Introduction to Parallel Computing (CMSC416)

Course Overview

Alan Sussman, Department of Computer Science

UNIVERSITY OF MARYLAND

About the instructor

- Professor at UMD for ~20 years, research scientist before that
  - Research is in high performance parallel and distributed computing
- Recently returned from a rotation at the National Science Foundation as a program director, in the Office of Advanced Cyberinfrastructure

Introductions

- Name
- Sophomore/Junior/Senior
- Something interesting/ unique about yourself
- Why this course? (optional)

This course is

- An introduction to parallel computing
- Upper Level CS Coursework / General Track / Area 1: Systems
- Work expected:
  - Four programming assignments
  - Four quizzes
  - Midterm exam: in class on March 30 (tentative)
  - Final exam: on May 19 (10:30AM -12:30PM)

Course topics

- Introduction to parallel computing (1 week)
- Parallel architectures and networks (1 week)
- Shared memory parallel programming (1 week)
- Distributed memory parallel programming (1 week)
- GPU programming (1 week)
- Parallel algorithms (2 weeks)
- Debugging & Instrumentation (1 week)
- Performance tools (1 week)
- Performance optimizations (1 week)
- Parallel I/O and Networks (1 week)
- Scientific and other applications (1 week)

Tools we will use for the class

- Syllabus, lecture slides, assignment descriptions on course website:
  - http://www.cs.umd.edu/class/spring2023/cmsc416
- Assignment submissions and quizzes on ELMS
- Discussions: Piazza
  - piazza.com/umd/spring2023/cmsc416/home
- If you want to send an email, cc both TAs and me
  - Put [CMSC416] in the subject line

Zaratan accounts

- Zaratan is the university compute cluster where you will do the programming assignments
- The TAs will provide instructions for logging on to zaratan
- Helpful resources:
  - https://hpcc.umd.edu/hpcc/help/usage.html
  - https://missing.csail.mit.edu
  - https://gitlab.cs.umd.edu/mmarsh/quick-refs

Excused absence

Any student who needs to be excused for an absence from a single lecture, due to a medically necessitated absence shall make a reasonable attempt to inform the instructor of his/her illness prior to the class. Upon returning to the class, present the instructor with a self-signed note attesting to the date of their illness. Each note must contain an acknowledgment by the student that the information provided is true and correct. Providing false information to University officials is prohibited under Part 9(i) of the Code of Student Conduct (V-1.00(B) University of Maryland Code of Student Conduct) and may result in disciplinary action.

Self-documentation may not be used for Major Scheduled Grading Events (midterm and final exams) and it may only be used for one class meeting during the semester. Any student who needs to be excused for a prolonged absence (two or more consecutive class meetings), or for a Major Scheduled Grading Event, must provide written documentation of the illness from the Health Center or from an outside health care provider. This documentation must verify dates of treatment and indicate the timeframe that the student was unable to meet academic responsibilities. In addition, it must contain the name and phone number of the medical service provider to be used if verification is needed. No diagnostic information will ever be requested.

What is parallel computing?

- Serial or sequential computing: doing a task in sequence on a single processor
- Parallel computing: breaking up a task into sub-tasks and doing them in parallel (concurrently) on a set of processors (often connected by a network)
- Some tasks do not need any communication: embarrassingly/pleasingly parallel

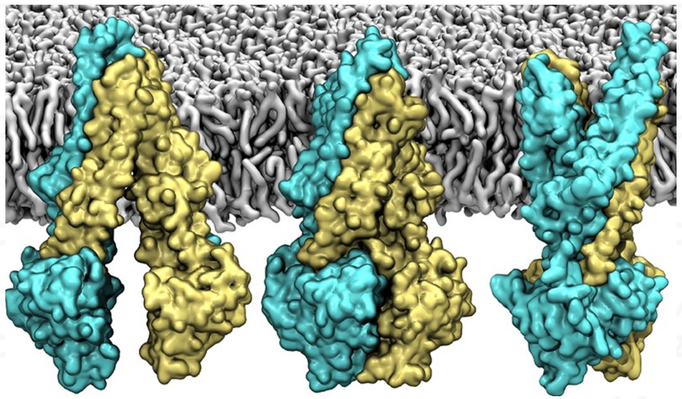
What is parallel computing?

- Does it include:
  - Grid computing
  - Distributed computing
  - Cloud computing
- Does it include:
  - Superscalar processors
  - Vector processors
  - Accelerators (GPUs, FPGAs)

The need for parallel computing or HPC

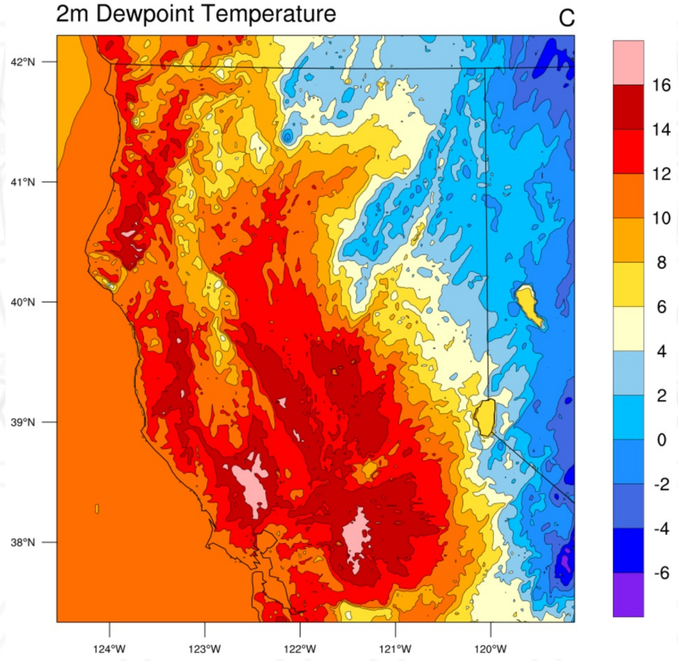
HPC stands for High Performance Computing

Drug discovery



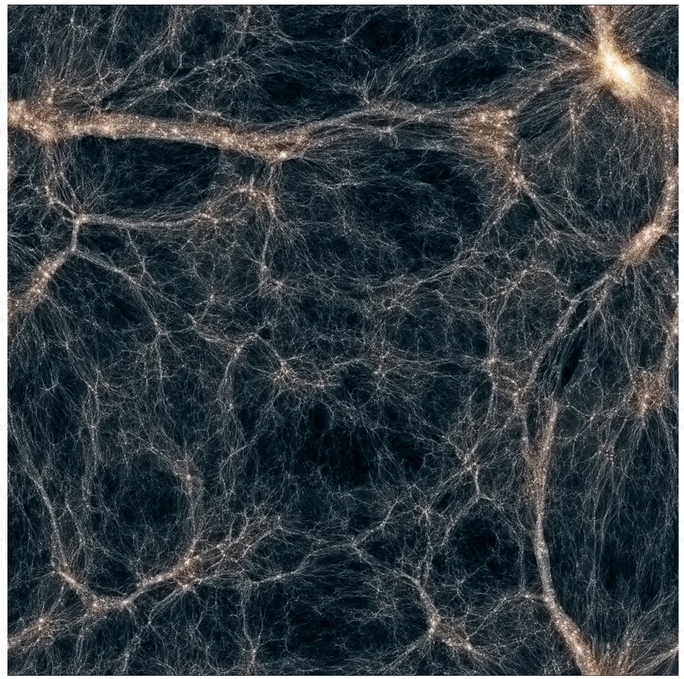
https://www.nature.com/articles/nature21414

Weather forecasting



https://www.ncl.ucar.edu/Applications/wrf.shtml

Study of the universe



https://www.nas.nasa.gov/SC14/demos/demo27.html

Why do we need parallelism?

- Make some science simulations feasible in the lifetime of humans
  - Either due to speed or memory requirements
- Provide answers in realtime or near realtime
- Overall, 2 main reasons
  - To structure your program so parts run concurrently
  - To achieve the performance you need for your application, which can't be obtained from a single processor/core (and its associated memory)

Large supercomputers

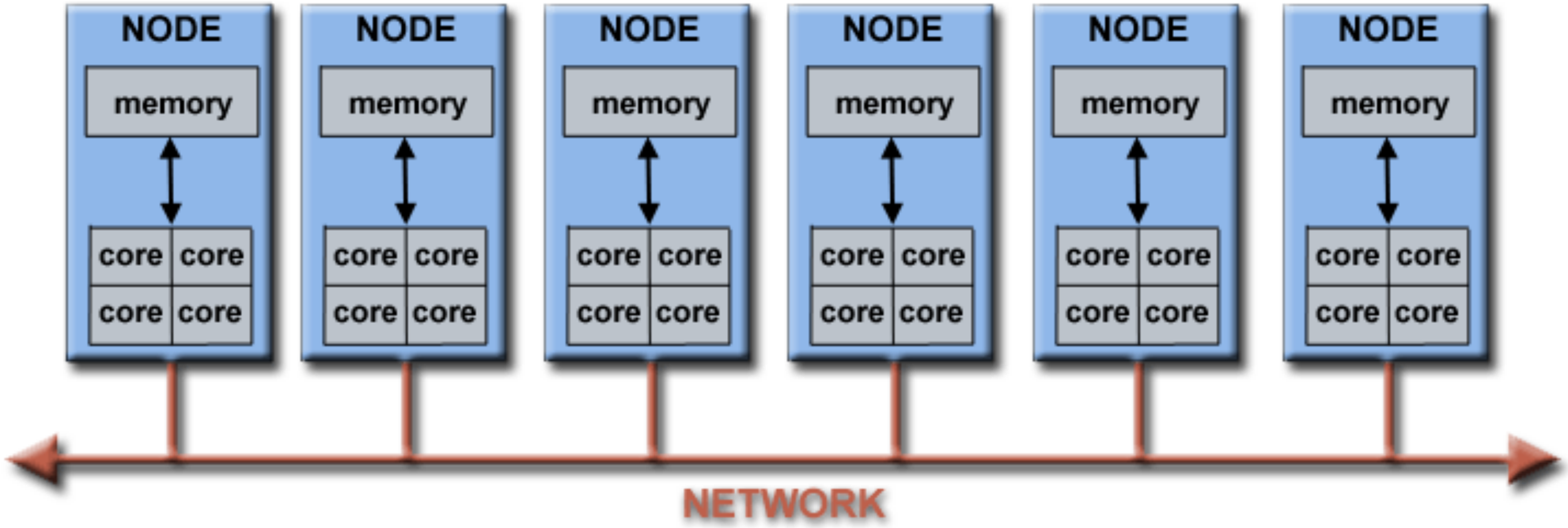
- Top500 list: https://www.top500.org/lists/top500/2022/11/

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

https://www.olcf.ornl.gov/frontier/

Parallel architecture

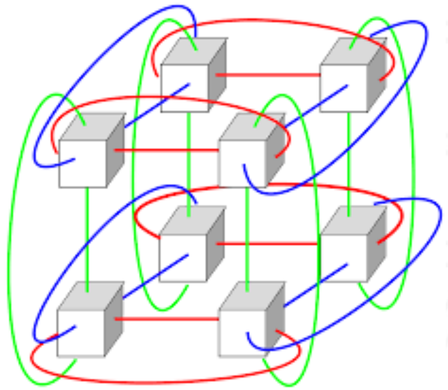
- A set of nodes or processing elements connected by a network.



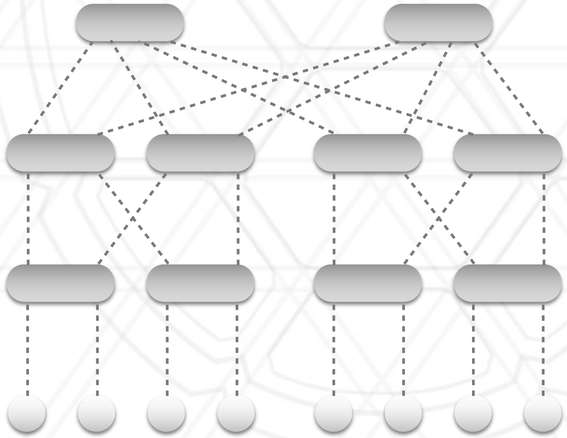
https://computing.llnl.gov/tutorials/parallel_comp

Interconnection networks

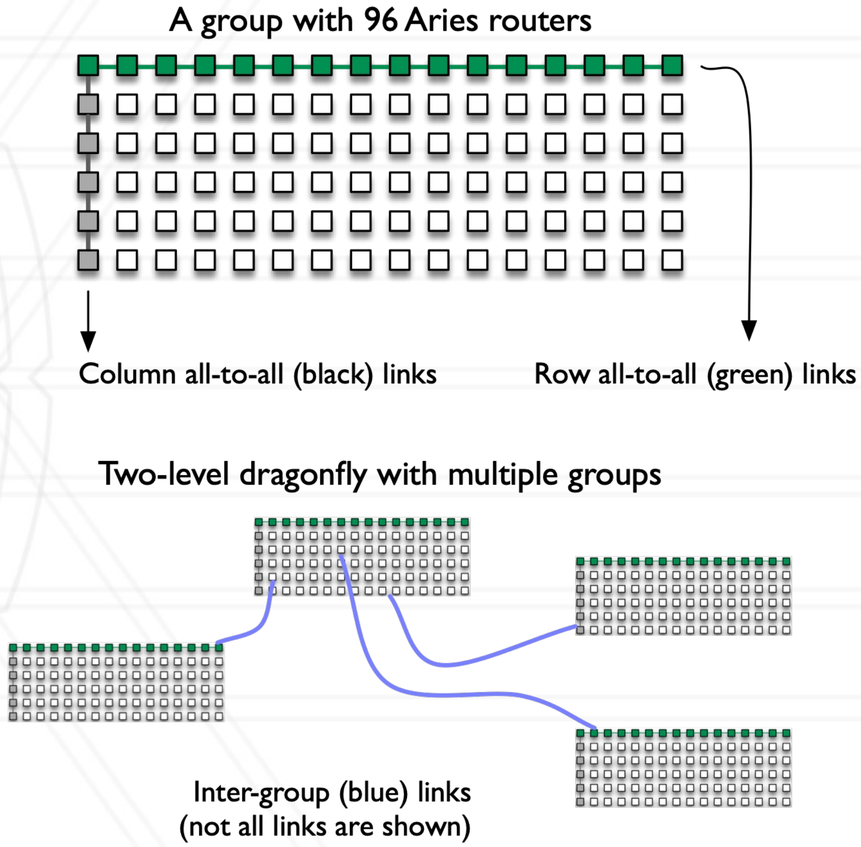
- Different topologies for connecting nodes together
- Used in the past: torus, hypercube
- More popular currently: fat-tree, dragonfly



Torus



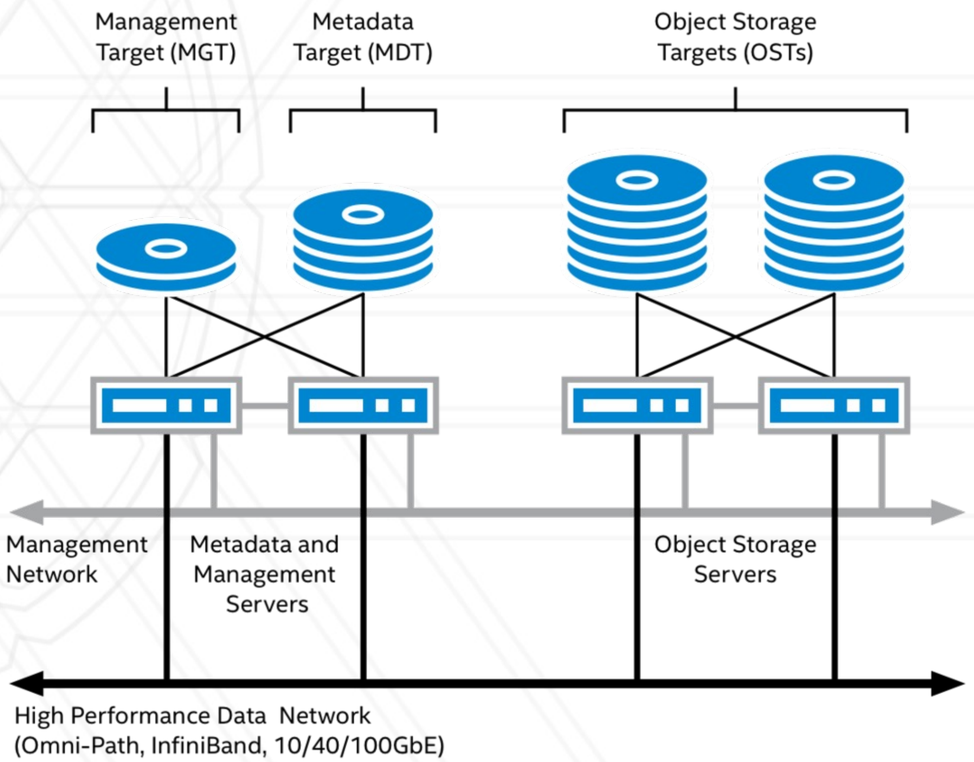
Fat-tree



Dragonfly

I/O sub-system / Parallel file system

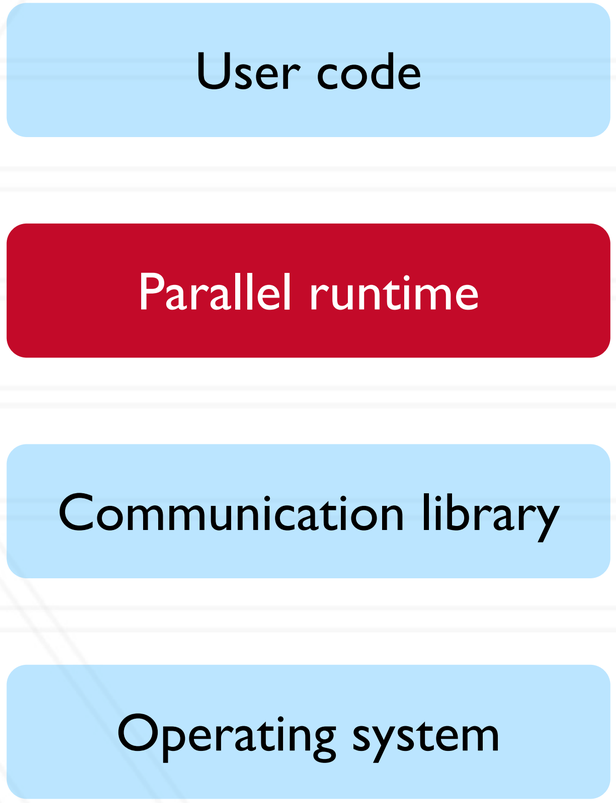
- Home directories and scratch space typically on a parallel file system
- Mounted on all login and compute nodes



http://wiki.lustre.org/Introduction_to_Lustre

System software: models and runtimes

- Parallel programming model
  - Parallelism is achieved through language constructs or by making calls to a library and the execution model depends on the language/library used.
- Parallel runtime [system]:
  - Implements the parallel execution model
- Shared memory/address-space
  - Pthreads, OpenMP, PGAS
- Distributed memory
  - MPI, Charm



Terminology and Definitions

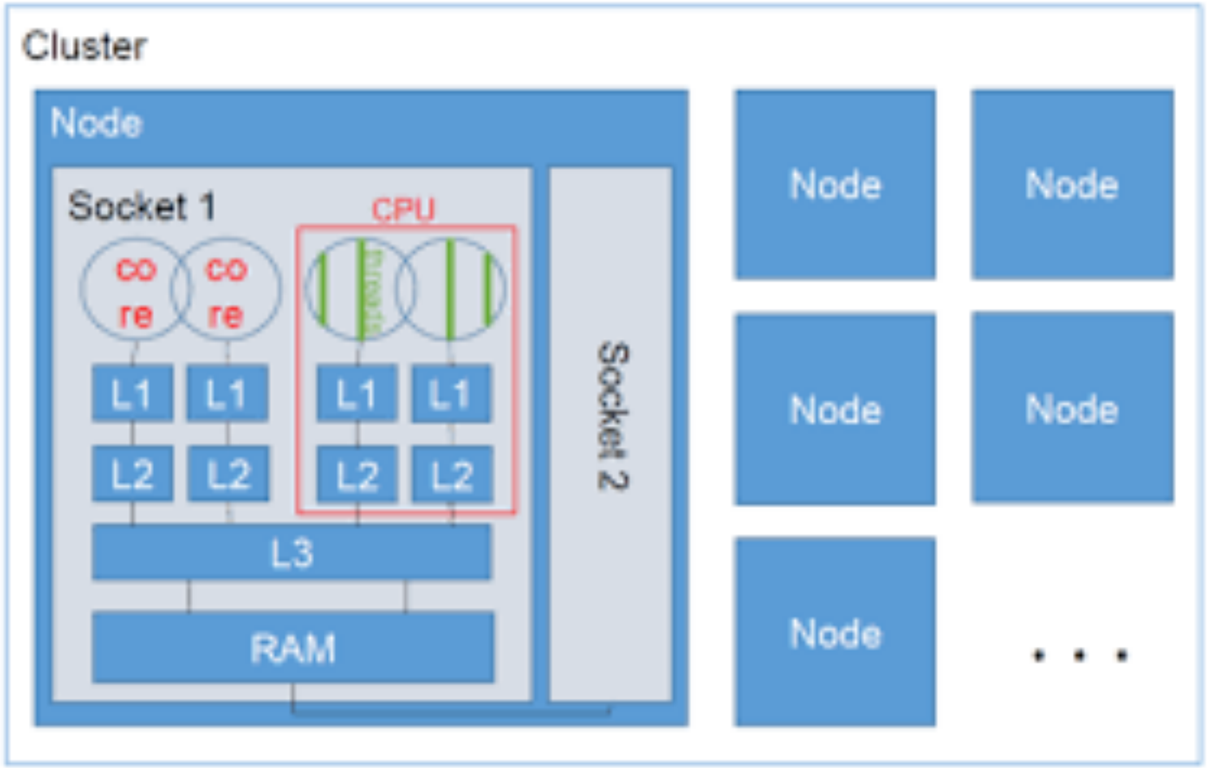
UNIVERSITY OF MARYLAND

Announcements

- Course ELMS site has been published
  - First announcement includes how to sign up for Piazza (need an access code now) – please sign up!
- Office hours will start next week, for both the TAs and me
  - Will be posted on class home web page
  - Both in-person and online hours

Cores, sockets, nodes

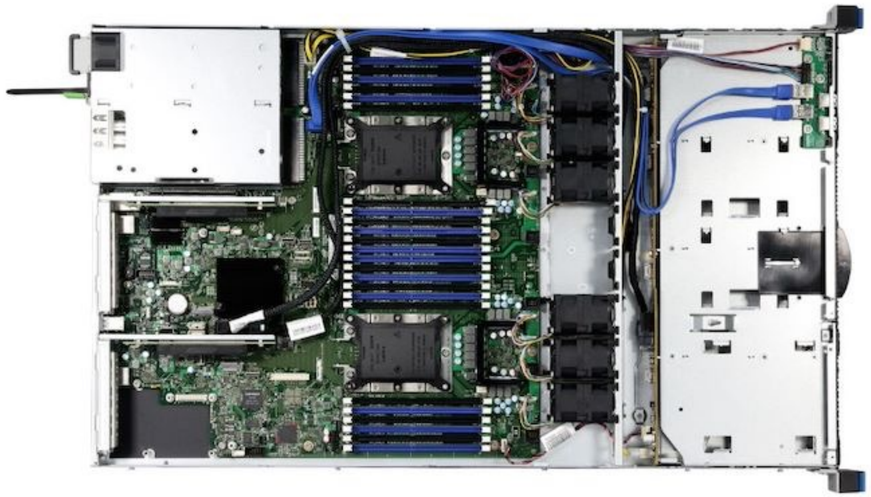
- Core: a single execution unit that has a private L1 cache and can execute instructions independently
- Processor: several cores on a single Integrated Circuit (IC) or chip are called a multi-core processor
- Socket: physical connector into which an IC/chip or processor is inserted.
- Node: a packaging of sockets - motherboard or printed circuit board (PCB) that has multiple sockets

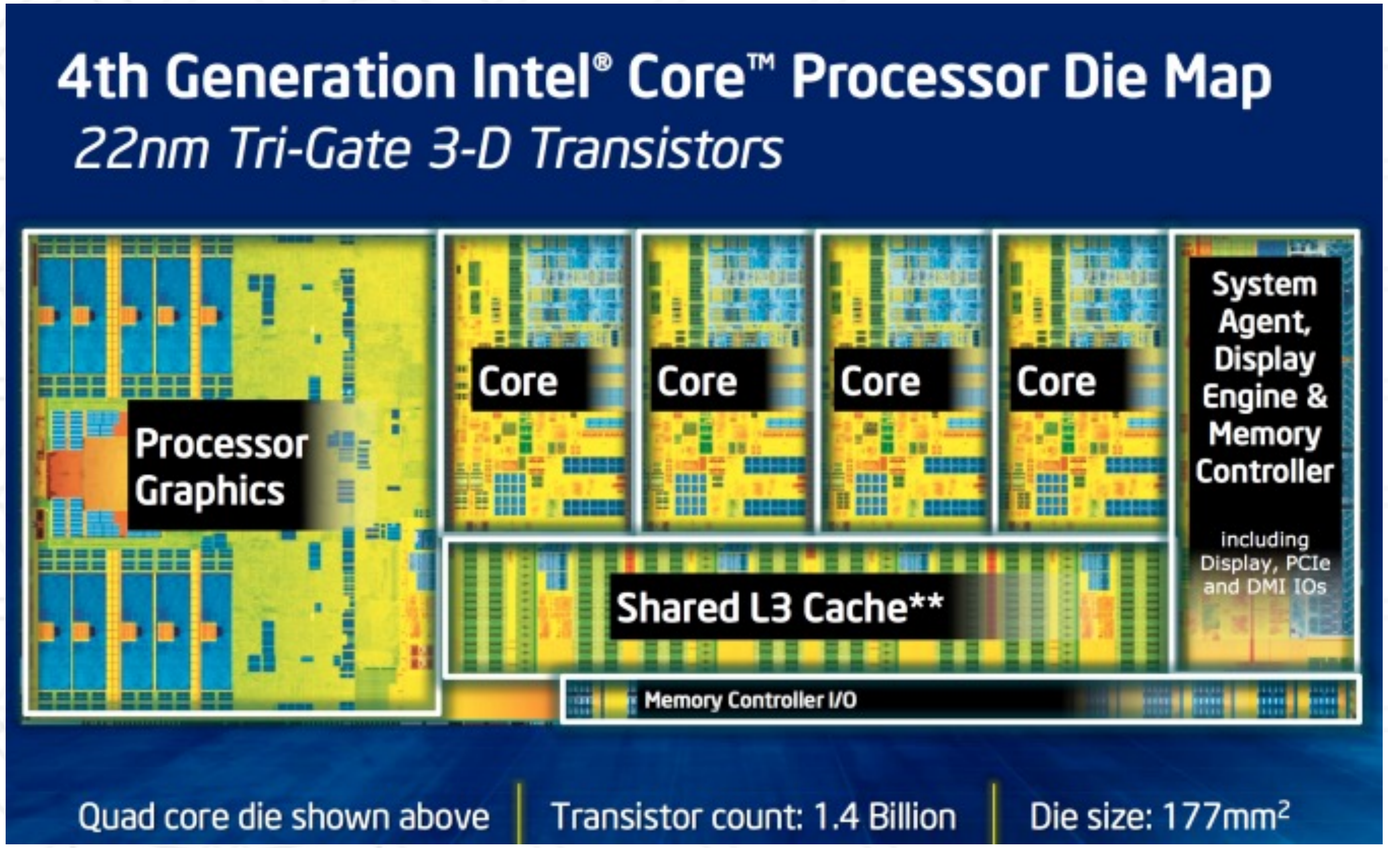


https://hpc-wiki.info/hpc/HPC-Dictionary

Rackmount servers

Rackmount server motherboard





https://www.anandtech.com/show/15924/chenbro-announces-rb13804-dual-socket-1u-xeon-4-bay-hpc-barebones-server

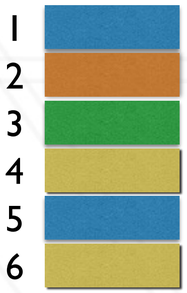
https://www.anandtech.com/show/7003/the-haswell-review-intel-core-i74770k-i54560k-tested

Job scheduling

- HPC (and clusters) systems use job or batch scheduling
- Each user submits their parallel programs for execution to a "job" scheduler
- The scheduler decides:
  - what job to schedule next (based on an algorithm: FCFS, priority-based, ….)
  - what resources (compute nodes) to allocate to the ready job

- Compute nodes: dedicated to each job (at least part of a node)
- Network, filesystem: shared by all jobs

Job Queue



#Nodes Requested	Time Requested
128	30 mins
64	24 hours
56	6 hours
192	12 hours
…	…
…	…

Compute nodes vs. login nodes

- Compute nodes: dedicated nodes for running jobs
  - Can only be accessed when they have been allocated to a user by the job scheduler
- Login nodes: nodes shared by all users to compile their programs, submit jobs etc.
- Also I/O nodes to act as intermediaries between compute/login nodes and the parallel I/O system (shared by all compute and login nodes)

Supercomputers vs. commodity clusters

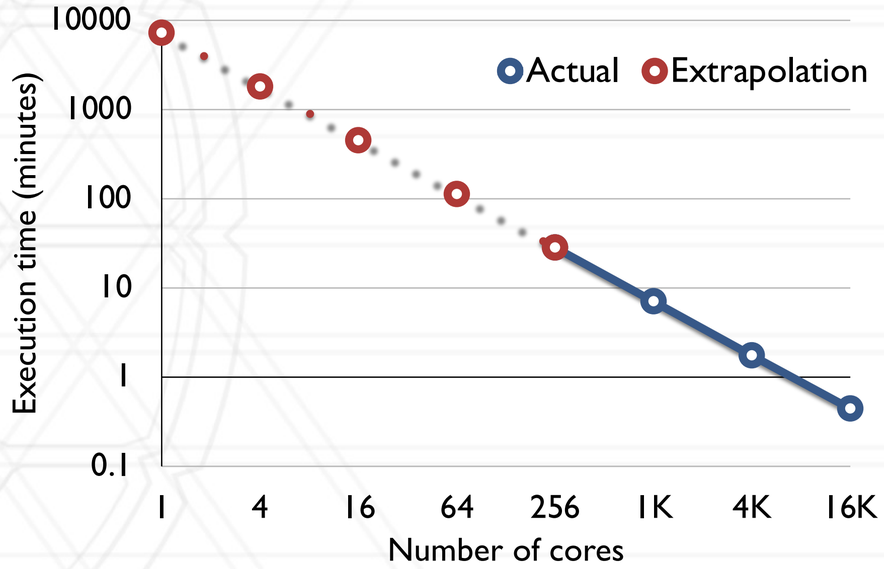
- Supercomputer refers to a large expensive installation, typically using at least some custom hardware
  - High-speed interconnect
  - IBM Blue Gene, Cray XT, Cray XC
  - But generally defined as the fastest machines currently available
- Cluster refers to a cluster of nodes, typically put together using commodity (off-the-shelf) hardware

Serial vs. parallel code

- Thread: a thread or path of execution managed by the OS
  - Threads share the same memory address space
- Process: heavy-weight, processes do not share resources such as memory, file descriptors etc.
  - Usually defined as an address space with one or more threads running in it (from CMSC216)
- Serial or sequential code: can only run on a single thread or process
- Parallel code: can be run using one or more threads or processes

Scaling and scalable

- Scaling: running a parallel program on 1 to n processes
  - 1, 2, 3, … , n
  - 1, 2, 4, 8, …, n
- Scalable: A program is *scalable* if its performance improves when using more resources



Weak versus strong scaling

- Strong scaling: *Fixed total* problem size as we run on more resources (processes, threads)
  - Sorting n numbers on 1 process, 2 processes, 4 processes, …
- Weak scaling: Fixed problem size per thread/process but *increasing total* problem size as we run on more threads/processes
  - Sorting n numbers on 1 process
  - 2n numbers on 2 processes
  - 4n numbers on 4 processes

Speedup and efficiency

- Speedup: Ratio of execution time on one process to that on p processes

$$\text{Speedup} = \frac{t_1}{t_p}$$

- Efficiency: Speedup per process

$$\text{Efficiency} = \frac{t_1}{t_p \times p} \qquad \text{So 1, if } t_p = \frac{t_1}{p}$$

Amdahl's law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial "bottleneck"
- Lets say only a fraction f of the code can be parallelized on p processes

$$\text{Speedup} = \frac{1}{(1-f) + f/p}$$

Amdahl's law

$$\text{Speedup} = \frac{1}{(1-f) + f/p}$$

```
fprintf(stdout,"Process %d of %d is on %s\n",
    myid, numprocs, processor_name);
fflush(stdout);

n = 10000;          /* default # of rectangles */
if (myid == 0)
startwtime = MPI_Wtime();
```

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h   = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs)
{
x = h * ((double)i - 0.5);
sum += f(x);
}
mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Total time on 1 process = 100s
Serial portion = 40s
Portion that can be parallelized = 60s

$$f = \frac{60}{100} = 0.6$$

$$\text{Speedup} = \frac{1}{(1-0.6) + 0.6/p}$$

Communication and synchronization

- Each process may execute serial code independently for a while
- When data is needed from other (remote) processes, messaging occurs
  - Referred to as communication or synchronization or MPI messages
- Intra-node vs. inter-node communication
- Bulk synchronous programs: All processes compute simultaneously, then synchronize/communicate
- Similar for threads, but replace messages with shared variables, and synchronization is through locks, semaphores, …

Different models of parallel computation

- SIMD: Single Instruction Multiple Data
  - GPUs and CPU SIMD instructions
- MIMD: Multiple Instruction Multiple Data
  - Each thread/process runs different code on different data
- SPMD: Single Program Multiple Data
  - Typical in HPC and parallel computing with many threads/processes
  - Each thread/process runs the same code on different data

UNIVERSITY OF
MARYLAND