HPC Programming Bootcamp

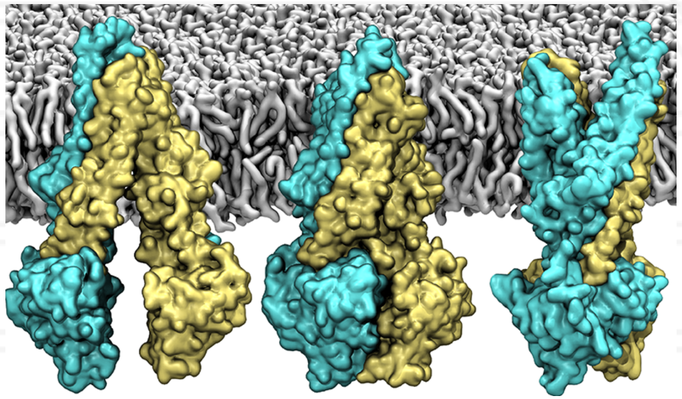
Day 1:

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

The need for high performance computing

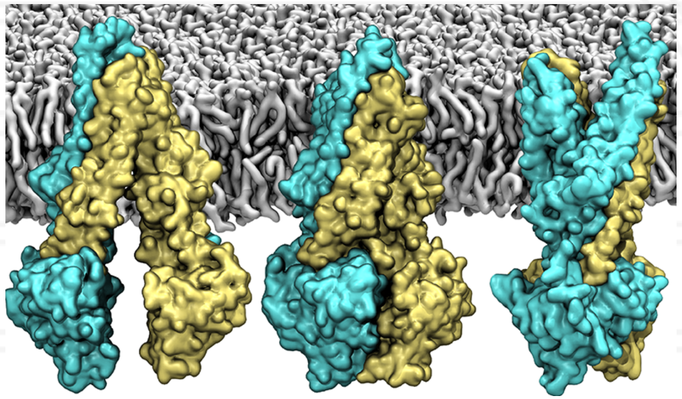
Drug discovery



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

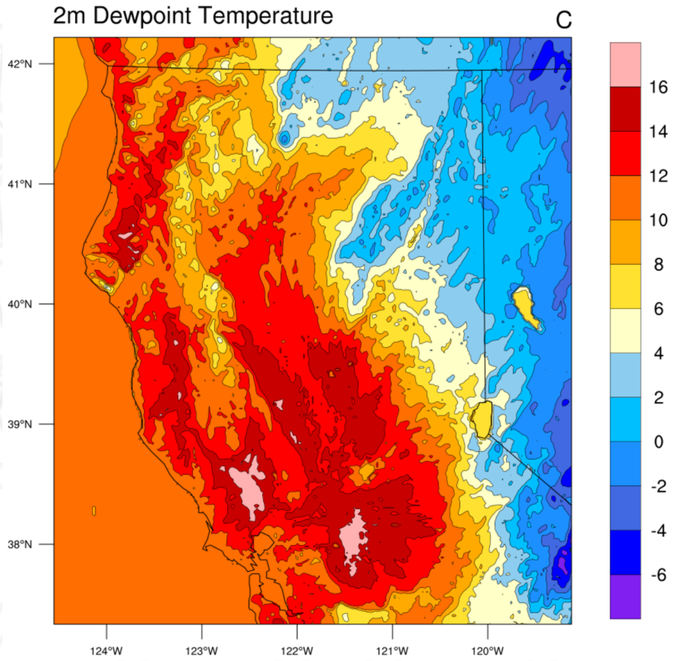
The need for high performance computing

Drug discovery



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

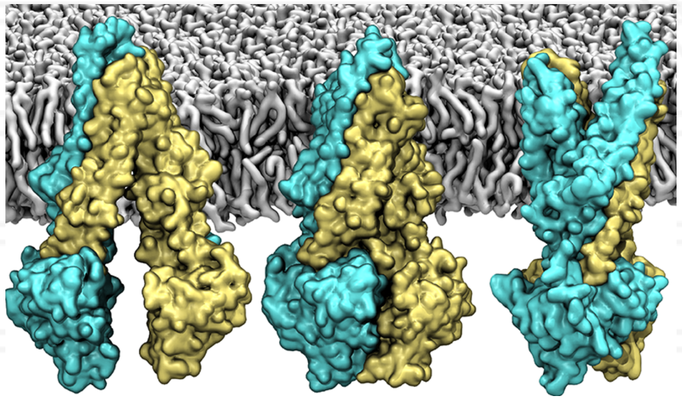
Weather forecasting



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

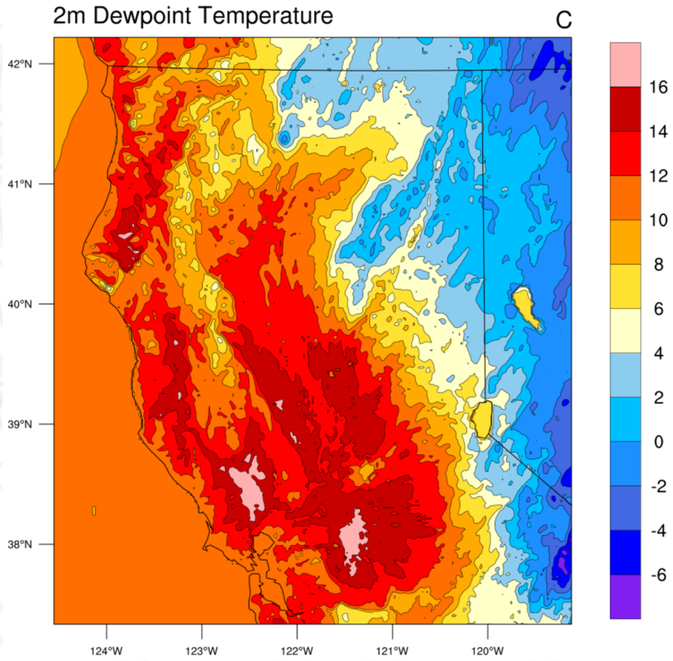
The need for high performance computing

Drug discovery



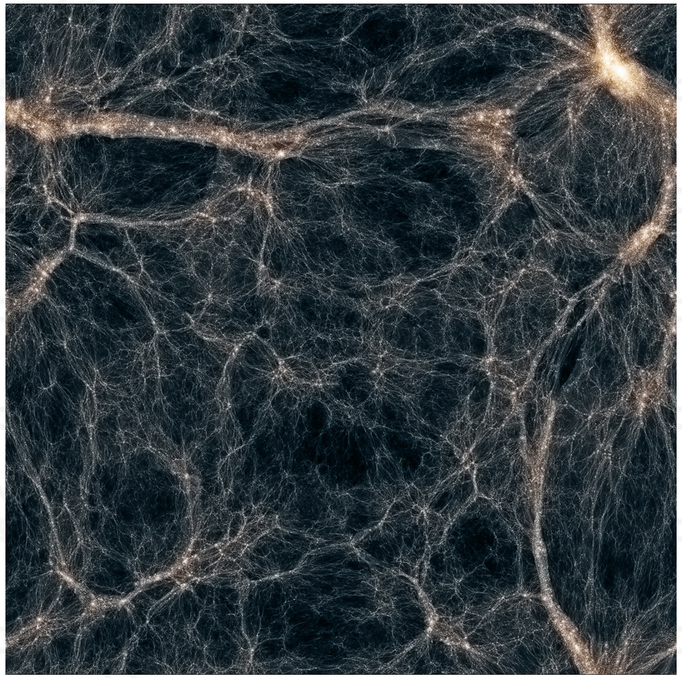
https://www.nature.com/articles/nature21414

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

Terms and Definitions

What is parallel computing?

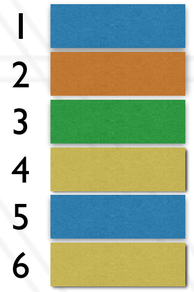
- Does it include:
  - Grid computing
  - Distributed computing
  - Cloud computing
- Does it include:
  - Superscalar processors
  - Vector processors
  - Accelerators (GPUs, FPGAs)

Job (batch) scheduling

Job (batch) scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a "job" scheduler

Job Queue

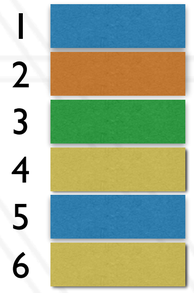


#Nodes Requested	Time Requested
128	30 mins
64	24 hours
56	6 hours
192	12 hours
…	…
…	…

Job (batch) scheduling

- HPC 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

Job Queue



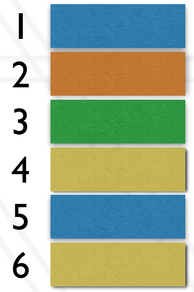
#Nodes Requested	Time Requested
128	30 mins
64	24 hours
56	6 hours
192	12 hours
…	…
…	…

Job (batch) scheduling

- HPC 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
- Network, filesystem: shared by all jobs

Job Queue



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

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 it's performance improves when using more resources

Weak versus strong scaling

- Strong scaling: *Fixed total* problem size as we run on more processes
- Weak scaling: Fixed problem size per process but *increasing total* problem size as we run on more processes

Speedup and efficiency

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

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

- Efficiency: Speedup per process

$$\text{Efficiency} = \frac{t_1}{t_n \times n}$$

Amdahl's law

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

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

Supercomputers vs. commodity clusters

- Typically, supercomputer refers to customized hardware
  - IBM Blue Gene, Cray XT, Cray XC
- Cluster refers to a parallel machine put together using off-the-shelf hardware

Communication and synchronization

- Each physical node might compute independently for a while
- When data is needed from other (remote) nodes, 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 together

Parallel Programming

Different models of parallel computation

- SIMD: Single Instruction Multiple Data
- MIMD: Multiple Instruction Multiple Data
- SPMD: Single Program Multiple Data
  - Typical in HPC

Writing parallel programs

- Decide the algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality
- Computation: how to divide work among threads/processes?

Writing parallel programs: examples

- Molecular Dynamics
- N-body Simulations

Load balance and grain size

- Load balance: try to balance the amount of work (computation) assigned to different threads/ processes
- Grain size: ratio of computation-to-communication
  - Coarse-grained vs. fine-grained

System software: Programming models

- Shared memory/ address-space
  - Explicit: Pthreads
  - Implicit: OpenMP
- Distributed memory
  - Explicit: MPI
  - Implicit: Task-based models (Charm++)

User code

Parallel runtime

Communication library

Operating system

Writing OpenMP programs

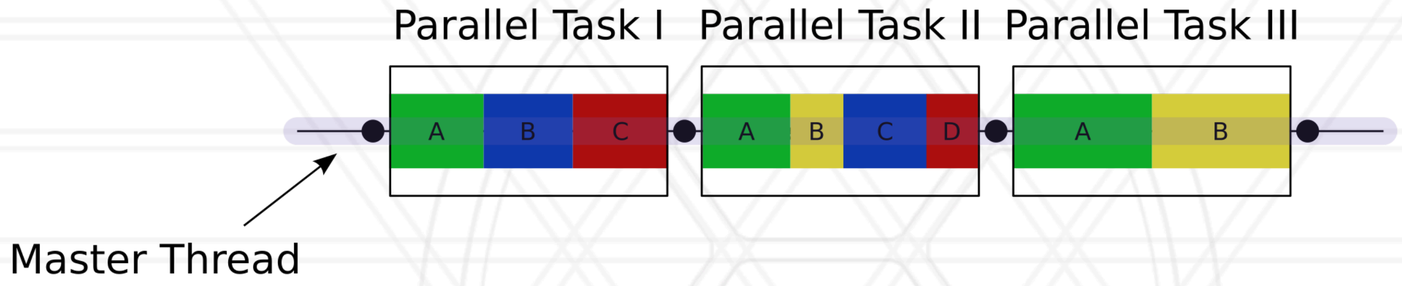
Shared memory programming & OpenMP

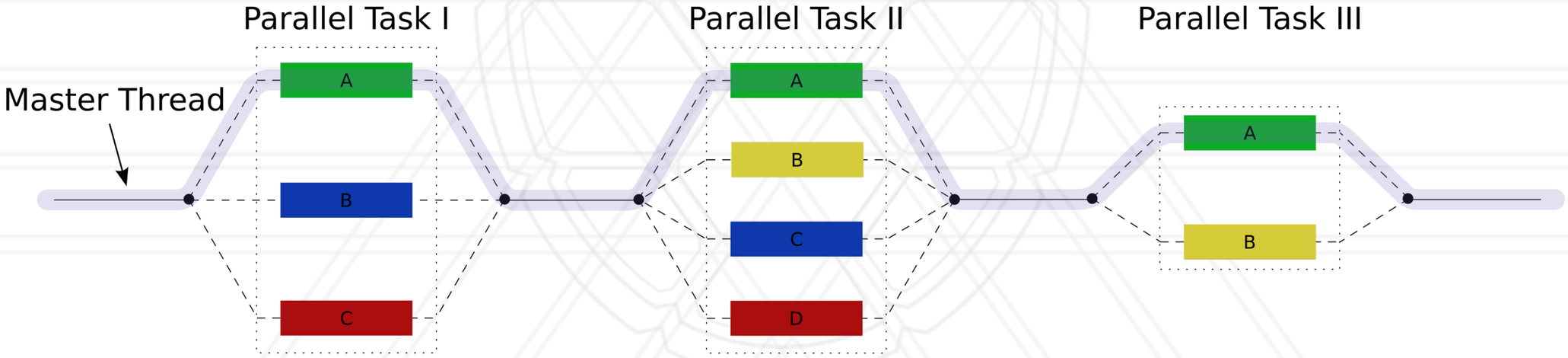
- OpenMP is a language extension that enables parallelizing C/C++/Fortran code via compiler directives and library routines
  - Compiler converts code to multi-threaded code
- Meant for certain kinds of programs/computational kernels
  - Parallelism can be specified for regions and loops
- Fork/join model of parallelism

OpenMP

- Support for on-node parallelization
- Directives for parallel loops, regions, functions
- Cannot be used for multi-node jobs

Fork-join parallelism





https://en.wikipedia.org/wiki/OpenMP

Hello World in OpenMP

```
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

Compiling: `gcc -fopenmp hello.c -o hello`

```
export OMP_NUM_THREADS=2
```

Parallel loop in OpenMP

```
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

Parallel region in OpenMP

```
int main(int argc, char **argv)
{
    double a[1000];
    omp_set_num_threads(4);

    #pragma omp parallel
    {
        int id = omp_thread_num();
        foo(id, a);
    }
    printf("all done \n");

    return 0;
}
```

Pragma

- Pragma: a compiler directive in C or C++
- Mechanism to communicate with the compiler
- Compiler may ignore pragmas

```
#pragma omp ...
```

Shared and private variables

- Shared variable: All threads have the same address for a variable
- Private variable: Each thread has a different address for a variable
- A thread cannot access the private variables of another thread

OpenMP functions

- `void omp_set_num_threads(int num_threads)`
  - Set the number of OpenMP threads to be used in parallel regions
- `int omp_get_num_procs(void);`
  - Returns the number of available processors

private clause

- Optional component of a pragma
- Direct compiler to make variables private

```
#pragma omp parallel for private(j)
for (i = 0; i < BLOCK_SIZE(id,p,n); i++)
   for (j = 0; j < n; j++)
      a[i][j] = MIN(a[i][j],a[i][k]+tmp);
```

firstprivate, lastprivate clause

- firstprivate: variable gets initial value identical to the variable controlled by the master thread as the loop is entered
- lastprivate: value copied from the last sequentially executed iteration

Questions?

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu