MPI – Message passing interface

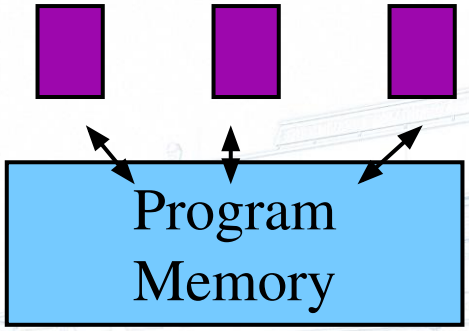
Advanced Programming in C/C++

14 . 01 . 2019

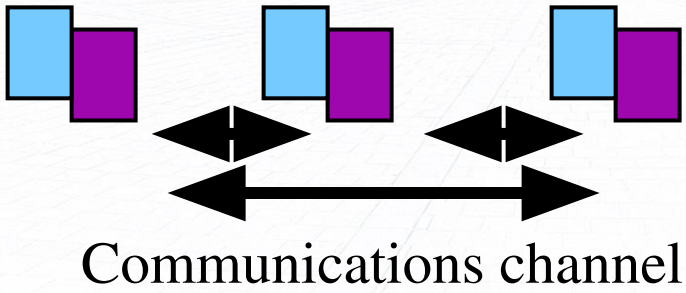
BY: Narges Mehran & Dr. Dragi Kimovski

Shared vs. Distributed Memory Programming

Shared Memory – All tasks access the same memory, hence the same data - **OpenMP**.



Distributed Memory – All memory is local. Data sharing is by explicitly transporting data from one task to another – **MPI**.



MPI

- Message Passing Interface (MPI) is a message-passing standard created to function on a wide variety of parallel architectures.

- It is standardized in collaboration between industry and academia.

- There are multiple implementations of the standard, such as:
  - MPICH: http://www.mpich.org/
  - OpenMPI: https://www.open-mpi.org/

A Simple Example

- A serial program

```
#include<stdio.h>
#define PID 0
main()
{
    int i;
    printf("Greetings from process %d!/n", PID);
}
```

Output: Greetings from process 0

A Simple Example (cont.)

- A parallel program using MPI

```
#include<mpi.h>
main(int argc, char** argv){
  ...
  MPI_Init(&argc, &argv); /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD, &size); /* Get the number of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* Get my process number (rank) */

                Parallel Region

  MPI_Finalize(); /* Exit MPI */
}
```

A Simple Example (cont.)

- MPI_Comm_size: Determines the size of the group associated with a communicator (the number of processes)
  - int MPI_Comm_size(MPI_Comm MPI_COMM_WORLD, int *size)
- MPI_Comm_rank: Determines the rank of the calling process in the communicator (a number between 0 and size-1)
  - int MPI_Comm_rank(MPI_Comm MPI_COMM_WORLD, int *rank)
- Sending a message:
  - int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm MPI_COMM_WORLD)
  - “buf” is the address of the message to be sent, with “count” elements of type “datatype”
- Receiving a message:
  - int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm MPI_COMM_WORLD, MPI_Status *status)

A Simple Example (cont.)

- A parallel program using MPI (cont.)

```
....
int rank, size, tag, rc, i;
MPI_Status status;
char message[20];
if (rank==0)
{
        strcpy(message, "Hello, world");
        for (i=1;i<size;++i)
                rc = MPI_Send(message, 13, MPI_CHAR, i, tag,
MPI_COMM_WORLD);
}
else
        rc = MPI_Recv(message, 13, MPI_CHAR, 0, tag, MPI_COMM_WORLD,
&status);
printf("Greetings from process %d of %d\n", rank, size);
```

A Simple Example (cont.)

- A parallel program using MPI (cont.)

  Output:

  Greetings from process 0 of 3

  Greetings from process 1 of 3

  Greetings from process 2 of 3

MPICH

- A freely available, portable implementation of MPI on Linux clusters
- Primarily developed by Argonne National Laboratory in the USA
- http://www.mpich.org/
- MPI Routines and Constants:
  - https://www.mpich.org/static/docs/v3.3/

Installation

- Installing MPICH:

```
>>> tar -xzf mpich-3.3.tar.gz
>>> cd mpich-3.3
>>> ./configure --disable-fortran
>>> make; sudo make install
>>> mpiexec - -version
        HYDRA build details: Version: 3.3
        Release Date: Wed Nov 21 11:32:40 CST 2018
        CC: gcc
        CXX: g++
        F77:
        F90:
```

Compilation

- GNU Compiler Collection (GCC) and MPICH
- In Linux Terminal, enter one of the following commands:
  - mpicc mpi_code.c -o mpi_code.out
  - mpicxx mpi_code.cc -o mpi_code.out

Compilation (cont.)

```c
#include <stdio.h>
#include "mpi.h"
int main(int argc, char** argv)  {
   MPI_Init(&argc, &argv);
   printf("Hello world \n");
   MPI_Finalize();                }
```

```fortran
program Hello_World
include 'mpif.h'
integer ierror
call MPI_INIT(ierror)
print *, 'Hello World'
call MPI_FINALIZE(ierror)
end
```

```cpp
#include "mpi.h"
#include <iostream>
int main(int argc, char** argv)  {
   MPI::Init(argc, argv);
   cout <<"Hello world \n";
   MPI::Finalize();               }
```

Compilers

C

```
$ mpicc
```

Fortran

```
$ mpif90
```

C++

```
$ mpicxx
```

Execution

- MPI (using multi-core processors)
  - mpdboot
  - mpiexec -n #processes ./ mpi_code.out
  - mpirun
- Running hellow
  - mpiexec -n 3 --hosts Narges,localhost hellow.out
  - mpirun -np 3 --hosts Narges,localhost ./hellow.out

Let's start coding your program!

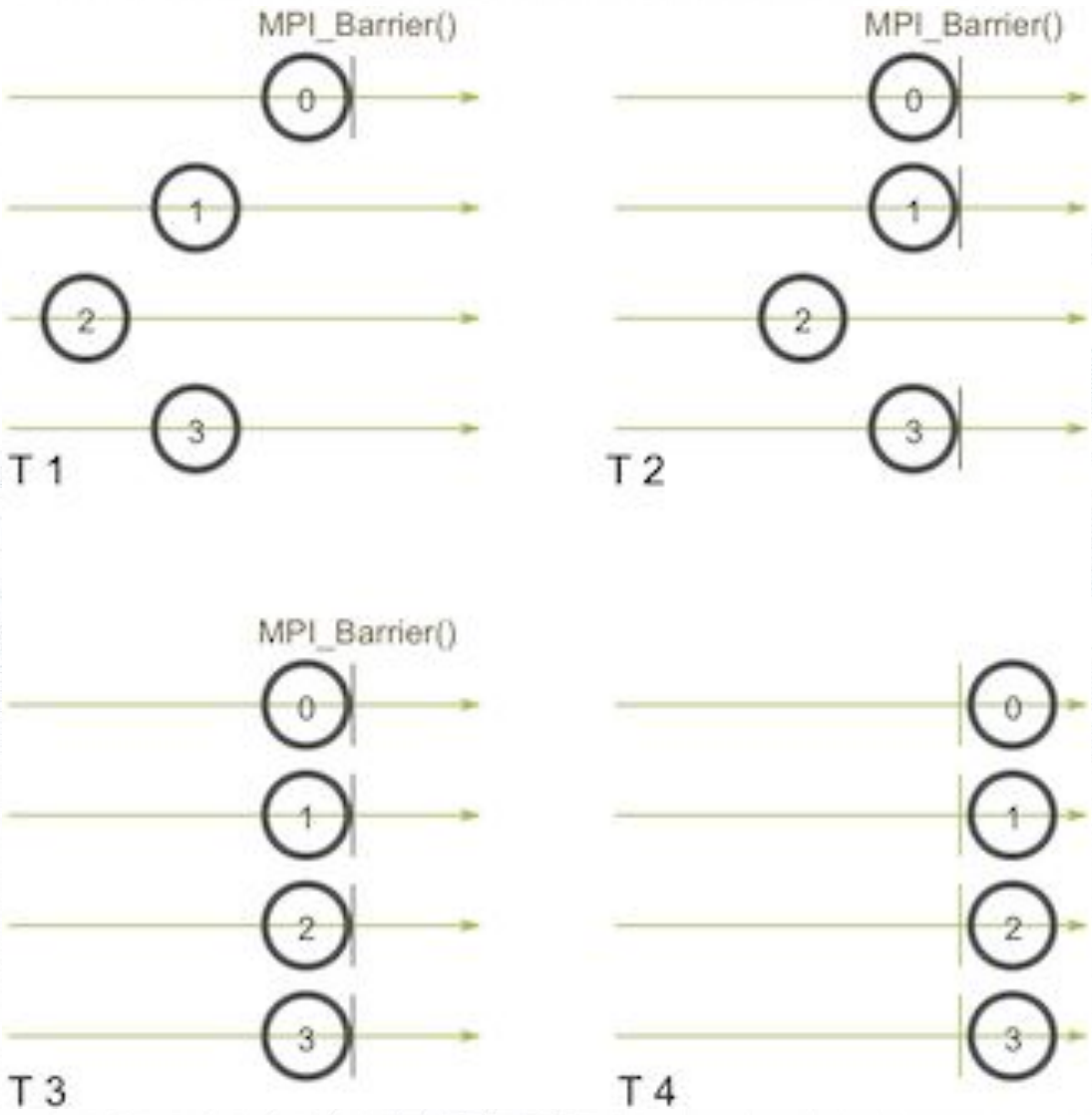
1. Write an MPI based program which starts communicating a message (like a number) back and forth from process 0 to process 1

- Such as an SMS between your friend and you
  - A point-to-point communication
- Use MPI_Send and MPI_Recv methods

A parallel program using MPI_BCAST

- One processor sends some data to all processors in a group
- The message is sent from the root process to all processes in the group, including the root process
  - int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm MPI_COMM_WORLD)
- To synchronize all processes within a communicator
- A node calling it will be blocked until all nodes within the group have called it:
  - int MPI_Barrier(MPI_Comm MPI_COMM_WORLD)

A parallel program using MPI_BCAST (cont.)



http://mpitutorial.com

A parallel program using MPI_BCAST (cont.)

```
...
MPI_Bcast(data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
total_my_bcast_time -= MPI_Wtime();
MPI_Barrier(MPI_COMM_WORLD);
total_mpi_bcast_time += MPI_Wtime();
if ( my_rank == 0)
        // Print off timing information  if (my_rank == 0)
        {
                printf("Data size = %d \n", num_elements * (int)sizeof(int) );
                printf("Avg MPI_Bcast time = %lf\n", total_mpi_bcast_time);
        }
```

Use MPI_BCAST to write a new program!

- **MPI_Bcast** distributes data from one process (the root) to all others in a communicator

2. Extend your previous program for three processes and use broadcasting
   - It can be found in this link: http://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/

References

1) Gropp, William D., et al. Using MPI: portable parallel programming with the message-passing interface. Vol. 1. MIT press, 1999.
2) Parallelization with OpenMP and MPI A Simple Example (C)
3) https://www.mpi-forum.org/
4) https://www.mpich.org/documentation/guides/
5) Parallel programming - with MPI and OpenMP
6) http://mpitutorial.com/tutorials/

Happy parallel programming.