CSE 470/570: Introduction to Parallel and Distributed Processing

Course Information

Date/Time:	MoWeFr 4pm-4:50pm
Location:	Cooke 127A
Credits:	4 CSE 470, 3 CSE 570
Instructor:	Dr. Jaroslaw Zola
Office:	Davis Hall 335
Email:	jzola@buffalo.edu
Phone:	(716) 645-3187

CSE Focus Area: Software and Information Systems (SW)

Course Description

This course is intended for students interested in the efficient use of modern parallel systems ranging from multi-core and many-core processors to large-scale distributed memory clusters. The course puts equal emphasis on the theoretical foundations of parallel computing and practical aspects of different parallel programming models. It begins with a survey of common parallel architectures and types of parallelism, and then follows with an overview of formal approaches to assess scalability and efficiency of parallel algorithms and their implementations. In the second part, the course covers the most common and current parallel programming techniques and APIs, including for shared address space, many-core accelerators, distributed memory clusters and big data analytics platforms. Each component of the course involves solving practical computational and data driven problems, ranging from basic algorithms like sorting or searching, to graphs and numerical data analysis.

Course Organization

The course consists of a series of lectures organized into five topical modules. Each lecture module is complemented with a programming assignment exposing practical aspects of the covered material. The course outline is provided below:

1. Overview of parallel processing landscape: why and how, types of parallelism, Flynn's taxonomy and brief overview of parallel architectures, Exascale computing vs. Exascale data, practical demonstration of CCR as an example HPC center. (3 lectures) Basic concepts in parallel processing: formal definition of parallelism, concepts of work, speedup, efficiency, overhead, strong and weak scalability (Amdahl's law, Gustafson's law), practical considerations using parallel sum and parallel prefix. (4 lectures)

2. Multi-core programming: shared memory and shared address space, data and task parallelism, Cilk+, OpenMP, Intel TBB data structures (time permitting), parallel merge sort, pointer jumping, parallel BFS. (9 lectures)
3. Distributed memory programming: Message Passing Interface (including one-sided communication, derived datatypes and MPI-IO), interconnect topologies, latency + bandwidth model, parallel matrix-vector product, parallel connected components, sample sort. (9 lectures)
4. Higher-level programming models: MapReduce, Apache Spark and Resilient Distributed Datasets, Bulk Synchronous Parallel model, Pregel and Apache GraphX, triangle counting, connected components, single source shortest path. (9 lectures)
5. Many-core programming: SIMD parallelism, massively parallel GPGPU accelerators, data movement and organization, matrix-matrix product, connected components. (6 lectures)

Tentative lecture content and schedule are provided below:

Lecture	**Topic**
1	Syllabus overview: course organization and expectations.
2-3	Parallel computing landscape: motivation, evolution of computing, Flynn's taxonomy and parallel architectures. Exascale computing and exascale data, organization of HPC centers and data centers.
4	Practical demonstration of HPC center: interaction with CCR resources.
5-6	Definition of parallelism, relation to concurrency, basic example using parallel sum, definitions of work, overhead, speedup, efficiency, Work/Span model. Speed vs. efficiency, example efficiency analysis, strong and weak scalability (Amdahl's law and Gustafson's law).
7-8	Parallel prefix (recursive and butterfly algorithm), broadcast and reduction primitives, solving problems via parallel prefix (ranking processors, linear recurrences).
9	Midterm exam.
10	Introduction to multi-core programming: shared memory/shared address space concept. Task vs. data parallelism, basic parallel patterns: fork-join and stencil.
11-12	From sequential to parallel: Cilk+ API, parallel recursion and semi-recursion. Parallel mergesort and parallel tree traversal. Measuring performance of a parallel code on the actual parallel system.
13	Cilk+ for data parallelism, array notation, naive daxpy. Matrix multiplication using Strassen algorithm.
14	Pitfalls in multi-core programming: races, synchronization and Cilk+ reducers, tree traversal revisited.
15-16	OpenMP to gain finer control over execution. Basic pragmas and data sharing constructs. Implementing parallel prefix and pointer jumping. Finding tree root and connected components via pointer jumping.
17-18	Managing memory in a multi-core environment. Intel TBB data structures and containers. Parallel BFS using Cilk+ reducers and Intel TBB queues.
19	Introduction to distributed memory programming: concept of distributed memory, processor interconnect topology, basics of MPI.
20-21	Point to point communication, latency + bandwidth model, basic collective operations and their cost on different interconnects (mesh, hypercube, tree).

22-23	1D/2D data decomposition: matrix-vector product and connected components, analysis of scalability.
24	Data movement in MPI, parallel sample sort.
25-26	Custom data types, custom communicators, parallel I/O. One-sided communication, simple DHT implementation.
27	Measuring performance of MPI codes on the actual parallel system.
28	Introduction to higher-level programming models: need for fault-tolerance, elasticity, high level APIs and integrated platforms. MapReduce model overview, simple algorithm to build node degree histogram.
29-30	Iterative MapReduce in Spark: Spark stack overview, Spark context and execution environment. Concept of Resilient Distributed Dataset: creation, lifetime, lineage, evaluation and persistence.
31-33	Common Spark transformations and actions, Spark execution plans and scheduling. Counting triangles in large graphs, finding connected components.
34-35	BSP model basics: superstep, computation, communication, synchronization. Practical realizations in Pregel and GraphX. Connected components and pointer jumping revisited.
36	Single Source Shortest Path problem and triangle counting in GraphX.
37	Introduction to massively parallel accelerators: host and device, GPGPU hardware organization.
38-39	CUDA API, computational kernels, threads organization, memory management, data transfers, stencil compute pattern on simple 1D smoothing kernel. Comparison with Thrust C++ API.
40-41	Optimizing memory access and threads execution, memory access coalescing. Examples including, prefix computations and connected components.
42	Q&A in preparation for the final exam.

Course Prerequisites

The course has no specific prerequisites for graduate students. For undergraduate students, CSE 331 “Introduction to Algorithms” is the prerequisite, as the course requires some experience in synthesis and analysis of algorithms. The course has significant programming component, hence a rudimentary ability to learn new programming constructs is expected. Specifically, C++ (multi-core, many-core, MPI), Python (MapReduce, Spark) and Scala (GraphX) will be used extensively during the course. However, no significant expertise with these languages is needed!

Program Outcomes

Upon completion of this course you will:

- Gain basic understanding of fundamental concepts in parallel computing.
- Be able to identify and leverage common parallel computing patterns.
- Be able to properly assess efficiency and scalability of a parallel algorithm/application.
- Become proficient in using at least one parallel programming technique, and familiar with several others.

Course Requirements

The course has three requirements:

1. Midterm exam testing your understanding of a parallel computing landscape and basic concepts in parallel processing. Midterm will be held after the material of the first eight lectures is covered.
2. Programming assignments exposing you to the practical aspects of the covered material. There will be four assignments in total, one at the end of each course module (one after multi-core programming, one after distributed memory and so on). In each assignment, you will be asked to implement and benchmark a specific parallel algorithm relevant for real-life applications (using resources provided by CCR).
3. Final exam testing your overall understanding of the material.

Grading Policy

The final grade will be weighted average: 20% midterm exam, 30% final exam, 50% programming assignments. Exam and programming assignments will be the same for graduate and undergraduate students. However, criteria to decide the final grade will be different for graduate and undergraduate students. Specifically, the number-to-letter grade mapping will be done as indicated in the tables below.

For graduate students:

Percentage score	Grade	Quality points
90-100	A	4.0
85-89	A-	3.67
80-84	B+	3.33
75-79	B	3.0
70-74	B-	2.67
65-69	C+	2.33
60-64	C	2.0
55-59	C-	1.67
50-54	D	1.0
0-49	F	0.0

For undergraduate students:

Percentage score	Grade	Quality points
88-100	A	4.0
82-87	A-	3.67
77-81	B+	3.33
72-76	B	3.0
67-71	B-	2.67
62-66	C+	2.33
56-61	C	2.0
53-55	C-	1.67
50-52	D	1.0
0-49	F	0.0

In general, no incomplete grades ("IU" or "I") will be given. However, in special circumstances that are truly beyond your control and justify incomplete grade, we will follow the university policy on incomplete grades, available at: `http://grad.buffalo.edu/study/progress/policylibrary.html` (for graduate students) and `http://undergrad-catalog.buffalo.edu/policies/grading/explanation.html` (for undergraduate students).

Extra Points Opportunities

Throughout the course you will have opportunities to earn extra points that will be added towards your final exam score. Specifically:

1. Bug bounty program: if you spot a mistake (bug) in the slides/notes provided by the instructor before the instructor you will get 1 point for each bug. Examples include typos, missing elements in code snippets, copy-and-paste artifacts, etc. (sophisticated/questionable grammar mistakes or missing articles do not count). For each bug only the person who first spotted it will be awarded. The bug bounty program applies to this syllabus as well!
2. Best question asked: every lecture will be starting with a brief Q&A session where you will have a chance to ask questions regarding previous lectures. If your question is particularly insightful you will get 1 point.

Course Materials

This course does not rely on one specific textbook. The following books are suggested but not required:

- M. McCool, J. Reinders, A. Robison, "Structured Parallel Programming: Patterns for Efficient Computation," Morgan Kaufmann, 2012, ISBN-13: 9780124159938.
- A. Grama, G. Karypis, V. Kumar, A. Gupta, "Introduction to Parallel Computing (2nd Ed.)," Pearson, 2003, ISBN-13: 9780201648652.
- H. Karau, A. Konwinski, P. Wendell, M. Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis," O'Reilly Media, 2015, ISBN-13: 9781449358624.
- D.B. Kirk, W.W. Hwu, "Programming Massively Parallel Processors: A Hands-on Approach," Morgan Kaufmann, 2010, ISBN-13: 9780123814722.

Before ordering any of these books, please contact your instructor regarding books availability! Additionally, the following papers will be referenced during the course:

- F. McSherry, M. Isard, and D.G. Murray, "Scalability! but at what cost?," in Proc. USENIX Conference on Hot Topics in Operating Systems (HOTOS), 2015.
- C.E. Leiserson and T.B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)," in Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2010, pp. 303-314.
- S. Goddard, S. Kumar, and J.F. Prins, "Connected components algorithms for mesh-connected parallel computers," Parallel Algorithms 3rd DIMACS Implement. Chall., vol. 30, pp. 43-58, 1997.
- S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in Proc. International Conference on World Wide Web, 2011, pp. 607-614.
- R. Kiveris, S. Lattanzi, V. Mirrokni, V. Rastogi, and S. Vassilvitskii, "Connected Components in MapReduce and Beyond," in Proc. ACM Symposium on Cloud Computing (SOCC), 2014, pp. 1-13.

Significant online resources, including tutorials, API documentations, "quick start guides", etc. will be referenced during the course, and can be easily found using your favorite search engine. Piazza or similar workspace will be available for the duration of the course.

Computing Resources

For the duration of the course you will be granted access to the resources (including a storage) provided by the UB Center for Computational Research (CCR). CCR is the state-of-the-art HPC and data center hosting clusters, multi-core compute nodes, compute nodes with GPGPU accelerators and Intel Xeon Phi co-processors. It provides programming and execution environments supporting all types of parallelism covered in this course. Additionally, a virtual environment with a Linux distribution and all required compilers and runtime systems will be available for offline use.

Academic Integrity

You should be familiar with the university and departamental policies on academic integrity. The university policies for graduate students are available at:

`http://grad.buffalo.edu/study/progress/policylibrary.html`

and for undergraduate students at:

`http://undergrad-catalog.buffalo.edu/policies/course/integrity.shtml`.

The CSE policies are available from the CSE web page ↗ .

Any violation of these policies, including but not limited to cheating on any course deliverable (e.g. homework project, exam, etc.), will result in automatic failure of the course. There will be no leniency! If you decide to use a code from some external source, e.g. an open source project, you must include a proper and clearly visible attribution in your product (it is a good idea to contact your instructor to check if the code you plan to use is admissible).

Accessibility Resources

If you have any disability that requires reasonable accommodations to enable you to participate in this course, please contact the Office of Accessibility Resources, 25 Capen Hall, Phone: (716) 645-2608, and also the instructor of this course. The office will provide you with information and review appropriate arrangements for reasonable accommodations.