Introduction to MPI (Message Passing Interface)

Dr. Tassadaq Hussain

RIPHAH INTERNATIONAL UNIVERSITY

Microsoft Research Centre

Tasks

- Task 1: Programming API using OpenMP
  - Performance Results
- Task 2: Execute the Given Code Using OpenMP
- Task3: Apply OpenMP techniques on IPUM and RLS algorithms (given)

The MPI is a library that collected the best features of many message-passing systems that have been developed over the years.

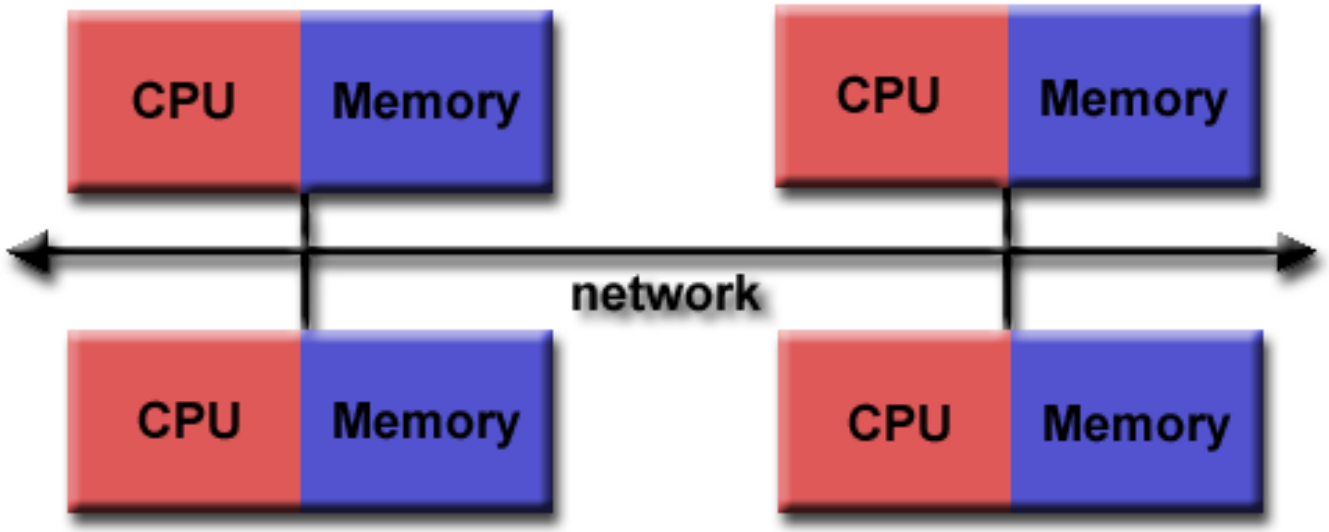
These features were improved, where appropriate, and standardized. As a result the MPI specifies the names, calling sequences, and results of subroutines to be called from Fortran programs, the functions to be called from C programs, and the classes and methods that make up the MPI C++ library.

MPI

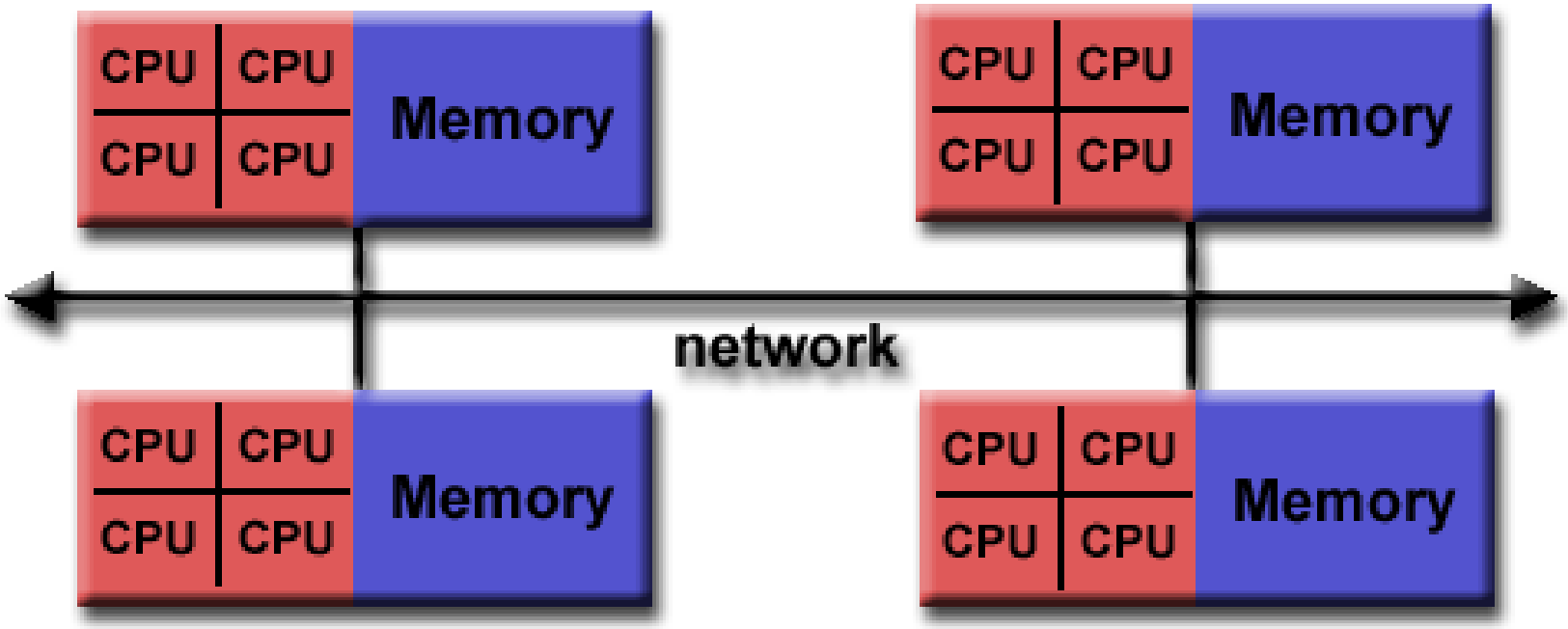
- **Advantages**:
  - Practical
  - Portable
  - Efficient
  - Flexible

- **Drawback**:
  - Overhead. Communications can often create a large overhead, which needs to be minimized.
  - Parallelization. Significant changes to the code are often required, making transfer between the serial and parallel code difficult.

Distributed Memory Architecture



Hybrid (Shared and Distrubuted Systems Architecture)



Thread Safe

Thread safety is a computer programming concept applicable in the context of multithreaded programs.

A piece of code is thread-safe if it manipulates shared data structures only in a manner that guarantees safe execution by multiple threads at the same time

Open MPI

Open MPI is a thread-safe, open source MPI implementation developed and supported by a consortium of academic, research, and industry partners.

Programming Using MPI

```
#include "mpi.h"
...
main(int argc, char** argv) {
...
/* No MPI functions called before this */
MPI_Init(&argc, &argv);
...
MPI_Finalize();
/* No MPI functions called after this */
...
} /* main */
```

Environment Management Routines

MPI_Init(&argc, &argv)

int MPI_Comm_rank(MPI_Comm comm, int rank)

int MPI_Comm_size(MPI_Comm comm, int size)

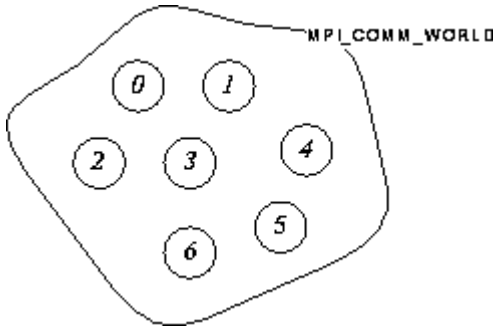
int MPI_Abort(MPI_Comm comm, int errorcode)

MPI_Get_processor_name (&name,&resultlength)

MPI_Finalize();

Communicator

MPI_INIT defines something called MPI_COMM_WORLD for each process that calls it. MPI_COMM_WORLD is a communicator. All MPI communication calls require a communicator argument and MPI processes can only communicate if they share a communicator.



MPI Example

```
#include "mpi.h"
main( argc, argv )
int argc;
char **argv;
{
char message[20];
int myrank;
MPI_Status status;
MPI_Init( &argc, &argv ); //Initialize the MPI execution
environment.
MPI_Comm_rank( MPI_COMM_WORLD, &myrank ); //Determines the rank
of the
                //calling process in the communicator.
if (myrank == 0) /* code for process zero */
{
strcpy(message,"Hello World!");
MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99,
MPI_COMM_WORLD);
}
else if (myrank == 1) /* code for process one */
{
MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD,
&status);
printf("received :%s:\n", message);
}
MPI_Finalize();
}
```

mpirun -np 4 ring_c