CS406 / CS531 – Parallel Computing – Spring 2020-2021
(3 credits)

Prerequisite: CS301

Learning objectives

- At the end of the course the student will understand the evolution of high performance and parallel computing with respect to laws and modern hardware.
- At the end of the course the student will be competent to measure the performance of parallel and distributed programs.
- At the end of the course the student will be competent to evaluate whether a parallel and distributed application is efficient or not and propose techniques to increase the performance.
- At the end of the course the student will be able to design, apply, and analyze parallel algorithms in problem solving.
- At the end of the course the student will gain hands-on experience with GPU programming with CUDA and multicore programming with OpenMP.

Topics covered

❑ **Introduction to (High-Performance, Parallel, etc.) Computing (2 weeks)**
a. **Computing** – *History, Turing Machines von Neumann architecture*
b. **Implicit Parallelism**: *Trends in microprocessor architectures - Superscalar execution, branch prediction*
c. **Limitations of Memory System Performance** – *Measuring system performance*, *memory wall, latency, bandwidth, caches (direct mapping, set associativity), temporal/spatial locality, multithreading, prefetching*
d. **Dichotomy of Parallel Computing Platforms** - *Control unit, SIMD, MIMD, vectorized instructions (SSE, AVX)*
e. **Communication Costs in Parallel Machines** - *Shared address space, message passing, NUMA and UMA architectures, interconnection networks*
f. **Messaging Cost Models and Routing Mechanisms** - *Store and forward routing, packet routing, cut-through routing, congestion, rules of thumb for communication*

❑ **Principles of Parallel Algorithm Design (1 week)**
a. **Tasks and Dependency Graphs** - *Granularity, degree of concurrency, critical path length*
b. **Task Interaction Graphs** - *Processes and mapping, partitioning, characteristics of task interactions*
c. **Decomposition** - *Recursive, data, exploratory, speculative, owner-computes rule*
d. **Mapping** - *Static and dynamic mapping, maximizing data locality, minimizing congestion, overlapping communication and computation.*

❑ **Analytical Modeling of Parallel Performance: (1 week):** Performance metrics for parallel systems Scalability of parallel systems Cost optimality, scalability, efficiency Asymptotic isoefficiency Minimum execution time Other scalability metrics —memory and time constrained scaling —serial fraction —Amdahl's law

❑ **Programming Shared-Memory Machines with OpenMP (2.5 weeks)**
a. **Multithreading –** *Concurrency, parallelism, shared memory computers, false sharing*
b. **Synchronization primitives –** *OpenMP memory model. barrier, locks, atomic operations, critical regions*
c. **Worksharing –** *Loops, static/dynamic/guided scheduling, reduction, shared/private variables,*
d. **OpenMP Tasking –** *Scheduling and dependencies*
e. **Thread binding and Memory Placement –** *OpenMP memory model, system topology, affinity and NUMA affects*

❑ **Programming GPUs with CUDA (3.5 weeks)**
a. **Simple GPU Usage:** *CUDA software stack, thread hierarchy (thread, block, grid), CUDA memory model, calling kernels, compilation, nvprof, cuda-memcheck*
b. **Synchronization in CUDA:** *Cooperating threads, warps, block synchronization, global synchronization, data races, atomic operations*
c. **Memory model:** *Using global memory, array of structures, structure of arrays, coalesced access, hiding latency, configuring and programming with shared memory, shared-memory banks and bank conflicts*
d. **GPU occupancy***: Resource limits, registers, shared memory*
e. **Unified memory:** *Zero-copy memory, pinned memory, mapping, on-demand paging, oversubscription*
f. **CUDA Streams:** *Pipelining, overlapping, advanced scheduling, creating streams, pinned memory*

❑ **Collective Communication Operations (1 week):** *One-to-all broadcast and all-to-one reduction. all-to-all broadcast and reduction. all-reduce and prefix-sum operations, scatter and gather, all-to-all personalized communication, circular shift Optimizing collective patterns*

- **Algorithms for Dense Matrices, Vectors, and Arrays (1 week):** *Matrix-vector multiplication, 1D and 2D partitionings, matrix-matrix multiplication, 2D partitioning and Cannon's algorithm, communication bounds, 2.5D matrix multiplication*

- **Parallel Sorting (if time permits, 1 week):** *Issues in sorting on parallel computers, sorting networks, bubble sort and its variants, quicksort, bucket and sample sort, other sorting algorithms*

Instructor

Dr. Kamer Kaya, FENS G012, ext. 9566, kaya@sabanciuniv.edu
Assistants: Information (offices, office hours, etc.) will be available through SUCourse.

Lecture Hours:

- Mon, 08:30-10:30: https://sabanciuniv.zoom.us/j/91765131031?pwd=M3JGck8xMldUeFM2amFmcnVwM1lPQT09
- Tue, 15:30-16:30: https://sabanciuniv.zoom.us/j/91765131031?pwd=M3JGck8xMldUeFM2amFmcnVwM1lPQT09

Textbook(s): There are no textbooks, slides are the main material
Reference books are

- Introduction to Parallel Computing, Second Edition. Ananth Grama. Anshul Gupta. George Karypis. Vipin Kumar.
- Programming Massively Parallel Processors: A Hands-on Approach, David B. Kirk and Wen-mei W. Hwu.

Reminder: we will not stick to the textbooks; you are responsible material covered in class. Thus it is important to attend to classes. This being said, attendance will not be used in grading.

Tentative Grading (subject to change)

- No Midterms or Finals (0%)
- Quizzes (30%)
  - Dates: TBA (there will be 4 quizes)
  - Maximum 3/4 will be taken into account
    - 30-40 minutes, attendance will be taken through Zoom,
    - cameras, audio will be open etc.
- Homework assignments (40%) – There will be 3-4 homework assignments which are not of equal weight. Homework grading will mostly be based on correctness, as well as the performance of the execution. No debugging will be done during grading.
- Project (30%) – For undergraduates, the problem will be announced. Graduate students can bring their own projects – see the explanations at the end of the document.

Other Rules and Remarks

- Weighted average is not the only criterion in letter grading; quiz/project score may also be taken into consideration in case of uncorrelated grades.
- Since there is no midterm or final, there is no makeup.
- **For proctored exams, your webcam and microphone should be on during the exam. In the case of non-compliance with this and other declared exam procedures, your exam will be void. Make sure to check that your webcam and microphone function properly before the exam.**
- **You must attend the synchronous Zoom lectures, recitations, etc. and real-time online exams with your SU email account.**
- **You may be asked to take an oral exam if the instructor of the course thinks that it is necessary.**

See course website for other, but important, details

Plagiarism, Homework Trading and Cheating will not be tolerated

This document is drafted from Harvard University CS205 (Computing Foundations for Computational Science) Project Page

Project Assignment Instructions: The final project is to solve a problem with parallel processing on **gandalf** and **nebula** (you will have the accounts during the term). You will identify a problem, analyze its compute scaling requirement, collect the data, design and implement a parallel software that can run both on multicore CPUs and manycore GPUs, and demonstrate scaled performance of an end-to-end application.

Project Requirements: Consider that your project should

- Demonstrate the need for parallel processing.
- Solve a problem for a non-trivial computation.
- Be implemented on a shared-memory architecture by simultaneously using a many-core and a multi-core device via hybrid OpenMP-CUDA parallelism.
- Be evaluated on various problem sizes and thread counts to demonstrate both weak and strong scaling using appropriate metrics (throughput, efficiency, iso-efficiency...).

Group Size: Students are required to form teams and to partition the work among the team members. The project must be done in teams with 4/5 students each (**not less, not more**). You can use SUCourse forum to find prospective team members. You may also find and discuss project ideas on the forum. In general, we do not anticipate that the grades for each group member will be different. However, we reserve the right to assign different grades to each group member if it becomes apparent that one of them put in a vastly different amount of effort than the others.

Project Milestones: There are five milestones for your final project. It is critical to note that no extensions will be given for any of these milestones for any reason. Projects submitted after the final due date will not be graded.

- **Team formation and tentative topic**
- A written, **3 page project proposal** (deadline: 30/03/2020)
- A written, **5 page project design and progress** (24/04/2020)
- **Project presentation** to teaching staff (29/05/2020 - Tentative)
- The **final report and project deliverables** submission (02/06/2020)

Project Proposal: Your group needs to prepare a project proposal (and submit the PDF via SuCourse) with the following sections:

- What is the problem you are trying to solve with this application?
- What is the need for parallelization and what can be achieved thanks to parallel processing?
- Describe your model and/or data in detail: where does it come from, what does it mean, etc.
- A basic literature survey.
- Which tools and infrastructures you are planning to use to build the application? What are their properties?

Project Progress: Your group needs to submit a project progress report in PDF covering the main aspects in the design and implementation of the parallel application with the following sections:

- Define the type of your application, the levels of parallelism exploited, the types of parallelism within the application.
- Specify programming model and infrastructure that you are using, and provide an analysis/profiling of the existing sequential parallel code (OpenMP. CUDA or both).
- Describe the main overheads (communication, synchronization, load balancing and sequential sections) in the parallelization and techniques that are applied to mitigate them
- Describe the numerical complexity of the algorithm
- Estimate the theoretical speed-up and scalability expected

Project Deliverables (see the details below)

- Web site as final report
- Software with evaluation data sets, test cases
- Final presentation and demonstration (during the last week or in the final exam week) to the teaching staff

Project Web Site: An important piece of your final project is a public web site that describes all the great work you did for your project. The web site serves as the final project report, and needs to describe your complete project. You can use GitHub Pages, or the README file on the GitHub repository, so you can easily refer to the software at the GitHub repository. You should assume the reader has no prior knowledge of your project and has not read your proposal. It should address the following aspects:

- Description of problem and the need for HPC.
- Description of solution and comparison with existing work on the problem
- Description of your model and/or data in detail: where did it come from, how did you acquire it, what does it mean, etc.
- Technical description of the parallel application, programming models, platform and infrastructure
- Links to repository with source code, evaluation data sets and test cases
- Technical description of the software design, code baseline, dependencies, how to use the code, and system and environment needed to reproduce your tests
- Performance evaluation (speed-up, throughput, weak and strong scaling) and discussion about overheads and optimizations done
- Final discussion about goals achieved, improvements suggested, lessons learnt, future work, interesting insights…
- Citations

Project Software: Your final project can be implemented using any API or programming language you would like. Make your own repository on GitHub with a link to your project web page. Software with evaluation data sets, test cases should be available on the repo. Include a README that describes the code and application files, and how your program should be run. We will be grading these projects on a variety of platforms, so you must include detailed instructions on how to run or compile your code. If we cannot run your application from the instructions included with your submission, we will not be able to grade this portion of your project. Your performance results should be reproducible, so you should provide all the information of the system and the environment needed to reproduce your tests.

Project Presentations: You will have 25 minutes to briefly present your project followed by 5 minutes of discussion time. You may prepare 5 slides for problem introduction, but we will enforce the 25-minute time limit. Focus the majority of your presentation on your main contributions rather than on technical details. What do you feel is the coolest part of your project? What insights did you gain? What is the single most important thing you would like to show the class? Upload the presentation to the GitHub repo.

Project Grading: The final project grades are dependent on the following criteria:

- Attempted difficulty: Some projects are harder than others (but every project can get 100). For example, an assignment based off of one of the homework assignments is probably easier than a completely new application, so you need to do more if you want to get full credits.
- Did you meet your major goals? The most important grading criteria is functionality: A working program will always garner the majority of available points; no credit will be given for non-working programs. A modest solution that works will be graded much more favorably than an ambitious "solution" that core dumps!

Project will be graded on the depth of work undertaken and communication (web site, presentation):

- 10%: Project proposal
- 20%: Project design and progress
- 40%: Software source code, design and documentation
- 30%: Final presentation and the quality of the deliverables

Extra points may be earned for the use of **advanced features** like:

- Using multiple GPUs at once.
- Additional use of modules of platforms not explained in class (e.g., ML or Graph modules in Spark...)
- Advanced techniques to mitigate overheads (overlap of communication and computation, optimizing existing libraries...)
- Challenging parallelization or implementation aspects