Introduction to High Performance Computing

\\

SDS406 – Fall semester, 2024 - 2025

\\

L01: Introduction, 7^{th} October 2024

SDS406

`SDS406`: Introduction to high performance computing

- Day and time?
- Interleaved lecture and hands-on $\Rightarrow$ please always be with your laptops
- MSc mandatory course
- Elective for PhD students (CyI and UCY)

Assessment

- Coursework
  - Three homework assignments (35%)
  - In-class participation in labs and small homework exercises (15%)
- Final examination (50%)
  - Final assignment

Preliminaries

- Laptop computers for exercises
  - To log in to educational cluster
  - Need an `ssh` client
  - Need to learn to use a text editor on a remote system
    - e.g., VS Code, Emacs, Vim
  - Go through lesson slides sds406.online
- "Lab" format
  - Each week's lesson will include taught components and practical exercises
  - For the exercises we will use C (or C++) and Python:
    - C for performance-targeted exercises
    - Python for post-processing (analysis and visualization of results)
- Access to the educational cluster will be provided with dedicated course accounts
  - Accounts prepared for you before this lesson
  - You will use these accounts during the in-class exercises, homeworks, and assignments
- I assume that:
  - this **is not** your first *programming* course
  - this **is** your first *parallel programming* course

This Lesson

Preliminaries on parallel computing

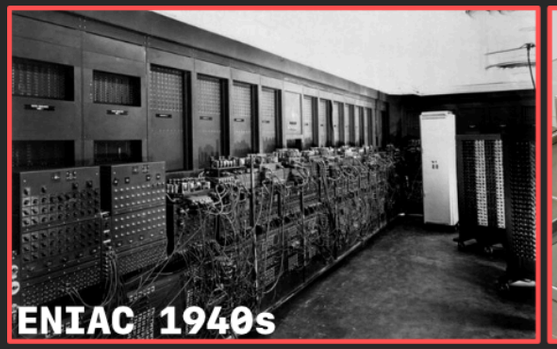
- Terminologies and definitions
- Overview of high performance computing landscape

Preparation for labs

- Log into educational system
- Editing source code files remotely

High-performance computing

"The use of supercomputers to solve complex computational tasks"



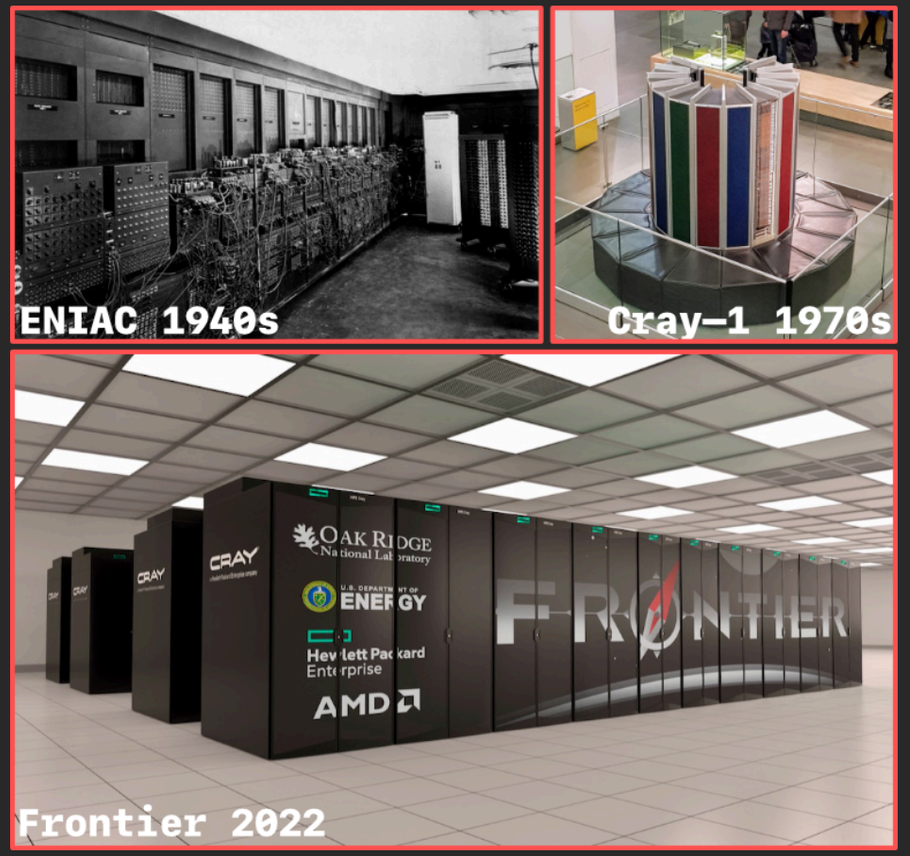
High-performance computing

"The use of supercomputers to solve complex computational tasks"

(Super)Computing evolution

1940s — First computers, e.g. ENIAC

- are effectively supercomputers — expensive to buy and operate; large footprint



High-performance computing

"The use of supercomputers to solve complex computational tasks"

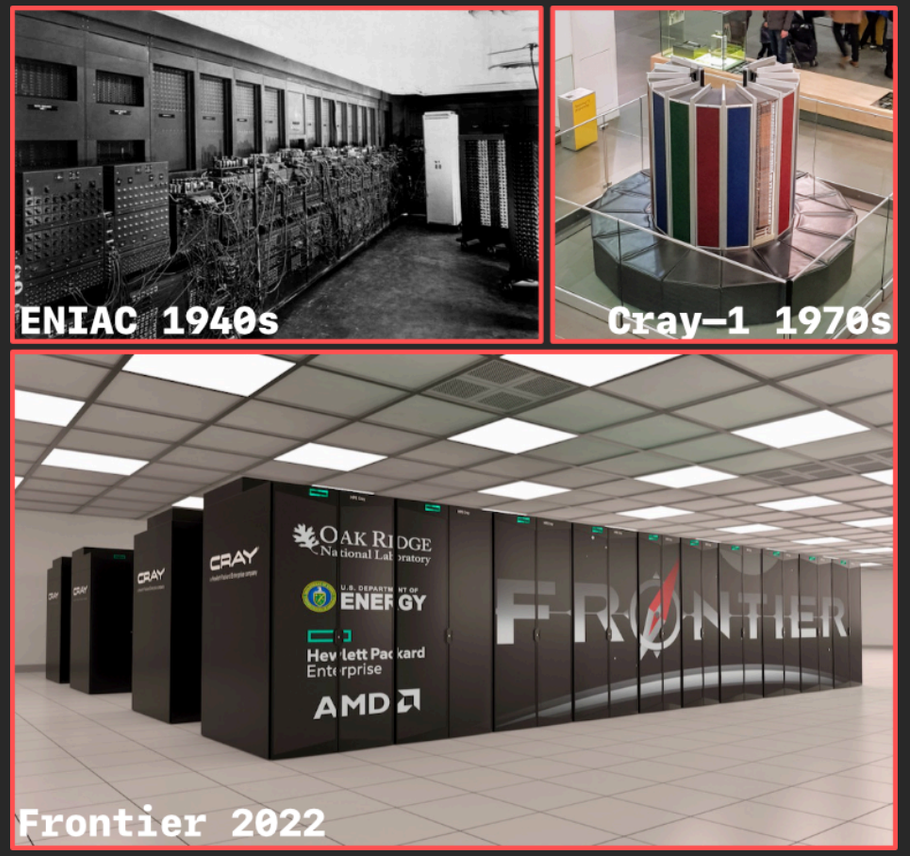
(Super)Computing evolution

1940s — First computers, e.g. ENIAC

- are effectively supercomputers — expensive to buy and operate; large footprint

1970s, 80s — Dawn of personal computing, but supercomputers needed for specialized tasks, e.g. Cray-1

- parallelism emerges: vectorization, parallel instructions



High-performance computing

"The use of supercomputers to solve complex computational tasks"

(Super)Computing evolution

1940s — First computers, e.g. ENIAC

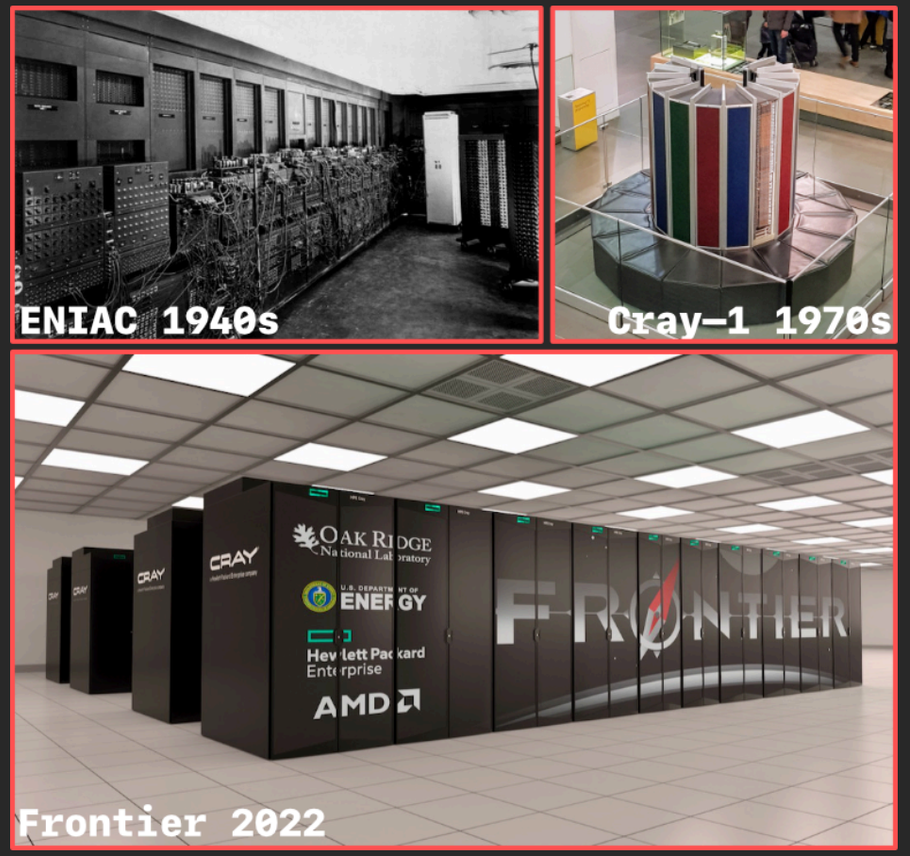
- are effectively supercomputers — expensive to buy and operate; large footprint

1970s, 80s — Dawn of personal computing, but supercomputers needed for specialized tasks, e.g. Cray-1

- parallelism emerges: vectorization, parallel instructions

1990s, 2000s — Supercomputers via integration of commodity components (e.g. Beowulf clusters)

- Parallelism as we understand it today: distributed and shared memory, message passing, etc.



High-performance computing

"The use of supercomputers to solve complex computational tasks"

(Super)Computing evolution

1940s — First computers, e.g. ENIAC

- are effectively supercomputers — expensive to buy and operate; large footprint

1970s, 80s — Dawn of personal computing, but supercomputers needed for specialized tasks, e.g. Cray-1

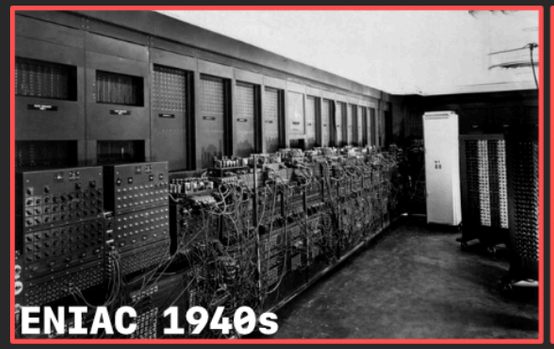
- parallelism emerges: vectorization, parallel instructions

1990s, 2000s — Supercomputers via integration of commodity components (e.g. Beowulf clusters)

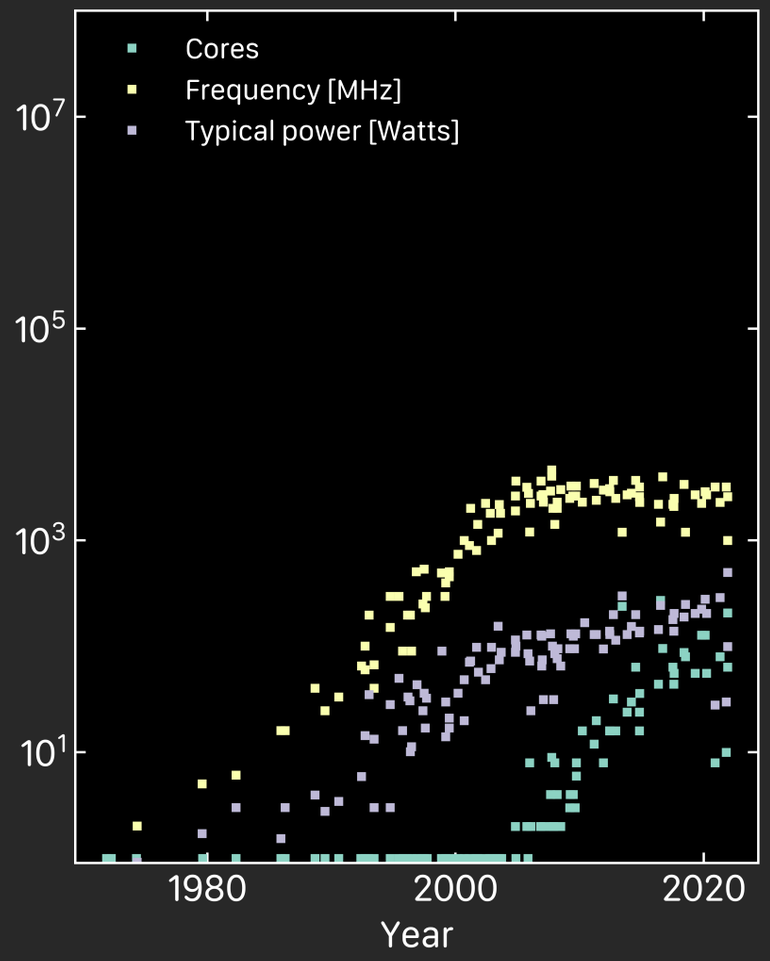
- Parallelism as we understand it today: distributed and shared memory, message passing, etc.

2010s onwards — Heterogeneous supercomputers

- Many sockets per node; co-processors, e.g. GPUs, potentially multiple architectures within same system



Supercomputing means Parallel Computing

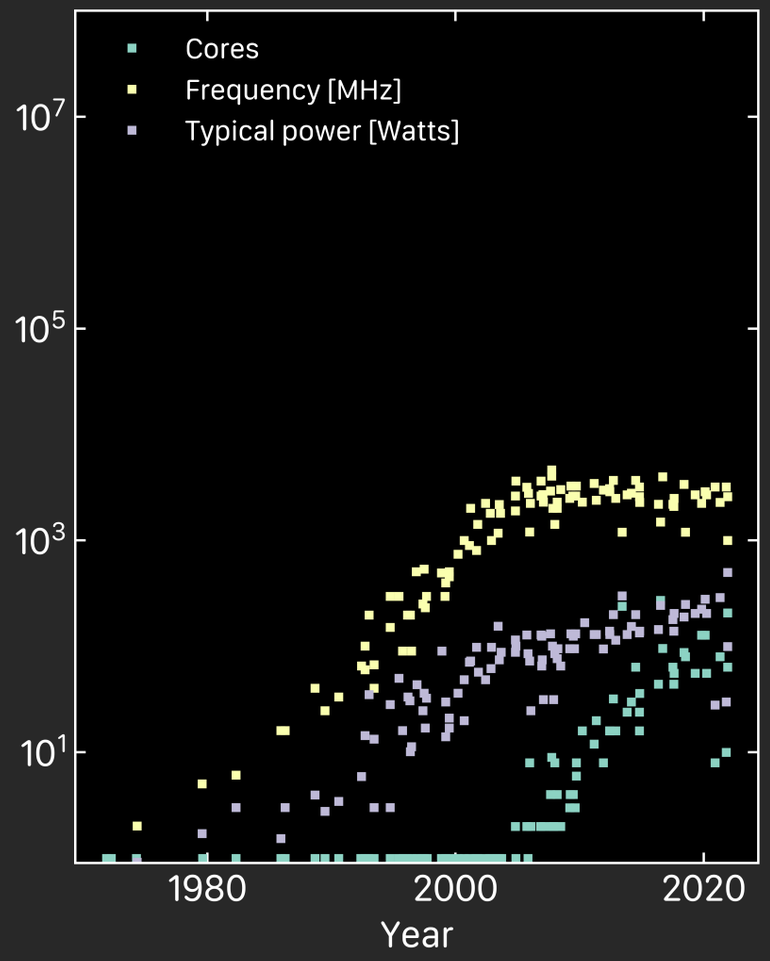


https://github.com/karlrupp/microprocessor-trend-data

No *Dennard scaling* after ~2005

- $\mathrm{P} \propto A f V^2$ (P: power, A: area, f: freq., V: voltage)
- Roughly, as transistors get smaller, the *power density* stays constant
- This efficiency was typically used towards increasing frequency

Supercomputing means Parallel Computing



https://github.com/karlrupp/microprocessor-trend-data

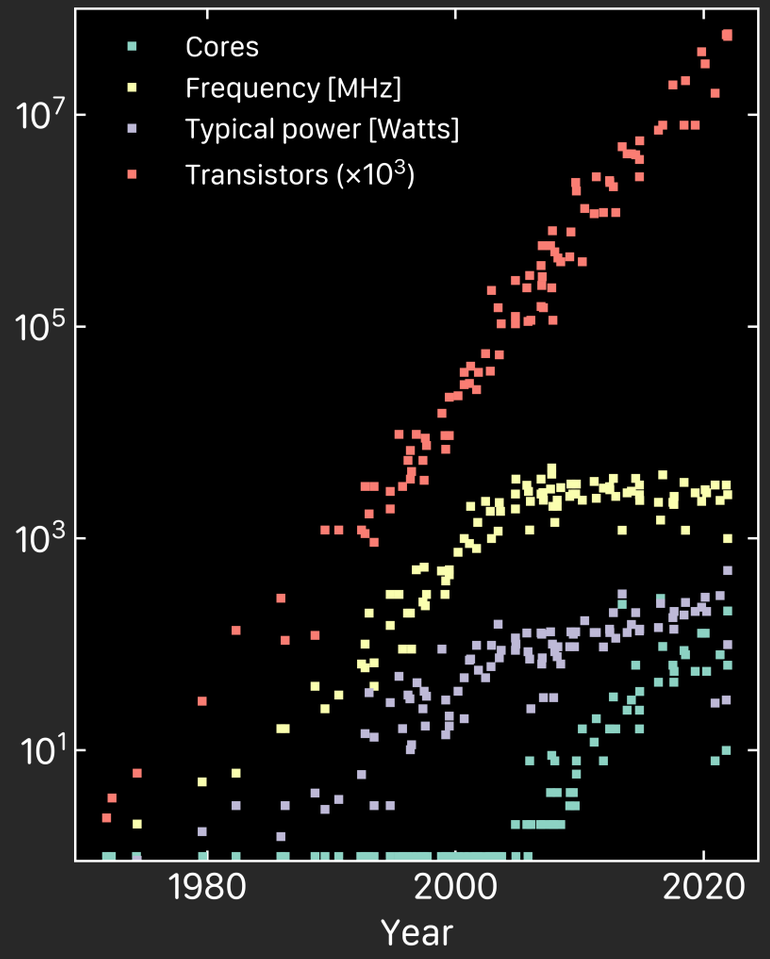
No *Dennard scaling* after ~2005

- $\mathrm{P} \propto A f V^2$ (P: power, A: area, f: freq., V: voltage)
- Roughly, as transistors get smaller, the *power density* stays constant
- This efficiency was typically used towards increasing frequency

What about Moore's law?

- Moore's law is a statement about *transistor count*

Supercomputing means Parallel Computing



https://github.com/karlrupp/microprocessor-trend-data

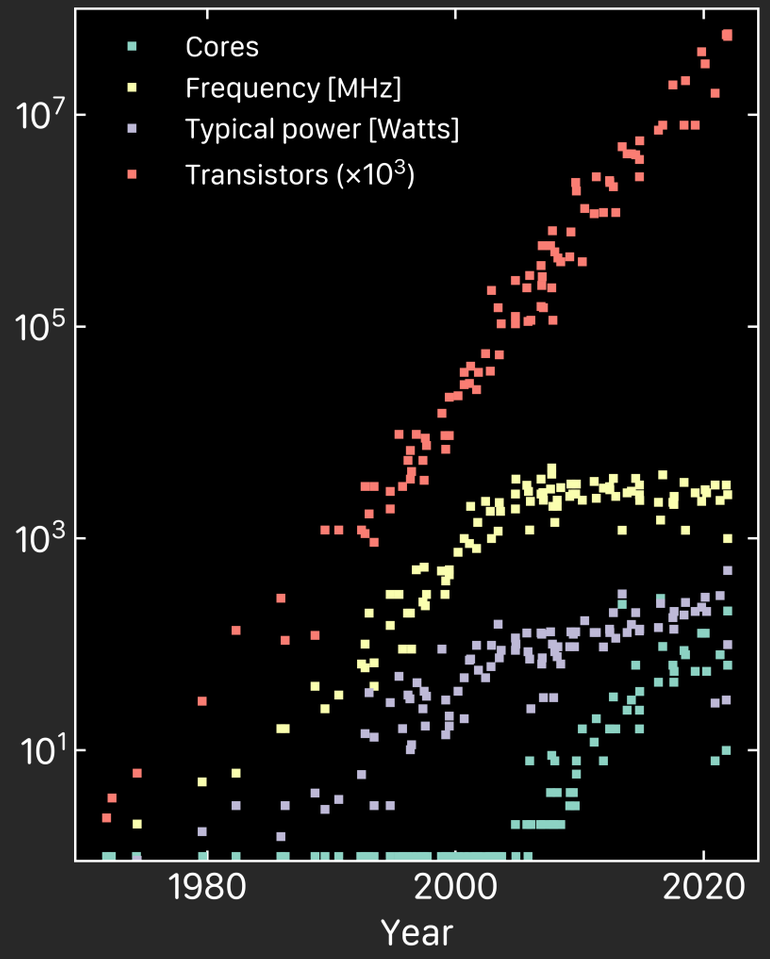
No *Dennard scaling* after ~2005

- $\mathrm{P} \propto \mathrm{A} \mathrm{f} \mathrm{V}^2$ (P: power, A: area, f: freq., V: voltage)
- Roughly, as transistors get smaller, the *power density* stays constant
- This efficiency was typically used towards increasing frequency

What about Moore's law?

- Moore's law is a statement about *transistor count*
- And so far seems to be holding strong

Supercomputing means Parallel Computing



https://github.com/karlrupp/microprocessor-trend-data

No *Dennard scaling* after ~2005

- $\mathrm{P} \propto A f V^2$ (P: power, A: area, f: freq., V: voltage)
- Roughly, as transistors get smaller, the *power density* stays constant
- This efficiency was typically used towards increasing frequency

What about Moore's law?

- Moore's law is a statement about *transistor count*
- And so far seems to be holding strong

$\Rightarrow$ Efficiencies are now used to increase parallelism rather than frequency

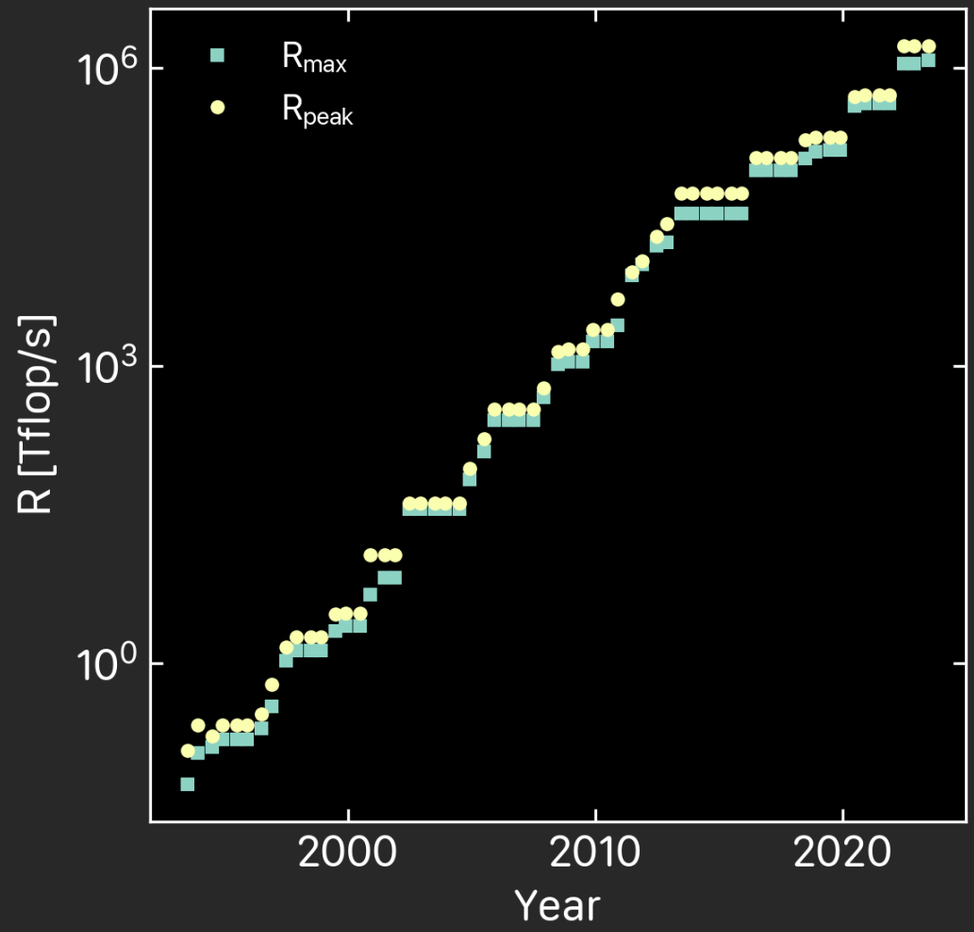
- HPC now means more parallel hardware, rather than faster scalar hardware
- HPC primarily deals with tackling the challenges of parallelism, from all aspects (hardware, software, algorithmic, etc.)

Evolution of Supercomputing

The "Top500" list

- Ranked list of top 500 supercomputers populated twice per year
- Data shown here since 1993
- Shown is High-Performance Linpack (HPL) performance achieved

⇒ Shows that exponential increase has been maintained



https://www.top500.org

Evolution of Supercomputing

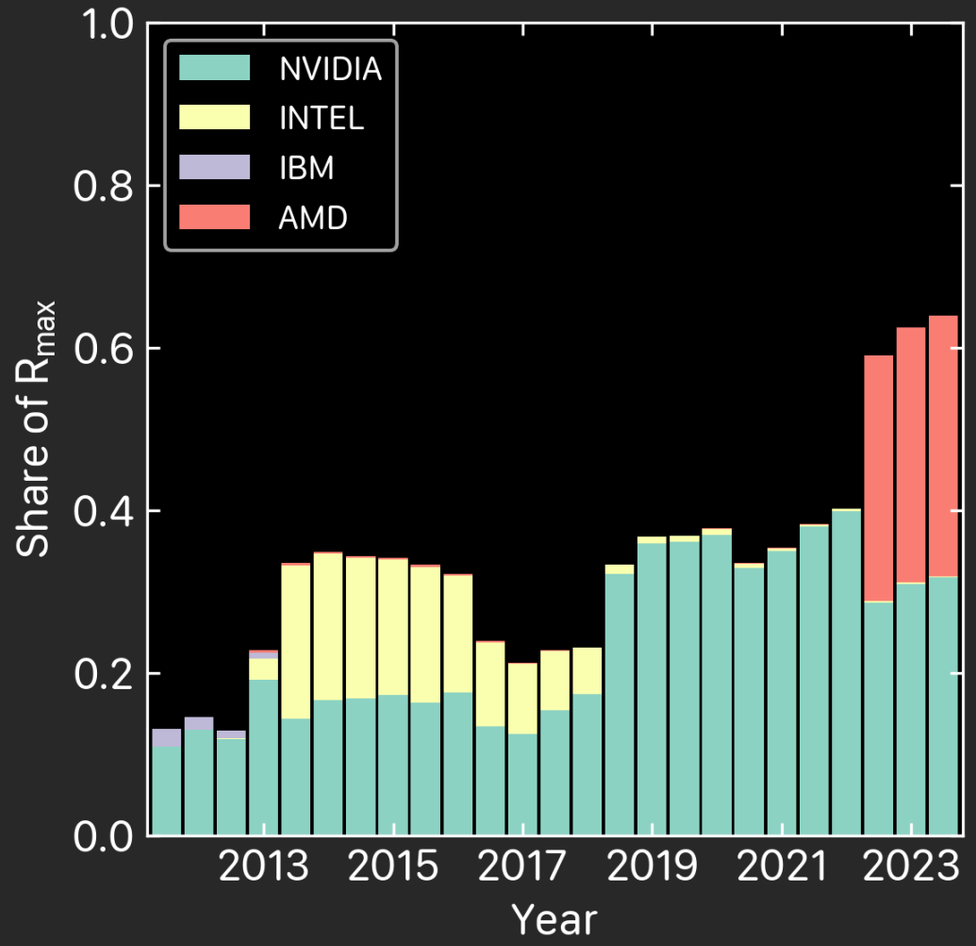
The "Top500" list

- Ranked list of top 500 supercomputers populated twice per year
- Data shown here since 1993
- Shown is High-Performance Linpack (HPL) performance achieved

⇒ Shows that exponential increase has been maintained

- Proliferation of accelerators
  - Latest AMD accelerators are Instinct GPUs
  - Intel between 2012 and 2017 includes Xeon Phi
  - IBM accelerators refer to IBM Cell

⇒ Since 2021, more than half of performance over Top500 now from accelerated systems



https://www.top500.org

Parallel computing as the main paradigm

High performance computing means parallel computing

- Technology trends mean parallelism is essential for advanced computing

Parallel computing as the main paradigm

High performance computing means parallel computing

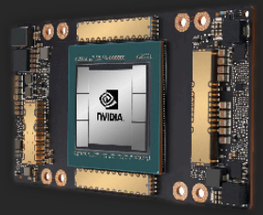
- Technology trends mean parallelism is essential for advanced computing
- Practitioners of computational science and engineering benefit from knowledge of concepts and challenges of parallel computing

Parallel computing as the main paradigm

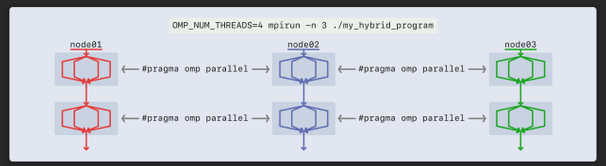
High performance computing means parallel computing

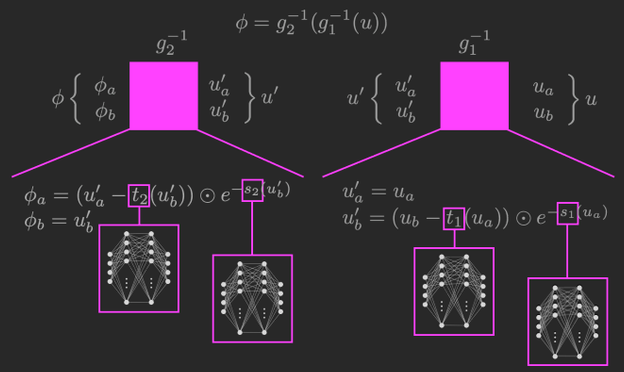
- Technology trends mean parallelism is essential for advanced computing
- Practitioners of computational science and engineering benefit from knowledge of concepts and challenges of parallel computing
  - Architectures and their characteristics
  - Algorithms and how amenable they are to parallelisation
  - Performance metrics and their significance, e.g. sustained and peak floating point performance, bandwidth, scalability

Architectures

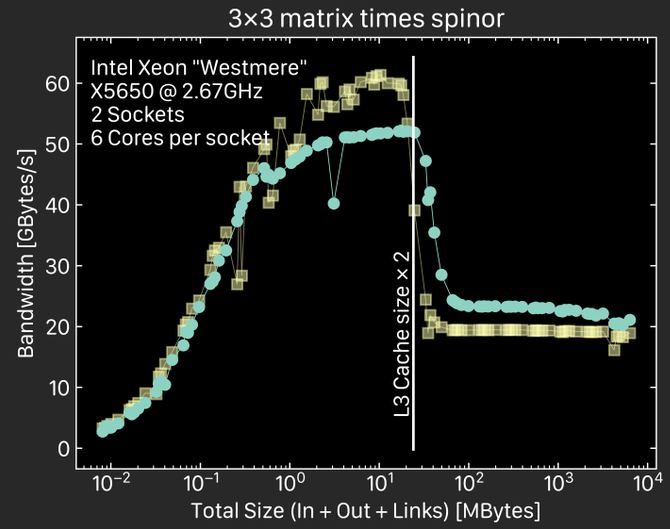


Algorithms





Performance metrics



Characterization of Parallel Workloads

Task vs Data Parallelism

Taxonomy of computer architectures, *Flynn's Taxonomy*:

- Single Instruction stream, Single Data stream (SISD)
- Multiple Instruction streams, Single Data stream (MISD)
- Single Instruction stream, Multiple Data streams (SIMD)
- Multiple Instruction streams, Multiple Data streams (MIMD)

Characterization of Parallel Workloads

Task vs Data Parallelism

Taxonomy of computer architectures, *Flynn's Taxonomy*:

- Single Instruction stream, Single Data stream (SISD)
- Multiple Instruction streams, Single Data stream (MISD)
- Single Instruction stream, Multiple Data streams (SIMD)
- Multiple Instruction streams, Multiple Data streams (MIMD)

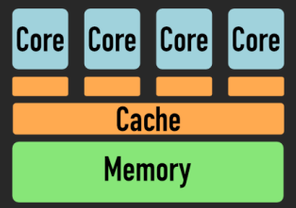
Parallel computing

- Most computing devices have underlying SIMD architecture units: GPUs, CPUs, etc.
- Most supercomputers can be considered MIMD architectures: multiple interconnected computing devices that can issue the same or different instructions on multiple data

Shared vs Distributed memory paradigm

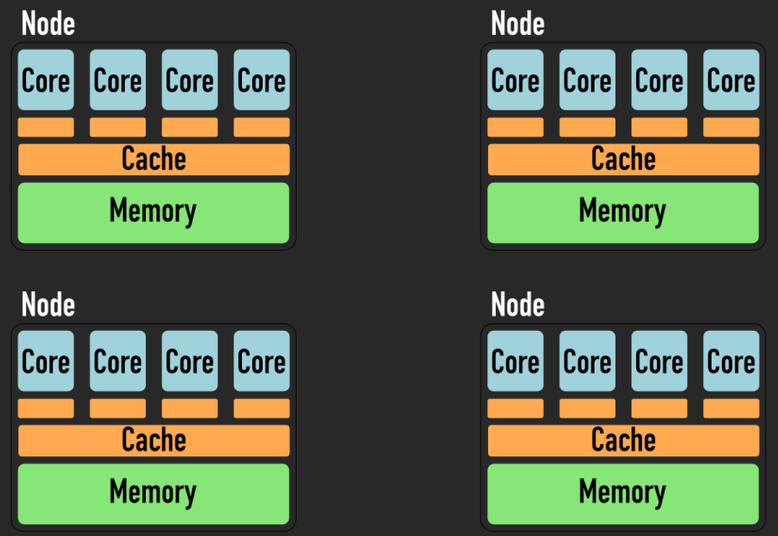
Shared memory

- Multiple processes share common memory (common *memory address space*)
- E.g. multi-core CPU, multi-socket node, GPU threads, etc.
- Programming models: OpenMP, pthreads, MPI, CUDA



Distributed memory

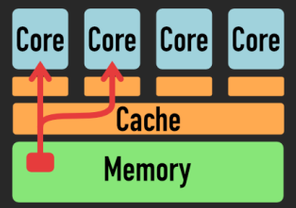
- Processes have distinct memory domains (different *memory address space*)
- E.g. multiple nodes within a cluster, multiple GPUs within a node
- Programming models: MPI



Shared vs Distributed memory paradigm

Shared memory

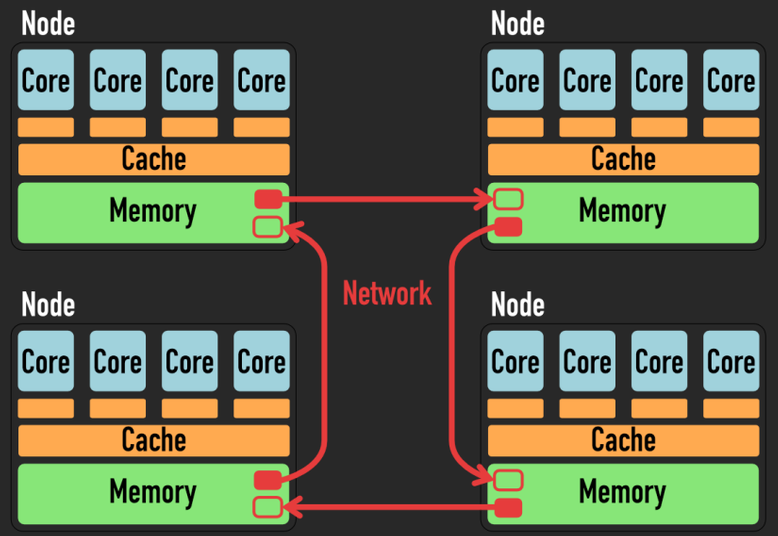
- Multiple processes share common memory (common *memory address space*)
- E.g. multi-core CPU, multi-socket node, GPU threads, etc.
- Programming models: OpenMP, pthreads, MPI, CUDA



Data shared via memory

Distributed memory

- Processes have distinct memory domains (different *memory address space*)
- E.g. multiple nodes within a cluster, multiple GPUs within a node
- Programming models: MPI



Data shared via explicit communication over a network

Capacity vs Capability

How do we "spend" parallelism?

- *Capacity computing*
  - Improve time-to-solution of a problem that can also run on less number of processes
  - E.g. solve many small problems
- *Capability computing*
  - Solve a problem that was *impossible* to solve on less processes
  - E.g. solve a problem using $\mathbb{N}$ nodes, that cannot fit in memory of less nodes

Capacity vs Capability

How do we "spend" parallelism?

- *Capacity computing*
  - Improve time-to-solution of a problem that can also run on less number of processes
  - E.g. solve many small problems
- *Capability computing*
  - Solve a problem that was *impossible* to solve on less processes
  - E.g. solve a problem using $\mathbb{N}$ nodes, that cannot fit in memory of less nodes

---

High Throughput Computing sometimes used to identify capacity computing, with HPC used to mean capability computing

Capacity vs Capability

How do we "spend" parallelism?

- *Capacity computing*
  - Improve time-to-solution of a problem that can also run on less number of processes
  - E.g. solve many small problems
- *Capability computing*
  - Solve a problem that was *impossible* to solve on less processes
  - E.g. solve a problem using $\mathbb{N}$ nodes, that cannot fit in memory of less nodes

---

High Throughput Computing sometimes used to identify capacity computing, with HPC used to mean capability computing

More *"capacity-like"*

More *"capability-like"*

Defining Scalability

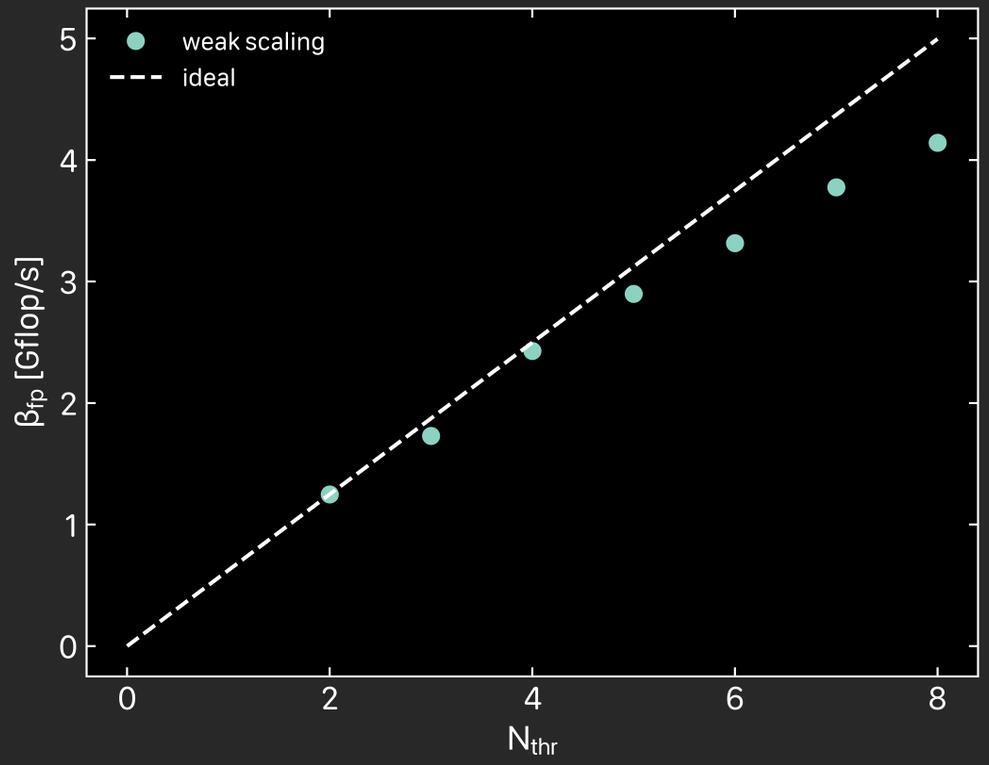
Concepts of parallel computing

- **Scalability**: The rate at which time-to-solution improves as we increase processing units
- **Weak scaling**: Increase processing units; keep the **local** problem size fixed $\Rightarrow$ for increasing global problem size
- **Strong scaling**: Increase processing units; keep the **global** problem size fixed $\Rightarrow$ for decreasing local problem size

Defining Scalability

Concepts of parallel computing

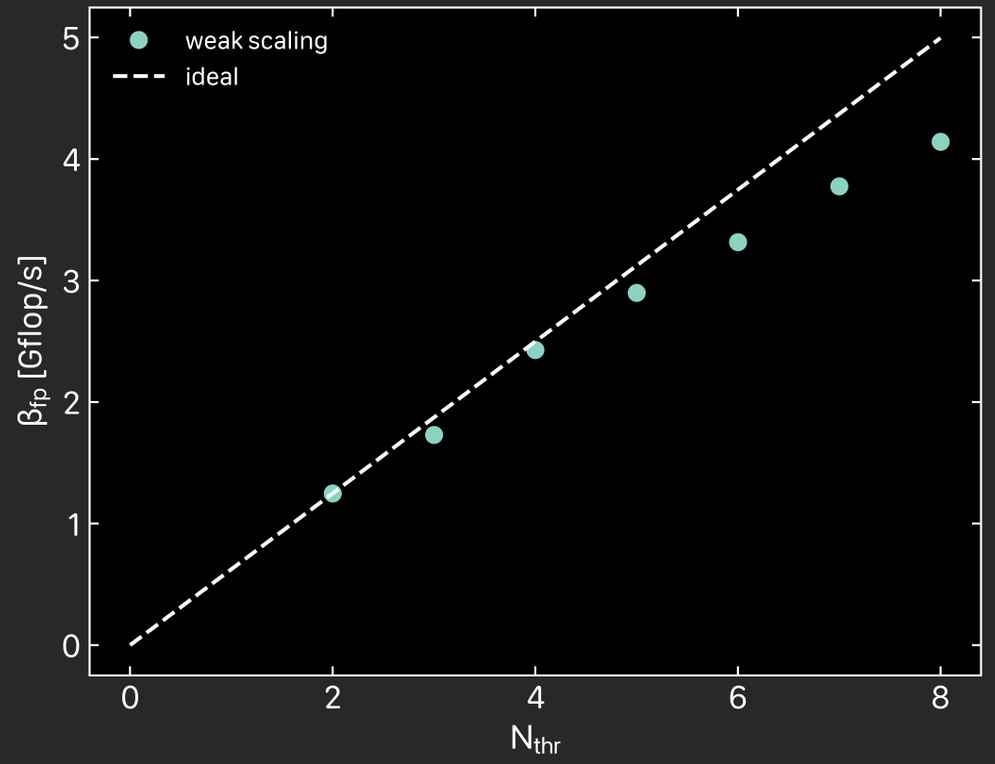
- **Scalability**: The rate at which time-to-solution improves as we increase processing units
- **Weak scaling**: Increase processing units; keep the **local** problem size fixed $\Rightarrow$ for increasing global problem size
- **Strong scaling**: Increase processing units; keep the **global** problem size fixed $\Rightarrow$ for decreasing local problem size

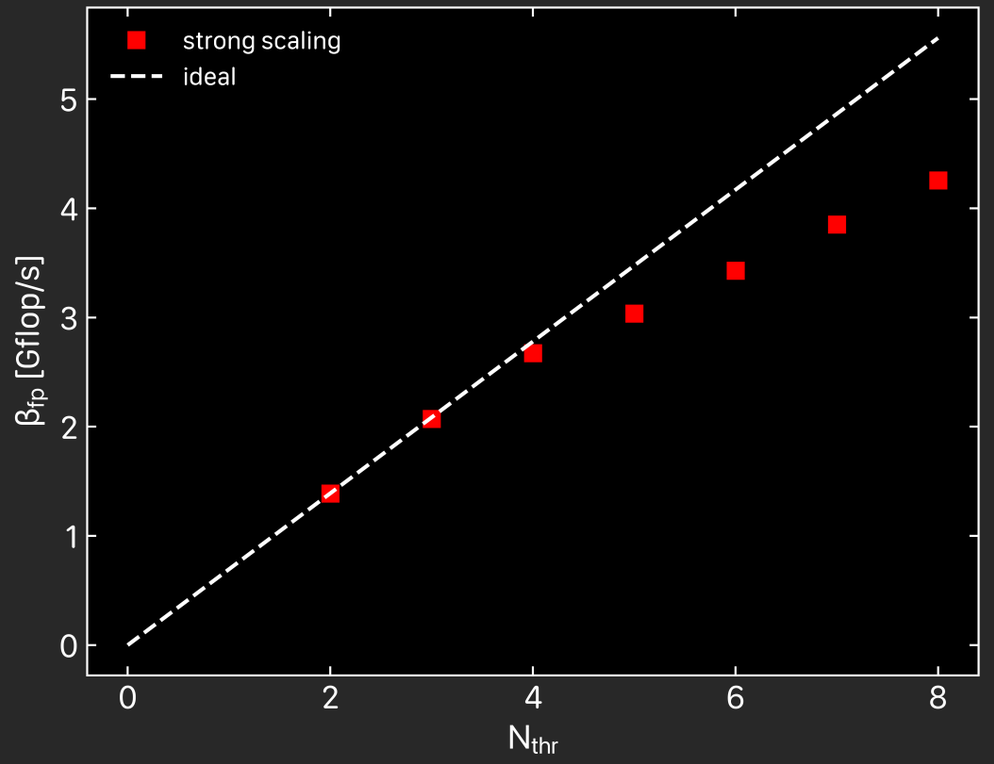


Defining Scalability

Concepts of parallel computing

- **Scalability**: The rate at which time-to-solution improves as we increase processing units
- **Weak scaling**: Increase processing units; keep the **local** problem size fixed $\Rightarrow$ for increasing global problem size
- **Strong scaling**: Increase processing units; keep the **global** problem size fixed $\Rightarrow$ for decreasing local problem size





Quantifying Scalability

Scalability

- Speedup S when using N processes

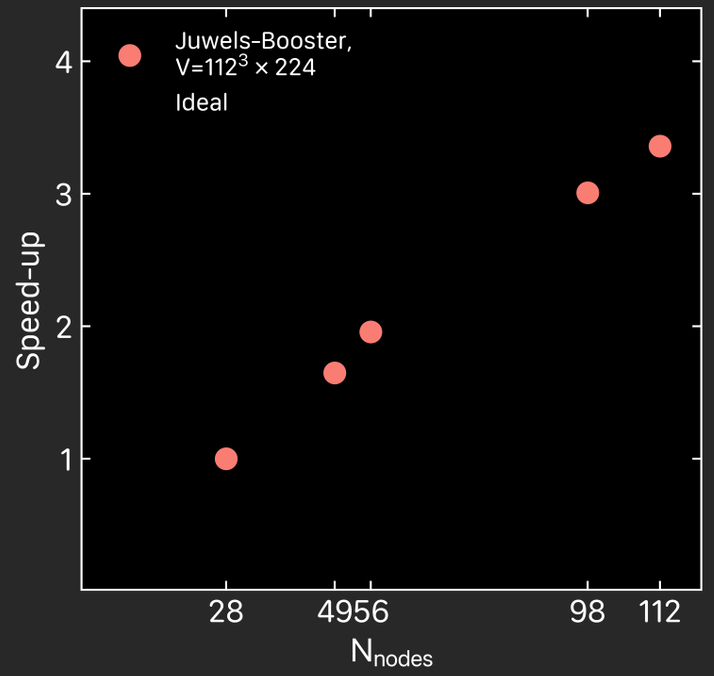
$$S(N) = \frac{T_0}{T(N)},$$

  - T_0: Reference time-to-solution (using N_0 processors, nodes, GPUs, etc.)
  - $T(N)$: Time-to-solution using $N > N_0$ processes

- Parallel efficiency ϵ

$$\epsilon = S(N)\frac{N_0}{N}$$

- Ideal scaling: $\epsilon \simeq 1$



N	T [hrs]	S	ϵ
28	8.02	–	–
49	4.87	1.65	0.94
56	4.10	1.96	0.98
98	2.67	3.01	0.86
112	2.39	3.36	0.84

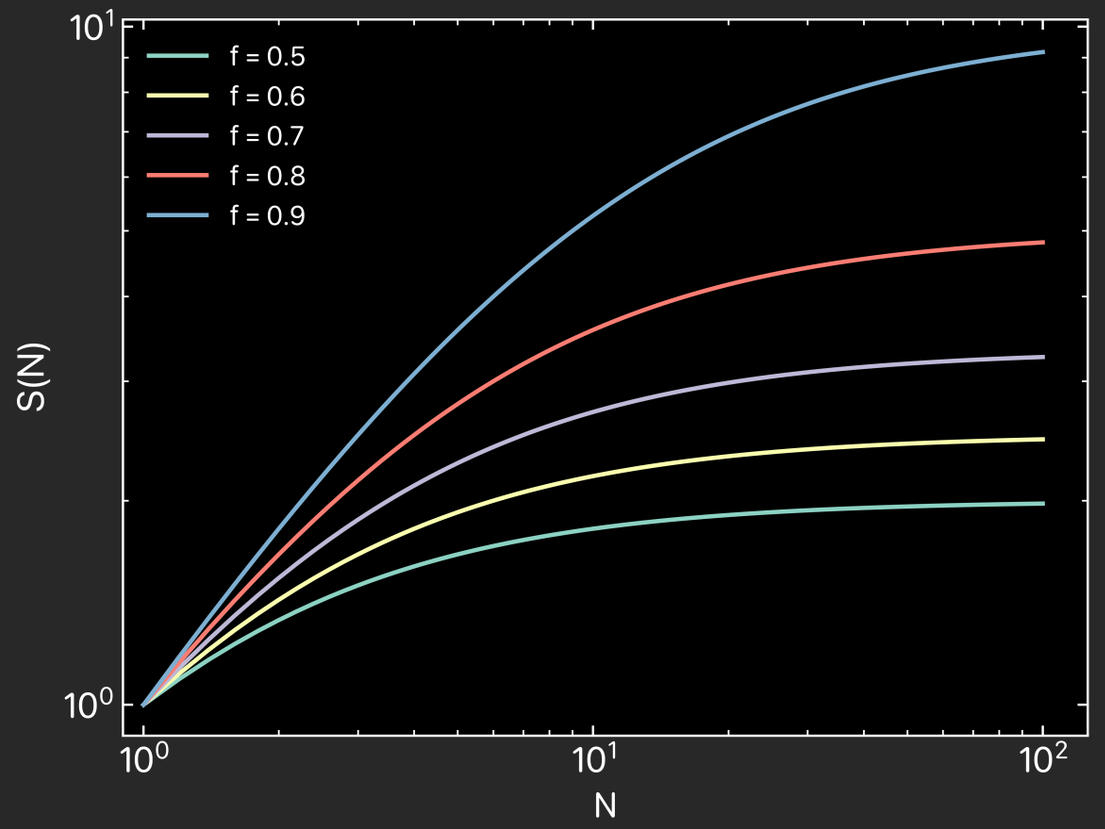
Modeling Scalability

Amdahl's Law

- f: fraction of application that can be parallelized
- T_0: time-to-solution of code when using one process
- N: Number of processes

$$T(N) = (1 - f)T_0 + f\frac{T_0}{N}$$

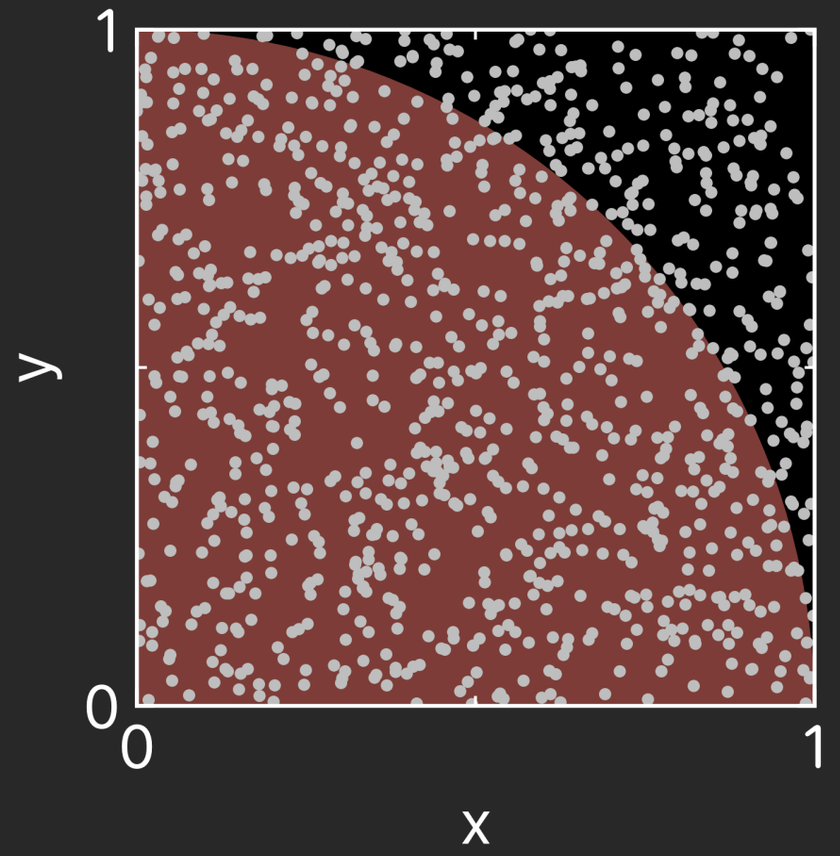
$$S(N) = \frac{T_0}{T(N)} = \frac{1}{1 - f + \frac{f}{N}}$$



Amdahl's Law

A practical example

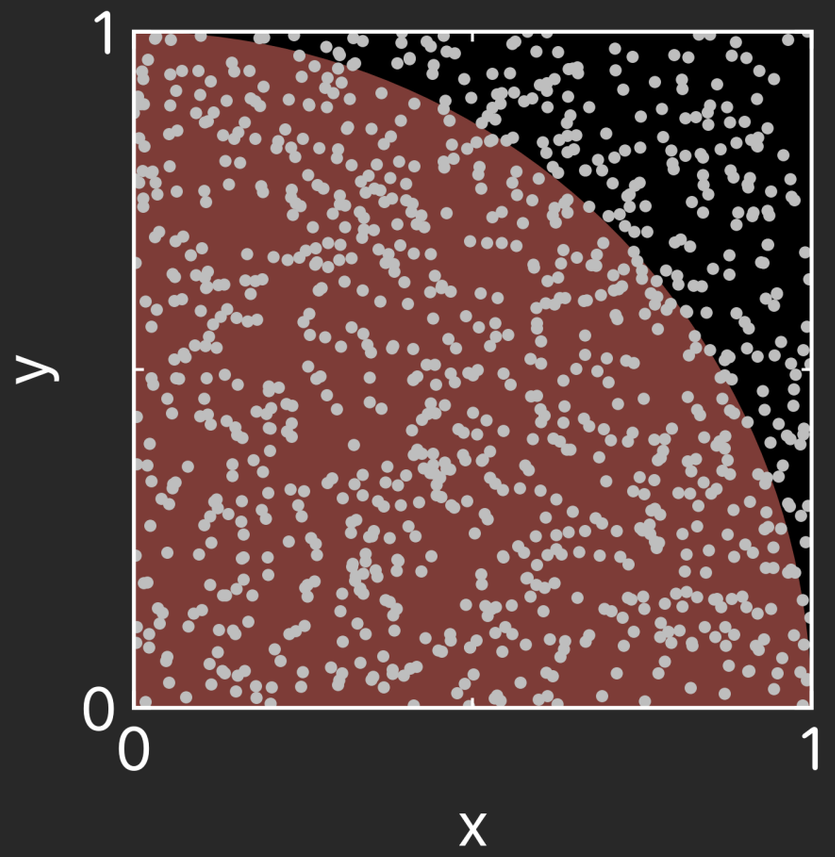
- Consider the calculation of π via simple Monte Carlo:
  - Define a unit square. Set $n_{hit} = 0$
  - Randomly pick points (x, y) within the unit square
  - If $x^2 + y^2 < 1, n_{hit} += 1$
  - Repeat N times
- The ratio n_{hit}/N approaches the area of a circle quadrant $\Rightarrow \frac{\pi}{4}$



Amdahl's Law

A practical example

```
unsigned long int hits = 0;
for(unsigned long int i=0; i<N; i++) {
    double x = drand48();
    double y = drand48();
    if(x*x + y*y < 1)
        hits += 1;
}
```

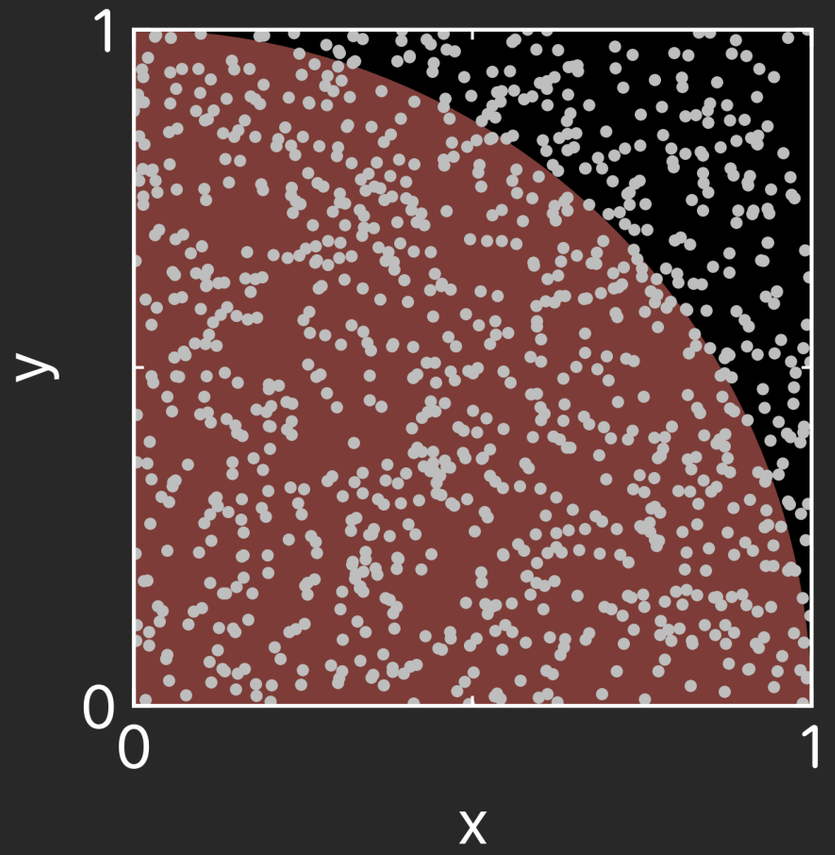


Amdahl's Law

A practical example

```
unsigned long int hits = 0;
for(unsigned long int i=0; i<N; i++) {
    double x = drand48();
    double y = drand48();
    if(x*x + y*y < 1)
        hits += 1;
}
```

- Parallelizable parts
  - The loop over `N`

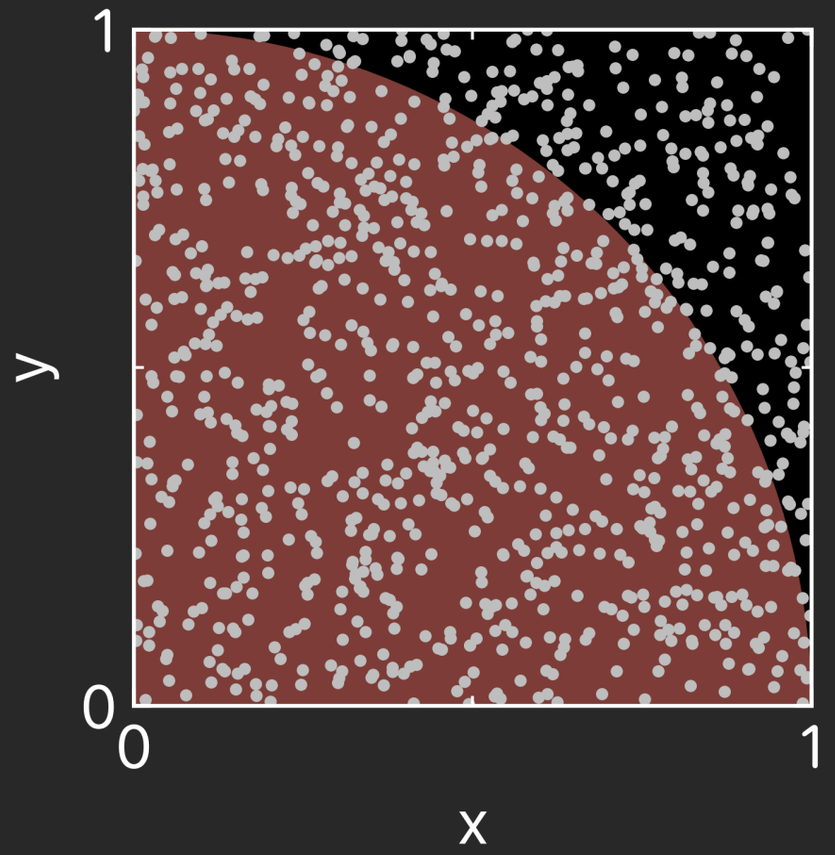


Amdahl's Law

A practical example

```
unsigned long int hits = 0;
for(unsigned long int i=0; i<N; i++) {
    double x = drand48();
    double y = drand48();
    if(x*x + y*y < 1)
        hits += 1;
}
```

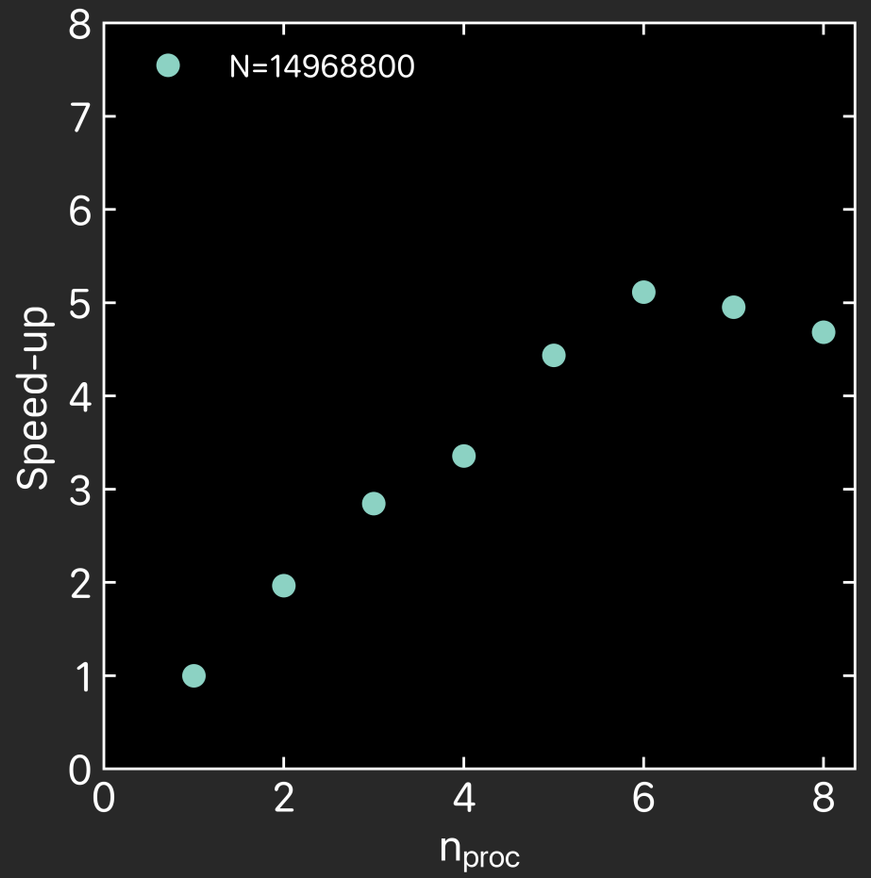
- Parallelizable parts
  - The loop over `N`
- Scalar parts
  - Initialization
  - Summing the partial `nhit` and division by `N`



Monte Carlo estimation of π

- N = 14,968,800

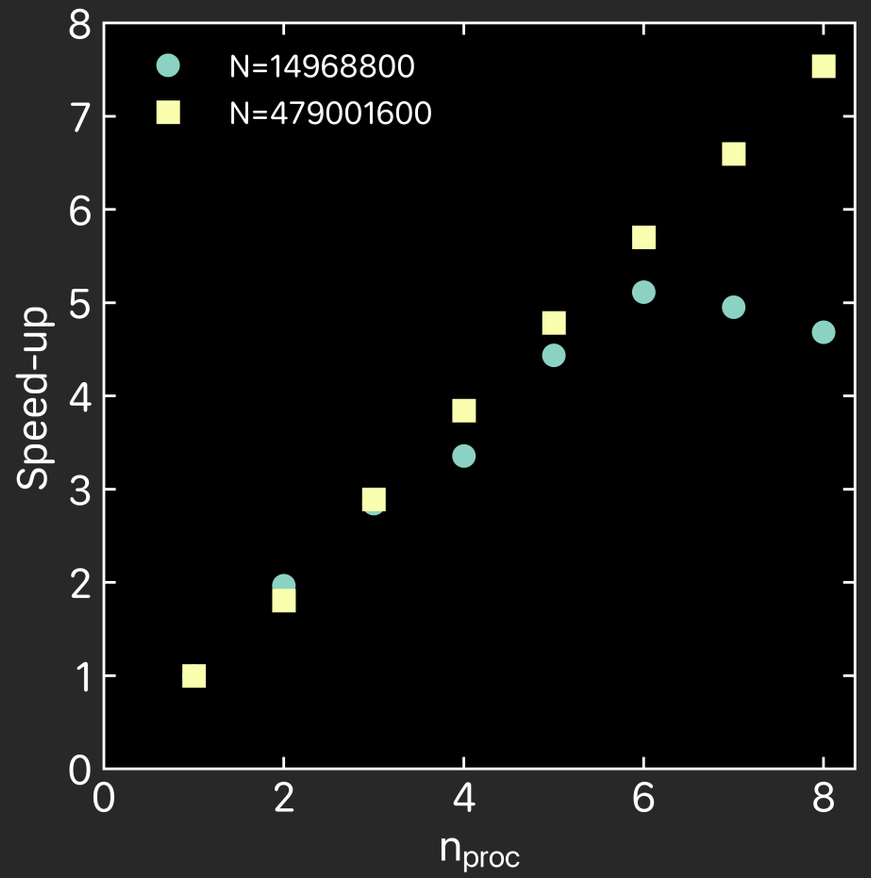
n_{proc}	t (sec)	π
1	0.53	3.141543
2	0.27	3.141355
3	0.19	3.141089
4	0.16	3.141137
5	0.12	3.141135
6	0.10	3.141290
7	0.11	3.141533
8	0.11	3.141195



Monte Carlo estimation of π

- N = 479,001,600

n_{proc}	t (sec)	π
1	13.70	3.141558
2	7.58	3.141605
3	4.74	3.141603
4	3.56	3.141647
5	2.86	3.141650
6	2.40	3.141604
7	2.08	3.141625
8	1.82	3.141646



Cluster computing

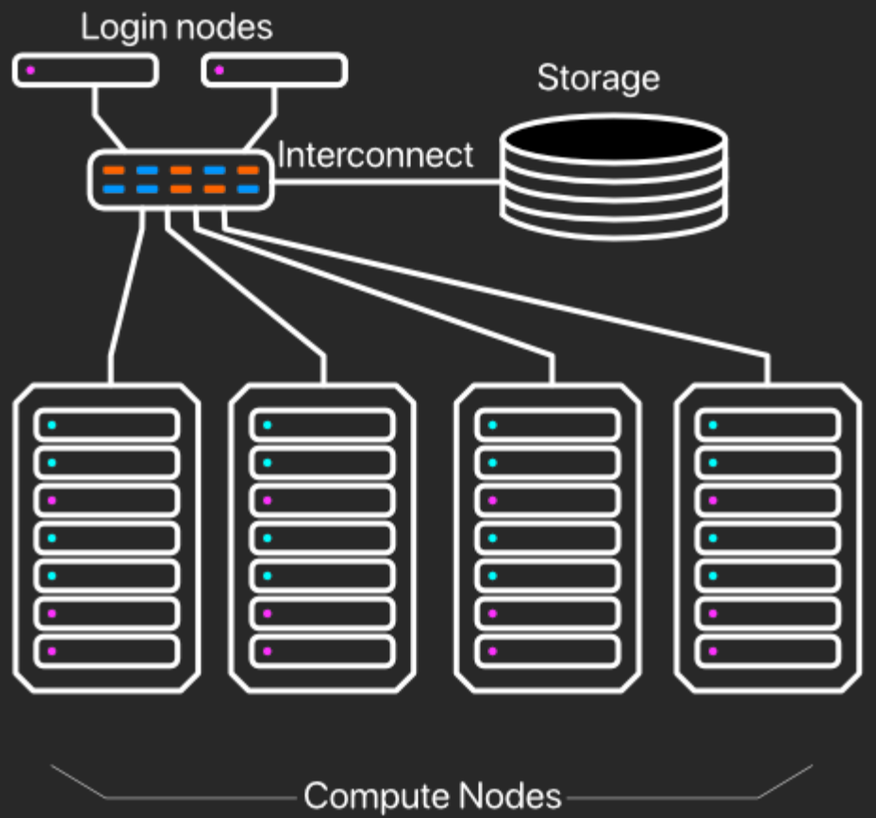
By now you should have followed instructions to:

$\longrightarrow$ Generate an ssh key-pair

$\longrightarrow$ Log into our system

Cluster Computing

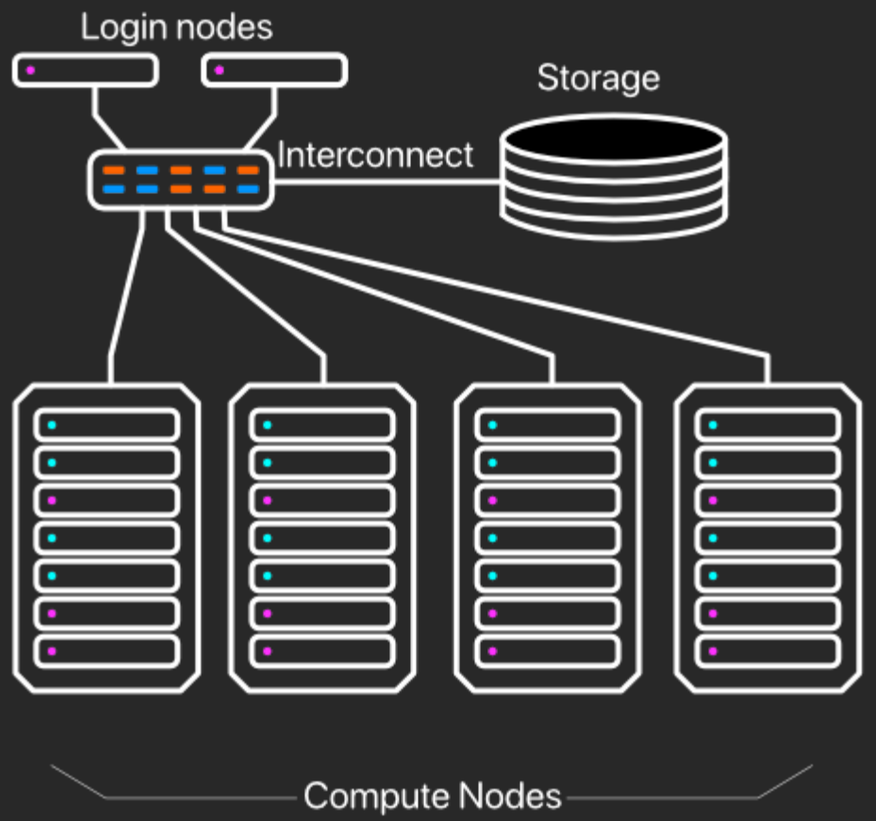
What's in a supercomputer?



Cluster Computing

What's in a supercomputer?

- Compute Nodes
  - Computational units - CPU and potentially a co-processor, e.g. a GPU
  - Memory (i.e. RAM)
  - Some storage and/or NVMe
  - Network interfaces, possibly separate between management and workload

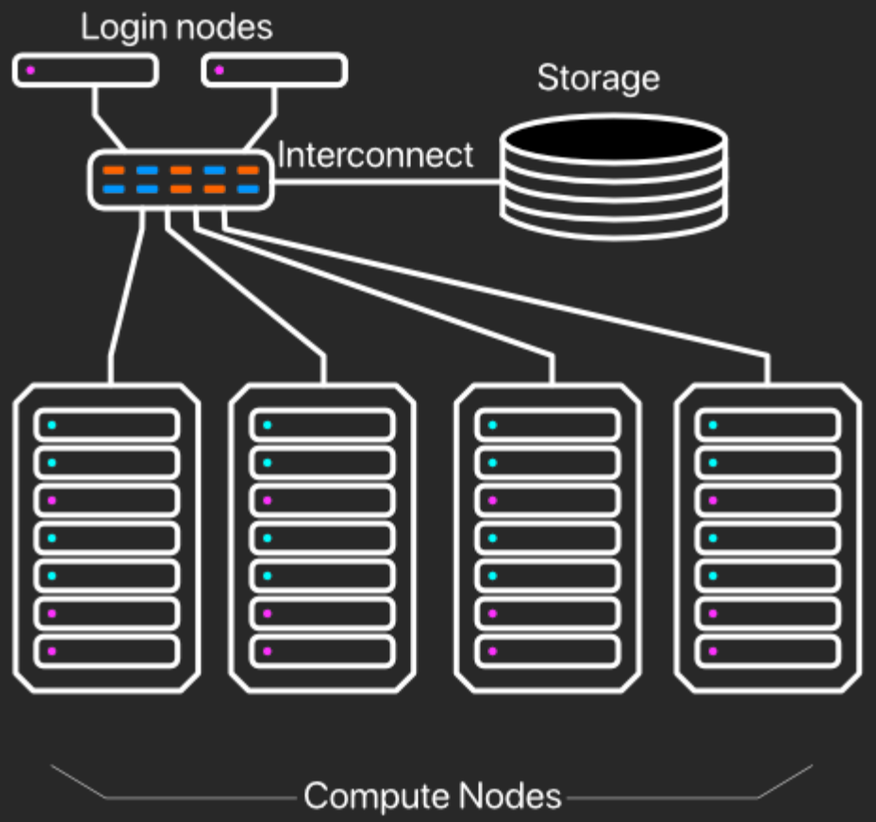


Cluster Computing

What's in a supercomputer?

- Compute Nodes
  - Computational units - CPU and potentially a co-processor, e.g. a GPU
  - Memory (i.e. RAM)
  - Some storage and/or NVMe
  - Network interfaces, possibly separate between management and workload

- Interconnect
  - Interfaces on nodes
  - Wiring and switches



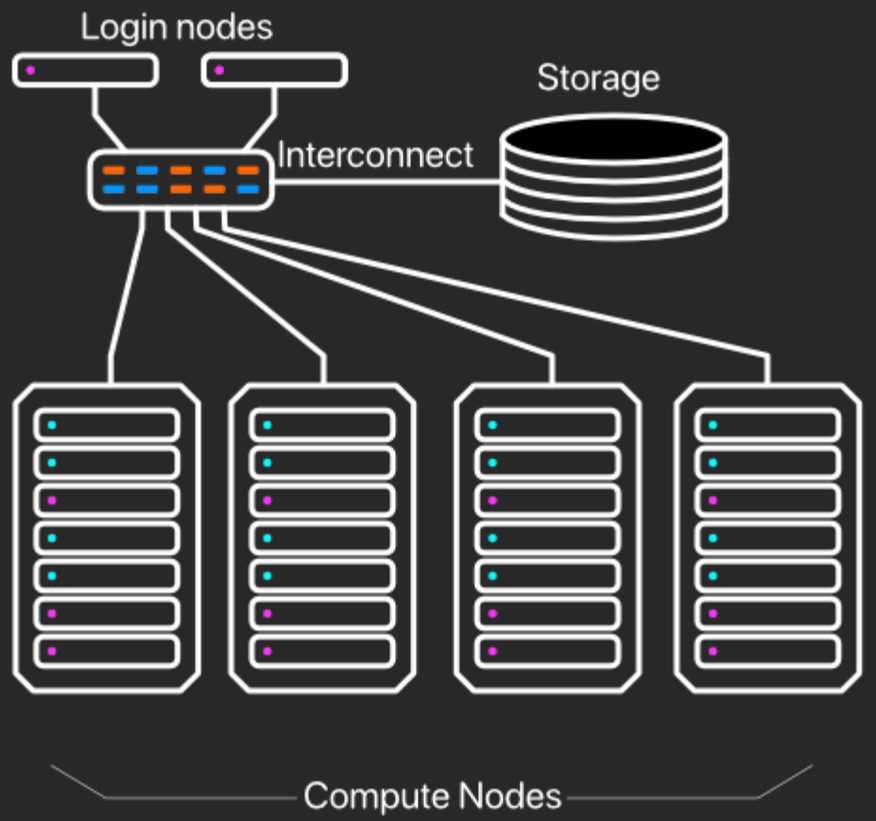
Cluster Computing

What's in a supercomputer?

- Compute Nodes
  - Computational units - CPU and potentially a co-processor, e.g. a GPU
  - Memory (i.e. RAM)
  - Some storage and/or NVMe
  - Network interfaces, possibly separate between management and workload

- Interconnect
  - Interfaces on nodes
  - Wiring and switches

- Storage
  - Still predominantly spinning disks
  - Solid state drives are emerging for smaller scratch space
  - Tape systems for archiving



Cluster Computing

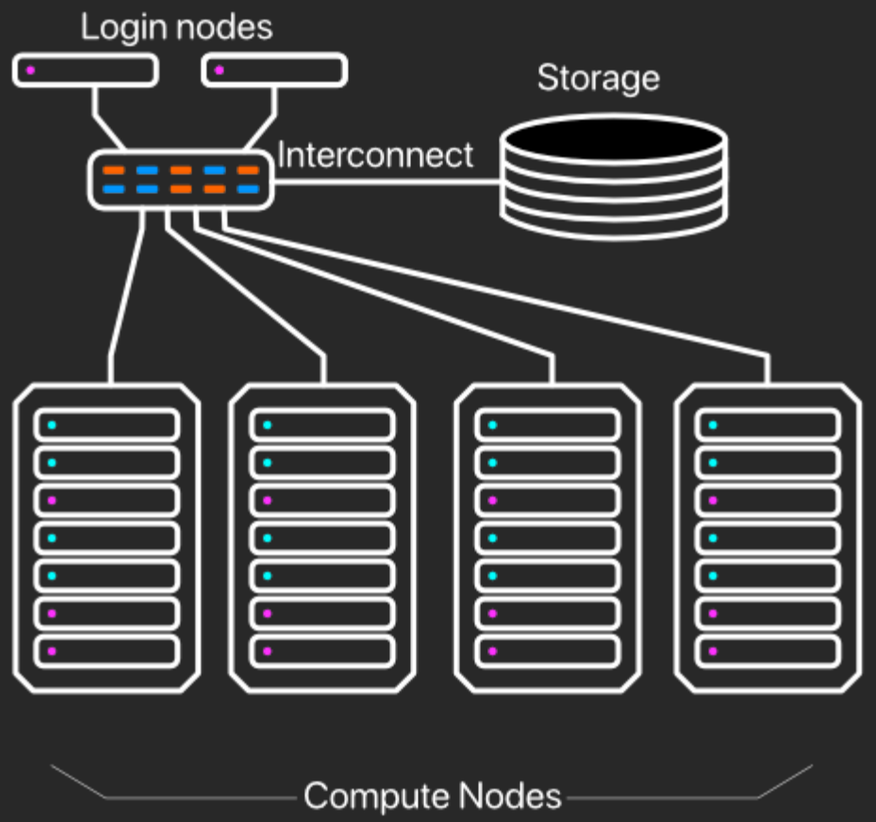
What's in a supercomputer?

- Compute Nodes
  - Computational units - CPU and potentially a co-processor, e.g. a GPU
  - Memory (i.e. RAM)
  - Some storage and/or NVMe
  - Network interfaces, possibly separate between management and workload

- Interconnect
  - Interfaces on nodes
  - Wiring and switches

- Storage
  - Still predominantly spinning disks
  - Solid state drives are emerging for smaller scratch space
  - Tape systems for archiving

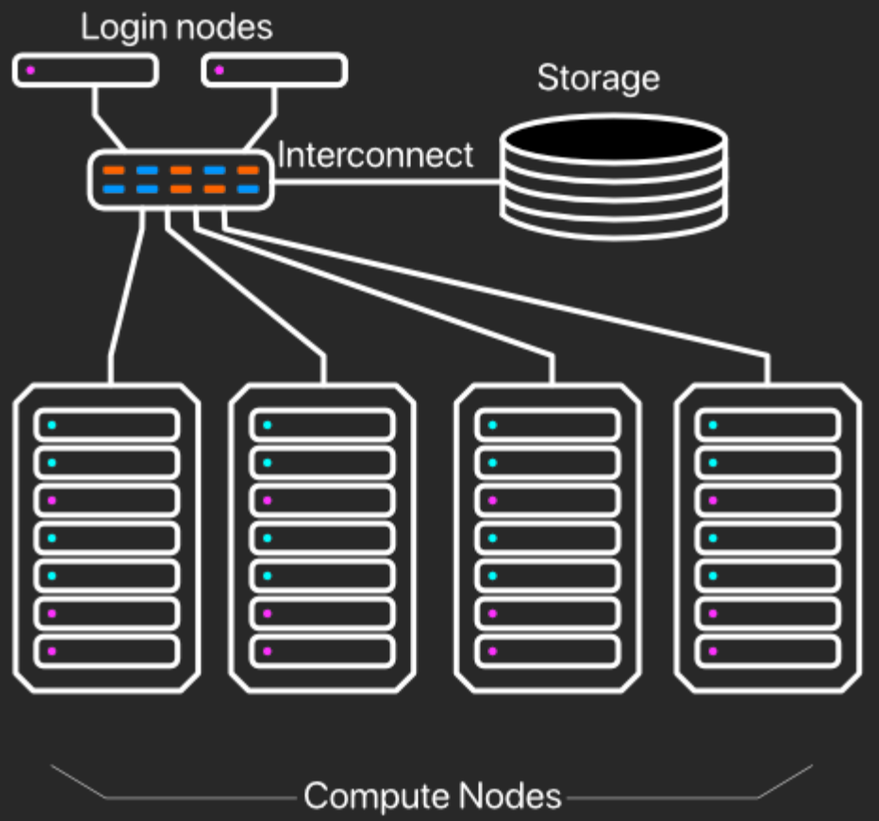
- Front-end nodes
  - For user access
  - Compiling, submitting jobs, etc.



Cluster Computing

Specific configuration of the Cyl cluster

- Various nodes with different architectures
  - Hostnames: `cyc0{1, ... ,8}`, `ph0{1,2,3,4}`, `cwp0{1,2}`, `cwg0{1,2}`, `sim0?`, etc.
- We will be using nodes from the `p100` partition for now:
  - Hostnames are `cyc01`, `cyc02`, ..., `cyc08`. Three are reserved for our lesson
  - 2× 16-core Intel Xeon
  - 128 GBytes RAM
  - 2× P100 GPUs each
- Common storage for our course: `/onyx/data/sds406f24/`



Cluster Computing

- Log in to a *login node* or *frontend node*. Login node in our case has hostname `front02`
- To run programs on *compute nodes*, a *job scheduler* is available
- Distinguish between *interactive* and *batch* jobs

Cluster Computing

- Log in to a *login node* or *frontend node*. Login node in our case has hostname `front02`
- To run programs on *compute nodes*, a *job scheduler* is available
- Distinguish between *interactive* and *batch* jobs

SLURM job scheduler

- See currently running and waiting jobs: `squeue`
- Ask for an interactive job: `salloc`
- Submit a batch job: `sbatch`
- Run an executable: `srun`

Cluster Computing Introductory Example

- Log in:

```
[localhost ~]$ ssh <username>@front02.hpcf.cyi.ac.cy
```

Cluster Computing Introductory Example

- Log in:

```
[localhost ~]$ ssh <username>@front02.hpcf.cyi.ac.cy
```

- Type `hostename`. This tells you the name of the node you are currently logged into:

```
[ikoutsou@front02 ~]$ hostname
front02
```

  this is the login node.

Cluster Computing Introductory Example

- Log in:

```
[localhost ~]$ ssh <username>@front02.hpcf.cyi.ac.cy
```

- Type `hostename`. This tells you the name of the node you are currently logged into:

```
[ikoutsou@front02 ~]$ hostname
front02
```

  this is the login node.

- Ask for a node:

```
[ikoutsou@front02 ~]$ salloc -N 1 -p p100 --reservation=sds406 -A sds406f24
salloc: Granted job allocation 298614
[ikoutsou@cyc01 ~]$
```

- Type `hostname` again:

```
[ikoutsou@cyc01 ~]$ hostname
cyc01
```

  this is a compute node.

Cluster Computing Introductory Example

- Release the node:

```
[ikoutsou@cyc01 ~]$ exit
exit
salloc: Relinquishing job allocation 69053
[ikoutsou@front02 ~]$
```

  we're back on `front02`

Cluster Computing Introductory Example

- Release the node:

```
[ikoutsou@cyc01 ~]$ exit
exit
salloc: Relinquishing job allocation 69053
[ikoutsou@front02 ~]$
```

  we're back on `front02`

- Our course reservation includes the nodes with `cyc` in the hostname.

Cluster Computing Introductory Example

- Release the node:

```
[ikoutsou@cyc01 ~]$ exit
exit
salloc: Relinquishing job allocation 69053
[ikoutsou@front02 ~]$
```

  we're back on `front02`

- Our course reservation includes the nodes with `cyc` in the hostname.

Please do not hold nodes unnecessarily; when you have nodes `salloc`ed you may be blocking other users from using those nodes.

Cluster Computing Introductory Example

- Release the node:

```
[ikoutsou@cyc01 ~]$ exit
exit
salloc: Relinquishing job allocation 69053
[ikoutsou@front02 ~]$
```

  we're back on `front02`

- Our course reservation includes the nodes with `cyc` in the hostname.

Please do not hold nodes unnecessarily; when you have nodes `salloc`ed you may be blocking other users from using those nodes.

- Use `srun` instead of `salloc`:

```
[ikoutsou@front02 ~]$ srun -n 1 -N 1 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203373 queued and waiting for resources
srun: job 203373 has been allocated resources
cyc01
```

Allocates a node, runs the specified command (in this case `hostname`), and then exits the node, releasing the allocation.

Cluster Computing Introductory Example

- Run multiple instances of `hostname` in parallel:

```
[ikoutsou@front02 ~]$ srun -N 1 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203374 queued and waiting for resources
srun: job 203374 has been allocated resources
cyc01
cyc01
```

Cluster Computing Introductory Example

- Run multiple instances of `hostname` in parallel:

```
[ikoutsou@front02 ~]$ srun -N 1 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203374 queued and waiting for resources
srun: job 203374 has been allocated resources
cyc01
cyc01
```

  - `-N 1`: use one node
  - `-n 2`: use two processes

Cluster Computing Introductory Example

- Run multiple instances of `hostname` in parallel:

```
[ikoutsou@front02 ~]$ srun -N 1 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203374 queued and waiting for resources
srun: job 203374 has been allocated resources
cyc01
cyc01
```

  - `-N 1`: use one node
  - `-n 2`: use two processes

- Run on more than one node:

```
[ikoutsou@front02 ~]$ srun -N 2 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203375 queued and waiting for resources
srun: job 203375 has been allocated resources
cyc01
cyc02
```

  runs one instance of `hostname` on each node

Cluster Computing Introductory Example

- Run multiple instances of `hostname` in parallel:

```
[ikoutsou@front02 ~]$ srun -N 1 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203374 queued and waiting for resources
srun: job 203374 has been allocated resources
cyc01
cyc01
```

  - `-N 1`: use one node
  - `-n 2`: use two processes

- Run on more than one node:

```
[ikoutsou@front02 ~]$ srun -N 2 -n 2 -p p100 --reservation=sds406 -A sds406f24 hostname
srun: job 203375 queued and waiting for resources
srun: job 203375 has been allocated resources
cyc01
cyc02
```

  runs one instance of `hostname` on each node

- Try:

```
[ikoutsou@front02 ~]$ srun -N 2 -n 4 -p p100 --reservation=sds406 -A sds406f24 hostname
```

```
[ikoutsou@front02 ~]$ srun -N 2 -n 3 -p p100 --reservation=sds406 -A sds406f24 hostname
```

Cluster Computing Introductory Example

Need to set up a text editor to edit files on the cluster

- Emacs and Vim are available on the cluster

Cluster Computing Introductory Example

Need to set up a text editor to edit files on the cluster

- Emacs and Vim are available on the cluster
- Other options are fine, *as long as you know what you're doing*

Cluster Computing Introductory Example

- Make a directory. List it.

```
[ikoutsou@front02 ~]$ mkdir SDS406
[ikoutsou@front02 ~]$ ls
SDS406
[ikoutsou@front02 ~]$
```

Cluster Computing Introductory Example

- Make a directory. List it.

```
[ikoutsou@front02 ~]$ mkdir SDS406
[ikoutsou@front02 ~]$ ls
SDS406
[ikoutsou@front02 ~]$
```

- Change into it:

```
[ikoutsou@front02 ~]$ cd SDS406/
[ikoutsou@front02 SDS406]$
```

Cluster Computing Introductory Example

- Make a directory. List it.

```
[ikoutsou@front02 ~]$ mkdir SDS406
[ikoutsou@front02 ~]$ ls
SDS406
[ikoutsou@front02 ~]$
```

- Change into it:

```
[ikoutsou@front02 ~]$ cd SDS406/
[ikoutsou@front02 SDS406]$
```

- While in `SDS406/`, make another one for this week's lesson and change into it:

```
[ikoutsou@front02 SDS406]$ mkdir l01
[ikoutsou@front02 SDS406]$ cd l01/
[ikoutsou@front02 l01]$
```

Cluster Computing Introductory Example

- Make a directory. List it.

```
[ikoutsou@front02 ~]$ mkdir SDS406
[ikoutsou@front02 ~]$ ls
SDS406
[ikoutsou@front02 ~]$
```

- Change into it:

```
[ikoutsou@front02 ~]$ cd SDS406/
[ikoutsou@front02 SDS406]$
```

- While in `SDS406/`, make another one for this week's lesson and change into it:

```
[ikoutsou@front02 SDS406]$ mkdir l01
[ikoutsou@front02 SDS406]$ cd l01/
[ikoutsou@front02 l01]$
```

- `pwd` will tell you where you are in the file system:

```
[ikoutsou@front02 l01]$ pwd
/home/ikoutsou/SDS406/l01
```

Cluster Computing Introductory Example

Let's write our own `hostname` command (you were hoping for "Hello world" 🙃 ?)

Cluster Computing Introductory Example

Let's write our own `hostname` command (you were hoping for "Hello world" 🙃 ?)

- Use `emacs` to write the source code from the terminal:

```
[ikoutsou@front02 l01]$ emacs my_hn.c
```

Cluster Computing Introductory Example

Let's write our own `hostname` command (you were hoping for "Hello world" 🙃 ?)

- Use `emacs` to write the source code from the terminal:

```
[ikoutsou@front02 l01]$ emacs my_hn.c
```

- Type in the following program:

```c
#include <unistd.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    char hname[256];
    gethostname(hname, 256);
    printf(" Hostname is: %s\n", hname);
    return 0;
}
```

Cluster Computing Introductory Example

Let's write our own `hostname` command (you were hoping for "Hello world" 🙃 ?)

- Use `emacs` to write the source code from the terminal:

```
[ikoutsou@front02 l01]$ emacs my_hn.c
```

- Type in the following program:

```c
#include <unistd.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    char hname[256];
    gethostname(hname, 256);
    printf(" Hostname is: %s\n", hname);
    return 0;
}
```

- To save, hold down `ctrl`, then hold and release `x`, then hold and release `s`
- To exit emacs, hold down `ctrl`, then hold and release `x`, then hold and release `c`

Cluster Computing Introductory Example

Let's write our own `hostname` command (you were hoping for "Hello world" 🙃 ?)

- Use `emacs` to write the source code from the terminal:

```
[ikoutsou@front02 l01]$ emacs my_hn.c
```

- Type in the following program:

```c
#include <unistd.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    char hname[256];
    gethostname(hname, 256);
    printf(" Hostname is: %s\n", hname);
    return 0;
}
```

- To save, hold down `ctrl`, then hold and release `x`, then hold and release `s`
- To exit emacs, hold down `ctrl`, then hold and release `x`, then hold and release `c`
- Bookmark the Emacs reference card:
  https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

- `ls` should show the file `my_hn.c`, which you just typed in and saved:

```
[ikoutsou@front02 l01]$ ls
my_hn.c
```

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

- `ls` should show the file `my_hn.c`, which you just typed in and saved:

```
[ikoutsou@front02 l01]$ ls
my_hn.c
```

- It's time to *compile* it into an executable we can run:

```
[ikoutsou@front02 l01]$ module load gompi
[ikoutsou@front02 l01]$ gcc my_hn.c -o hn
```

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

- `ls` should show the file `my_hn.c`, which you just typed in and saved:

```
[ikoutsou@front02 l01]$ ls
my_hn.c
```

- It's time to *compile* it into an executable we can run:

```
[ikoutsou@front02 l01]$ module load gompi
[ikoutsou@front02 l01]$ gcc my_hn.c -o hn
```

  - `-o hn` means "name the resulting executable `hn`". If you don't specify `-o` the executable name defaults to `a.out`

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

- `ls` should show the file `my_hn.c`, which you just typed in and saved:

```
[ikoutsou@front02 l01]$ ls
my_hn.c
```

- It's time to *compile* it into an executable we can run:

```
[ikoutsou@front02 l01]$ module load gompi
[ikoutsou@front02 l01]$ gcc my_hn.c -o hn
```

  - `-o hn` means "name the resulting executable `hn`". If you don't specify `-o` the executable name defaults to `a.out`

- Type `ls` to make sure it has been created. Then run it on the frontend node:

```
[ikoutsou@front02 l01]$ ls
hn my_hn.c
[ikoutsou@front02 l01]$ ./hn
Hostname is: front02
```

Cluster Computing Introductory Example

After exiting emacs you are back at the command line

- `ls` should show the file `my_hn.c`, which you just typed in and saved:

```
[ikoutsou@front02 l01]$ ls
my_hn.c
```

- It's time to *compile* it into an executable we can run:

```
[ikoutsou@front02 l01]$ module load gompi
[ikoutsou@front02 l01]$ gcc my_hn.c -o hn
```

  - `-o hn` means "name the resulting executable `hn`". If you don't specify `-o` the executable name defaults to `a.out`

- Type `ls` to make sure it has been created. Then run it on the frontend node:

```
[ikoutsou@front02 l01]$ ls
hn my_hn.c
[ikoutsou@front02 l01]$ ./hn
Hostname is: front02
```

Note: commands like `gcc` or `ls` are globally accessible because their locations are included in your shell environment's search path. For `hn` though, which we just created, you need to explicitly give its path, in this case via `./` which means "current directory".