Distributed-Memory Parallel Computers

- Scalable interconnection network.
- Some memory physically local, some remote
- Major issues: network latency and bandwidth, message-passing overhead, data distribution.
- Incremental parallelization and debugging can be difficult.
Logical model
Physical model
Interconnection Network Topology

- Very interesting theoretical topic.
- For typical users of modern parallel machines, the details of interconnection topology are not a first-order concern*.

* Your mileage may vary.
Message passing programming model

- N nodes connected by an interconnection network
- Nodes are processor(s)/memory/network
- K processes are distributed to a subset of N nodes
- Processes have separate address spaces, code, data, stack, …
- Processes communicate by explicitly sending/receiving messages.
- A message is an array of, possibly typed, bytes
- Message passing is two-sided.
- Flexible, very scalable model
What is MPI?

• De facto standard API for explicit message-passing MIMD/SPMD programming.
• Many implementations over many networks
• Developed in mid 90’s by consortium, reflecting lessons learned from machine-specific libraries and PVM.
• Focused on: homogeneous MPPs, high performance, library writing, portability.
Six-Function MPI

- MPI_Init
- MPI_Finalize
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Send
- MPI_Recv
MPI “Hello World” in Fortran77

implicit none
include 'mpif.h'
integer myid, numprocs, ierr

call mpi_init( ierr )
call mpi_comm_rank( mpi_comm_world, myid, ierr )
call mpi_comm_size( mpi_comm_world, numprocs, ierr )

print *, "hello from ", myid, " of ", numprocs

call mpi_finalize( ierr )
stop
end
MPI “Hello World” in C

```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int myid, numprocs, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name, &namelen);
    printf("hello from %s: process %d of %d\n",
            processor_name, myid, numprocs);
    MPI_Finalize();
}
```
Function MPI_Send

MPI_SEND(buf, count, datatype, dest, tag, comm)

IN buf initial address of send buffer (choice)
IN count number of entries to send (integer)
IN datatype datatype of each entry (handle)
IN dest rank of destination (integer)
IN tag message tag (integer)
IN comm communicator (handle)

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

MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERR)

<type> BUF(*)
IN INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERR
Function MPI_Recv

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

OUT  buf     initial address of receive buffer (choice)
IN   count   max number of entries to receive (integer)
IN   datatype datatype of each entry (handle)
IN   dest    rank of source (integer)
IN   tag     message tag (integer)
IN   comm    communicator (handle)
OUT  status  return status (Status)

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

MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERR)
<type> BUF(*)
   INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM
   INTEGER STATUS(MPI_STATUS_SIZE), IERR