Lab1
Platform introduction & MPI

Parallel Programming
2020/09/24

Outline

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

Platform instruction - Apollo

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

Software

- OS: Arch Linux
- Compilers: GCC 10.1.0, Clang 10.0.1
- MPI: Intel MPI Library, Version 2019 Update 8
- Scheduler: Slurm 20.02.4

Available resources

- 1 login node (apollo31) (200%CPU max)
- 16 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
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Lab1 - Pixels in circle

Login to Apollo

- Address: apollo.cs.nthu.edu.tw
- Username: check email
- Password: check email

SSH - Linux and Mac

- Open terminal
- `ssh pp20sXX@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)
- `ssh pp20sXX@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
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- Lab1 - Pixels in circle

Some useful command

- Login: `ssh pp20sXX@apollo.cs.nthu.edu.tw`
- File transfer:
  `rsync -avhP filename pp20sXX@apollo.cs.nthu.edu.tw:filename`
- Editors: `vim` `emacs` `nano`
- SLURM usage: `squeue`
- Disk quota: `quota -s`
- Change password: `passwd`
- Download file: `wget` `aria2c`
- Code syntax highlighting: `pygmentize`

Outline

- Platform introduction - Apollo
- Login to Apollo
- Linux command
- MPI hello world
- Compile and job submission
- Time measurement
- 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 down this code directly on apollo.

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

Outline

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

Compilation

- `gcc` - C compiler
- `g++` - C++ compiler
- `mpicc` - MPI C compiler wrapper
- `mpicxx` - MPI C++ compiler wrapper
- `mpicc` and `mpicxx` actually call `gcc` and `g++`
- You can use `clang` instead by add the flags `-cc=clang` for c and `-cxx=clang++` for c++
- Compile the hello world program:
  ```
  mpicc -O3 hello.c -o hello
  ```

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
```

Job submission

SLURM workload scheduler: 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 ensure 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

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 number of total processes to launch
  - `-c CPUS`: CPUS is the number of cpus available to each process
  - `-t TIME`: The time limit in "minutes" or "minutes:seconds"
  - `-J NAME`: The name of the job. Will be displayed on squeue

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`

Job control

- `sinfo`: view status of nodes
- `squeue`: view submitted jobs in queue
- `scancel JOBID`: cancel a job with its JOBID

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
- 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`

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, ...)

```
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
- Lab1 - Pixels in circle

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

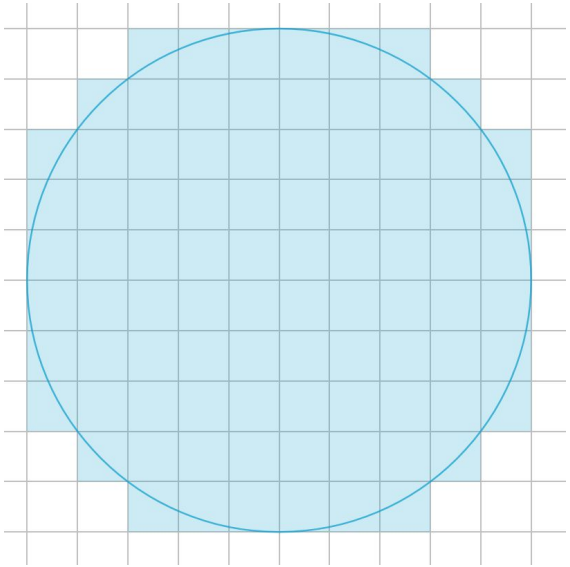
```
int MPI_Reduce(const void *sendbuf,
               void *recvbuf,
               int count,
               MPI_Datatype datatype,
               MPI_Op op,
               int root,
               MPI_Comm comm)
```

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.

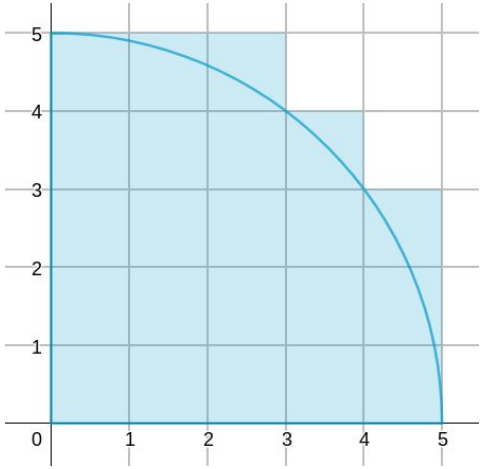
For example 88 pixels are filled when r=5.



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/pp20/share/lab1/sample`, copy these files to your home directory.
- All the test cases can be found in `/home/pp20/share/lab1/testcases`
- Within the same directory of `lab1.cc` and `Makefile`, run `lab1-judge` to check.
- Scoreboard
- Submit your code to iLMS:
  - lab1.cc
  - Makefile (optional, if you change any compile flags)
  - Due 10/01 23:59
- Full score for AC in all 12 test cases, otherwise zero.