1 Introduction to MPI

1.1 Parallel Computing

- Separate workers or processes.
- Interact by exchanging information.

1.2 Types of parallel computing

All use different data for each worker

Data-parallel Same operations on different data. Also called SIMD.

SPMD Same program, different data.

MIMD Different programs, different data.

SPMD and MIMD are essentially the same because any MIMD can be made SPMD.
SIMD is also equivalent, but in a less practical sense. MPI is primarily for SPMD/MIMD.

1.3 Communicating with other processes

Data must be exchanged with other workers;

- **Cooperative** — all parties agree to transfer data.
  Message-passing is an approach that makes the exchange of data cooperative. Data must both be explicitly sent and received.
  An advantage is that any change in the *receiver's* memory is made with the receiver's participation.

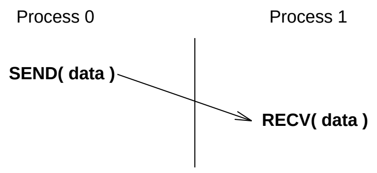

Figure 1: Cooperative–Communicating with other processes.

- **One sided** — one worker performs transfer of data.
  One-sided operations between parallel processes include remote memory reads and writes.
  An advantage is that data can be accessed without waiting for another process.

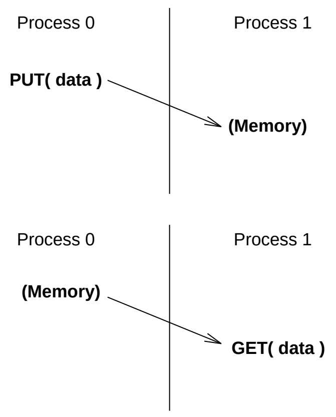

Figure 2: One sided–Communicating with other processes.

1.4 Hardware models

- Distributed memory (e.g., Paragon, IBM SPx, workstation network)
- Shared memory (e.g., SGI Power Challenge, Cray T3D)

Either may be used with SIMD or MIMD software models. ***All memory is distributed.***

1.5 What is MPI?

- A *message-passing library specification*
  - message-passing model.
  - not a compiler specification.
  - not a specific product.
- For parallel computers, clusters, and heterogeneous networks.
- Full-featured.

- Designed to permit (unleash?) the development of parallel software libraries.
- Designed to provide access to advanced parallel hardware for
  - end users.
  - library writers.
  - tool developers.

1.6 Who Designed MPI?

- Broad participation
- Vendors
  - IBM, Intel, TMC, Meiko, Cray, Convex, Ncube
- Library writers
  - PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda
- Application specialists and consultants

Companies	Laboratories	Universities
ARCO	ANL	UC Santa Barbara
Convex	GMD	Syracuse U
Cray Res	LANL	Michigan State U
IBM	LLNL	Oregon Grad Inst
Intel	NOAA	U of New Mexico
KAI	NSF	Miss. State U.
Meiko	ORNL	U of Southampton
NAG	PNL	U of Colorado
nCUBE	Sandia	Yale U
ParaSoft	SDSC	U of Tennessee
Shell	SRC	U of Maryland
TMC		Western Mich U
		U of Edinburgh
		Cornell U.
		Rice U.
		U of San Francisco

1.7 MPI Implementations

- MPICH (Argonne National Laboratory).
- UNIFY (Mississippi State University).
- CHIMP (Edinburgh Parallel Computing Centre).
- LAM (Ohio Supercomputer Center).
- MPI for the Fujitsu AP1000 (Australian National University).
- Cray MPI Product for the T3D (Cray Research and the Edinburgh Parallel Computing Center).
- IBM's MPI for the SP.
- SGI's MPI for 64-bit mips3 and mips4.
- PowerMPI for Parsytec Systems.
- HP's MPI implementation.

1.8 Is MPI Large or Small?

- MPI is large (125 functions)
  - MPI's extensive functionality requires many functions.
  - Number of functions not necessarily a measure of complexity.

MPI_ABORT MPI_ADDRESS MPI_ALLGATHER MPI_ALLGATHERV MPI_ALLREDUCE MPI_ALLTOALL
MPI_ALLTOALLV MPI_ATTR_DELETE MPI_ATTR_GET MPI_ATTR_PUT MPI_BARRIER MPI_BCAST MPI_BSEND
MPI_BSEND_INIT MPI_BUFFER_ATTACH MPI_BUFFER_DETACH MPI_CANCEL MPI_CARTDIM_GET MPI_CART_COORDS
MPI_CART_CREATE MPI_CART_GET MPI_CART_MAP MPI_CART_RANK MPI_CART_SHIFT MPI_CART_SUB
MPI_COMM_COMPARE MPI_COMM_CREATE MPI_COMM_DUP MPI_COMM_FREE MPI_COMM_GROUP
MPI_COMM_RANK MPI_COMM_REMOTE_GROUP MPI_COMM_REMOTE_SIZE MPI_COMM_SIZE MPI_COMM_SPLIT
MPI_COMM_TEST_INTER MPI_DIMS_CREATE MPI_ERRHANDLER_CREATE MPI_ERRHANDLER_FREE
MPI_ERRHANDLER_GET MPI_ERRHANDLER_SET MPI_ERROR_CLASS MPI_ERROR_STRING MPI_FINALIZE
MPI_GATHER MPI_GATHERV MPI_GET_COUNT MPI_GET_ELEMENTS MPI_GET_PROCESSOR_NAME MPI_GRAPHDIMS_GET
MPI_GRAPH_CREATE MPI_GRAPH_GET MPI_GRAPH_MAP MPI_GRAPH_NEIGHBORS MPI_GRAPH_NEIGHBORS_COUNT
MPI_GROUP_COMPARE MPI_GROUP_DIFFERENCE MPI_GROUP_EXCL MPI_GROUP_FREE MPI_GROUP_INCL
MPI_GROUP_INTERSECTION MPI_GROUP_RANGE_EXCL MPI_GROUP_RANGE_INCL MPI_GROUP_RANK
MPI_GROUP_SIZE MPI_GROUP_TRANSLATE_RANKS MPI_GROUP_UNION MPI_IBSEND MPI_INIT MPI_INITIALIZED
MPI_INTERCOMM_CREATE MPI_INTERCOMM_MERGE MPI_IPROBE MPI_IRECV MPI_IRSEND MPI_ISEND
MPI_ISSEND MPI_KEYVAL_CREATE MPI_KEYVAL_FREE MPI_OP_CREATE MPI_OP_FREE MPI_PACK MPI_PACK_SIZE
MPI_PCONTROL MPI_PROBE MPI_RECV MPI_RECV_INIT MPI_REDUCE MPI_REDUCE_SCATTER MPI_REQUEST_FREE

MPI_RSEND MPI_RSEND_INIT MPI_SCAN MPI_SCATTER MPI_SCATTERV MPI_SEND MPI_SENDRECV MPI_SENDRECV_REPLACE MPI_SEND_INIT MPI_SSEND MPI_SSEND_INIT MPI_START MPI_STARTALL MPI_TEST MPI_TESTALL MPI_TESTANY MPI_TESTSOME MPI_TEST_CANCELLED MPI_TOPO_TEST MPI_TYPE_COMMIT MPI_TYPE_CONTIGUOUS MPI_TYPE_EXTENT MPI_TYPE_FREE MPI_TYPE_HINDEXED MPI_TYPE_HVECTOR MPI_TYPE_INDEXED MPI_TYPE_LB MPI_TYPE_SIZE MPI_TYPE_STRUCT MPI_TYPE_UB MPI_TYPE_VECTOR MPI_UNPACK MPI_WAIT MPI_WAITALL MPI_WAITANY MPI_WAITSOME MPI_WTICK MPI_WTIME

- MPI is small. Many parallel programs can be written with just 6 basic functions.
  - **MPI_Init**– Initialise MPI.
  - **MPI_Comm_size**– Find out how many processes there are.
  - **MPI_Comm_rank**– Find out which process I am.
  - **MPI_Send**– Send a message.
  - **MPI_Recv**– Receive a message.
  - **MPI_Finalize**– Terminate MPI.
- MPI is just right
  - One can access flexibility when it is required.
  - One need not master all parts of MPI to use it.

1.9 Where to use MPI?

- You need a portable parallel program.
- You are writing a parallel library.
- You have irregular or dynamic data relationships that do not fit a data parallel model.

Where *not* to use MPI:

- You can use HPF or a parallel Fortran 90.
- You don't need parallelism at all.
- You can use libraries (which may be written in MPI).

1.10 How To Use MPI?

- When possible, start with a debugged serial version.
- Design parallel algorithm.
- Write code, making calls to MPI library.
- Compile and run using implementation specific utilities.
- Run with a few nodes first, increase number gradually.

1.11 Getting started

1.11.1 Writing MPI programs

First program with MPI (hello.c)

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

int main( argc, argv )
int argc;
char **argv;
{
MPI_Init( &argc, &argv );
printf( "Hello world\n" );
MPI_Finalize();
return 0;
}
```

- `#include "mpi.h"`

  provides basic MPI definitions and types.
- `MPI_Init`

  starts MPI.
- `MPI_Finalize`

  exits MPI.
- Note that all non-MPI routines are local; thus the

  `printf`

  run on each process.

1.11.2 Compiling and linking

Best to use a standard ***Makefile***. MPICH implementation has examples in

```
/opt/mpich-1.2.5.10-ch_p4-gcc/examples/
```

This file is a *template* **Makefile**. The program (script)

```
mpireconfig
```

translates this to a **Makefile** for a particular system. This allows you to use the same **Makefil**e for a network of workstations and a massively parallel computer, even when they use different compilers, libraries, and linker options.

```
mpireconfig Makefile
```

Note that you must have *mpireconfig* in your *PATH.*
Sample Makefile.in:

```
##### User configurable options #####

ARCH        = @ARCH@
COMM        = @COMM@
INSTALL_DIR = @INSTALL_DIR@
CC          = @CC@
F77         = @F77@
CLINKER     = @CLINKER@
FLINKER     = @FLINKER@
OPTFLAGS    = @OPTFLAGS@
#
LIB_PATH    = -L$(INSTALL_DIR)/lib/$(ARCH)/$(COMM)
FLIB_PATH   = @FLIB_PATH_LEADER@$(INSTALL_DIR)/lib/$(ARCH)/$(COMM)
LIB_LIST    = @LIB_LIST@
#
INCLUDE_DIR = @INCLUDE_PATH@ -I$(INSTALL_DIR)/include

### End User configurable options ###

CFLAGS  = @CFLAGS@ $(OPTFLAGS) $(INCLUDE_DIR) -DMPI_$(ARCH)
FFLAGS = @FFLAGS@ $(INCLUDE_DIR) $(OPTFLAGS)
LIBS = $(LIB_PATH) $(LIB_LIST)
FLIBS = $(FLIB_PATH) $(LIB_LIST)
EXECS = hello
```

```
default: hello

all: $(EXECS)

hello: hello.o $(INSTALL_DIR)/include/mpi.h
$(CLINKER) $(OPTFLAGS) -o hello hello.o \
$(LIB_PATH) $(LIB_LIST) -lm

clean:
/bin/rm -f *.o *~ PI* $(EXECS)

.c.o:
$(CC) $(CFLAGS) -c $*.c
.f.o:
$(F77) $(FFLAGS) -c $*.f
```

But, at these stage, It is better to compile with;

```
mpicc -o hello hello.c
```

1.11.3 Running MPI programs

```
mpirun -np 2 hello
```

mpirun is not part of the standard, but some version of it is common with several MPI implementations. The version shown here is for the *MPICH* implementation of MPI.
Another Example (Again no messsage-passing) (hello1.c):

```
#include <stdio.h>
#include <mpi.h>
main(argc, argv)
int argc;
char *argv[];
{
char name[BUFSIZ];
int length;
MPI_Init(&argc, &argv);
MPI_Get_processor_name(name, &length);
printf("%s: hello world\n", name);
MPI_Finalize();
}
```

1.11.4 Finding out about the environment

Two of the first questions asked in a parallel program are: How many processes are there? and Who am I?
How many is answered with ***MPI_Comm_size*** and who am I is answered with ***MPI_Comm_rank***. **The rank** is a number between zero and **size**-1.

1.11.5 A simple program

Again hello (hello2.c);

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

int main( argc, argv )
int argc;
char **argv;
{
int rank, size;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
printf( "Hello world! I'm %d of %d\n",
        rank, size );
MPI_Finalize();
return 0;
}
```

1.11.6 Exercise - Getting Started

Objective: Learn how to login, write, compile, and run a simple MPI program.
Run the "Hello world" programs. Try two different parallel computers. What does the output look like?