Message-Passing Programming with MPI

Message-Passing Concepts

David Henty
d.henty@epcc.ed.ac.uk
EPCC, University of Edinburgh
Overview

- This lecture will cover
  - message passing model
  - SPMD
  - communication modes
  - collective communications
# Programming Models

## Serial Programming

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Languages</th>
<th>Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>C/C++</td>
<td>gcc -O3</td>
</tr>
<tr>
<td>Subroutines</td>
<td>Python</td>
<td>icc</td>
</tr>
<tr>
<td>Control flow</td>
<td>Java</td>
<td>crayftn</td>
</tr>
<tr>
<td>Variables</td>
<td>Fortran</td>
<td>craycc</td>
</tr>
<tr>
<td>Human-readable</td>
<td>struct</td>
<td>javac</td>
</tr>
</tbody>
</table>

## Message-Passing Parallel Programming

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Libraries</th>
<th>Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
<td>MPI</td>
<td>Intel MPI</td>
</tr>
<tr>
<td>SPMD</td>
<td>MPICH2</td>
<td>MPICH2</td>
</tr>
<tr>
<td>Send/Receive</td>
<td>OpenMPI</td>
<td>Cray MPI</td>
</tr>
<tr>
<td>Collectives</td>
<td>IBM MPI</td>
<td>IBM MPI</td>
</tr>
<tr>
<td>Groups</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Libraries

- MPI
- MPI_Init()
Message Passing Model

• The message passing model is based on the notion of processes
  – can think of a process as an instance of a running program, together with the program’s data

• In the message passing model, parallelism is achieved by having many processes co-operate on the same task

• Each process has access only to its own data
  – ie all variables are private

• Processes communicate with each other by sending and receiving messages
  – typically library calls from a conventional sequential language
Sequential Paradigm

Message-Passing Programming: Lecture 1

- Memory
- Processor
- Process

Diagram:

- Memory connected to Processor
- Processor connected to Process
- Process connected to Memory

Diagram shows a sequential paradigm with a memory, processor, and process connected in a loop.
Parallel Paradigm

Processes

0 1 2 3

Message Passing Interface

Communication Network
Distributed-Memory Architectures

Interconnect

PM
PM
PM
PM
PM
PM
Process Communication

Process 1

Program

\[ a = 23 \]

\[ \text{Send}(2, a) \]

Data

\[ 23 \]

Process 2

\[ \text{Recv}(1, b) \]

\[ a = b + 1 \]

\[ 24 \]

\[ 23 \]

\[ 23 \]
• Most message passing programs use the Single-Program-Multiple-Data (SPMD) model
• All processes run (their own copy of) the same program
• Each process has a separate copy of the data
• To make this useful, each process has a unique identifier
• Processes can follow different control paths through the program, depending on their process ID
• Usually run one process per processor / core
main (int argc, char **argv)
{
    if (controller_process)
    {
        Controller( /* Arguments */ );
    }
    else
    {
        Worker ( /* Arguments */ );
    }
}
PROGRAM SPMD

    IF (controller_process) THEN
        CALL CONTROLLER ( ! Arguments ! )
    ELSE
        CALL WORKER ( ! Arguments ! )
    ENDIF

END PROGRAM SPMD
Messages

• A message transfers a number of data items of a certain type from the memory of one process to the memory of another process

• A message typically contains
  – the ID of the sending processor
  – the ID of the receiving processor
  – the type of the data items
  – the number of data items
  – the data itself
  – a message type identifier
Communication modes

• Sending a message can either be synchronous or asynchronous

• A synchronous send is not completed until the message has started to be received

• An asynchronous send completes as soon as the message has gone

• Receives are usually synchronous - the receiving process must wait until the message arrives
Synchronous send

- Analogy with faxing a letter.
- Know when letter has started to be received.
Asynchronous send

- Analogy with posting a letter.
- Only know when letter has been posted, not when it has been received.
Point-to-Point Communications

• We have considered two processes
  – one sender
  – one receiver

• This is called point-to-point communication
  – simplest form of message passing
  – relies on matching send and receive

• Close analogy to sending personal emails
Collective Communications

• A simple message communicates between two processes
• There are many instances where communication between groups of processes is required
• Can be built from simple messages, but often implemented separately, for efficiency
Barrier: global synchronisation

Message-Passing Programming: Lecture 1
Broadcast: one to all communication
Broadcast

- From one process to all others
Scatter

- Information scattered to many processes
Gather

• Information gathered onto one process
Reduction Operations

- Combine data from several processes to form a single result

**Strike?**

- [ ] [ ]
- [x] [x]

- [ ] [ ]
- [x] [x]
Reduction

- Form a global sum, product, max, min, etc.
Launching a Message-Passing Program

• Write a *single piece* of source code
  – with calls to message-passing functions such as send / receive

• Compile with a *standard compiler* and link to a *message-passing library* provided for you
  – both open-source and vendor-supplied libraries exist

• Run *multiple copies of same executable* on parallel machine
  – each copy is a separate *process*
  – each has its own private data completely distinct from others
  – each copy can be at a completely different line in the program

• Running is usually done via a launcher program
  – “please run *N* copies of my executable called *program.exe*”
Issues

• Sends and receives must match
  – danger of deadlock
  – program will stall (forever!)

• Possible to write very complicated programs, but …
  – most scientific codes have a simple structure
  – often results in simple communications patterns

• Use collective communications where possible
  – may be implemented in efficient ways
Summary (i)

• Messages are the *only* form of communication
  – all communication is therefore explicit

• Most systems use the SPMD model
  – all processes run exactly the same code
  – each has a unique ID
  – processes can take different branches in the same codes

• Basic communications form is point-to-point
  – collective communications implement more complicated patterns that often occur in many codes
• Message-Passing is a programming model
  – that is implemented by MPI
  – the Message-Passing Interface is a library of function/subroutine calls

• Essential to understand the basic concepts
  – private variables
  – explicit communications
  – SPMD

• Major difficulty is understanding the Message-Passing model
  – a very different model to sequential programming

```python
if (x < 0):
    print("Error");
    exit;
```