For the final examination question will be chosen from the lectures in the following general areas. You can expect three to five questions from each group.

SCORING: For most questions you get marks for a correct answer, and zero if it is unanswered or incorrect. There are a few questions where you can select multiple answers, and for those you get points deducted for a wrong answer. No question can score less than zero points.

Compilers and Queues

- Version control (e.g. "git"): clone, pull, add, commit, push.
- Phases of a compiler (compile, link).
- Importance of compiler optimization and how can affect the results.
- Basics of "make"
- Purpose and function of batch queue systems (e.g. SLURM). Scheduling: priority versus FIFO. How you request resources (nodes, cores, etc.)

Performance

- Moore's Law and how it relates to the Top500 list
- Strong and weak scaling
- Latency and bandwidth.
- Memory, cache and bus (e.g., attached network or GPU card) hierarchy and their relative performance.
- Different ways of instrumenting (benchmarking) your code.

OpenMP

- Directive based
- Model (shared memory, threads, a "thread" runs on a "core")
- The serial and "parallel" (region) parts.
- The "parallel for" loop.
- Synchronization between threads, for example "reduce" clause, or "atomic" or "critical" pragmas. Performance of each.
- Shared versus private variables.
- How OpenMP schedules work between threads.

MPI

- Model (distributed memory, message passing, a "rank" is a process).
- How to split work between "ranks" (also known as load balancing or domain decomposition).
- How to compile MPI programs (e.g., mpicc, cc, mpicxx, CC, etc.).
- Some of the common functions (those covered in the lectures).
- How message passing works and common problems (e.g., deadlocks).

Cloud & Containers

- Difference between Cloud (Virtual Machines) and containers
    - VM: a “virtual” computer. Has memory, CPUs, network, disks, etc.
- How to handle “persistence” with VMs and containers, e.g., “snapshots”.
- Ephemeral computing (create a resource, compute, throw away the resource).
- You are “root” in virtual machines and containers and what this means.
- Dockerhub: like github but for containers.
- How to access data
    - images and snapshots for VMs; mounting host directories for containers.

MapReduce

- Why the “compute” is sent to the “data” instead of the normal “HPC” way.
- Data model: write once and read (process) multiple times.
- What the “map” and “reduce” phases do.
- What are the “key” and “value”?

Hybrid Computing

- CPU “sockets” and “cores”, and GPU “SMs” and “cores”.
- SIMD (AVX) on CPUs and “Warps” on GPUs.
- CPU versus GPU
    - CPU: small number of high performance cores and ~ one thead per core.
    - GPU: large number of lower performance cores and many threads per core.
    - Memory bandwidth to memory on each.
- Divergence: how it is handled on the CPU (SIMD) and on the GPU (Warps).
- Data alignment: what it is and why is it important.
- Latency hiding and Occupancy on GPUs. Latency to start kernels or data transfer. Latency of instruction on the GPU versus the CPU.

OpenACC

- Directive Based
- Basic GPU operations: allocate, copy, kernel launch.
- Data management: how OpenACC gets your data to where it needs to be (GPU or CPU) and how you can steer this with a “data” construct.
- Difference between “kernels” construct and “parallel” construct.
- Difference between “grid”, “worker” and “vector”.
- Shared versus private variables.
- Synchronization on the GPU, for example “reduce” clause, or “atomic” or “critical” pragmas. Performance of each.
- What asynchronous operations do and way you would want to use them.

CUDA

- How to compile CUDA code (nvcc).
- What a streaming multiprocessor (SM) is.
- What is a “grid”, “block”, “warp” and “thread” and how they relate to the SM.
- How indexing works. How to turn a thread and block index into a global index.