Introduction to Parallel Computing (CMSC498X / CMSC818X)

Lecture 4: Message Passing

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Lecture schedule is online now
- Only use RHEL8 nodes on deepthought2
  - Login: ssh <login>@rhel8.deepthought2.umd.edu
  - Usage docs: https://hpcc.umd.edu/hpcc/help/usage.html
  - Quickstart: http://www.cs.umd.edu/class/fall2020/cmsc498x/deepthought2.shtml

Programming models

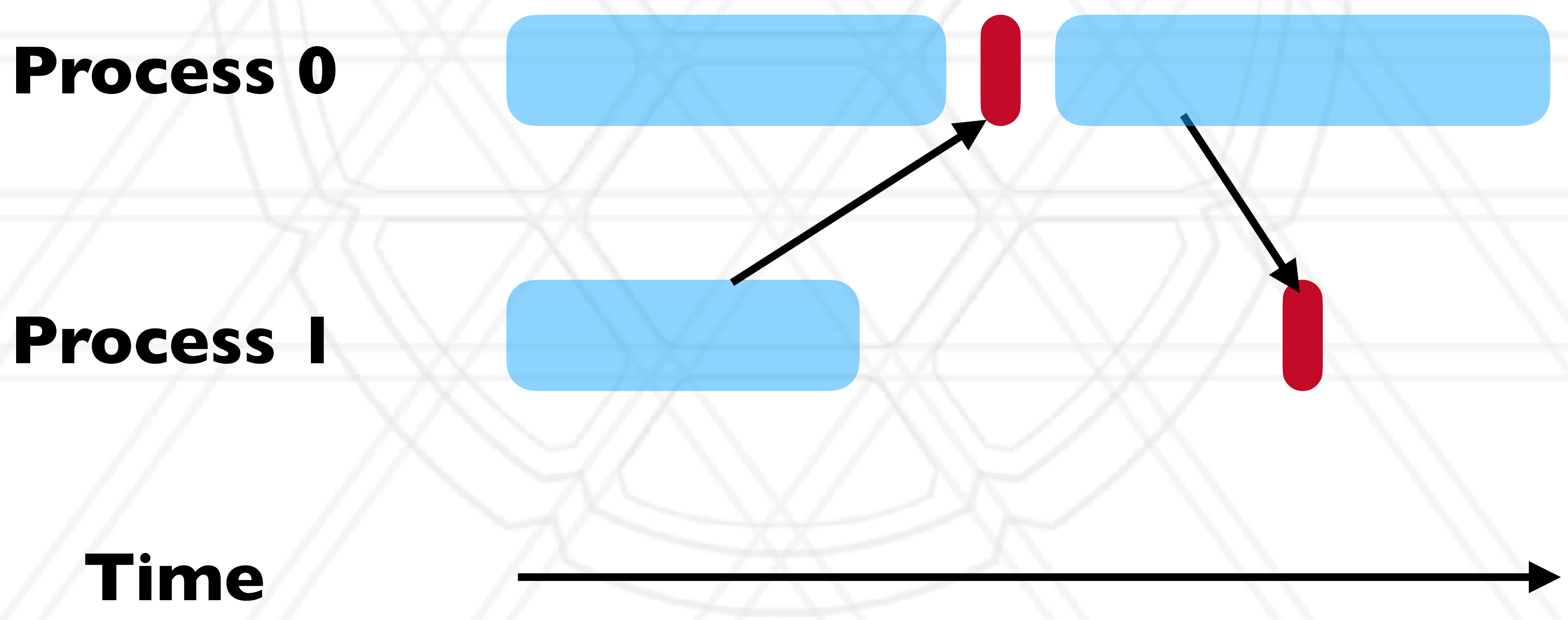
- Shared memory model: All threads have access to all of the memory
  - Pthreads, OpenMP
- Distributed memory model: Each process has access to their own local memory
  - Also sometimes referred to as message passing
  - MPI, Charm++
- Hybrid models: Use both shared and distributed memory models together
  - MPI+OpenMP, Charm++ (SMP mode)

Distributed memory / message passing

- Each process can use its local memory for computation
- When it needs data from remote processes, it has to send messages
- PVM (Parallel Virtual Machine) was developed in 1989-1993
- MPI forum was formed in 1992 to standardize message passing models and MPI 1.0 was released around 1994
  - v2.0 - 1997
  - v3.0 - 2012

Message passing

- Each process runs in its own address space
  - Access to only their memory (no shared data)
- Use special routines to exchange data



Message passing

- A parallel message passing program consists of independent processes
  - Processes created by a launch/run script
- Each process runs the same executable, but potentially different parts of the program
- Often used for SPMD style of programming

Message Passing Interface (MPI)

- It is an interface standard — defines the operations / routines needed for message passing
- Implemented by vendors and academics for different platforms
  - Meant to be “portable”: ability to run the same code on different platforms without modifications
- Some popular implementations are MPICH, MVAPICH, OpenMPI

Hello world in MPI

```
#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);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  printf("Hello world! I'm %d of %d\n", rank, size);

  MPI_Finalize();
  return 0;
}
```

Compiling and running an MPI program

- Compiling:

```
mpicc -o hello hello.c
```

- Running:

```
mpirun -n 2 ./hello
```

Process creation / destruction

- `int MPI_Init( int argc, char **argv )`
  - Initializes the MPI execution environment
- `int MPI_Finalize( void )`
  - Terminates MPI execution environment

Process identification

- `int MPI_Comm_size( MPI_Comm comm, int *size)`
  - Determines the size of the group associated with a communicator
- `int MPI_Comm_rank( MPI_Comm comm, int *rank)`
  - Determines the rank (ID) of the calling process in the communicator
- Communicator — a set of processes
  - Default communicator: `MPI_COMM_WORLD`

Send a message

```
int MPI_Send( const void *buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm )
```

`buf`: address of send buffer

`count`: number of elements in send buffer

`datatype`: datatype of each send buffer element

`dest`: rank of destination process

`tag`: message tag

`comm`: communicator

Receive a message

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status )
```

`buf`: address of receive buffer

`count`: maximum number of elements in receive buffer

`datatype`: datatype of each receive buffer element

`source`: rank of source process

`tag`: message tag

`comm`: communicator

`status`: status object

Simple send/receive in MPI

```
int main(int  argc, char *argv) {
  ...
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  int data;
  if (rank == 0) {
    data = 7;
    MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  } else if (rank == 1) {
    MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received data %d from process 0\n", data);
  }

  ...
}
```

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu