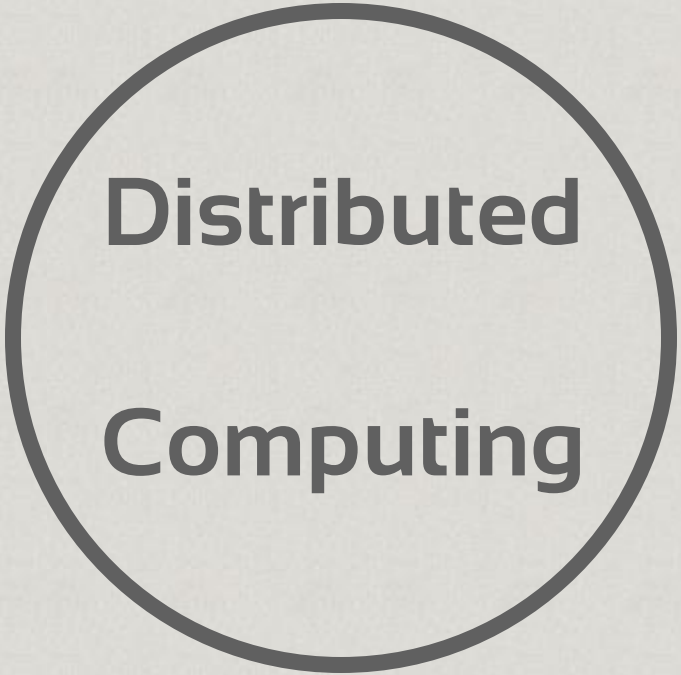

MPI

Mahmoud Badry, mma18@fayoum.edu.eg

AGENDA

Mahmoud Badry, mma18@fayoum.edu.eg

Knowing and installing MPICH

Mahmoud Badry, mma18@fayoum.edu.eg

MPICH (www.mpich.org)

- MPICH is a **freely** available, **portable** implementation of **MPI**, a standard for message-passing for **distributed-memory** applications used in **parallel** computing.
- The CH part of the name was derived from "**Chameleon**", which was a portable parallel programming library developed by **William Gropp**, one of the **founders** of MPICH.
- History:
  - **Before 2001**: MPICH1 which implements MPI-1
  - **Between 2001-2012**: MPICH2 which implements MPI-2
  - **After 2012**: MPICH v3.0 which implements MPI-3

Mahmoud Badry, mma18@fayoum.edu.eg

MPICH installation (For our lab)

- MPICH installation requirements:
  - Windows XP.
  - .NET framework 2.0.
  - VS2005.
- To install MPICH in your machine use the instructions that can be found in this link: http://www.cs.utah.edu/~delisi/vsmpi/
- After installation and compiling your MPI use this command on CMD:
  - `mpiexec -np 4 yourprogram.exe`

Mahmoud Badry, mma18@fayoum.edu.eg

Point to point commuincation in MPI

Mahmoud Badry, mma18@fayoum.edu.eg

Communicators

- In MPI, all communication operations are executed using a **communicator**. A communicator represents a communication domain which is essentially a **set** of **processes** that exchange messages between each other.
- The MPI default communicator MPI COMM WORLD is used for the communication. This **communicator captures** all **processes** executing a **parallel** program

Mahmoud Badry, mma18@fayoum.edu.eg

MPI_Send operation

- ```
  int MPI_Send(void *smessage,
               int count,
               MPI_Datatype datatype,
               int dest,
               int tag,
               MPI_Comm comm).
  ```
- **smessage** specifies a send buffer which contain the data elements.
- **count** is the number of elements to be sent from the send buffer.
- **datatype** is the datatype of each entry in the send buffer.
- **dest** specifies the rank of the target process.
- **tag** is a message tag which can be used by the receiver to distinguish different messages from the same sender.
  ```

MPI_Recv operation

- ```
  int MPI_Recv(void *rmessage,
               int count,
               MPI_Datatype datatype,
               int source,
               int tag,
               MPI_Comm comm,
               MPI_Status *status).
  ```
- Like MPI_Send except that:
  - **rmessage** specifies the receive buffer in which the message should be stored.
  - **status** specifies a data structure which contains information about a message after the completion of the receive operation. Can be ignored using MPI_STATUS_IGNORE
  ```

MPI_Recv operation Cont.

- Like MPI_Send except that:
  - By using **`source`** = `MPI_ANY_SOURCE`, a process can receive a message from any arbitrary process.
  - Similarly, by using **`tag`** = `MPI_ANY_TAG`, a process can receive a message with an arbitrary tag.
- After completion of the receive operation, status variable will contain these information:
  - status.MPI_SOURCE specifies the rank of the sending process.
  - status.MPI_TAG specifies the tag of the message received.
  - status.MPI_ERROR contains an error code.

MPI Datatypes

MPI Datentyp	C-Datentyp
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	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_WCHAR	wide char
MPI_PACKED	special data type for packing
MPI_BYTE	single byte value

Mahmoud Badry, mma18@fayoum.edu.eg

What happends while sending and receiving ?

1. The data elements to be sent are **copied** from the send buffer **smessage** specified as parameter into a **system buffer** of the MPI runtime system. The **message** is **assembled** by adding a **header** with information on the **sending process**, the **receiving process**, the **tag**, and the **communicator** used.
2. The **message** is sent via the **network** from the **sending** process to the **receiving** process.
3. At the **receiving** side, the data entries of the **message** are **copied** from the **system buffer** into the **receive buffer** rmessage specified by MPI_Recv().

Mahmoud Badry, mma18@fayoum.edu.eg

Send and receive operations nature

- Both MPI_Send() and MPI_Recv() are blocking, asynchronous operations.
  - MPI_Recv() operation can also be **started** when the **corresponding** MPI_Send() operation has not yet been **started**.
  - The **process** executing the MPI_Recv() operation is **blocked** until the specified receive **buffer** contains the data elements sent.
  - Similarly, an MPI_Send() **operation** can also be **started** when the corresponding MPI_Recv() **operation** has not yet been **started**.

Mahmoud Badry, mma18@fayoum.edu.eg

Point to point operation examples

Mahmoud Badry, mma18@fayoum.edu.eg

QUESTIONS

Mahmoud Badry, mma18@fayoum.edu.eg

THANK YOU!

Mahmoud Badry, mma18@fayoum.edu.eg