SYLLABUS FOR GRAD 510 (3-CREDIT COURSE)
FUNDAMENTALS OF HIGH PERFORMANCE "SCIENTIFIC" COMPUTING

1. Contact Information

Instructor : Dr. Stephen Guzik
Phone : (970) 491-4682
Email : Stephen.Guzik@colostate.edu
Office : A103H Engineering

2. Resources

Koenig et al., "Accelerated C++", Addison-Wesley, 2000.
Meyers, "Effective Modern C++", O'Reilly, 2014.
Stroustrup, "The C++ Programming Language, 4e", Addison-Wesley, 2013.
Vandevoorde et al., "C++ Templates: The Complete Guide, 2e", Addison-Wesley Professional, 2018.
Gropp et al., "Using MPI, 3e", The MIT Press, 2014.
Gropp et al., "Using Advanced MPI", The MIT Press, 2014.
Chapman et al., "Using OpenMP", The MIT Press, 2007.
Online: GNU/Linux Command-Line, `https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/GNU-Linux-Tools-Summary.html`
Online: Bash Manual, `https://www.gnu.org/software/bash/manual/bash.html`
Online: Advanced Bash-Scripting Guide, `https://tldp.org/LDP/abs/html/`
Online: Git Documentation, `https://git-scm.com/doc`
Online: GNU Make, `https://www.gnu.org/software/make/manual/`
Online: GNU Autoconf, `https://www.gnu.org/software/autoconf/manual/`
Online: CUDA Programming Guide, `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`

3. Course Learning Objectives

This course prepares students for starting or joining large software projects designed for execution on modern supercomputers. Upon successful completion of this course, students will be able to:

(1) Efficiently use the Linux operating system
(2) Apply best-practices in software engineering
(3) Develop scientific software using C++ and object-oriented-design paradigms
(4) Use parallel programming models and parallel I/O to achieve high performance on parallel architectures
(5) Integrate libraries and multi-language programming into applications
(6) Design applications and libraries using levels of abstraction.

4. Prerequisites

There are no formal prerequisites for the course. However, students should be both familiar with and enjoy programming; there will be a lot of it in the course. The course will be taught using C++. Knowledge of C++ is not a prerequisite but students should have a good understanding of basic data types, control structures, and functions common to most languages. Finally, an understanding of undergraduate mathematics for scientists and engineers is expected.

5. Course Outline

Week	Topics
1	Introduction, Linux (host/VM), Shell, Architectures (compute units, memory hierarchy, temporal and spatial caching), Programming models and abstract machines, 7 Motifs of scientific computing.
2	Memory addressing (physical/virtual) and leaks, Pointers and references, Primitive types, Loops.
3	Version control with Git, Application and library design principles, Translation units, Object-oriented programming and the C++ class.
4	Makefiles, Unit/regression testing, Metadata vs. data (e.g., set-calculus), Constructors (L- and R-value references), STL containers and pitfalls.
5	Doxygen (documentation), Heap vs. stack allocations, RAII/smart-pointers/memory management, Multidimensional array classes.
6	Debugger (and Valgrind), Iterators, Re-shaping linear memory and multidimensional array indexing, Stencil calculations, Timing code.
7	Multi-language programming (argument passing), object code, assembly code, LAPACK.
8	Distributed (or not) meta-data and data, Abstracting parallel vs. serial programming, Serial and parallel iterators.
9	Configure, MPI, Caching motion patterns, Serial and parallel exchange, Debugging in Parallel.
10	Parallel I/O, HDF5, CGNS and I/O application.
11	Capstone distributed-memory application and algorithm review. State and operators. Parallel scaling.
12	Using supercomputers, Queuing systems, Remote visualization, OpenMP (programming-model injection), Shared-memory programming.
13	CUDA, Multidimensional arrays on GPUs, accelerated Laplacian (wave equation or similar).
14	No new content: Lectures converted to labs.
15	C++ concurrency library, vectorization, task-based parallelism, data-flow programming, advanced topics.
16	Capstone application submissions. Code performance competition results!

6. Course Grading Scheme and Policies

(1) **Grading is a composite of the following weighted items**: The final grade will be based on marks obtained for the following components.
   (a) 50% for the weekly assignments (evaluated by testing infrastructure)
   (b) 50% for the capstone applications (evaluated by performance and results)

   The grading schema is given in the following table:

Letter Grade	Percentage
A+	90–100%
A	85–89%
A-	80–84%
B+	77–79%
B	73–76%
B-	70–72%
C	60–69%
F	0–69%

(2) **Course Policies**:
   (a) Assignments must be electronically submitted by 11:59 PM on the due date. A penalty of 25% per day will be imposed on late assignments. Written assignments must be submitted in class and presented clearly.
(3) This course will adhere to the Academic Integrity Policy of the Colorado State University General Catalog and the Student Conduct Code.