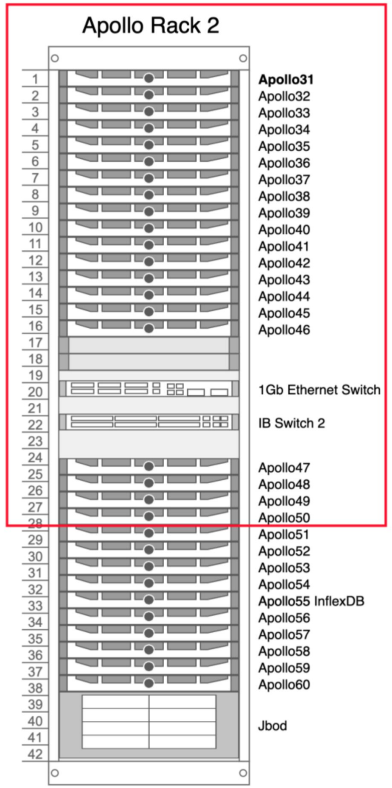
Lab1
Platform Introduction & MPI

Parallel Programming 2025
Sep. 11

Platform Introduction - Apollo Origo



- 40 nodes for this course
  - 1 head node: apollo1
  - 35 compute nodes
  - 4 storage nodes: apollo[16-19]
- 2x Intel X5670 6 cores @ 2.93GHz
  - HyperThreading is enabled only on the head node
- 96GB RAM on each node
- 40Gb/s QDR InfiniBand
- Debian 12 (kernel 6.1.0)
- Storage
  - /home & /share: BeeGFS parallel file system
    - For performance benchmarking
  - /opt: NFS

Available resources

- 1 login node (apollo1) (200% CPU max)
- 35 compute nodes (1200% CPU max)
  - Use `squeue` to view Slurm queues

Login to the Apollo Origo Cluster

Login to Apollo

- Address: origo.apollo.cs.nthu.edu.tw
- Username: check email
- Password: check email
- MINING IS PROHIBITED. Also, do not attack the server.

SSH

- Open terminal
- `ssh pp25sXXX@origo.apollo.cs.nthu.edu.tw`
- Enter password
- You'll be asked to change your password on the first login
    - So it’s recommended not to connect via VS Code at the first time

Some useful command

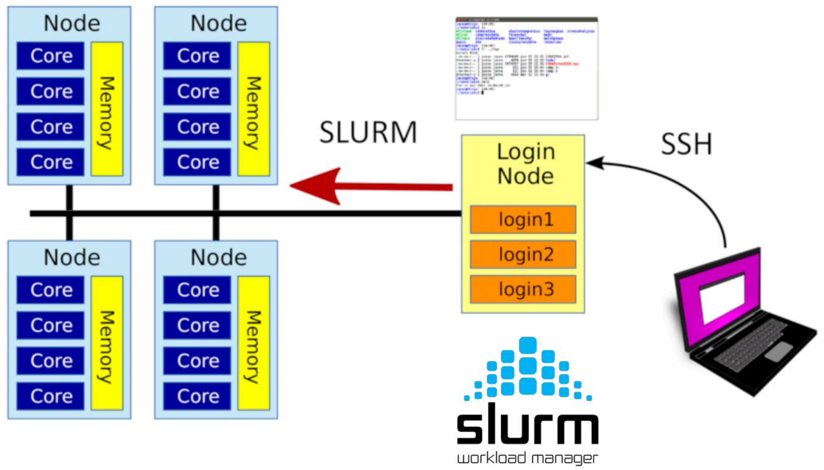
- Login: `ssh pp25sXXX@origo.apollo.cs.nthu.edu.tw`
- File transfer:
  ```
  rsync -avhP $FILE
  pp25sXXX@origo.apollo.cs.nthu.edu.tw:$FILE
  ```
- Editors: `vim`, `emacs`, `nano`
- Slurm job queue: `squeue`
- Change password: `passwd`
- Download file: `wget`, `aria2c`
- Code syntax highlighting: `pygmentize`

Using the Apollo Origo Cluster

Slurm Workload Manager

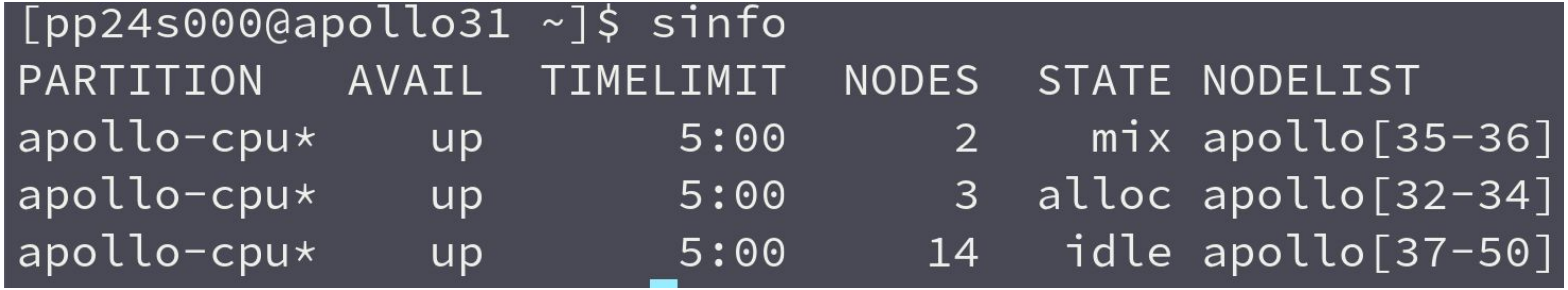
On a cluster system, there are multiple users and multiple nodes. Slurm schedules jobs submitted by users across different nodes, so that the same resource is not used by two jobs at the same time (to ensure accuracy of performance-critical experiments), and also increases the utilization of the cluster.

- Note: Slurm prefers the following jobs:
  - short jobs (you can set time limit)
  - less resource demanding jobs
  - jobs queued for a long time
  - users that haven't run a lot of jobs recently



Slurm Workload Manager

- Show partition information
  - `sinfo`
    - Displays the information about Slurm nodes and partitions, providing an overview of the cluster's status.
    - We have only one partition, which is the CPU partition
      - idle: The nodes are not allocated
      - alloc: The nodes are fully allocated
      - mix: The nodes are partially allocated
      - down / drain: The nodes are out of service (we will fix them ASAP!)



```
[pp24s000@apollo31 ~]$ sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
apollo-cpu*    up       5:00      2    mix apollo[35-36]
apollo-cpu*    up       5:00      3  alloc apollo[32-34]
apollo-cpu*    up       5:00     14   idle apollo[37-50]
```

Slurm Workload Manager

- Interactive Job submission
    - `srun [options] ./executable [args]`
        - Options:
            - `-N $NODES`: The the **minimum number of nodes** to run the job
            - `-n $PROCESSES`: The number of **total process** to launch
            - `-c $CPUS`: The number of cpus available to **each process** (i.e. threads per process)
            - `-t $TIME`: The time limit in "minutes" or "minutes:seconds"
            - `-J $NAME`: The name of the job. Will be displayed on squeue.
    - `srun -N 2 -n 8 -c 2 ./hello_world`
        - Runs a total of 8 processes, each using 2 CPUs, on at least 2 nodes
            - 6 processes * 2 cpus on 1st node (CPU Allocation: 12/12)
            - 2 processes * 2 cpus on 2nd node (CPU Allocation: 4/12)

Slurm Workload Manager

- Batch Job submission
  - `sbatch script.sh`

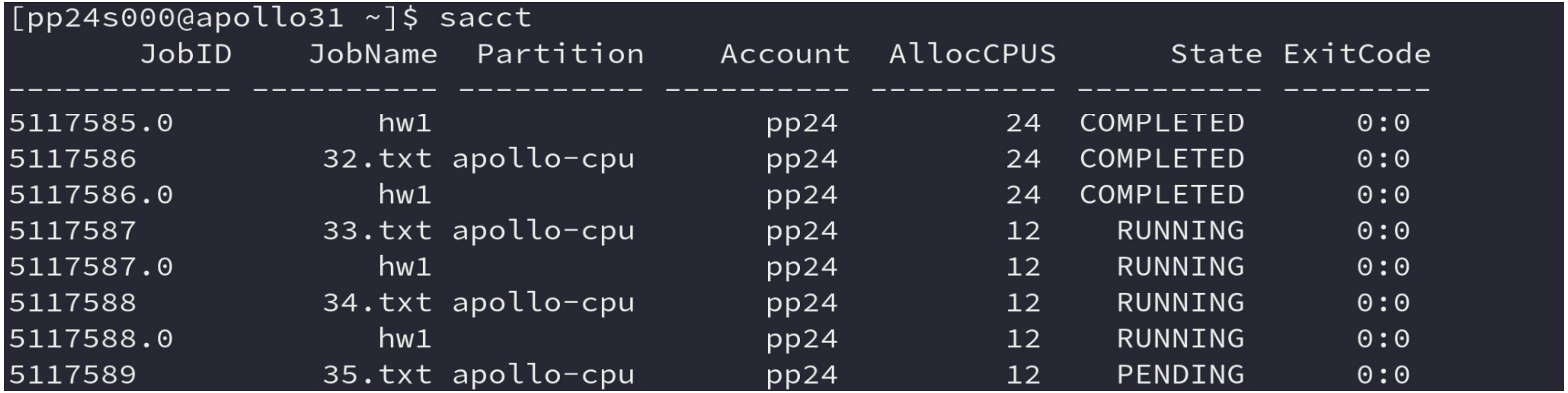
script.sh

```
#!/bin/bash
#SBATCH -J BatchRun
#SBATCH -N 2
#SBATCH -n 4
#SBATCH -c 6

srun ./hello_world
```

Slurm Workload Manager

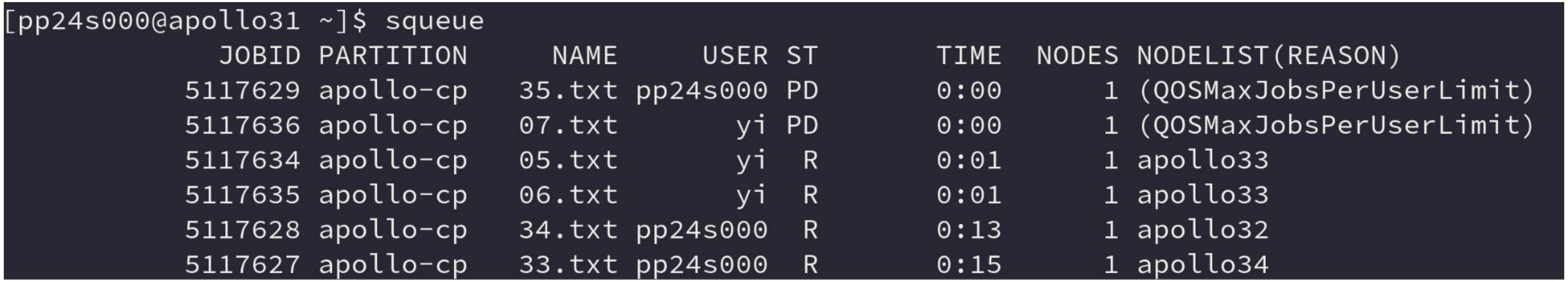
- Tracking your jobs
    - `sacct`



```
[pp24s000@apollo31 ~]$ sacct
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
5117585.0           hw1                  pp24         24  COMPLETED      0:0
5117586          32.txt apollo-cpu       pp24         24  COMPLETED      0:0
5117586.0           hw1                  pp24         24  COMPLETED      0:0
5117587          33.txt apollo-cpu       pp24         12    RUNNING      0:0
5117587.0           hw1                  pp24         12    RUNNING      0:0
5117588          34.txt apollo-cpu       pp24         12    RUNNING      0:0
5117588.0           hw1                  pp24         12    RUNNING      0:0
5117589          35.txt apollo-cpu       pp24         12    PENDING      0:0
```

Slurm Workload Manager

- Show the job queue
  - `squeue`
  - `squeue -u $USER`



```
[pp24s000@apollo31 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5117629 apollo-cp   35.txt pp24s000 PD       0:00      1 (QOSMaxJobsPerUserLimit)
           5117636 apollo-cp   07.txt       yi PD       0:00      1 (QOSMaxJobsPerUserLimit)
           5117634 apollo-cp   05.txt       yi  R       0:01      1 apollo33
           5117635 apollo-cp   06.txt       yi  R       0:01      1 apollo33
           5117628 apollo-cp   34.txt pp24s000  R       0:13      1 apollo32
           5117627 apollo-cp   33.txt pp24s000  R       0:15      1 apollo34
```

Slurm Workload Manager

- Cancel a job
  - `scancel [Job ID]`

Slurm Limits on Apollo Origo

- Maximum 4 nodes per job
  - Job with > 4 nodes will be denied
- Maximum 48 cores per user
  - Later submitted jobs will be queued
- Maximum 2 running jobs per user
  - Later submitted jobs will be queued
- Maximum 8 job submissions per user
  - Later submitted jobs will be denied

Environment Modules (`module`, `ml`)

- A tool to simplify shell initialization and dynamically manage environment settings using “modulefiles”.
- Dynamically update environment variables like `PATH`, `LD_LIBRARY_PATH`.
- Simplify the process of switching between different software versions.
- Ideal for managing complex software dependencies.

- Used on almost ALL supercomputing clusters

Common usages

- List Available Modules
    - `module avail`
    - Shortcut: `ml av`
- Load a Module
    - `module load $MODULE_NAME`
    - E.g., `module load mpi/latest`
    - Shortcut: `ml mpi`
- Unload a Module
    - `module unload $MODULE_NAME`
    - E.g., `module unload mpi/latest`
    - Shortcut: `ml -mpi`
- List Loaded Modules
    - `module list`
    - Shortcut: ``ml``
- Show Module Info
    - `module show $MODULE_NAME`
    - E.g., module show openmpi
- Swap Modules
    - `module swap $MODULE1 $MODULE2`
    - E.g., `module swap mpi/latest openmpi`
- Unload All Modules
    - `module purge`

MPI Hello World

MPI Hello World (Send, Recv Test)

```c
#include "mpi.h"
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   int i, rank, size, namelen;
   char name[MPI_MAX_PROCESSOR_NAME];
   MPI_Status stat;

   MPI_Init(&argc, &argv);

   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Get_processor_name(name, &namelen);

   if (rank == 0) {
       printf("Hello world: rank %d of %d running on %s\n", rank, size, name);

       for (i = 1; i < size; i++) {
           MPI_Recv(&rank, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &stat);
           MPI_Recv(&size, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &stat);
           MPI_Recv(&namelen, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &stat);
           MPI_Recv(name, namelen + 1, MPI_CHAR, i, 1, MPI_COMM_WORLD, &stat);
           printf("Hello world: rank %d of %d running on %s\n", rank, size, name);
       }
   } else {
       MPI_Send(&rank, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
       MPI_Send(&size, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
       MPI_Send(&namelen, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
       MPI_Send(name, namelen + 1, MPI_CHAR, 0, 1, MPI_COMM_WORLD);
   }

   MPI_Finalize();
   return (0);
}
```

Get this code on Apollo:

```
cp /home/pp25/share/lab1/mpi_test.c
```

MPI_Send

```
int MPI_Send(const void *buf,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag,
             MPI_Comm comm);
```

MPI_Recv

```
int MPI_Recv(void *buf,
             int count,
             MPI_Datatype datatype,
             int source,
             int tag,
             MPI_Comm comm,
             MPI_Status *status);
```

Provided Compilers & MPIs

- Compilers
  - GCC 12.2.0 (Default)
    - `gcc` / `g++`
  - Intel Compiler Classic
    - `module load icc`
    - `icc` / `icpc`
  - Intel oneAPI Compiler
    - `module load compiler`
    - `icx` / `icpx`

- MPI Implementations
  - Intel MPI
    - `module load mpi`
  - OpenMPI 5.0.2 + UCX
    - `module load openmpi`

Using different combinations of MPI & compilers

- Intel MPI + GCC
  - `module load mpi`
  - `mpicxx …`
- Intel MPI + ICC
  - `module load mpi icc`
  - `I_MPI_CXX=icpc mpicxx …`
- OpenMPI + GCC
  - `module load openmpi`
  - `mpicxx …`
- OpenMPI + ICC
  - `module load openmpi icc`
  - `OMPI_CXX=icpc mpicxx …`

Practice

- Compile and run the hello world program.
    - `mpicxx test.c -o test`
    - `./test`
- Compare different srun options

Measuring time in your code

Correct measurement method

- `srun -n4 time ./hello`
- `sbatch` & `time srun`
- `MPI_Wtime()`
- `omp_get_wtime()`
- `clock_gettime(CLOCK_MONOTONIC, ...)`
- `std::chrono::steady_clock`

```
#!/bin/bash
#SBATCH -n 4
#SBATCH -N 2
time srun ./hello
```

Example: `MPI_Wtime()`

```c
double tik, tok;
tik = MPI_Wtime();
// ....  stuff to be timed  ...
tok = MPI_Wtime();
printf("That took %f seconds\n", tok - tik);
```

Example: `clock_gettime(CLOCK_MONOTONIC, ...)`

```
int main() {
   struct timespec start, end, temp;
   double time_used;
   clock_gettime(CLOCK_MONOTONIC, &start);

   // ....  stuff to be timed  ...

   clock_gettime(CLOCK_MONOTONIC, &end);
   if (end.tv_nsec - start.tv_nsec < 0) {
       temp.tv_sec = end.tv_sec - start.tv_sec - 1;
       temp.tv_nsec = 1000000000 + end.tv_nsec - start.tv_nsec;
   } else {
       temp.tv_sec = end.tv_sec - start.tv_sec;
       temp.tv_nsec = end.tv_nsec - start.tv_nsec;
   }
   time_used = temp.tv_sec + (double) temp.tv_nsec / 1000000000.;

   printf("%f second\n", time_used);
}
```

Wrong measurement method

- `time srun -n4 ./hello`: this time include queuing time
- `time(NULL)`: the resolution is too low (1-second)
- `clock()`: it will count 2x time when using two threads and will not include I/O time.
- `clock_gettime(CLOCK_REALTIME, ...)`: it will be affected by NTP adjustments and DST changes.
- `std::high_resolution_clock::now()`: it may be affected by NTP adjustments and DST changes.

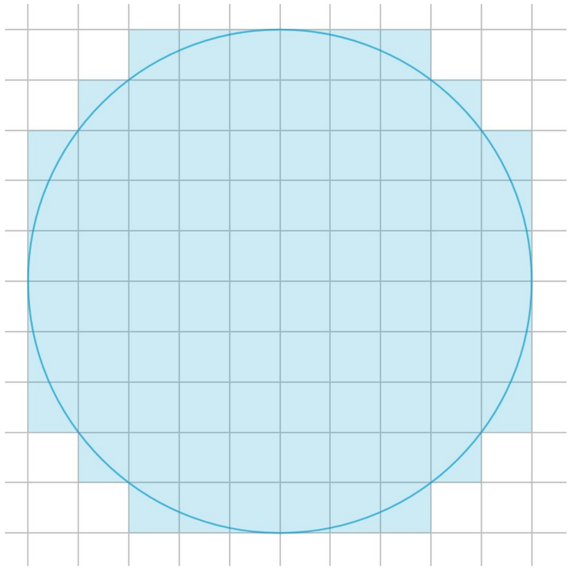
Lab 1 - Pixels in circle

Pixels in circle

Suppose we want to draw a filled circle of radius r on a 2D monitor, how many pixels will be filled?

We fill a pixel when any part of the circle overlaps with the pixel. We also assume that the circle center is at the boundary of 4 pixels.

For example, 88 pixels are filled when $r = 5$.



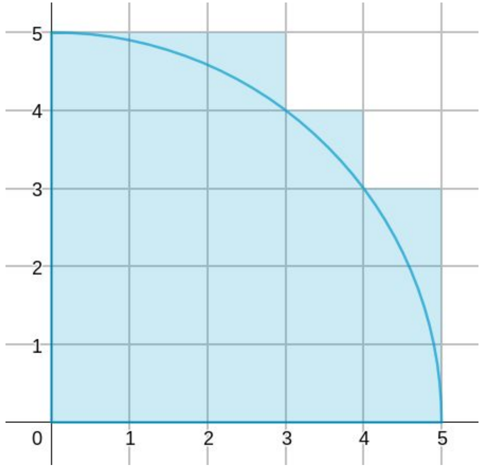
Pixels in circle

Equation:

$$\text{pixels}(r) = 4 \times \sum_{x=0}^{r-1} \left\lceil \sqrt{r^2 - x^2} \right\rceil$$

Example: r = 5

$$\begin{aligned}\text{pixels}(5) &= 4\left(\left\lceil\sqrt{25-0}\right\rceil + \left\lceil\sqrt{25-1}\right\rceil + \left\lceil\sqrt{25-4}\right\rceil + \left\lceil\sqrt{25-9}\right\rceil + \left\lceil\sqrt{25-16}\right\rceil\right)\\ &= 4(5+5+5+4+3)\\ &= 88\end{aligned}$$

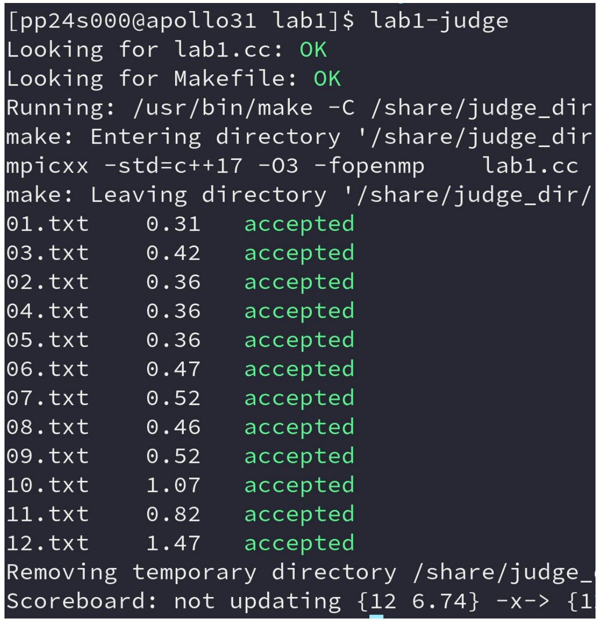


Lab Spec

- Parallelize the calculation using MPI.
- Program input format: `srun -N$NODE -n$PROC ./lab1 $R $K`
    - `$NODE`: number of nodes
    - `$PROC`: number of MPI processes
    - `$R`: the radius of circle, integer
    - `$K`: integer
- Program output: `pixels % k` (Since the output pixels may be very large, we output the remainder instead.)
- Your program should be at least (*n*/2) times faster than the sequential version when running with *n* processes. For example, when running with 12 processes, your execution time should not exceed 1/6 of the sequential code.

Lab Spec

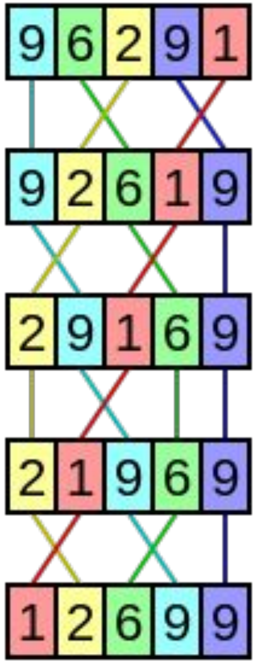
- The sequential code `lab1.cc` and a build file `Makefile` can be found at `/home/pp25/share/lab1/sample`, copy these files to your home directory.
- All of the test cases can be found in `/home/pp25/share/lab1/testcases`
- Within the same directory of your `lab1.cc` and `Makefile`, run `lab1-judge` to check.
- Scoreboard
- Submit your code to eeclass:
  - `lab1.cc`
  - `Makefile` (optional, if you change any compile flags)
  - module.list (optional)
    - By default, will use GCC & Intel MPI to judge your code
    - If you are using other modules (such as openmpi), run `module save ./module.list` to generate it
- Due: 9/18 (Thu.) 23:59 (1 week)
- Full score for AC of all 12 test cases; otherwise, zero.



```
[pp24s000@apollo31 lab1]$ lab1-judge
Looking for lab1.cc: OK
Looking for Makefile: OK
Running: /usr/bin/make -C /share/judge_dir
make: Entering directory '/share/judge_dir
mpicxx -std=c++17 -O3 -fopenmp    lab1.cc
make: Leaving directory '/share/judge_dir/
01.txt    0.31   accepted
03.txt    0.42   accepted
02.txt    0.36   accepted
04.txt    0.36   accepted
05.txt    0.36   accepted
06.txt    0.47   accepted
07.txt    0.52   accepted
08.txt    0.46   accepted
09.txt    0.52   accepted
10.txt    1.07   accepted
11.txt    0.82   accepted
12.txt    1.47   accepted
Removing temporary directory /share/judge_
Scoreboard: not updating {12 6.74} -x-> {1
```

HW1: Odd-Even Sort

- Spec
- Due: 10/5 23:59



Next Lab on 10/2 (3 weeks later)

- Lab2 & HW2 release
  - HW2: Deadline on 11/2, on another CPU platform (maybe)