COMP4901Q: High Performance Computing (HPC)

Tutorial 3:
Programming with MPI on clusters

TA: Mingkai TANG(mtangag@connect.ust.hk)

Contents

- Part 1: Programming environment setup

- Part 2: Programming with MPI
  - Demo / Compile / Debug / Run

Part 1: Programming Environment Setup

Environment Setup

- Setup SSH passwordless login between nodes
- Setup OpenMPI environment

SSH Passwordless Login Between Nodes

- 1. Generate key
- `$ ssh-keygen -t rsa -b 4096`
- `#repeatedly press <enter> until finish`

- 2. Add your RSA key to authorized_keys file:
- `$ touch ~/.ssh/authorized_keys`
- `$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`

SSH Passwordless Login Between Nodes

- 3. Use ssh to access the server you want to use. The server will add into your “known_hosts”
- $ ssh csl2wk02
- # type <yes> in your terminal.
- # you should login csl2wk03 successfully,
- # please type <exit> in you terminal to back to your working workstation.
- # Repeat above steps if you want to add more nodes to the `known_hosts`

- For example, on csl2wk01, we add csl2wk02 & csl2wk03 & csl2wk04 into “known_hosts”

```
csl2wk01:mtangag:99> ls ~/.ssh/
authorized_keys  id_rsa  id_rsa.pub  known_hosts
```

SSH FS - cs12wk01 > homes > mtangag > .ssh > known_hosts

```
1  csl2wk02,143.89.238.2 ecdsa-sha2-nistp256 AAAAE2VjZHN
2  csl2wk03,143.89.238.3 ecdsa-sha2-nistp256 AAAAE2VjZHN
3  csl2wk04,143.89.238.4 ecdsa-sha2-nistp256 AAAAE2VjZHN
4
```

Check & Install OpenMPI

- Installed on CSLab2 servers
  - mpiexec --version
  - We use OpenMPI-3.0.0 (/usr/local/software/openmpi)

```
csl2wk02:mtangag:44> mpirun --version
mpirun (Open MPI) 3.0.0

Report bugs to http://www.open-mpi.org/community/help/
csl2wk02:mtangag:45> which mpirun
/usr/local/software/openmpi/bin/mpirun
csl2wk02:mtangag:46> ls -al /usr/local/software/openmpi
lrwxrwxrwx 1 root root 13 Dec 28  2017 /usr/local/software/openmpi -> openmpi-3.0.0
```

- If not found, set the PATH:
- echo 'setenv PATH "${PATH}:/usr/local/software/openmpi/bin"'>> ~/.cshrc_user
- source ~/.cshrc_user

- If you want to install OpenMPI in your own computer
  - With package manager apt, yum, etc.
  - Build from source code: link

Part 2: Programming With MPI

Demo / Compile /Run / Debug

MPI

- MPI (Message Passing Interface) is a library specification for message-passing. MPI consists of
  - a header file mpi.h
  - a library of routines and functions, and
  - a runtime system.
- MPI is for parallel computers, clusters, and heterogeneous networks.
- MPI can be used with C/C++, Fortran, and many other languages.
- MPI is actually just an Application Programming Interface (API).

Example MPI Routines

- The following routines are found in nearly every program that uses MPI:
  - MPI_Init() starts the MPI runtime environment.
  - MPI_Finalize() shuts down the MPI runtime environment.
  - MPI_Comm_size() gets the number of processes, N_p.
  - MPI_Comm_rank() gets the process ID of the current process which is between 0 and $N_p - 1$, inclusive.
- These last two routines are typically called right after MPI_Init().

Demo 1(MPIHello1)

```
#include<stdio.h>
#include<mpi.h>
int main (int argc, char *argv[])
{
   int rank;
   int number_of_processes;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD , &number_of_processes);
   MPI_Comm_rank(MPI_COMM_WORLD , &rank);
   printf("hello from process %d of %d\n", rank, number_of_processes);
   MPI_Finalize();
   return 0;
}
```

1. int MPI_Init(int *argc_p, char **argv_p)
2. int MPI_Comm_size(MPI_Comm comm, int* comm_sz_p)
3. int MPI_Comm_rank(MPI_Comm comm, int* my_rank_p)
4. int MPI_Finalize(void)

Compilation

- mpic++ -Wall -o MPIHello1 MPIHello1.cpp
  - Mpic++: wrapper script to compile for C++, mpicc for C Language
  - -Wall: turns on all warnings
  - -o: create this executable file name (as opposed to default a.out)

Single Machine Execution

- On single machine:
- mpiexec -n 2 MPIHello1
  - -n: number of processes, usually related to number of cores
    - If the number of slots needed is larger than the number of cores:
    - mpiexec is the same as mpirun
    - Use --oversubscribe. It will be degraded!!
    - mpiexec --oversubscribe -n 4 MPIHello1

More Example MPI Routines

- Some of the simplest and most common communication routines are:
- Point-to-point communication:
  - MPI_Send() sends a message from the current process to another process (the destination).
  - MPI_Recv() receives a message on the current process from another process (the source).
- Collective communication
  - MPI_Bcast() broadcasts a message from one process to all of the others.
  - MPI_Reduce() performs a reduction (e.g. a global sum, maximum, etc.) of a variable in all processes, with the result ending up in a single process.
  - MPI_Allreduce() performs a reduction of a variable in all processes, with the result ending up in all processes.

Demo 2(MPIHello2)

```
int main(void) {
  char greeting[100];
  int comm_sz;
  int my_rank;
  MPI_Init((&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  if (my_rank != 0) {
    sprintf(greeting, "Greetings from process %d of %d!",
        my_rank, comm_sz);
    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
        MPI_COMM_WORLD);
  } else {
    printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
    for (int q = 1; q < comm_sz; q++) {
      MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
        0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      printf("%s\n", greeting);
    }
  }
  MPI_Finalize();
  return 0;
}
```

1. int MPI_Send(void* buf, int count, MPI_Datatype datatype,
   int dest, int tag, MPI_Comm comm);
2. int MPI_Recv(void* buf, int maxsize, MPI_Datatype datatype, int source, int
   tag, MPI_Comm comm, MPI_Status* status_p);

Sum

- Calculate the following formula
- $\sum_{\mathrm{i}=0}^{n-1} \sum_{j=0}^{n-1} a[i] / a[j]$
- Where $a[i] = i\%10 + 1$

Sum

- For master process:
  - Initializes arrays
  - Distributes the portion of array to child processes to calculate their partial sums
  - Adds its own partial sums
  - Collects partial sums from other processes
  - Returns the final sum

Sum

- For slave process:
  - Receives array.
  - Adds its own sub array.
  - Sends its partial sum back to the master process.

Sum

```
int main(int argc, char **argv)
{
    int n, number_of_processes, rank;
    double *a, start_time;
    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &number_of_processes);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        printf("Please input n:\n");
        scanf("%d",&n);
        a = new double[n];
        for (int i = 0; i<n; i++) {
            a[i] = i % 10 + 1;
        }
    }
```

Read n in process 0.

Sum

```
start_time = MPI_Wtime();
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

if (rank != 0) {
   a = new double[n];
}

MPI_Bcast(a, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

int elements_per_process = ceil(n * 1.0 / number_of_processes);
int init = elements_per_process * rank;
double local_sum = 0;
for (int i = init; i < init + elements_per_process && i < n; i++)
   for (int j = 0; j < n; j++)
      local_sum += a[i] / a[j];

printf("Local sum for process %d - %f\n",
          rank, local_sum);
```

Get the time.

Broadcast the array to other process.

```
int MPI_Bcast(void *buf, int count,
                  MPI_Datatype datatype,
                  int root, MPI_Comm comm)
```

Calculate the local sum.

Sum

```
    double global_sum;
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE,
                        MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0)
    {
        printf("Total sum = %f\n", global_sum);
        double finish_time = MPI_Wtime();
        printf("Elapsed time is %f seconds\n", finish_time-start_time);
    }
    delete a;
    MPI_Finalize();
    return 0;
}
```

Combine the result of all processes.

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

Output the result.

Multi Machine Execution

- Running by 1 process on single machine
- mpiexec -n 1 Sum

```
csl2wk01:mtangag:129> mpiexec -n 1 Sum
Please input n:
20000
Local sum for process 0 - 644373015.859192
Total sum = 644373015.859192
Elapsed time is 1.499913 seconds
```

- Running by 2 process on single machine
- mpiexec -n 2 Sum

```
csl2wk01:mtangag:133> mpiexec -n 2 Sum
Please input n:
20000
Local sum for process 1 - 322186507.918950
Local sum for process 0 - 322186507.918950
Total sum = 644373015.837899
Elapsed time is 0.673603 seconds
```

- Running by 4 process on single machine
- mpiexec --oversubscribe -n 4 Sum
- The running time cannot decrease.

```
csl2wk01:mtangag:136> mpiexec -n 4 --oversubscribe Sum
Please input n:
20000
Local sum for process 0 - 161093253.968544
Local sum for process 1 - 161093253.968544
Local sum for process 2 - 161093253.968544
Local sum for process 3 - 161093253.968544
Total sum = 644373015.874176
Elapsed time is 0.702187 seconds
```

Multi Machine Execution

- Execution on Multi Machine
- mpiexec -n 4 --hostfile hostfile Sum
  - --hostfile: Provide a hostfile to use. (hostname and slot numbers)

SSH FS - csl2wk01 > homes > mtangag > lab3-sample-code > hostfile

```
csl2wk01 slots=2
csl2wk02 slots=2
csl2wk03 slots=2
csl2wk04 slots=2
```

```
csl2wk01:mtangag:142> mpiexec -n 4 --hostfile hostfile Sum
Please input n:
20000
Local sum for process 3 - 161093253.968544
Local sum for process 0 - 161093253.968544
Local sum for process 2 - 161093253.968544
Local sum for process 1 - 161093253.968544
Total sum = 644373015.874176
Elapsed time is 0.338579 seconds
```

  - It can run 8 process at most.
  - Note that at environment setup step, we only add csl2wk02, csl2wk03 and csl2wk04 into “known_hosts”. But in the hostfile we use csl2wk01. In this situation, we only can run the program on csl2wk01, otherwise it will cause hang up.

A possible Error On Windows

- If you are on windows, don’t use the text editor to modify the hostfile in the server, otherwise you will get an error when running the program.

```
--------------------------------------------------------------------------
Open RTE detected a parse error in the hostfile:
    host_file
It occured on line number 37 on token 1.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
An internal error has occurred in ORTE:

[[59525,0],0] FORCE-TERMINATE AT (null):1 - error base/ras_base_allocate.c(302)

This is something that should be reported to the developers.
--------------------------------------------------------------------------
```

- When use “cat –A” command to watch the hostfile. You will see some “^M” at the end of the file. It’s because on Windows, it uses “\r\n” to end a line but on Linux, it uses “\n”.

```
csl2wk02:mtangag:51> cat -A hostfile
csl2wk01 slots=2^M$
csl2wk02 slots=2^M$
csl2wk03 slots=2^M$
csl2wk04 slots=2^M$
```

- You can use vi to edit the file or use some commands like
- echo "csl2wk02 slots=2" >> hostfile

Debug

- Commercial software, eg. totalview
- Serial debuggers (such as gdb)? Yes. (FAQ on MPI debugging item #6)
  - Attach to individual MPI processes after they are running.
  - Use mpirun to launch separate instances of serial debuggers.
- Use printf() to print key information

Debug

- When compiling, add -g
  - mpic++ -g MPIHello1_debug.cpp -o MPIHello1_debug
- Attach to individual MPI processes after they are running.
  - Modify codes,
  - **1.** add header **#include <unistd.h>:**
  - **2.** add code:

```
volatile int debug = 0;
char hostname[256];
gethostname(hostname, sizeof(hostname));
printf("PID %d on %s ready for attach\n", getpid(), hostname);
fflush(stdout);
while (0 == debug)
sleep(5);
```

Debug

- Attach to individual MPI processes after they are running.
  - In one session, execute the program
    - mpiexec -n 2 MPIHello1_debug
  - In another session, attach one of the processes with gdb
    - gdb attach $PID
  - Once you attach with a debugger, go up the function stack until you are in this block of code (you'll likely attach during the sleep()) then set the variable debug to a nonzero value. With GDB, the syntax is:

```
(gdb) set var debug = 7
```

  - Continue debugging

Debug

- Attach to individual MPI processes after they are running.

```
csl2wk01:mtangag:185> mpicc -g MPIHello1_debug.cpp -o MPIHello1_debug
csl2wk01:mtangag:186> mpirun --np 2 MPIHello1_debug
PID 2166 on csl2wk01 ready for attach
PID 2167 on csl2wk01 ready for attach
```

```
csl2wk01:mtangag:19> gdb attach 2166
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
```

```
(gdb) n
Single stepping until exit from function nanosleep,
which has no line number information.
0x00007f3b643c6894 in sleep () from /lib64/libc.so.6
(gdb) n
Single stepping until exit from function sleep,
which has no line number information.
main (argc=1, argv=0x7fffc5715488) at MPIHello1_debug.cpp:18
18          while (0 == debug)
(gdb) set var debug = 7
(gdb) n
21          printf("hello from process %d of %d\n", rank, number_of_processes);
(gdb) n
22          MPI_Finalize();
```

Debug

- Use mpirun to launch separate instances of serial debuggers.
  - With Linux GUI
    - mpirun -np 4 xterm -e gdb my_mpi_application
  - No GUI, but tmux
- On CSLab2 machines, we use tmux, wrapped in script tmpi
  - Download the script from tmpi
  - chmod +x tmpi
  - tmpi 2 gdb my_mpi_application

Debug

Step 1: Compile.
mpic++ -g MPIHello1.cpp -o MPIHello1

Step2: Run tmpi.
tmpi 2 gdb MPIHello1

Step 3:Add breakpoint.
b 9

Step 4:Run.
r

Step 5:Next step.
n

Step 6:Print variable.
p rank

```
(gdb) b 9
Breakpoint 1 at 0x400810: file MPIHello1.cpp, line 9.
(gdb) r
Starting program: /homes/mtangag/lab3-sample-code/MPIHello1
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7ffff52f7700 (LWP 13082)]
[New Thread 0x7fffeffff700 (LWP 13092)]

Breakpoint 1, main (argc=1, argv=0x7fffffffd8d8) at MPIHello1.cpp:9
9           MPI_Comm_rank(MPI_COMM_WORLD , &rank);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-325.el7_9.x86_64 libgcc-4.8.5-44.el7.x86_64 libs
tdc++-4.8.5-44.el7.x86_64 nvidia-driver-latest-dkms-cuda-libs-470.57.02-1.el7.x86_64 zlib-1.2.7-19.el7_9.x86_64
(gdb) n
10          printf("hello from process %d of %d\n", rank, number_of_processes);
(gdb) p rank
$1 = 0
(gdb)
```

```
Breakpoint 1 at 0x400810: file MPIHello1.cpp, line 9.
(gdb) r
Starting program: /homes/mtangag/lab3-sample-code/MPIHello1
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7ffff52f7700 (LWP 13083)]
[New Thread 0x7fffeffff700 (LWP 13093)]

Breakpoint 1, main (argc=1, argv=0x7fffffffd8d8) at MPIHello1.cpp:9
9           MPI_Comm_rank(MPI_COMM_WORLD , &rank);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-325.el7_9.x86_64 libgcc-4.8.5-44.el7.x86_64 libs
tdc++-4.8.5-44.el7.x86_64 nvidia-driver-latest-dkms-cuda-libs-470.57.02-1.el7.x86_64 zlib-1.2.7-19.el7_9.x86_64
(gdb) n
10          printf("hello from process %d of %d\n", rank, number_of_processes);
(gdb) p rank
$1 = 1
```

Debug

- Use printf() to print key information
  - Refer to Sum.cpp, where local sum is printed in each process.

Debug: Advantages and Disadvantages

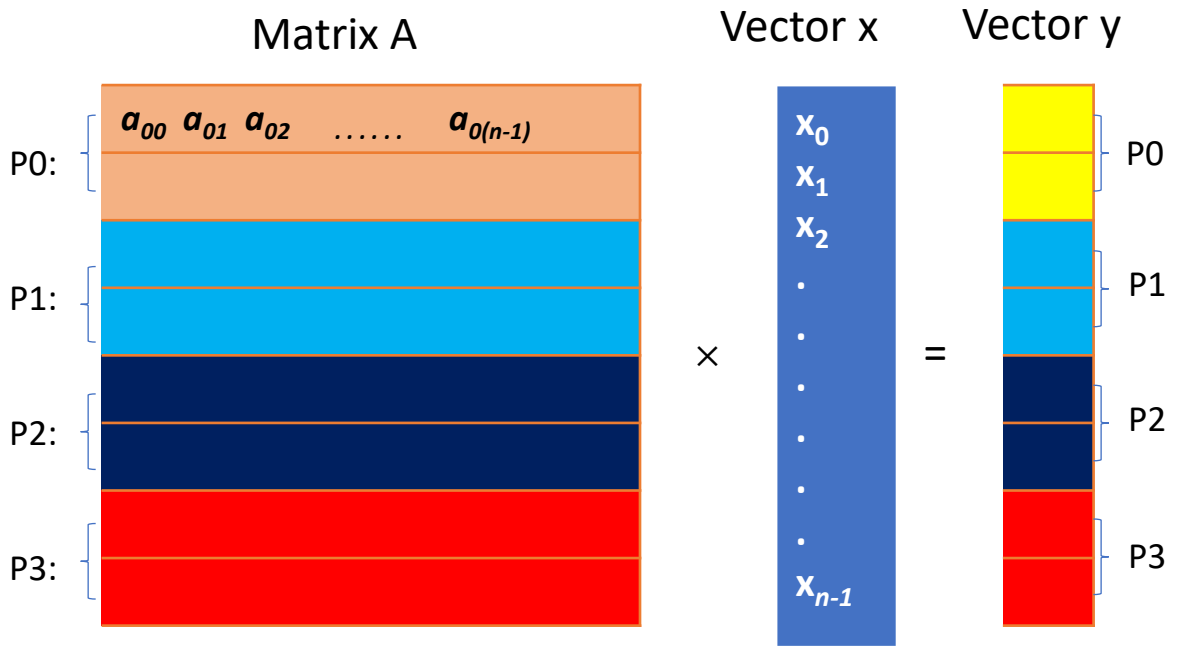
	Commercial software	**GDB: Attach to individual MPI processes after they are running.**	**GDB: Use mpirun to launch separate instances of serial debuggers.**	**printf()**
Advantages	Easy to use	Free	Free, no modification	Free, easy
Disadvantages	Expensive	Modify codes Debug one process every time	May handle many windows	May add lots of output statements

Recall: Matrix-Vector-Multiplication

- Description: cross-multiplying a matrix by a vector in parallel.
- Complete codes in MVMul.cpp

Row-wise 1-D Partitioning

- Given p processes, Matrix A (m x n) is partitioned into p smaller matrices, each with dimension (m/p x n).
  - For simplicity, we assume p divides m (or, m is divisible by p).
  - Or we deal with the last process (portion) separately.



Parallel Matrix-Vector Multiplication: Framework

- Assumptions
  - A total of p processes
  - Matrix A (m x n) and vector x (n x 1) are created at process 0
    - called “master process” because it coordinates the work of other processes (i.e., “slave processes”)
- Message passing:
  - Process 0 will send (p-1) sub-matrices to corresponding processes
  - Process 0 will send vector x to all other p-1 processes
- Calculations:
  - Each process carries out its own matrix-vector multiplication
- Message passing:
  - Processes 1 to (p-1) send the results (i.e., part of vector y) back to process 0