Message-Passing Communication

COMPUTAÇÃO DE ALTO DESEMPENHO 2018/2019

HERVÉ PAULINO

SLIDES ADAPTED FROM “STRUCTURED PARALLEL PROGRAMMING - CIS 410/510, UNIVERSITY OF OREGON”

Bibliography

Sections 5.1 to 5.3 of Parallel Programming for Multicore and Cluster Systems (2nd edition), Thomas Rauber and Gudula Rünger. Springer, 2013

Sections 6.1 to 6.3 of Introduction to Parallel Computing (2nd Edition) 2nd Edition, Ananth Grama, George Karypis, Vipin Kumar and Anshul Gupta. Pearson, 2003

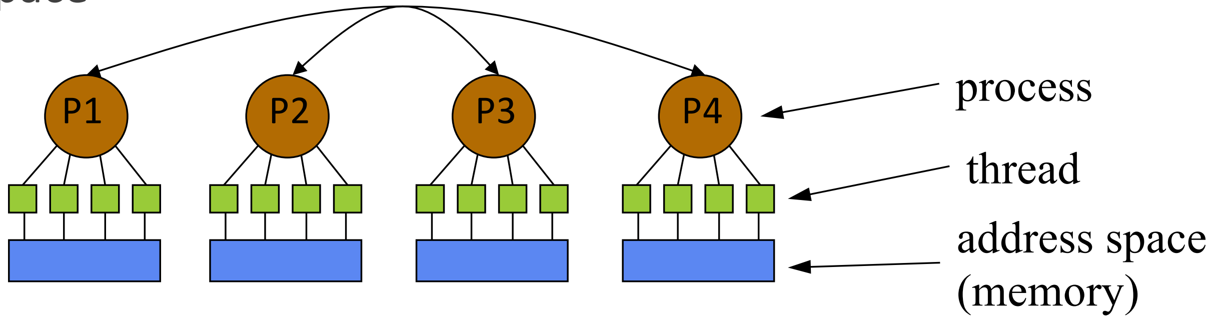
Sections 3.1 to 3.5 of An Introduction to Parallel Programming, Peter Pacheco. Elsevier, 2011.

A Comprehensive MPI Tutorial Resource http://mpitutorial.com

The Message-Passing Model

A process is a program counter and address space

Processes can have multiple threads (program counters and associated stacks) sharing a single address space



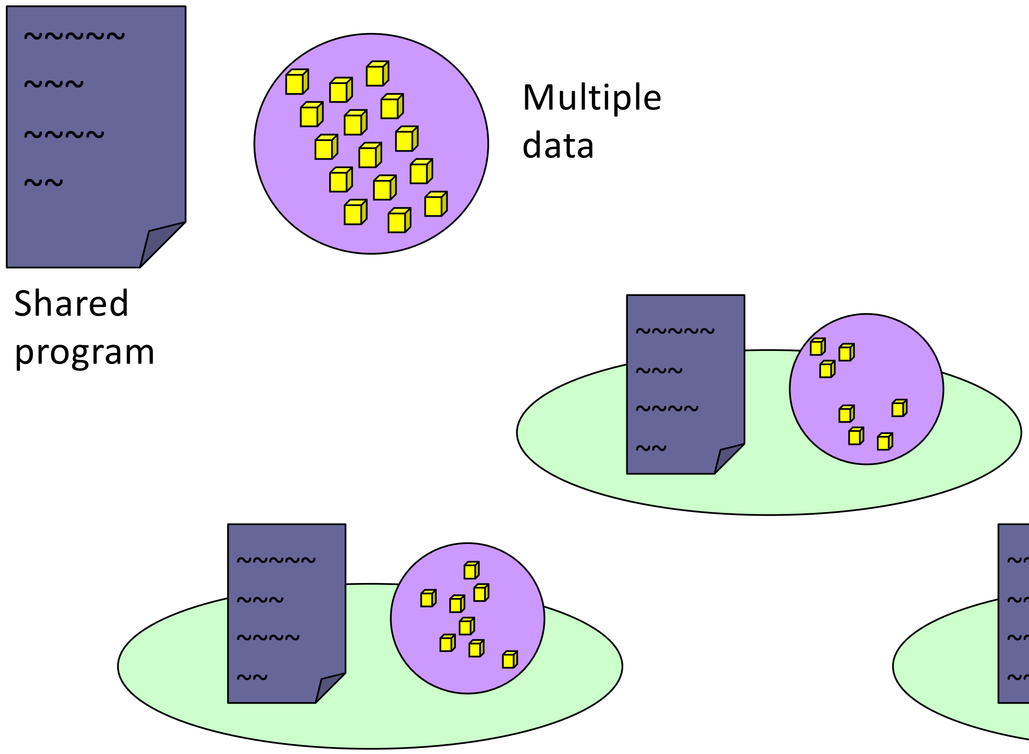
MPI is for communication among processes

- Not threads

Interprocess communication consists of

- Synchronization
- Data movement

Single Program Multiple Data (SPMD)



~~~~~
~~~
~~~~
~~
Multiple data
Shared program

"Owner compute" rule: Process that "owns" the data (local data) performs computations on that data
~~~~~

Message Passing Programming

Defined by communication requirements

- Data communication (necessary for algorithm)
- Control communication (necessary for dependencies)

Program behavior determined by communication patterns

Message passing infrastructure attempts to support the forms of communication most often used or desired

- Basic forms provide functional access
  - Can be used most often
- Complex forms provide higher-level abstractions
  - Serve as basis for extension
  - Example: graph libraries, meshing libraries, …
- Extensions for greater programming power

Communication Types

Two ideas for communication

- Cooperative operations
- One-sided operations

Cooperative Operations for Communication

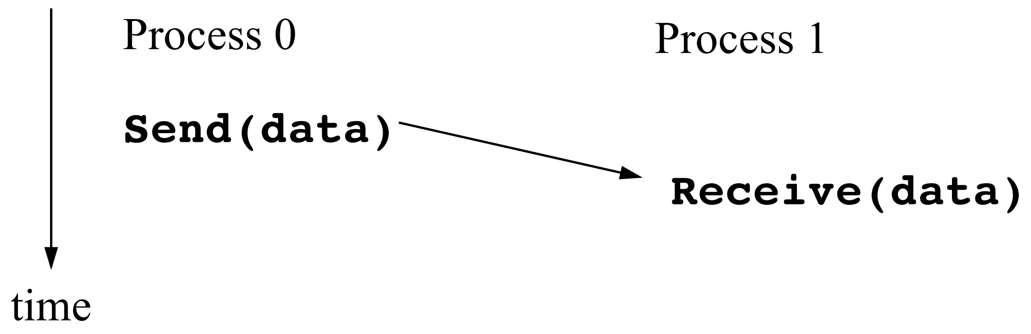
Data is cooperatively exchanged in message-passing

Explicitly sent by one process and received by another

Advantage of local control of memory

- Any change in the receiving process’s memory is made with the receiver’s explicit participation

Communication and synchronization are combined



One-Sided Operations for Communication

One-sided operations between processes

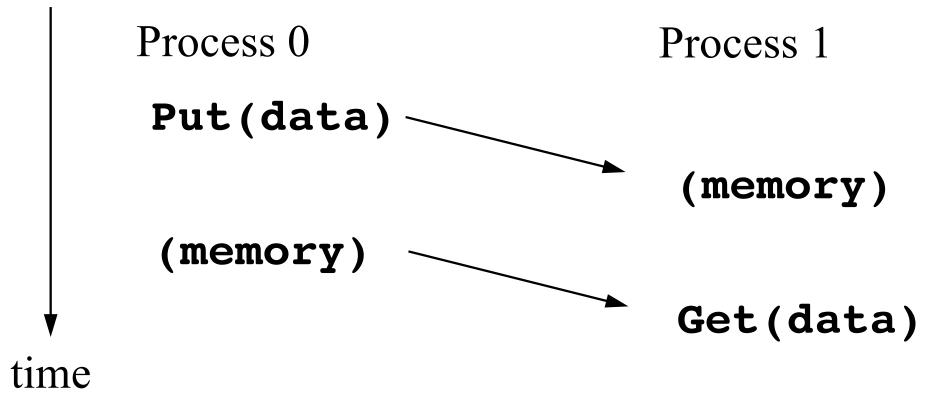
- Include remote memory reads and writes

Only one process needs to explicitly participate

- There is still agreement implicit in the SPMD program

Advantages?

- Communication and synchronization are decoupled



Pairwise vs. Collective Communication

Communication between process pairs

- Send/Receive or Put/Get
- Synchronous or asynchronous (we' ll talk about this later)

Collective communication between multiple processes

- Process group (collective)
  - Several processes logically grouped together
- Communication within group
- Collective operations
  - Communication patterns
    - broadcast, multicast, subset, scatter/gather, …
  - Reduction operations

What is MPI (Message Passing Interface)?

Message-passing library (interface) specification

- Extended message-passing model
- Not a language or compiler specification
- Not a specific implementation or product

Targeted for parallel computers, clusters, and NOWs (network of workstations)

Specified in C, C++, Fortran 77, F90

Full-featured and robust

Designed to access advanced parallel hardware

End users, library writers, tool developers

Three versions: MPI-1, MPI-2, MPI-3 (just released!)

- Robust implementations including free MPICH (ANL)

Message Passing Interface (MPI) Forum

- http://www.mpi-forum.org/
- http://www.mpi-forum.org/docs/docs.html

Features of MPI

General

- Communicators combine context and group for security
- Thread safety (implementation dependent)

Point-to-point communication

- Structured buffers and derived datatypes, heterogeneity
- Modes: normal, synchronous, ready, buffered

Collective

- Both built-in and user-defined collective operations
- Large number of data movement routines
- Subgroups defined directly or by topology

Features of MPI (continued)

Application-oriented process topologies

- Built-in support for grids and graphs (based on groups)

Profiling

- Hooks allow users to intercept MPI calls
- Interposition library interface (PMPI)
- Many tools (e.g., TAU) use PMPI

Environmental

- Inquiry
- Error control

MPI: the Message Passing Interface

The minimal set of MPI routines.

`MPI_Init`	Initializes MPI.
`MPI_Finalize`	Terminates MPI.
`MPI_Comm_size`	Determines the number of processes.
`MPI_Comm_rank`	Determines the label of calling process.
`MPI_Send`	Sends a message.
`MPI_Recv`	Receives a message.

Finding Out About the Environment

Two important questions that arise in message passing

- How many processes are being use in computation?
- Which one am I?

MPI provides functions to answer these questions

- MPI_Comm_size reports the number of processes
- MPI_Comm_rank reports the rank
  - number between 0 and size-1
  - identifies the calling process

Simple Program: “Hello World”

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

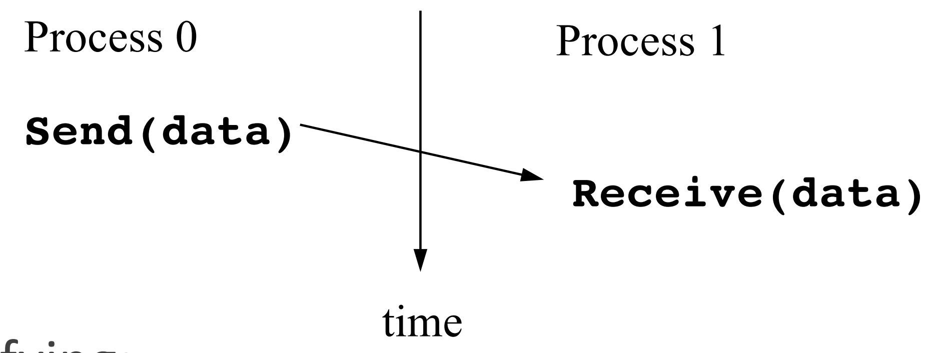
int main( int argc, char *argv[] ) {
   int rank, size;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );
   printf( "I am %d of %d\n", rank, size );
   MPI_Finalize();

   return 0;
}
```

MPI Basic Send/Receive

We need to fill in the details in:

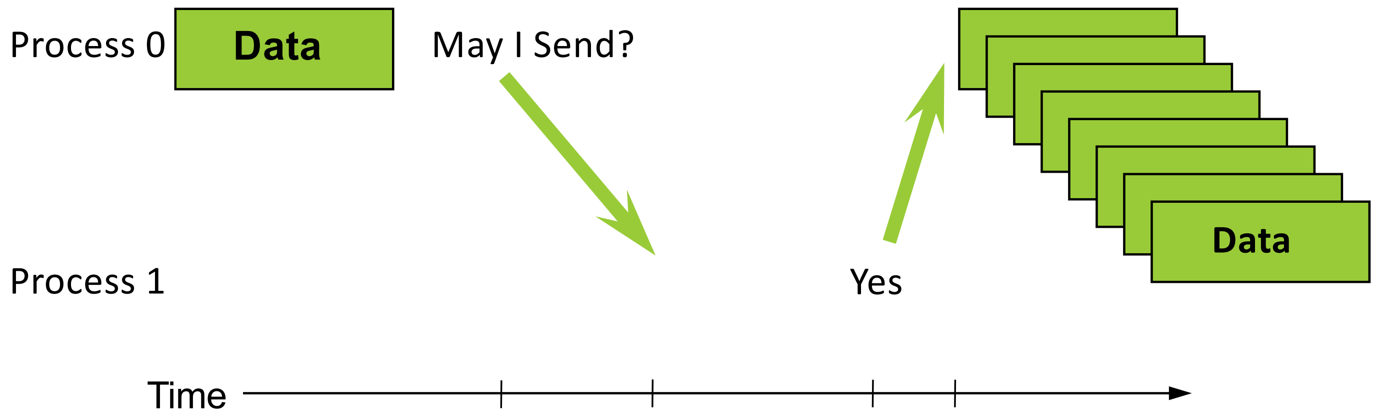


Things that need specifying:

- How will “data” be described?
- How will “processes” be identified?
- How will the receiver recognize/screen messages?
- What will it mean for these operations to complete?

What is message passing?

Data transfer plus synchronization



Requires cooperation of sender and receiver

Cooperation not always apparent in code

Some Basic Concepts

Processes can be collected into groups

Each message is sent in a context

- Must be received in the same context!

A group and context together form a communicator

A process is identified by its rank

- With respect to the group associated with a communicator

There is a default communicator MPI_COMM_WORLD

- Contains all initial processes

MPI Datatypes

Message data (sent or received) is described by a triple

- address, count, datatype

An MPI datatype is recursively defined as:

- Predefined data type from the language
- A contiguous array of MPI datatypes
- A strided block of datatypes
- An indexed array of blocks of datatypes
- An arbitrary structure of datatypes

There are MPI functions to construct custom datatypes

- Array of (int, float) pairs
- Row of a matrix stored columnwise

MPI Tags

Messages are sent with an accompanying user-defined integer tag

- Assist the receiving process in identifying the message

Messages can be screened at the receiving end by specifying a specific tag

- MPI_ANY_TAG matches any tag in a receive

Tags are sometimes called “message types”

- MPI calls them “tags” to avoid confusion with datatypes

Why Datatypes?

All data is labeled by type in MPI

Enables heterogeneous communication

- Support communication between processes on machines with different memory representations and lengths of elementary datatypes
- MPI provides the representation translation if necessary

Allows application-oriented layout of data in memory

- Reduces memory-to-memory copies in implementation
- Allows use of special hardware (scatter/gather)

Overlapping Communication with Computation

In order to overlap communication with computation, MPI provides a pair of functions for performing non-blocking send and receive operations.

- `MPI_Isend`
- `MPI_Irecv`

These operations return before the operations have been completed. Function `MPI_Test` tests whether or not the non-blocking send or receive operation identified by its request has finished.

`MPI_Wait` waits for the operation to complete.

Some Collective Operations in MPI

Called by all processes in a communicator

MPI_BCAST

- Distributes data from one process (the root) to all others

MPI_REDUCE

- Combines data from all processes in communicator
- Returns it to one process

In many numerical algorithms, SEND/RECEIVE can be replaced by BCAST/REDUCE, improving both simplicity and efficiency

Summary

The parallel computing community has cooperated on the development of a standard for message-passing libraries

There are many implementations, on nearly all platforms

MPI subsets are easy to learn and use

Lots of MPI material is available