High Performance Computing
CSE 443/543
Exam #1 Study Guide

Thursday, Sept 30th 2021
During class/lab

Text Coverage:

- *Introduction to Parallel Computing*: Chapter 1 and Chapter 2.1, 2.2. See Canvas Syllabus page for links.
- "C++ How to Program" by Paul Deitel and Harvey Deitel. Prentice Hall. February 12 2013 (ISBN-10: 0-13-337871-1, ISBN-13: 978-0-13-337871-9). See Canvas Syllabus page for links.

Types of Questions to expect:

Combination of multiple-choice, fill in the blanks, short answer, "what does the following code do", and "write the C++ code to do the following" questions. You will need to be able to read and write C++ code. You will be asked to write various C++ programs similar to those in lecture notes, exercises, and homework assignments. Few of the past exams have been included in the study material to provide example of the type of questions to expect in the exam.

Available Materials:

Closed book, simple calculators (cannot be a graphing calculator, have a QWERTY keyboard, or be integrated with other devices like cell phone, iPOD etc.). Other than a calculator, no other electronic devices will be permitted during the exam. You will be provided with the selected method list (see supplied common methods and commands document) along with your exam for your reference.

Concepts you should know:

- **Generic Concepts:** Units of memory sizes (KB, MB, GB), CPU speeds (Hz, MHZ, GHZ), Time (sec, milliseconds (msec), microseconds (usec), nanoseconds (nsec)) FLOPS and conversion between related families of units.
- **Architecture:** Terminology and concepts covered in course including their advantages and drawbacks -- Basics of CPUs and caches. Concepts related to Superscalar, SIMD, SSE CPUs. Basics of concurrency versus parallelism. Flynn's taxonomy
- **Timing measurement and analysis:** Difference between user time, system time, and elapsed time. Why timings for programs vary and why multiple runs are needed. Computing average timing.

- **Compiler optimization & libraries:** Basics of compiler optimization. Flags to use for GCC.

- **Instruction Level Parallelism (ILP):** Superscalar processing, SIMD, and SSE concepts. Pipelining and 4-stages in a pipeline (IF, ID, EX, WB).
    - Hazards in a pipeline:
        - data hazards,
        - control hazards,
        - and structural hazards. Effect of caches – cache hit vs. cache miss.
    - Reducing hazards in a pipeline
        - Restructuring code – using switch vs. if, loop unrolling.
        - Method inlining.
        - Using basic profile information to reorganize a program
        - Changing memory access patterns to improve caching
    - Profiling and profile guided optimization (PGO) approaches.

- **Benchmarking and Profiling:** Concept of benchmarking and types of benchmarks. Issues involved in designing micro-benchmarks. Assessing validity of micro-benchmarks using profilers. Concept of profiling and types of profilers. Using Linux `perf`. Interpreting results from Linux `perf`.
- Using `/usr/bin/time` to measure runtime characteristics of programs
    - User time, System time, Elapsed time, %CPU
    - Observing peak memory usage
    - Interpreting output of `/usr/bin/time` for single vs. multithreaded programs

- **Concepts of efficiency:** Runtime/CPU efficiency, memory efficiency, energy efficiency. Identifying efficiency constraints in application. Rewriting programs to balance efficiency requirements – example – rewrite method to reduce runtime while using more memory and vise versa.

- **C++ programming:**
    a. **Basic program constructs**
        i. Variables & expressions
        ii. if-else and if-else statements
        iii. switch statement
        iv. Looping constructs (for, while, do…while, range-for)
        v. Functions/methods
            1. Pass by value versus pass by reference
            2. Performance and memory impact of pass-by-value
            3. Using `const` keyword for parameters.
        vi. Default values for parameters

    b. **Classes and objects**
        i. Using `std::string`
            1. Constructors for string.
            2. Methods for operating and accessing strings

3. Conversion to-and-from numeric data types to `std::string`.

c. Arrays

i. Basics of arrays.
ii. 1-D arrays
iii. 2-D arrays
    1. Row major organization
iv. Command-line arguments

d. Vectors, iterators, and algorithms

i. Use of vectors and iterators for processing collection of data
ii. Create type aliases via the `using` clause in C++
    1. Creating aliases given English description
    2. Tracing aliases back to their original types.
iii. Operations on a vector: adding elements, accessing elements, removing elements, etc.
iv. Standard algorithms such as: `for_each`, `copy`, `copy_if`, `copy_unique`, `sort`, `find`, `min_element`, `max_element`, `min`, `max`.
v. Using external functions with algorithms
vi. Using lambdas with algorithms.
vii. Reading/printing/writing vectors to I/O streams

e. Hash maps (`unordered_map`)

i. Use of `unordered_map`
ii. Using `unordered_map` as associative arrays
iii. Defining and using `unordered_maps` of different data types
iv. Looking-up values in `unordered_maps`
v. Iterating over all the entries in a map and processing them

f. Basic text file I/O operations

i. Reading and writing data to console using `std::cin` and `std::cout`.
ii. Using stream-insertion (<<) and stream-extraction (>>) operators to read and write data.
    1. Understanding these operators and how they handle whitespaces.
iii. Using `std::getline` method to read a full line of text
iv. Using `std::ifstream` and `std::ofstream` to read/write text files.
v. Using `std::istringstream` and `std::ostringstream` to perform I/O with strings.

g. Generic concepts

i. Fundamentals of problem solving
ii. Source code, pseudo code, algorithm
iii. Syntax errors and troubleshooting them

iv. Semantic errors and troubleshooting them
v. Functional testing (using `diff` to compare outputs)
vi. Debugging, debugger, break points, stack trace
vii. Concept of data types and information that can be inferred from data types
viii. Basics of files: path, absolute vs. relative path, directory vs. file. Executable vs. source file.
ix. Performance profiling and using profilers
x. Interpreting output of profilers to draw conclusions

h. Other exercises

i. Converting English statements to corresponding C++ statements
ii. Describing C++ statements in English
iii. Code walkthroughs to determine operation and output from a C++ program
iv. Developing a C++ program given a functional description
v. Identifying performance or memory issues in C++ programs
vi. Rewriting C++ program to address memory or performance issue
vii. Interpreting data in the form of a chart/graph
viii. Interpreting data in the form of a table and computing runtime statistics.

- **Basic Linux Commands:** basic operations on Linux, changing directory, creating programs, compiling and executing programs, observing processes, listing files, redirection of I/O at the shell.

- **SLURM scripts:**
  1. Reading SLURM script.
  2. Identify key settings, including: job name, number of cores requested, peak memory requested, wall time requested.
  3. Modifying/completing a given SLURM script to run a specific command.

Preparation Suggestions:

1. Do read the textbook materials while paying attention to implementation/application details.
2. Redo lab exercises. Develop short programs to test/verify your understanding of concepts. Review vectors, how to use vectors.
3. Review homework solutions.
4. **Review the functionality of pertinent methods and commands in the supplied method/command sheet**.
5. Review the handouts material.