Chapter 1

Parallel Computers

Demand for Computational Speed

Continual demand for greater computational speed from a computer system than is currently possible

Areas requiring great computational speed include numerical modeling and simulation of scientific and engineering problems.

Computations must be completed within a “reasonable” time period.

Grand Challenge Problems

A grand challenge problem is one that cannot be solved in a reasonable amount of time with today’s computers.

Obviously, an execution time of 10 years is always unreasonable.

Examples

- Modeling large DNA structures
- Global weather forecasting
- Modeling motion of astronomical bodies.

Weather Forecasting

Atmosphere modeled by dividing it into 3-dimensional cells. Calculations of each cell repeated many times to model passage of time.

Example

Whole global atmosphere divided into cells of size 1 mile × 1 mile × 1 mile to a height of 10 miles (10 cells high) - about 5×10^8 cells.

Suppose each calculation requires 200 floating point operations. In one time step, 10^{11} floating point operations necessary.

Weather Forecasting

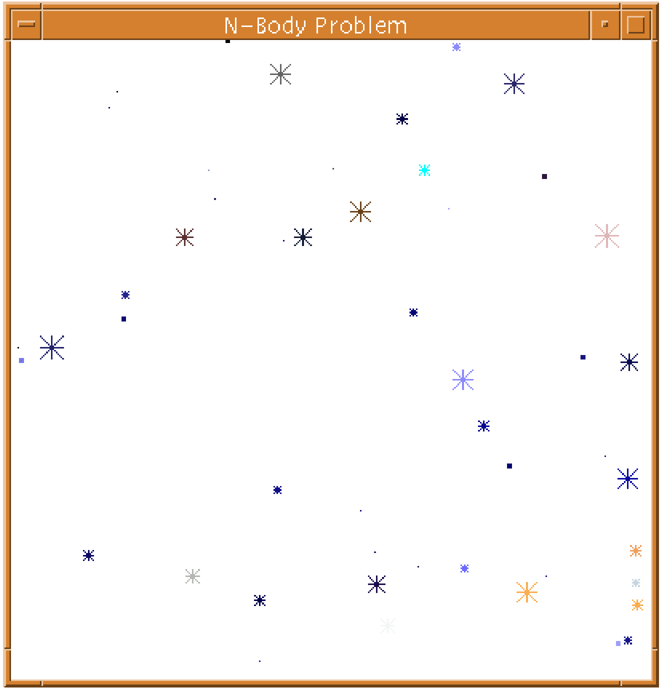
To forecast the weather over 10 days using 10-minute intervals, a computer operating at 100 Mflops (10^8 floating point operations/s) would take 10^7 seconds or over 100 days.

To perform the calculation in 10 minutes would require a computer operating at 1.7 Tflops (1.7×10^{12} floating point operations/sec).

Modeling Motion of Astronomical Bodies

Each body attracted to each other body by gravitational forces. Movement of each body predicted by calculating total force on each body. With N bodies, $N - 1$ forces to calculate for each body, or approx. N^2 calculations. ($N \log_2 N$ for an efficient approx. algorithm.) After determining new positions of bodies, calculations repeated.

A galaxy might have, say, 10^{11} stars. Even if each calculation could be done in 1μs (an extremely optimistic figure), it would take 10^9 years for one iteration using the N^2 algorithm and almost a year for one iteration using an efficient $N \log_2 N$ approximate algorithm.



Astrophysical *N*-body simulation by Scott Linssen (undergraduate University of North Carolina at Charlotte [UNCC] student).

Parallel Computing

Using more than one computer, or a computer with more than one processor, to solve a problem.

Motives

Usually faster computation - very simple idea - that *n* computers operating simultaneously can achieve the result *n* times faster - it will not be *n* times faster for various reasons.

Other motives include: fault tolerance, larger amount of memory available, ...

Background

Parallel computers - computers with more than one processor - and their programming - parallel programming - has been around for more than 40 years.

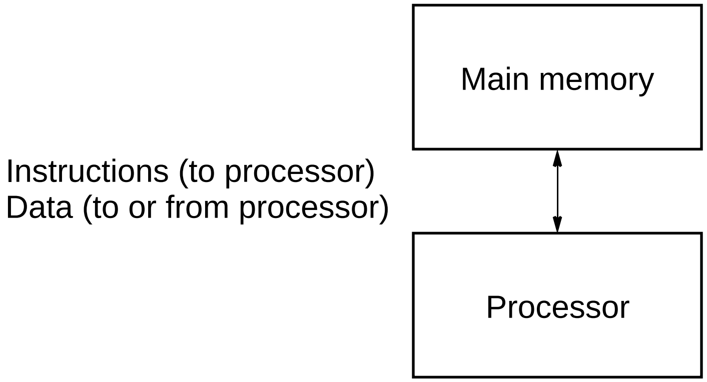
Gill writes in 1958:

"... There is therefore nothing new in the idea of parallel programming, but its application to computers. The author cannot believe that there will be any insuperable difficulty in extending it to computers. It is not to be expected that the necessary programming techniques will be worked out overnight. Much experimenting remains to be done. After all, the techniques that are commonly used in programming today were only won at the cost of considerable toil several years ago. In fact the advent of parallel programming may do something to revive the pioneering spirit in programming which seems at the present to be degenerating into a rather dull and routine occupation ..."

Gill, S. (1958), "Parallel Programming," *The Computer Journal*, vol. 1, April, pp. 2-10.

Conventional Computer

Consists of a processor executing a program stored in a (main) memory:



Each main memory location located by its *address*. Addresses start at 0 and extend to $2^n - 1$ when there are n bits (binary digits) in the address.

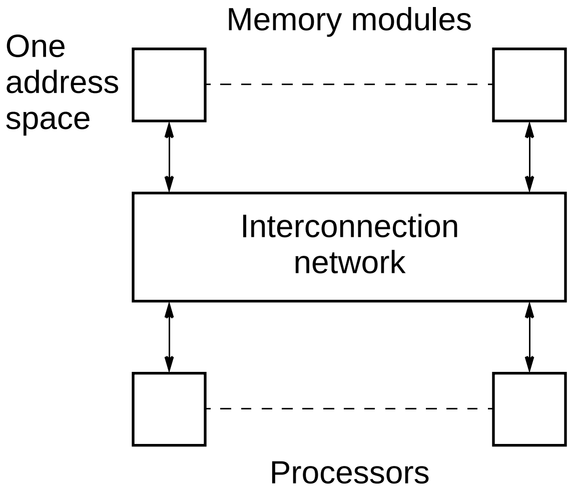
Types of Parallel Computers

Two principal types:

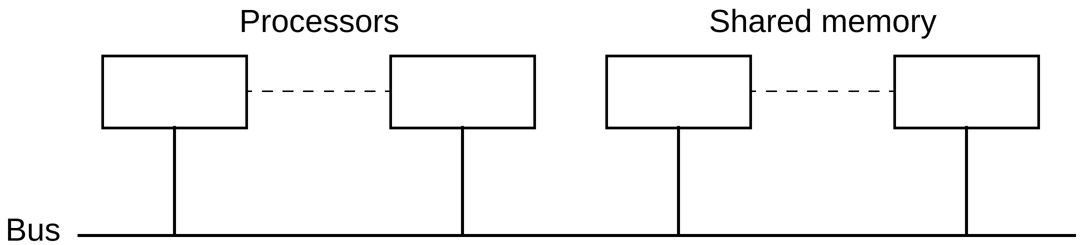
- Shared memory multiprocessor
- Distributed memory multicomputer

Shared Memory Multiprocessor System

Natural way to extend single processor model - have multiple processors connected to multiple memory modules, such that each processor can access any memory module - so-called *shared memory* configuration:



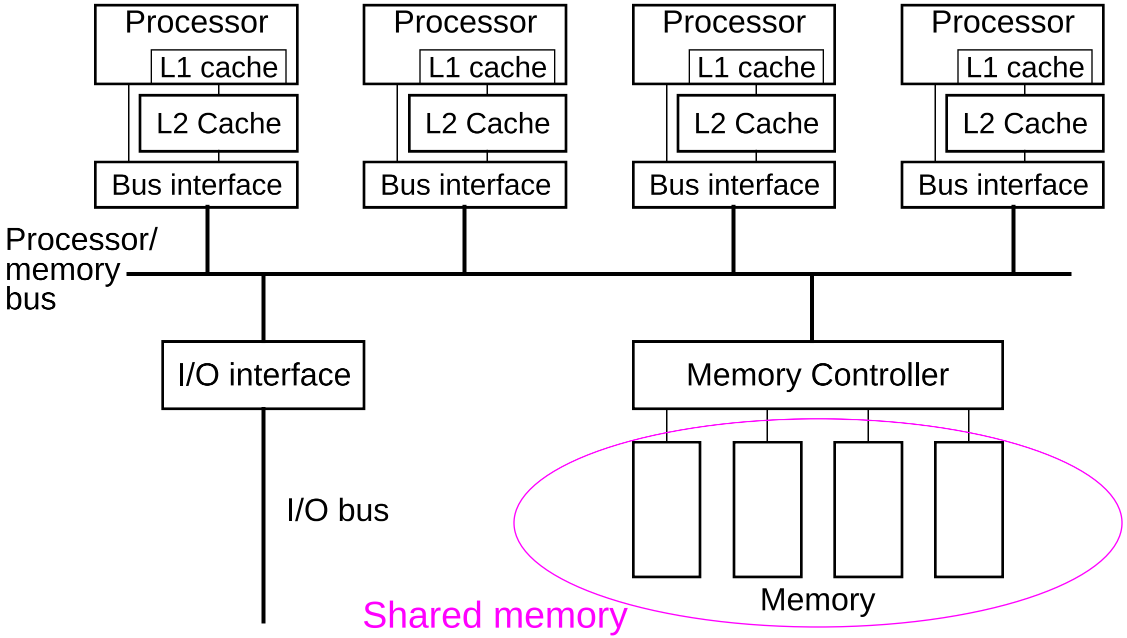
Simplistic view of a small shared memory multiprocessor



Examples:

- Dual Pentiums
- Quad Pentiums

Quad Pentium Shared Memory Multiprocessor



Shared memory multiprocessor system

Any memory location can be accessible by any of the processors.

A *single address space* exists, meaning that each memory location is given a unique address within a single range of addresses.

Generally, shared memory programming more convenient although it does require access to shared data to be controlled by the programmer (using critical sections etc.)

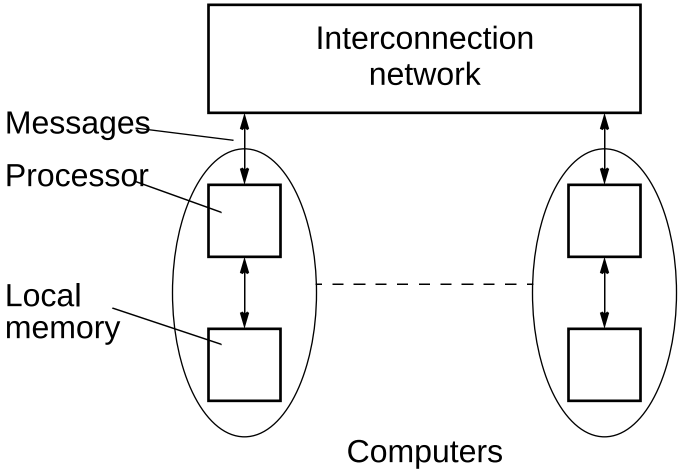
Several Alternatives for Programming Shared Memory Multiprocessors:

Using:

- Threads (Pthreads, Java, ..) in which the programmer decomposes the program into individual parallel sequences, each being thread, and each being able to access variables declared outside the threads.
- A sequential programming language with preprocessor compiler directives to declare shared variables and specify parallelism. Example OpenMP - industry standard
- A sequential programming language with user-level libraries to declare and access shared variables.
- A parallel programming language with syntax for parallelism, in which the compiler creates the appropriate executable code for each processor (not now common)
- A sequential programming language and ask a parallelizing compiler to convert it into parallel executable code. - also not now common

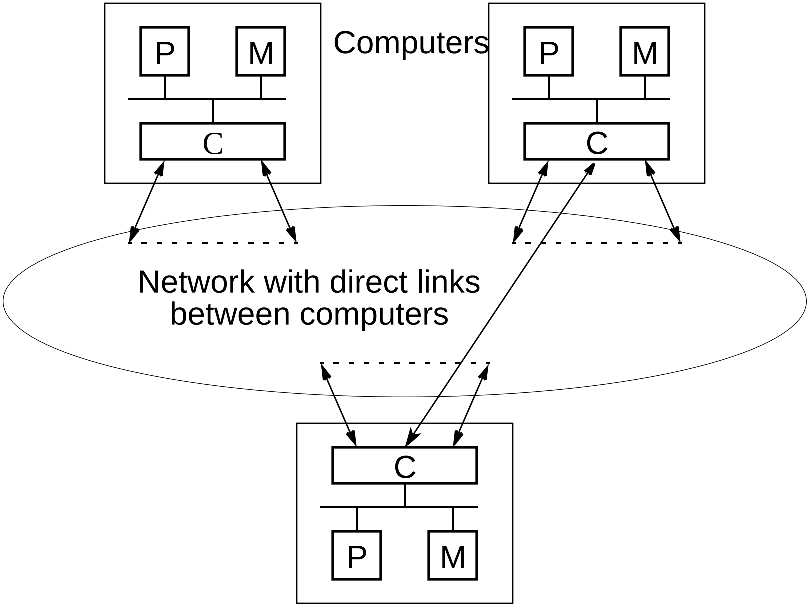
Message-Passing Multicomputer

Complete computers connected through an interconnection network:



Static Network Message-Passing Multicomputers

Computers connected by direct links:

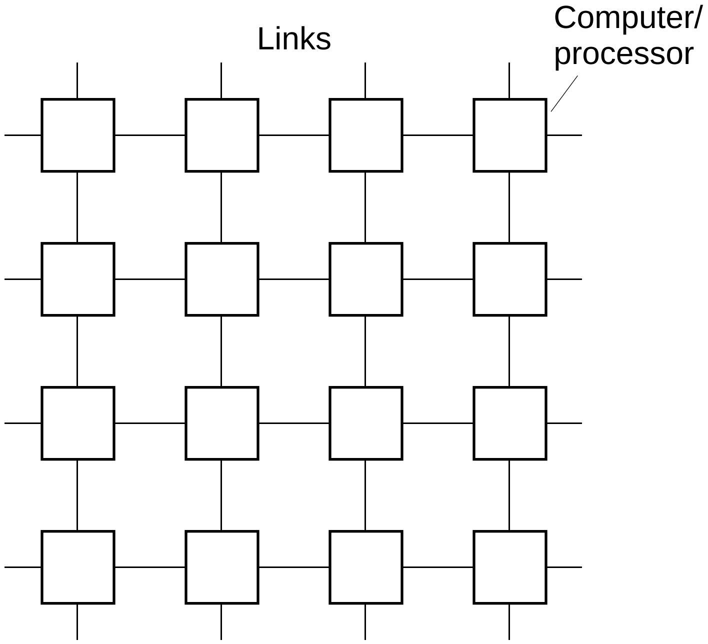


Static Link Interconnection Networks

Various:

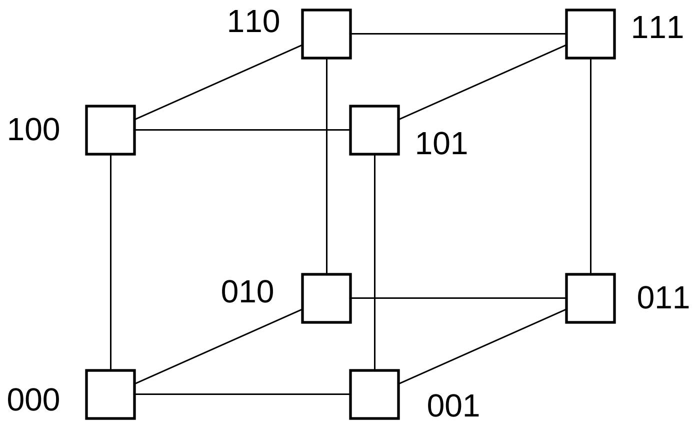
- Ring
- Tree
- 2-D and 3-D arrays
- Hypercube

Two-dimensional array (mesh)

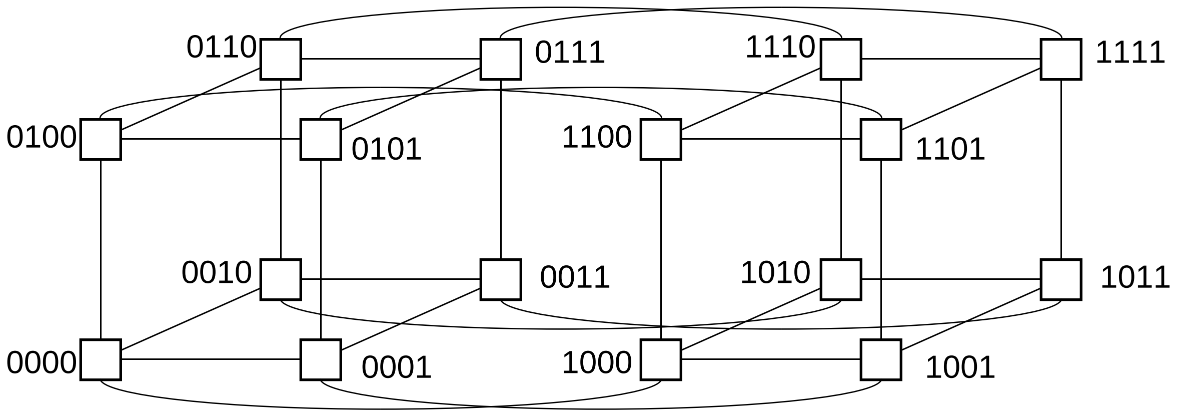


Also three-dimensional - used in some large high performance systems.

Three-dimensional hypercube



Four-dimensional hypercube



Hypercubes popular in 1980's - not now

Networked Computers as a Multicomputer Platform

A *network of workstations* (NOWs) became a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing in early 1990's.

Several Projects

- Berkely NOW project

Key advantages:

- Very high performance workstations and PCs readily available at low cost.
- The latest processors can easily be incorporated into the system as they become available.
- Existing software can be used or modified.

Beowulf Clusters*

A group of interconnected "commodity" computers achieving high performance with low cost.

Typically using commodity interconnects - high speed Ethernet, and Linux OS.

* Beowulf comes from name given by NASA Goddard Space Flight Center cluster project.

Cluster Interconnects

- Originally fast Ethernet on low cost clusters
- Gigabit Ethernet - easy upgrade path

Using Ethernet switches to connect computers

More Specialized/Higher Performance

- Myrinet - 2.4 Gbits/sec - disadvantage: single vendor
- cLan
- SCI (Scalable Coherent Interface)
- QsNet
- Infiniband - may be important as infininbnand interfaces may be intergrated on next generation PCs

See Beowulf reference book for more details.

Message Passing Parallel Programming Software Tools for Clusters

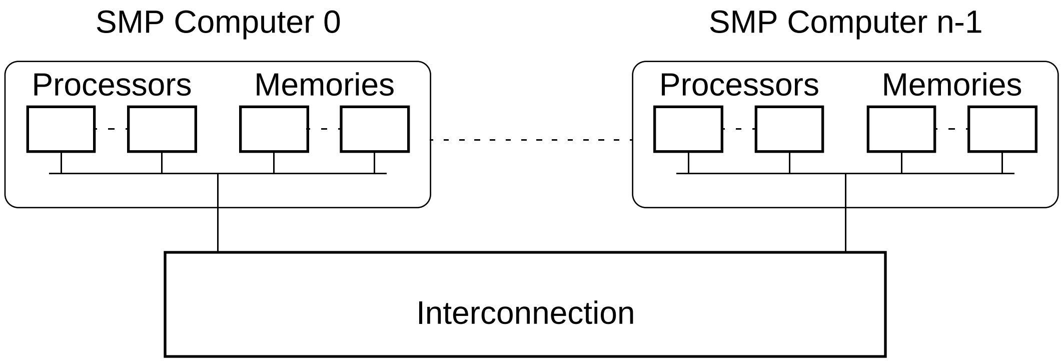
Parallel Virtual Machine (PVM) - developed in late 1980’s. Became very popular.

Message-Passing Interface (MPI) - standard defined in 1990s.

Both provide a set of user-level libraries for message passing. Use with regular programming languages (C, C++, ...).

SMP Cluster

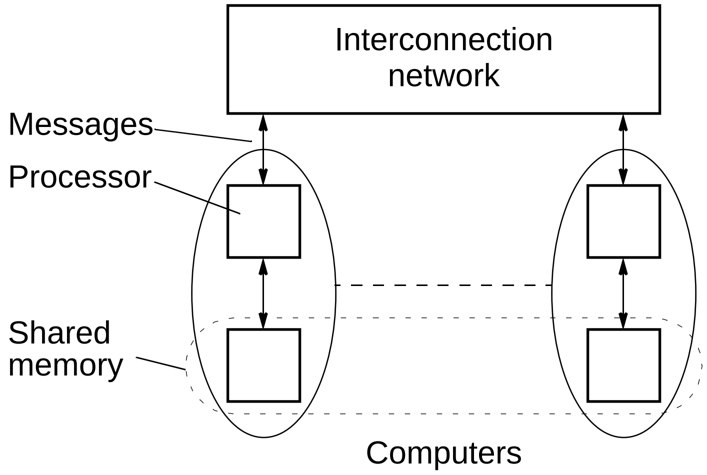
Can have a cluster of shared memory computers (symmetrical multiprocessors)



Distributed Shared Memory

Making the main memory of a cluster of computers look as though it is a single memory with a single address space.

Then can use shared memory programming techniques.



Flynn’s Classifications

Flynn (1966) created a classification for computers based upon instruction streams and data streams:

Single instruction stream-single data stream (SISD) computer

In a single processor computer, a single stream of instructions is generated from the program. The instructions operate upon a single stream of data items. Flynn called this single processor computer a *single instruction stream-single data stream* (SISD) computer.

Multiple Instruction Stream-Multiple Data Stream (MIMD) Computer

General-purpose multiprocessor system - each processor has a separate program and one instruction stream is generated from each program for each processor. Each instruction operates upon different data.

Both the shared memory and the message-passing multiprocessors so far described are in the MIMD classification.

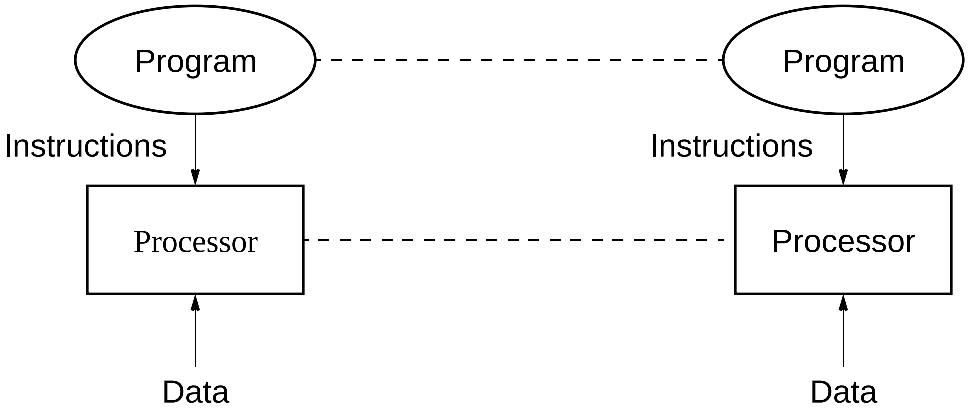
Single Instruction Stream-Multiple Data Stream (SIMD) Computer

A specially designed computer in which a single instruction stream is from a single program, but multiple data streams exist. The instructions from the program are broadcast to more than one processor. Each processor executes the same instruction in synchronism, but using different data.

Developed because there are a number of important applications that mostly operate upon arrays of data.

Multiple Program Multiple Data (MPMD) Structure

Within the MIMD classification, which we are concerned with, each processor will have its own program to execute:



Single Program Multiple Data (SPMD) Structure

Single source program is written and each processor will execute its personal copy of this program, although independently and not in synchronism.

The source program can be constructed so that parts of the program are executed by certain computers and not others depending upon the identity of the computer.

Speedup Factor

$$S(n) = \frac{\text{Execution time using one processor (single processor system)}}{\text{Execution time using a multiprocessor with n processors}} = \frac{t_s}{t_p}$$

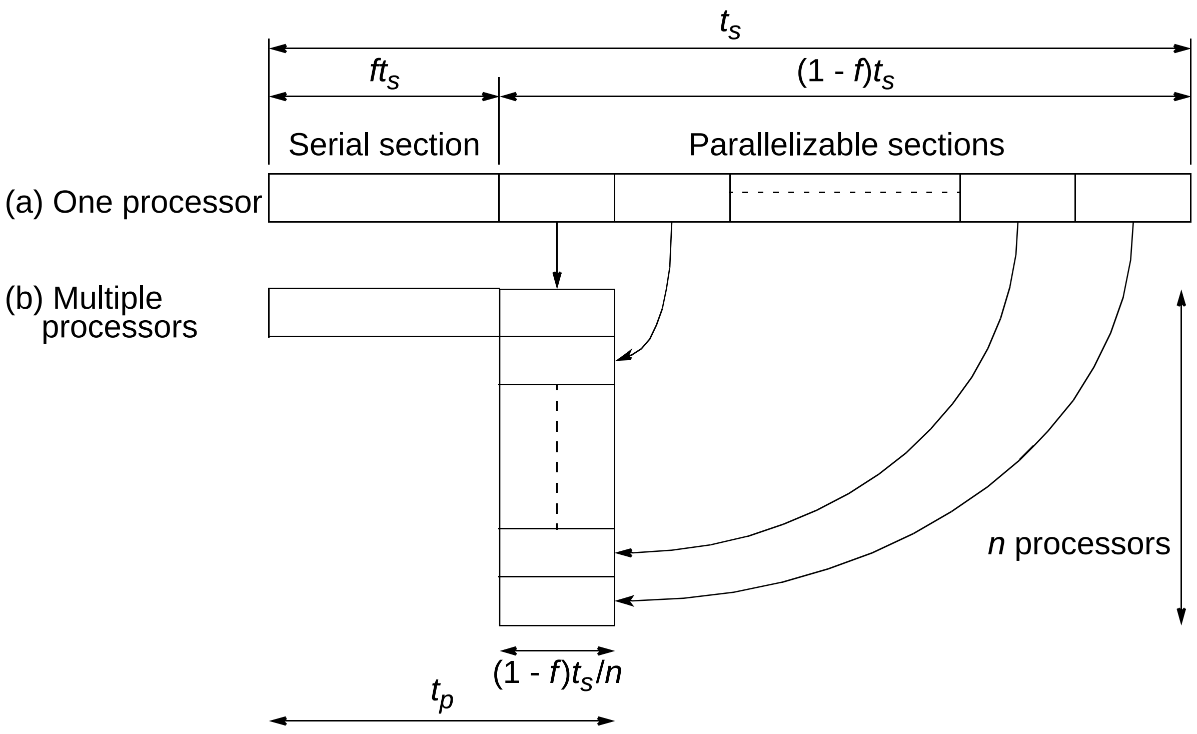
where t_s is execution time on a single processor and t_p is execution time on a multiprocessor. $S(n)$ gives increase in speed by using multiprocessor. Underlying algorithm for parallel implementation might be (and is usually) different.

Speedup factor can also be cast in terms of computational steps:

$$S(n) = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with n processors}}$$

Maximum speedup is (usually) n with n processors (*linear speedup*).

Maximum Speedup - Amdahl's law

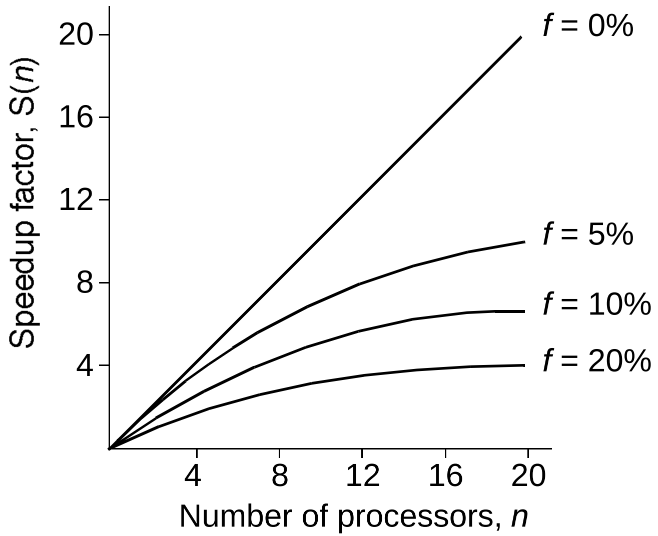


Speedup factor is given by:

$$S(n) = \frac{t_s}{ft_s + (1 - f)t_s/n} = \frac{n}{1 + (n - 1)f}$$

This equation is known as *Amdahl's law*

Speedup against number of processors



Even with infinite number of processors, maximum speedup limited to 1/*f*.

Example: With only 5% of computation being serial, maximum speedup is 20, irrespective of number of processors.

Intentionally blank