/prasad/

Gupta, Anshul
IBM Research
Business Analytics & Mathematical Sciences
Homepage: http://researcher.ibm.com/view.php?person=us-anshul

Kant, Krishna
Program director in the Computer and Networks Systems (CNS) division cluster of CISE/CNS division within National Science Foundation
Center of Secure Information Systems (CSIS) at George Mason University
Homepage: http://www.kkant.net/

Lumsdaine, Andrew
Director, Computer Science Program
Professor of Computer Science
Director, Open Systems Laboratory
Associate Director, Digital Science Center
Computer Science Department, Indiana University
Homepage: http://osl.iu.edu/~lums/

Padua, David
Professor
Siebel Center for Computer Science
University of Illinois at Urbana-Champaign
Homepage: http://polaris.cs.uiuc.edu/~padua/.

Robert, Yves
Professor at ENS Lyon & INRIA
Fellow of the IEEE Senior Member, Institut Universitaire de France, Visiting scientist, University of Tennessee Knoxville
Homepage: http://graal.ens-lyon.fr/~yrobert/

Rosenberg, Arnold
Research Professor of Computer Science at Northeastern University
Distinguished University Professor Emeritus of Computer Science at University of Massachusetts Amherst
Homepage: http://people.cs.umass.edu/~rsnbrg/

Sussman, Alan
Professor, Department of Computer Science & UMIACS
University of Maryland
Homepage: http://www.cs.umd.edu/~als/

Weems, Charles
Associate Professor,
Dept. of Computer Science
University of Massachusetts
Co-director of Architecture and Language Implementation research group
Homepage:http://people.cs.umass.edu/~weems/homepage/Main_Page.html