

Parallel Models

Different ways to exploit parallelism

The logo for EPSRC (Engineering and Physical Sciences Research Council) features the acronym in a bold, purple, sans-serif font. It is framed by two horizontal teal lines, one above and one below the text.The logo for NERC (Natural Environment Research Council) consists of the acronym 'NERC' in white, bold, sans-serif font on a dark olive green rectangular background. To its right, the words 'SCIENCE OF THE ENVIRONMENT' are written in a smaller, white, sans-serif font on a light yellow-green rectangular background.The logo for the ARCHER project features a stylized target icon on the left, composed of concentric red and white circles. To the right of the icon, the word 'archer' is written in a white, lowercase, sans-serif font on a black rectangular background.The logo for Cray, 'THE SUPERCOMPUTER COMPANY', features the word 'CRAY' in a large, blue, stylized, sans-serif font. Below it, the full name 'THE SUPERCOMPUTER COMPANY' is written in a smaller, blue, sans-serif font.The logo for EPCC (Edinburgh Parallel Computing Centre) features the lowercase letters 'epcc' in a blue, sans-serif font. The text is flanked by two vertical red lines on either side.

Outline

- Shared-Variables Parallelism
 - threads
 - shared-memory architectures
- Message-Passing Parallelism
 - processes
 - distributed-memory architectures
- Practicalities
 - compilers
 - libraries
 - usage on real HPC architectures



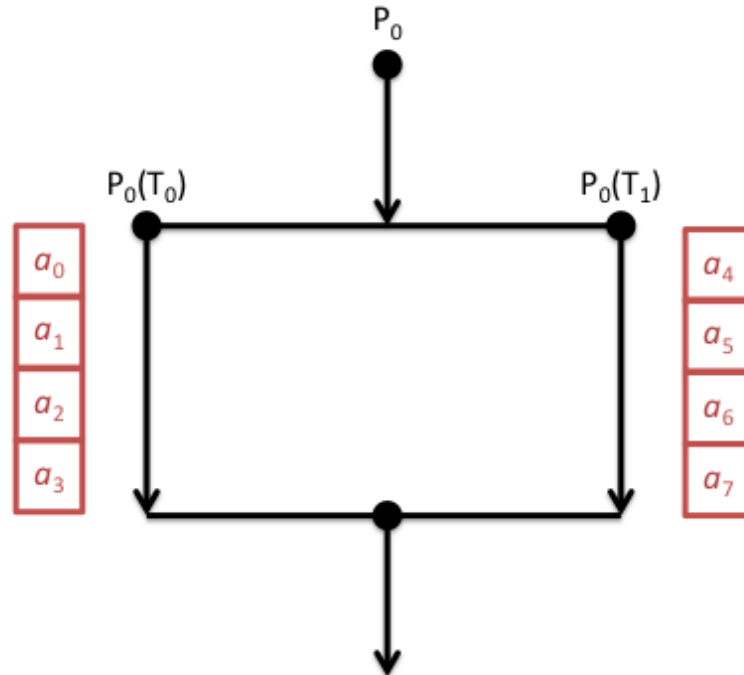
Shared Variables

Threads-based parallelism



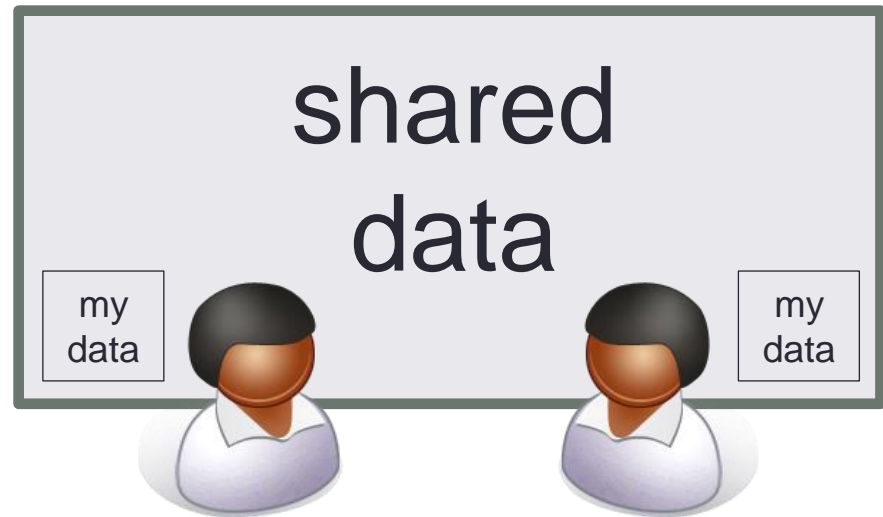
Shared-memory concepts

- Have already covered basic concepts
 - threads can all see data of parent process
 - can run on different cores
 - potential for parallel speedup



Analogy

- One very large whiteboard in a two-person office
 - the shared memory
- Two people working on the same problem
 - the threads running on different cores attached to the memory
- How do they collaborate?
 - working together
 - but not interfering
- Also need *private* data



Thread Communication

Thread 1

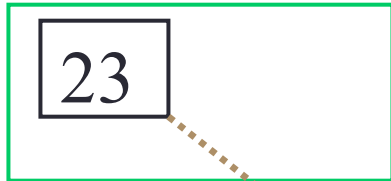
Thread 2

Program

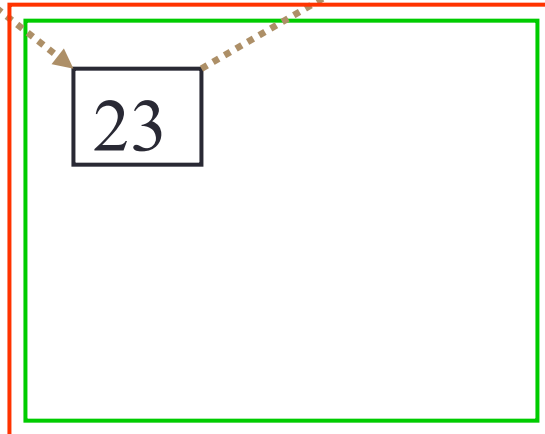
```
mya=23  
a=mya
```

```
mya=a+1
```

Private
data



Shared
data



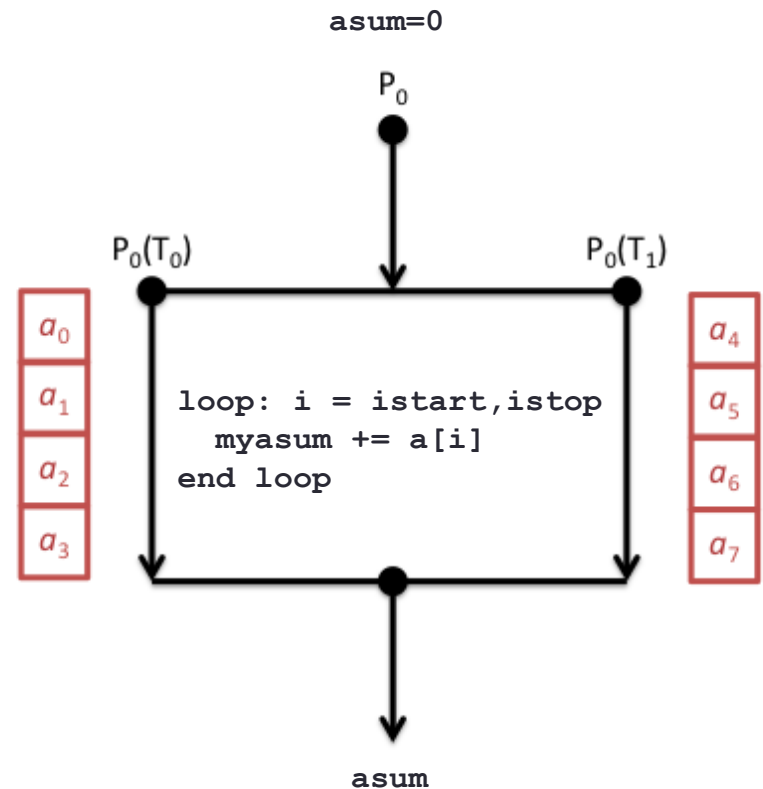
Synchronisation

- Synchronisation crucial for shared variables approach
 - thread 2's code must execute *after* thread 1
- Most commonly use global barrier synchronisation
 - other mechanisms such as locks also available
- Writing parallel codes relatively straightforward
 - access shared data as and when its needed
- Getting correct code can be difficult!



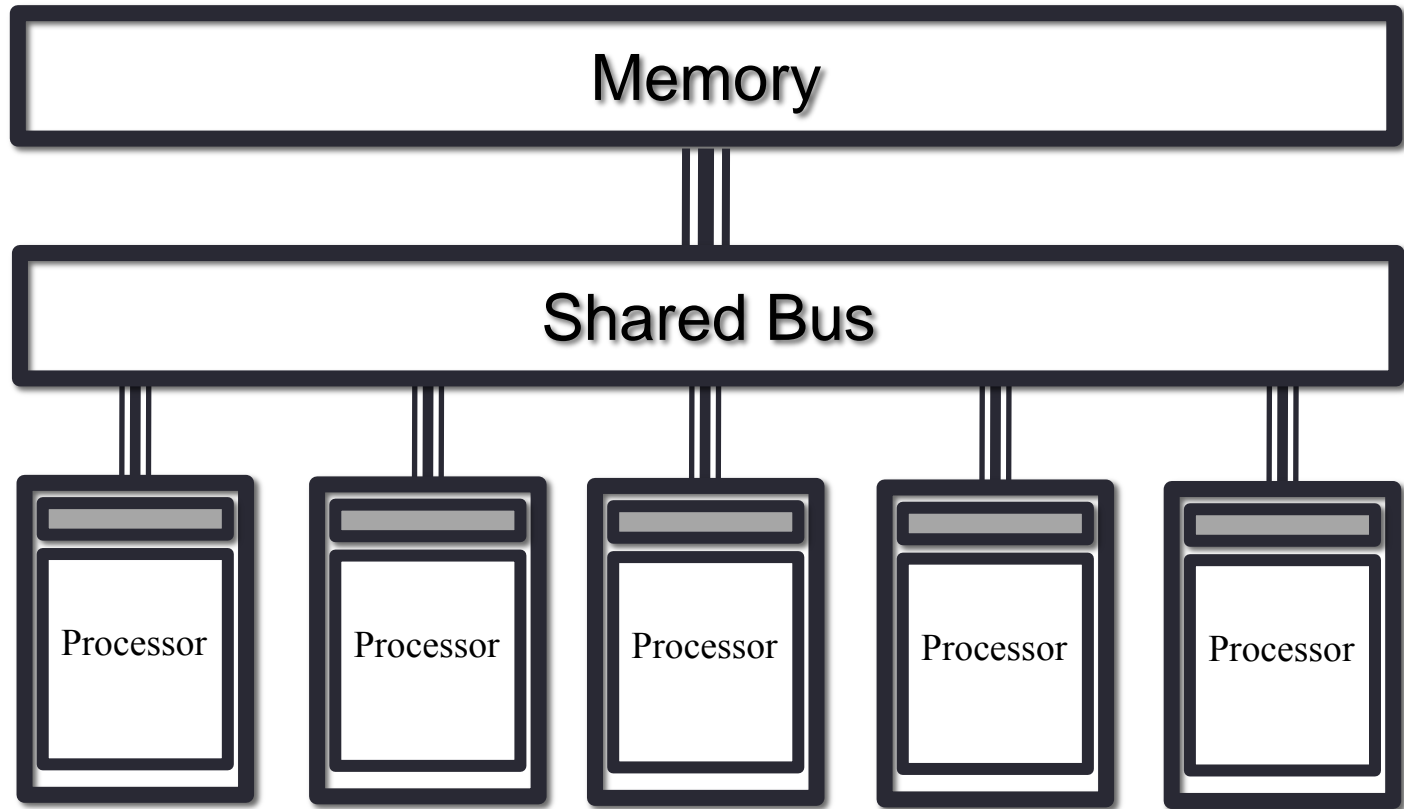
Specific example

- Computing $asum = a_0 + a_1 + \dots + a_7$
 - shared:
 - main array: **a [8]**
 - result: **asum**
 - private:
 - loop counter: **i**
 - loop limits: **istart, istop**
 - local sum: **myasum**
 - synchronisation:
 - thread0: **asum += myasum**
 - barrier
 - thread1: **asum += myasum**



Hardware

- Needs support of a shared-memory architecture



Threads: Summary

- Shared blackboard a good analogy for thread parallelism
- Requires a shared-memory architecture
 - in HPC terms, cannot scale beyond a single node
- Threads operate independently on the shared data
 - need to ensure they don't interfere; synchronisation is crucial
- Threading in HPC usually uses OpenMP directives
 - supports common parallel patterns
 - e.g. loop limits computed by the compiler
 - e.g. summing values across threads done automatically



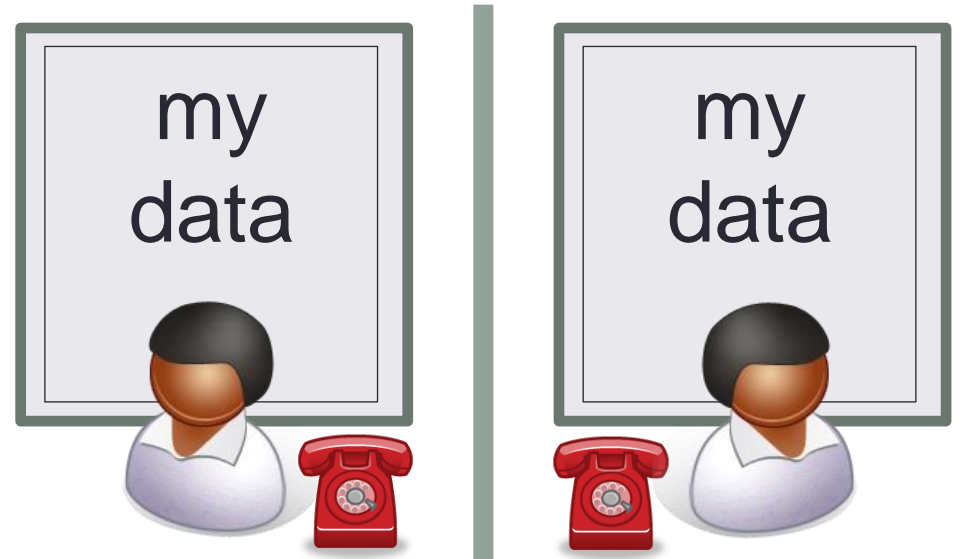
Message Passing

Process-based parallelism



Analogy

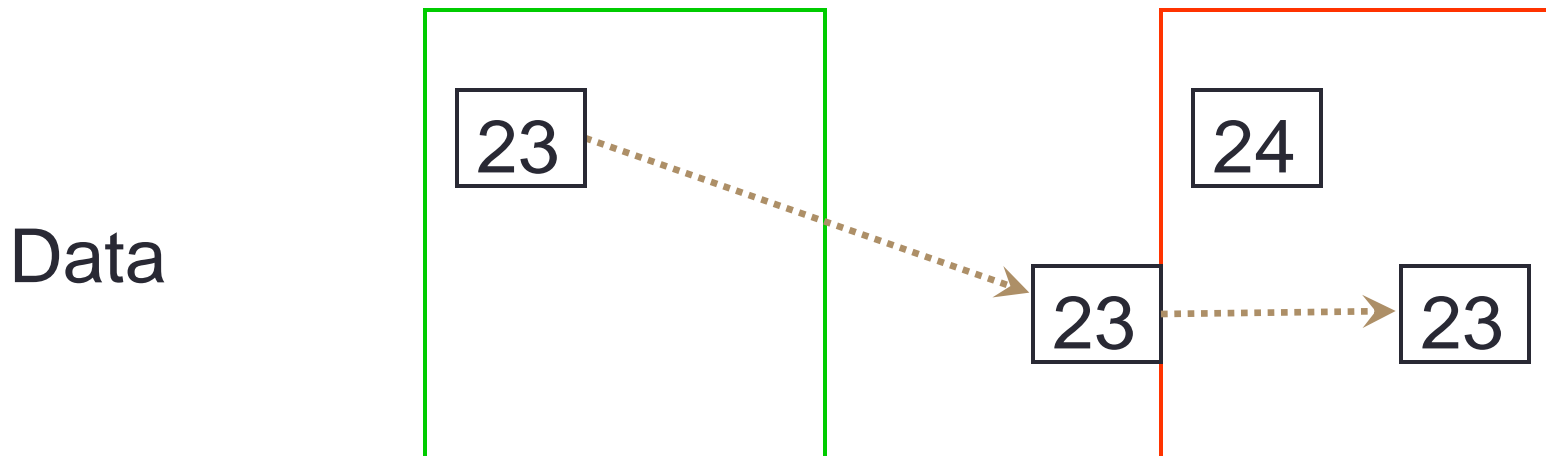
- Two whiteboards in different single-person offices
 - the distributed memory
- Two people working on the same problem
 - the processes on different nodes attached to the interconnect
- How do they collaborate?
 - to work on single problem
- Explicit communication
 - e.g. by telephone
 - no shared data



Process communication

Program

Process 1	Process 2
<code>a=23</code>	<code>Recv (1, b)</code>
<code>Send (2, a)</code>	<code>a=b+1</code>

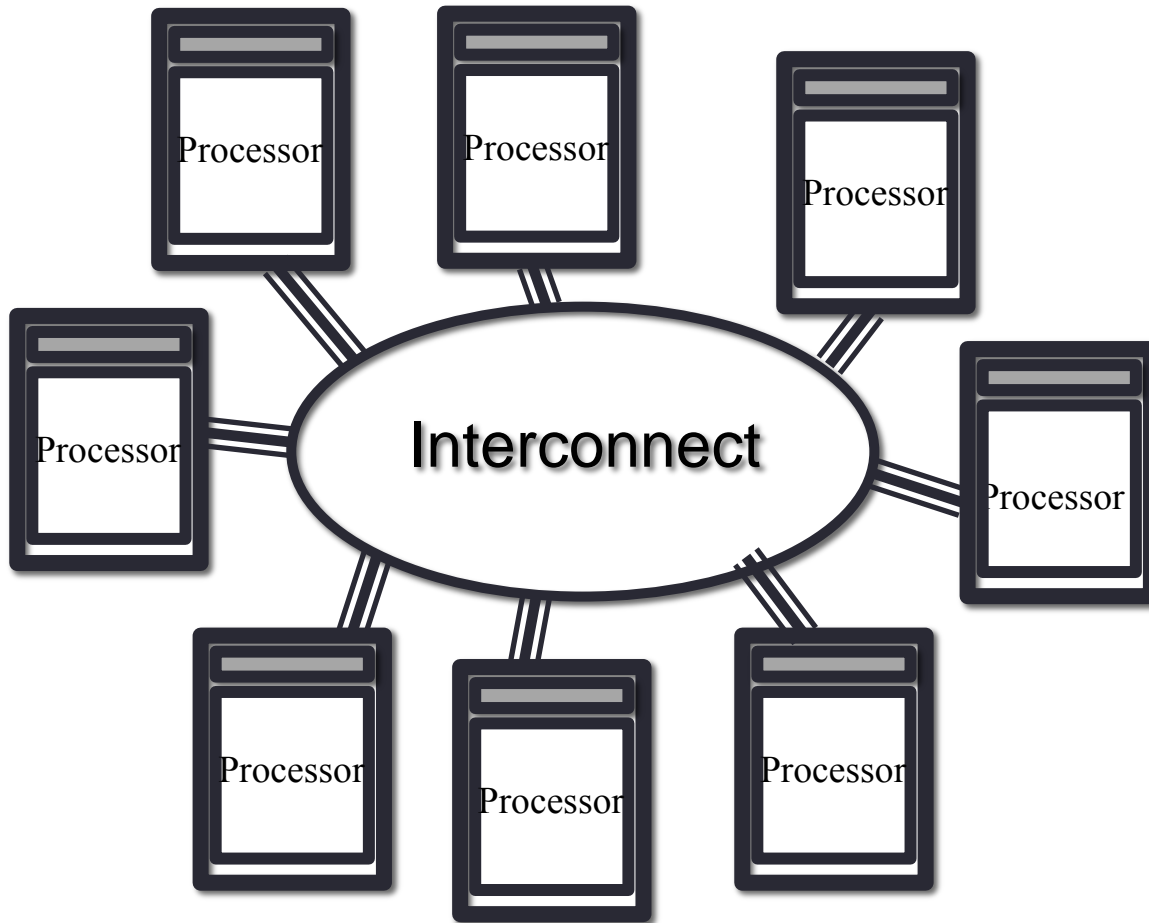


Synchronisation

- 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



Hardware



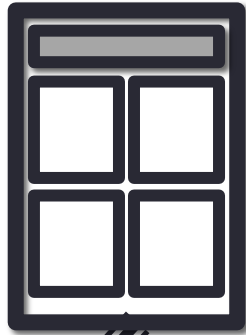
- Natural map to distributed-memory
 - one process per processor-core
 - messages go over the interconnect, between nodes/OS's

Processes: Summary

- Processes cannot share memory
 - ring-fenced from each other
 - analogous to white boards in separate offices
- Communication requires explicit *messages*
 - analogous to making a phone call, sending an email, ...
 - synchronisation is done by the messages
- Almost exclusively use Message-Passing Interface
 - MPI is a library of function calls / subroutines



Practicalities



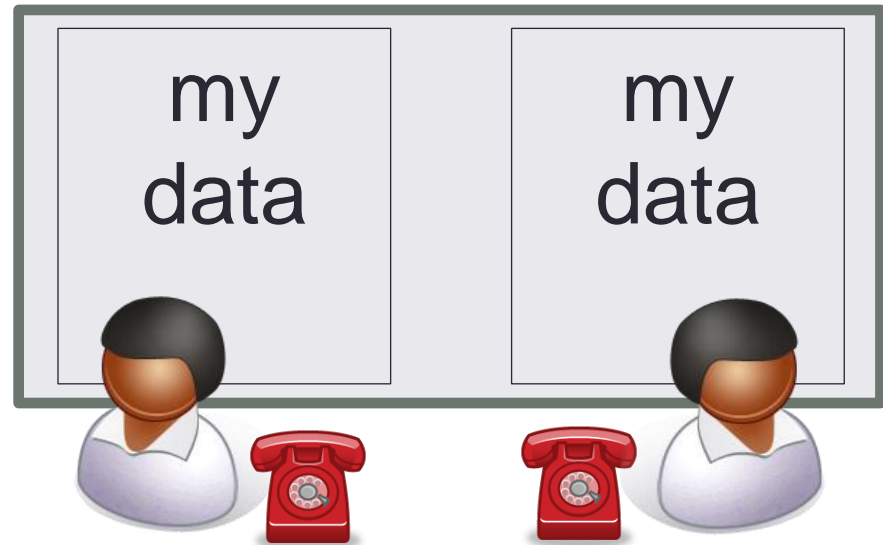
Interconnect



- 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



Summary

- Shared-variables parallelism
 - uses threads
 - requires shared-memory machine
 - easy to implement but limited scalability
 - in HPC, done using OpenMP compilers
- Distributed memory
 - uses processes
 - can run on any machine: messages can go over the interconnect
 - harder to implement but better scalability
 - on HPC, done using the MPI library

