2023 Fall Parallel Programming

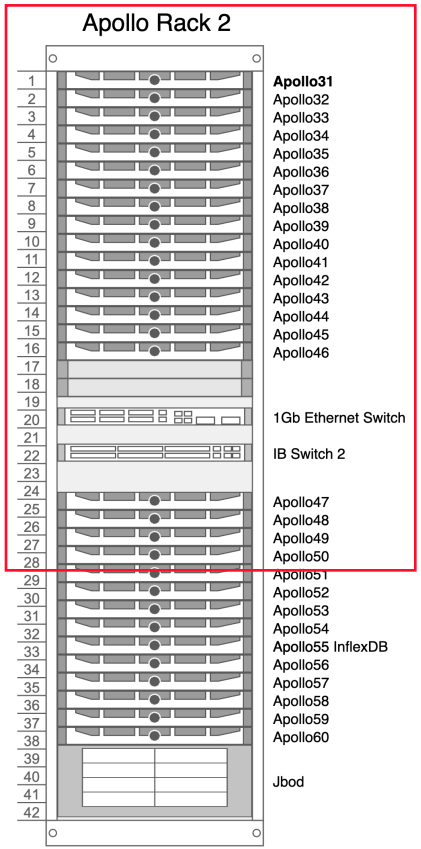
Platform Introduction & MPI Lab1

NTHU LSA-Lab

Outline

- Platform introduction - Apollo
- Login to Apollo
- How to use Apollo cluster
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

Platform introduction - Apollo



Hardware Spec

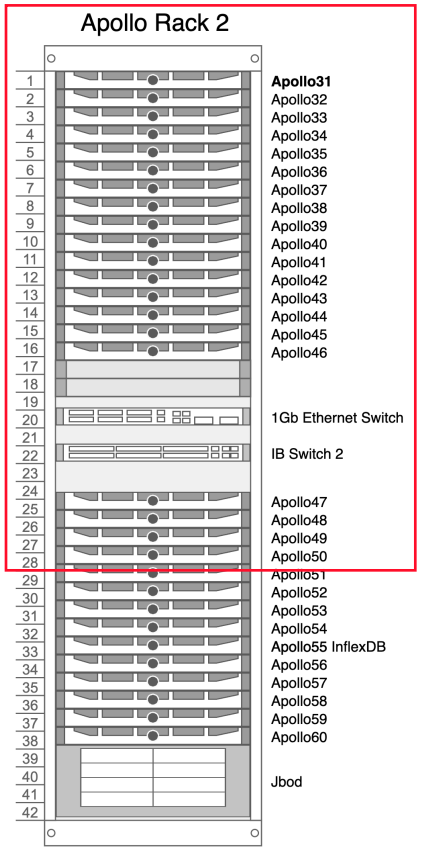
- 20 nodes for this course (apollo31-50)
- Intel X5670 2x6 cores @ 2.93GHz (Hyper threading disabled)
- 96GB RAM (each node)
- 5.5TB shared RAID5 disk
- QDR Infiniband (40 Gb/s)

Software Spec

- OS: Arch Linux kernel 5.15
- MPI: IntelMPI version 2023, OpenMPI 4.1.5
- Compilers: GCC 10.2.0, Clang 11.0.1
- Workload Manager: Slurm
- Env module
- Shared file system : NFS v4 on Apollo31

LSA-Lab

Platform introduction - Apollo



Available Resource

- 1 login node (apollo31) (200%CPU max)
- 19 compute nodes (1200% CPU max)
- Use `squeue` to view SLURM usage
- Cluster monitor: http://apollo.cs.nthu.edu.tw/monitor
- 48GB disk space per user
- Use `quota -s` to view disk quota

Outline

- Platform introduction - Apollo
- Login to Apollo
- How to use Apollo cluster
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

Login to Apollo

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

SSH - Linux and Mac

- Open terminal
- `ssh pp23sXX@apollo.cs.nthu.edu.tw`
- Enter password
- You'll be ask to change your password on first login

SSH - Windows

- Tools
  - MobaXterm
  - Putty
  - Cmd or Powershell (Windows 10)
  - Windows Terminal (Windows 11)
- `ssh pp23sXX@apollo.cs.nthu.edu.tw`
- Enter password
- You'll be ask to change your password on first login

Outline

- Platform instruction - Apollo
- Login to Apollo
- How use Apollo cluster
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

Some useful command

- Login: `ssh pp23sXX@apollo.cs.nthu.edu.tw`
- File transfer:
  - `rsync -avhP filename pp23sXX@apollo.cs.nthu.edu.tw:filename`
- Editors: `vim, emacs, nano`
- Disk quota: `quota -s`
- Change password: `passwd`
- Download file: `wget, aria2c`
- Code syntax highlighting: `pygmentize`

Introduction to Environment Modules

What are Environment Modules?

- 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.

Introduction to Environment Modules

Important Commands

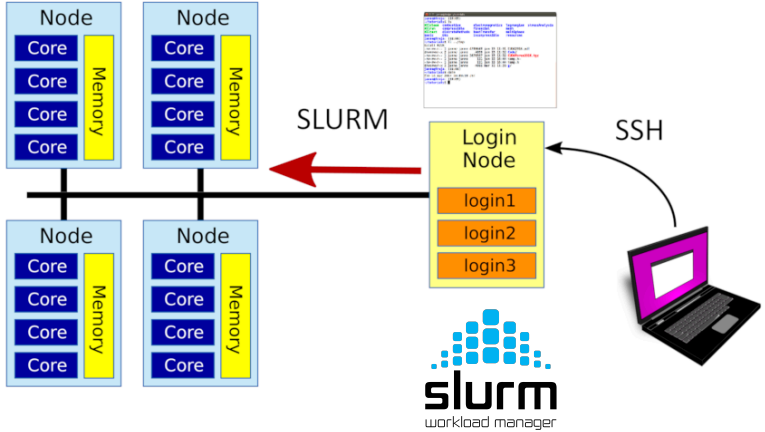
- **Load a Module:** `module load <module_name>`
  - `E.g., module load mpi/latest`
- **Unload a Module:** `module unload <module_name>`
  - `E.g., module unload mpi/latest`
- **Swap Modules:** `module swap <module1> <module2>`
  - E.g., module swap python/2.7 python/3.8
- **Show Module Info:** `module show <module_name>`
  - E.g., module show gcc/9.2
- **List Loaded Modules:** `module list`
- **Available Modules:** `module avail`
- **Purge Modules:** `module purge`

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.

SLURM prefer 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



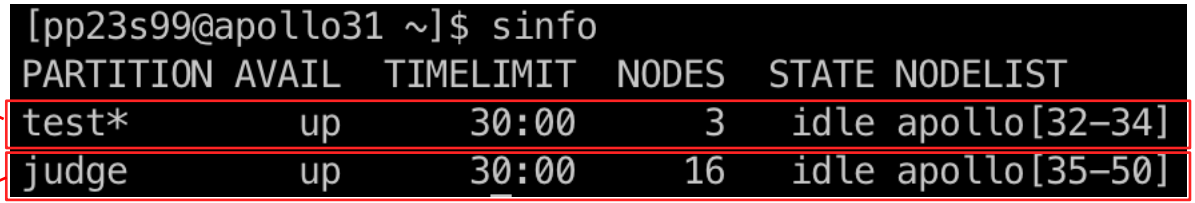
Slurm Workload manager

Dump slurm information

- `sinfo`: It displays information about SLURM nodes and partitions, providing an overview of the cluster's status.

Default partition for testing and debugging

Judge partition for judge script



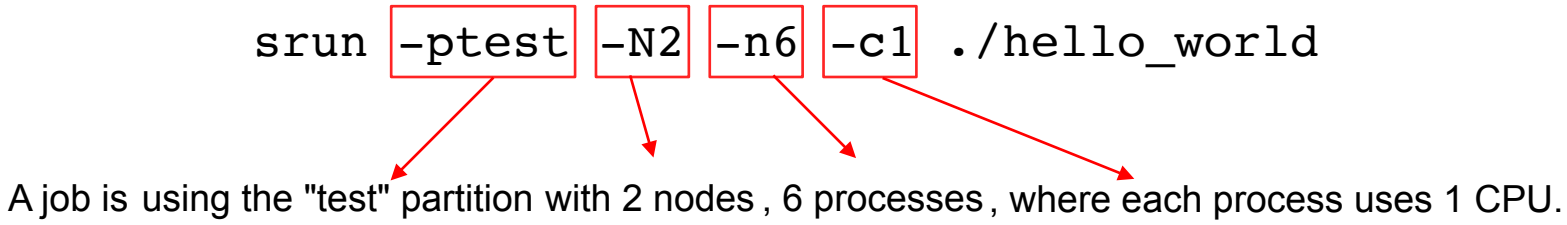
Slurm Workload manager

Job submission using srun

- `srun [options] ./executable [args]`
- Options:
  - `-N NODES`: NODES is the number of nodes to run the job
  - `-n PROCESSES`: PROCESSES is the number of total process to launch
  - `-c CPUS`: CPUS is the number of cpus available to each process
  - `-t TIME`: The time limit in "minutes" or “minutes:seconds"
  - `-p PARTITION`: Partition you want, if not slurm will using default partition
  - `-J NAME`: The name of the job. Will be displayed on squeue

Slurm Workload manager

Job submission using srun



Slurm Workload manager

Job submission using `sbatch`

- Using `sbatch` command to submit jobs in the background
- You can write a simple script to do that

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

- `$ sbatch script.sh`

Slurm Workload manager

Tracking a slurm job

- `squeue`: view submitted jobs in queue
- `scancel JOBID`: cancel a job with its JOBID
- `scontorl show job JOBID`: see more info for a specific job

```
[root@apollo31 ~]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           4320151       410   09.txt   kerwin PD       0:00      2 (QOSMaxJobsPerUserLimit)
           4320152       410   10.txt   kerwin PD       0:00      3 (QOSMaxJobsPerUserLimit)
           4320150       410   08.txt   kerwin  R       0:01      3 apollo[44-46]
           4320143       410   03.txt   kerwin  R       0:08      4 apollo[40-43]
```

More information see https://man.twcc.ai/@twccdocs/doc-twnia2-main-zh/https%3A%2F%2Fman.twcc.ai%2F%40twccdocs%2Fguide-twnia2-job-state-zh

Outline

- Platform introduction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

MPI hello world

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

int main(int argc, char *argv[]) {
  int rank, size;
  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank); // the total number of process
  MPI_Comm_size(MPI_COMM_WORLD, &size); // the rank (id) of the calling process

  printf("Hello, World.  I am %d of %d\n", rank, size);

  MPI_Finalize();
  return 0;
}
```

You can download this code directly on apollo.

```
wget https://www.open-mpi.org/papers/workshop-2006/hello.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)
```

MPI_Reduce

```
int MPI_Reduce(const void *sendbuf,
               void *recvbuf,
               int count,
               MPI_Datatype datatype,
               MPI_Op op,
               int root,
               MPI_Comm comm)
```

Outline

- Platform instruction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

Compilation

- Load Intel mpi module before you compile your code

```
[kerwin@apollo31 hw1]$ module avail
------------------------------------------------------------------ /opt/intel/oneapi/modulefiles ------------------------------------------------------------------
advisor/2023.2.0   debugger/latest          dnnl-cpu-tbb/2023.2.0  dpl/latest                 intel_ippcp_ia32/2021.8.0     mkl/latest         tbb/2021.10.0
advisor/latest     dev-utilities/2021.10.0  dnnl-cpu-tbb/latest    inspector/2023.2.0         intel_ippcp_ia32/latest       mkl32/2023.2.0     tbb/latest
ccl/2021.10.0      dev-utilities/latest     dnnl/2023.2.0          inspector/latest           intel_ippcp_intel64/2021.8.0  mkl32/latest       tbb32/2021.10.0
ccl/latest         dnnl-cpu-gomp/2023.2.0   dnnl/latest            intel_ipp_ia32/2021.9.0    intel_ippcp_intel64/latest    mpi/2021.10.0      tbb32/latest
dal/2023.2.0       dnnl-cpu-gomp/latest     dpct/2023.2.0          intel_ipp_ia32/latest      itac/2021.10.0                mpi/latest         vtune/2023.2.0
dal/latest         dnnl-cpu-iomp/2023.2.0   dpct/latest            intel_ipp_intel64/2021.9.0 itac/latest                   oclfpga/2023.2.0   vtune/latest
debugger/2023.2.0  dnnl-cpu-iomp/latest     dpl/2022.2.0           intel_ipp_intel64/latest   mkl/2023.2.0                  oclfpga/latest

------------------------------------------------------------------ /opt/modulefiles ------------------------------------------------------------------
modules  openmpi/4.1.5  openmpi/4.1.5-onucx  papi/7.0.1  ucx/1.14.1  use.own
```

```
module load mpi or
module load mpi/2021.10.0
```

```
[kerwin@apollo31 lab1]$ module list
Currently Loaded Modulefiles:
 1) tbb/latest   2) compiler-rt/latest   3) oclfpga/latest   4) compiler/latest   5) mpi/latest

Key:
auto-loaded
```

Check this image

Compilation

- `mpicc/mpicxx` is an compiler wrapper that you could choose different c/c++ compiler to be it's backend
  - Using `-cc=<c compiler> for c and -cxx=<c++ compiler> for c++`
  - `gcc,g++` - GNU C/C++ compiler
  - `icx/icpx` - Intel C/C++ compiler (replace icc/icpc in 2024)
  - `clang/clang++` - Clang C/C++ compiler
- Compile the hello world program:
  - ➡ `mpicc -O3 hello.c -o hello`
- Different compilers implement performance optimization differently, leading to varying performance across platforms.

More information check Intel® MPI Library Developer Reference

Intel MPI Library Compiler Wrappers

Compiler Command	Default Compiler	Supported Languages
Generic Compilers		
mpicc	gcc, cc	C
mpicxx	g++	C/C++
mpifc	gfortran	Fortran77*/Fortran 95*
GNU* Compilers		
mpigcc	gcc	C
mpigxx	g++	C/C++
mpif77	gfortran	Fortran 77
mpif90	gfortran	Fortran 95
Intel® Fortran, C++ Compilers		
mpiicc	icc	C
mpiicx	icx	C
mpiicpc	icpc	C++
mpiicpx	icpx	C++
mpiifort	ifort	Fortran77/Fortran 95
mpiifx	ifx	Fortran77/Fortran 95

LSA-Lab

Run the hello world program

```
$ srun -n4 ./hello
```

Output:

```
Hello, World.  I am 3 of 4
Hello, World.  I am 1 of 4
Hello, World.  I am 2 of 4
Hello, World.  I am 0 of 4
```

Practices

Compile and run the hello world program.

Outline

- Platform introduction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

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()`

```
double starttime, endtime;
starttime = MPI_Wtime();
....  stuff to be timed  ...
endtime   = MPI_Wtime();
printf("That took %f seconds\n",endtime-starttime);
```

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

```c
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 / 1000000000.0;

   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.

Outline

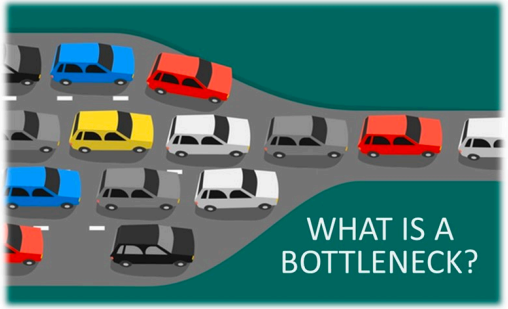
- Platform introduction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
  - IPM
  - mpiP
- Lab1 - Pixels in circle

Profile your MPI program - IPM

IPM is a portable profiling tool for parallel codes, focusing on communication, computation, and IO. It offers low-overhead performance metrics for both production and development use in HPC centers. Runtime-adjustable detail levels are available through text and web reports.

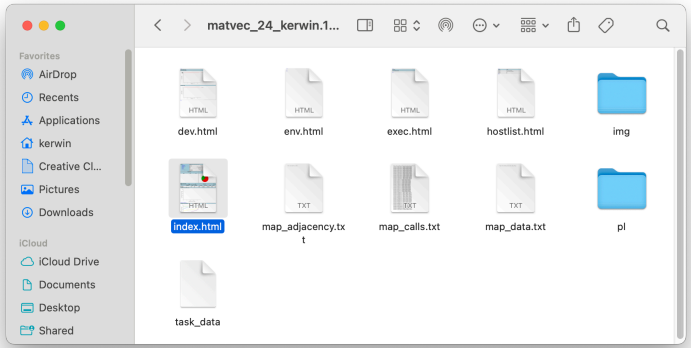
Importance of Profiling

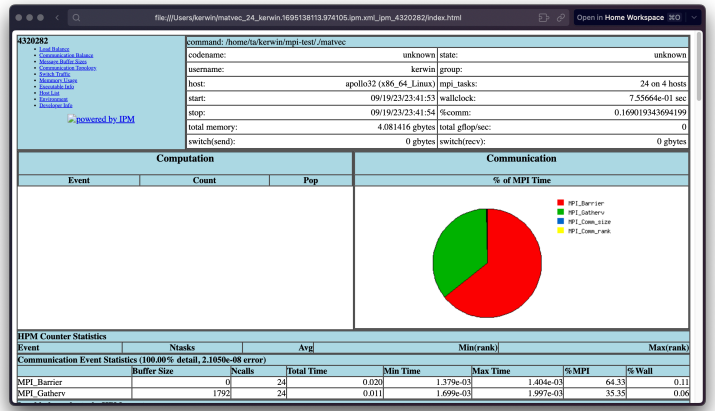
- Identify bottlenecks
- Understand data flow and communication patterns
- Optimize resource usage



Profile your MPI program - IPM

- `module load ipm` : Load IPM module
- Using `LD_PROLOAD` to attach profiler when you run your program
  - ➡ `IPM_REPORT=full IPM_REPORT_MEM=yes IPM_LOG=full LD_PRELOAD=/opt/ipm/lib/libipm.so srun -n<process> ./<yout program>`
  - ➡ `ipm_parse -html <output_file>.ipm.xml`
- Your can download your output dir and open the html file on your computer.





IPM with Performance Application Programming Interface (PAPI)

The Performance Application Programming Interface (PAPI) is a library that provides a standard way to collect performance metrics from hardware components (CPU cycle, Instruction counts). It helps developers understand and optimize software performance in relation to the underlying hardware.

- Use `papi_avail` to dump all available metrics

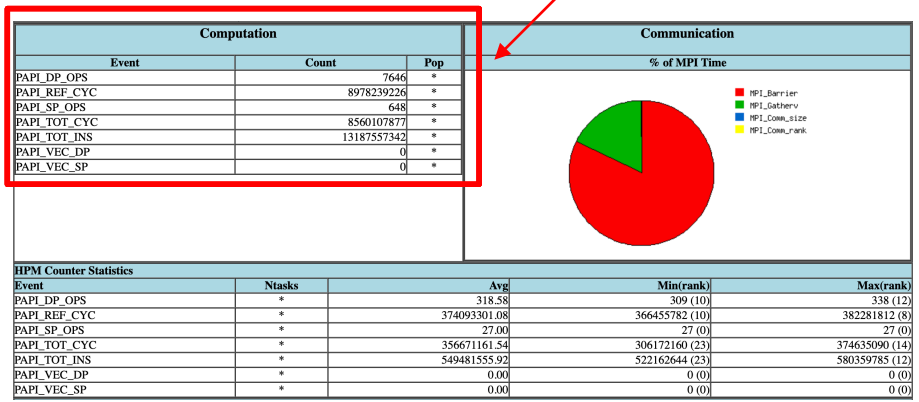
```
#!/bin/bash
#SBATCH -ptest
#SBATCH -N3
#SBATCH -n24

module load mpi

export IPM_REPORT=full
export IPM_REPORT_MEM=yes
export IPM_LOG=full
export LD_PRELOAD=/opt/ipm/lib/libipm.so
export IPM_HPM="PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_REF_CYC,\
PAPI_SP_OPS,PAPI_DP_OPS,PAPI_VEC_SP,PAPI_VEC_DP"

mpirun ./<your_program>
```

PAPI Event



Computation

Event	Count	Pop
PAPI_DP_OPS	7646	*
PAPI_REF_CYC	8978239226	*
PAPI_SP_OPS	648	*
PAPI_TOT_CYC	8560107877	*
PAPI_TOT_INS	13187557342	*
PAPI_VEC_DP	0	*
PAPI_VEC_SP	0	*

Communication

% of MPI Time

HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_DP_OPS	*	318.58	309 (10)	338 (12)
PAPI_REF_CYC	*	374093301.08	366455782 (10)	382281812 (8)
PAPI_SP_OPS	*	27.00	27 (0)	27 (0)
PAPI_TOT_CYC	*	356671161.54	306172160 (23)	374635090 (14)
PAPI_TOT_INS	*	549481555.92	522162644 (23)	580359785 (12)
PAPI_VEC_DP	*	0.00	0 (0)	0 (0)
PAPI_VEC_SP	*	0.00	0 (0)	0 (0)

In this case, we highly recommend using sbatch to submit jobs.
Here is the sbatch script example !!

LSA-Lab

Profile your MPI program - mpiP

mpiP is a light-weight profiling library for MPI applications. Because it only collects statistical information about MPI functions, mpiP generates considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local. It only uses communication during report generation, typically at the end of the experiment, to merge results from all of the tasks into one output file.

```
#!/bin/bash
#SBATCH -ptest
#SBATCH -N3
#SBATCH -n24

module load mpi

export MPIP="-y -l"
export LD_PRELOAD=/opt/mpiP/lib/libmpiP.so

mpirun ./<your_program>
```

```
[kerwin@apollo31 hw1]$ ls -al
total 2097356
drwxr-xr-x  2 kerwin kerwin        290 Sep 20 14:48 .
drwxr-xr-x 12 kerwin kerwin        213 Sep 15 04:17 ..
-rw-r--r--  1 kerwin kerwin          0 Sep 12 15:44 4318744.err
-rw-r--r--  1 kerwin kerwin          0 Sep 12 15:44 4318744.log
-rwxr-xr-x  1 kerwin kerwin      31464 Sep 18 03:31 hw1
-rw-r--r--  1 kerwin kerwin      40094 Sep 19 13:18 hw1.12.5419.1.mpiP
-rw-r--r--  1 kerwin kerwin      47420 Sep 18 22:30 hw1.15.4007.1.mpiP
-rw-r--r--  1 kerwin kerwin      13216 Nov 21  2021 hw1.cc
-rw-r--r--  1 kerwin kerwin      46449 Sep 19 23:31 kerwin.1695137444.992549.ipm.xml
-rw-r--r--  1 kerwin kerwin        160 Sep 15 17:45 Makefile
-rwxr-xr-x  1 kerwin kerwin        184 Sep 18 02:59 multi.sh
-rw-r--r--  1 kerwin kerwin 2147479552 Sep 19 23:31 out
-rwxr-xr-x  1 kerwin kerwin       7714 Sep  6 16:30 runner.py
-rw-r--r--  1 kerwin kerwin        712 Sep 12 15:44 sbatch.sh
```

Output file will be *.mpiP

LSA-Lab

Profile your MPI program - Understanding mpiP output report

- **Header information**
  - provides basic information about your performance experiment.
  - MPI_Time (by each processes) and Callsites.
- **Agregate information**
  - Including Time, Message Size, Collective time, P2P sent size and some statistics.
- **Callsites information**
  - I/O statistics and more....
- Revise the online document to understand the meaning of each section.

Outline

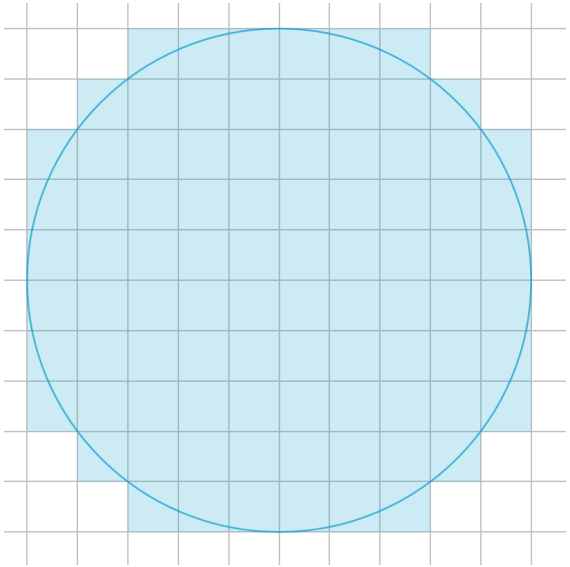
- Platform introduction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Profile your program
- Lab1 - Pixels in circle

Pixels in circle

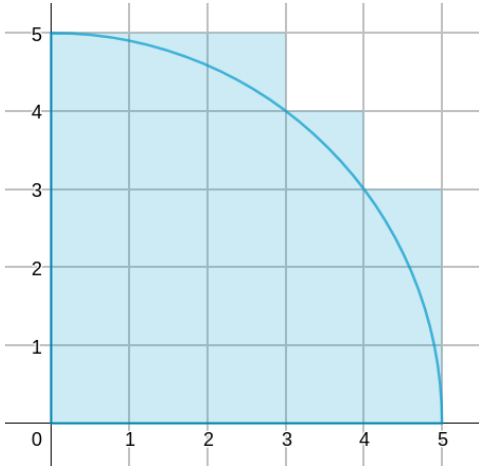
Suppose we want to draw a filled circle of radius r on a 2D monitor, how many pixels will be filled?

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 -Nnode -nproc ./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/pp23/share/lab1/sample`, copy these files to your home directory.
- All of the test cases can be found in `/home/pp23/share/lab1/testcases`
- Within the same directory of `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)
  - Due: 10/02 (Thu.) 23:59
- Full score for AC of all 12 test cases; otherwise, zero.

```
enmingw32@apollo31 ~/lab1-code> ls
lab1.cc  Makefile*
enmingw32@apollo31 ~/lab1-code> lab1-judge
Looking for lab1.cc: OK
Looking for Makefile: OK
Running: /usr/bin/make -C /home/pp22/enmingw32/.judge.225368503 lab1
make: Entering directory '/home/pp22/enmingw32/.judge.225368503'
mpicxx -std=c++17 -O3    lab1.cc    -o lab1
make: Leaving directory '/home/pp22/enmingw32/.judge.225368503'
04.txt    1.02   accepted
03.txt    1.07   accepted
01.txt    0.82   accepted
02.txt    0.77   accepted
05.txt    1.22   accepted
06.txt    1.32   accepted
07.txt    1.27   accepted
08.txt    1.27   accepted
09.txt    1.52   accepted
10.txt    1.97   accepted
11.txt    1.92   accepted
12.txt    2.57   accepted
Removing temporary directory /home/pp22/enmingw32/.judge.225368503
Scoreboard: not updating {12 15.90} -x-> {12 16.72}
```

Note !!

- If you have any problem on using Apollo Cluster:
  - Google or ChatGPT first, most common question is all about your personal setting.
  - If you really don't know what error you're encountering, or if you suspect that we're experiencing hardware failure or connectivity issues, please take a screenshot of your error message and make note of your Slurm job ID (if you're using Slurm).
  - Send an email with the subject "About Apollo Environment" to pp@lsalab.cs.nthu.edu.tw
- Do not attack server !! (including using multi processes to lunch judge.)
- We have several system monitors:
  - Cluster monitor
  - SLURM monitor
  - Infiniband monitor
- Any other question, just send a mail to pp@lsalab.cs.nthu.edu.tw