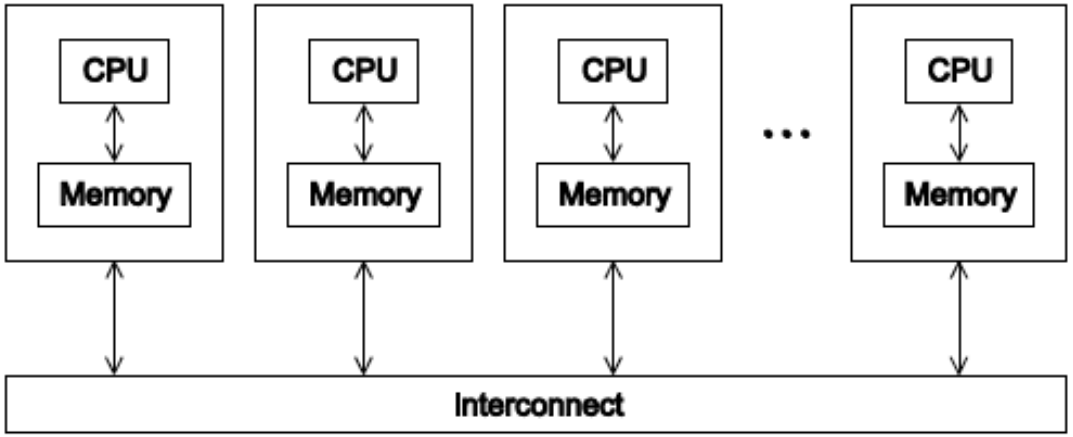
CS6068
Module 10
Introduction to MPI
Cluster Programming using MPI

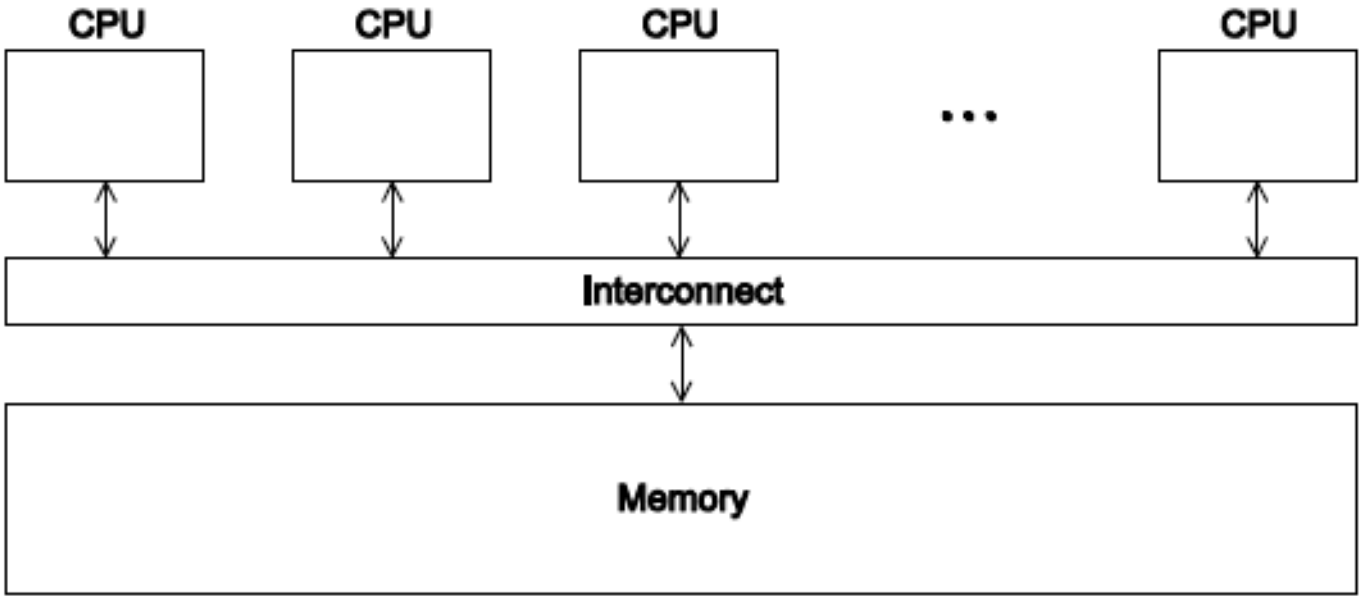
From Devices to Clusters
From Kernels to || Processes

This is What We Target With MPI



We will talk about processes

Distributed System: UMA



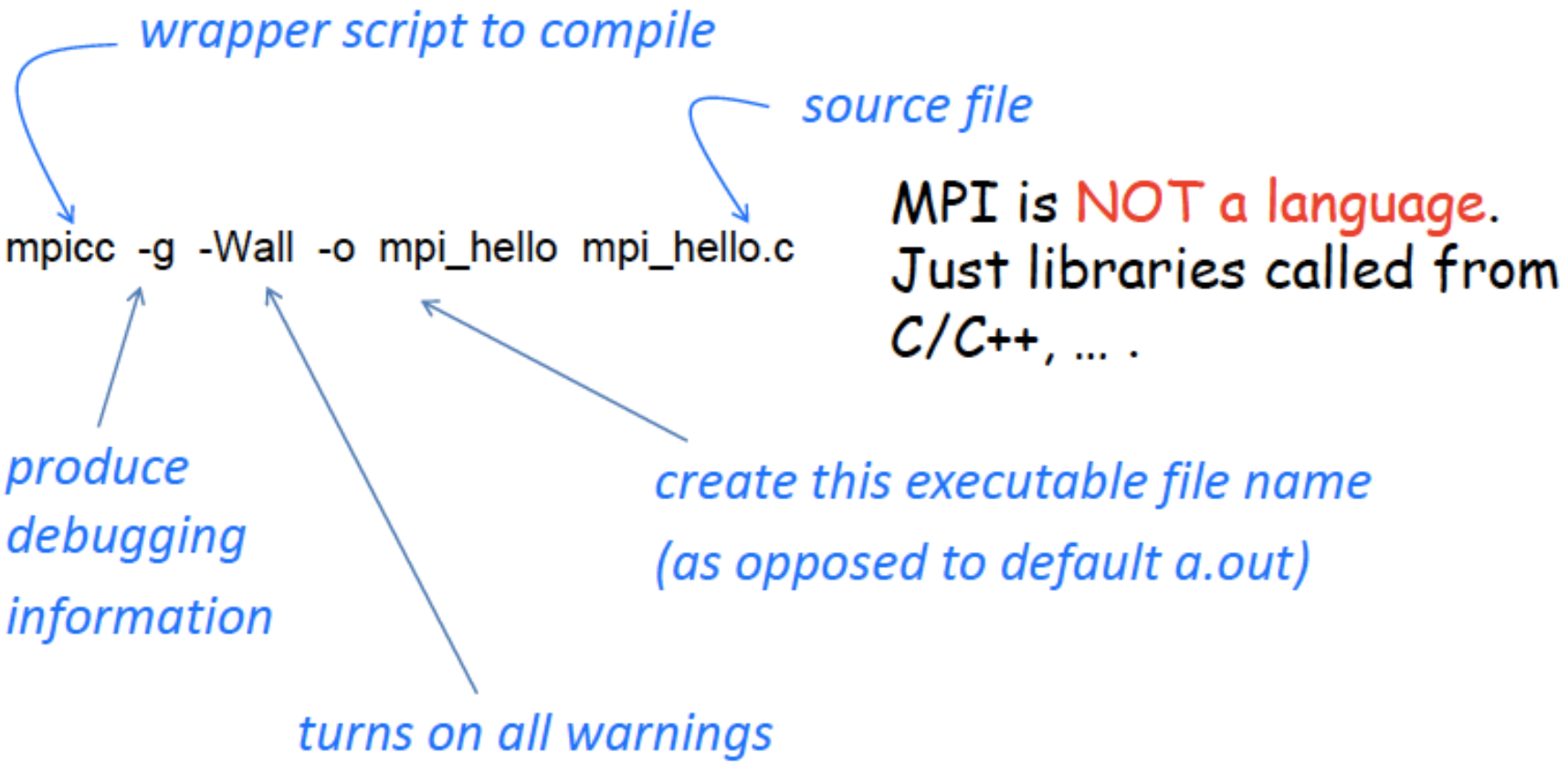
We will talk about Threads

OpenMP targets UMA systems

MPI Language and Processes

- There are collections of tools for compiling C/ Fortran code and execution of cluster of communicating MPI Processes.
- MPICH: freely available, portable implementation of MPI, current version MPI-3.
- MPICH wrapper scripts: mpicc, mpiexec
- Scripts create a distributed cluster of N processes with unique ranks 0,1,2,…,N-1

Compilation



Execution

mpiexec -n <number of processes> <executable>

---

mpiexec -n 1 ./mpi_hello

run with 1 process

mpiexec -n 4 ./mpi_hello

run with 4 processes

A first MPI example program:

https://drive.google.com/open?id=0BxrnHLmeZLsIUGdmWU1JOEZFZ3M

```
#include <stdio.h>
#include <string.h>
#include <mpi.h>
int main(void) {
    char        greeting[MAX_STRING];
    int         comm_sz;
    int         my_rank
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD,
&comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD,
&my_rank);
```

```
if (my_rank != 0) {
    /* Create message */
    globalvar += my_rank;
    sprintf(greeting, "Greetings from process %d of %d!",  my_rank,
comm_sz);
    /* Send message to process 0 */
    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
MPI_COMM_WORLD);
}

if (my_rank == 0) {
for ( q = 1; q < comm_sz; q++) {
       /* Receive message from process q */
       MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
         0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
       /* Print message from process q */
       printf("%s\n", greeting);
}
```

Communication

```
int MPI_Send(
   void*          msg_buf_p      /* in */,
   int            msg_size       /* in */,
   MPI_Datatype   msg_type       /* in */,
   int            dest           /* in */,
   int            tag            /* in */,
   MPI_Comm       communicator   /* in */);
```

To distinguish messages

rank of the receiving process

Message sent by a process using one communicator cannot be received by a process in another communicator.

Copyright © 2010, Elsevier Inc.

Receiving Messages

- Processes can request receiving messages
  - Synchronously or Asynchronously
  - Anonymous or not
  - Tagged or not

MPI_Reduce

has size:
sizeof(datatype) * count

```
int MPI_Reduce(
      void*         input_data_p    /* in  */,
      void*         output_data_p   /* out */,
      int           count           /* in  */,
      MPI_Datatype  datatype        /* in  */,
      MPI_Op        operator        /* in  */,
      int           dest_process    /* in  */,
      MPI_Comm      comm            /* in  */);
```

only relevant
to dest_process

```
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

```
double local_x[N], sum[N];
. . .
MPI_Reduce(local_x, sum, N, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

MPI_Reduce is called by all processes involved.

Copyright © 2010, Elsevier Inc.
All rights Reserved

Collective vs. Point-to-Point Communications

- All the processes in the communicator must call the same collective function.
  - For example, a program that attempts to match a call to MPI_Reduce on one process with a call to MPI_Recv on another process is erroneous.
- The arguments passed by each process to an MPI collective communication must be "compatible."
  - For example, if one process passes in 0 as the dest_process and another passes in 1, then the outcome of a call to MPI_Reduce is erroneous.

Scan, Scatter, Gather, and Bulk-Sync

- MPI_Scan calculates the prefix sum based on process ranks
- MPI_Scatter is one to many communication and distributes an array across a communication group
- MPI_Gather is the reverse many to one
- MPI_Barrier insures synchronization of group

Strengths of MPI

- Many applications can be written using only 6 basic functions
- Extensive library for optimizing code
  - Contains over 125 APIs
- Scalable and Flexible
  - Programs can be written to be targeted to network using point-to-point communication
  - Programs can be ported to other platforms

Parallel Big Data Apps

- Apache Spark and MPI are distributed frameworks for processing enormous datasets.
- Recent results show MPI/OpenMP outperforms Spark by more than one order of magnitude in terms of processing speed and provides more consistent performance. However, Spark shows better data management infrastructure and the possibility of dealing with other aspects such as node failure and data replication.