MPI / OpenMP Track
IHPCSS 2016, Ljubljana

Overview

David Henty
d.henty@epcc.ed.ac.uk
EPCC, University of Edinburgh
Who am I?

- David Henty
  - EPCC (Edinburgh Parallel Computing Centre)
    - University of Edinburgh, Scotland, UK
  - background in theoretical particle physics
    - (computational)
  - at EPCC since 1995
  - in charge of training including our 1-year masters course in HPC, PRACE Advanced Training Centre, ARCHER training, ...
  - generally interested in parallel languages and models

- EPCC runs the UK national supercomputer ARCHER
  - Cray XC30 with 118,000 cores
  - around 70 full time staff
  - a range of work: national systems, research projects, European collaborations, MSc in HPC, commercial software development, ...
Edinburgh
Overview

• An introduction to
  – message-passing programming with MPI
  – shared-memory programming with OpenMP
  – hybrid (both MPI and OpenMP at the same time)

• Assumptions
  – you have used MPI
  – you have some knowledge of OpenMP
  – you have looked at the background material:
    – www.xsede.org/web/international-hpc-summer-school/2016-wiki
    – see Hands-on Session Prerequisites -> MPI/OpenMP (Classic Track)

• All exercises are based around the parallel traffic model
Materials

- Slides on XSEDE wiki
  - [www.xsede.org/web/international-hpc-summer-school/2016-wiki](http://www.xsede.org/web/international-hpc-summer-school/2016-wiki)

- Also at: [tinyurl.com/ihpcss-mpi-openmp](http://tinyurl.com/ihpcss-mpi-openmp)

- Additional material other than slides:
  - Instructions for running on Bridges: [Bridges-cribsheet.pdf](http://Bridges-cribsheet.pdf)
  - MPI/OpenMP exercise sheet: [traffic-ihpcss16.pdf](http://traffic-ihpcss16.pdf)
  - MPI/OpenMP codes: [IHPCSS-pi.tar](http://IHPCSS-pi.tar) and [IHPCSS-traffic.tar](http://IHPCSS-traffic.tar)
  - Challenge code: [challenge.tar](http://challenge.tar)
Timetable: Monday

- 13:30 Introduction and recap
- 14:00 Log on; walkthrough of pi example
- 14:30 Communicators, tags and modes
- 15:00 Break
- 15:30 Non-blocking communications
- 16:15 Practical session: traffic model
- 17:30 Close
Timetable: Tuesday

• 11:45 OpenMP overview
• 12:15 Walkthrough of pi example
• 12:30 Lunch
• 13:30 Advanced worksharing and orphaning
• 14:15 Practical session: traffic model
• 15:00 Coffee
• 15:30 Hybrid MPI / OpenMP
• 16:15 Practical session
• 17:15 HPC Challenge example
• 17:30 Close
• A challenge to teach an audience with such a wide variety of previous experiences ...

• Practical
  – a range of options from basic to advanced
  – identical parallelisation to HPC challenge so a useful playground

• Lectures
  – I am happy to cover whatever you want to know
  – let me know!
Message-Passing

Parallel Programming using Processes
Outline

• Message-Passing Parallelism
  • processes
  • messages
  • communications patterns

• Practicalities
  • usage on real HPC architectures
Analogy

• Two whiteboards in different single-person offices
  • the distributed memory

• Two people working on the same problem
  • the processes on different nodes attached to the interconnect

• How do they collaborate?
  • to work on single problem

• Explicit communication
  • e.g. by telephone
  • no shared data
Process communication

Process 1

Program

Data

Process 2
Process communication

Program

Process 1

a = 23

Data

Process 2
Process communication

Process 1

Program

a = 23

Data

23

Process 2
Process communication

Process 1

Program

a = 23
Send(2, a)

Data

23

Process 2
Process communication

Process 1

Program

a = 23
Send(2, a)

Data

23

Process 2

23
Process communication

Process 1

Program
a = 23
Send(2, a)

Data

23

Process 2

Recv(1, b)

23
Process communication

Program

Process 1

a=23
Send(2, a)

Process 2

Recv(1, b)

Data
Process communication

Program

Process 1
a = 23
Send(2, a)

Process 2
Recv(1, b)
a = b + 1

Data

23

23

23

23
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]
Synchronisation

• Synchronisation is automatic in message-passing
  • the messages do it for you

• Make a phone call …
  • … wait until the receiver picks up

• Receive a phone call
  • … wait until the phone rings

• No danger of corrupting someone else’s data
  • no shared blackboard
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
Broadcast: one to all communication
Broadcast

- From one process to all others
Broadcast

- From one process to all others
Broadcast

- From one process to all others
Broadcast

- From one process to all others
Scatter

- Information scattered to many processes
Scatter

- Information scattered to many processes
Scatter

- Information scattered to many processes
Gather

- Information gathered onto one process
Gather

- Information gathered onto one process
Gather

- Information gathered onto one process
Reduction Operations

• Combine data from several processes to form a single result

Strike?
Reduction

- Form a global sum, product, max, min, etc.
Reduction

- Form a global sum, product, max, min, etc.
Hardware

• Natural map to distributed-memory
  • one process per processor-core
  • messages go over the interconnect, between nodes/OS’s
Practicalities

- 8-core machine might only have 2 nodes
  - how do we run MPI on a real HPC machine?

- Mostly ignore architecture
  - pretend we have single-core nodes
  - one MPI process per processor-core
  - e.g. run 8 processes on the 2 nodes

- Messages between processes on the same node are fast
  - but remember they also share access to the network
Message Passing on Shared Memory

• Run one process per core
  • don’t directly exploit shared memory
  • analogy is phoning your office mate
  • actually works well in practice!

• Message-passing programs run by a special job launcher
  • user specifies #copies
  • some control over allocation to nodes
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
  • Single Program Multiple Data
  • 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
Summary (ii)

• **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

```c
if (x < 0)
    print(“Error”);
exit;
```
Exercise: computing pi

An approximation to the value $\pi$ can be obtained from the following expression

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1 + x^2} \approx \frac{1}{N} \sum_{i=1}^N \frac{1}{1 + \left(\frac{i - \frac{1}{2}}{N}\right)^2}$$

where the answer becomes more accurate with increasing $N$. Iterations over $i$ are independent so the calculation can be parallelised.

- Will use this as a simple example for MPI and OpenMP

- Traffic Model (see later) is a much better analogue of a real simulation code
  - but pi calculation illustrates basic concepts