Chapter 31
Introduction to Parallel Programming

'Can you do addition?' the White Queen asked. 'What's one and one and one and one and one and one and one and one and one and one?'
'I don't know' said Alice. 'I lost count.'
'She can't do addition,' the Red Queen interrupted.
Lewis Carroll, Through the Looking Glass and What Alice Found There

Aims
The aims of this chapter is to provide a short introduction to parallel programming.

31.1 Introduction

Parallel programming involves breaking a program down into parts that can be executed concurrently. Here is a simple diagram to illustrate the idea.

```
Sequential  Parallel    Step
Execution  Execution
   |           |         1
              / \        #
   |         |   |       2
   |         |   |       3
              \ /        @
   |           |         4
```

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_31

```
     |              |        5
                   / \      ##
     |            |   |      6
     |            |   |      7
                   \ /      @@
     |              |        8
```

On the left hand side we have a sequential program and this steps through linearly from beginning to end. The right hand side has the same program that has been partially parallelised. There are two parallel regions and the work here is now shared between two processes or threads. At each parallel part of the program we have the following

	Parallel Region 1	Parallel Region 2
Set up cost	Step #	Step ##
Parallel section	Steps 2,3	Steps 6,7
Synchronisation cost	Step @	Step @@

The theory is that the overall run time of the program will have been reduced or we will have been able to solve a larger problem by parallelising our code. In the above example we have divided the work between two processes or threads. Here are some details of a range of processors which support multiple cores. Visit the AMD and Intel sites for up to date information.

Processor	Cores	Hyper Threading
AMD Phenom II X6	6	
Intel Core i7 920	4	* 2
Intel Core i7 2600K	4	* 2
AMD Opteron Shanghai	4	
Istanbul	6	
Magny Cours	8	
Magny Cours	12	
Intel E5-2697	12	* 2

Intel introduced hyperthreading technology in 2002. For each physical processor core the Intel chip has the operating system can see or address two virtual or logical cores, and can share the workload between them when possible. See the Wikipedia entry for more information.

```
http://en.wikipedia.org/wiki/Hyper-threading
```

There are several ways of doing parallel programming, and this chapter will look at three ways of doing this in Fortran. There are a common set of concepts and terminology that are useful to know about, whichever method we use, and we will cover these first.

31.2 Parallel Computing Classification

Parallel computing is often classified by the way the hardware supports parallelism. Two of the most common are:

- multi-processor and multi-core computers having multiple processing elements within a single system
- clusters or grids with multiple computers connected to work together.

Modern large systems are increasingly hybrids of the two above.

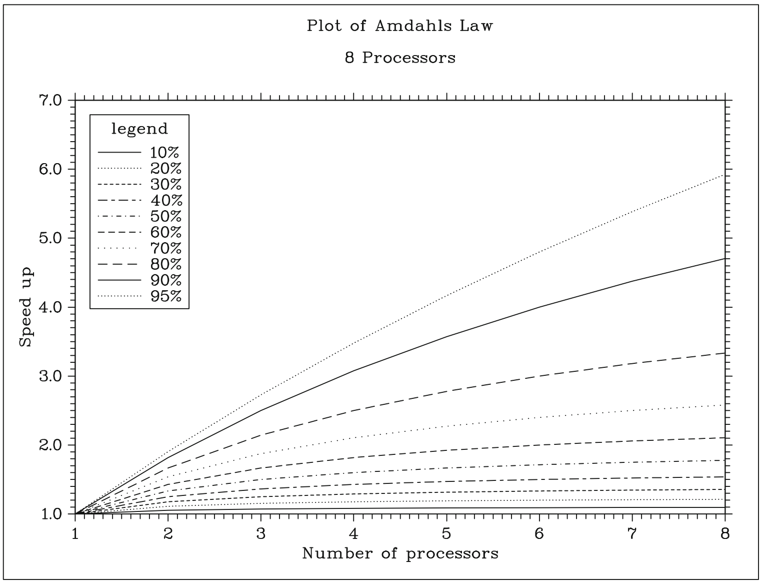
31.3 Amdahl's Law

Amdahl's law is a simple equation for the speedup of a program when parallelised. It assumes that the problem size remains the same when parallelised. In the equation below

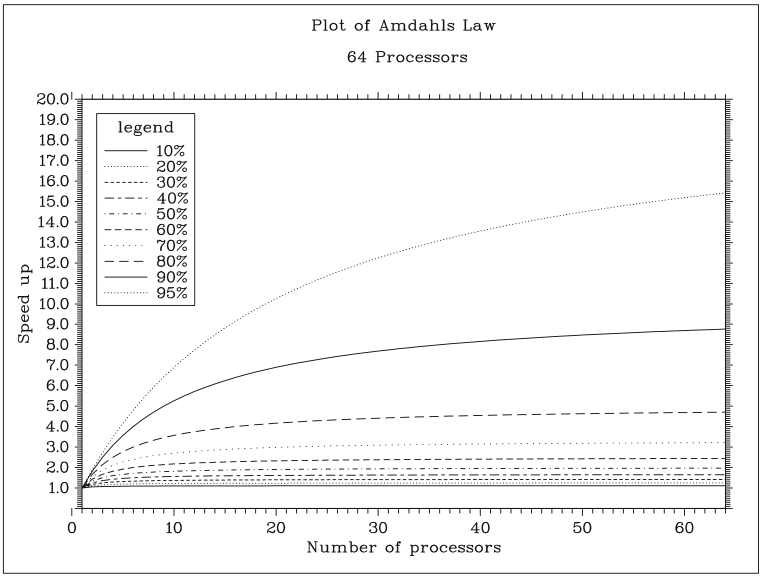
- P is the proportion of the program that can be parallelised
- (1-P) is the serial proportion
- N is the number of processors
- speedup = 1 / ((1-P) + P/N).

We have included a couple of graphs to illustrate the above. We have written programs that use the dislin graphics library to do the plots. More information on these programs can be found in Chap. 35, where we have a look at third party numeric and graphics libraries.

31.3.1 Amdahl's Law Graph 1–8 Processors or Cores



31.3.2 Amdahl's Law Graph 2–64 Processors or Cores



31.4 Gustafson's Law

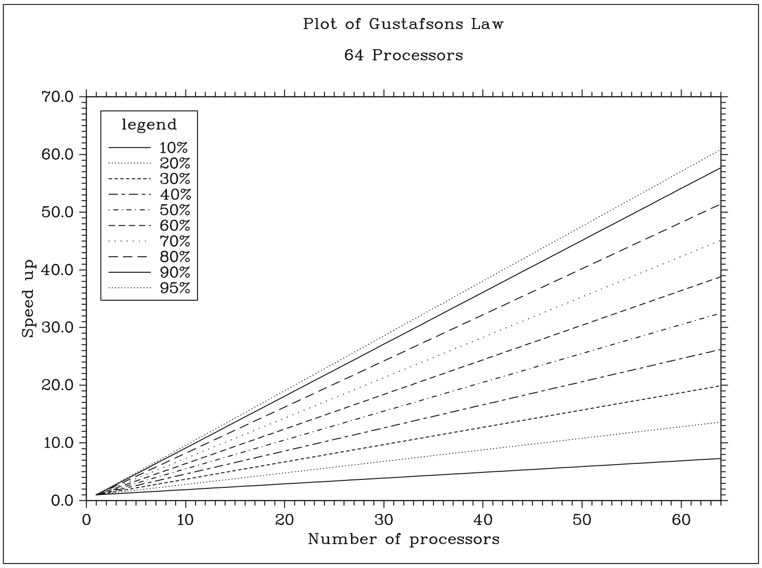
Gustafson's Law is often seen as a contradiction of Amdahl's Law. Simplistically it states that programmers solve larger problems when parallelising programs.

The equation for Gustafson's Law is given below.

- N is the number of processors
- Serial is the proportion that remains serial
- Speedup(N) = N - Serial * (N - 1).

We have again included a graph to illustrate the above.

31.4.1 Gustafson's Law Graph 1–64 Processors or Cores



31.5 Memory Access

Memory access times fall into two main categories that are of interest in parallel computing

- uma - uniform memory access. Each element of main memory can be accessed with the same latency and bandwidth. Multi-processor and multi-core computers typically have this behaviour.

- numa - non uniform memory access. Distributed memory systems have non-uniform memory access. Clusters or grids with multiple computers connected to work together have this behaviour.

31.6 Cache

Modern processors have a memory hierarchy. They typically have two or more levels:

- main memory
- cpu memory

and there is a speed and cost link. Main memory is cheap and relatively slow in comparison to the cpu memory.

The cpu memory or cache is used to reduce the effective access time to memory. If the information that the program requires is in the cpu cache then the average latency of memory accesses will be closer to the cache latency than to the latency of main memory. Getting high performance from a computer normally means writing cache friendly programs. This means that the data and instructions that the program needs are already in the cache and don't need to be accessed from the much slower main memory.

In a multi-core and multi-cpu system each core and cpu will have their own memory or cache. This introduces the problem of cache coherency - i.e. the consistency of data stored in local caches compared to the data in the common shared memory. This problem must obviously be addressed when doing parallel programming.

31.7 Bandwidth and Latency

Bandwidth is the rate at which data can be transferred. Latency is the start up time for a data transfer. We normally want a high bandwidth and low latency. Table 31.1 looks at some figures for several interconnects.

Table 31.1 Bandwidth and latency

	MPI bandwidth or theoretical maximum GB/s	latency μs
Gigabit ethernet	0.125	≈100
Infiniband	1.3	4.0
Myrinet 10-G	1.2	2.1
Quadrics QsNet II	0.9	2.7
Cray SeStar2	2.1	4.5

31.8 Flynn's Taxonomy

Flynn's taxonomy is an old, but still widely used, classification scheme for computer architecture.

- Single Instruction, Single Data stream (SISD) A sequential computer which exploits no parallelism in either the instruction or data streams. Term rarely used.
- Single Instruction, Multiple Data streams (SIMD) A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelised. For example, an array processor or GPU.
- Multiple Instruction, Single Data stream (MISD) Multiple instructions operate on a single data stream. Term rarely used.
- Multiple Instruction, Multiple Data streams (MIMD) Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space. Essentially separate computers working together to solve a problem.

We also have the term

- Single Program Multiple Data - An identical program executes on a MIMD computer system. Conditional statements in the code mean that different parts of the program execute on each system.

31.9 Consistency Models

Parallel programming languages and parallel computers must have a consistency model (also known as a memory model). The consistency model defines rules for how operations on computer memory occur and how results are produced.

31.10 Threads and Threading

In computing a thread of execution is often regarded as the smallest unit of processing that can be scheduled by an operating system. The implementation of threads and processes generally varies with operating system.

31.11 Threads and Processes

From a strict computer science point of view threads and processes are different. However when looking simply at parallel programming the term can often be used interchangeably. In the following we use the term thread.

31.12 Data Dependencies

A data dependency is when one statement in a program depends on a calculation from a previous statement. This will obviously hinder parallelism.

31.13 Race Conditions

Race conditions can occur in programs when separate threads depend on a shared state or variable.

31.14 Mutual Exclusion - Mutex

A mutex is a programming construct that is used to allow multiple threads to share a resource. The sharing is not simultaneous. One thread will acquire the mutex and then lock the other threads from accessing it until it has completed.

31.15 Monitors

In concurrent programming, a monitor is an object or module intended to be used safely by more than one thread. The defining characteristic of a monitor is that its methods are executed with mutual exclusion. That is, at each point in time, at most one thread may be executing any of its methods. This mutual exclusion greatly simplifies reasoning about the implementation of monitors compared with code that may be executed in parallel.

31.16 Locks

In computing a lock is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. Locks are one way of enforcing concurrency control policies.

31.17 Synchronization

The concept of synchronisation is often split into process and data synchronisation.

In process synchronisation several processes or threads come together at a certain part of a program.

Data synchronisation is concerned with keeping data consistent.

31.18 Granularity and Types of Parallelism

Granularity is a useful concept in parallel programming. A common classification is

- Fine-grained - a lot of small components, larger amounts of communication and synchronisation
- Coarse-grained - a small number of larger components, hence smaller amounts of communication and less synchronisation

The terms are of course relative.

We also have the concept of

- Embarrassingly parallel - very little effort is required to partition the task and there is little or no communication and synchronisation.

A simple example of this would be a graphics processor processing individual pixels.

31.19 Partitioned Global Address Space - PGAS

PGAS is a parallel programming model. It assumes a global memory address space that is logically partitioned and a portion of it is local to each processor. The PGAS model is the basis of Unified Parallel C, Coarray Fortran, Titanium, Fortress, Chapel and X10.

31.20 Fortran and Parallel Programming

Most Fortran compilers now offer support for parallel programming. We next provide a brief coverage of three methods

- MPI - Message Passing Interface
- OpenMP - Open Multi-Processing
- CoArray Fortran.

Subsequent chapters look at simple examples using each method.

31.21 MPI

MPI started with a meeting that was held at the Supercomputing 92 conference. The attendants agreed to develop and implement a common standard for message passing. The first MPI standard, called MPI-1 was completed in May 1994. The second MPI standard, MPI-2, was completed in 1998.

MPI is effectively a library of C and Fortran callable routines. It has become widely used and is available on a number of platforms. Some useful web addresses are given below. The first is hosted at Argonne National Laboratory.

```
http://www.mcs.anl.gov/research/projects/mpi/
```

MPI was designed by a broad group of parallel computer users, vendors, and software writers. These included

- Vendors - IBM, Intel, TMC, Meiko, Cray, Convex, Ncube
- Library writers - PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda
- Companies - ARCO, Convex, Cray Research, IBM, Intel, KAI, Meiko, NAG, nCUBE, Parasoft, Shell, TMC
- Laboratories - ANL, GMD, LANL, LLNL, NOAA, NSF, ORNL, PNL, Sandia, SDSC, SRC
- Universities - UC Santa Barbara, Syracuse University, Michigan State University, Oregon Grad Inst, University of New Mexico, Mississippi State University, University of Southampton, University of Colorado, Yale University, University of Tennessee, University of Maryland, Western Michigan University, University of Edinburgh, Cornell University, Rice University, University of San Francisco.

So whilst MPI is not a formal standard like Fortran, C or C++, its development has involved quite a wide range of people. The following site has details of MPI meetings.

```
http://meetings.mpi-forum.org/
```

The steering committee (March 2015) and affiliations are given below

- Jack Dongarra - Computer Science Department, University of Tennessee
- Al Geist - Group Leader, Computer Science Research Group, Oak Ridge National Laboratory
- Richard Graham
- Bill Gropp - Computer Science Department, University of Illinois Urbana-Champaign
- Andrew Lumsdaine - Computer Science Department, Indianna University
- Ewing Lusk - Mathematics and Computer Science Division, Argonne National Laboratory
- Rolf Rabenseifner - High Performance Computing Center, Germany.

Another useful site is the Open MPI site.

```
http://www.open-mpi.org/
```

The following is taken from their site.

The Open MPI Project is an open source MPI implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.

Both sites provide free down loadable implementations. Commercial implementations are available from

- Cray
- IBM
- Intel
- Microsoft

amongst others.

MPI is, at the time of writing, the dominant parallel programming method used in Fortran. MPI and Fortran currently account for over 80% of the code running on the Archer Service in Edinburgh. Archer is the UK's national supercomputing resource, funded by the UK Research Councils. Visit

```
http://www.archer.ac.uk
```

for more information.

31.22 OpenMP

OpenMP (Open Multi-Processing) is an application programming interface that supports shared memory multiprocessing programming in three main languages (C, C++, and Fortran) on a range of hardware platforms and operating systems. It consists of a set of compiler directives, library routines, and environment variables that determine the run time behaviour of a program.

The OpenMP Architecture Review Board (ARB) has published several versions

- October 1997 - OpenMP for Fortran 1.0. October the following year they released the C/C++ standard.
- 2000 - Fortran version
- 2005 - Fortran 2.5
- 2008 - OpenMP 3.0. Included in the new features in 3.0 is the concept of tasks and the task construct.

- 2011 - OpenMP 3.1
- 2013 - OpenMP 4.0 was released in July 2013.

A number of compilers from various vendors or open source communities implement the OpenMP API, including

- Absoft
- Cray
- gnu
- Hewlett Packard
- IBM
- Intel
- Lahey/Fujitsu
- Nag
- Oracle/Sun
- PGI

The main OpenMP web site is:

```
http://www.openmp.org/
```

31.23 Coarray Fortran

Coarrays became part of Fortran in the 2008 standard. The original ideas came from work by Robert Numrich and John Reid in the 1990s. They are based on a single program multiple data model. A coarray Fortran program is interpreted as if it were duplicated several times and all copies execute asynchronously. Each copy has its own set of data objects and is termed an image. The array syntax of Fortran is extended with additional trailing subscripts in square brackets to provide a concise representation of references to data that is spread across images.

The syntax is architecture independent and may be implemented on:

- Distributed memory machines.
- Shared memory machines.
- Clustered machines.

Work has now been completed on additional Coarray functionality and is in the Fortran 2018 standard.

31.24 Other Parallel Options

There are a number of additional parallel methods. They are covered for completeness.

31.24.1 PVM

Parallel Virtual Machine consists of a library and a run-time environment which allow the distribution of a program over a network of (even heterogeneous) computers. Visit

- `http://www.netlib.org/pvm3/`

for more details.

31.24.2 HPF

To quote their home page

`http://hpff.rice.edu/index.htm`

'The High Performance Fortran Forum (HPFF), a coalition of industry, academic and laboratory representatives, works to define a set of extensions to Fortran 90 known collectively as High Performance Fortran (HPF). HPF extensions provide access to high-performance architecture features while maintaining portability across platforms.'

They also provide details of:

- Surveys of HPF compilers and tools.
- Currently available commercial HPF compilers.
- public domain HPF compilation systems.
- Research prototypes of HPF and HPF-related compilation systems.
- Mailing list.

31.25 Top 500 Supercomputers

Have a look at

`https://www.top500.org/`

for a lot of links to supercomputing centres and information on parallel computing in general. To see what can be done with all this processing power visit:

`http://www.metoffice.gov.uk/`

31.26 Summary

Fortran has long been one of the main languages used in parallel programming. This chapter has provided a brief coverage of some of the background to parallel programming in general, and Fortran in particular.

In the next three chapters we will look at a small number of programs that introduce some of the basic syntax of parallel programming with MPI, OpenMP and Coarray Fortran. We will also look at solving one problem serially and then solve it using the parallel features provided by MPI, OpenMP and Coarray Fortran. We provide timing details so that we can see the benefits that parallel solutions offer.

Bibliography

The ideas involved in parallel computing are not new and we've included a couple of references about computer hardware and operating systems, which provide information for the more inquisitive reader. Wikipedia is an on-line source of information in this area.

Up to date hardware information can be found at most hardware vendor sites. Here are the web sites for AMD, IBM and Intel.

```
AMD
http://developer.amd.com/pages/default.aspx
IBM
http://www.ibm.com/products
Intel
http://www.intel.com/en_UK/
products/processor/index.htm
```

Baer J.L., Computer Systems Architecture, Computer Science Press, 1980.

The chapters on the memory hierarchy and memory management are old, but well written.

Deitel H.M., Operating Systems, Addison Wesley, 1990.

Part two of the book (process management) has chapters on process concepts, asynchronous concurrent processes, concurrent programming and deadlock and indefinite postponement. The bibliographies at the end of each chapter are quite extensive.

The following four books provide a good coverage of the essentials of MPI and OpenMP.

Chandra R., Dagum L., Kohr D., Maydan D., McDonald J., Menon R., Parallel Programming in OpenMP, Morgan Kaufmann.

Chapman B., Jost G., Van Der Pas R., Using OpenMP, MIT Press.

Gropp W., Lusk E., Skjellum A., Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press.

Pacheco P., Parallel Programming with MPI, Morgan Kaufmann.