COMP 4007: Parallel Processing and Computer Architecture

Tutorial 3:
Programming with MPI on clusters

TA: Hucheng Liu (huchenglew@qq.com)

Contents

- Part 1: Programming environment setup

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

Part 1: Programming Environment Setup

Environment Setup

- Install OpenSSH-Server and Setup hostname
  - sudo bash ./install_openssh_server.sh
- Setup SSH passwordless login between nodes
- Install OpenMPI if not

SSH Passwordless Login Between Nodes

- Generate SSH key pair
  - ssh-keygen
- Propagating the public key to a remote server
  - ssh-copy-id
  - Or just append the public key to ~/.ssh/authorized_keys
- Test
  - ssh -o StrictHostKeyChecking=no $HOST

SSH Passwordless Login Between Nodes Cont.

- For example, we use 2 nodes: e.g., node1 & node2

```
[redacted]:121> ssh-keygen -q -t rsa -N "" -f ~/.ssh/id_rsa
[redacted]:122> ls ~/.ssh/
authorized_keys  id_rsa  id_rsa.pub  known_hosts
```

```
[redacted]:126> ssh-copy-id [redacted]
/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/homes/[redacted]/.ssh/id_rsa.pub"
The authenticity of host [redacted] can't be established.
ECDSA key fingerprint is SHA256:+e9a1bgdyfbVywViBGVj1Oc3ELjr+VFFMzSSrlcowCQ.
ECDSA key fingerprint is MD5:f6:92:1f:c4:e9:e5:b0:fe:dd:c4:17:fc:50:18:79:2c.
Are you sure you want to continue connecting (yes/no)? yes
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to i
nstall the new keys
Password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh '[redacted]'"
and check to make sure that only the key(s) you wanted were added.
```

Check & Install OpenMPI

- Installed on T2210 servers
  - Test if mpi is installed: mpirun –version

- If not, install
  - 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().

MPI Hello World

```
#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;
}
```

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.

Sum of Array (1/3)

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

Sum of Array (2/3)

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

Sum of Array (3/3)

- With point-to-point communications
  - SumArrayP2P.c
- With collective communications
  - SumArrayCol.c

Compilation

- mpicxx -Wall -lm -o SumArrayCol SumArrayCol.c
  - mpicxx: mpic++ compiler for C++
  - -Wall: turns on all warnings
  - -lm: link lib math.h (SumArrayCol.c includes math.h)
  - -o: create this executable file name (as opposed to default a.out)

Execution

- On single machine:
- mpirun -n 2 SumArrayCol
- mpiexec -n 2 SumArrayCol
  - -n: number of processes, usually related to number of cores
    - If the number of slots needed is larger than the number of cores:
    - --oversubscribe

Execution

- Distributed:
- Mpirun --wd /home/lenovo/Lab3 --hostfile hostfile SumArrayCol
- mpiexec --wd /home/lenovo/Lab3 --hostfile hostfile SumArrayCol
  - --hostfile: Provide a hostfile to use. (hostname and slot numbers)
  - --wd: Assign the working directory

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
  - mpicxx -g MPIHello_debug.c -o MPIHello_debug
- Attach to individual MPI processes after they are running.
  - Modify codes, add:

```
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);
```

  - mpirun --np 2 MPIHello_debug

Debug

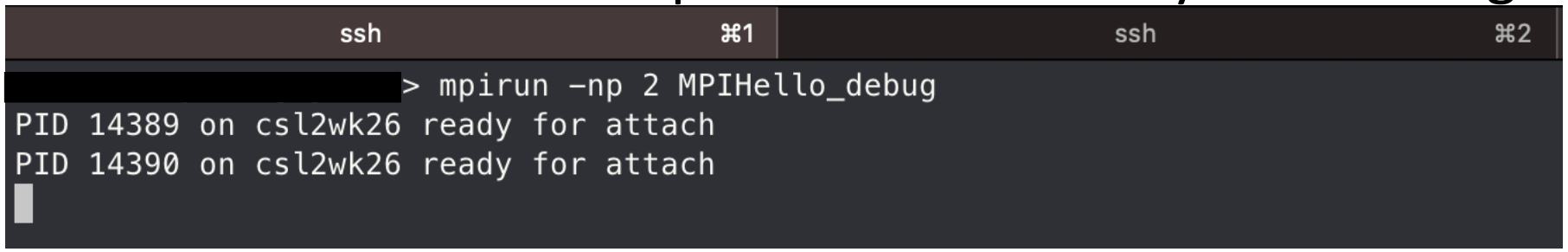
- Attach to individual MPI processes after they are running.
  - Modify codes, add:
  - In one session, execute the program
    - mpirun --np 2 MPIHello_debug
  - In another session, attach one of the processes with gdb
    - gdb attach $PID
  - Once you attach with a debugger, go up the function stack 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:

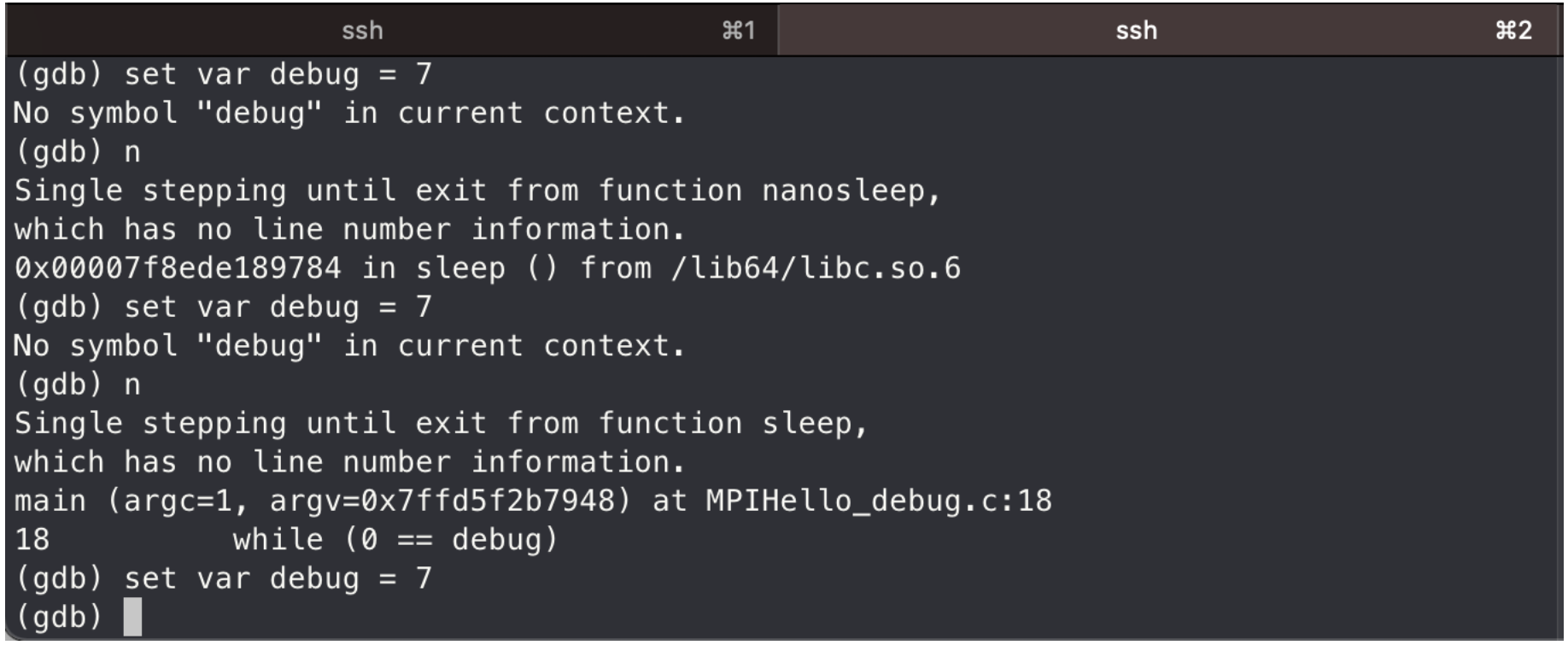
    (gdb) set var debug = 7

  - Continue debugging

Debug

- Attach to individual MPI processes after they are running.





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
  - tmpi 2 gdb MPIHello

Debug

```
Breakpoint 1, main (argc=1, argv=0x7fffffffd958) at MPIHello.c:9
9           MPI_Comm_rank(MPI_COMM_WORLD , &rank);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-323.el7_9.x86_6
4 nvidia-driver-latest-dkms-cuda-libs-460.27.04-1.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, main (argc=1, argv=0x7fffffffd958) at MPIHello.c:9
9           MPI_Comm_rank(MPI_COMM_WORLD , &rank);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-323.el7_9.x86_6
4 nvidia-driver-latest-dkms-cuda-libs-460.27.04-1.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
(gdb)
```

Debug

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

Debug: Advantages and Disadvantages

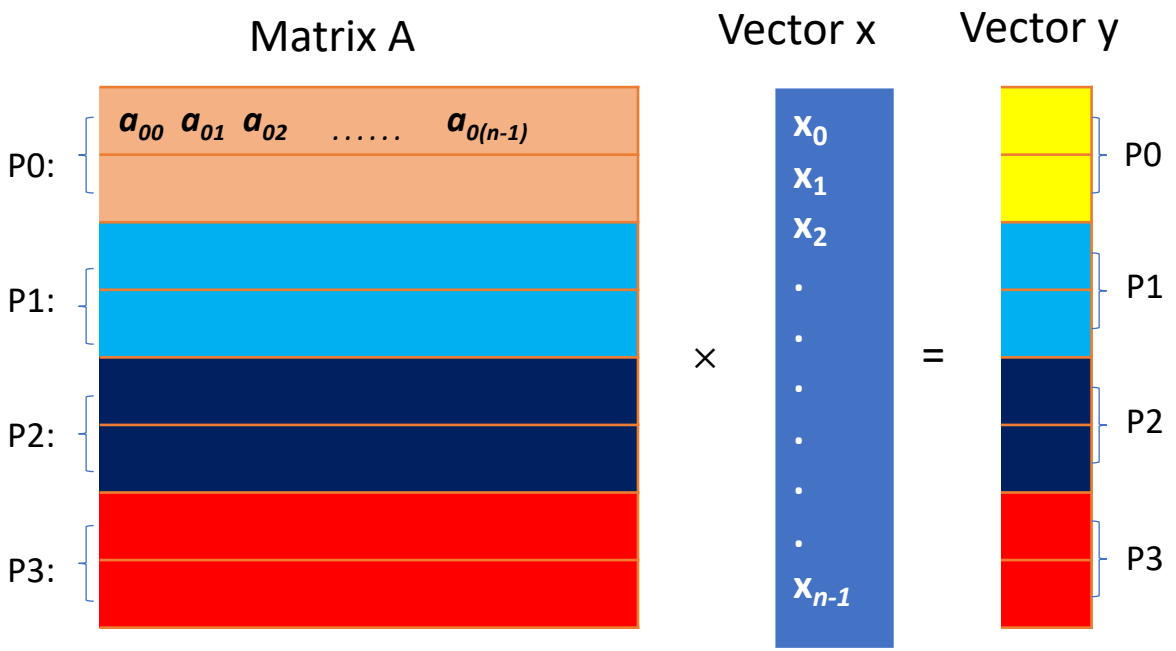
	Commercial software	**GDB: Attach to individual MPI processes after they are running.**	**GDB: Use mpirun to launch separate instances of serial debuggers.**	**printf()**
Advantages	Easy to use	Free	Free, no modification	Free, easy
Disadvantages	Expensive	Modify codes Debug one process every time	May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

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