Introduction to MPI
Concepts and Strategies

A. Balaz
Institute of Physics
Belgrade
Republic of Serbia
Introduction to MPI: Concepts and Strategies

Antun Balaz
Scientific Computing Laboratory
Institute of Physics Belgrade, Serbia
http://www.scl.rs/
Overview

- Why serial is not enough
- Computing architectures
- Parallel paradigms
- Message Passing Interface
- How to compile and run MPI programs
Serial computing

- Using a single computer to complete a single task
  - concurrent computing
- To improve performance
  - Optimize program code
  - Use mathematical libraries
  - Improve the hardware
    - Moore’s law - empirical observation made in 1965 that the number of transistors on an integrated circuit for minimum component cost doubles every 24 months.
    - Bigger, faster and more memory (DDR3, FBDIMMS)
    - More storage!

Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
Why serial is not enough

- Realistic simulations require really really
  - Large numbers of particles
  - Large MC samples
  - Large statistics
  - Combinatorially large spaces to be searched
  - Excessively fine multidimensional discretizations
  - Huge data inputs to be processed
  - …

- We want to solve problems harder, faster, better, stronger!
- Parallel hardware is available (clusters)
- Parallel software is available (libraries)
- And we want to learn something new…
Modern computing architectures

- Shared memory (SMP)
  - Single large system where all CPUs can access the whole available memory

- Distributed memory
  - Each CPU can access only local memory attached to it (nodes with one single-core CPU)

- Hybrid systems (majority of clusters)
  - Nodes with several single-core CPUs
  - Nodes with a single multicore CPU
  - Nodes with several multicore CPUs
Parallel paradigms (1)

- The two (three) architectures determine two basic paradigms
  - Data parallel (shared memory)
    - Single memory view, all processes (usually threads) could **directly access the whole memory**
  - Message Passing (distributed memory)
    - All processes could **directly access only their local memory**
Parallel paradigms (2)

- It is easy to adopt a Message Passing scheme in a Shared Memory computers (*Unix processes have their private memory*)
- It is less easy to follow a Data Parallel scheme in a Distributed Memory computer (*emulation of shared memory*)
- It is relatively easy to design a program using the message passing scheme and implementing the code in a Data Parallel programming environments (*using OpenMP or HPF*)
- It is not easy to design a program using the Data Parallel scheme and implementing the code in a Message Passing environment.
## Parallel paradigms (3)

<table>
<thead>
<tr>
<th>Programming environments</th>
<th>Message Passing</th>
<th>Data Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard compilers</td>
<td>Ad hoc compilers</td>
<td></td>
</tr>
<tr>
<td>Communication libraries</td>
<td>Source code directive</td>
<td></td>
</tr>
<tr>
<td>Ad hoc commands to run program</td>
<td>Standard Unix shell to run program</td>
<td></td>
</tr>
<tr>
<td>Standard: MPI</td>
<td>Standard: OpenMP</td>
<td></td>
</tr>
</tbody>
</table>
## Parallel paradigms (4)

<table>
<thead>
<tr>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed memory</td>
</tr>
<tr>
<td>Shared memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming paradigm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message passing</td>
</tr>
<tr>
<td>Data parallel</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain decomposition</td>
</tr>
<tr>
<td>Functional decomposition</td>
</tr>
</tbody>
</table>
Programming models

- **Domain decomposition**
  - Data divided into equal chunks and distributed to available CPUs
  - Each CPU process its own local data
  - Exchange of data if needed

- **Functional decomposition**
  - Problem decomposed into many sub-tasks
  - Each CPU performs one of sub-tasks
  - Similar to server/client paradigm
Flint’s taxonomy (1)

- **SISD** (Single instruction, single data)
- **SIMD** (Single instruction, multiple data)
  - the same instructions are carried out simultaneously on multiple data items
  - SSE is a good example
- **MISD** (Multiple instruction, single data)
- **MIMD** (Multiple instruction, multiple data)
  - different instructions on different data
- **SPSD** (Single program, single data)
- **SPMD** (Single program, multiple data)
  - not synchronized at individual operation level
  - equivalent to MIMD since each MIMD program can be made SPMD
Flint’s taxonomy (2)

- SPSD (Single program, single data)
- SPMD (Single program, multiple data)
  - not synchronized at individual operation level
  - equivalent to MIMD since each MIMD program can be made SPMD
- MPSD (Multiple program, single data)
- MPMD (Multiple program, multiple data)
## Parallel paradigms (5)

<table>
<thead>
<tr>
<th>Model</th>
<th>Paradigm</th>
<th>Flint’s taxonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain decomposition</td>
<td>Message Passing</td>
<td>SPMD</td>
</tr>
<tr>
<td>Data Parallel - HPF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional decomposition</td>
<td>Data Parallel - OpenMP</td>
<td>MPSD</td>
</tr>
<tr>
<td>Message Passing</td>
<td></td>
<td>MPMD</td>
</tr>
</tbody>
</table>

- **Domain decomposition**: Message Passing, Data Parallel - HPF
- **Functional decomposition**: Data Parallel - OpenMP, Message Passing
Parallelism requires…

- Balancing of the load
  - Applies to computation, I/O operations, network communication
  - Relatively easy for domain decomposition, not so easy for functional decomposition

- Minimizing communication
  - Join individual communications
  - Eliminate synchronization – the slowest process dominates

- Overlap of computation and communication
  - This is essential for true parallelism!
Effective parallel performance

Amdahl’s law:
Parallel speedup vs. Sequential fraction

Number of processors

Speedup

0.1
0.2
0.5
Linear

Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
Message Passing

- Parallel programs consist of separate processes, each with its own address space
  - Programmer manages memory by placing data in a particular process
- Data sent explicitly between processes
  - Programmer manages memory movement
- Collective operations
  - On arbitrary set of processes
- Data distribution
  - Also managed by the programmer
Distributed memory

- Nothing is shared between processes
What is MPI? (1)

- Message Passing Interface
- A message-passing library specification
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for end users, library writers, and tool developers
What is MPI? (2)

MPI is a standard

- A list of rules and specifications
- Left up to individual implementations as to how it is implemented.
- There are several implementations available over several different networks

Goals of MPI

- To provide source-code portability
  - Virtually every supercomputer on Earth can use MPI
- To allow efficient implementation of parallel computing
MPI references

- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML

- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of stuff, including talks and tutorials, a FAQ, other MPI pages
MPI Implementations

Because MPI is a standard, there are several implementations

  - Freely available, portable implementation
  - Available on everything

- OpenMPI - http://www.open-mpi.org/
  - Includes the once popular LAM-MPI

- Vendor specific implementations
  - CRAY, SGI, IBM
MPI-1 vs. MPI-2

MPI-1

- Specifies traditional sender/reciever message passing scheme
- Two-sided communication model
- Communication involves both the sender and receiver
- Limited and not completely scalable without Herculean effort

MPI-2

- Implements many concepts that became popular since MPI-1
- Remote memory access, parallel I/O and dynamic processing
- One-sided communication model
- All communication parameters are handled by one process
OpenMPI

- Open source implementation of MPI-2
  - Single library supports all networks
    - TCP, Myrinet, InfiniBand
  - Network and process fault tolerance
  - VampirTrace
    - Performance analysis
    - Visualisation
When do you need 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
- You care about performance
When MPI is not needed?

- You can use parallel Fortran 90 or any other data parallelism mechanism
- You don’t need parallelism at all
- You can use libraries (which may be written in MPI)
- You need simple threading in a slightly concurrent environment
Writing MPI program

- MPI is a library
- All operations are performed with function (subroutine) calls
- Basic definitions are in
  - mpi.h for C/C++
  - mpif.h for Fortran 77 and 90
  - MPI module for Fortran 90 (optional)
MPI functions

Functions may be roughly divided into 4 classes:

- Calls used to initialize, manage, and terminate communications
- Calls used to communicate between pairs of processes (Point-to-point communication)
- Calls used to communicate among groups of processes (Collective communication)
- Calls to create data types
Hello, MPI world program

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

int main(int argc, char **argv )
{
    MPI_Init(&argc, &argv);
    printf("Hello, MPI world!\n");
    MPI_Finalize();
    return 0;
}
```
How to compile an MPI program?

- No standard, left to implementations

- Generally:
  - You should specify the appropriate include directory:
    - -I/mpidir/include
  - You should specify the mpi library
    - -L/mpidir/lib -lmpi
  - With GCC
    - gcc -I/usr/local/mpich/include -L/usr/local/mpich/lib
      -mpi -o mpi-hello
  - Usually MPI compiler wrappers do this job for you. (i.e. mpicc, mpif77, mpif90, mpicxx)
    - mpicc -o mpi-hello mpi-hello.c
  - Check on your machine...
Example: MPI ID program

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

int main(int argc, char **argv)
{
    int myid, np;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    printf("Process %d out of %d\n", myid, np);
    MPI_Finalize();
    return 0;
}
```
The MPI-1 Standard does not specify how to run an MPI program, just as the Fortran standard does not specify how to run a Fortran program.

Many implementations provide mpirun to run an MPI program

mpirun –np 4 mpi-hello

In general, starting an MPI program depends on the implementation of MPI you are using, and might require various scripts, program arguments, and/or environment variables.

mpixec is part of MPI-2, as a recommendation, but not as a requirement

Many parallel systems use a *batch environment to share resources among users*

The specific commands to run a program on a parallel system are defined by the environment installed on the parallel computer
What is next?

- Learn MPI function types and syntax
- Learn how to compile and run MPI programs on a single node
- Learn how to run MPI programs on a cluster, in batch mode
- If this is not enough, use the Grid!
  - One Ring to rule them all,
    One Ring to find them,
    One Ring to bring them all,
    and in the darkness bind them