PARALLEL PROCESSING

Mohammed Alabdulkareem

kareem@ksu.edu.sa

Office 2247

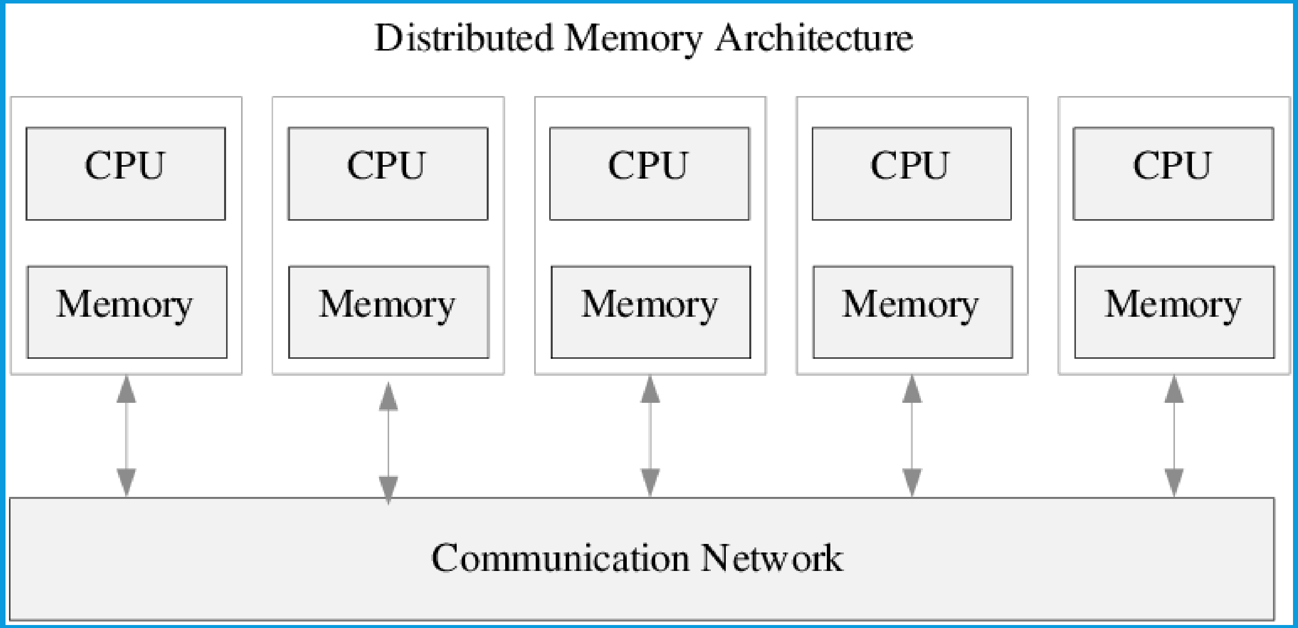
11

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- Shared memory machines have faster communication time than distributed memory machine but it is less scalable in terms of number of processors.
- Distributed memory machines are more scalable in number of processors. But with more communication time.
- Hence the speed of the communication network need to be fast enough to close this gap.

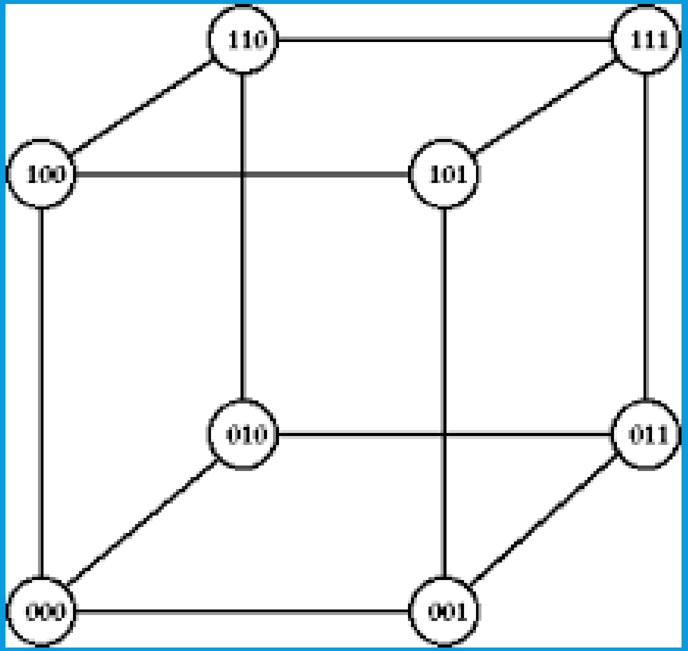
PROGRAMMING DISTRIBUTED MEMORY MACHINE

- Distributed memory machines can not communicate through shared memory.
- Exchanging data and communications are done using communication network.

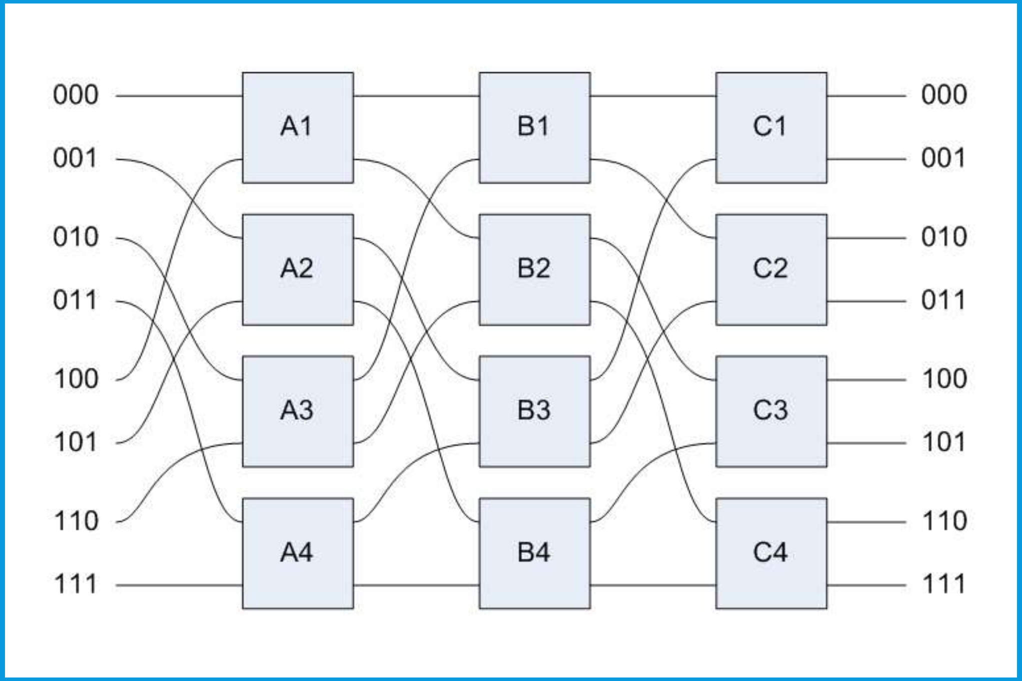


PROGRAMMING DISTRIBUTED MEMORY MACHINE

Interconnection networks (examples)



Hypercube



Omega

PROGRAMMING DISTRIBUTED MEMORY MACHINE

Supercomputers with interconnection network

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- In distributed memory machines cooperation among processes implies the data exchange using message passing (communication overhead).
- The overall execution time is consequently a sum of computation and communication time.
- Algorithms with only local communication between neighboring processors are faster and more scalable than the algorithms with the global communication among all processors.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI is **M**essage **P**assing **I**nterface.
- MPI is introduced from the practical point of view, with a set of basic operations that enable implementation of parallel programs.
- The MPI library interface is a specification, not an implementation.
- The MPI is not a language, and all MPI operations are expressed as functions, subroutines, or methods, according to the appropriate language bindings for C and Fortran.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI standard includes:
  - Process creation and management.
  - Language bindings for C and Fortran.
  - Point-to-point and collective communications.
  - Group and communicator concepts.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI program consists of autonomous processes that are able to execute their own code in the sense of Multiple Instruction Multiple Data (MIMD) paradigm.
- The processes communicate via calls to MPI communication operations independently of operating system.
- Any MPI program should have operations to initialize execution environment and to control starting and terminating procedures of all generated processes.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI processes can be collected into **groups** of specific ***size*** that can communicate in its own environment where each message sent in a context must be received only in the same context.
- A process group and context together form an MPI communicator.
- A process is identified by its rank in the group associated with a communicator.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI has default communicator **MPI_COMM_WORLD**
- The group associated with this communicator includes all initial processes with default context.
- MPI operations return a status of the execution success; in C routines it is returned as the value of the function.
- The MPI standard *does not* specify how **st dout** from different nodes should be collected for printing at the originating process.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI_SEND (buf, count, datatype, dest, tag, comm)
- The basic MPI communication is characterized by two fundamental MPI operations MPI_SEND and MPI_RECV.
- MPI_SEND is blocking call it will not complete until there is matching MPI_RECV on the other receiving process.
- The MPI_RECV will empty the input send buffer ***buf*** of matching MPI_SEND.
- MPI_SEND will return when the message data has been delivered to the communication system and the send buffer ***buf*** of the sender process source can be reused.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI_SEND (buf, count, datatype, dest, tag, comm)
  - ***buf*** is a pointer to the send buffer
  - ***count*** is the number of data items
  - ***datatype*** is the type of data items (more about data types in MPI)
  - ***dest*** the rank of receiver process (similar to thread number)
  - ***tag*** a message tag to distinguish between different messages for the same receiver process
  - ***comm*** the communicator.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI_RECV (buf, count, datatype, source, tag, comm, status)
  - ***buf*** is a pointer to the output buffer
  - ***count*** is the number of data items to be received
  - ***datatype*** is the type of data items
  - ***source*** the rank of sending process
  - ***tag*** a message tag
  - ***comm*** the communicator.
  - ***status*** contains further information in case of error

PROGRAMMING DISTRIBUTED MEMORY MACHINE

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- Wildcard may be used at the destination side

MPI_RECV (buf, count, datatype, <u>source</u>, <u>tag</u>, comm, status)

  - ***Source*** MPI_ANY_SOURCE
  - ***Tag*** MPI_ANY_TAG

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- **MPI_Init** initializes the MPI execution environment.
- **MPI_Comm_size(MPI_COMM_WORLD, &size)** returns size, which is the number of started processes.
- **MPI_Comm_rank(MPI_COMM_WORLD, &rank)** that returns rank or the ID of each process.
- **MPI_Finalize** exits the MPI

PROGRAMMING DISTRIBUTED MEMORY MACHINE

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

PROGRAMMING DISTRIBUTED MEMORY MACHINE

```
% mpicc -o helloworld helloworld.c
% mpirun -np 4 helloworld
Hello world from process 0 of 4
Hello world from process 3 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
%
```

PROGRAMMING DISTRIBUTED MEMORY MACHINE

```
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int number;
if (world_rank == 0) {
  number = -1;
  MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  } else if (world_rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,  MPI_STATUS_IGNORE);
        printf("Process 1 received number  %d from process  0\n",  number);
 }
```

PROGRAMMING DISTRIBUTED MEMORY MACHINE

MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);

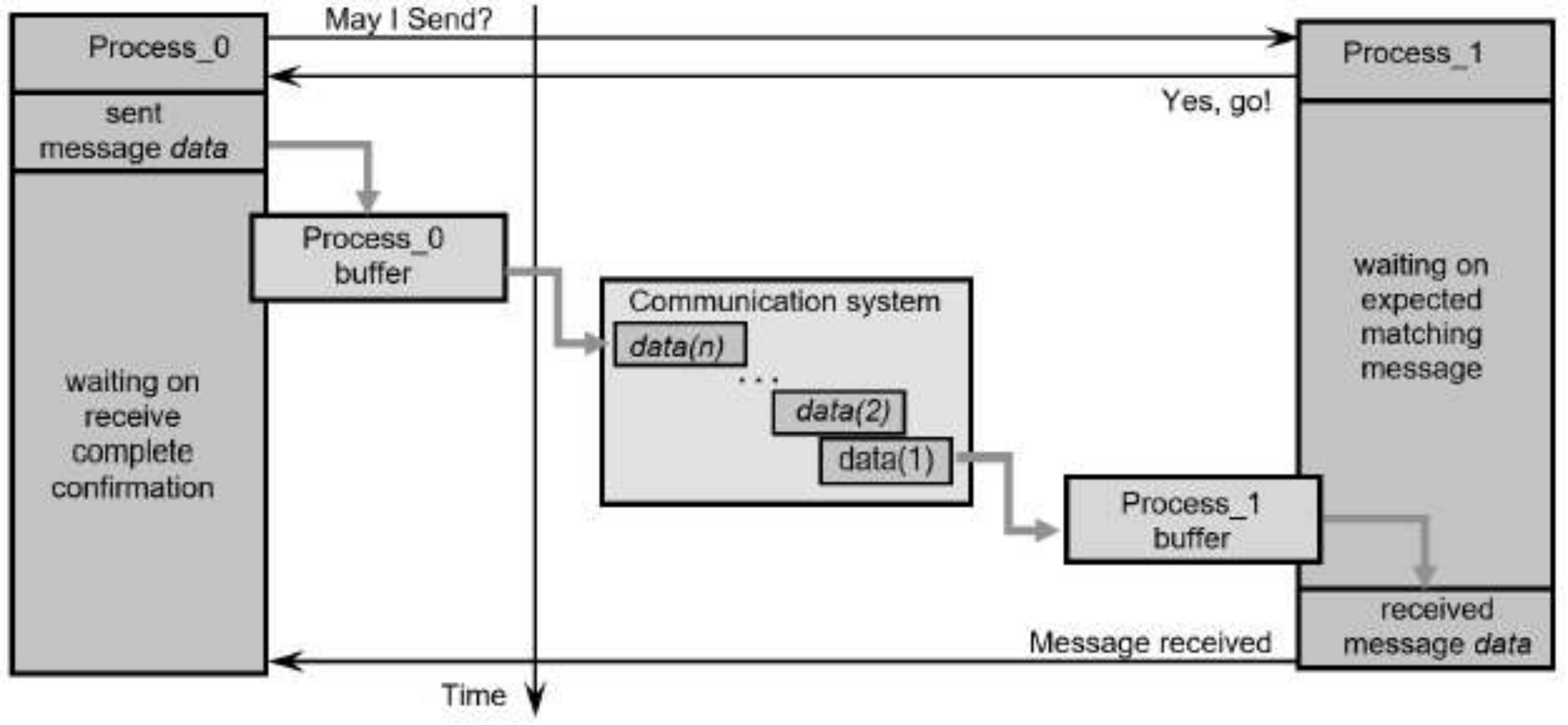
- ***Buf*** &number
- ***count*** 1
- ***datatype*** MPI_INT
- ***dest*** 1
- ***tag*** 0
- ***comm*** MPI_COMM_WORLD.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

```
mpirun -n 2 ./send_recv
Process 1 received number -1 from process 0
```

PROGRAMMING DISTRIBUTED MEMORY MACHINE

How it works



PROGRAMMING DISTRIBUTED MEMORY MACHINE

MPI_RECV (buf, count, datatype, source, tag, comm, status)

- The number of received data items of datatype must be equal of fewer as specified by count. (count >= 0).
- Receiving more data items results in an error and the status value return more information.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI_SENDRECV (sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- MPI_SENDRECV (sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

- MPI_SENDRECV combines a sending of message to destination process **dest** and a receiving of another message from process **source**, in a single call in sender and receiver process.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

Measuring time

```
double start, finish;
start = MPI_Wtime ();
... //MPI program segment to be measured
finish = MPI_Wtime ();
printf ("Elapsed time is %f\n", finish - start);
```

PROGRAMMING DISTRIBUTED MEMORY MACHINE

- The elapsed time (wall-clock) between two points in an MPI program can be measured by using operation MPI_WTIME ().
- The time measured in seconds.

MPI_BARRIER (comm)

- This operation is used to synchronize the execution of a group of processes specified within the communicator **comm.**
- When a process reach this operation it waits until all other processes reach the **MPI_BARRIER**.
- The barrier is a simple way of separating two phases of a computation to ensure that messages generated in different phases do not interfere.

PROGRAMMING DISTRIBUTED MEMORY MACHINE

MPI_BARRIER (comm)